TRABAJO PRÁCTICO N°0

- Fecha de entrega: 28/09

El formato de entrega de este trabajo práctico será el siguiente:

1. Los ejercicios se entregaran en archivos .m con el siguiente formato de nombre:

```
ej_m_inciso_n.m
```

   y las funciones con el siguiente:

```
ej_m_inciso_n_funcion(par1, par2, .., park)
```

   Ejemplo: Ejercicio 3 inciso b. El nombre del archivo será "ej_3_inciso_b.m", y dentro del archivo, la definición de la función será:

```
function ej_3_inciso_b(x, y, z):
    codigo
end
```

2. Las respuestas a las preguntas conceptuales deben ser escritas como comentarios en el código.
3. Los errores que surjan como resultado de la correcta realización de algunos ejercicios también deben ser escritos como comentarios en el código.
4. Los diagramas de flujo pedidos deberán entregarse como imágenes.
5. Finalmente, todo lo anterior deberá ser incluido en un archivo comprimido y subido al espacio asignado en Moodle, previo al segundo parcial.

Recordar que para la aprobación de este trabajo práctico, todo los ejercicios deberán estar bien resueltos.

PARTE 1 - INTRODUXION A LA PROGRAMACIÓN

Introducción

La computadora es tan obediente como estúpida. Y eso es, a la vez, potente y peligroso. Lea nuevamente el título de esta práctica, asumiendo que:

- Se ha horrorizado por la ortografía de la cátedra
- Comprendió de qué se trata la práctica

Podemos marcar la primer e insalvable diferencia entre usted Y la máquina: su inteligencia le proporciona capacidad de interpretación, sobre la base del lenguaje español, un error en la sintaxis no fue obstáculo para la comprensión del mensaje. Esto es totalmente imposible para la máquina, el lenguaje que usaremos para la cátedra (MATLAB), debe ser escrito con una sintaxis totalmente exacta y rígida a fin de que el mismo pueda ser comprendido por la computadora.

Errores de tipeo, forma o formato, darán lugar a errores o resultados diferentes a los buscados. Tenga esto siempre presente al momento de interactuar con la computadora, los procesos que para Ud. son elementales y en muchos casos inconscientes, deben ser explicitados en forma secuencial y completa a fin de que la máquina pueda reproducirlos.

1. **Asignación de variables y operaciones básicas**

   a) Ejecutar los siguientes comandos

```
a = 2
a = a + 2
2 + a = a
a = b
```

   ¿Se ejecutaron todos sin error? No. ¿Cuáles fueron los errores? ¿Qué significan? ¿ Cómo puede evitarse?

   b) Calcular lo siguiente:

$$\frac{5^2 + 15}{2}$$
$$\sqrt{(-4)^2}$$
$$\left(\sqrt{-4}\right)^2$$
$$(1, 2, 3) \cdot (4, 5, 6)$$

   Nota: la última operación es el producto punto entre dos vectores.

   c) Repetir el inciso b), pero asignando variables en lugar de escribiendo uno por uno los números.

   d) ¿Hay restricciones para los nombres de las variables?

2. **Entrada de datos por teclado, y salida de datos por pantalla**

   Para dar salida de texto por pantalla se usan los comandos **disp()** y **printf()**. Para solicitar entrada de usuario por teclado se usa el comando **input()**

   a) Ejecutar los siguientes comandos:

```
disp(3)
disp("3")
printf(3)
printf("3")
printf("3\n")
```

   Explicar los resultados obtenidos

   b) Pedir que se ingrese por teclado un número de 1 a 10, sumarle 5 y luego mostrarlo en pantalla. Indicación: cerrar cada línea de código con un punto y coma (";"). Usar **printf()** y **disp()** para mostrar el resultado final (o intentarlo al menos). Explicar los resultados.

