Práctica 1. Introducción a Matlab.

Todos los scripts y funciones se copiarán en un Word de resultados, atendiendo a la Sección y número del ejercicio en el que nos encontremos.
Para aquellos scripts o funciones que sean llamados desde consola de comandos se copiará, además, la llamada realizada, así como el resultado generado, tanto numérico como figuras.

MATLAB diferencia entre las mayúsculas y minúsculas. No es lo mismo escribir EJEMPLO que Ejemplo o ejemplo.

SECCIÓN 1. Aprendiendo comandos, tipos de variables, funciones y operadores en Matlab.

Las siguientes órdenes pueden serte útiles para el desarrollo de esta práctica:

- ***help***: Te ofrece ayuda sobre una función, incluyendo algún ejemplo sobre su uso y otras funciones relacionadas. Por ejemplo, teclea >>*help clc*
- ***lookfor***: Lista todas las funciones en cuya ayuda aparece una palabra clave. Por ejemplo, si quisiéramos saber qué funciones tiene Matlab para manipular directorios podríamos teclear >>*lookfor directory* y a continuación teclear >>*help pwd*, >>*help dir*, etc. para saber cómo funciona cualquier función que nos interese.

En la siguiente tabla se indican algunas órdenes útiles en MATLAB.

ORDEN	***SIGNIFICADO DE LA ORDEN***
HELP	● **Ayuda sobre órdenes y funciones internas de Matlab.** ● **Ayuda sobre nuestras propias funciones .m**
WHAT	**Da una lista de funciones .m en el directorio especificado (*Por ejemplo: >> what a:\ejemplos*)**
CD	● **Cambia al directorio padre o directorio anterior *(>> cd ..)*** ● **Cambia al directorio que se especifique *(>> cd a:\ejemplos*)**
DIR	**Da una lista del contenido del directorio en el que me encuentro**
PWD	***¿Qué hace este comando?***

1. Haciendo uso del comando **HELP** puedes saber las funciones de los comandos **diary, who, clear, hold, format y edit.** ¿Qué diferencia existe entre los comando **edit** y **type?**

Las siguientes órdenes son interesantes para poder conocer los operadores definidos en Matlab. Algunos de ellos los vas a utilizar frecuentemente.

**(a) help **

(b) help arith.

(c) help slash.

(d) help relop.

2. Haz lo siguiente usando las líneas de comando de Matlab (indicadas por >>)

Operación	***Ejemplo***	***Indica los resultados de las operaciones***
Suma +	>>format short >>a=7 >>b=2; %variable escalar **>>suma=a+b; %**El ; hace que no se vea el resultado pero sí se ejecuta la orden o sentencia. **>>suma**	Resultados de a, b y suma:
Resta -	>>resta=a-b; **>>resta**	Resultado resta:
Multiplicación *	**>>multiplicar=a*b**	Resultado multiplicar:
División /	**>>div1=a/b**	Resultado div1:
***División ***	**>>div2=a\b** **>>div3=b\a %¿Es igual que div1?**	Resultado div2: Resultado div3:
Potencia ^	**>>a=3;potencia=a^2**	Resultado potencia:
	>>a=9; **>>a^3**	Resultado: *¿Cómo se llama la variable a la que se le asigna el resultado?*
	>>ans **>>c=7;d=8;ans=1** **>>(c*d)/(ans+7)** **>>(c*d)/ans+7**	*¿Cuáles son los valores que va tomando la variable por defecto ans?*

(a) Ejecuta en la línea de comandos el comando **who**. ¿Qué aparece en pantalla?
(b) Ejecuta: **>> clear b.** Pregunta por el valor de **b**. ¿Qué sale en pantalla?
(c) Ejecuta el comando **who**. ¿Cuál es la diferencia que encuentras con la ejecución anterior?

