UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL HAEDO

MATLAB

APUNTE GUIA

MECANICA ORBITAL

INGENIERÍA AERONÁUTICA/AEROESPACIAL

Ing. Anibal Cesar Vettorel

Ing. Carlos Mario Carbonel

Indice

Indice 2

Definiciones iniciales 3

- Current Folder (Carpeta Actual) 3
- Workspace (Área de Trabajo) 3
- Command Windows (Ventana de Comandos) 4
- Ventana de Variables 4
- Script 5

Funciones 5

- Ejercicio 1: Función con varias salidas 6
- Ejercicio 2: Función en archivo de script 7
- Ejercicio 3: Funciones añadidas 8
- Ejercicio 4: Caída libre 9
- Ejercicio 5: Tiro parabólico 15

DEFINICIONES INICIALES

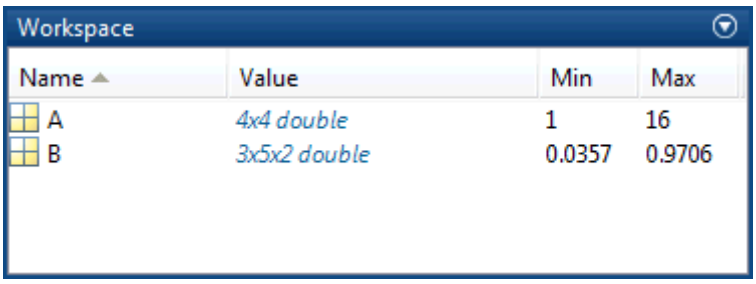
Current Folder (Carpeta Actual)

Nos muestra los archivos de la carpeta donde MATLAB se encuentra ubicado actualmente. Quiere decir que todo lo que haga MATLAB, desde el llamado de funciones o códigos script, MATLAB va a buscar primero en esa carpeta para ver si encuentra lo que él necesita.

Si no puede visualizarla, utilice el comando filebrowser

Workspace (Área de Trabajo)

Se visualizan todas las variables creadas, su tipo de variable, el valor, el minimo, máximo entre otras informaciones útiles.



Para ver el contenido del área de trabajo, utilice whos.

Las variables del área de trabajo no se mantienen al salir de MATLAB. Guarde los datos para usarlos posteriormente mediante el comando save.

El área de trabajo queda guardada en la carpeta de trabajo actual, en un archivo comprimido con extensión .mat. Estos archivos se denominan "archivos MAT".

Para borrar todas las variables del área de trabajo, use el comando clear.

Para restaurar datos de un archivo MAT en el área de trabajo, utilice load.

Enlace:
https://la.mathworks.com/help/matlab/learn_matlab/workspace.html?s_tid=srchtitle_site_search_1_workspace

Command Windows (Ventana de Comandos)

Es la ventana principal de MATLAB, donde podrás ingresar todos los cálculos numéricos como si fuera una calculadora. Además aqui podrás visualizar los resultados de la operaciones que hagas con el Software.

La ventana de comandos le permite ingresar declaraciones individuales en la línea de comando , indicada por el mensaje (>>). A medida que ingresa declaraciones, la ventana de comandos muestra los resultados.

Recuerda que puedes crear variables simplemente asignándolas con el signo “=” y si no deseas ver el resultado debes finalizar con “;” (punto y coma). Para crear comentarios basta con colocar el comando “%” al comienzo y podemos escribir un comentário que MATLAB no lo va a interpretar

