MORSE CODE: 1ST EXERCISE

We are going to create a function that takes as input a text file and as output gives a vector containing the text file's message translated into Morse code. Additionally, this function will create a WAV file with the Morse-encoded message.

Start by creating a script called morse_code (which we will turn into a function later) and the first instruction should be a variable called *text*, whose value is the string 'welcome back, my friends, to the show that never ends'. We will use this string for testing and later replace it with a text file.

1 THE DICTIONARY

We will first generate the text-Morse dictionary. We will do this in two separate variables of the same size 39x1, where 39 = 26+10+3, which are, respectively, the 26 letters of the Latin alphabet, the 10 digits (1-9,0), and 3 for the comma, the period and the space.

1. Google the International Morse Code. Wikipedia has a good-enough entry, though it does not have the code for the punctuation marks (comma = - - . . - -, period = . - . - . -). A point will be a single 1, a dash will be three 1s in succession, and a silence will be a 0, We will use an inter-symbol space of one silence, inter-character of two silences, and the space (between two words) of three silences. For example, comma = 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 1 0 0.
2. Create the vector ABC, assign to it a string whose elements are abcde… xyz1234…0,. (add a space at the end).
3. Create the cell array MORSE, in which you will save the Morse code for all the alphanumeric characters, in the same order of ABC. For example, the first element should be 1011100, that is, point-blank-dash-blank-blank, where the first blank is the inter-symbol silence, and the last two blanks are added to create the inter-character silence. A pre-filled example of this can be found in the file define_morse_dictionary.m; you can fill out the missing entries.
4. **(OPTIONAL)** Create another script called inverse_morse. Copy the dictionary into that script and save it. We will do further work on it later.

2 THE TRANSLATION

We will now use the vector ABC and cell structure MORSE to translate the text string. The steps are as follows:

1. Start by initializing an empty morse_out vector, which will later contain the Morse translation of our text message.

2. Using a For loop, go through each character in the text string. This means that you should create a For loop that goes from 1 to the number of characters in the text string.
3. Inside this For loop, find the element of ABC that equals the current character from the text string. Do this by using the function *find*. For example, if the current character is ‘c’, using the *find* function will return a number 3, which is the index of the element of ABC that equals ‘c’ (i.e. ABC(3) = ‘c’). Save this number to a variable current_char.
4. Using current_char, add to morse_out the corresponding Morse code for the current character (in the example above, that would be MORSE{3} or MORSE{current_char}).
5. Use the string *text* to test your results so far. The result morse_out should be a vector with 1s and 0s only.
6. **(OPTIONAL)** You can now start working on the Inverse Morse problem, which is explained at the end of this document (section ‘Inverse Morse Code’).

3 THE SOUND FILE

We will use the morse_out vector (of 0s and 1s) to create a sound file. First, we will create a Beep sound and then we will use it to create the whole translated message.

3.1.1 THE BEEP

First we will create a Beep sound. A Beep is characterized by four numbers: frequency (how high or low the tone is), volume (how loud or quiet the sound is), sampling rate (technical term, explained below) and the duration (in seconds).

1. Create four variables for the four parameters above. Their values should be, respectively, 1000, 1, 4400, 0.15.
2. We now need to create a time sequence for the sound. A single tone is given by a sinusoidal curve Y = A*sin(F*T), where T is a time vector as long as the duration of the beep; A is the volume (from 0 to 1) and F is the frequency (tone). To create this, first create a vector T from zero to tone_duration, in increments the size of 1/sampling_rate. Then, create the sound vector Y with the formula given above. Hint: Matlab has a built-in sine function called *sin*.
3. Test the results by using the function *sound*. See the help file for details.