3 (Ejercicio para hacer en casa). Busca el significado de las siguientes **funciones internas** de Matlab, pon un ejemplo y su resultado. Averigua cuál se puede escribir en minúsculas, mayúsculas o indistintamente. Normalmente en el segundo párrafo se especifica la sintaxis de la función. En general, en cualquier lenguaje de alto nivel las funciones trigonométricas deben tener el argumento o variable de entrada en radianes. *Matlab también dispone de funciones trigonométricas cuyo argumento se defina en grados. Búscalas.*

Función	***Ejemplo***	***Resultado ejemplo***
ABS		
SQRT		
RAND		
SIN		
SIND		
TAN		
ASIN		
SINH		
EXP		
LOG		
LOG10		
REM		
ROUND		
EPS		
PI		

4. Introducción a vectores. Como en todos los lenguajes de alto nivel las variables numéricas se clasifican básicamente en variables escalares (un número), variables vectoriales (vector) o variables matriciales. En ejemplos anteriores se ha visto la asignación numérica escalar, ahora vamos a ver una de las formas para asignar valores/elementos a un vector. Busca el significado de las órdenes ***LINSPACE*** usando el comando *help*, y ejecuta

```
>> x=linspace(0,2*pi,30)
```

Nota. Esta línea de comandos permite asignar valores numéricos a la variable llamada *x*. A diferencia de las asignaciones anteriores la variable *x* es un vector. Sin embargo, hay que aclarar que realmente Matlab trabaja con variables con estructura matricial, en donde p.e. *x* será una matriz de 1 fila y 30 columnas.

(a) Escribe el resultado de la asignación anterior.
(b) Sea la siguiente orden **>> y=sin(x)**

Escribe los valores numéricos de la variable **y** .

(c) Busca información sobre el comando o la orden **plot**. Ejecuta >>plot(x,y,x,y,´r*´). Como ves se ha generado una figura con los elementos de dos vectores.
(d) Para determinar el número de elementos de un vector usa la función interna **length**. Indica la orden que has ejecutado para saber el número de elementos de los vectores x e y.
(e) La siguiente ejecución es incorrecta: **>> z=x^2.** Modifícala para que tengamos el cuadrado de cada uno de los elementos de vector x. Recuerda el ejercicio 1.

SECCIÓN 2. Variables

Los ejercicios que aparecen en verde son ejercicios que podéis hacer para reforzar conocimientos.

1. Indica los resultados tras ejecutar las siguientes órdenes. Si hubiera errores, corrígelos. Fíjate en las órdenes que han dado lugar a matrices o variables matriciales

```
>> X=[ 1 2  9 4  5]
>> X=[1,2,9,4,5]
>> X=[ 1;2;9;4;5]
>>X'
>>X(3)
>>X(0)
>>X([1,3,5])
>>X(1:4)
>>X=20:-2:1
>>X(end)
>>longitud=length(X)
>>tamano =size(X)
>>[fil,col]=size(X)
>>a=[1:10,2:12,3:13]
>> a=[1 2 7
5 6 8
3 2 6]
>> b=[1 2 3]
>> c=[a;b]
>>d=[1,2,3;4,3,0;6,8,1]
>>e=[1 2 3;4 3 0;6 8 1]
>>f=e', g=b';
```

2. Crear un vector *x* de 4 componentes equi-espaciado entre los valores 6 y 7.

3. Sumar 1 al tercer elemento del vector x.

4. Crear un vector *y* de las mismas dimensiones que *x* con los primeros números impares.

5. Crea un vector v que contenga la tabla de multiplicar del 3, cuyo primer elemento sea cero y el último sea 30.

6. Crear un vector y con la diferencia entre sucesivos elementos de v.

7. Sumar 5 a los elementos del vector v con índice par.

8. Crear los vectores a=[0 6 8 3] y b=[-1 7 8*2 4]. (a) Crear una matriz A de tamaño 2x4 cuyas filas sean los vectores a y b y (b) Crear una matriz 4x2 cuyas columnas sean a y b.

9. Escribe un vector z que vaya del 0 al 1 con un espaciado de 0.1. Calcular un vector con los valores de la función interna *sin* aplicada a los elementos de z. Hazlo también para la función interna *exp*.

10. Crear un vector x con las fracciones 1, 1/2 , 1/3, 1/4, … 1/10.

11. Generar los primeros 10 números de la serie: 0, 1/2, 2/3, 3/4, …

12. Crear la serie $(-1)^n$ con n= 0, …, 10

13. Crear un vector con los números de la serie: $\frac{(-1)^n}{2n+1}$ desde n = 0, …, 100 . Usa la función interna *sum* para sumar todos los elementos del vector. Repetir el paso anterior con 10000 términos. ¿Cuál de los dos resultados está más próximo al valor de π? Nota: $\sum_{n=0}^{\infty}\frac{(-1)^n}{2n+1}=\frac{1}{1}-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\ldots=\frac{\pi}{4}$ se corresponde con la fórmula de Leibniz para el cálculo de π.

