Universidad Nacional de San Juan

Facultad de Ingeniería

MÉTODOS NUMÉRICOS

Ingeniería CIVIL

Equipo de Cátedra

Profesor Titular	Beatriz Morales
Profesor Asociada	Agustina Garcés
Profesora Adjunta	Marion Castro
Jefe de Trabajos Prácticos	Pablo Marcuzzi

AÑO 2020

TUTORIAL DE OCTAVE

OCTAVE no puede considerarse como un lenguaje de programación, como Fortran o C, aunque sería difícil describirlo en algunas palabras, he aquí algunas de las características principales para los análisis numéricos.

La programación es mucho más sencilla

Hay continuidad entre valores enteros, reales y complejos

La amplitud de intervalo y la exactitud de los números son mayores

Cuenta con una biblioteca matemática amplia

Abundantes herramientas gráficas

Capacidad de vincularse con los lenguajes de programación tradicionales

OCTAVE no puede reemplazar a Fortran ni a C , ya que éstos siguen siendo importantes para la computación de alto rendimiento que requiere abundante memoria o un tiempo de cómputo largo. La velocidad con Octave es significativamente más baja que con Fortran o C porque Octave paga un precio elevado por sus características agradables. Además no hace falta conocer Fortran o C para entender Octave.

Este tutorial intenta dar una pequeña introducción en el lenguaje para que al desarrollar los temas propios de la materia no tengan dificultad con el lenguaje.
GNU Octave es un programa creado para trabajar con matrices, por lo tanto, este punto esprobablemente el más importante y en el que mejor tenemos que aclararnos para empezar a trabajar. Tenemos muchas opciones para trabajar con ellas, podemos intercambiar matrices, permutarlas, invertirlas; GNU Octave es una herramienta de cálculo muy potente en lo que a matrices se refiere

El símbolo >> es el promt de OCTAVE siempre precede a los comandos

Comandos Generales

Ayuda !! Si no entiende bien el significado de un comando, teclee

>> ***help nombre-comando***

Ejemplo:

>>help plot (enter)

```
Explica como graficar funciones
```

Quién ? El comando **who** produce una lista de las variables del espacio de trabajo actual; exhibe información adicional acerca de cada variable

Diario El comando **diary** escribe todo lo que se introduce por teclado , y aparece en pantalla lo envía a un archivo llamado *nombre*

>> ***diary nombre de archivo on*** (enter)

Y cuando se quiere dejar de copiar en el archivo, se escribe

>> **diary off** (enter)

Si ya existe el archivo *diary* las salidas de la pantalla se anexarán a ese archivo. El archivo es un archivo de texto que puede abrirse con cualquier procesador de texto, se puede editar e imprimir.

Operaciones Básicas

- **Cómo iniciar los cálculos**

Como ejemplo evaluemos : area= πr^2

```
>> r =5; (enter)
>> area=pi*r^2  (enter)
  area =
   78.5398
```

Debemos asignar valores a las variables antes de realizar los cálculos, debido a que OCTAVE trabaja en forma numérica. Además el programa tiene almacenadas algunas constantes como en este caso el número π con el nombre **pi**

Observación:

Cada línea es un comando , si termina con un signo de punto y coma el valor de la variable queda almacenado en memoria pero no se imprime en pantalla. La ausencia del punto y coma, indica que el valor del resultado o variable será presentado en la pantalla, lo cual en la mayoría de los casos es incómodo o poco práctico

La forma más sencilla de exhibir el resultado es teclear el nombre de la variable y pulsar enter La computadora exhibirá:

```
>>r = (enter)
      5
```

- **Operadores Aritméticos**

  Los operadores aritméticos como **+** , **-** , ***** y **/** son los mismos que los de cualquier lenguaje de programación , OCATVE emplea un operador no tradicional \ que puede llamarse *división inversa* o sea a\b produce b/a Por ejemplo

```
>> c=3\1  (enter)
c =
    0.3333
```

- **Variables y nombres de variables**

  No es necesario declarar los nombres de las variables ni sus tipos. Esto se debe a que en OCTAVE no hay diferencia en los nombres para las variables enteras, reales y complejas. Cualquier variable puede adoptar valores reales, complejos y enteros. Además no es necesario declarar previamente el tamaño de un arreglo

  En principio cualquier nombre puede usarse siempre que sea compatible con Octave. Se debe tener presente dos situaciones incompatibles:

  1. puede suceder que Octave no acepte el nombre
  2. el nombre es aceptado pero anula el significado original de un nombre reservado

  Para evitar problemas conviene utilizar los símbolos tradicionales i j k l m n como nombres de variables enteras En la siguiente tabla se presentan ejemplos de nombres variables reservados que tienen significado especial. Se puede verificar la existencia de una variable o un archivo con el comando **exist**

  Octave es sensible al cambio de mayúsculas y minúsculas

  A continuación una pequeña tabla con algunas variables ya existentes

Nombre de variable	Significado	Valor
eps	Epsilon de la máquina	2.2204e-16
pi	π	3.14159...
i y j	Unidad imaginaria	√-1
inf	infinito	∞
NaN	No es un número	
date	fecha	
flops	Contador de operaciones de punto flotante	
nargin	Número de argumentos de entrada de una función	
nargout	Número de argumentos de salida de una función	

Formato

Por omisión los números de exhiben con cinco dígitos:

```
>>pi  (enter)
ans =
    3.1416
```

Sin embargo, los mismos pueden exhibirse con 16 dígitos si se emite la orden **format long;** por ejemplo:

```
>> format long  (enter)
>> pi  (enter)
ans =
   3.14159265358979
```

Si desea volver al formato corto, utilice **format short** Se recomienda leer el help del comando **format** para ver otras opciones

Cómo borrar variables ?

Al ejecutarse los comandos, Octave guarda las variables utilizadas en una memoria temporaria. Sus valores permanecen en la memoria hasta que se sale de Octave o hasta que se borran las variables, lo cual se hace con el comando **clear**. Si sólo se desea borrar algunas variables, sus nombres se indican después de la palabra clear por ejemplo:

```
>>clear x y z  (enter)
```

Lectura y escritura

Hay varias formas de pasar datos a y de Octave, los métodos pueden agruparse en tres clases:

a) operación interactiva mediante teclado o el ratón
b) lectura o escritura en un archivo de datos
c) empleo de **save** y **load**

a) Puede aceptar datos de entrada a través del teclado mediante el comando **input** Si se desea leer un número, un enunciado básico sería

```
>> z=input('ingrese el valor del radio ')  (enter)
```

La parte *ingrese el valor del radio* es un mensaje de solicitud que se exhibe en la pantalla de la siguiente forma:

ingrese el valor del radio

cuando se ingresa el valor (por ejemplo 10) y se pulsa la tecla enter este valor se guardará en la variable

```
z =
     10
```

También es posible introducir cadenas desde el teclado . Ejemplo:

>> z=input('Indique su nombre(encerrado en apóstrofos):') (enter)

aparece en pantalla el mensaje

```
Indique su nombre  (encerrado en apóstrofos):
```

y al escribir el nombre entre literales (por ejemplo 'Cecilia') la variable z será

z =

Cecilia

b) Para guardar datos se utiliza el comando **save**

Si se utiliza solamente

>> **sabe datos** (enter)

todas las variables se guardarán en el archivo con el nombre datos

La orden

>> **load datos** (enter)

es el inverso de save y recupera todas las variables guardadas

Si sólo desea guardar ciertas variables, escriba sus nombres después de nombre-archivo, por ejemplo:

>> **save datos x y z** (enter)

De esta manera se guardan en el archivo datos las variables x y z. Todas las variables se guardan en formato binario de doble precisión. Cuando quiera cargar los valores contenidos en el archivo datos se escribe

>>**load datos** (enter)

Ejemplo:

```
>>x=2  ; A=5;   (enter)
>>save datos x A  (enter)
>>clear   (enter)
>>load datos  (enter)
```

Si escribimos **x** obtenemos `x = 2`

Variables de arreglo

a) ***Unidimensionales***: Las variables de arreglo unidimensionales tienen la forma de fila o columna y están íntimamente relacionadas con los vectores En Octave, *arreglo de fila* es lo mismo que vector fila y *arreglo de columna* es lo mismo que vector columna. El tamaño del vector no tiene que declararse previamente pues se ajusta automáticamente. Para ingresar los elementos de un vector se tiene que tener en cuenta dos cosas:

las filas se generan con **;**

las columnas se generan dejando espacio en blanco

Los elementos deben quedar encerrados entre corchetes (no dejar espacios en blanco sin necesidad.) Por ejemplo:

```
>>v1=[1 2 3]  (enter)
v1 =
     1     2     3
>> v2=[1;3;5]  (enter)
v2 =
     1
     3
     5
```