3. **Vectores y matrices**

   Matlab no significa otra cosa que "MATrix LABoratory". En su nombre tiene implícito su principal poder: manejar vectores y matrices con eficiencia. Los vectores y las matrices son de mucha utilidad, no solamente en Matlab sino en todos los lenguajes de programación, y su utilidad trasciende la pura y estrictamente matemática. Por ejemplo, los vectores pueden usarse como índices, y las matrices como mapas. En definitiva, más allá de su valor, por así decirlo, matemático, deben ser considerados *estructuras* de datos en lugar de *entidades matemáticas*. Aprender a generar, llenar, navegar y manipular estas estructuras de datos es crucial a la hora de programar.

   a) Armar un vector de tres componentes con los números del 1 al 3, otro con los números del 4 al 6 y otro con los números del 7 al 9.

   b) Armar una matriz de 3x3, con los números del 1 al 9.

   c) Construir una matriz con los tres vectores generados en el inciso a) y compararla con la generada en el inciso b).

   d) Transponer la matriz generada en el inciso b) o c) (deberían ser iguales).

   e) Generar una matriz con los vectores generados en el inciso a) transpuestos.

   f) Escribir un pequeño "Manual para generar una matriz en Matlab" en base a la experiencia ganada en los puntos anteriores.

   g) Multiplicar los dos primeros vectores generados en el inciso a) utilizando los operadores "*" y ".*"(punto por).¿Se pueden hacer ambas operaciones?¿Cuál es la diferencia entre ambos operadores? Repetir para el primer vector y el transpuesto del segundo.

   h) Generar dos matrices de 3x3 arbitrarias, no singulares. Repetir el punto anterior para estas dos matrices.

4. **Estructuras de control**

   En todos los lenguajes de programación existen formas de automatizar tareas repetitivas. En Matlab se utilizan *ciclos* o *loops* basados en las instrucciones *for* y *while*.

   a) ¿Cuáles son las diferencias entre los distintos tipos de ciclo?

   b) En qué consiste un bucle infinito (infinite loop)?¿Con qué comandos se puede salir de ellos?

   c) Escribir un programa que sume los números del 1 al 100 usando solamente bucles tipo *for* e instrucciones de decisión *if, else if* y *else*.

   d) Repetir el inciso anterior con bucles *while*.

   e) Modificar los dos programas anteriores para que tomen el número hasta el que hay que sumar por entrada de teclado.

5. **Funciones**

   Una función es una porción de código más o menos autónomo, que puede o no recibir argumentos, puede o no realizar operaciones que los involucren, y puede o no devolver un resultado. A las funciones, en general, pueden "dársele de comer" variables de tipo numérico, vectores, matrices, cadenas de caracteres, otras funciones, o cualquier otra estructura de datos imaginable. Una función, por ejemplo, podría tomar como argumento una imagen y devolver su color predominante.

   Una función que devuelve valores y toma argumentos se define mediante el comando *function* de la siguiente manera:

```
function nombre_de_variable_a_devolver = nombre_de_la_funcion(
    argumento_1, argumento_2, .. , argumento_n)

end
```

Las funciones se pueden llamar desde la ventana de comandos (command window) o desde otro programa o función. Se puede asignar a una variable el resultado de la función, en la forma:

```
resultado = nombre_de_la_funcion(argumento_1, argumento_2, .. ,
    argumento_n);
```

(el punto y coma es para que no muestre en pantalla el resultado de la evaluación de la función y la posterior asignación a la variable "resultado").

a) Construya una función que modifique el programa del punto anterior que sumaba los números del 1 al 100 para que, ahora, sume los números desde uno hasta un fin arbitrario. Revise que el resultado sea el mismo que el programa del punto anterior, usando el 100 como argumento de prueba.