14. Crear una matriz *B* cuya primera fila corresponda al vector *a*, cuya segunda fila sea un vector de ceros, la tercera un vector de unos y la cuarta el contenido de *b. Ver ones y zeros.*

15. Definir un vector *k* cuyos elementos sean iguales a los dos primeros elementos de la diagonal de la matriz *B*. Utiliza la función interna *diag.*

16. Crea una submatriz *C* de tamaño 3x3 que sea el contenido de las tres primeras filas y columnas de la matriz *B*.

17. Crear dos matrices *A* y *B*:

$$A=\begin{pmatrix}1 & 2\\ 4 & -1\end{pmatrix}, B=\begin{pmatrix}4 & -2\\ -6 & 3\end{pmatrix}$$

(a) Resultados de las dos operaciones suma y resta entre matrices: *C = A+B* y *D=A-B.*
(b) Calcula los determinantes de ambas matrices y sus inversas.
(c) Resultados de: E=A*B y F=B*A.
(d) Empleando la operación elemento a elemento obtén la matriz G que sale de la siguiente expresión.

$$g_{ij}=a_{ij}-b_{ij}\cdot a_{ij}^{2/3}$$

Si observas algo extraño averigua el porqué.
(e) Tiene sentido realizar las siguientes operaciones *A/B* y *A\B*? Razona tu respuesta

18. Resolución de sistemas de ecuaciones. Sea el siguiente sistema de ecuaciones de la forma A·x=b, resolverlo mediante los operadores matriciales estudiados en clase

$$A=\begin{pmatrix}1 & 3 & 6\\ 8 & 9 & 1\\ 3 & 10 & 9\end{pmatrix}, b=\begin{pmatrix}1\\ 3\\ 4\end{pmatrix}$$

19. Veamos ahora el indexado lógico de matrices y vectores:
Sea el vector `x=[-1 0 2 4 -2 3 1 4 7 0 -3 -1];`. Si queremos asignar a cero los elementos de x mayores de cero hacemos en la línea de comandos **`x(x>0) = 0.`**

(a) Poner a -5 los elementos de x menores de 0
(b) Poner a 3 los elementos de x mayores de 1 y menores de 5:
(c) Crear un vector y con los elementos de x menores o iguales a 3:
(d) Sumar 2 a los elementos mayores de 5
(e) Crear un nuevo vector que tenga 1's en las posiciones de los elementos de x que sean mayores que la media (usar la función interna *mean*) y 0's en las de los menores que la media. NOTA: primero haz >>y=x %para evitar modificaciones del vector x y, por tanto, también de su media.
(f) Asignar a cero los elementos pares (usar función interna *rem)*.
(g) Cambiar el signo de los valores de x verificando: 2<=x(i) < 5.

SECCIÓN 3. Funciones y scripts.

1. Introducción a un script (leer en casa). Un script es un programa que construye el usuario de Matlab. Este programa se construye con el lenguaje de alto nivel de Matlab. Todas las asignaciones que se realicen en el script o/y en la ventana de comandos se comparten. Todo script tiene que tener un nombre y una extensión fija .m. Esta extensión indica que el script o fichero está escrito en el lenguaje de Matlab. El script se puede crear con un editor (p.e. *edit*) o con la pestaña *File* del menú superior del entorno de matlab. Cuando se haya terminado la edición grabarlo (*save as*) con nombre y extensión .m (la extensión normalmente se pone por defecto, es decir no hay que ponerla). Tras la grabación el programa estará en el directorio donde se ha grabado (en nuestro caso en d:\alumnos\work). Al ejecutarlo en la ventana de comandos solo se escribe el nombre del script. Fíjate en los detalles. Los apóstrofos están en la tecla del cierre de interrogación.

Ejemplo de script	**Ejecución de un script en la ventana de comandos**
% Script ejemplo_script.m disp('Cuánto vale A_script y B_vc en ejemplo_script') A_script=37.5 disp([B_vc]) disp('Abandono el script')	>>edit ejemplo_script.m >>B_vc=5555; >> ejemplo_script %ejecución de un script >>disp('Cuánto vale a_script en ventana de comandos') >>A_script

IMPORTANTE: Matlab diferencia entre las letras mayúsculas y minúsculas como ya se indicó. Luego, en el ejemplo anterior si se pregunta por **A_script** dará el valor asignado 37.5, mientras que si se pregunta por **a_script** nos dirá que esta variable no existe.

