Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Física

FI1A2 – Sistemas Newtonianos

Guía Básica de Matlab

Por: Cristián Cruz D.

Santiago, 29 de Julio de 2007

Índice

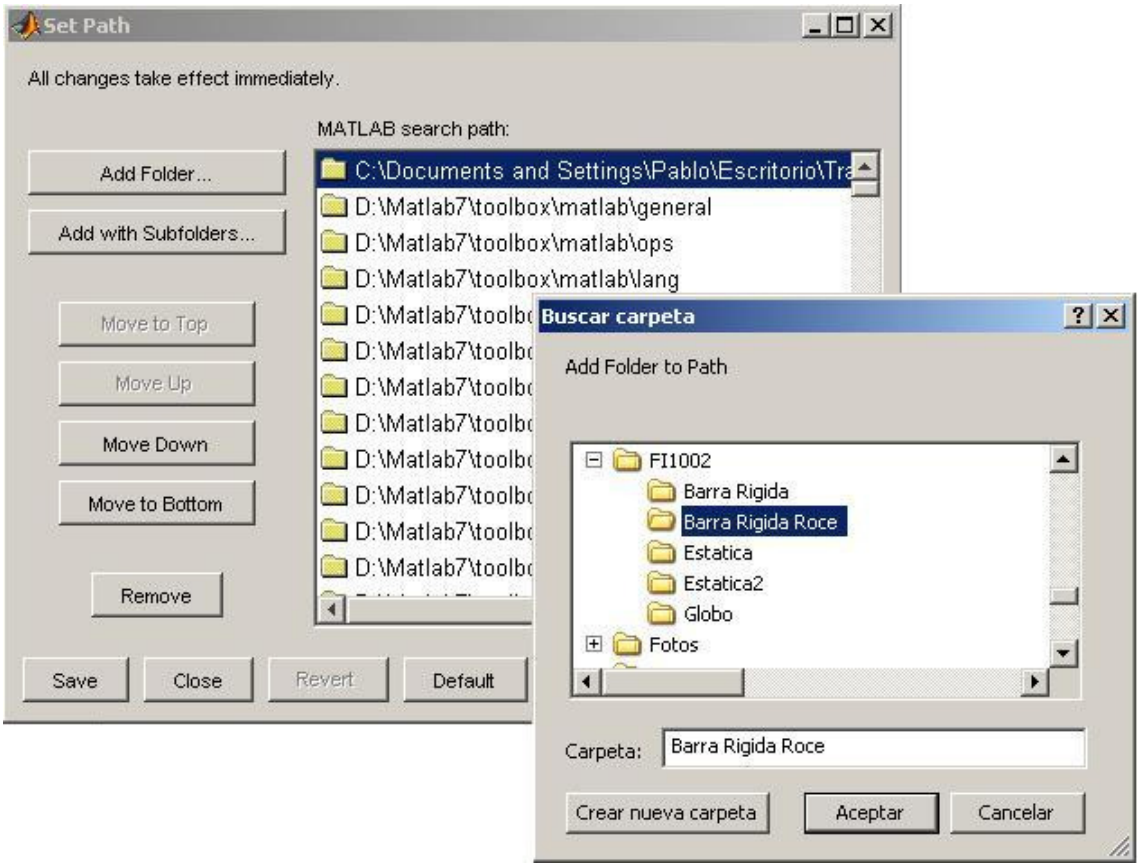
1. Introducción – *Matlab* : Matrix Laboratory

Como su nombre lo indica, *Matlab* es un software que se especializa en el trabajo con matrices. Es un lenguaje de programación que permite realizar operaciones matriciales en forma simple y práctica, cosa que en otros lenguajes (como JAVA) resulta bastante engorroso.

El objetivo de esta guía es explicar los comandos básicos para comenzar a programar en *Matlab*.

2. Directorios

Antes de comenzar, debemos indicarle al programa en qué directorio vamos a trabajar. Para esto vamos a *File* > *Set Path* y hacemos clic sobre el botón 'Add Folder'. Una vez que hayamos seleccionado la carpeta de trabajo, cerramos la ventana haciendo clic en 'Close'.



NOTA: Al cerrar la ventana el programa preguntará si desea guardarse la carpeta que agregamos como directorio para futuras sesiones. Se recomienda NO guardar, de modo de evitar que futuros alcances de nombres en funciones impidan un correcto desempeño del programa.

3. Definición de Variables y Operatoria Básica

Al contrario de lo que sucede en JAVA, para definir una variable en *Matlab* simplemente la declaramos de la siguiente forma:

```
x =15
```

A lo que el programa responderá desplegando el valor de x en pantalla:

```
>> x=15

x =

    15
```

Para evitar que se despliegue en pantalla un resultado se agrega un punto y coma al final de la operación:

```
x =15;
```

Como pueden observar, no es necesario declarar el tipo de datos que contiene la variable x. A menos que se indique lo contrario, *Matlab* trabaja por defecto con **arreglos**. En el ejemplo, x queda definido como una matriz de 1x1 cuyo único elemento es el número 15.

