Problem set 2b: M-file Programming

Handed out: Saturday, October 17, 2016.

Due: 11:55pm, Thursday, October 27, 2016.

You must hand in two files:

- One file, named your_name_E2a.pdf, with the solutions to the exercises in this set, following the template available in the virtual campus. For every exercise, you must explain how you solved it, the necessary MATLAB code, the results obtained (run transcripts or plots), and some final comments (if any).
- The second file, named your_name_E2a.zip, must contain all the MATLAB code you used to solve the exercises, organized in one or more text (.m) files, to allow the teacher to check your solution.

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**. Otherwise, you must clearly specify which parts are not, and properly attribute them to the source (names of collaborators, links to webpages, book references,...). Read the Course Information document for more details on this subject.

In those exercises where a function is implemented, a "help" section must be included in the software so that when the Matlab command help is called with the name of that function proper information is provided.

Comment: In this unit it is assumed that you already know how to operate vector and matrix variables. The goal in this exercises is to implement the requested task as efficiently as possible, considering that efficiency is mostly related to the software execution time. In general, Matlab programs should be written avoiding as much as possible the use of loops, resorting to vectorization. As a rule of thumb, the shorter the Matlab program file is the faster it will execute. Although the use of specific data types might improve the efficiency of some exercises in this assignment, this question is beyond the scope of this unit.

Exercise 6. Morse code

The Morse code is a method of transmitting text information as a series of on-off tones, lights or clicks. In this code the letters, numbers and a small set of punctuation are mapped into a unique sequence of short and long signals called "dots" and "dashes". You are requested to write the Matlab script to do a Morse encoder. The encoder must follow the following rules:

- The dash-dot sequence associated to each character is shown below.
- The dots/dashes are mapped to short/long lasting beeps, whereas silences are inserted in between to indicate the end of a character or a word. The dot duration is the basic unit of time measurement in code transmission. Encoding is done as follows:
  - The duration of a dash is three times the duration of a dot.
  - Each dot or dash is followed by a short silence, equal to the dot duration.
  - The letters of a word are separated by a silence equal to three dots (one dash).
  - The words are separated by a silence equal to seven dots.

A	.—	P	.——.	_	..——.—	1	.————
B	—...	Q	——.—	-	—....—	2	..———
C	—.—.	R	.—.	,	——..——	3	...——
D	—..	S	...	;	—.—.—.	4	....—
E	.	T	—	:	———...	5	.....
F	..—.	U	..—	!	—.—.——	6	—....
G	——.	V	...—	?	..——..	7	——...
H	....	W	.——	.	.—.—.—	8	———..
I	..	X	—..—	’	.————.	9	————.
J	.———	Y	—.——	”	.—..—.	0	—————
K	—.—	Z	——..	(	—.——.		
L	.—..			)	—.——.—		
M	——			@	.——.—.		
N	—.			/	—..—.		
O	———			&	.—...		
				+	.—.—.		
				=	—...—		
				$	...—..—		

You are asked to:

1) Write the Matlab function `Morse_encoder` that:

- It loads the file "morse.mat", that contains in `Morse{i}` the dash/dot code corresponding to the character stored in `Alpha(i)`. Note that `Morse` is a cell array.
- It accepts as input parameter `input_string:` a string with the text to be encoded.
- It returns two output parameters:
  - o `dashdot_seq` contains a string with a sequence of dash ('`-`') dots ('.') and spaces (' ') of the Morse-encoded sequence, so that each space represents a silence of the same duration of one dot.
  - o `pulse_seq` contains a string with a sequence of 1's and 0's, where each '1' and '0' correspond respectively to a pulse and a silence of duration of one dot (i.e., each dash is mapped to '111').

(**`load`**, **`find`**, **`switch`**, **`for`**...**end**,**`==`**)

2) Write a Matlab function `Morse_beep` that:

- It accepts as input parameters:
  - o The string `pulse_seq` (see definition above).
  - o `tone_freq`: the frequency of the tone employed to signal a dash or a dot (in Hz).
  - o `dot_duration`: the duration of one dot (in seconds).
  - o `sampling_freq`: the sampling frequency employed in sound reproduction.
- It has no output parameters
- It computes the pulse duration in samples (`dot_samples`) and the discrete frequency of the tone when a sampling frequency of `sampling_freq` is employed.
- It produces a sound that corresponds to the Morse encoding of `pulse_seq`.
- It depicts in a figure the analog waveform employed to produce the sound.