IMPORTANTE: Como en todo lenguaje de alto nivel hay palabras reservadas, p.e. **disp**., que no se pueden usar como nombres de variable para realizar asignaciones.

IMPORTANTE: El separador de la parte entera y de la parte decimal en los números reales es un punto no una coma.

IMPORTANTE: El directorio, por defecto, en donde se graba un script lo puedes saber ejecutando >>pwd o >>cd. Si quieres ir al *directorio padre* teclea >>cd … Si quieres volver al directorio en donde trabajabas teclea >>cd nombredeldirectorio. Si quieres trabajar con tu pen usando el comando cd indica a continuación el nombre con el que se identifica tu pen.

2. Introducción básica a funciones (leer en casa). Una función al igual que un script la tiene que crear un programador o usuario de Matlab. En Matlab hay dos tipos de funciones, función interna, como p.e. la función ya conocida ***sin,*** y función externa o creada por el usuario de Matlab. Centrándonos en esta última, una función tiene que tener un nombre y la extensión .m, se crea con un editor y se graba en un directorio. **CUIDADO**: La función se debe grabar con el mismo nombre que está indicado en la primera línea (la línea donde está la palabra reservada *function*). Se ejecuta en la ventana de comandos. Otro ejemplo de palabra reservada es **function**. Vas a realizar muchas funciones a lo largo del curso; a continuación sólo se muestra un ejemplo que debes hacer. Presta atención.

Ejemplo de función	**Ejecución de una función en la ventana de comandos**

function [c_fun,d_fun]=ejemplo_fun(a,b)
%ejemplo_fun.m
 	
disp('Cuánto vale a,b, c_fun y c_fun dentro de ejemplo_fun')
disp(a)
disp(b)
 	
c_fun=a+b;
d_fun=a*b;
 	
disp([c_fun d_fun]) % Fíjate en este disp.
disp('Abandono la función')
	>>a=10; b=5;
	>> [C_fun,D_fun]=ejemplo_fun(a,b)
	>> disp('Los valores numéricos de las variables c_fun y d_fun definidas dentro de la función se han asignado a las variables C_fun y D_fun en la ventana de comandos tras ejecutar la función ejemplo_fun en dicha ventana')
	>> disp([C_fun D_fun])
	>>% las variables a y b son las denominadas **variables o parámetros de entrada** a la función.
	>>% las variables C_fun y D_fun son las denominadas **variables o parámetros de salida,** obtenidos tras la ejecución de la función.
	>> disp('Sin embargo, en la ventana de comandos no conoce las variable c_fun y d_fun, ya que únicamente están definidas dentro de la función. La siguiente ejecución así lo dirá')

	>> disp([c_fun d_fun])

3. Escribe un script que calcule la superficie y volumen de un cilindro de radio 3 m y altura 5 m. La asignación a variables escalares hazlas fuera del script. A este script llámale sup_vol_script.m. Escribe el script y las órdenes en la ventana de comandos.

Script	***Líneas de ejecución del script en la ventana de comandos de Matlab***
	>> >>

4. Escribe una función que tenga como variables de entrada el radio y la altura, y como variables de salida el volumen y la superficie. Llama a esta función sup_vol_fun.m. Escribe la función y las órdenes.

Escribe la función	***Líneas de ejecución de la función en la ventana de comandos de Matlab***
	>> >>

5. Crear la función *suma_vectores* que tome como parámetros de entrada los vectores *a* y *b*, devolviendo un vector *c* que realice su suma. Hazlo también con un script, llámalo *suma_vectores_s.* ¿Qué diferencia aprecias entre *suma_vectores* y *suma_vectores_s*? ¿Las variables tienen el mismo ámbito? ¿La forma de invocarlos es la misma?

6. Escribir una función que reciba como argumento un valor *x* y un entero *n* y devuelva el valor de la función matemática: $f(x) = \frac{1}{1+|x|^n}$. Construye una figura que muestre cómo varía f(x) si n=3 y x varía entre -40 y 40. Pon etiquetas a los ejes usando los comandos xlabel e ylabel.