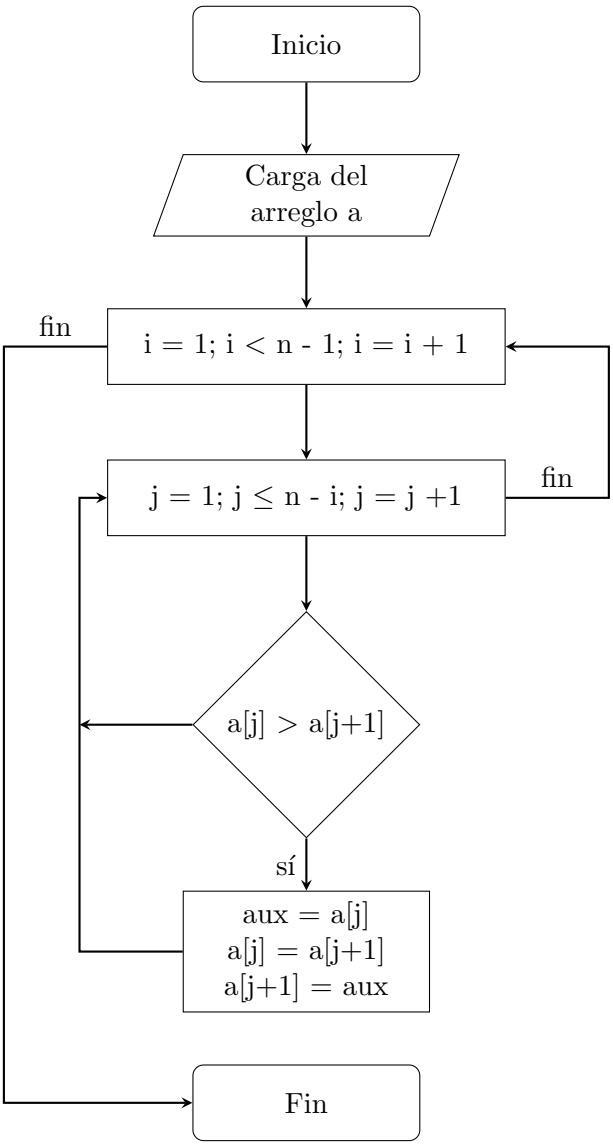
b) Modifique la función del punto a) para que, ahora, acepte también un inicio arbitrario. Revise que el programa devuelva resultados correctos usando 1 y 100 como argumentos de prueba.

c) Escriba una función que devuelva las raíces reales (si existen) de un polinomio de segundo grado con sus coeficientes como argumentos.

d) Escriba un programa que pida la entrada por teclado de los coeficientes de un polinomio de segundo grado, que use la función del punto c) para calcular las raíces del polinomio, y que solamente devuelva raíces positivas, si las tiene. Para pensar: ¿Qué puede hacerse si el polinomio no tiene raíces reales positivas?

6. **Método de la burbuja**

El método de la burbuja es una forma simple de reordenar los elementos de un arreglo de menor a mayor, el cual se puede describir mediante el siguiente diagrama de flujo:



A partir del diagrama de flujo expuesto, programe una función que reciba un arreglo como entrada y entregue como salida el arreglo ordenado.

7. **Gráficos**

   Matlab (u Octave) solamente sabe graficar pares ordenados (puntos). Para graficar una curva (por ejemplo, la gráfica de una función), hay que alimentar a la instrucción encargada de realizar gráficos con **vectores de igual longitud**. Para graficar en 2D se necesita dar dos vectores, uno con las coordenadas X y otro con las coordenadas Y. Para graficar en 3D, se necesita otro vector más con las coordenadas Z. Es obligatorio que los vectores tengan la misma cantidad de componentes, dado que no pueden quedar puntos con un conjunto incompleto de coordenadas (por ejemplo, puntos en 3D que solamente tengan coordenadas en X y en Z, pero no en Y).

   Para realizar un gráfico en 2D se usa la instrucción **plot()**. Esta instrucción toma dos vectores de la misma longitud como argumentos. Cada vez que se llama a la función **plot()**, el gráfico anterior es reemplazado por el nuevo. Para evitar que el *lienzo* (canvas) se borre al realizar un nuevo gráfico, se usa la instrucción **hold on**. Para permitir que se borre el gráfico anterior, se usa la instrucción **hold off**. Si se desea crear un nuevo lienzo, se usa la instrucción **figure()**. El argumento de la función **figure()** es un número entero.

a) Grafique una recta de pendiente 1 y que pase por el origen, entre $x = -5$ y $x = 5$.

b) Grafique una parábola con coeficientes 1,1 y 0 entre -5 y 5.

c) Grafique ambas funciones en el mismo gráfico.

d) Construya una función que, al pasarle como argumento los vectores de abscisa y ordenada, grafique las funciones en el mismo gráfico. Además, debe haber otro argumento de la función que permita establecer los límites del gráfico.

e) Modifique la función anterior para que permita, mediante un argumento, decidir si se grafican ambas curvas en el mismo gráfico o en gráficos separados.

