Introducción a la Programación y Análisis Numérico

Año 2022

Práctica 2: Programación en OCTAVE/MATLAB

Como lenguaje de programación vamos a utilizar Matlab/Octave. En estos lenguajes existen dos maneras de trabajar: de forma directa, ingresando comandos por la línea de comandos, o bien generando un script de extensión **.m** que se ejecuta desde la consola escribiendo el nombre del archivo sin la extensión. Un script es un archivo de texto plano que contiene una serie de instrucciones que Matlab/Octave puede interpretar y ejecutar. En general trabajaremos con un programa principal, guardado en un archivo **nombre-programa.m**, y en algunos casos también con subprogramas que consistirán de funciones que serán llamadas por el programa principal.

Variables

Respecto de los nombres de las variables, Matlab/Octave distingue entre mayúsculas y minúsculas, es decir, la variable $'hola'$ es distinta de la variable $'HOLA'$, y a su vez de la variable $'Hola'$. Para asignar un valor a una variable se utiliza el operador igual (=). A una variable se le puede asignar un valor aislado o también el resultado de una operación. Las variables pueden ser enteras, reales, complejas, de tipo carácter o variables lógicas. Matlab/Octave cuenta con variables predefinidas, esto es, nombres reservados con valores ya establecidos. Por ejemplo la variable *pi* tiene asignado el valor π. Algunas variables predefinidas que veremos a menudos son ans, Inf, NaN.

Para Matlab/Octave todos los elementos son arreglos matriciales. En particular, los escalares son matrices de dimensión 1×1, los vectores fila son matrices de dimensión $1 \times$ n, y los vectores columna son matrices de dimensión n $\times 1$. Las matrices se declaran entre corchetes ([]), las columnas se separan con espacios o con coma (,), y para indicar el comienzo de una nueva fila se utiliza el punto y coma (;).

Hay varias funciones que permiten construir matrices con valores particulares, como por ejemplo:

función	operación
`eye(n)`	matriz identidad de dimensión n
`zeros(n)`	matriz con elementos nulos cuadrada de dimensión n
`zeros(n,m)`	matriz con elementos nulos de dimensión n $\times$ m
`ones(n)`	matriz de unos cuadrada de dimensión n
`ones(n,m)`	matriz de unos de dimensión n $\times$ m
`diag(v, m)`	matriz cuadrada con los elementos del vector v en la diagonal m

Muchas veces vamos a necesitar construir vectores fila que vayan de un valor inicial a un valor final con un paso constante. Dos formas simples de construir estos vectores son:

- especificando entre paréntesis o corchetes el valor inicial, el paso y el valor final, donde si el paso no es especificado por defecto toma el valor 1: `x=[valor inicial:paso:valor final]`;
- a partir de la función `linspace: x=linspace(valor inicial,valor final,cantidad de puntos)`.

La primera especifica el paso mientras que la segunda la cantidad de puntos a considerar dentro del intervalo. Como veremos en un ejemplo más adelante, ambas maneras no son necesariamente equivalentes y dependiendo de la situación convendrá utilizar una u otra.

A los elementos de una matriz se puede acceder mediante sus índices. A su vez, también pueden ser asignados a una nueva variable, generando un subarreglo del arreglo original. Octave/Matlab considera como primer índice al valor 1, mientras que el último valor estará dado por la dimesión, el cual también se puede obtener mediante la palabra `end`. Para acceder a un elemento particular de un vector se indica entre parentesís el índice correspondiente a dicho elemento. Luego, por ejemplo para

un arreglo de dimensión 1×n, v(i) permite obtener el elemento de la columna i, donde 1 ≤ i ≤`end`. Para un subarreglo de mayor dimensión, se especifica el rango a extraer separado por dos puntos (**:**), es decir, v(**inicio** : **fin**), donde 1 ≤ **inicio** ≤ **fin** y **inicio**≤ **fin** ≤`end`. De manera análoga se procede con un vector columna o con una matriz, donde en este último caso se necesita especificar dos índices (o dos rangos de índices) separados por coma, el primero para la(s) fila(s) y el segundo para la(s) columna(s).