Acción	Cómo realizar la acción
Ingrese varias declaraciones en varias líneas antes de ejecutar cualquiera de las declaraciones.	Ingrese las múltiples declaraciones en la línea de comando, presionando **Shift+Enter** entre declaraciones. Esta combinación de teclas no es necesaria cuando ingresa una declaración de palabra clave emparejada en varias líneas, como `for` y `end`.
Borre una declaración de la línea de comando sin ejecutarla.	Presione la tecla Esc (**Esc**).
Recordar declaraciones anteriores.	Presione la tecla de flecha hacia arriba ↑. Se abre la ventana Historial de comandos y muestra un registro de declaraciones anteriores. Para recuperar una declaración específica, escriba cualquier parte de la declaración y luego presione la tecla de flecha hacia arriba. Por ejemplo, para recuperar el comando `b = 2`, escriba `b` y luego presione la tecla de flecha hacia arriba.
Borre la ventana de comandos.	Llame a la `clc`función. Para borrar la ventana de comandos sin eliminar ningún texto, llame a la `home`función. Al llamar a la `home`función, se mueve el cursor a la esquina superior izquierda de la ventana de comandos y se desplaza todo el texto visible fuera de la vista, dando la apariencia de limpiar la pantalla sin eliminar ningún texto.
Evalúe una declaración que ya se encuentra en la ventana de comandos.	Seleccione una declaración, haga clic derecho y luego seleccione **Evaluar selección** .
Ejecute solo una parte del código que se encuentra actualmente en la línea de comando.	Seleccione el código en la línea de comando y presione **Enter** . `>> disp("hello"), disp("world")`

Enlace: https://la.mathworks.com/help/matlab/ref/commandwindow.html

Ventana de Variables

Cuando creamos una variable, sabemos que ella estará en el wordspace, si damos doble click sobre esa variable creada, se abrirá la conocida ventana de variables, en esta parte podremos modificar el valor de la variable, o incluso podemos adicionar más valores para convertirla en un vector o matriz.

Enlace: https://la.mathworks.com/help/matlab/matlab_env/create-and-edit-variables.html?searchHighlight=ventana%20de%20variables&s_tid=srchtitle_support_results_1_ventana%20de%20variables

Script

Normalmente en nuestro entorno de trabajo de MATLAB, no vamos a estar colocando cada calculo en la ventana de comandos, porque generalmente vamos a tener que recalcular algunas funciones o si deseamos abrir el software al día siguiente vamos a tener que re ejecutar todas las lineas que habíamos hecho. Para eso creamos los Script en MATLAB, para ir colocando todas las lineas de código dentro de un archivo de texto. De esa forma podremos tener una mejor visión de lo que estamos programando y además nos facilita el poder reabrir el archivo otro día o abrirlo en otra computadora. Por eso comenzaremos a crear nuestros ficheros que se conocen como SCRIPT los cuales tienen una exención .m.

A diferencia de los archivos de texto plano (.m), los live scripts (con extensión. mlx) permiten no solo escribir código, sino también integrar texto con formato, ecuaciones matemáticas, imágenes y comentarios explicativos dentro del mismo documento. Esto significa que, en lugar de tener el código por un lado y los resultados por otro, en un live script podemos visualizar de manera conjunta tanto las instrucciones como sus resultados, ya que estos se muestran directamente debajo de cada bloque de código. Además, ofrecen herramientas de formato similares a las de un procesador de texto, lo que facilita la documentación y comprensión del desarrollo realizado.

Enlace:
https://la.mathworks.com/help/matlab/learn_matlab/scripts.html?searchHighlight=script&s_tid=srchtitle_support_results_1_script

FUNCIONES

Una función es una colección de operaciones cuyo objetivo es realizar una tarea particular. En general una función puede recibir uno o varios valores de entrada (*input1, input2, …*) y producir como salida uno o varios valores (*output1, output2, …*). También es posible que la función no reciba ninguna entrada o no produzca ninguna salida.

Sintaxis:

function [y1,...,yN] = myfun(x1,...,xM)

Declara una función llamada *myfun* que acepta entradas *x1,...,xM* y devuelve salidas *y1,...,yN*.

Los nombres de función válidos comienzan con un carácter alfabético y pueden contener letras, números o guiones bajos. Esta instrucción de declaración debe ser la primera línea ejecutable de la función.

Puede guardar la función:

- En un archivo de función que solo contenga definiciones de funciones. El nombre del archivo debe coincidir con el nombre de la primera función del archivo.
- En un archivo de script que contenga comandos y definiciones de funciones. Las funciones deben estar al final del archivo. Los archivos de script no pueden tener el mismo nombre que una función del archivo. Las funciones son compatibles con los scripts a partir de la versión R2016b.