3.1 Declaración de Matrices:

Los elementos de cada fila se separan por comas o espacios, para terminar con una fila y empezar la siguiente se pone un punto coma.

Ejemplos:

$A = [1,2,3;4,5,6;7,8,9]$ $\quad$ $B = [1,2,3]$ $\quad$ $C = [1;4;9]$

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \qquad C = \begin{bmatrix} 1 \\ 4 \\ 9 \end{bmatrix}$$

También se pueden crear vectores en forma rápida utilizando V=inicio:paso:fin

$$V = 1:0.2:2 \quad genera \quad V = [1 \quad 1.2 \quad 1.4 \quad 1.6 \quad 1.8 \quad 2]$$

o crear vectores equiespaciados utilizando linspace(inicio, fin, Nº puntos)

$$V = linspace(1,2,6) \quad genera \quad V = [1 \quad 1.2 \quad 1.4 \quad 1.6 \quad 1.8 \quad 2]$$

3.2 Indexación de Elementos de una Matriz:

Si definimos A como una matriz de m x n, podemos acceder a sus elementos a través de los siguientes comandos:

A(i,j) : Muestra el elemento de la fila i y columna j.
A(i,:) : Muestra la fila i.
A(:,j) : Muestra la columna j.
A([1 3],j) : Muestra los elementos de las filas 1 y 3 de la columna j.
A(i,[1 4]) : Muestra los elementos de las columnas 1 y 4 de la columna i.

También se puede asignar valores a elementos específicos de una matriz, combinando los comandos de declaración e indexación. Por ejemplo:

A(1,2)=17 : Guarda el número 17 en la fila 1, columna 2.
A(i,:)=17 : Guarda el número 17 en todos los elementos de la fila i.
A(:,j)=17 : Guarda el número 17 en todos los elementos de la columna j.

3.3 Matrices Especiales

En ciertas ocasiones se necesita definir matrices especiales, como identidades, matrices nulas, unitarias, diagonales, tridiagonales, etc.

A' : Matriz Transpuesta de A
eye(n) : Crea una matriz identidad de n x n.
eye(m,n) : Crea una matriz de ceros de m x n con unos en si 'diagonal'.

diag(V): Crea una matriz diagonal con los elementos del vector V.

diag(V,n): Crea una matriz 'diagonal desplazada' en n con los elementos del vector V. Si n>0 la diagonal se desplaza hacia la derecha, si n<0 a la izquierda.

diag(A): Crea un vector con la diagonal de la matriz A.

ones(m,n): Crea una matriz de unos de m x n.

zeros(m,n): Crea una matriz de ceros de m x n.

rand(m,n): Crea una matriz de m x n de valores aleatorios.

Existen muchas más formas para definir matrices, y las que antes fueron mencionadas poseen más funciones. Para conocer más acerca de éstas utilicen la ayuda de *Matlab*, que es MUY buena.

3.4 Operaciones con Matrices:

A* B : Multiplicación matricial de A y B

A .* B : Multiplicación de los términos de A por los de B (conmutativa)

A / B : Multiplicación matricial de A por la inversa de B.

A \ B : Multiplicación matricial de B por la inversa de A.

A . / B : División de los términos de A por los de B.

A + B : Suma de los términos de A con los de B

A - B : Resta de los términos de A con los de B

Si U y V son vectores (matrices de una fila):

dot(U,V) : Producto punto de U con V ($\sum u_i \cdot v_i$).

cross(U,V) : Producto cruz de U con V.

4. M-Files

En *Matlab* se programa en M-Files, que son archivos de texto con una secuencia de instrucciones que luego se ejecutan en el programa. Para crear uno nuevo vamos a File > New > M-File.

4.1 Creación de Funciones

Al igual que en JAVA, en *Matlab* podemos crear funciones que ejecuten determinadas secuencias, las cuales pueden ser llamadas desde otro M-File. Para crear una nueva función debemos declararla como tal:

```
function [a,b] = nombre(c,d)
```

Donde a y b son las variables de salida (lo que entrega) y c y d son las variables de entrada (lo que recibe para poder ser ejecutada).
Ejemplo:

```
function [s,r] = sumaResta(a,b)
s=a+b;
r=a-b;
```

Guardamos el archivo en el directorio definido al comienzo y vamos a *Command Window* donde escribimos:

```
>> [a,b]=sumaResta(3,23)
a =
    26
b =
   -20
```

Al igual que en JAVA, es posible incluir comentarios en los M-Files. Para esto anteponemos % a lo que se quiera comentar. Si anotamos comentarios justo después del nombre de la función, esto se mostrará como ayuda si escribimos el comando *help nombre de la función* en el *command window*.