Finalmente, si bien hay todo un universo de funciones definidas para arreglos, dos que valen la pena presentar dado que serán muy utilizadas en este curso son `length()`, que devuelve la longitud (número de elementos) de un objeto, y `size()`, que devuelve un vector fila con el tamaño (número de elementos) de cada dimensión del arreglo.

Algunos ejemplos para probar desde la consola:

```
>> x0 = 1
x0 = 1
>> x1 = [1, 2]
x1 =
   1 2
>> x2 = [1 2]
x2 =
   1 2
length(x2)
ans = 2
>> x3 = [1; 2]
x3 =
   1
   2
>> x3(1)
ans = 1
>> x3(2)
ans = 2
>> x3(end)
ans = 2
>> a = [1, 2, 3; 4, 5, 6; 7, 8, 9]
a =
   1 2 3
   4 5 6
   7 8 9
>> a(2, 1)
ans = 4
>> a(3, 3)
ans = 9
>> a(end)
ans = 9
>> a(end, end)
ans = 9
>> length(a)
ans = 3
>> size(a)
ans = 3 3
>> aa = a(1:2,1)
aa =
   1
   4
>> b = zeros(2, 3)
b =
   0 0 0
   0 0 0
>> c = 3*ones(1,4)
c =
   3 3 3 3
>> size(c)
ans =
   1 4
>> d = [1:2:10]
d =
   1 3 5 7 9
>> length(d)
ans = 5
>> e = linspace(1,10,5)
e =
   1.0000 3.2500 5.5000 7.7500 10.0000
>> length(e)
ans = 5
```

A partir de la comparación de los vectores **d** y **e** podemos notar que no es exactamente lo mismo indicar el paso con el que se construye un vector que la cantidad de puntos. Si bien para este ejemplo la longitud de los vectores es la misma, podemos ver que en el primer caso Octave/Matlab incrementa el valor inicial con el paso indicado hasta llegar al valor final pero sin la necesidad de considerarlo, mientras que en el segundo caso, el paso es tal que permite respetar ir desde el valor inicial al final considerando la cantidad de puntos indicada. Si quisiéramos construir un vector indicando el paso y que tome el valor final, el paso no es necesariamente arbitrario, sino que se puede calcular a partir de la

cantidad de puntos que se quiere obtener mediante **paso = (valor final - valor inicial)/(cantidad de puntos -1)**. De esta manera obtenemos el mismo resultado que con la función `linspace`. Para chequearlo, calcule nuevamente **d** pero con paso = (10 - 1)/(5 - 1) = 2.25

Como en Octave/Matlab todas las variables son arreglos, los mismos pueden ser utilizados para realizar operaciones vectoriales y matriciales o para realizar operaciones *elemento a elemento*. Para distinguir una operación matricial de una elemento a elemento es necesario anteponer un '.' antes de los operadores sobrecargados `*, ^, /,` y `\`. Algunas de las operaciones que más vamos a utilizar son:

símbolo	operación
`+`	suma
`-`	resta
`*`	producto
`.*`	producto elemento a elemento
`^`	exponenciación
`.^`	exponenciación elemento a elemento
`/`	división
`./`	división elemento a elemento
`\`	división invertida
`.\`	división invertida elemento a elemento
`'`	operador transpuesta

Más ejemplos para probar desde la consola, esta vez de operaciones matriciales y elemento a elemento. Trate de reproducirlos en la ventana de comandos, de entender qué operación se está realizando y comprender el resultado obtenido.

```
>> a = [1 1 1]
a =
   1 1 1

>> b = a*a
error: operator *: nonconformant arguments
      (op1 is 1x3, op2 is 1x3)

>>b = a*a'
b = 3

>> c = a.*a
c =
   1 1 1

d = a.*a'
d =
   1 1 1
   1 1 1
   1 1 1
e = (2*a)^2
error: for x^y, only square matrix arguments are permitted
and one argument must be scalar. Use .^ for elementwise power.
```

```
e = (2*diag(a))^2
e =
Diagonal Matrix
   4 0 0
   0 4 0
   0 0 4

>> e = diag([1, 1], 1) + diag([3, 3], -1) + e
e =
   4 1 0
   3 4 1
   0 3 4

>> f = e \ a'
   0.25000
   0.00000
   0.25000

>> g = inv(e)*a'
   2.5000e-01
   -2.7756e-17
   2.5000e-01
```

