# MODULO: COMUNICACIÓN ELECTRONICA

# INTRODUCCION AL USO DEL

# SOFTWARE MATEMATICO

# FreeMat / Matlab /Octave

***Temas a acreditar en FreeMat (Matlab u Octave)***

**El alumno tiene que dominar (acreditar) los temas que se detallan a continuación:**

- **Comandos de FreeMat/Matlab/Octave.**
- **Definición de variables: escalares, vectores y matrices.**
- **Operadores punto, dos puntos, punto y coma, comentarios.**
- **Definición y evaluación de polinomios y sus raíces, polyval, poly y roots.**
- **Definición de funciones no polinómicas, en línea. Inline**
- **Cálculo de raíces de ecuaciones con fzero**
- **Escritura de matrices y vectores en FreeMat/Matlab/Octave.**
- **Solución de sistemas de ecuaciones lineales con Matlab/FreeMat/Octave.**
- **Graficación de funciones. Comandos: plot, xaxis, xlabel, grid, title, etc.**
- **Gráfico de un sistema de ecuaciones lineales de 2x2**
- **Como hacer un Informe de un TP resuelto en Matlab/FreeMat/Octave**

NOTA: **Cuando se utiliza el término 'acreditar' significa que el alumno debe saber realizar esas operaciones y demostrarlo sobre una PC.**

***<u>Breve introducción al uso de FreeMat/ MATLAB/Octave.</u>***

**La mayoría de los comandos básicos que se utilizan en este curso de FreeMat, se explican para Matlab, son compatibles con otros programas similares: Octave y Scilab.**

**FreeMat es un entorno gratuito resolver problemas de matemáticas en ingeniería rápida, creación de prototipos científicos, aplicar en resolución de métodos numéricos y procesamiento de datos. Es similar a sistemas comerciales como MATLAB de Mathworks e IDL de Research Systems, pero es de código abierto (OPEN SOURCE).**

MATLAB es un software para matemáticas y métodos numéricos muy moderno, potente y fácil de aprender. Es el nombre abreviado de **MAT**rix **LAB**oratory y está orientado al cálculo con matrices y vectores, pero también puede trabajar con escalares y caracteres. Es una buena herramienta para realizar gráficos. Octave es un soft similara los anteriores, con características GNU Linux. Existen además otros softs similares, por ejemplo Scilab.

En síntesis, las características más salientes de estos soft son:

- Permiten realizar cálculos y operaciones en modo Comando.
- No es necesario declarar previamente las variables. Al asignarlas se crean automáticamente.
- Disponen de una amplia biblioteca de funciones científicas.
- Permiten definir funciones. Se pueden hacer programas breves (scripts) muy fácilmente.
- Pueden realizar fácilmente gráficos de 2D y 3D.
- Tienen una excelente ayuda en línea y también en Internet.

En este curso se va a trabajar **en modo comando**, para que el alumno se familiarice con este sistema fin de utilizarlo en el futuro en otras materias. Para mayor información, y si el alumno tiene la inquietud de aprender más, hay una extensa bibliografía y sitios en Internet para avanzar en su uso. Además, en el caso de MATLAB, su Help es excelente.

***Recordemos que en informática a veces, más que saber "hacer algo", hay que saber averiguar cómo hacerlo.***

El libro a partir del cual se elaboró este material se encuentra en biblioteca de la Facultad: "Aprenda a utilizar MATLAB en Ingeniería" que es parte de una colección de la Universidad Politécnica de Madrid https://cimec.org.ar/foswiki/pub/Main/Cimec/CursoFEM/matlab70primero.pdf (Ultimo acceso 29/05/2020).

**Veremos en este apunte los siguientes puntos (recordar que MATLAB es equivalente a FreeMat, en este caso):**

1. **MATLAB en modo comando. Comandos básicos y entorno de trabajo.**

2. **Funciones Polinómicas. Ceros.**

3. **Gráficos con MATLAB.**

4. **Funciones NO polinómicas. Definición. Ceros. Graficación.**

## 5. Operaciones básicas con matrices y vectores. Resolución de SEL.

### 1. ***Comandos básicos de Matlab y Entorno de trabajo.***

Independientemente de la versión de MATLAB (7.0, 6.00 R12 o anteriores, 5.3 p.ej.), lo que nos interesa es aprender a trabajar en la Ventana de Comandos (Command Window). Decimos esto pues MATLAB 6.00 tiene un entorno de trabajo con varias ventanas, mientras que la versión 5 sólo tiene la ventana de comandos. Para uniformar y no detenernos demasiado en el trabajo de entornos, que son muy familiares en Windows, trabajaremos solamente en la **Command Window.**

**Prompt**
Al ejecutar MATLAB aparece esta ventana COMMAND WINDOW con un **prompt** (*aviso* de que espera se ingrese un comando).

El prompt de MATLAB (o de Octave) es el símbolo **>>** y el el de FreeMat **-- >**

Al lado del mismo escribiremos los comandos según lo que se desee hacer.