Siguiendo con el ejemplo anterior:

```
function [s,r] = sumaResta(a,b)
%sumaResta(a,b) retorna un vector con la suma de a y b en su
%primer componente y la resta de ambos en el segundo.
s=a+b;      %Suma
r=a-b;      %Resta
```

En *Command Window:*

```
>> help sumaResta
 sumaResta retorna un vector con la suma de a y b en su primer componente
y la resta de ambos en el segundo.
```

4.2 Operadores Lógicos

a) For

Genera ciclos con incrementos de una variable, por ejemplo:

```
for i=1:10
    a(i)=2*i;
end
```

crea el vector $a = [2 \quad 4 \quad 6 \quad \cdots \quad 20]$

b) If

La secuencia de instrucciones bajo el If se ejecutará sólo si se cumple la condición especificada, por ejemplo:

```
for i=1:10
    a(i)=2*i;
    if a(i)>=10
        b(i)=-a(i);
    end
end
```

Crea los vectores $a = [2 \quad 4 \quad 6 \quad \cdots \quad 20]$ y $b = [-10 \quad -12 \quad -14 \quad \cdots \quad -20]$

También existen los condicionales elseIf y else, que se tienen la misma función que en JAVA.

A continuación se muestra una lista con los operadores relacionales más utilizados:

Operador	Significado
>	Mayor que
<	Menor que
>=	Mayor o igual
<=	Menor o igual
&	Y (and)
|	O (or)

4.3 Gráficos

Matlab permite graficar los resultados obtenidos. Para esto utilizamos el comando plot. Además, existen una serie de funciones que aportan información adicional al gráfico:

plot(vector x, vector y, opciones):

Grafica los pares ordenados (vector x (i), vector y (i)). Por defecto *Matlab* une los puntos que se grafican, pero esto se puede deshabilitar utilizando las opciones del comando plot. Para mayor información sobre éstas consulten la ayuda del programa. Las opciones van desde cambiar los colores hasta el tipo de figura que se desea para marcar los puntos.

xlabel('nombre'): Da el título al eje coordenado x.
ylabel('nombre'): Da el título al eje coordenado y.
title('nombre'): Da el título al gráfico.
grid on : Hace visible la rejilla del gráfico.
hold on : Mantiene el gráfico anterior mientras se hace uno nuevo
figure : Crea una nueva ventana de gráfico
legend('nombre'): Identifica el gráfico en pantalla (etiqueta el gráfico)

Ejemplos:

i) Graficaremos el movimiento oscilatorio amortiguado que tiene una estructura de un grado de libertad (Esto es una versión muy simplificada de cómo oscila un edificio), cuya ecuación es:

$$x(t) = A \cdot \exp(-\beta \cdot \omega \cdot t) \cdot sen(\omega \cdot t)$$

donde

A [m] : Amplitud

β : Coeficiente de amortiguamiento

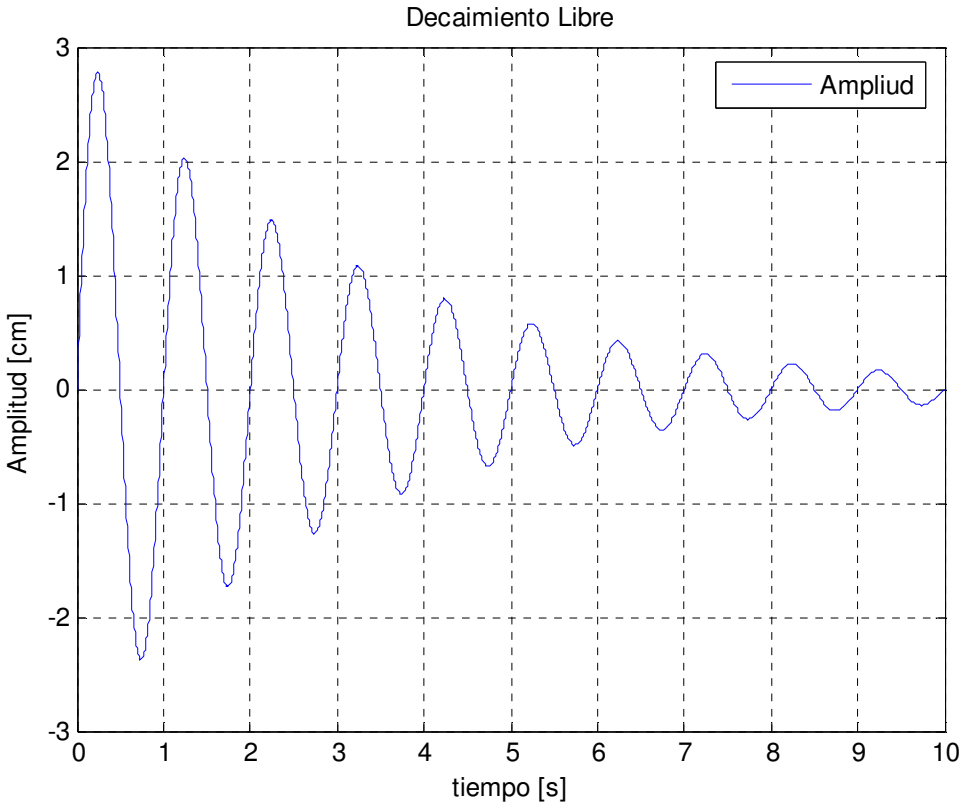
ω [1/s]: Frecuencia natural de oscilación de la estructura

t [s] : Tiempo

Crearemos un M-File que resuelva el problema:

```
%Definimos el vector de tiempo:
t=0:1/200:10;

%Definimos los parámetros de la ecuación:
w=2*pi;     %Frecuencia
b=0.05;     %Amortiguamiento
A=3;        %Amplitud

%Calculamos la ecuación de movimiento:
y=A*exp(-b*w*t).*sin(w*t);      %va con .* ya que la exponencial y el
                                %seno son vectores que evalúan la serie
                                %de tiempo

%Graficamos:
plot(t,y)
title('Decaimiento Libre')
xlabel('tiempo [s]')
ylabel('Amplitud [cm]')
legend('Ampliad')
grid on
```