En principio en **f** y en **g** se está realizando la misma operación. Sin embargo vemos que los valores obtenidos y su forma de expresarlos no son exactamente iguales. Esto se debe a que el operador \ evita el cálculo directo de la inversa de la matriz **e**. Si la matriz de coeficientes es singular, Octave/-Matlab emite un mensaje de advertencia (*warning: matrix singular to machine precision*) y calcula una solución en el sentido de norma mínima.

Operadores relacionales y lógicos

Los operadores relacionales son utilizados para la comparación de variables y su implementación devuelve una variable lógica *True* (que equivale al valor numérico 1) o *False* (que equivale al valor numérico 0).

símbolo	operador
<	menor
<=	menor o igual
>	mayor
>=	mayor igual
==	igual a (equivalencia)
!=; ~=	distinto a

Los operadores lógicos por su parte operan entre dos variables lógicas y su resultado es nuevamente una variable lógica:

símbolo	operador
|	OR: el resultado será cierto si es cierto el valor de al menos una de las dos variables.
||	OR con cortocircuito. Igual a OR pero evalúa las condiciones mientras sean verdaderas.
&	AND: el resultado de la operación es cierto si son ciertas las dos variables.
&&	AND con cortocircuito. Igual a AND pero evalúa las condiciones mientras sean verdaderas.
xor()	OR EXCLUSIVO: el resultado sólo es verdadero si lo es una de las dos variables.
!; ~	NOT: opera sobre una sola variable lógica, da como resultado el valor contrario al de la variable.

Los operadores con cortocircuito sólo operan sobre escalares, los demás operadores pueden usarse sobre escalares, vectores y matrices, siempre que tengan la misma dimensión. El resultado será un arreglo lógico (arreglo con valores unos (*true*) y ceros *false*). Más ejemplos:

```
>> x1 = 0;     %variable escalar
>> x2 = 10;    %variable escalar
>> x1 > x2
ans = 0     %variable lógica
>> ∼x1
ans = 1     %variable lógica
>> ∼x2
ans = 0     %variable lógica
>> x1 == x2
ans = 0     %variable lógica
>> a = [5 5 0];     %variable vectorial
>> b = [0 5 5];     %variable vectorial
>> a & b
 ans = 0 1 0 %arreglo lógico de comp. elemento a elemento
>> a | b
 ans = 1 1 1 %arreglo lógico de comp. elemento a elemento
>> v = true  % definición de variable lógica verdadero
 v = 1     % notar que le asigna el valor 1
>> f = false  % definición de variable lógica falso
 f = 0     % notar que le asigna el valor 0
>> v > f  % Cuidado: no distingue entre las variables
 ans 1     % lógicas y las variables enteras 1 y 0
 >> vs = 'true'  % def. variable caracter con valor true
     vs = true
    >> fs = 'false'  % def. variable caracter con valor false
       fs = false
    >> vs & fs
error: nonconformant arguments (op1 is 1x4, op2 is 1x5)
Intenta comparar elementos de 2 variables con dif. longitud
    pera = 'pera'; kiwi = 'kiwi'; banana = "";
    >> pera & kiwi
       ans = 1 1 1 1
    >> pera != kiwi
       ans = 1 1 1 1
    >> pera && kiwi
       ans = 1
    >> pera && banana  % Una variable caracter es falsa si
       ans = 0      % está vacía como la variable banana
    >> x = 2;
    >> 1 < x <3  % comparación de izquierda a derecha
       ans = 1
    >> x = 5;
    >> 1 < x <3  % como 1<5 = 1 esta sentencia
       ans = 1              % resulta verdadera
    >> 1 < x & x < 3  % forma correcta de realizar
       ans = 0              % una doble comparación
    >> (1 < x & x) < 3  % los paréntesis pueden alterar
       ans = 1              % el orden de las operaciones
```