Si desea un elemento particular teclee el nombre con su subíndice correspondiente:

```
>>v2(3)
  ans =
       5
```

Si quiere modificar un elemento en particular puede hacerlo directamente:

```
>> v1(3)=10;
obteniendo el vector
     v1 =
     1     2    10
```

El número de elementos puede incrementarse definiendo elementos adicionales, rellana con ceros en caso de necesidad. Por ejemplo:

```
>> v1(7)=4
  v1 =
     1     2    10     0     0     0     4
```

Otra forma de escribir un arreglo fila con un incremento o decremento fijo es :

```
>> x=-2:3:8  (enter)
x =
    -2     1     4     7
>>y=7:-0.5:4  (enter)
y =
    7.0000    6.5000    6.0000    5.5000    5.0000
4.5000    4.0000
```

El operador apóstrofe equivale al operador de transposición en álgebra de matrices .Por ejemplo:

```
>> y'  (enter)
ans =

    7.0000
    6.5000
    6.0000
    5.5000
    5.0000
    4.5000
    4.0000
```

Si necesita saber el tamaño de un vector utilice el comando **length** de la siguiente manera:

```
>> length(x)  (enter)
ans =
     4
```

b) ***Bidimensionales***: Un arreglo bidimensional es lo mismo que una matriz, se puede definir especificando sus elementos, recordando que espacios generan columnas y el símbolo punto y coma filas. Sea por ejemplo una matriz de orden 3x3 definida por:

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 3 & 8 & 5 \end{bmatrix}$$

Los elementos se introducen así :

```
>> m=[1 2 3;4 5 6;3 8 5]
m =
     1     2     3
     4     5     6
     3     8     5
```

Cómo extraer filas o columnas de una matriz *?*

Podemos extraer una fila o columna completa de una matriz empleando el signo **:** Recordando que a (i , j) se refiere al elemento de la i-ésima fila y la j-ésima columna. Si queremos la segunda fila de la matriz m definida anteriormente escribimos **m(2,:)** donde 2 se refiere al número de fila y el símbolo **:** significa todas las columnas

```
>> m(2,:)  (enter)
ans =
     4     5     6
```

Idem en el caso de querer obtener una columna determinada, pero ahora se indicará la columna y todas las filas con el símbolo **:** de la siguiente manera

```
>> m(:,3)  (enter)
ans =
     3
     6
     5
```

Si necesitamos trabajar con un elemento determinado se indicará la fila y columna correspondiente:

```
>> m(2,3)  (enter)
ans =
     6
```

También podemos extraer una parte de la fila o columna para ello trabajamos con los vectores índices. Por ejemplo si tenemos la matriz

```
B=[1 3 5 7;2 4 6 8;1 2 3 4;5 6 7 8]
  B =
       1     3     5     7
       2     4     6     8
       1     2     3     4
       5     6     7     8
```

Supongamos que queremos obtener de la columna 3 los elementos de la 1º 3º y 4º fila, construimos un vector de índices para las filas o sea u=[1 3 4] y luego procedemos así:

```
>> u=[1 3 4];B1=B(u,3)  (enter)
B1 =
     5
     3
     7
```

Análogamente si queremos obtener de la fila 2 los elementos ubicados en la 1º, 2º 3º columna. El vector de índices no necesariamente debe definirse por separado

```
>> B2=B(2,1:3)  (enter)
B2 =
     2     4     6
```

Otro ejemplo puede ser obtener una submatriz de B formada por la 2º y 3º fila y la 1º y 3º columna

```
>> B3=B([2 3],1:2:3)  (enter)
B3 =
     2     6
     1     3
```

Cómo agregar filas o columnas de una matriz ?

Construyamos con el comando **rand** una matriz de números aleatorios entre 0 y 1 de orden 3x4, por ejemplo:

```
>> M=rand(3,4)  (enter)
M =
    0.2190    0.6793    0.5194    0.0535
    0.0470    0.9347    0.8310    0.5297
    0.6789    0.3835    0.0346    0.6711
```

Supongamos que queremos aumentar una fila, debemos construir un vector de igual longitud que las filas de M, por ejemplo:

```
>> f4=rand(1,4)  (enter)
f4 =
    0.0077    0.3834    0.0668    0.4175
```