Matlab guarda los comandos en **un historial** o pila, y con las flecha de dirección 'hacia arriba' se pueden recuperar los últimos escritos y reutilizarlos (corregirlos o modificarlos). **No se puede ir con el mouse a 'rescatar' un comando anterior, se debe 'subir' con la tecla flecha.**

***En este apunte escribiremos el prompt para indicar el comando que se debe usar, pero recordemos que <u>lo escribe MatLab esperando nuestra orden</u>.***

**<u>Modo compacto</u>**
Para que MATLAB (solo Matlab) no deje tantos renglones en blanco en sus respuestas (y aprovechar mejor la pantalla) conviene tipear al inicio del trabajo este comando:

***>>format compact***

**<u>Como escribir un comando en MATLAB.</u>**

Si se escribe un comando, el resultado se muestra de inmediato.

Ejemplo:

```
>> a=5
a =
    5

>> b=3
b =
    3

>> c=a*b
c =
```

15

Como se observa, se definieron las variables a y b.
Al escribir el comando c= a*b se está definiendo, con un cálculo, el valor que almacena c.

Si en una fórmula hay constantes se usan directamente, por ejemplo para calcular $a^2$ y almacenar su valor en d, el 2 es una constante:

**>>d=a^2**

Si se escribe directamente la operación sin asignarla a una variable (c, por ejemplo) Matab guarda el resultado en una variable propia llamada **ans** (**ans**wer o respuesta):

**>> a*b**
ans=
15

**Punto y coma (;)**

Para que no aparezca la respuesta, hasta que no la solicitemos, se escribe al final de la línea un punto y coma **(;)**

**>> a=5 ;**
**>> b=3 ;**
**>> c=a*b**
15

Aparece o muestra 15 porque al asignar **c** no se colocó el ( **;** ) al final de esa línea.

**Operadores aritméticos:**

Son los usuales: + - * / y ^ (potencia).
Los paréntesis se utilizan para agrupar operaciones. Los corchetes son usados para definir matrices y vectores.

***Definición de variables: escalares, vectores y matrices.***

**Variables.**

En el ejemplo anterior hemos usado las variables a, b y c, es decir le asignamos a la variable **a** el valor 5 y a la variable **b** el valor 3, luego a la expresión (a*b) la hemos llamado **c**, generando así una nueva variable **c**.

Los nombres de variables deben empezar con una letra y pueden tener hasta 31 caracteres (letras y números) de largo. MATLAB **distingue** entre mayúsculas y minúsculas.

Así los nombres: ***c**audal* y ***C**audal* representan dos variables distintas.

Se recomienda usar nombres de variables **nemotécnicos** (velocidad, altura, ancho).

Como norma a las **matrices** se las escribe con nombres de variables en **mayúsculas**.
Se reservan las **minúsculas** para **escalares y vectores**.

A su vez el comando siguiente da error, pues está definida la variable **a** minúscula no la **A** mayúscula:

**>>c = A*b**
La respuesta de Matlab es
***??? Undefined function or variable 'A'.***

Lo mismo pasa si se escribe:

**>>c = ancho*base** (ninguna de esas variables tienen valores asignados)

**Recordemos que una variable escalar es aquella a la que se le puede asignar un solo valor.**

Ejemplo, si escribimos:

**>>a=5** estamos definiendo la variable escalar **a** con el valor 5 (‘un solo lugar de memoria en la PC’)

Si en cambio escribimos

**>>a= [ 8 1 3 6 9 4 ]**

Estamos definiendo un vector **a** de 6 elementos donde el elemento a(1) (a sub 1) vale 8 y a(6) vale 4.

Algunos nombres en MATLAB están reservados, por ejemplo:

**pi** (en minúsculas representa 3.1416..).

Tampoco se pueden usar comandos o funciones de MATLAB como nombres de variables.
end=4 produce un error porque end es un comando reservado.

sin =5 <u>no da error</u> pero va a anular la función seno de MATLAB en **esta sesión de trabajo**. Si se comete este tipo de error se debe salir de Matlab y volver a entrar para regenerar la función, en este caso seno.

<u>**Operador dos puntos (:)**</u> Este operador es muy importante y puede usarse de varias formas.

Su formato es: ***Valor inicial: Incremento: Valor final***

Si se usan 2 operadores se entiende que son: ***Valor Inicial : Valor Final*** y el ***incremento*** vale ***1 por omisión.***

Ejemplo: **definir una variación de x desde -1 a 1 con incremento de 0.25**

Se parte de un **xinicial** con un **dx (salto o incremento)** hasta un **xfinal.**

>> **x = -1.00 : 0.25 : 1.00** este comando define un vector **x** con los siguientes valores:

-1.0000 -0.7500 -0.5000 -0.2500 0.0000 0.2500 0.5000 0.7500 1.0000

**Para definir vectores o serie de datos (Rangos).**

>> **x = 1 : 6** define un vector fila x de 6 elementos donde x(1) vale 1 y x(2) vale 2 ....

```
x =
    1     2     3     4     5     6
```

En este caso, al no poner el incremento se lo supone 1, y 6 es el Valor Final.

