Master in Biophysics, 2010-2011
Universidad Autónoma de Madrid

Matlab Programming

Class problems

These problems will be explained during the classes, and sufficient time will be given for the students to complete them. Students are strongly encouraged to do all problems. Some problems are mandatory, and will count 50% of the final grade. These programs must be e-mailed to alfonso.perez.escudero@gmail.com or jklett@cbm.uam.es after each class. If some mandatory problems are not complete at the end of the class, they must be e-mailed before the deadlines that will be posted in the subject's web page. Please, mail each program in a separate .m file, with the name indicated in this document. Write your name as a comment in the first line of the program.

The remaining 50% of the grade will correspond to the final exercise. Each student must select one among the several options that will be offered.
Comments in the programs and text messages may be in English or Spanish. But please, keep the name of the programs as it is at least for mandatory programs, so that we recognize them easily.

Day 1 problems

1. **HelloWorld.** Write a script that prints "Hello World" on the screen.
2. **Pitagoras.** Write a script that asks the user for the length of both catheti of a right triangle and stores the hypotenuse in a variable, using the well known formula $h = \sqrt{c_1^2 + c_2^2}$, where c_1, c_2 are the catheti and h the hypotenuse. The program must show a sentence like "The hypothenuse is 5".
3. **Pitagoras_limpio.** Create a script identical to the previous one ("Pitagoras"), but in this case make sure that all variables needed (apart from the hypotenuse variable) are erased at the end of the program. Also, the script should not display the assignment of values to the variables on the screen.
4. **Adivinanza.** Write a script that asks the user for a number between 1 and 10. In case that the number typed equals 5 it should print on the screen "Correct". It should also show "Thanks for playing" at the end of the script execution. Make sure all variables are erased afterwards.
5. **Adivinanza2.** Create a script identical to the previous one ("Adivinanza"), but that prints "Incorrect" if the number is not 5.
6. **Adivinanza_pistas (mandatory).** Create a script identical to "Adivinanza2", but that also prints if a wrong answer is bigger or smaller than 5.
7. **Pesadito.** Write a script that prints 10 times the sentence "Soy un pesao".
8. **TablaMultiplicar.** Write a script that asks for a number to show its multiplication table, and then shows the results of the multiplication of that number times 1, 2...10. For example, in the case of "table of 3" the program must print 3 x 1 = 3
   3 x 2 = 6, etc.
   Make sure all variables created by the program are cleared at the end.
9. **Factorialaco (mandatory).** Problem that gathers a long and boring tradition in the Programming classrooms. Write a script that asks for an integer number and calculates its factorial, using the maybe-not-so-well-known formula $n! = n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1$,

where *n*! is the factorial of *n*. You will need a "for" loop. No vectors are necessary. The result must be stored in a variable and the rest should be erased.

10. **Fibonacci.** The Fibonacci sequence is 0, 1, 1, 2, 3, 5, 8, 13... (each number is the sum of the two previous ones). Write a script that asks how many terms of the progression you want, and then prints them in the screen (separated by commas). All variables must be erased at the end. Note: You may use vectors for this program, but it is not necessary (we recommend to try this exercise without vectors).

Day 2 problems

1. **PesaditoWhile.** Modify the "Pesadito" script so that it gives the same output but using a "while" loop.
2. **AdivinanzaWhile.** Modify the "Adivinanza" script so that it keeps asking input numbers until the user guesses the correct one. Extra work for skilled programmers: you may want to try with a random number between 1 and 10 instead. You should check then out the "rand" and "ceil" functions.
3. **AdivinanzaCompleto.** Same as the previous one, but the player has only 3 opportunities. The program must show a different message when the correct result is found than when the player runs out of opportunities.
4. **Tabla5.** Write a script that prints the results of multiplying all the numbers from 1 to 5 between them ("all against all"). Hint: It is not necessary to use fprintf to show the result, just ask Matlab to show the variable.
5. **Tabla5_nopairs.** Create a script similar to the previous one ("Tabla5") but that avoids printing the same pairs twice (1*5 = ~~5*1~~). Hint: try with "i=j:N"…
6. **Sinxplot.** Write a script that represents graphically the following functions: sin(*x*), sin(2*x*), sin(3*x*), sin(4*x*) for $0 < x < 4\pi$. The graph should show the values with 0.1 intervals. Remember using loops and vectors!
7. **Suma_script (mandatory).** Write a script that finds the sum of the components of a vector (which must be previously defined). You are not allowed to use Matlab's function "sum", which does the same thing as this script.
8. **Acumulada.** Write a script that finds the "cumulative vector" of a previously defined vector. Each element of the cumulative vector stores the sum of all previous elements of the original vector. For example, if the original vector is [1 1 3 2 4] the cumulative vector is [1 2 5 7 11]. This script does the same thing as Matlab's function "cumsum". You cannot use this function for the script.
9. **Encontrar_script (mandatory).** Write a script that requires two variables to be previously defined: a vector and a number. The program must find the elements of the vector that are equal to the number, and store their indices in another vector. If there are no coincidences, the created vector should be empty. You are not allowed to use Matlab's function "find".
10. **ProtAlign (mandatory).** Write a script that represents graphically the identities in sequence with itself for the following protein:

    'AAAEYDSLEYDSLGYENEAAAEYDSLEYDSLGYENE'

    The output should be a binary matrix (1=same amino acid; 0 otherwise). Use for the output the "imagesc" function.