Luego agregamos la fila a la matriz, recordando que el comando que agrega filas es **;**

```
>> M1=[M;f4]  (enter)
M1 =
    0.2190    0.6793    0.5194    0.0535
    0.0470    0.9347    0.8310    0.5297
    0.6789    0.3835    0.0346    0.6711
    0.0077    0.3834    0.0668    0.4175
```

Análogamente si queremos aumentar una columna a una matriz

```
>> c5=[1;2;3;4];    M2=[M1 c5]  (enter)
M2 =
    0.2190    0.6793    0.5194    0.0535    1.0000
    0.0470    0.9347    0.8310    0.5297    2.0000
    0.6789    0.3835    0.0346    0.6711    3.0000
    0.0077    0.3834    0.0668    0.4175    4.0000
```

Operaciones con arreglos

a) Los arreglos bidimensionales se pueden sumar, restar y multiplicar, siguiendo las propiedades de dimensionamiento propias del álgebra matricial Ejemplos: Sean

```
>> a=[1 2 3];   (enter)
>> b=[4 5 7];   (enter)
>> c=[1 2 3; 2 4 6];   (enter)
>>d=[2 3 1 1; 2 3 4 5;1 2 3 4]  (enter)
>> ab=a+b  (enter)
ab =
     5     7    10
>> F=a'-b'  (enter)
F =
    -3
    -3
    -4
>> G=a*b'    (enter)
G =
    35
>> H=c*d     (enter)

H =
     9    15    18    23
    18    30    36    46
```

Como ejercicio pruebe las operaciones: J=a*b K=a+c M=b*d

Qué conclusiones obtiene ?

b) Los arreglos bidimensionales se pueden multiplicar, dividir, elevar a la potencia n-ésima y aplicar cualquier función a cada uno de sus elementos, considerados como números independientes, se diferencia de la anterior al escribirse anteponiendo el punto. Así el comando **.*** indica multiplique elemento por elemento. Por ejemplo de los arreglos dados anteriormente calcule:

a1=a.*b a2=a+2 a3=a.^2 a4=a./b a5=a.\b

Comandos especiales

Tamaño de una matriz

Para saber el tamaño de una matriz el comando **size** nos proporciona el número de filas y de columnas por ejemplo:

```
>> [m n]=size(M)   (enter)
m =
      3
n =
      4
```

Suma de elementos de columnas

El comando **sum** aplicado a una matriz genera un vector fila cuyos elementos son la suma de las columnas de la matriz ; por ejemplo pruebe Ud. introduciendo una matriz C de orden 3x4 y luego aplique sum(C).

Cómo haría para obtener la suma de los elementos de las filas ?

Ordenamiento

Octave cuenta con el comando **sort** que permite ordenar en forma creciente los elementos de un vector , guardando las posiciones originales de los elementos en un vector de índices, por lo que este comando tiene dos argumentos de salida Veamos el siguiente ejemplo:

Sea el vector a=[1 9 7 3] y necesitamos ordenar sus elementos procedemos así

```
>> a=[1 9 7 3];   (enter)
>> [aa i] = sort(a)   (enter)
aa =
     1     3     7     9
i =
     1     4     3     2
```

El vector **i** guarda las posiciones originales de los elementos por ejemplo el 3 estaba ubicado en la columna 4 del vector **a**

Máximos y mínimos:

Dado un vector el comando **max** calcula el valor máximo de los elementos del vector, en caso de una matriz, entrega por resultado un vector cuyos elementos son el máximo de los vectores columnas de la matriz.

Es un comando que tiene dos argumentos de salida. Por ejemplo si se aplica a un vector, el comando **max** entrega como resultado la componente máxima del vector y la posición que ocupa esa componenete máxima en el vector dado

En el caso de una matriz calcula el valor máximo de cada columna, entregando como resultado el vector que contiene los valores máximos y un vector posición que guarda el númro de la fila en la que está ubicado cada valor máximo de las diferentes columnas

Ver otras opciones en el **help max**. Idem para el comando **min**

Ejemplos:

```
>> a=[ 2 -1 15 7]
>>[Ma ia]=max(a),   (enter)
    Ma =
       15
    ia =
        4

>> [ma ja]=min(a)   (enter)
   ma =
       -1
   ja =
        2
>>d=[2 1 2 4; 1 3 4 5;1 2 3 1]  (enter)
    d=
              2     1     2     4
              1     3     4     5
              1     2     3     1
>> [MD id]=max(d)  (enter)
MD =
     2     3     4     5
Id =
     1     2     2     2
```

>> [md jd]=min(d) (enter)

```
md =
      1     1     2     1
jd=
      2     1     1     3
```