**<u>El operador. (punto)</u>**

Este operador sirve para realizar las operaciones elemento por elemento en vectores y matrices.

Por ejemplo se tiene definido un vector de valores de **x**. Se quieren calcular los valores **$y=x^2$.**

Se debe escribir el comando en Matlab del siguiente modo:

**>> y = x.^2**

Con un **punto** luego de la variable **x** o antes del operador aritmético de la potencia.

El operador punto (**.**) se usa en el caso de los operadores *** /** y **^.** No es necesario ponerlo para la suma y resta.
Si queremos hacer la operación y = x / (x + 1), el comando debe ser:

**>> y=x. / (x+1)** **(**con el punto luego de la **x.** Los valores de x ya están definidos)

O para hallar la función inversa **y = 1/x**, para ese conjunto de valores de **x**, se debe escribir:

>>**z = 1. /x (No olvidar el punto y no confundir el punto con el punto decimal)**

Ejemplo si queremos hallar la función inversa z = 1.25 / x se debe escribir:

**>> z = 1.25 . /x**

Veamos un ejemplo más:

Calcular el valor de **$x^2$** para x entre 0 y 8 variando de a 2.

**>> x= (0 : 2 : 8 )**
x = 0 2 4 6 8

**>> y = x . ^2**
y = 0 4 16 36 64

Entonces tenemos 2 vectores o tabla de datos **x-y**

**Comentarios con Matlab**

Si queremos hacer una aclaración o comentario en Matlab, se usa el operador **%** que se puede poner en la misma línea del comando o como una línea aparte. NO es un comando ejecutable, solo para aclaración.

**>> y = x . ^2 % Aquí se eleva x al cuadrado y se genera el vector y en función de x**

Otro ejemplo:

**% En el comando siguiente se calculan los logaritmos de y**

**>> y1 = log(y)**

## 2. ***Polinomios con Matlab***

Se define como polinomio de una variable a la expresión:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0 x^0, \qquad x \notin A$$

Donde cada uno de los exponentes son números enteros.

Si tenemos el polinomio: p(x)= **1** $x^3$ **- 1** $x^2$ **- 4** x + **4**

En Matlab/FreeMat/Octave se define un polinomio a través de sus coeficientes mediante un vector con esos coeficientes **ORDENADOS**, o sea comenzando con el coeficiente del término de mayor grado. Para este ejemplo se define el vector llamado **poli3 (u otro nombre cualquiera).** *El nombre del vector que define al polinomio **es a libre elección**, elegimos como nombre nemotécnico **poli3** para indicar (y recordar) que estamos usando un vector que define un polinomio de grado 3. Pero podríamos haber elegido otro nombre.*

**>> poli3=[1 -1 -4 4]**

<u>**Polinomios incompletos y rectas.**</u>

**Si tenemos el polinomio incompleto** p(x)= **1** $x^3$ **- 4** x + **4** se debe COMPLETAR para definirlo. Para ello se pone un 0 en el coeficiente que falte, en este ejemplo para $x^2$**:**

**>> poli3=[1 *0* -4 4]**

**Las rectas** son polinomios de grado 1, así la recta **y= 2x -1** se escribe como polinomio:

**>> r=[2 -1]** el nombre usado es **r** pero podría ser **poli1** o **recta**

Una recta con ordenada al origen igual a 0, por ejemplo: **y=1.5 x** es un polinomio incompleto de grado 1 y se escribe:

**>> r=[1.5 0]**

Una recta con pendiente 0 es una línea paralela al eje x, o sea se define como y=cte. Por ejemplo **y=2** es un polinomio y se escribe:

**>> r=[0 2]**

**Función roots (nombre polinomio)**: **calcula las raíces ó ceros de un polinomio.** Su argumento es el vector que define al polinomio:

>> **roots(poli3)**

```
ans =
   -2.0000
    2.0000
    1.0000
```

También se podría haber escrito

>>**roots( [1 -1 -4  4] )**

**Evaluar el polinomio. FUNCION POLYVAL**

Evaluar un polinomio es encontrar el valor del mismo en determinados valores de la variable independiente x.

Se define un vector **x** ó valores de **x** entre -2 y +2, con incrementos de 0.2, usando el operador dos puntos ***( : )***

>>  **x = -2 : 0.2 : 2**

Se tienen entonces 21 valores de x :  -2 , -1.80  -1,60 ….. 0.00  0.20  0.40 …… 2.00

Para **evaluar el polinomio poli3** se usa la función **polyval** que **evalúa polinomios** en Matlab.

>> **polyval(polinomio , x)**

Esta función tiene 2 argumentos: el **nombre del polinomio** y un **vector de valores de x**. (También puede ser un único valor de x)

>> **y = polyval(poli3,x)**

Se calculan entonces 21 valores de y: 0    2.1280    3.7440 ….     0   -0.5120   -0.8160   -0.8640   -0.6080         0

Si queremos VER la tabla generada por esos valores, en la forma clásica, usamos el **comando transponer**,  que es el **apostrofe ‘** (agregado al nombre del **vector horizontal x** y igual para **el vector fila y**). Además para que se muestren juntos, los ‘pegamos’ en una matriz T de dos columnas, de este modo:

T = [x’ y’]

Se verá:

```
    -2.0000   0
   -1.8000    2.128
   -1.6000    3.744
   ……..
```

## Generación de un polinomio a partir de sus raíces: función poly

### Comando poly

Si queremos generar un polinomio cuyas raíces conocemos, por ejemplo: X1= - 2 y X2= 2

Vemos que será un polinomio de grado 2, ya que tiene dos raíces.