7. Crear una función llamada *alcance.m* que determine el alcance máximo horizontal (*xmax*), de un objeto lanzado con una velocidad inicial *vo* en el plano xy. Considera como variables de entrada el ángulo ϑ y la velocidad de origen *vo* del lanzamiento del objeto, ambas serán variables escalares. La variable de salida será el alcance máximo.

$$x_{max} = \frac{v_0^2 \cdot sen(2 \cdot \theta)}{g}$$

8. Haz lo mismo con un script, llámale alcance_s.m

9. Crea una función tiempo.m que determine el tiempo máximo del lanzamiento. Variables de entrada: ángulo y velocidad inicial. Variable de salida: tiempo máximo.

$$t_{max} = \frac{2 \cdot v_0 \cdot sen(\theta)}{g}$$

10. Crear una función llamada *alcance_tiempo.m* que llame a las funciones alcance.m y tiempo.m. Las variables de salida serán el alcance y el tiempo máximo.

11. Crear una función llamada alcance_vect.m con variable de salida una matriz que contenga todos los ángulos, velocidades iniciales, tiempos en alcanzar de nuevo el suelo y los alcances máximos de todas las trayectorias generadas. Las variables de entrada serán el ángulo y la velocidad.(a) Aplícalo a un ángulo de 45º y una velocidad de entre 40 y 60 m/s con un salto de 10 m/s. (b) Aplícalo a ángulos entre 10º y 90º con un salto de 5º y una velocidad de 40 m/s. (c) Aplícalo a una velocidad entre 20 y 40 m/s con un salto de 10 m/s y ángulos entre 10º y 30º con un salto de 10º.

SECCIÓN 4. Bucles y condicionales

Se tiene que usar bucles o/y condicionales en todos los programas de esta sección.

1. Escribe una función potencia.m que reciba como argumentos de entrada un número x y un entero n y devuelva el valor de la n-ésima potencia de x (x^n). El segundo argumento de entrada puede ser opcional. Si se omite debe tomarse como n=2 (devolviendo el cuadrado de x). Ver la función interna nargin. Usa la estructura if.

2. Crea una función que determine la suma de los inversos de los impares menores de 1000. Usa las funciones *tic* y *toc* para determinar el tiempo que se tarda. Usa la estructura for.

$$Serie = 1 + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \ldots + \frac{1}{999}$$

3. Los números de Fibonacci, F(n) son una serie de números enteros, inicializados con F(1)=1 y F(2)=2. Los demás términos se tienen como la suma de los dos anteriores: F(n) = F(n-1) + F(n-2). Escribe un script que genere los primeros 100 números de la serie de Fibonacci, almacenándolos en un vector F. Usa la estructura while.

4. Escribir las líneas de código para que dado un vector x, genere otro vector con el orden de los elementos invertido.

5. Implementar una función que tome como entrada un vector *x* y que devuelva la posición y el contenido del primer elemento negativo. Sintaxis de llamada: [elem, pos] = buscaelem(x). (a) Añadir una condición de parada si no hubiese ningún elemento negativo, que muestre por pantalla el mensaje: *No hay ningún elemento negativo.*

6. Implementar una función que dado un vector devuelva el menor valor del vector y la posición donde se encuentra sin usar la función *min* o *max*. Por ejemplo v=[1 2 3 4 5 6 -1 -3 -2] devolvería el valor de -3 y la posición 8.

7. Crear una función que dados los valores escalares de n, a_1, a_2 devuelva un vector de n componentes tal que $a_n = a_{n-1} - 3a_{n-2}$.

8. Crear un script que obtenga el valor medio y desviación media del vector *x* descrito como:

$$med = \frac{\sum_{i=1}^{n} x_i}{n}, desv = \sqrt{\frac{\sum_{i=1}^{n} (x_i - med)^2}{n}}$$

Compara los resultados con las funciones internas mean y std. Compara ahora el resultado de std con tu variable desv si la divides entre (n-1) en vez de n.

9. Implementar una función que tome como entrada una matriz cuadrada *A* y que devuelva la suma de todos los elementos de las 2 diagonales de dicha matriz. No utilizar la función *diag*().

10. Implementar una función que dadas dos matrices A y B definidas como $(a_{ij})_{mxn}$ y $(b_{ij})_{nxp}$, implemente la multiplicación de matrices C=A·B mediante bucles, definida como $(c_{ij})_{mxp}$ donde$c_{ij} = \sum_{n}^{r=1} a_{ir} \cdot b_{rj}$.