Parte 2 - Representación de números y aritmética finita

Hay dos tópicos de suma relevancia en el análisis numérico que suelen pasarse por alto. Uno de ellos es la forma en que se representan los distintos tipos de números, y el otro es el hecho de que, para las computadoras, los infinitos no existen. Los números enteros que una computadora puede manejar tendrán un máximo, dado por la *cantidad de memoria* que se asigne a cada número, y al tiempo que se quiera emplear en realizar operaciones entre tales números. Con los números *reales*, pasa algo similar: además de estar acotados, *siempre existirá una distancia finita (limitada) entre dos números adyacentes*, cosa que no sucede con la recta real, al ser el conjunto de números reales un *conjunto de medida nula*.

Aritmética finita

Por *aritmética finita* nos referimos a la necesidad de usar un conjunto limitado de caracteres para representar un número determinado. Esto impone restricciones sobre los números que se pueden representar con aritmética finita. Un ejemplo de aritmética finita suele presentarse en la primera clase del curso, en una típica conversación entre el profesor y un alumno aleatorio:

Profesor: ¿Cuánto es 1 dividido 3?
Alumno Aleatorio: 0,33
P: ¿Por tres?
AA: 0,99
P: ¿Ve? Hizo solamente dos operaciones y ya acumuló un 1 % de error

En este caso, el Alumno Aleatorio definió su resultado usando solamente tres dígitos: uno para la parte entera del número, y dos para los decimales. El resultado que dio el Alumno Aleatorio *no es el resultado verdadero de la operación*, pero sí es *el mejor resultado que podía dar con un número de ese "tamaño"*. En ese sentido, cabe realizar la siguiente pregunta:

1. ¿Cuántos dígitos se necesitan para almacenar el resultado *exacto* de la operación 1/3?

Otras preguntas interesantes:

2. ¿Cuál es el número más grande que puede representarse con tres dígitos?¿Y el más pequeño distinto de cero?

   (Nota del JTP: seguro que todos responden esta mal)

   Parte de la respuesta: depende. ¿Se trata de números enteros?¿Pueden tener decimales?¿Se puede hacer "trampa"? Ejemplo de trampa: con tres dígitos se puede expresar 5^{27}.

3. ¿Cuántos dígitos se necesitan para representar el número 5000000 (cinco millones)?¿Y para el número 5430000?¿Y para el número 5432109?

4. Usando una representación con 3 dígitos, ¿Cuál es la distancia entre el número cero y "el número de al lado del cero"?¿Y entre el número 7 y su siguiente? En ambos casos, hay que considerar decimales. ¿Y cuál es el número que le sigue al 10000 con este tipo de representación?

Sistemas de representación

Un amigo mío manda a su hijo pequeño a un jardín de infantes caracterizado por un nivel de "jipismo" (hippismo) algo extremo. Los niños en ese jardín tienen un contacto con la naturaleza que podría calificarse de "cuestionable", al punto tal de que usan arañas para aprender a contar: en lugar de usar los dedos de la mano, sostienen una araña en cada mano y cuentan usando las patas de cada araña, es decir, 8 patas por araña, 16 patas entre las dos manos. Al nene le va

muy bien con sus arañas. Al que no le va tan bien es al padre, ya que no entiende muy bien el sistema numérico que usa su hijo. Tratemos de ayudarlo.

5. Suponiendo que los caracteres de este sistema numérico son los del conjunto:

   $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

   a) Diseñar un algoritmo que permita transformar cualquier número en base 10 a números en base 16.

   b) Diseñar un algoritmo que permita la transformación inversa.

   c) ¿Tienen alguna limitación los algoritmos planteados o son de aplicación general?¿Cuáles son los requisitos para usar cualquiera de los algoritmos planteados? Plantear un algoritmo que verifique el cumplimiento de estos requisitos (esto se conoce como *chequeo de consistencia de los valores de entrada*)

   d) Escribir un código en Octave/Matlab que permita transformar un número ingresado por teclado de base 10 a 16.

   e) Ídem d), pero de base 16 a 10.

   f) Escriba un programa en Octave/Matlab que:

   - Permita transformar en ambos sentidos, dependiendo de una elección del usuario ingresada por teclado.
   - Tome el número a transformar, lo muestre en pantalla y muestre el número transformado.
   - Revise que las entradas de datos sean correctas y, si no lo son, termine el programa en forma prematura y devuelva un mensaje de error.

   g) Transforme el programa anterior en una *función* que tome dos números; el número a transformar y una *bandera*, es decir, otro número que sirva para indicar el sentido de la transformación. Sugerencia: convierta los dos primeros programas en funciones.

   h) Escriba un código que permita *validar* los resultados de la función anterior. Por ejemplo, probando valores conocidos.