Debemos aplicar la función poly, indicando sus dos raíces:

**>> pol2 = poly([- 2 2] )**

Obtenemos el vector **pol2= [1 0 -4]**

Entonces se generó **el polinomio incompleto asociado**: $y = x^2 - 4$

## Generación de un polinomio de mayor grado

Si escribimos

>>**cubica= poly([ -3 1 3 ])**

obtenemos el vector cubica= [1 -1 – 9 9 ]

Que representa el polinomio: $y = 1\ x^3 - 1\ x^2 - 9\ x + 9$

## 3. Gráficos en Matlab

El comando **plot ( x , y)** grafica pares de valores en un plano x y. Generalmente los valores de x e y son vectores que deben tener la **misma cantidad** de elementos cada uno.

**Graficar un polinomio:**

**Dado el polinomio incompleto** p(x)= **1** $x^3$ **- 4** x + **4,** completarlo y definirlo.

**>> poli3=[1 *0* -4 4]**

Si definimos x:

>> **x = -2 : 0.2 : 2**

Se tienen entonces 21 valores de x : -2 , -1.80 -1,60 ….. 0.00 0.20 0.40 …… 2.00

Para **evaluar el polinomio poli3** se usa la función **polyval** que **evalúa polinomios** en Matlab.

>> **y = polyval(poli3,x)**

Se calculan entonces 21 valores de y: 0 2.1280 3.7440 …. 0 -0.5120 -0.8160 -0.8640 -0.6080 0

Definimos la tabla T:

>>T = [x' y']

Luego con el comando **plot(x,y)** se obtiene la gráfica y=f(x), donde y es la función polinómica:

y = **1** $x^3$ **- 1** $x^2$ **- 4** x + **4**

**>>plot(x,y) %se grafica en la Figura 1, que activa MatLab**

El comando **plot grafica pares de valores**, por eso se pueden graficar más de una tabla de valores x, y

Por ejemplo el comando plot(X,Y,X,Z) hace una gráfica de Y vs. X y 'encima' (en el mismo par de ejes) otra de Z vs. X.

Generalmente los valores de X son los mismos para los distintos valores que van en el eje Y.

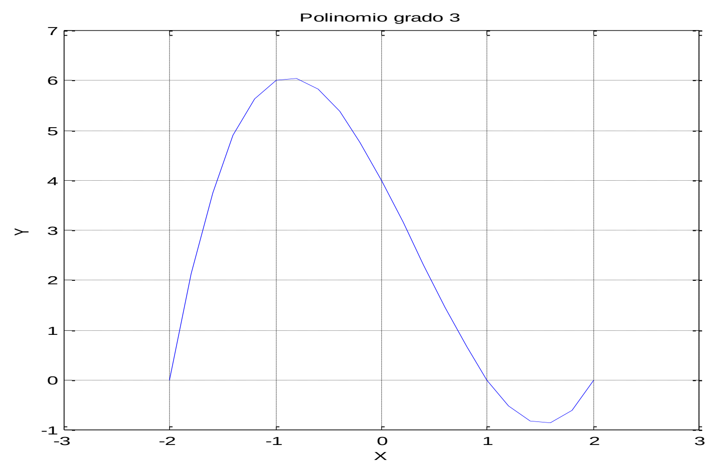
***Ejes, títulos y grilla.***

Los siguientes comandos son complementos para mejorar los gráficos, agregándoles títulos, grilla, ejes:

**Importante:** Para poder agregar elementos al gráfico base, se debe usar el **comando hold on** el cual mantiene el gráfico activo para agregarle, por ejemplo, los nombres a los ejes, las escalas.

| | |
|---|---|
| **hold on** | Mantiene activo el gráfico para agregarle ejes, títulos u otro gráfico si no se indica **hold on** se borra el gráfico para realizar otro |
| **grid on** | Agrega una grilla al gráfico |
| **axis( [ xmin xmax ymin ymax ] )** | Fija las escalas de los ejes x e y |
| **xlabel('Título eje x')** | Agrega los nombres del eje X |
| **ylabel('Título eje y')** | Agrega los nombres del eje Y |
| **title ('Polinomio grado 3')** | Agrega un título |

***Observar: Se ven las raíces en –2 1 y 2 que se calcularon con roots, al cruzar la función el eje x.***




## Marcar las raíces en el eje x

Para marcar por ejemplo la raíz en x=1 se debe definir una recta vertical que pase por la raíz hallada o sea por x=1 y vaya desde un valor de y negativo (por ejemplo -1) a otro positivo (por ejemplo +2).