En los últimos ejemplos vemos que cuando consideramos más de una operación relacional a la vez, las mismas son realizadas de izquierda a derecha. Al mismo tiempo, Octave/Matlab primero realiza las comparaciones relacionales y luego las lógicas. Es por esto que al considerar los paréntesis en el último ejemplo cambió el resultado respecto del ejemplo anterior. Por último, en estos ejemplos en algunas líneas luego de definir el valor de las variables hicimos uso del **;**. Usar **;** al finalizar la línea evita que sea mostrada en pantalla, algo que suele ser muy útil, sobre todo cuando escribamos código en archivos. Cuando queremos que una variable sea impresa en pantalla utilizamos el comando `disp()` o el comando `fprintf()`. Este último necesita que se especifique el formato del tipo de dato a imprimir.

Estructuras de selección y de iteración

Estructura de selección: permite que conjuntos de instrucciones alternativas puedan ejecutarse según se cumpla (o no) una determinada condición. Para ello utilizamos la sentencias ***if, elseif, else***:

Estructura:

```
if (condición)
    sentencias
elseif (condición)
    sentencias
⋮
elseif (condición)
    sentencias
else
    sentencias
end
```

Ejemplo:

```
>> a = true
a = 1
>> if(a)
disp('Hola')
elseif(a)
disp('Chau')
end
Hola
```

Notar que definir a **a** como true es equivalente a asignarle el valor 1. ¿Por qué muestra el valor 'Hola' y no el valor 'Chau'?

donde puede aparecer cualquier número de ***elseif*** o incluso ninguno. De manera secuencial se van probando cada una de las condiciones y cuando una es verdadera, se ejecutan las sentencias dentro de ella. Si ninguna de las condiciones es verdadera y la cláusula ***else*** está presente, se ejecutan las sentencias dentro del ***else***. Sólo puede aparecer una cláusula ***else***, y debe ser la última declaración. La condición en una sentencia ***if*** se considera verdadera si su resultado es no vacío y contiene sólo elementos no nulos, reales o lógicos. De lo contrario, la expresión será falsa. Una expresión lógica puede formarse comparando los valores de expresiones aritméticas utilizando operadores relacionales y pueden combinarse usando operadores lógicos.

Estructuras de iteración: una estructura de control iterativa permite la repetición de una serie determinada de instrucciones. Este conjunto de instrucciones a repetir se denomina bucle (*loop* en inglés) y cada repetición del mismo se denomina iteración. Para bucles donde el número de iteraciones es fijo y conocido de antemano usamos la sentencia ***for***:

Estructura:

```
for var = valor inicial:incremento:valor final
    sentencias
    ⋮
    sentencias
end
```

Ejemplo:

```
>> x = [-4, -3, -2, -1, 0, 1, 2, 3, 4];
>> for i = 1:2:length(x)
disp(x(i))
end
 -4, 2, 0, 2, 4
```

En este ejemplo se muestran los elementos del vector **x** con índice impar.

donde la variable **var** es una variable entera que puede ser o bien un vector o definirse como: **var = valor inicial:incremento:valor final**, donde **var** tomará los valores entre **valor inicial** y **valor final** con paso **incremento**. Cuando **paso** es igual a 1 no necesita ser especificado. Al ser **var** una variable entera puede tomar valores negativos, al igual que el **paso** puede ser ascendente o descendente.

Por otra parte, para bucles donde el número de iteraciones es desconocido de antemano usamos la

sentencia ***while***. En este caso el bucle se repite mientras se cumple una determinada condición (bucles condicionales):

Estructura:

```
while(condición)
    sentencias
    ⋮
    sentencias
end
```

A continuación veremos un ejemplo que involucra un contador:

```
>> a =  1;
>> while(a<=3)
disp(a)
a = a + 1; % que equivale a a++
end
 3,  2,  1
```

Notar que en el ejemplo elegido, **a** debe ser modificado (incrementado en este caso) para que el lazo pueda terminar en algún momento.