6. Repetir el ejercicio anterior si el conjunto fuera 0, 1

El gemelo del nene en cuestión, en su correcto uso de su libertad, decidió que no quería usar las arañas para aprender a contar y, en su lugar, usó sus manos y pies aunque no en la manera tradicional (recordemos que se trata de una escuela hippie, en la que cada nene sigue su propio camino). Este nene utiliza *mano/pie arriba* y *mano/pie abajo* para contar, dejando a su padre aún más confundido. Tratemos de darle una mano.

7. ¿Cuál es el número entero más grande que puede contar el gemelo?¿Y el original?¿Y si el padre ayuda al gemelo con sus manos y pies a qué número pueden llegar?

Esta decisión del fundador de la escuela está lejos de ser casual, ya que les enseña a los nenes a manejarse en el sistema hexadecimal (nene original) y binario (gemelo). Estos sistemas de representación numérica son ampliamente usados en la industria del software, lo que lleva a pensar que el fundador de la escuela es un tecnócrata infiltrado en el movimiento hippie.

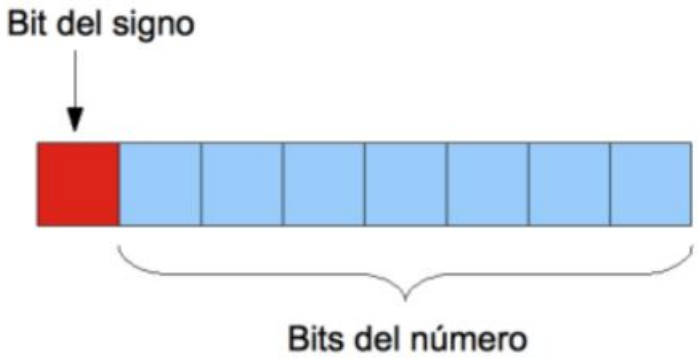
Formas de representación de números

-Mamá, mamá! En la escuela me dicen que soy como una computadora
-¿Por qué?¿Haces las cuentas muy rápido?
- Sí. Y no

Ese chiste (muy) malo solamente sirve para indicar que las computadoras, en su nivel más básico, únicamente entienden números binarios. Y después traducen esos números binarios a otros sistemas de representación, como el hexadecimal o el decimal.

Números enteros

Hasta ahora, tratamos con números naturales pero ¿Qué pasa cuando queremos representar números enteros? Respuesta: se utiliza un bit, es decir, un dígito binario, para representar el signo.



La figura muestra un *entero con signo* (signed integer, o signed int) de 8 bits, entre los que el primer bit se usa para el signo. En general, cuando el bit de signo tiene valor 0 el número es positivo, y cuando el bit de signo toma el valor 1 el número es negativo.

Supongamos el sistema de representación siguiente. Para el caso de los positivos, los números se construyen de la misma manera que en el ejercicio del gemelo binario: cada dígito representa una potencia de 2. Para los números negativos, el primer bit (el de la izquierda, el más significativo) sirve para invertir (hacer negativo) el número que se construye con los bits restantes. Entonces, cabe la siguiente pregunta:

8. ¿Cuál es el valor más grande que puede representarse con un número de

   a) 4 bits sin signo
   b) 8 bits sin signo
   c) 12 bits sin signo
   d) 16 bits sin signo
   e) Ídem a-d pero con signo?

9. ¿Y los menores valores (más negativos)?

10. ¿Qué pasa con el cero?