Los archivos pueden incluir varias funciones locales o funciones anidadas. Para facilitar la lectura, utilice la palabra clave *end* para indicar el final de cada función en un archivo. La palabra clave *end* es obligatoria cuando:

- Una función del archivo contiene una función anidada.
- La función es una función local dentro de un archivo de función, y cualquier función local del archivo utiliza la palabra clave *end*.

La función es una función local dentro de un archivo de script.

Ejercicio 1: Función con varias salidas

Defina una función en un archivo llamado *stat.m* que devuelva la media y la desviación estándar de un vector de entrada.

```
function [m,s] = stat(x)
    n = length(x);
    m = sum(x)/n;
    s = sqrt(sum((x-m).^2/n));
end
```

Llame a la función desde la línea de comandos.

stat.m

/MATLAB Drive/Mecánica Orbital 2026/Práctica 1 - Introducción al Matlab/stat.m

```
function [m,s] = stat(x)
    n = length(x);
    m = sum(x)/n;
    s = sqrt(sum((x-m).^2/n));
end
```

Command Window

```
>> values = [12.7, 45.4, 98.9, 26.6, 53.1]; [ave,stdev] = stat(values)

ave =

   47.3400


stdev =

   29.4124

>>
```

Ejercicio 2: Función en archivo de script

Defina un script en un archivo llamado *integrationScript.m* que calcule el valor del integrando en $2\pi^3$ y calcule el área bajo la curva de 0 a π. Incluya una función local que defina el integrando $y = sin(x)^3$

```
% Compute the value of the integrand at 2*pi/3.
x = 2*pi/3;
y = myIntegrand(x)

% Compute the area under the curve from 0 to pi.
xmin = 0;
xmax = pi;
f = @myIntegrand;
a = integral(f,xmin,xmax)

function y = myIntegrand(x)
      y = sin(x).^3;
end
```

integrationScript.m

/MATLAB Drive/Mecánica Orbital 2026/Práctica 1 - Introducción al Matlab/integrationScript.m

```
% Compute the value of the integrand at 2*pi/3.
x = 2*pi/3;
y = myIntegrand(x)

% Compute the area under the curve from 0 to pi.
xmin = 0;
xmax = pi;
f = @myIntegrand;
a = integral(f,xmin,xmax)

function y = myIntegrand(x)
    y = sin(x).^3;
end
```

Command Window

```
>> integrationScript

y =

    0.6495


a =

    1.3333

>>
```

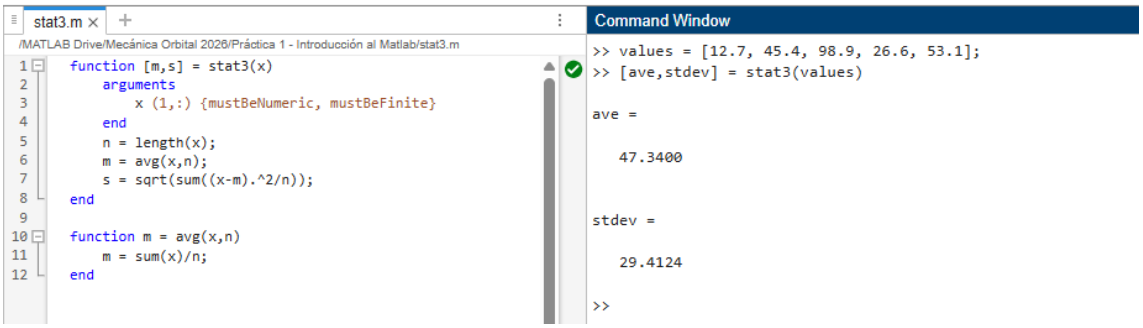
Ejercicio 3: Funciones añadidas

Defina dos funciones en un archivo llamado *stat3.m*, donde la primera función llame a la segunda. Se debe restringir la entrada a un vector numérico que no contenga elementos *Inf* ni *NaN*. Esta función utiliza la palabra clave *arguments*, que es válida para la versión R2019b de MATLAB® y posteriores.