Recordemos que para definir una recta se necesitan 2 puntos, y luego hacemos un plot con esos 2 puntos

¿Qué puntos? los puntos o pares ordenados deben tener como x el valor de la raíz y los valores de Y deben ser tales que corten el eje X. O sea, los pares ordenados: (1,-1) y (1,2) definen una recta vertical, paralela al eje Y. Como las rectas son verticales los valores de x son iguales, o sea ambos 1:

```
>> plot([ 1 1] , [-1 +2])
```

Otra opción: definir primero los puntos como series de datos de x y de y:

```
>> xr = [ 1   1]    % los valores de x son iguales pues es una vertical
```

```
>> yr = [-1   2]   % los de y son -1 +2 para pasar de los y negativos a los y positivos. Podría haber sido una recta más larga [-4 +4]
```

Entonces el plot sería: `>> plot(xr , yr)`

## Otros gráficos con Matlab/FreeMat/Octave

Podemos también **graficar funciones trigonométricas:**

Por ejemplo si se generan los ángulos (en radianes, pues recordar que los soft usan radianes para los ángulos)

**>> clf** (este comando se usa para borrar de la Figura 1 los gráficos anteriores, que quedaron por efecto de hold on).

**>>alfa=0 : 0.2 : 2*PI** (genera una tabla de valores de ángulos entre 0 y 2 PI radianes o 0 y 360º)

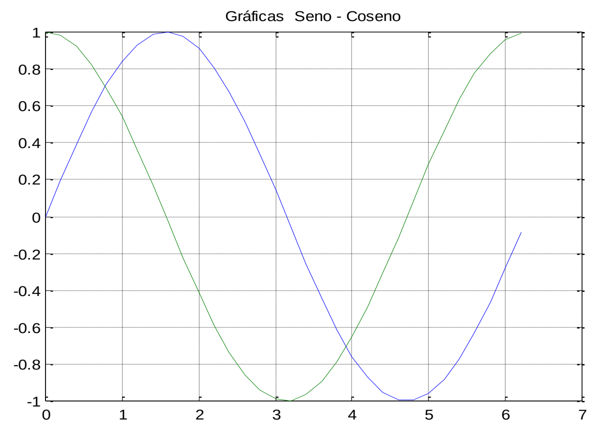
**>> y=sin(alfa)**

**>> z=cos(alfa)**

**>> plot (alfa, y , alfa , z)**

**>> grid on**

**>> title(‘Gráficas de Seno – Coseno)**




**Líneas más gruesas**
Para dibujar con un trazo más grueso se puede agregar el parámetro LineWidth, 2 al final de los pares de valores graficados:

**>> plot ( alfa ,sin(alfa), alfa , z , 'Linewidth' , 2 )**

**Otros parámetros usados en plot**

El parámetro ‘- -‘ se puede agregar al final de una de las series a dibujar, en este caso la 2, y usa línea a trazos.

Podría agregarse un parámetro ‘r’ que hace la línea de esa serie sea color roja (red) o green ‘g’ (verde)

Ver el help del comando plot para los distintos tipos de trazos y colores.

Ejemplo plot( x , yr1 ,’g’ , x , yr2 ,’r‘ ) dibujan la recta 1 en verde y la 2 en rojo.

***Todos los comandos tienen una ayuda, basta con escribir en el help de Matlab:***
***help plot o help polyval por ejemplo.***

<u>**Diferencias a tener en cuenta en los gráficos entre Matlab y Octave**</u>

Las diferencias para los comando básicos de este curso aparecen en la parte de gráficos. En la parte de álgebra y polinomios las funciones y comandos son iguales.

**Comando gráficos ON/OFF**

Básicamente la diferencia es que en Octave los comandos gráficos requieren escribir entre paréntesis y entre comillas o apóstrofes los parámetros on y off

| Matlab | Octave |
|---|---|
| **grid on** | **grid ( “on” )** |
| **grid off** | **grid ( “off”)** |
| **hold on** | **hold( “on” )** |
| **hold off** | **hold( “off’ )** |

**Visualizar el gráfico**

No cerrar la ventana del gráfico en Octave, hay que minimizarla pues si se cierra la ventana **gnuplot graph** no se la puede volver a abrir en esa sesión. *Se deberá salir de Octave y volver a entrar*.

## 4 - Definición de funciones no polinómicas

### *Comando inline*

**Si debemos definir y/o graficar una función que no es un polinomio,** la podemos escribir una vez o podemos definirla como función ‘en línea’ y dejarla como una f(x) para ser usada cuando se desee, con solo asignarle un nombre y el valor o los valores de x que necesitemos.

Este comando es útil para definir funciones ‘en línea’ durante una sesión de trabajo. Conviene usarlo para definir funciones que se van a usar muchas veces en una misma sesión.