Lo primero que hace la declaración ***while*** es probar la condición. Si la condición es verdadera, ejecuta las sentencias y vuelve a probar la condición, si sigue siendo cierta, las sentencias se ejecutan de nuevo. Este proceso se repite hasta que la condición deja de ser cierta. Si la condición es inicialmente falsa, el cuerpo del bucle nunca se ejecuta.

Funciones

Matlab/Octave tiene definido funciones matemáticas tales como: `cos(x)`, `sin(x)`, `tan(x)`, `abs(x)`, `exp(x)`, `sqrt(x)`, `log(x)`, `log2(x)`, `log10(x)`, `rem(x)`, `sign(x)`, etc., donde la operación realizada por cada función no siempre resulta evidente, y si quiere una explicación sobre la operación que realiza o sobre cómo utilizarla puede escribir el comando `help` seguido del nombre de la función en la ventana de comandos. Algo importante a tener en cuenta es que las funciones trigonométricas operan sobre radianes y no sobre grados. Matlab/Octave también posee una gran librería de funciones predefinidas que son de gran utilidad y que algunas iremos mencionando durante el curso. Pero puede ocurrir que necesitamos definir funciones propias, éstas pueden ser definidas de varias formas y en particular vamos a ver dos, funciones anónimas y funciones de usuario definidas como un subprograma. Las funciones anónimas son una forma simple de definir funciones de línea. La forma general es:

```
nombre_de_la_función =@(argumentos)expresión
```

donde **nombre_de_la_función** es el nombre con el que llamamos a la función, **argumentos** son las variables de la función y **expresión** es la función en sí misma. Este tipo de funciones son útiles cuando se quiere realizar una misma operación repetidas veces. Para hacer uso de las funciones anónimas, se invoca a la función mediante su nombre especificando entre paréntesis los argumentos previamente definidos. Por otro lado, para definir una función de usuario en un archivo que será llamado por un programa principal, la misma debe ser especificada como sigue:

```
function[argumentos de salida] = nombre (argumentos de entrada)
    declaraciones de variables locales
    sentencias
end
```

donde **argumentos de entrada** son los valores que serán pasados desde el programa principal a la función para que pueda realizar las operaciones y devolver el resultado de la evaluación de la función en la variable **nombre**, así como otros resultados que puedan generarse y serán devueltos en los

argumentos de salida. Algo muy importante a tener en cuenta es que el nombre de la función debe coincidir con el nombre del **archivo.m** en el que es guardada. Las funciones son llamadas e invocadas en el programa principal mediante:

[argumentos de salida] =nombre (argumentos de entrada)

Gráficas

Matlab/Octave cuenta con una gran variedad de funciones destinadas a realizar gráficos. Entre ellas encontramos `plot()`, `stem(), bar(), polar(), semilogx()` y muchas otras. La función `plot()` es la que más utilizaremos a lo largo de este curso. Para utilizarla en general necesitamos especificar entre paréntesis dos variables separadas por coma, que representen las coordenadas cartesianas x e y. Por defecto, para graficar, el comando `plot()` utiliza líneas. Pero muchas veces es conveniente graficar utilizando símbolos ('*', 'o', '+', etc.) y para ello se especifica el mismo como un tercer argumento entre comillas. Otro parámetro que puede ser especificado es el color ('k', 'r', 'g', 'b', etc.). Para mayor detalle escriba en la ventana de comandos `help plot`

Si se desea graficar distintas curvas en una misma figura es necesario indicarle al programa que una nueva invocación a `plot()` no debe "borrar"los gráficos realizados anteriormente. Esto se logra con el commando `hold on`. De forma similar, si es necesario volver a indicarle al programa que sólo debe graficar la última invocación de la función `plot()`, se debe usar el comando `hold off`. Por defecto, el commando `hold` se encuentra en el estado off. Existen comandos que se encargan de agregar etiquetas. Algunos importantes son: `title(), xlabel(), ylabel` y `legend()`. Existen otras funciones para manipular propiedades de los gráficos como `axes(), axis(), grid` y otras. Para conocer cómo funcionan, se recomienda consultar el comando `help`. Para guardar una figura en un archivo utilizamos el comando `print`. Por ejemplo, `print figure1.pdf` guarda la figura en formato **pdf** en el archivo **figure1.pdf**, mientras que `print -djpg figure1` guarda la figura con formato **jpg** en el archivo **figure1.jpg**. Para mayor detalle, `help print`. Como último detalle, mediante el comando `figure()` pueden generarse distintos gráficos, es decir, nuevas ventanas en donde graficar las distintas variables. Como argumento (opcional) esta función toma el número de identificación de cada ventana. Si ningún número es especificado, por defecto se toma el siguiente valor disponible.