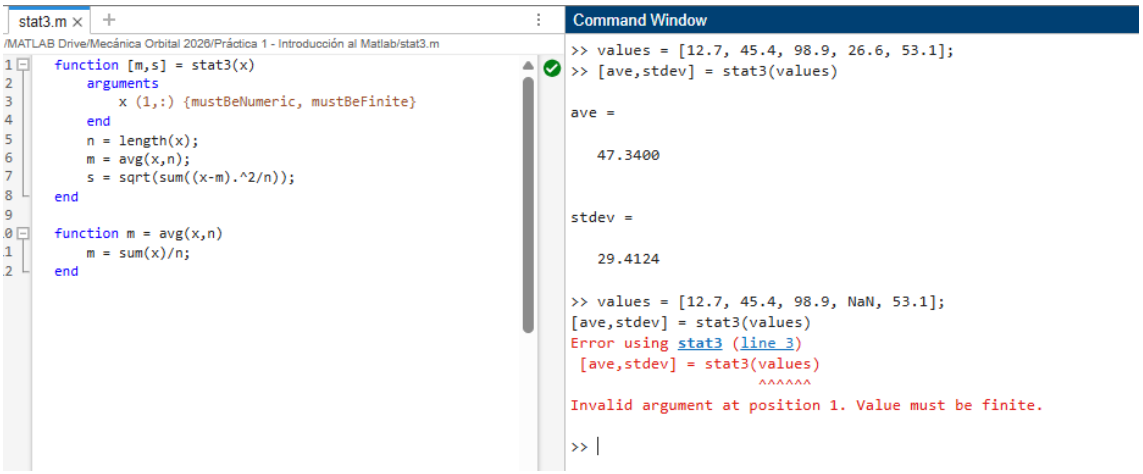
```
function [m,s] = stat3(x)
      arguments
            x (1,:) {mustBeNumeric, mustBeFinite}
      end
      n = length(x);
      m = avg(x,n);
      s = sqrt(sum((x-m).^2/n));
end

function m = avg(x,n)
      m = sum(x)/n;
end
```

En el bloque de código *arguments, (1,:)* indica que *x* debe ser un vector. Las funciones de validación, *{mustBeNumeric, mustBeFinite},* restringen los elementos de *x* a valores numéricos que no son *Inf* ni *NaN*.



Llamar a la función con un vector que contiene un elemento *NaN* infringe la declaración del argumento de entrada. Esta infracción provoca un error de la función de validación *mustBeFinite*.

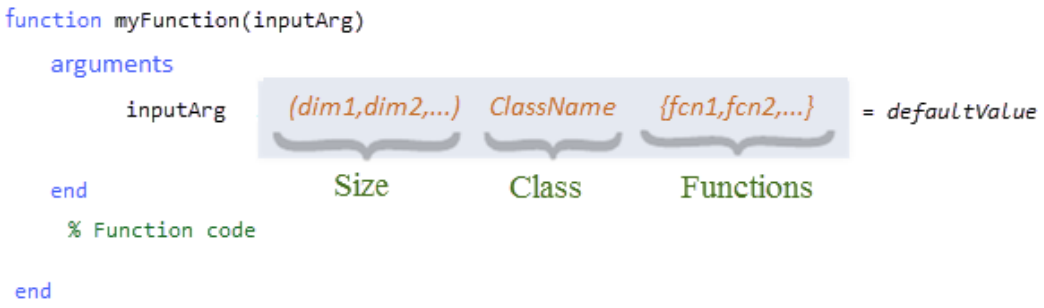


El bloque:

```
arguments
      x (1,:) {mustBeNumeric, mustBeFinite}
end
```

se lee como “x debe ser un vector fila numérico, con valores finitos.”