11. Crear una función llamada *alcance_bucle.m* que determine el alcance máximo horizontal y el tiempo máximo de un objeto lanzado con una velocidad inicial *vo* y un ángulo *ϑ, siendo ambos vectores de distinta longitud. Aplícalo* los ángulos de 10º a 90º saltando de 5º en 5º y una velocidad inicial del lanzamiento de 40 m/s a 60 m/s con un salto de 10 m/s.

SECCIÓN 5. Depurador (esta sección es opcional)

1. Utiliza la funcionalidad del depurador de Matlab estudiando el comportamiento del código de la función calcula_phi.m. Descarga del campus virtual calcula_phi.m y dibuja_valores.m. Ejecuta la *calcula_phi(0.01*)

a. Añade un punto de ruptura (*breakpoint*) en la línea 13.
b. Ejecuta una introducción pasa a paso (*Step*). ¿Cómo cambia el valor de phi y por qué?
c. ¿Cómo va evolucionando el valor de E y por qué?
d. Coloca un *breakpoint* en la línea 16: "dibuja_valores(iter,phi,'b*')"
   i. ¿Qué diferencia observas entre *Step*, *Step In* y *Step Out*?
   ii. ¿Y con *Continue*?

SECCIÓN 6. Representación gráfica

1. Escribe la función *dibujo_parabola.m* que muestre paso a paso en una figura la trayectoria del tiro parabólico para v_o 40 m/s y θ=45º y para v_o igual a 60 m/s y θ=60º. En cada instante se mostrará con un círculo rojo la posición (x,y). Llama a alguna de las funciones anteriores. Usa la función *plot y hold on*

$$x = v_0 \cdot cos(\theta) \cdot t \; y = v_0 \cdot sen(\theta) \cdot t - \frac{1}{2} g \cdot t^2$$

(a) Añade el nombre de cada eje con *xlabel*, *ylabel* y el título de la figura con *title*
(b) Emplea los comandos de la ventana que contiene el gráfico para cambiar el grosor y el color de las líneas de las trayectorias.

2. Usa únicamente una sola figura para representar en diferentes paneles (usa subplot) las siguientes funciones. En el panel 1, dibujar la función $y = e^{(-x^2)}$en el intervalo [-2 2] empleando 100 datos equiespaciados y en los paneles 3 y 5, respectivamente, las funciones $y = e^{\left(-\left(\frac{x}{2}\right)^2\right)}$ y $y = e^{(-(2x)^2)}$. Cada función debe ir en un color y en un estilo de línea diferente.

Añade sólo en la figura del panel 5 el comando grid on. En el panel 6 se representará la última función en escala logarítmica en uno de los ejes. En el panel 2 se representará la primera función empleando *stem*. En el panel 4 se representará la segunda función empleando *hist*. ¿Tiene sentido lo que observas?

3. Crear una función *dibujatriangulo* que tenga como argumento de entrada los 3 vértices de un triángulo y devuelva una figura con el triángulo dibujado.
4. Crear una función *dibujaMUC* que dibuje la trayectoria circular de una partícula conociendo su velocidad angular *ω* y el radio de giro *r*. Recuerda el significado del periodo de un movimiento circular.
Las ecuaciones cartesianas son:

$$x = r \cdot cos(\omega \cdot t)\ y = r \cdot sen(\omega \cdot t)$$

(a) Añadir a la trayectoria en cada punto el vector velocidad con el comando *quiver*, mediante las ecuacione s de la velocidad:

$$v_x = x' = -r \cdot \omega \cdot sen(\omega \cdot t)\ v_y = y' = r \cdot \omega \cdot cos(\omega \cdot t)$$

5 (Ejercicio complementario). Crear una función *dibujatrayectoria3D* que tenga como parámetros de entrada el radio *r* y la velocidad angular *ω* y dibuje la trayectoria en 3 dimensiones de una partícula que describe un MCU en el plano XY moviéndose en el eje z con una velocidad constate v_z.