(**`plot`**, **`sound, repmat, cos`**)

3) Write a Matlab script that produces the Morse code of the text "MAE-Unit 2b" with parameters:

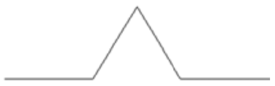
dot_duration = 0.05sec tone frequency = 900 Hz, sampling frequency = 8000Hz

The software must show in the command window the original text and the sequence of dash/dot/spaces corresponding to that text. Besides, it must call the function `Morse_beep` to depict the sound waveform and to reproduce it.

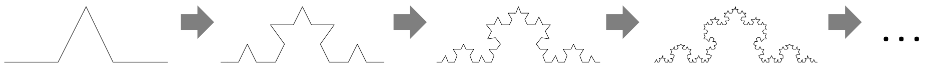
Exercise 7. Koch fractal curve

This problem aims at drawing the Koch curve of order *n*, recursively defined in the following way:

- Start from an horizontal segment of unit length (order zero curve).
- Transform the *n*-th order curve into the (*n+1*)-th order curve by replacing every segment with the triangular pattern shown below (the endpoints remain the same).



So the Koch curves of increasing order look like:



Note that:

- The first order curve is exactly the basic triangular pattern above.
- Any polygonal curve in the x-y plane with endpoints (0,0) and (1,0) can be transformed into another one with the same shape but with arbitrary endpoints (u_0,v_0) and (u_1,v_1) by means of an affine transformation that rotates/scales/translates the curve as follows:

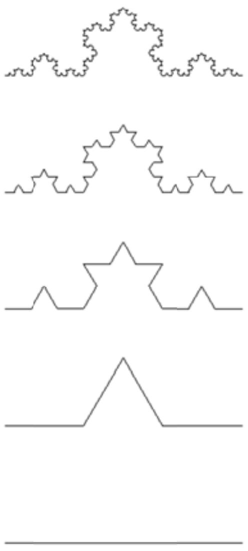
$$u = u_0 + (u_1 - u_0)x - (v_1 - v_0)y$$

$$v = v_0 + (v_1 - v_0)x + (u_1 - u_0)y$$

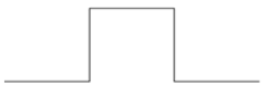
- By cleverly using the operator **`:`** the curve of order *n+1* can be obtained from the curve of order *n* with a single **`for`** loop with as many iterations as the number of segments in the basic triangular pattern above (that is, irrespective of the order *n* only one loop of length 4 is required).

You are asked to:

1) Create a function with name **`koch.m`** such that **`M=koch(n)`** outputs a matrix **`M`** with two columns, containing the x and y coordinates of the vertices of the *n*-th order curve. If no output argument is provided, simply calling **`koch(n)`**produces the figure shown on the right, containing the curves of all intermediate orders. *Your function should only contain two nested loops:* one of length **`n`** (each iteration generates one curve) and another one of length 4 (the number of segments in the basic triangular pattern).
(**`plot`**, **`subplot`**, **`for`**,**`:`**,**`axis equal`**, **`axis off`**)



2) Modify the previous function (and call it **`genkoch.m`**) to admit an input matrix **`M1`** containing an arbitrary polygonal curve (e.g. a rectangular pattern) with the same format as the previous matrix **`M`** to be used as the transformation pattern. The endpoints of the polygonal curve must always be (0,0) and (0,1).



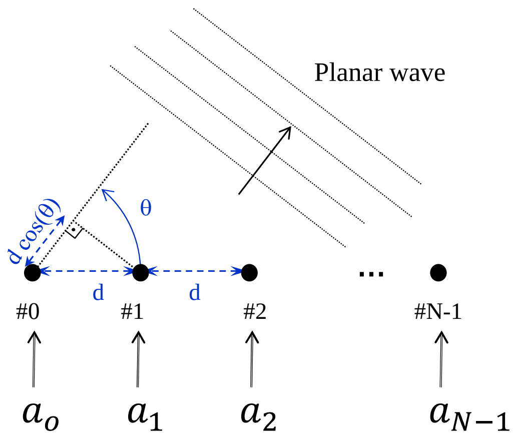
Exercise 8. ULA antenna gain diagram

Consider an antenna array where identical antenna elements are placed in a line with uniform spacing (Uniform Linear Array, ULA). These arrays can be employed to obtain very directive radiation patterns that point at the desired direction by choosing the right antenna distance and applying the right weights (antenna feed network).