La sintaxis es la siguiente:



https://la.mathworks.com/help/matlab/matlab_prog/function-argument-validation-1.html

Ejercicio 4: Caída libre

Se deja caer una piedra desde el reposo, despreciando la resistencia del aire, en un campo gravitatorio uniforme con aceleración g=$9{,}8\ m/s^2$.

Se pide:

1. Modelar el movimiento de la piedra durante los primeros **10 segundos** utilizando MATLAB.

2. Calcular la **posición y velocidad** del objeto en cada instante de tiempo, considerando un paso de integración de 1 segundo .

3. Determinar cuántos metros recorre la piedra durante el **primer segundo de caída**.

4. Representar gráficamente la posición y velocidad en función del tiempo

En definitva esto es un M.R.U.V. (movimiento rectilineo uniforme variado), cuyas formulas son:

$$v = v_0 + a \cdot t$$

$$x = x_0 + v_0 t + \frac{1}{2} a t^2$$

$$a = \text{cte}$$

Donde:

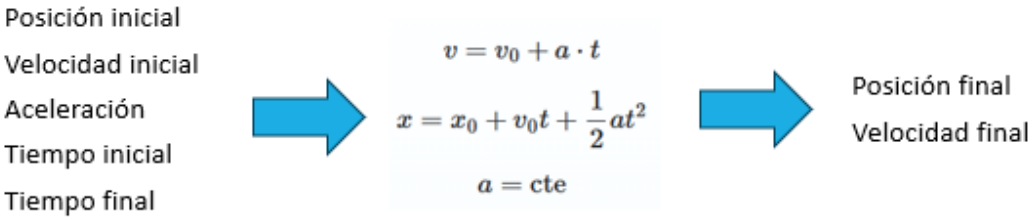
- *x*, *x0*: La ***posición*** del cuerpo en un instante dado (*x*) y en el instante inicial (*x0*). Su unidad en el Sistema Internacional (S.I.) es el metro (m)

- *v*,*v0*: La ***velocidad*** del cuerpo en un instante dado (*v*) y en el instante inicial (*v0*). Su unidad en el Sistema Internacional es el metro por segundo (m/s)

- *a*: La ***aceleración*** del cuerpo. Permanece constante y con un valor distinto de 0. Su unidad en el Sistema Internacional es el metro por segundo al cuadrado (m/s2)

- *t*: El intervalo de ***tiempo*** estudiado. Su unidad en el Sistema Internacional es el segundo (*s*)

Para armar la función, hay que identificar cuales son nuestros datos de entrada, cuales son las ecuaciones que debemos resolver en nuestra función, y las salidas de datos.



Con esto en mente, podemos armar la siguiente función:

```
%% ---------------------------------------------------------------------
% UTN - FRH
% FUNCIÓN DE MOVIMIENTO RECTILÍNEO UNIFORMEMENTE VARIADO (MRUV)
% ---------------------------------------------------------------------

function [x, v] = MRUV(xi, vi, a, ti, tf)

%% Notas aclaratorias
% Esta función calcula la posición y la velocidad de una partícula
% bajo movimiento rectilíneo uniformemente variado (MRUV).
%
% Se basa en las ecuaciones cinemáticas clásicas para aceleración constante.
%
% Entradas:
% xi - Posición inicial [m]
% vi - Velocidad inicial [m/s]
% a  - Aceleración [m/s^2]
% ti - Tiempo inicial [s]
% tf - Tiempo final [s]
%
% Salidas:
% x  - Posición en el instante final [m]
% v  - Velocidad en el instante final [m/s]
%
% Funciones utilizadas: Ninguna

%% Validación de datos de entrada
arguments
    xi {mustBeNumeric, mustBeFinite}
    vi {mustBeNumeric, mustBeFinite}
    a  {mustBeNumeric, mustBeFinite}
    ti {mustBeNumeric, mustBeFinite, mustBeNonnegative}
    tf {mustBeNumeric, mustBeFinite, mustBeNonnegative}
end

%% Cálculo del movimiento (MRUV)
dt = tf - ti;

x = xi + vi*dt + 0.5*a*dt^2;
v = vi + a*dt;

end
```

Es una buena práctica colocar como comentarios, cuales son los elementos de entrada, y si representan magnitudes físicas, sus unidades. Lo mismo sucede con las salidas.