Day 3 problems

1. **Suma.** Same as suma_script, but in functional form: Write a function that takes a vector as input, finds the sum of its components, and gives it as output. You are not allowed to use Matlab's function "sum", which is equivalent.
2. **Encontrar.** Same as Encontrar_script, but in functional form: Write a function that requires two input arguments: a vector and a number. The output must be a vector which contains the indices of the elements of the input vector that are equal to the input number. If there are no coincidences, the output should be an empty vector. This is identical to the "find" Matlab funtion. Do not use it!
3. **Encontrar_eficiente.** Same as Encontrar, but must use preallocated variables.
4. **Maximo.** Write a function that takes a vector as input and gives two outputs: The maximum of the vector, and the index corresponding to it. If there are more than one element equal to the maximum, the index must correspond to the first one. This is equivalent to the "max" Matlab function. You are not allowed to use it.
5. **Ordernar (mandatory).** This function will have one input argument, which will be a vector. It will have two output arguments, which will be two vectors of the same length as the input one. The first output vector must be the input vector, but with its elements sorted in increasing order. The second output vector must contain the positions occupied by each element of the sorted vector in the original one. This function is equivalent to Matlab's function "sort". As you probably imagine by now, you cannot use "sort" for this exercise. You may use functions "min" and "max".
6. **Bisiesto.** Write a script that takes as input a year and has no output arguments. It must display a message showing if it is a leap year or it is not. The leap year rules are the following: A year is leap if it is divisible over 4, except those divisible over 100. But also, years divisible over 400 are leap. For example, 1996, 2000 and 2004 were leap years. On the contrary, 1900 was not, because it is divisible over 100 and not divisible over 400. In order to know if *a* is divisible by *b*, the remainder of *a*/*b* must be 0. For Matlab, *a* is divisible over *b* if *rem(a,b)==0*.
7. **Bisiesto_comprimido.** Identical to the previous one ("Bisiesto"), but with only one "if" statement (you have to use logical operators "&", "|").
8. **Histograma (mandatory).** Write a function with two input arguments: A vector with the data, and a scalar with the number of bins. It must generate the histogram of these data, plotting it as a bar graph (use instruction "bar"). Also, it must have two output vectors. The first one must contain the number of data falling in each bin, and the second one must contain the centers of the bins. Bins must be equispaced, and must contain the full range of data (that is, the lower extreme of the first bin must match the minimum datapoint, and the higher extreme of the last bin must match the maximum datapoint). Each bin must contain data that are higher than its lower extreme, and lower or equal than its higher extreme (except for the first bin, which must also contain data equal to its lower extreme). This function is similar to Matlab's function hist, and you cannot use hist for this exercise. You may create the data vector with instruction randn (thus, the histogram will be a nice gaussian).

Day 4 problems

1. **ReadProtein.** Write a function whose input is the name of a file containing the positions of all atoms of a protein (these files are in proteins.zip). It must have two outputs: a matrix with size [N 3] (where N is the number of atoms) with the (x,y,z) coordinates of each atoms, and a vector with the charges of all atoms. Note: Once you have the coordinates of the atoms, you can use plot3 (with '.' markers) to visualize the protein. Hint: Use importdata for this problem.

2. **BoxSize.** Write a script that calculates the dimensions of a box that contains the protein (you will be provided of a coordinate's file of the protein atoms). You may want to use the Max and Min functions from previous exercises. Although, you are allowed in this case to use the "max" and "min" Matlab functions. (NB: Do not worry about rotating or translating the coordinates of the protein! Just calculate the dimensions of the box with the protein coordinates as they are given).
3. **Affinity.** The affinity between the receptor and a ligand depends (approximately) on the coulombic potential between them. In order to estimate this affinity, write a function or script that:
    a. Reads the receptor file and one ligand file (use function ReadProtein for this).
    b. Calculates the sum of coulombic interaction of all the protein atoms against all the ligand atoms The coulombic interaction between two atoms is the following:

$$\text{Atom 1: position } (x_1, y_1, z_1), \text{ charge } Q_1$$
$$\text{Atom 2: position } (x_2, y_2, z_2), \text{ charge } Q_2$$
$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2};$$
$$Coul = \frac{Q_1 Q_2}{d};$$

    Note: The affinity depends on the relative positions of receptor and ligand. But you do not need to take care of this, because the files provided correspond to a ligand which is located in the active part of the receptor (where the affinity is maximal). You may see this by plotting together the receptor and the ligand (use plot3 and different colors for each one).
4. **Affinity_full (mandatory).** Write a script that:
    a. Reads the receptor file
    b. Reads, one by one, all the ligands (use a "for" loop, and construct the names of the files by appending text strings to the counter. Do not forget to use num2str where appropriate).
    c. Computes the coulombic potential between each ligand and the receptor. Note: You may do this after each ligand is loaded, and then overwrite the next ligand.
    d. Plots the coulombic potential for each ligand with the protein.
    e. Shows a message saying which ligand is binding with the lowest coulombic interaction energy (this means highest affinity).
    f. Plots the receptor and the ligand with lowest coulombic potential together (in different colors).
5. **WriteProtein**. Write a function that takes the coordinates, radii and charges of a set of atoms, and creates a .dat file, which may be read by ReadProtein. Do not forget to close the file at the end of the function. You will need the functions fopen, fclose and fprintf (or fwrite).

<u>Day 5 problems</u>

1. **Affinity_efficient.** Modify the problems of the fourth day, so that they run as efficiently as possible.
2. **FindCenter.** Write a function that opens the image manchaca.tif and finds the center of the shadow. Calculate the center in two different ways: as the center of mass of the

pixels, using their grayscale level as “mass”, and by setting a threshold and finding the center of the region with pixels above the threshold.

3. **Integrator.** Write a function that finds the area under the curve $y= x+x\cdot\sin(x)$, between $x=0$ and $x=4\pi$. This area must be approximated by the sum of the areas of rectangles that fit under the curve. Decrease the width of these rectangles to increase precission. The algorithm must finish when three consecutive iterations have not increased precission more than 10^{-5}. Write this program as efficiently as possible.