**>> nombre=inline(‘expresión de la función’)**

Por ejemplo si tenemos la función potencial (x elevado a un exponente no entero), por ejemplo x elevado a la 2.5:

**$y = x^{2.5}$** y queremos evaluarla en el intervalo [0, 2] con un dx de 0.2

*Nota: Se denomina función potencial a las funciones del tipo **$f(x)=a*x^b$**. Donde b generalmente no es un valor natural. (Si b fuera natural sería un polinomio incompleto de grado b)*

Con inline definimos ‘nuestra función’ que aquí llamaremos **f1** (se puede llamar de distintas formas, siempre que no se usen nombres de funciones propias de Matlab como sin, cos, abs log, por eso generalmente se usa f seguida de un número):

```
>> f1=inline(' x.^2.5 ')
f1 =
    Inline function:
    f1(x) = x.^2.5
```

Matlab nos indica entonces que tenemos una función ‘en línea’ que es $f1(x)= x^{2.5}$.

*Observar que se colocó un punto luego de la x, así de esta manera se puede evaluar un conjunto de valores de x (que es lo que queremos hacer de 0 a 2). Si no se colocaba ese punto la función podía evaluar un solo valor de x y no un conjunto.*

Definimos ahora el conjunto de valores de x:

```
>> x=0 : 0.2 : 2
x = 0    0.2000    0.4000    0.6000    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000
2.0000
```

Y escribimos el comando para evaluar los valores de x, obtenemos los valores de y:

```
>> y=f1(x)
y =      0    0.0179    0.1012    0.2789    0.5724    1.0000    1.5774    2.3191    3.2382    4.3469
5.6569
```

*Importante: x se usó como el nombre del argumento de la función pero no necesariamente los valores que se le dan a la función se deben llamar x.*

Si definimos
**>>a=2** y luego escribimos

**>>f1(a)** obtendremos 5.6569

O a la inversa se puede definir una f(z) como f1(z)=z.^2.5 que funciona igual.

**Ejemplos de inline**
La function f(x)=sen$^2$(x) se escribe:

**>> f2=inline( 'sin(x).^2' )** % recordar el operador punto **.** para que la función sirva para un conjunto de valores

o la función f(x)=sen$^2$(x) + cos$^2$(x) (que siempre da 1)

**>> f3=inline( ' sin(x).^2+cos(x).^2 ' )**

**>> f3(x**) recordemos que x va de 0 a 2 (en este caso de 0 a 2 radianes)
ans =
1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

Ejemplos:

f(x) = 1.45 * x $^{2.5}$ En este caso a=1.45 y b=2.5

f(x) = 1.00 * x $^{0.5}$ Esta función en realidad es la raíz cuadrada de x pues estamos elevando x a la 1/2

En Matlab para definir funciones de este tipo debemos usar inline, pero si queremos que los valores a y b se pasen como argumentos (o valores para generalizar el cálculo con distintos valores de x) debemos observar que la función inline nos 'obliga' a pasar estos datos de a y b junto con los de x cuando se invoque la función:

**>>fpote=inline('a*x.^b') % se define ypote**

fpote =
Inline function:

**fpote(a , b , *x* ) = a * x.^b**

La respuesta de Matlab es que debemos calcular la función con **1 valor para a, otro para b** y **otro (o varios) para x**. Es decir que para Matlab esta es una función de una sola variable (x) y 2 argumentos que pueden tomar un valor por vez, para distintos valores de x.

Por eso en la respuesta de definir con inline, x figura al final.

Para evaluar la función con a=1.45 y b=2.5 y en x valiendo 2 se escribe
**>>fpote(1.45,2.5,2)**

O bien:

**>>y = fpote(1.45,2.5,2)**

y =
 8.2024

Veamos un ejemplos con valores simples, donde **a** vale siempre **1** y x toma valores de 1 a 5 y vamos variando b:

**Cambiaremos el valor de b al usar la función fpote.**

**>> x=1:1:5**
x =
 1 2 3 4 5
**>> y = fpote(1,1,x)**
y =
 1 2 3 4 5 el resultado es igual a x porque la función queda y = 1 * $x^1$

**>> y=fpote(1,2,x)**
y =
 1 4 9 16 25 el resultado es igual a $x^2$ porque la función queda y = 1 * $x^2$
**>> y= fpote(1,0.5,x)**
y =
 1.00 1.4142 1.7321 2.00 2.2361

El resultado obtenido es igual a las raíces de los números 1 2 3 4 y 5, ya que la función queda y = 1 * $x^{0.5}$ o como dijimos antes $x^{1/2}$ que es lo mismo que la raíz cuadrada.

***<u>Función fzero</u>***

La función fzero calcula la (o las raíces de una ecuación).