Si respondieron bien las preguntas anteriores, habrán notado que para cada entero con signo se obtiene un rango de -n a +n y dos valores para el cero: +0 y -0. Esto trae dos inconvenientes. El primero es la redundancia (del cero), y el otro es el rango: al repetir un símbolo para el cero, dejamos de lado un símbolo que podría usarse para extender el rango. Para solucionar esto (y, de

paso, hacer las cosas más fáciles para los procesadores que tiene que hacer cuentas con números binarios), se usa el siguiente sistema: cuando hay un 1 en el primer bit, *todos los otros bits se dan vuelta, se* ***suma 1*** *y se cambia el signo* (se suma el número 1, en lugar de hacer el número resultante más negativo en 1). Veamos qué haría nuestro gemelo binario al llegar a la primaria, para formar un número negativo con un *entero con signo de 4 bits*:

$$1001_2 = -(110_2 + 001_2) = -(6_{10} + 1_{10}) = -7_{10}$$

Los números en subíndice indican la base. Base 2, binario; base 10, decimal.
A este tipo de representación con signo se la conoce como de *complemento a dos*.

11. Con esta representación, ¿Cuál es el rango de los enteros con signo de 4, 8, 16, y 32 bits (expresados de "dos a la tanto a menos dos a la tanto")?

12. Escribir una función que tome como entrada a un número binario con signo de 8 bits y lo transforme en un número decimal, usando complemento a dos.

13. Escribir una función que tome como entrada un número decimal del rango de un entero con signo de 8 bits y lo transforme a un número entero binario de 8 bits con signo.

14. Usar la función del punto 13 para convertir a binario los números 71 y 63. Sumar los números binarios. Luego usar la función del punto 12 para convertir a decimal la suma. Analizar el resultado.

Representación binaria de números reales

Los números reales se representan, en la computadora, de manera diferente a los números enteros. En general, el número real binario se divide en tres partes: Signo, exponente y mantisa.

El signo, en general, se representa con el primer bit (tal como vimos antes para los números enteros). Si ese bit es 1, el número es negativo.

El exponente corresponde al segundo bloque de bits, y se maneja como un número entero. En el caso de un número real de 32 bits, son los 8 bits siguientes. De esos 8 bits, el primer corresponde al *signo del exponente*, y los otros 7 bits forman el *valor del exponente* (en *binario*).

La mantisa se forma con los restantes 23 bits, y son los que contiene el valor de los dígitos significativos del número. Pero este valor no se forma como en un número entero, sino con el mismo concepto que forman los números "con decimales" en base 10. Veamos:

$$6{,}4789_{10} = 6 \times 10^0 + 4 \times 10^{-1} + 7 \times 10^{-2} + 8 \times 10^{-3} + 9 \times 10^{-4}$$

En un número binario, se tendrá que:

$$1{,}001011_2 = 1 \times 2^0 + 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} + 1 \times 2^{-6}$$

1	1	0	0	1	0	1	1
-1	+1/2	+1/4	1/8	1/16	1/32	+1/64	1/128
-1	+0.5	+0	+0	+0.0625	+0.03125	+0.015625	+0.0078125

La representación decimal de este número es -0.3828125.

15. Plantear un algoritmo (y escribir la correspondiente función) que convierta un número racional (con decimales) de base 10 a base 2 (binario), con notación de complemento a 2.

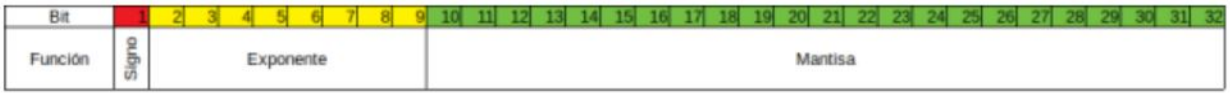
    a) Usar solamente 8 bits para la representación. ¿Cuál es el número más cercano a cero que se puede representar?

b) Usar 16 bits. ¿Cuál es el número más cercano a cero que se puede representar?

c) ¿Cuál es el número más cercano a cero que se puede representar con 23 bits?

Cuando se agrega la parte del *exponente*, se pueden representar números fuera del intervalo $[-1, 1]$. Tal como se mencionó antes, el exponente es binario. Esto significa que el número completo toma la forma

$$\text{número} \times 2^{\text{exponente}}$$



16. Investigar la estructura del número de coma flotante de 64 bits. ¿Cuál es el número más pequeño que se puede representar (el más cercano a cero)?¿Y el más grande?

17. En el lenguaje de la computadora, ¿Qué significa que algo sea "infinito" (*inf*)?¿Y qué significa que algo "no es un número (*Not a Number*, o *NaN*)?