Ahora resta desarrollar un script principal que utilice la función MRUV para calcular la posición y velocidad del objeto en cada instante de tiempo y almacene los resultados en vectores.

es una buena práctica estructurar el programa en **bloques o secciones**. Esto permite ordenar el desarrollo, facilitar la lectura del código y simplificar su ejecución por partes.

Cada bloque se define con %%, lo que permite ejecutar secciones de manera independiente. Esta forma de trabajo resulta muy útil para depurar errores, analizar resultados parciales y mejorar la comprensión del algoritmo.

Una recomendación fundamental es comenzar siempre con una **limpieza del entorno de trabajo**, para evitar que variables o gráficos anteriores interfieran con la ejecución actual. Esto se logra con las siguientes instrucciones:

- `close all`: cierra todas las ventanas de figuras abiertas
- `clear`: elimina las variables almacenadas en memoria
- `clc`: limpia la ventana de comandos

A partir de allí, el programa puede organizarse en bloques típicos como:

- **Ingreso de datos:** definición de constantes, condiciones iniciales y parámetros
- **Inicialización:** creación de vectores y asignación de valores iniciales
- **Cálculo o iteración:** desarrollo del modelo o algoritmo
- **Resultados:** visualización de datos y generación de gráficos

Este tipo de estructura no solo mejora la claridad del código, sino que también refleja una forma ordenada de pensar y resolver problemas, lo cual es fundamental en programación y en el análisis de sistemas físicos.

```
%% ----------------------------------------------------------------
% UTN - FRH
% SIMULACIÓN DE MOVIMIENTO RECTILÍNEO UNIFORMEMENTE VARIADO (MRUV)
% ----------------------------------------------------------------

%% Notas aclaratorias
% Este script tiene como objetivo simular el movimiento de una partícula
% bajo aceleración constante (MRUV), calculando su posición y velocidad
```

```
% en función del tiempo.
%
% El cálculo se realiza de manera iterativa utilizando un modelo
% implementado en una función externa.
%
% La estructura del programa está dividida en bloques para facilitar su
% lectura, depuración y reutilización.
%
% Variables utilizadas:
% g   - Aceleración de la gravedad [m/s^2]
% x0  - Posición inicial [m]
% v0  - Velocidad inicial [m/s]
% ti  - Tiempo inicial [s]
% tf  - Tiempo final [s]
% dt  - Paso de integración temporal [s]
% t   - Vector de tiempo [s]
% x   - Vector de posición [m]
% v   - Vector de velocidad [m/s]
%
% Funciones utilizadas:
% MRUV - Función externa que calcula la posición y velocidad del sistema
%        entre dos instantes de tiempo, a partir de las condiciones
%        actuales del movimiento.