**Es similar a roots** (pero roots sirve SOLO PARA HALLAR raíces de polinomios).

Se debe definir la función con **inline** y dar un valor próximo a la raíz para que a partir de ella Matlab busque la solución (veremos que es parecida a la Herramienta Buscar Objetivo de Excel)

Para hallar la raíz de la ecuación:

**sqrt(x) – 2 = 0** (una solución de la raíz cuadrada de 4 es +2)

Se debe:
Definir la función con inline (la llamamos **fecua**, pero puede tener otro nombre).
La función raíz cuadrada en Matlab es sqrt o puede en su lugar usarse $\mathbf{x^{(1/2)}}$ o $\mathbf{x^{0.5}}$ **.**
.
**>> fecua = inline( ‘sqrt(x) – 2’ )**

**o >>fecua =inline(‘ x.^0.5 - 3 ’)** ***(poner el punto luego de la x)***

Para saber dónde está la raíz, aproximadamente, se puede primero graficar la función sqrt(x) – 2 entre 0 y 5 y observar dónde la curva corta el eje x. En este ejemplo se observa perfectamente que es en el valor x=4

**>> x=0 : 0.2 : 5** % definimos x entre 0 y 5
**>> y= fecua(x)** % evaluamos los valores de y
**>> plot (x,y)** % graficamos
**>> grid on** % agregamos una grilla y observamos que la raíz es cercana a 4

**Para aplicar fzero**, se requieren 2 argumentos, el nombre de la función y un valor próximo a la raíz, por ejemplo 3 (esto lo observo en el gráfico:

**>> fzero(fecua,3)**
ans=
4

## 5.Operaciones con matrices en Matlab/FreeMat/Octave

***Sintéticamente aprenderemos a escribir matrices y vectores, y a usar comandos y funciones básicas. Los fundamentos matemáticos y demás elementos se ven en la materia correspondiente. El alumno podrá usar Matlab FreeMat/Octave para resolver y operar con matrices en otras materias sabiendo de antemano los comandos de esta herramienta.***

**Operaciones básicas con Matrices** (Recordar que la norma es ponerles nombres en mayúsculas).

**Se escriben sus elementos por filas**, separados por comas o espacios en blanco.

**Para cambiar de fila** se escribe un punto y coma (;) y se encierra todo entre corchetes. Si queremos escribir la matriz:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

**Para definir esta matriz** en Matlab se escribe el comando:

```
>>A = [ 1  2  ;  3  4]
A =
    1    2
    3    4
```

Si la matriz es cuadrada de 3x3 por ejemplo la matriz **H:**

**>> H = [ 1  2  3 ; 1  1  9 ; 1  0  2 ]**

Obtenemos:

```
H =
    1    2    3
    1    1    9
    1    0    2
```

**Transpuesta de una matriz. Operador apóstrofe**

***Trasponer una matriz es pasar las filas a columnas y viceversa.***

La matriz transpuesta de la matríz definida A, se obtiene escribiendo A con un apóstrofe

```
>>B = A'
B =
    1    3
    2    4
```

## Cómo escribir un vector.

Similar a una matriz, recordando que si es una sola fila se tiene un **vector fila**:

>> **vf = [ -1 , 2 , 6 ]**

Si el vector se ingresa con sus elementos separados por punto y coma (;) se dice que es un **vector columna**.

>> **vc = [ -1 ; 2 ; 6 ]** el resultado es un vector columna:

```
vc=
  -1
   2
   6
```

Otra forma de obtener un vector columna es ***transponer*** el vector fila original con el operador apóstrofe

**>>vc = vf ’**

***Importante: el uso del apóstrofe sirve también para transformar en tablas pares de valores x, y, como se explicó antes.***

## SOLUCIÓN DE UN SISTEMA DE ECUACIONES LINEALES (SEL) CON MATLAB/FREEMAT

Matlab/FreeMat/Octave permiten resolver Sistema de Ecuaciones Algebraicas Lineales. Se muestra para el caso de 2x2, pero sirve para órdenes mayores.

Tenemos el sistema de ecuaciones de 2x2:

$$1\,x_1 - 1\,x_2 = -1$$
$$1\,x_1 + 1\,x_2 = 7$$

El sistema algebraico puede escribirse en **forma matricial** como la operación entre la matriz A y el vector x cuyo resultado es el vector b:

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{matrix} = \\ = \end{matrix} \begin{bmatrix} -1 \\ 7 \end{bmatrix}$$

En Algebra lineal verá que el sistema se expresa como: **A . x = b** donde A es la matriz de coeficientes, x y b son vectores.

Para resolver este sistema con Matlab/FreeMat/Octave, basta definir la matriz A y el vector b (como vector columna), calcular la inversa de A y multiplicar por b para hallar las **x soluciones**

Otra forma es aplicarles el operador ***/*** (que significa **multiplicar por la inversa**).

**Paso 1:** Definir la matríz de coeficientes A:

**>> A = [ 1  -1 ; 1  1  ]**

Obtenemos:

```
A =
    1    -1
    1     1
```

**Paso 2:** Escribir el vector columna b (o de términos independientes del sistema), se lo ingresa con los elementos separados por punto y coma, para que sea un vector ‘columna’;

**>> b = [ -1 ; 7 ]**

```
b =
    1
    7
```

**Paso 3:**

**>> x = A \ b**

Y obtenemos los valores de x, solución del SEL:

```
  x=
      3
      4
```

Estos son los dos valores de **x,** que es entonces la solución del sistema de ecuaciones de 2x2, quiere decir que $x_1$ vale 3 y $x_2$ vale 4.