Under the far-field assumption, the antenna array is characterized by the array factor (AF), which identifies to the radiation pattern if the array elements were isotropic:

$$AF(\Psi) = \sum_{n=0}^{N-1} a_n \, e^{jn\Psi}$$

where N is the amount of antenna elements in the array and the complex coefficients a_n depend on the antenna feed network. The antenna gain for a certain angle of the planar wave θ is obtained from the array factor setting $\Psi = 2\pi \frac{d}{\lambda} \cos\theta$, being d the distance between array elements (see figure below).



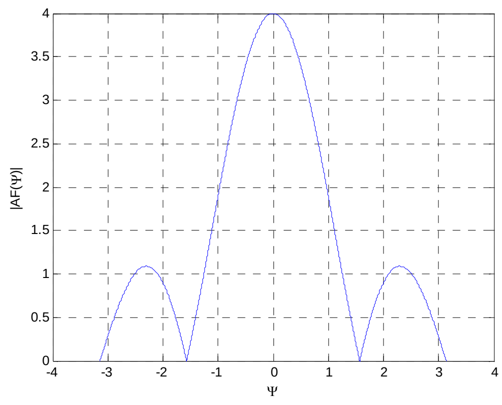
Therefore, the antenna gain in a certain direction can be written as:

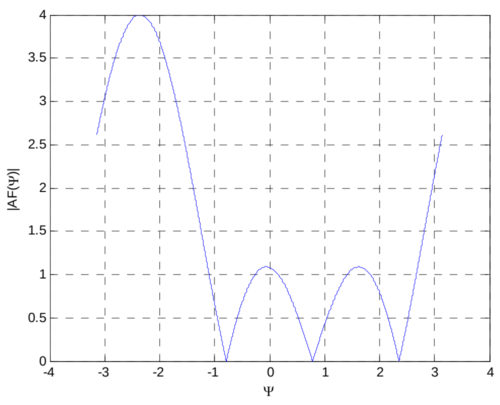
$$G(\theta) = \left| \sum_{n=0}^{N-1} a_n \, e^{jn2\pi\frac{d}{\lambda}\cos\theta} \right| = \quad -\pi \le \theta < \pi$$

Note that, due to array symmetry, the antenna gain for angles in the range $0 \le \theta \le \pi$ is the same as for angles in the range $-\pi \le \theta \le 0$ (there is rotationally symmetry with respect to to array axis).

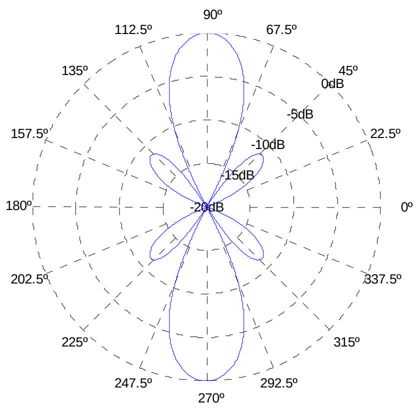
The following figure depicts the array factor for the N=4, $d = \frac{\lambda}{2}$, and two sets of antenna weights:

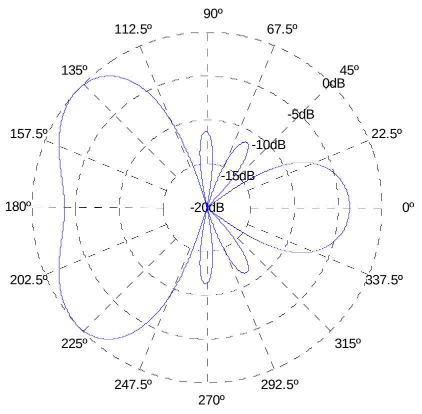
$a_1 = a_2 = a_3 = a_4 = 1$ (left) and $a_1 = 1, a_2 = e^{j\frac{3\pi}{4}}, a_3 = e^{j\frac{6\pi}{4}},\ a_4 = e^{j\frac{9\pi}{4}}$ (right)





The next figure depicts the polar plot of the antenna gain in dB normalized with respect to its maximum value (same parameters and same weights as before for the left/right plots):





Among the parameters that characterize the antenna radiation pattern we can mention:

- The direction at which the antenna is pointing: the angle $0 \leq \theta \leq \pi$ for which $G(\theta)$ is maximum $G_{max} = G(\theta_{max})$.
- The main lobe width measured at -3dB (where the gain decreases to $G_{max}/\sqrt{2}$)
- The antenna directivity, which in the case of the ULA simplifies to

$$D = \frac{2|AF_{max}|^2}{\int_0^{\pi} \left| AF(\Psi)|_{\Psi=2\pi\frac{d}{\lambda}\cos\theta} \right|^2 \sin\theta \, d\theta}$$

In the examples above these parameters are respectively:

	$a_1 = a_2 = a_3 = a_4 = 1$	$a_1 = 1, a_2 = e^{j\frac{3\pi}{4}}, a_3 = e^{j\frac{6\pi}{4}}, \quad a_4 = e^{j\frac{9\pi}{4}}$
Maximum gain direction	$\theta_{max} = \frac{\pi}{2}, G_{max} = 4$	$\theta_{max} = 0.77\pi, G_{max} = 4$
3dB beamwidth	$\theta_1 = 0.43\pi, \theta_2 = 0.57\pi$ $\theta_2 - \theta_1 = 0.146\,\pi$	$\theta_1 = 0.93\pi,\ \theta_2 = 0.67\pi$ $\theta_2 - \theta_1 = 0.258\,\pi$
Directivity	$D = 4$	$D = 4$

Note that if we want to evaluate the directivity with Matlab we need to compute the definite integral in the denominator. We will do it using the integral computation routine that you did in exercise 2 of the assignment E2a.

You are asked to:

1) Write a Matlab function called **`compute_AF`** that evaluates the array factor at the requested angle values. The routine should accept as parameters the vector storing the antenna weights **`a`** and the angle values **`Psi_values`** where it must be computed. The routine should return a vector **`AF_values`** with the array factor values at the requested angle values.

2) Write a Matlab function called **`compute_gain`** that evaluates the array factor in equispaced angles. The routine should accept as parameters the distance between array elements **`d`** normalized by the wavelength (i.e. if the distance is $\lambda/2$ the function parameter should be 1/2), the vector storing the antenna weights **`a`** and the number of points **`Npoints`** where it must be computed. It should return a vector **`theta_values`** with the equispaced values of $-\pi \leq \theta \leq \pi$ where it has been computed and another vector **`gain_values`** with the values of $G(\theta)$.

3) Write a Matlab function **`radiation_pattern_parameters`** to compute the maximum gain direction **`theta_max`**, the mainlobe beamwidth **`delta_theta`** and the directivity **`D`** and

to show them in the screen. The function should accept as inputs **`theta_values`** and **`gain_values`**.

4) Write a Matlab script named **`plot_AF`** that plots the array factor in a linear scale (as shown above)[1].

5) Write a Matlab function to do the polar plot of the array gain in a logarithmic scale (as shown in the examples above). Unfortunately the Matlab routine **`polar`** only works with linear gains, so you are asked to write a routine capable of working in dB's. Create a function with name **`polardB`** to generate polar plots of $20\log_{10}\left(G(\theta)\right)$ exactly as the plots shown in the previous examples (including the same number of grid lines). Input arguments must be **`theta_values`**, **`gain_values`** and **`min_db`**. The later one stands for the "central" dB value of the plot. The examples showed the result for `min_db=-20`.

6) Write a script called 'main.m' that defines the array parameters (**d**,**a**), generates the plot of the array factor in linear axis, generates the polar plot of the array gain in dB, computes the radiation pattern parameters (maximum gain direction **`theta_max`**, the mainlobe beamwidth **`delta_theta`** and the directivity **`D`**) and shows them in the screen.

Matlab commands: **`linspace`**, **`exp`**, **`cos`**, **`plot`**, **`axis`**, **`text`**, **`hold`**, **`1i`**, **`log10`**, **`find`**, **`<`** and **`>`**, **`abs`**, **`.'`**, **`figure`**, **`ones`**, **`.*`**, **`pi`**, **`xlabel`**)

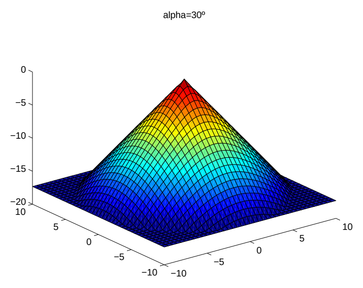
Exercise 9. Input and output arguments in functions

Create a function **`cones`** with a help message given by:

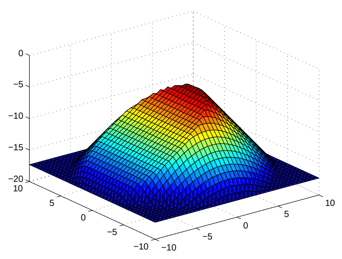
```
» help cones
CONES: function that plots cones and cone cuts.
[xx,yy,z]=cones(alpha)          Compute a cone with vertex at the origin and
                                vertex angle equal to alpha (degrees)
                                To plot it use surf(xx,yy,z) or contour(xx,yy,z)
[xx,yy,z]=cones(alpha,beta)     Compute the cone cut using a plane with angle
                                beta (degrees)
cones(alpha,'3D') or cones(alpha,beta,'3D'): 3D plot of the cone or the cone cut
cones(alpha,'C') or cones(alpha,beta,'C'): contour plot of the cone or the cone cut
```

Type the following two commands to check the correct execution of the function

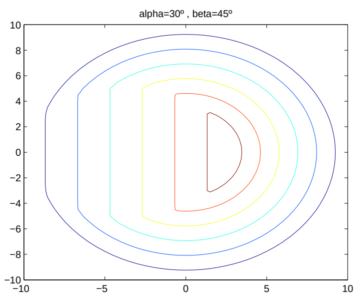
```
» cones (30,'3D')
```



```
» [xx, yy, z]= cones (30,45);
» surf (xx,yy,z)
```

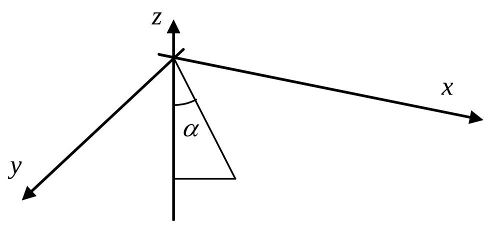


```
» cones (30,45,'C')
```

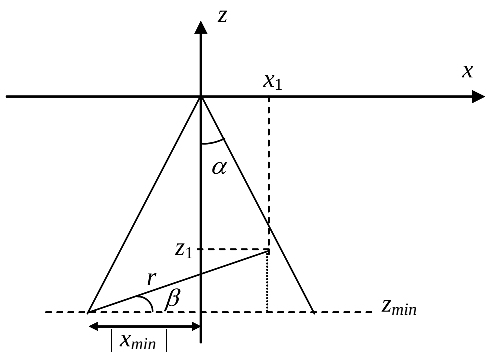


[1] Greek symbols can be introduced in Matlab figures using LaTeX notation: the symbol 'Ψ' can be displayed asking Matlab to write the string '\Psi'.

The following diagrams illustrate how to obtain the values of z for any pair of (x,y) (see figure on the left) and how to obtain the values of z for a plane that intersects with the cone in the values $x = x_{min}$, $y = 0$ and $z = z_{min}$ (see figure on the right).



$$\left.\begin{array}{l}\mathrm{sen}\,\alpha = \sqrt{x^2+y^2}\\ \cos\alpha = z\end{array}\right\} \Rightarrow z = -\frac{\sqrt{x^2+y^2}}{\mathrm{tg}\,\alpha}$$



$$\left.\begin{array}{l} r\cos\beta = |x_{min}| + x_1 \\ r\,\mathrm{sen}\,\beta = z_1 - z_{min}\end{array}\right\} \Rightarrow z_1 = \mathrm{tg}\,\beta\cdot\left(|x_{min}|+x_1\right)+z_{min}$$

Cone plot:

Write the statements that implement the following actions:

1) Generate a vector **`x=-10:0.5:10`** (41 samples) and a vector **`y=x`**.

2) Use **`meshgrid`** to obtain **`xx`** and **`yy`**. Use them to obtain the **z** values for the cone.

3) Check that the central column of **`xx`** (the 21st column) contains the central value of **`x`**. Idem for the central row of **`yy`**. In order to plot the "floor", assign the value **`z(1,21)`** to all values **`z`** which original value is less than **`z(1,21)`** (alternatively, you can assign the value **`z(21,1)`** to all **`z`**'s which original value is less than **`z(21,1)`**). (**`find`**, **`<`**, **`max`**, **`for...end`**)

4) Plot the cone with **`surf`** and the contour plot with **`contour`**. Label the plot with a title (**`title`**) indicating the α value in degrees (**`num2str, sprintf`**).

Cone cut:

1) Write the statements that implement the cone cut.

Use **`nargin`** and **`nargout`** (or **`varargin`** and **`varargout`**) to control the function result for the different usages defined in the help message.