Guardamos y corremos el programa, resultando el siguiente gráfico:



Si alguien se confundió con cómo se definió la función, un código alternativo puede ser utilizar un ciclo iterativo:

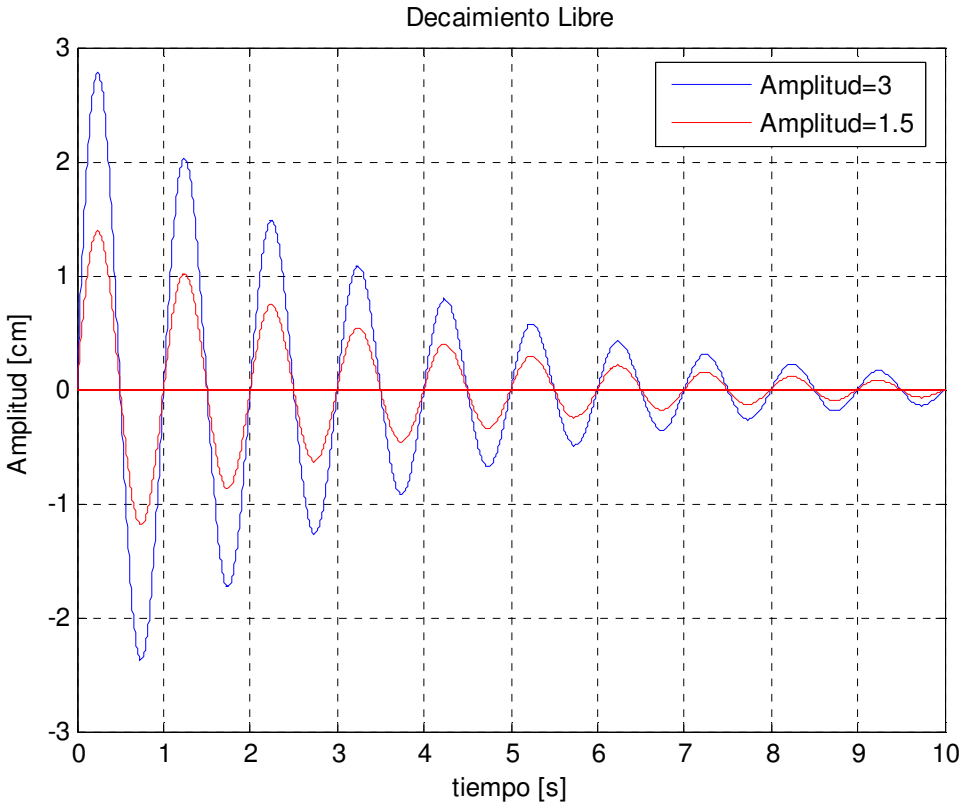
```
y=zeros(length(t));        %Damos dimensiones al vector creando un
                           %vector de ceros. Esto hace mucho más rápido
                           %el programa.

%Calculamos la ecuación de movimiento:
for i=1:length(t)
    y(i)=A*exp(-b*w*t(i))*sin(w*t(i));
end
```

ii) Grafiquemos ahora dos movimientos con distinta amplitud, en un mismo gráfico:

```
%Definimos el vector de tiempo:
t=0:1/200:10;

%Definimos los parámetros de la ecuación:
w=2*pi;     %Frecuencia
b=0.05;     %Amortiguamiento
A=3;        %Amplitud uno
B=1.5;       %Amplitud dos

%Calculamos la ecuación de movimiento:
y1=A*exp(-b*w*t).*sin(w*t);     %va con .* ya que la exponencial y el
                                %seno son vectores que evalúan la serie
                                %de tiempo

%Calculamos la segunda ecuación:
y2=zeros(length(t));        %Damos dimensiones al vector creando un
                            %vector de ceros. Esto hace mucho más rápido
                            %el programa.

%Calculamos la ecuación de movimiento:
for i=1:length(t)
    y2(i)=B*exp(-b*w*t(i))*sin(w*t(i));
end

%Graficamos:
plot(t,y1)
title('Decaimiento Libre')
xlabel('tiempo [s]')
ylabel('Amplitud [cm]')
grid on
```

```
hold on
plot(t,y2,'Red')
legend('Amplitud=3','Amplitud=1.5');
```

Resultando:

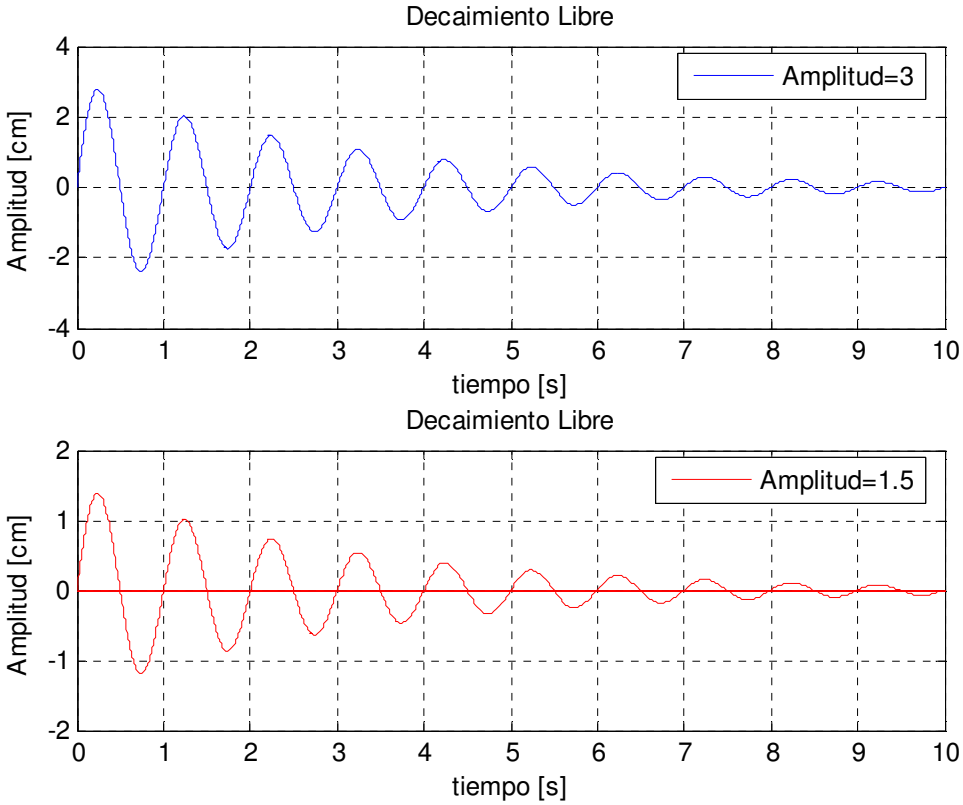


Probemos ahora la opción subplot para graficar. Si reemplazamos el código que grafica por:

```
%Graficamos:
subplot(2,1,1)
plot(t,y1)
title('Decaimiento Libre')
xlabel('tiempo [s]')
ylabel('Amplitud [cm]')
legend('Amplitud=3')
grid on
subplot(2,1,2)
```

```
plot(t,y2,'Red')
title('Decaimiento Libre')
xlabel('tiempo [s]')
ylabel('Amplitud [cm]')
legend('Amplitud=1.5')
grid on
```

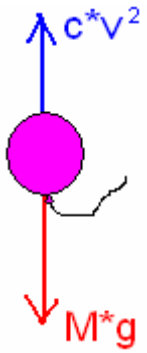
Tenemos:



5. Integración de Ecuaciones Diferenciales

Para poder resolver una ecuación de movimiento de Newton es necesario un método de integración numérica. Existen muchas formas de integrar una ecuación de movimiento; los métodos de Verlet y de Runge Kutta son ejemplos de métodos de integración. Para más detalles sobre los métodos matemáticos consulten la unidad uno de la guía del curso.

A continuación integraremos la ecuación de movimiento de la caída libre de un globo con roce viscoso cuadrático con el aire:



Del diagrama de cuerpo libre obtenemos las fuerzas del sistema. Igualando la suma de fuerzas a la masa por la aceleración del sistema se tiene:

$$Mg - c\dot{x}(t)^2 = M \cdot \ddot{x}(t)$$

Que es igual a resolver las ecuaciones:

$$Mg - cv^2 = M \cdot \dot{v}$$
$$\dot{x}(t) = v(t)$$

Utilizaremos el método runge kutta[1] de segundo orden (RK2) para resolver esta ecuación. El método dice que si se tiene una ecuación diferencial de la forma

$$y' = f(t, y)$$

Entonces se puede resolver numéricamente definiendo las variables auxiliares:

$$k_1 = h \cdot f(t_n, y_n)$$
$$k_2 = h \cdot f(t_n + \frac{h}{2}, y_n + \frac{1}{2}k_1)$$

Y que la solución iterativa será:

$$y_{n+1} = y_n + k_2$$

[1] P. Cordero, *Suplementos de Mecánica*

No ahondaremos en los fundamentos matemáticos del método ya que no es el tema de la guía.