**Observar: se han escrito 3 comandos para resolver el SEL.**

**Nota:** Internamente en MATLAB esta operación se llama **‘premultiplicación por la inversa’** que se representa por el operador \ (barra invertida).
Ya aprenderá en Algebra lineal que la operación **** es igual a **>> x = inv(A)*b**

**Comprobación o Verificación:**
Si multiplicamos ahora A * x (las soluciones) debemos obtener de nuevo b. Definiremos bc para no ‘destruir` o perder los valores de b y luego poder observar que los elementos de b y bc son iguales:

**>> bc = A * x**

***<u>Gráfico del sistema de ecuaciones lineales de 2x2 (Comprobación Gráfica)</u>***

Cada una de las ecuaciones del SEL es una recta, escrita en forma General.

Si representamos gráficamente el sistema de 2x2 en un eje ***x, y*** donde llamamos: $\mathbf{x_1}$ a ***x*** y $\mathbf{x_2}$ a ***y****,* se observarán 2 rectas que se cortan en x=3 e y=4, o par ordenado (3, 4).

Teníamos el sistema:
1 $x_1$ **-** 1 $x_2$ = -1
1 $x_1$ + 1 $x_2$ = 7
Que para expresarlo con variables x - y, cambiamos los nombres de las variables $x_1$ por x y $x_2$ por y.
1 x - 1 y = -1
1 x + 1 y = 7
Entonces es lo mismo hallar la solución ($x_1$, $x_2$**)** o la (x , y ) solo hicimos un cambio de notación.
Si despejamos y de cada ecuación
En la 1ª. **y = 1 x + 1**
En la 2ª **y = -1 x + 7**
Obtenemos **las ecuaciones paramétricas en x de las dos rectas**, donde en cada una el primer coeficiente es la pendiente y el término independiente (1 y 7) es la ordenada al origen.
Recordemos que el sistema de 2x2 se representa por dos rectas en el plano, uno para cada ecuación.
**La recta 1 es y = 1 x + 1**
**La recta 2 es y = -1 x + 7**
***Ahora bien, las rectas son polinomios de grado 1***
Entonces definimos los polinomios que representan cada recta en Matlab como un vector de sus coeficientes:

```
>> recta1=[1 1]
recta1 =
    1    1
>> recta2=[-1 7]
recta2 =
   -1    7
```

***recta1*** y ***recta2*** son dos vectores que definen los polinomios de grado 1.

Con ellos calculamos las series de y para graficar estas rectas entre x=0 y x=5 (se elige este intervalo pues ya sabemos que $x_1$, nuestra $x_1$ del sistema planteado, da como solución 3 o sea que las rectas se cortan en x=3).

Primero debemos definir los valores de x

**>> x = 0 : 0.5: 5 % ( o sea x= 0 0.50 1.00 1.50 ….. 5.00)**

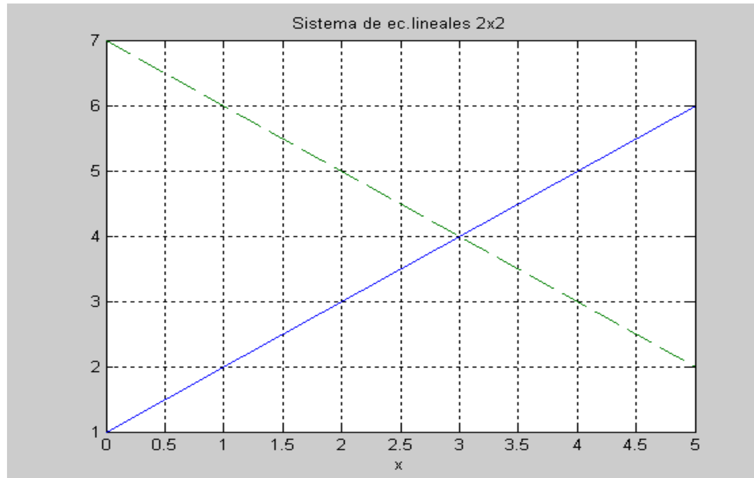
Evaluamos yrecta1 y luego yrecta2 con la función polyval:

**>>yrecta1 = polyval(recta1 , x)**
**>>yrecta2 = polyval(recta2 , x)**

Y graficamos los valores en la misma figura:
**>> plot(x , yrecta1 , x , yrecta2 ,'--')** *% el parámetro '--' es para que se trace con guiones*

**Nota**: luego del nombre de los valores y, se pueden agregar modificadores para la línea a trazar, si colocamos **‘-‘** la línea será con guiones, si se pone **‘o’** solo se marcan puntos, si se coloca **‘r’** la línea es color rojo, etc.

***Consultar en el Help el comando plot***




Y finalmente agregamos una grilla, el título para los ejes y un título del gráfico.

**>> grid on**

**>> xlabel(‘x)**

**>> ylabel('y')**

**>> title('Sistema de ec.lineales 2x2')**

Observamos que se cortan en x=3 e y=4, que son las soluciones del sistema
La recta 1 (ecuación 1) es la recta de línea llena, pendiente positiva y la recta 2 (ecuación 2) es la de líneas de guiones. Observar las pendientes + y-.