3.1.2 THE MESSAGE

We will now create a vector that will contain the message in Morse, in a form that Matlab can use to create sound. To do this, create an empty vector sound_out. Scan through the elements of morse_out, and for every 1 add a beep (given by Y from the previous part) and for every zero add a silence (the same duration of the Beep). Hint: you can do this with a single For loop with one line inside (plus the initialization of the sound_out variable). Test your results with the function *sound*; the sound will be long, but you can cancel it by pressing Ctrl+c, while on the command window.

4 Turning it into a function

We will now turn the whole script so far into a function that takes as input a text file or string, and gives as output both the morse_out vector and the name of the file where the sound was saved. We will also create a new function that defines the dictionary. This step is the one we do first:

1. Create a function called define_morse_dict, with no inputs and two outputs. Call these outputs ABC and Morse.
2. Copy the first part of your morse_code script into it. That is, the lines where you define ABC and MORSE. Save and exit.

Turning our script into a Function is simple: enclose the code within the definition of a function (one input, two outputs), and add one bit at the beginning that does the following:

1. First, check whether the input is the name of an existing file. Do this by using the function *exist*.
2. If the input is a file, read the file using *fileread*
3. If it is not a file (or the file does not exist), use the input as the string itself; that is, use the string that was given as input, as the string to translate.
4. Execute the function define_morse_dict and save its outputs to ABC and MORSE. Do this before the Tanslate part.

INVERSE MORSE PROBLEM

This is basically the same problem as before. You can use the same dictionary and follow roughly the same procedure. What changes is that you now have to identify the inter-character and inter-word separations. To keep it simple, we will use a trick to do this.

Create a function called inverse_morse, whose output is a string (the translated message) and whose inputs are a vector of 1s and 0s (call it morse_in), which is a Morse-encoded message.

First, we will identify the inter-word spaces in the messages. We will make use of the fact that an inter-word is the only instance of three consecutive zeros in the message. We will look for these three consecutive zeros in reverse (from the end of the message backwards) and replace them with something else:

1. Start a For loop that begins at the end of the message and ends at the third element (see below for an explanation). This will go through all the elements of morse_in.
2. Inside this loop, check (with an IF statement) if the current (m-th) character of morse_in is a zero, and if the next one ((m-1)th) and the next one (m-2) are too. If they all are, pick one (the m-th, for example) and turn it into a 5. Turn the other two into -10. This will become clear later. Because you are always checking two characters following the current one, the For loop must end 3 characters before the beginning; otherwise, you would get an error from Matlab.

Now, we will find the inter-character spaces, which are two consecutive zeros. Since we already got rid of the inter-word spaces (three consecutive zeros), any two consecutive zeros are now an inter-character space. So:

1. Start a For loop as before.
2. Check whether the current element of morse_in and the next one (m-1) are both zero.
3. If they are, replace one with a 3, the other with a -10.

Having done this, we will delete all the elements of morse_in that equal -10. This is easily done using logical indexing; for example, if you want to eliminate all elements of vector A that equal 5, you write *A(A==5) = []*. Do this with morse_in. As you can now see, every 3 in morse_in represents the beginning of a new character and every 5 represents the beginning of a new word. Ones and zeros are, as before, the Morse code itself. We will use this to translate.

1. Initialize a variable *flag* with value 1 and a *text* empty variable. We will use them later.

2. Begin a For loop from 1 to the number of elements in morse_in. Because of the elements we eliminated earlier, you have to find again the size of morse_in.
3. With an IF-Else IF check, determine if the current character of morse_in is a 3 or a 5.
4. If it is a 3, then all the preceding elements (starting from the last value of *flag*, which is 1 at first) together represent a letter or number. Take them all and search through the Morse dictionary from the previous exercise. Because it is a cell array, you cannot use *Find* and will instead have to use a For loop to compare. Once you find the right one, add the corresponding alphanumeric character to the variable *text*. This is almost identical to the previous exercise.
5. If it is a 5, then add a space to *text.*
6. Whether it is a 3 or a 5, after you have added the character to *text*, save the current index of the For loop plus one into *flag*. That way, the next time you find a 3 you can scan starting from *flag*.