Aplicando el método a nuestro problema tenemos:

$$\dot{v} = g - \frac{c}{M} v^2(t) \quad \Rightarrow \quad f(t,v) = g - \frac{c}{M} v^2(t)$$

$$k_1 = \Delta t \cdot \left[g - \frac{c}{M} v^2{}_n \right]$$

$$k_2 = \Delta t \cdot \left[g - \frac{c}{M} \cdot \left(v_n + \frac{1}{2} \cdot \Delta t \cdot \left\{ g - \frac{c}{M} v^2{}_n \right\} \right)^2 \right]$$

$$v_{n+1} = v_n + \Delta t \cdot \left[g - \frac{c}{M} \cdot \left(v_n + \frac{1}{2} \cdot \Delta t \cdot \left\{ g - \frac{c}{M} v^2{}_n \right\} \right)^2 \right]$$

$$con \quad v_0 = 0$$

Finalmente

$$x_n = \Delta t \cdot v_n$$

Programemos en *Matlab* lo anterior. Primero que todo, debemos crear una función que integre la ecuación de movimiento aplicando RK2.

Creamos un nuevo M-File, definimos la función y agregamos los comentarios para el help:

```
function [x,v,t] = caidaTurbulenta(m,c,v0,x0,dt,tf)

%caidaTurbulenta(m,c,v0,x0,dt,tf) resuelve la ecuación de movimiento
%de una caída libre con roce turbulento con el aire. Retorna la posición
%x, la velocidad v, y la serie de tiempo utilizada, como vectores en los
%que Xi=X(ti).
%m:     Masa del cuerpo
%c:     Coeficiente de roce viscoso
%x0:    Posicion inicial
%v0:    Velocidad inicial
%dt:    Paso de tiempo
%tf:    Tiempo final
```

Como verán, los comentarios iniciales definen la función y sus parámetros de entrada, de modo que sea simple y fácil entender cómo utilizarla.

A continuación definimos los parámetros base que necesita la función para resolver la ecuación de movimiento. Y luego aplicamos RK2 en un ciclo iterativo:

```
%Parametros base
g=9.8;

%Generamos el vector de tiempo:
t=(0:dt:tf);

%Definimos los vectores de resultados:
x=zeros(1,length(t));
x(1)=x0;    %Condiciones iniciales para posición
v=zeros(1,length(t));
v(1)=v0;    %Condiciones iniciales para velocidad

%RESOLUCIÓN POR RUNGE KUTTA 2:
for i=1:length(t)-1
    k1=dt*(g-c/m*v(i)^2);
    k2=dt*(g-c/m*(v(i)+0.5*k1)^2);
    v(i+1)=v(i)+k2;
    x(i+1)=x(i)+v(i)*dt;
end
```

Con esto ya tenemos una función que resuelve la ecuación diferencial. Ahora debemos crear un programa que la utilice de forma interactiva con el usuario. Para esto guardamos caidaTurbulenta y creamos un nuevo M-File.

En este nuevo programa, primero limpiamos la pantalla y cerramos los gráficos que estén abiertos:

```
%Limpiamos la pantalla:
close all
clc
```

Luego le pedimos al usuario que ingrese los datos del problema y calculamos la solución con la función que creamos anteriormente:

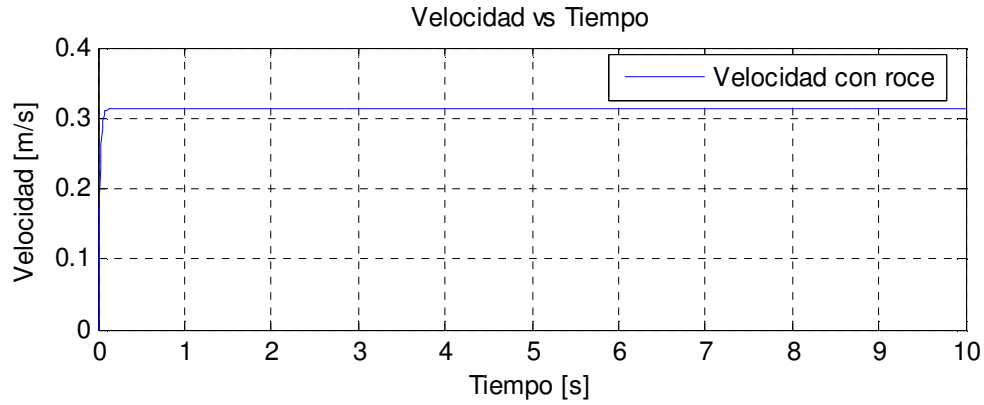
```
%Parametros base para el programa:
disp('INGRESE DATOS PARA LA INTEGRACIÓN: ');
disp(' ');
dt=input('Ingrese paso de tiempo [s] (recomendado 0.01): ');
tf=input('Ingrese tiempo final [s]: ');
disp(' ')

disp('INGRESE DATOS DEL PROBLEMA: ');
disp(' ')
h0=input('Ingrese altura de la caida [m]: ');
m=input('Ingrese masa globo [kg]: ');
c=input('Ingrese coeficiente de roce viscoso [kg/m]:');
x0=0;
v0=input('Ingrese velocidad inicial [m/s]:');
disp(' ')

%INICIO CALCULOS:
disp('Calculando...')
disp(' ')
[x,v,t] = caidaTurbulenta(m,c,v0,x0,dt,tf);
```

