Problem set 2b: M-file Programming

Handed out: Friday, October 11, 2024
Due: 23:55pm, Thursday, October 24, 2024

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

- One file, named `your_name_E2b.pdf`, with the solutions to the exercises in this set, following the template available in the virtual campus.
- The second file, named `your_name_E2b.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 for more details on this subject.
Please, incluse help information in every function implemented, so that the **`help`** command will provide that information to the user.
Comment: Remember that, for efficiency reasons, MATLAB programs should be written avoiding as much as possible the use of loops, resorting to vectorization.

Exercise 3. 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 implement a Morse encoder.

The dash-dot sequences associated to each character are shown in the table 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 in the code.

Encoding is done as follows:

- The duration of a dash is equivalent to three dots.
- Dots and dashes within a letter are separated by a short silence, equal to **one dot duration**.
- The letters in a word are separated by a silence equivalent to the duration of **three dots** (or one dash).
- The words are separated by a silence equivalent to **seven dots**.

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

As an example of Morse encoding, the text "I'm here" must produce the symbol sequence

```
.. .----. --   .... . .-. .
```

that can be represented by the numerical vector

```
[1 0 1 0 0 0 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 0 0 0 1 1 1 0
1 1 1 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 1 1 0 1 0 0
0 1],
```

where a one means a dot-duration tone and a zero means a dot-duration silence.

You are asked to:

1. Write a MATLAB function `pulse_seq = morse_encoder(message)` such 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 `message`: the text to be encoded (a character array, like `"hello world"`).
   - If no output parameters are specified in the function call (i.e., `nargout` is zero), then the Morse-encoded sequence of dashes ('-'), dots ('.') and spaces (' ') is printed, so that a single space corresponds to a letter separation, and three consecutive spaces correspond to an interword separator.
   - It returns the output argument `pulse_seq`, if specified in the function call, consisting of a numerical vector of 1's and 0's (not a string!), where each 1 and 0 correspond respectively to a dot-duration pulse and silence (e.g., a dash is mapped to `[1,1,1]`, while two consecutive dots are mapped to `[1,0,1]` because of the symbol separating silence, see the numerical vector in the above example).

   Useful commands: `load`, `find`, `switch`, `for...end`, `==`.

2. Write a MATLAB function `morse_beep(pulse_seq,tone_freq,dot_duration, sampling_freq)` such that:

- It accepts as input parameters:
  - The numerical vector **`pulse_seq`** as defined above.
  - **`tone_freq`**: the frequency (in Hz) of the dash and dots sounds.
  - **`dot_duration`**: the dot duration (in seconds).
  - **`sampling_freq`**: the sampling frequency (in Hz) of the produced sound.
- It has no output parameters.
- The function must compute the convenient number of samples per dot, using the values of the provided sampling frequency and dot duration. Notice that the resulting audible tone must be (nearly) independent of the employed sampling frequency.
- Then, it produces the Morse sound corresponding to the sequence in **`pulse_seq`**.
- It also depicts in a figure the analog waveform of the produced sound.

Useful commands: **`plot`**, **`sound`**, **`repmat`**, **`cos`**.

3. Write a MATLAB script that calls the previous two functions to produce the Morse code of the text "**`MAE - SPRING 2024`**" with parameters:

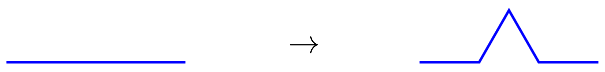
   `dot_duration` = 0.06sec **`tone_freq`** = 750 Hz **`sampling_freq`** = 8000Hz

   Compare the sound with the same sequence and parameters, but with a sampling frequency of 4000Hz (it should be the same, but perhaps with a slightly reduced quality).

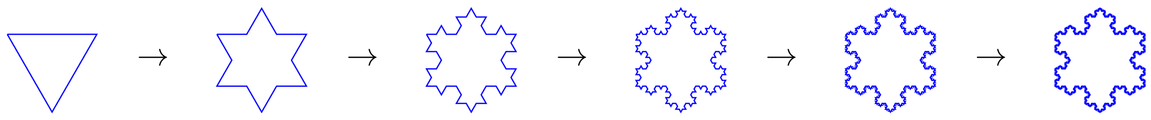
Exercise 4. Koch fractal curve.

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

- Start from an equilateral triangle of unit length sides (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:

- Any polygonal curve in the XY 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:

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

  where (x,y) is any point in the original curve while (u,v) is the corresponding point in the transformed curve.

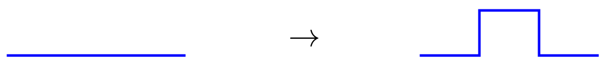
- By cleverly using the operator '**:**' the curve of order $n+1$ can be obtained from the curve of order n with a single **for** loop with as many iterations as the number of segments in the basic triangular pattern above (that is, irrespective of the order n only one loop of length 4 is required). It can even be implemented with no explicit loop at all.
- A common mistake is to repeat the endpoints of the transformed segments, that causes a huge memory overhead when the transformation is iterated. Check that the only point repetition occurs at the endpoints of the whole curve (which is unavoidable because the curve is closed).

In this exercise, you are asked to:

1. Create a function **koch** such that **M=koch(n)** outputs a two-row matrix **M**, containing the x and y-coordinates of the vertices of the n-th curve. If no output argument is provided, simply calling **koch(n);** plots the curves of all intermediate orders, vertically piled up (see the last figure below). Your function should contain at most two nested loops: one of length **n** (each iteration generates one curve) and, if needed, another one of length 4 (the number of segments in the triangular pattern described above). **Do not use recursive calls**.

   A simple way to stack the curves vertically is using **hold on** to gradually add new curves, and adding a suitable constant to the y-coordinates on every curve to avoid any overlapping.

   Useful commands: **plot**, **hold on**, **for**, **:**, **axis equal**, **axis off**, **nargout**.
2. Modify the previous function, and call it **genkoch**, to admit two extra input parameters: An input matrix **Patt** containing an arbitrary polygonal curve with endpoints $(0,0)$ and $(1,0)$, and an extra matrix **M0** defining the starting (order zero) curve. The matrix **Patt** should have two rows containing the x and y-coordinates of the vertices of the polygonal curve to be used as the transformation pattern. For example, for the rectangular curve pattern shown below and using an horizontal segment as starting (order zero) curve

   ```
   Patt=[0,1/3,1/3,2/3,2/3,1;0,0,.25,.25,0,0];
   M0=[0,1;0,0]
   ```

   the result of **genkoch(5,Patt,M0);** is shown on the right column of the figure, while the left column has been generated with **koch(5);**.