%% Limpieza del entorno de trabajo
close all   % Cierra todas las figuras abiertas
clear       % Elimina todas las variables del Workspace
clc         % Limpia la ventana de comandos

%% Datos de entrada
% Constante física
g = 9.8; % Aceleración de la gravedad [m/s^2]

% Condiciones iniciales
x0 = 0;   % Posición inicial [m]
v0 = 0;   % Velocidad inicial [m/s]

% Parámetros de simulación
ti = 0;   % Tiempo inicial [s]
tf = 10;  % Tiempo final [s]
dt = 1;   % Paso de tiempo [s]

%% Inicialización de variables
% Vector de tiempo
t = ti:dt:tf;

% Preasignación de memoria
x = zeros(size(t));
v = zeros(size(t));

% Condiciones iniciales
x(1) = x0;
v(1) = v0;

%% Iteración del modelo
% Resolución numérica del movimiento en forma iterativa
for i = 1:length(t)-1
    [x(i+1), v(i+1)] = MRUV(x(i), v(i), g, t(i), t(i+1));
end
```

```
%% Resultados
% Presentación de resultados numéricos
disp('   t(s)      v(m/s)      x(m)')
disp([t' v' x'])

%% Gráficos del sistema
% Se agrupan los resultados en una sola figura para facilitar el análisis
% comparativo entre posición, velocidad y aceleración en función del tiempo

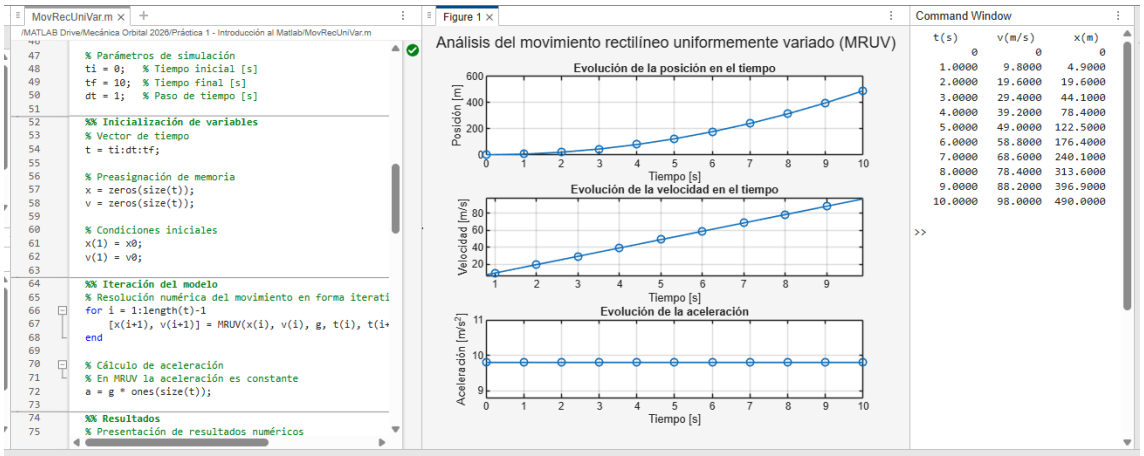
figure;

subplot(3,1,1)
plot(t, x, 'o-', 'LineWidth', 1.2)
xlabel('Tiempo [s]')
ylabel('Posición [m]')
title('Evolución de la posición en el tiempo')
grid on
box on

subplot(3,1,2)
plot(t, v, 'o-', 'LineWidth', 1.2)
xlabel('Tiempo [s]')
ylabel('Velocidad [m/s]')
title('Evolución de la velocidad en el tiempo')
grid on
box on

subplot(3,1,3)
plot(t, a, 'o-', 'LineWidth', 1.2)
xlabel('Tiempo [s]')
ylabel('Aceleración [m/s^2]')
title('Evolución de la aceleración')
grid on
box on

sgtitle('Análisis del movimiento rectilíneo uniformemente variado (MRUV)')
```



Ejercicio 5: Tiro parabólico

Se solicita implementar una función llamada **proyectil**, que permita calcular la posición y velocidad de un objeto en función del tiempo, considerando las siguientes ecuaciones:

- $x = x_0 + v_{x0} \cdot t$
- $v_x = v_{x0}$
- $y = y_0 + v_{y0} \cdot t + \frac{1}{2} a_y t^2$
- $v_y = v_{y0} + a_y t$

La función deberá recibir como entradas:

- Posición inicial horizontal y vertical (x0, y0)
- Velocidades iniciales (vx0, vy0)
- Aceleración en el eje vertical (g)
- Tiempo inicial (ti)
- Tiempo final (tf)
- Paso de tiempo (dt)

Y deberá devolver:

- Vector de tiempo t
- Posición horizontal x
- Posición vertical y
- Velocidad horizontal vx
- Velocidad vertical vy

Además de la función, se deberá desarrollar un **script principal** que:

- Utilice la función proyectil
- Genere los resultados para el sistema en cada instante de tiempo
- Almacene las variables en vectores
- Compare dos casos:
    - Con gravedad
    - Sin gravedad
- Grafique ambas trayectorias en una misma figura