Determinante de una matriz

El determinante de una matriz se calcula con el comando **det.** Ejemplo

>> A=[1 3 ;2 1]; (enter)

>> D=det(A) (enter)

```
     D =
       -5
```

Inversa de una matriz

La inversa de una matriz se calcula con el comando **inv** y se aplica de la siguiente manera

>> B=inv(A) (enter)

```
   B =

     -0.20000   0.60000
      0.40000  -0.20000
```

*Comando **diag***

El comando **diag** tiene dos resultados diferentes, según se aplique a un vector o a una matriz

a) Aplicado a una matriz da por resultado la diagonal que se indica de la matriz. Por defecto da la matriz principal Ejemplo

>> d=diag(A) (enter)

```
   d=
      1   1
```

>> d1=diag(A,1) (enter) % da por resultado la diagonal superior

```
    d1=
             3
```

>> d_1=diag(A,-1) (enter) % da por resultado la sub diagonal

```
    d_1=
        2
```

b) Aplicado a un vector da por resultado una matriz donde el vector ocupa el lugar de la diagonal indicado, y el resto de los elelemntos es cero. Por ejemplo

>> v=[3 1 2]

>> A1=diag(v)

```
A1=
     3     0     0
     0     1     0
     0     0     2
```

>>A2=diag(v,-1)

```
A2=
     0     0     0     0
     3     0     0     0
     0     1     0     0
     0     0     2     0
```

Matrices especiales

Los comandos especiales que se usan para crear matrices especiales son:

eye(n)	genera una matriz unidad de orden nxn
zeros(m,n)	genera una matriz nula de orden mxn
rand(m,n)	genera una matriz de números aleatorios de orden mxn
ones(m,n)	genera una matriz llena de unos de orden mxn

Programación

Como todo lenguaje de programación, podemos encontrar bifurcaciones y bucles. Las bifurcaciones sirven para realizar una u otra operación:

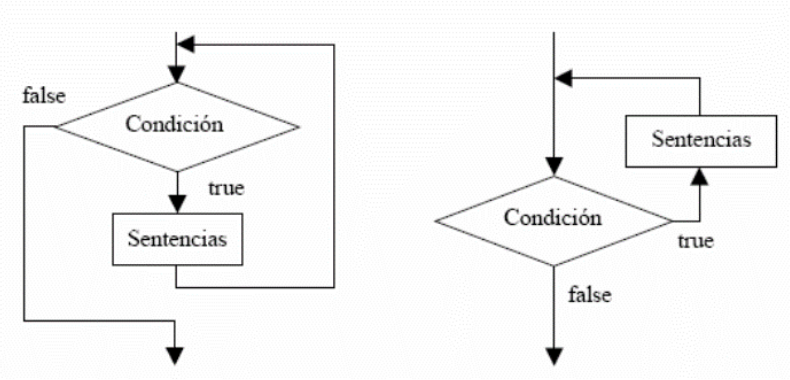
Los bucles nos permiten realizar varias iteraciones de un mismo proceso, o sea, realizar una misma operación sobre distintos elementos.

Podemos encontrar varios tipos de bucles:

Bifurcaciones:

a. if
b. if else

a. La sentencia **if / end** sirve para hacer bifurcaciones .También es admitido en Octave y su ejecución es la misma que en otros lenguajes.



while if else

Una sola sentencia que utilizamos si es verdadera y sino no hacemos nada:

```
if (condition)
    then-body
end
```

b. Utilizando la expresión *else* con la que conseguiremos hacer uso de una expresión u otra si es consecuentemente *true o false*.

```
if (condition)
    then-body
else
    else-body
end
```

Ejemplo

```
r=2;
if  r>0,
  area=pi*r^2,
end
area =
   12.5664
```

Repeticiones

a. While
b. For

a. While

While repite las sentencias en el cuerpo siempre que la condición sea verdadera.

```
while (condition)
      body
end
```

Ejemplo

```
>>r=0;
>>while r<5
       r=r+1;
      duplo=r*2;
      disp([r duplo])
   end
```

```
 1     2
 2     4
 3     6
 4     8
 5    10
```

b. For

Repite una serie de sentencias un número determinado de veces, sin importar los procesos que ocurran dentro, por lo que la única manera de salir del bucle es esperar que acabe o tulizar break(terminar)

```
for   i =vi :paso vf
      n sentencias
end
```

