OBJECT ORIENTED PROGRAMMING

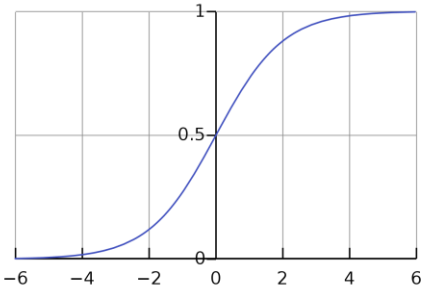
LABORATORY 2

OBJECTIVES

The main objective of this laboratory is to teach student about how to use functions, operate with arrays and write simple, user-friendly, menu-based application in C/C++. Also, you will learn about *doxygen*, the *de facto* tool for automatic documentation generation in C++.

PROPOSED PROBLEMS

1. Sigmoid $\sigma(x)$ is a mathematical function with an *s*-shaped appearance that maps the input value x into the interval [0, 1]:



$$\sigma(x) = \frac{1}{1+e^{-x}}.$$

It is often used in machine learning, for binary classification problems (i.e. yes-no output), to transform the output of a classifier from a real number into the range [0, 1], such that this output can be interpreted as a probability.

Write a *function* that takes as argument a real number x and returns the sigmoid of that number.

2. You have developed a smile detection algorithm that takes as input a video sequence of a person and outputs, for each frame of the video, a real number that encodes how much the person smiles in that video frame (positive values - it is more probable that the person is smiling, negative values - it is more likely that the person has another expression).

   You are given an array of real numbers that contains the output of this smile detector.

   - Apply the sigmoid function element wise to this array. Let's call the result of this operation *S*. Now, for each frame of the array you have a floating point number which can be interpreted as the probability of smiling.
   - Establish a threshold $th \in (0, 1)$ to decide if a person was smiling. Then, write a function to find the longest contiguous subsequence in the array *S*, where the

probability of smiling is greater or equal to *th*. You should return to the caller of the function the starting and the ending index of this subsequence.

- Write multiple several test cases to prove your solution is working properly. Think of all the corner cases that might occur!

3. Write a menu-based application that allows the user to choose between these two functions that you wrote functions you wrote.

4. Write comments for all the functions you wrote (in doxygen format) and use doxygen to generate the documentation of your project in .html format.

5. Caesar cipher. Breaking the cipher.
   Use a brute force approach to break Caesar's cipher. This is a really simple solution: you know that there are 26 letters in the English alphabet, so it will only take you 25 tries to test all the possible solutions.

   Write a function that takes as input an encrypted string and iteratively tries to decrypt the text with all the possible shifts (1, 2, ..., 25). The decrypted text is displayed and the user is prompted to establish if the text is intelligible. If so, the process ends.

EXTRA CREDIT I

Use frequency analysis to break Caesar's cipher (this is another brute force method to break the cipher).

Frequency analysis relies on the fact that some letters (or combination of letters) occur more in a language, regardless of the text size. For example, in English the letters E, A are the most frequent, while the Z and Q are the least frequent; miscellaneous fact: see *Etaion shrdlu*[1]). The distribution of all the characters in English is depicted in the figure below:

E	T	A	O	I	N	S	H	R	D	L	U	C
12.7	9.1	8.2	7.5	7.0	6.7	6.3	6.1	6.0	4.3	4.0	2.8	2.8
M	**W**	**F**	**Y**	**G**	**P**	**B**	**V**	**K**	**X**	**J**	**Q**	**Z**
2.4	2.4	2.2	2.0	2.0	1.9	1.5	1.0	0.8	0.2	0.2	0.1	0.1

Figure 1. *Distribution letters in English (image source: https://ibmathsresources.com/tag/vigenere-cipher/)*

The idea of this method is to compare the frequency of the letters with the Chi-Squared distance:

$$\chi^2\,(C,E) \;=\; \sum_{i\,=\,'a'}^{i\,=\,'z'} \frac{(C_i \;-\; E_i\,)^2}{E_i}$$

, where C_i represents the occurrence of the i[th] character, and E_i is the expected count of the i[th] character of the alphabet.

The formula seems complicated at a first glance, but it is really not that complicated. Basically, for each possible character (i goes from 'a' to 'z'), we measure the discrepancy between how often it appeared in the encrypted text (C_i) and how often it is expected to appear in English texts (E_i); the difference $C_i \;-\; E_i$ is squared such that we remove negative signs. The division by E_i is simply a normalization factor.

The lower the Chi square distance $\chi^2\,(C,E)$, the more similar the histograms C and E are.
As this algorithm is also a brute force method to break the cipher, you should compute the histogram for all possible shifts, and compute the Chi Squared distance between these histograms and the average distribution of the characters in English. The shift with the lowest Chi Squared distance is the solution.

You can find more information about this algorithm here:
https://ibmathsresources.com/2014/06/15/using-chi-squared-to-crack-codes/ .

To sum up, to solve this problem you need to:

[1] https://en.wikipedia.org/wiki/Etaoin_shrdlu

- Write a function that reads the distribution of the letters from a file (*distribution.txt*) and stores it into an array. The frequency of the letter 'a' is stored on the first line of the file (as a floating point number), the frequency of the letter 'b' is stored on the second line of the file etc.
- Write a function that computes the normalized frequency of each character (a histogram) in a text.
- Write a function that computes the Chi squared distance between two histograms.
- Write a function that breaks the Caesar's cipher using frequency analysis: iteratively shifts the encrypted code through all the possible permutations, computes the Chi squared distance between each permutation and the approximate distribution of letters in English, and returns the permutation with the least Chi squared distance as the solution.
- Finally, create a user-friendly menu based application for Caesar's cipher related tasks.

Break the message:

Uf ime ftq nqef ar fuyqe, uf ime ftq iadef ar fuyqe, uf ime ftq msq ar iuepay, uf ime ftq msq ar raaxuetzqee, uf ime ftq qbaot ar nqxuqr, uf ime ftq qbaot ar uzodqpgxufk, uf ime ftq eqmeaz ar xustf, uf ime ftq eqmeaz ar pmdwzqee, uf ime ftq ebduzs ar tabq, uf ime ftq iuzfqd ar pqebmud.