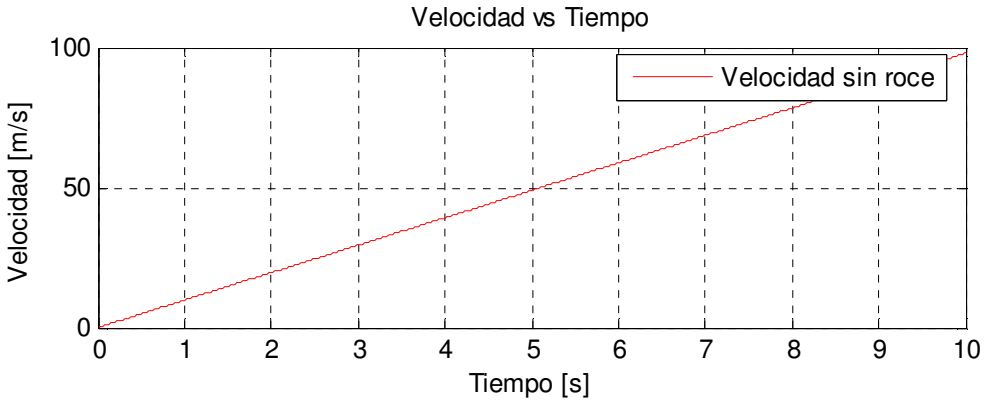
Con esto ya tenemos los resultados para los datos del usuario, a continuación graficaremos estos resultados y compararemos la caída con roce a una caída sin roce con el aire (es decir c=0).

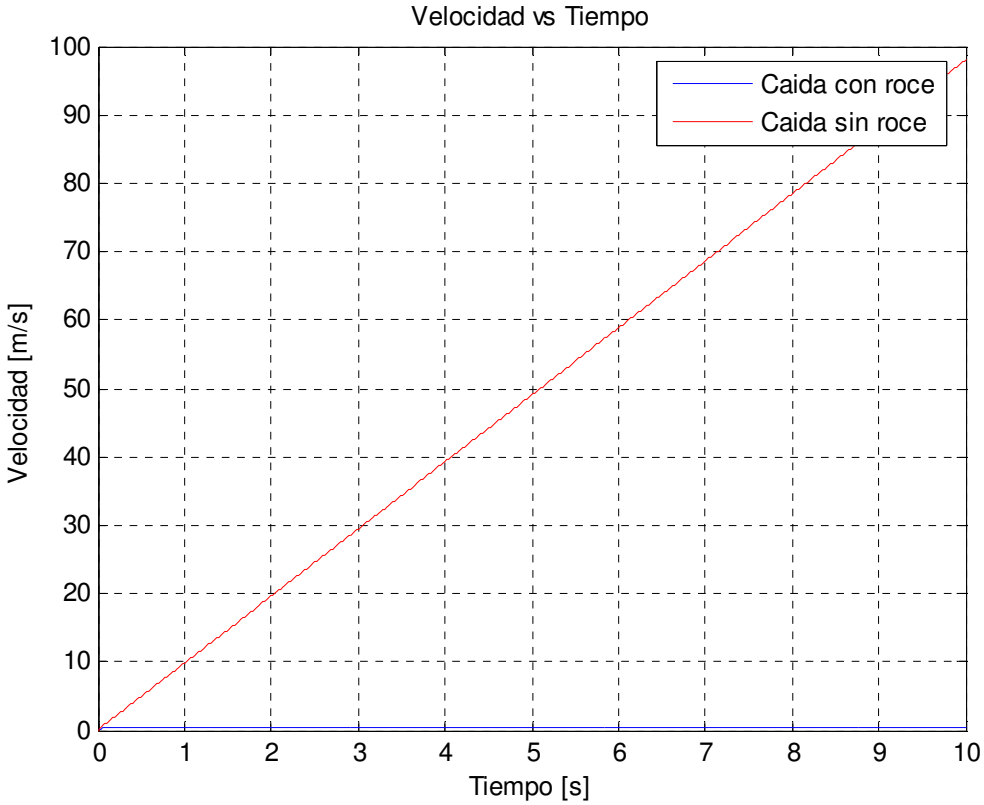
```
%Calculamos la caida sin roce:
[xs,vs,ts] = caidaTurbulenta(m,0,v0,x0,dt,tf);

%Calculamos h(t):altura
h=h0-x;
hs=h0-xs;

%Reajustamos los datos despues de que el globo toco el suelo
for i=1:length(h)
    if h(i)<0
        h(i)=0;                %Truncamos a cero valores negativos
    end
    if hs(i)<0
        hs(i)=0;               %Truncamos a cero valores negativos
    end
end

figure
%Graficamos velocidades por separado, misma ventana:
subplot(2,1,1)
plot(t,v,'blue')
title('Velocidad vs Tiempo')
xlabel('Tiempo [s]')
ylabel('Velocidad [m/s]')
legend('Velocidad con roce')
grid on
subplot(2,1,2)
plot(t,vs,'red')
title('Velocidad vs Tiempo')
xlabel('Tiempo [s]')
ylabel('Velocidad [m/s]')
legend('Velocidad sin roce')
grid on
```