Ejemplo Se desea construir un vector cuyas componenete son el duplo de ciertos número que se indican

```
>> for r=1 : 2 : 5
      duplo=i*2;
      disp([r,duplo])
   end
```

Los cálculos del ciclo no comenzarán hasta que se teclee end y se pulse la tecla return. El enunciado **disp** imprimirá los valores de r y duplo en una línea cada vez que se calcule duplo

```
1       2
2       4
5       10
```

Operadores de relación son :

mayor que	>
menor que	<
igual que	==
mayor o igual que	>=
menor o igual que	<=
distinto de	~=

Operadores lógicos

Los enunciados lógicos ***and*** **y** ***or*** se denotan con **&** y **|** respectivamente**.** Por ejemplo, la ecuación condicional

Si g > 3 o g < 0 entonces a = 6

se escribe

```
>>if      g>3 | g<0
        a=6;
    end
```

Funciones Matemáticas

Al igual que otros lenguajes de programación, Octave tiene numerosas funciones matemáticas, desde las elementales hasta las de alto nivel.

Las funciones en general presentan dos grandes diferencias con respecto a otros lenguajes

Se aplican a variables reales o complejas sin discriminación alguna

Se aplican a argumentos vectoriales y matriciales.

Las funcones se pueden definir de diferentes formas:

1. Funciones de Librerías

Argumentos complejos : Veamos un ejemplo

```
>> i^2
ans =
      -1
```

Argumentos de arreglo La mayor parte de las funciones de Octave pueden aceptar vectores y matrices como argumentos Por ejemplo:

```
>> x
     x =
          -1      4      3
           9     16      2
```

entonces la raíz cuadrada de x producirá

```
>>sqrt(x)
   ans =
           0 + 1.0000i    2.0000    1.7321
           3.0000         4.0000    1.4142
```

Si queremos calcular el seno a los elementos de la segunda columna obtenemos un vector de igual longitud dado por:

```
>> sin(x(:,2))
   ans =
       -0.7568
       -0.2879
```

Todas las funciones que se usan en calculadoras se pueden aplicar con Octave . A ellas hay que agregar funciones como

round(x) : redondeo hacia el entero más próximo

fix(x) : redondea hacia el entero más próximo a 0

floor(x) : valor entero más próximo hacia -∞

ceil(x) : valor entero más próximo hacia +∞

Al igual que en C, una función tiene ***nombre***, ***argumentos de salida*** y ***argumentos de entrada***.

argumentos de salida: pueden ser uno o más números y/ o mtarices

argumentos de entrada : pueden ser uno o más números y/ o mtarices

2. Creación de un programa en forma de archivo .m

La ejecución de comandos en la ventana de trabajo es apropiada si no son muchas líneas. Además en el caso de procesos iterativos se dispone del comando doskey que permite recuperar los últimos comandos. Sin embargo muchas veces conviene escribir un proceso o una función en un archivo.m que permita ejecutar un conjunto de comandos, los cuales quedan guardados en disco y pueden corregirse tantas veces como sea necesario.

Para crear un **archivo .m ,** en caso de trabajar en Windows se debe desplegar el menú *Archivo* de la parte superior de la ventana de comandos y seleccionar *Nuevo,* en el cual aparecen tres opciones, elegir *nueva función* , se abrirá una ventana del editor seleccionado al instalar Octave, allí se escriben los comandos necesarios y luego se sale grabando , guardando el archivo con extensión **.m** Luego desde la ventana de trabajo se podrá ejecutar con teclear el nombre del archivo, o también puede ejecutarse desde otro archivo

El signo **%** en un archivo **.m** indica que lo escrito a continuación en la misma línea es un comentario y debe ignorarse durante los cálculos. Esto es muy útil porque los comentarios en un archivo **.m** ayudan a explicar el significado de las variables y procedimientos.

En los archivos **.m** pueden escribirse funciones, procedimientos o programas. Como en otros lenguajes tenemos programas que incluyen varias subrutinas y/o funciones, de igual manera tenemos en Octave archivos que pueden llamar o ejecutar a otros archivos **.m**.