Entrada y salida de datos por archivo

La asignación de variables también puede realizarse mediante lectura de un archivo externo donde están guardados los valores. Para la lectura de datos en archivos existen varias funciones predefinidas (o comandos), en particular acá introducimos dos:

Ejemplo 1: datos sin encabezado:

```
A = load('nombre_archivo.txt');
xop1 = A(:,1);
yop1 = A(:,2);
```

Ejemplo 2: datos con encabezado:

```
fid = fopen('nombre_archivo.txt', 'r');
data = textscan(fid,'%f','HeaderLines',n);
fclose(fid);
xop2 = data{1};
yop2 = data{2};
```

Donde **nombre_archivo.txt** es el archivo en en que están guardados los valores de las variables. En el ejemplo 2 se debe aclarar el formato con el que se debe leer el dato. Por ejemplo, %d es el formato para datos enteros, %f para reales en notación decimal, %e para notación exponencial, %A

para carácter. Por otro lado, la cantidad de líneas que conforman el encabezado, n, debe ser un número entero mayor que cero.

Para guardar datos en un archivo hay varias formas, pero la más simple es mediante el uso del comando `save`, seguido del cual escribimos el nombre del archivo en el que vamos a guardar los datos y las variables a guardar. Por ejemplo, sean las variables `v1` y `v2`, para guardarlas escribimos: `save 'nombre-archivo-salida.dat' v1 v2`. De esta manera guarda las variables `v1` y `v2` cada una con su encabezado. Si queremos que no escriba encabezado y que las variables sean escritas una debajo de la otra, agregamos `-ascii`, es decir: `save -ascii 'nombre-archivo-salida.dat' v1 v2`. Otro comando muy útil es `csvwrite`. Con este comando escribimos los datos de salida separados por coma, pero los mismos antes tienen que haber sido acomodados en un sólo arreglo, por ejemplo `V`. Una vez hecho ésto, simplemente se ejecuta `csvwite('nombre-archivo-salida.dat', V)`.

Algunos detalles útiles: Una buena práctica es comentar en la primera línea el nombre del programa con una breve descripción del problema a resolver, así como también documentar las operaciones que se realizan en el mismo. Para ello se utiliza el símbolo %. Todo lo que escribamos detrás de él será interpretado como un comentario y no será ejecutado. También es buena práctica utilizar los comandos `clc; clear all; close all` al comienzo del programa. Con ellos limpiamos el espacio de trabajo, las variables y cerramos las figuras abiertas, respectivamente. Matlab/Octave siempre muestra el resultado de la operación realizada, lo que puede llegar a ser molesto. Como fuimos viendo en algunos de los ejemplos, para decirle que no queremos visualizar el resultado en la pantalla es necesario agregar un ';' al final de la línea ejecutada, tanto en la ventana de comandos como el editor de trabajo. Otra buena práctica es inicializar las variables con la finalidad de evitar inconvenientes. Las variables en general son mostradas con formato `format short`. Si queremos visualizar un mayor número de decimales, se consigue especificando `format long`. Si generamos un bucle infinito (un bucle que nunca termina por sí solo), se puede detener su ejecución pulsando Ctrl+c. Cuando tenga dudas sobre algo, puede utilizar el comando help o la internet, que puede proporcionar buenas respuestas siempre que haga preguntas claras!