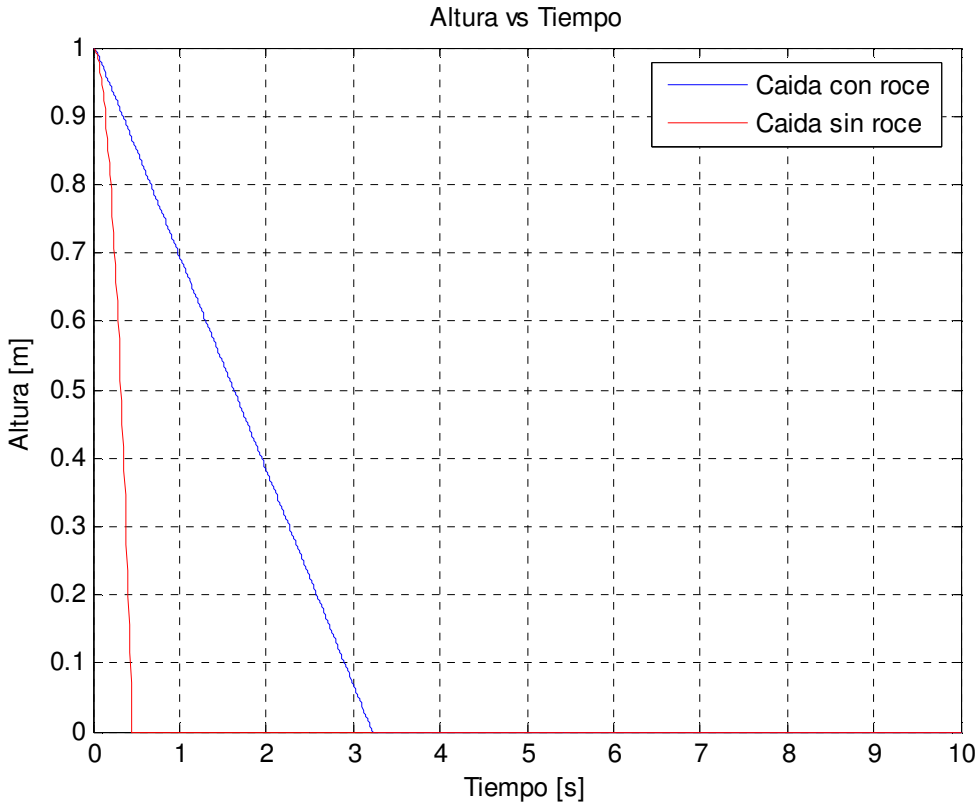
```
figure
%Graficamos Velocidades Juntas
plot(t,v,'blue')
title('Velocidad vs Tiempo')
xlabel('Tiempo [s]')
ylabel('Velocidad [m/s]')
grid on
hold on
plot(t,vs,'red')
legend('Caida con roce','Caida sin roce')

figure
%Graficamos caidas juntas
plot(t,h,'blue')
title('Altura vs Tiempo')
xlabel('Tiempo [s]')
ylabel('Altura [m]')
grid on
hold on
plot(t,hs,'red')
legend('Caida con roce','Caida sin roce')

%Terminamos
disp('Listo')
```

Corremos el programa y obtenemos los siguientes tres gráficos:


Velocidad vs Tiempo
Velocidad con roce
Velocidad [m/s]
Tiempo [s]
0
0.1
0.2
0.3
0.4
0
1
2
3
4
5
6
7
8
9
10


Velocidad vs Tiempo
Velocidad sin roce
Velocidad [m/s]
Tiempo [s]
0
50
100
0
1
2
3
4
5
6
7
8
9
10


Velocidad vs Tiempo
Caida con roce
Caida sin roce
Velocidad [m/s]
Tiempo [s]
0
10
20
30
40
50
60
70
80
90
100
0
1
2
3
4
5
6
7
8
9
10



Queda como ejercicio propuesto realizar este mismo programa utilizando el método de Verlet para resolver la ecuación de movimiento.

6. Acerca de la Ayuda de *Matlab*

Matlab cuenta con una excelente plataforma de ayuda al usuario. Es fundamental utilizarla ya que es imposible poner en una guía todo lo que se puede hacer con este programa. Alguien que utiliza bien *Matlab* no es aquel que se sabe todas las funciones de memoria, sino quien sabe usar de forma eficiente la ayuda.

Para llamar a la ayuda vamos a Help > MATLAB Help o presionamos F1. Les recomiendo que utilicen la pestaña index o search para buscar temas en la ayuda. Si quieren ver detalles de una función, pongan ahí el nombre de la función. Si quieren hacer algo, pero no saben si existe la función que lo hace, busquen por las 'palabras clave' de lo que quieren hacer. Es muy probable que encuentren lo que están buscando.

A continuación hay una lista de funciones que les pueden ser útiles en su vida estudiantil, busquen para qué sirven en la ayuda.

cla

clc

cumsum

cumtrapz

disp

eig

fft

getframe

guide

input

length

max

min

movie2avi

nargin

nargout

plot

plot3

roots

size

sort

sparse

sum

Busquen también las funciones mencionadas anteriormente en la guía, ya que tienen muchas más opciones que las que acá fueron explicadas.