Cómo escribir funciones de usuario propias

El término *función* es considerado en el mismo sentido que en Matemática, o sea tenemos variables independientes o *argumentos de entrada* y variables dependientes o *argumentos de salida* Una función **.m** puede tener más de una variable de entrada y también entregar más de un resultado, o sea más de una variable de salida. Para que Octave reconozca un archivo **.m** como función debe tener una sentencia que la declare como tal. Veamos el siguiente ejemplo

Función con una variable de salida

Se necesita calcular la siguiente función y= $\frac{2x^3 + e^x - 1}{x^2 + sen(x) - 3}$

El archivo **.m** se guarda con el nombre **ejem1.m** y se escribe así:

```
function   y=ejem1(x)
y = (2*x.^3+exp(x)-1)./(x.^2+sin(x)-3);
```

Observe que el nombre del archivo **.m** es idéntico al nombre de la función que aparece a la derecha del signo igual. En el archivo se utilizan los operadores aritméticos precedidos por el punto, lo cual indica que el argumento x puede ser un escalar, un vector o una matriz. Una vez que se guarda el archivo con el nombre ejem1.m, se puede ejecutar desde la ventana de comandos o en otro archivo **.m** de la siguiente manera:

```
>> y=ejem1(2)
    y =
      11.7263
```

Si el argumento es una matriz , **v=[1 7 4];** el resultado es

```
>> y=ejem1(v)
  y =
    -3.2095   38.1858   14.83266
```

Función con más de una variable de salida

Ejemplo: Dado un vector hallar el promedio de sus componentes y el máximo elemento. Procedemos de la siguiente manera:

```
function  [p,q]=ejem2(x)
      n=length(x)
       p=sum(x)/n;
       q=max(x);
```

Para ejecutarlo se evalúa así:

```
>>[prom , M]= ejem2(v)
       prom =
              4
        M =
             7
```

Función que utiliza otra función

El argumento de una función puede ser otra función. En dicho caso el nombre de la función debe ir entre literales si es un arcivo.m cuando se ejecuta desde la ventana de comandos

Ejemplo: Supongamos que se necesita calcular el máximo valor que alcanza la función del ejem1 , y el valor medio en el intervalo [-2 2]. Para ello construimos un vector x que tenga por extremos -2 y 2 con un paso de 0.1.

Se debe construir un archivo **ejem3.m** que ejecute lo mismo que ejem2 pero sobre los valores de la función ejem1 de la siguiente manera:

```
function  [p,q]=ejem3(F,x)
        y=feval(F,x)
        p=sum(y)/n;
        q=max(y);
```

El comando **feval** es un comando de dos argumentos de entrada, nombre de la función y el valor x. Evalúa la función de nombre F sobre los valores de x. Luego

```
>> x=-2:.1:2;
>>[prom,max]=ejem3('ejem1',x)
   prom =
      -88.5345
   max =
       43.3197
```

Función definida con el comando @

Este comando permite definir funciones que pueden usarse más fácilmente , tienen el comportamiento de una expresión simbólica

En su definición se debe indicar entre parántesis el nombre de la variable independiente. Puede tener más de una variable. Las variables pueden sere un número, un vector o una matriz.

Ejemplos:

```
>>f=@(x) x.*sin(x)
>> x=-pi:0.1:pi;
>> y=f(x);
```

es conveniente el comando **;** para que no muestre el vector en pantalla por tener muchos elementos. Recordar que el punto precede al producyo * para indicar que la multiplicación debe hacerse elemento por elemento

Otro ejemplo

Definir la función $y = c_1 e^{c_2 x}$ donde c=[2 -1] son las constantes de la función y calcularla en el intervalo [0 1]

```
>> c=[2 -1];
>> f=@(x) y=c(1)*exp(c(2)*x)
>> x=0:0.1:1;
>> y= f(x);
```

Este comando es muy práctico para definir una función cuanco se la quiere graficar

Graficación simple

Suponga que desea graficar un conjunto de datos dados por los puntos (x_i , y_i) , con i=1,2,...n Es necesario preparar los arreglos x e y de tal manera que tengan la misma longitud y ambos sean fila o columna. Los datos se grafican con **plot** por ejemplo

Ejemplo 1 Graficar la función $y=sen(x).e^{-0.4x}$, en el intervalo [0 , 10], se procede así:

a) lo primero que se debe hacer es definir los valores de la variable independiente en un vector ejemplo

```
>> x=0:0.05:10;
```

b) Se define la función con el comando @ y se evalúa en el vector x

```
>> f=@(x) sin(x).*exp(-0.4*x);
>> y=f(x);
```