Ejercitación

Ej. 1: Variables. A continuación se presentan sentencias de programación. Ejecutarlas en Octave/Matlab de manera secuencial, analizar el resultado y determinar si la variable es escalar, vectorial o matricial y el tipo de variable (real, compleja, lógica o caracter (string)).

sentencia	operación y resultado	tipo de variable
a = -1		
v = [a; -a]		
A = [v v]		
A(1, 2)		
A*v		
A.*v		
inv(A)		
v.*v		
v*v		
v*v'		
v'*v		
dot(v,v)		
raiz_a = sqrt(a)		
isreal(a)		
isreal(raiz_a)		
c = 'Hola'; d = 'mundo'		
[c, ' ', d]		
t = true; f = false		
3 < 2		
isreal(3 < 2)		

Ej. 2: Estructura de repetición For. Calcular la suma y el producto de los primeros 50 números enteros positivos.

Ej. 3: Estructura de repetición While. Genere un vector que guarde los números $a \in \mathbb{Z}^+$ tales que $a < 2000$ y que a sea múltiplo de 2, 7 y 13. Ayuda: Puede serle útil el comando `mod()`.

Ej. 4: Estructura de selección. Mediante el análisis del discriminante del polinomio cuadrático ($ax^2 + bx + c$) realice un programa que escriba en pantalla si las raíces serán reales e iguales, reales y distintas o complejas conjugadas. Para probarlo utilice como valores de los coeficientes del polinomio a = [0.1, 0.25, 1], b = 1 y c = 1.

Ej. 5: Funciones.

a) Buscar en la ayuda de Octave/Matlab ejemplos de funciones, para ello utilice `help function`, `help function_handle`.

b) Defina la siguiente función a trazos haciendo uso del comando ***if*** y de los operadores lógicos y relacionales. Implementarla como una función de usuario.

$$f(x) = \begin{cases} 2 & \text{si} \quad x < -2 \\ x^2 & \text{si} \quad -2 \leq x < 3 \\ \frac{1}{x} & \text{si} \quad 3 \leq x \leq 10 \\ x & \text{si} \quad 15 < x < 20 \\ 3 - x & \text{si} \quad x = 22 \\ 0 & \text{en} \quad \text{cc} \end{cases}$$

Evalúe $f(x)$ en el vector x= [-30:0.5:30] y grafique el resultado haciendo uso del comando `plot()`.

c) Modifique el ejercicio 4 para que calcule las raíces del polinomio cuadrático. Implementarlo como una función de usuario donde los argumentos de entrada sean los coeficientes del polinomio y los argumentos de salida sean las raíces calculadas. Calcule las raíces de:

i) $0.0x^2 + x + 1$,
ii) $0.1x^2 + x + 1$,
iii) $0.25x^2 + x + 1$,
iv) $x^2 + x + 1$.

¿Qué sucede en el item i)? Modifique la función para que le avise al programa principal si los coeficientes pasados como argumento de entrada no corresponden a un polinomio cuadrático.

d) Dentro de las funciones predefinidas en Matlab/Octave hay dos que serán de gran utilidad en las próximas prácticas, ellas son `polyfit()` y `spline()`. Busque y explique qué hace cada una de las funciones. Seguramente habrá encontrado que las mismas pueden ser utilizadas con dos o tres argumentos, explique en cada caso que es lo que devuelve la función. Si considera su implementación con dos argumentos, ¿qué otras funciones predefinidas debe utilizar para obtener el mismo resultado que con tres argumentos de entrada?

Ej. 6: Graficando funciones. Ilusión óptica.
Defina los vectores `t`, `x` e `y` de la siguiente manera:

```
t=[0:2*pi/50:2*pi];
x=2*cos(t);
y=5*sin(t);
```

a) Grafique `x` e `y` en función `t`. Los comandos `plot()`, `figure()` y `subplot()` pueden serle útiles.

b) La curva $\mathbf{r}(\mathbf{t}) = (\mathrm{x(t)}, \mathrm{y(t)})$ representa una elipse descripta en forma paramétrica. Para graficarla puede utilizar `plot(x,y)`.

c) ¿Cuál parece ser el eje mayor de la elipse? ¿Cuál es realmente el eje mayor de la elipse?¿A qué se debe ésto? La sentencia `axis equal` puede ser de utilidad.