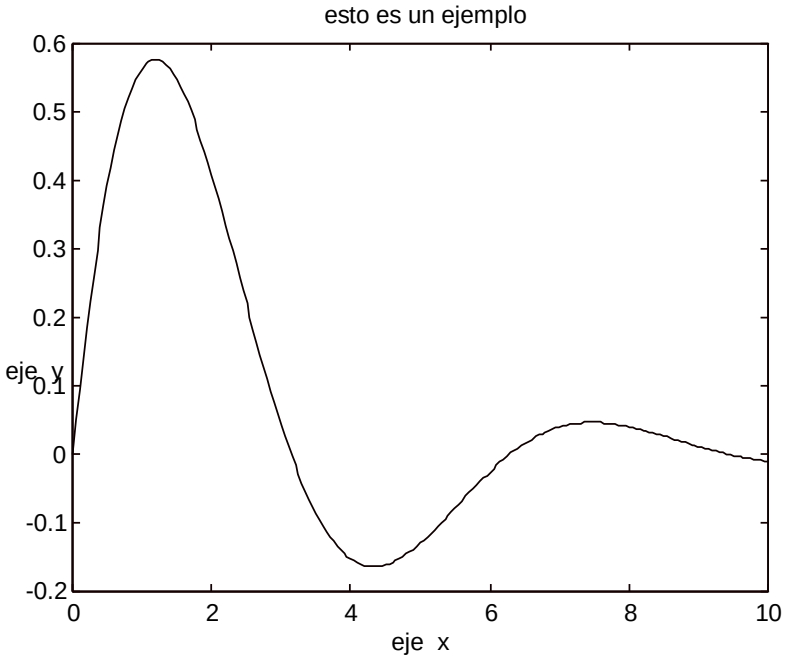
c) se grafica el par (**x,y**) con el comando plot

```
>>plot(x,y)
```

Hay comandos que permiten escribir etiquetas en el eje “x” en el eje “y” , título

```
>>xlabel(' eje  x ');
```

```
>>ylabel('  eje  y ');
>>title (' esto es un ejemplo')
```



Se sugiere para más información leer el help del comando plot. Se pueden cambiar los tipos de trazos, color, carteles, tamaños etc.

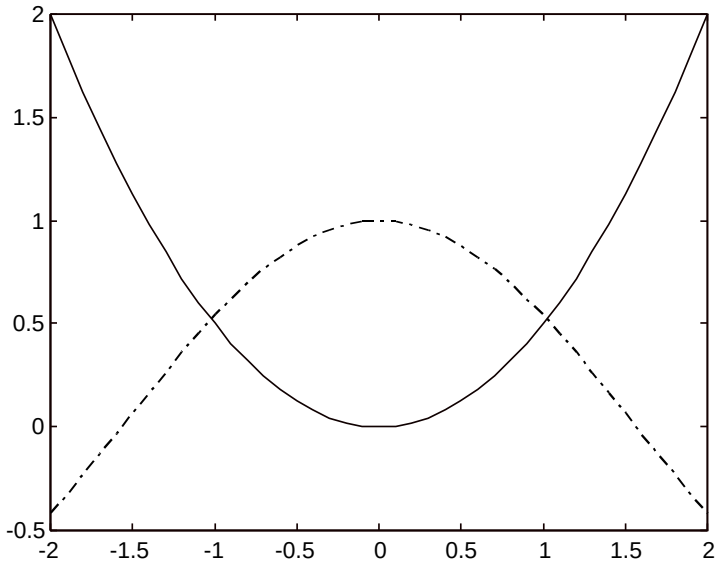
Si desea puede graficar más de una función al mismo tiempo. Por ejemplo:

Ejemplo 2

```
>> x=-2:0.1:2;
>> f=@(x) x.^2/2
>> y=f(x);
>> z=cos(x);
plot(x,y,x,z,'-.')
```

El mismo efecto se logra si se utiliza el comando **hold on** , que fija la ventana gráfica y agrega las gráficas que se indican a continuación . Ejemplo

```
>> x=-2:0.1:2;
>> f=@(x) x.^2/2
>> y=f(x);
>> z=cos(x);
>> plot(x,y)
>> hold on
>> plot(x,z), hold off
```



Graficación de funciones con fplot

Otra forma de graficar funciones individuales es con **fplot** y se usa del siguiente modo:

fplot('nombre-f ', [xmin , xmax])

nombre-f es el nombre de la función o del archivo **.m** de función que se desea graficar

[xmin , xmax] se define el vector donde se escribe el mínimo y máximo valor de x.

Ejemplo 3:

```
>>fplot('tan',[-2*pi 2*pi])
```