d) Grafique la siguiente curva descripta en forma paramétrica:

$$\begin{aligned} x &= \cos(t)\left(e^{\cos(t)} - 2\cos(4t) - sin^5\left(\frac{t}{12}\right)\right) \\ y &= sen(t)\left(e^{\cos(t)} - 2\cos(4t) - sin^5\left(\frac{t}{12}\right)\right) \end{aligned}$$

Vea que sucede al modificar el paso de la variable t (puede comenzar con el mismo t de los incisos anteriores). También pruebe extendiendo el límite superior de la variable t, por ejemplo, a 4π, 8π, etc... Resulta interesante ver cómo se va formando la curva a medida que varía el parámetro t. Para ello podemos intentar una especie de "animación" en Octave/Matlab con los comandos siguientes:

```
figure, axis([-3 3 -4 4]), hold on
for i=1:length(t)-1
plot(x(i:i+1),y(i:i+1));
pause(1/10);
end
hold off
```

Uniendo errores con programación.

Ej. 7. La función coseno puede evaluarse por medio de la serie infinita:

$$cos(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{(2n)}}{(2n)!}$$

Calcular la aproximación a $x = \pi/3$ con 8 cifras significativas. Guardar en un vector las sumas parciales y determinar la cantidad de términos a considerar para obtener la aproximación solicitada. Escribir la aproximación en función de la tolerancia considerada. Ayuda: para el cálculo del factorial utilice la función `factorial()` y para la solución exacta utilice `cos(pi/3)`.

Ej. 8. Verifique que las funciones:

$$f_1(x) = \sqrt{x}(\sqrt{x+1} - \sqrt{x}) \quad \text{y} \quad f_2(x) = \frac{\sqrt{x}}{\sqrt{x+1} + \sqrt{x}},$$

resultan analíticamente equivalentes.

a) Calcule utilizando Octave/Matlab la evaluación las funciones en $x = 10^{15}$. Para ello defina $f_1(x)$ y $f_2(x)$ como funciones anónimas.

b) ¿Qué observa entre los valores obtenidos? Explique a qué se deben los resultados anteriores.

c) Agregue a su programa las siguientes líneas de código y haga un breve análisis de los resultados.

```
x = 1; % inicializo la variable x en uno
format long e % formato exponencial
for k = 1:15
    fprintf('En x=%2.1e, f1(x)=%20.18f, f2(x)=%20.18f \n',x,f1(x),f2(x));
    x = 10*x;  % incremento de la variable x en un factor de 10.
end
disp('Operaciones para analizar el resultado de la evaluacion')
disp( 'de las funciones f1 y f2 en x = 1.0e+15:')
```

```
%En particular para x = 1.0e+E15 calculo:
sx = sqrt(x); %la raiz
sx1 = sqrt(x+1); %la raiz en x + 1
d = sx1 - sx; %la diferencia de las raices previas
s = sx1 + sx; %la suma de las raices previas

fprintf('sqrt(x+1) = %25.13f, sqrt(x) = %25.13f \n ',sx1,sx);
fprintf(' diff = %25.23f, sum = %25.23f \n',d,s);
```

Ej. 9. Considere las siguientes fórmulas equivalentes:

$$f_1(x) = \frac{1 - \cos x}{x^2}$$

y

$$f_2(x) = \frac{\sin^2 x}{x^2(1 + \cos x).}$$

Implementar $f_1(x)$ y $f_2(x)$ como funciones anónimas. Agregar las siguientes líneas de código donde ambas expresiones son evaluadas:

```
for k = 0:1 %k = 0: analiza aproximandose a cero, k = 1 aproximandose a pi.
    x = k*pi;
    tmp = 1;
    for k1 = 1:8
        tmp = tmp*0.1;
        x1 = x + tmp;
        fprintf('En x = %10.8f, \n', x1)
        fprintf('f1(x) = %18.12e; f2(x) = %18.12e \n', f1(x1),f2(x1));
    end
end
```

- Haga un análisis de los resultados obtenidos.
- ¿Qué sucede con ambas funciones para valores de $x \to 0$ y $x \to \pi$?