6. Crear una función *dibujatiroparabolico* que represente el tiro parabólico en 3D sabiendo que la función tiene como entrada los ángulos ϕ y θ de acuerdo a las siguientes expresiones:

$$x = v_0 \cdot cos(\theta) \cdot cos(\varphi) \cdot t\ y = v_0 \cdot cos(\theta) \cdot sen(\varphi) \cdot t\ z = v_0 \cdot sen(\theta) \cdot t - \frac{1}{2} g \cdot t^2$$

A la hora de representar la función, emplea el comando *plot3* y *pause* para ir viendo cómo se dibuja la trayectoria parabólica. Calcula el alcance máximo en cada eje previamente y fija de antemano los valores de los ejes empleando los comandos *xlim* ,*ylim* y *zlim* respectivamente.

7. Usando el Comando *meshgrid*, crea una retícula cuadrada en el intervalo x=[-1 1] y=[-2 2] emplea para ello un paso de malla de valor 0.1. Crea una matriz de ceros del tamaño de las matrices que definen la retícula. Representa, empleando el comando *mesh*, la matriz de ceros creada sobre la retícula. Representa gráficamente la superficie $z = e^{-(x^2+y^2)}$. Dibuja las curvas de nivel usando el comando *contour*

SECCIÓN 7. Entrada y salida

1. Usando el comando fprintf escribe en la pantalla el número pi con 10 decimales. Mira previamente las opciones de formato de fprintf.

2. Usando fprintf escribe el número e con 2 decimales en una línea y con 8 decimales en la siguiente.

3. Sea la variable x = 174, usa fprintf para ver en una sola línea en pantalla el valor de x como:

- un entero (%d),
- un entero reservando 4 columnas (%4d)
- un entero, reservando 4 columnas y rellenando con 0's (%04d)
- un entero con signo (%+d)
- un número real con 2 decimales (%.2f)
- un número real en notación científica (%e)

4. Usa el comando input para asignar el valor 222 a una variable. Pregunta a continuación por el valor de dicha variable en la línea de comando de matlab.
5. Crear un programa que dado un número introducido por el usuario, devuelva por pantalla la tabla de multiplicar del número introducido.

6. Crear cinco vectores aleatorios denominados v1, v2, v3, v4 y v5, a continuación hay que guardar su contenido en un fichero usando el comando *save*. Haz lo siguiente:

(a) Los vectores v deben tener la misma dimensión, ejecuta >> save ale1.txt v* –ascii, donde el nombre del fichero es ale1.txt en este fichero se almacenará la información contenida en todas las variables que empiezan por v y su formato será legible por eso se usa la opción -ascii. Usa type para visualizar el contenido de este fichero. Recupera la información usando *load*, es decir ejecuta >> load ale1.txt y asigna, por ejemplo, a una variable vectorial a el contenido de la segunda fila de las columnas de las 2 a la 4 ejecutando >>a=ale1(2,[2:4]). Borra el fichero usando >>delete ale1.txt

(b) Los vectores v deben tener diferente longitud. Al usar load verás que no es posible utilizarlo. Ejecuta las siguientes órdenes >>save ale2 v* %no lo graba en formato legible ya que no usas la opción –ascii, el fichero sin extensión se llamará ale2 y v* indica, como sabes,que se grabaran en este fichero las variables que comienzan con v, >>clear v* % para borrar todas las variables que empiezan por v del workspace, >> who %para ver si existen variables que comienzan con v, >> dir ale* % para ver los ficheros que comienzan con ale, ¿qué ves?, >>load ale %recupera información junto con asignación, >>v1 %verás que recupera la información asignada a v1.

(c) Volvemos a generar cinco vectores de longitud diferente y grabamos su información legible en un fichero ale3.txt con save. Si usas load no podrás recuperar la información. Vete a la pestaña de File y activa la opción Import Data hasta terminar. Mira si existe alguna variable con el nombre ale3 en el workspace y visualiza su contenido, ¿qué ves?, ¿podrías utilizar la información que contiene?

Ayuda en el uso de *fopen, fprintf y fclose* en ficheros de texto.

fid=**fopen**('nombrefichero','modo')	**fid** es el identificador del fichero que se utilizará en el resto de las operaciones. Es un número natural que asigna matlab al nombre del fichero que le hayamos dado en *nombrefichero*. Es decir, *fid* es un nombre de variable a la que se le va a asignar el anterior valor numérico, de tal forma que internamente matlab sabe que el número asignado a fid se refiere al *nombrefichero*. **modo.** son las opciones identificativas del fichero r – se refiere a un fichero existente del que se va a leer la información que contiene. w– fichero que queremos crear. Si nos equivocamos y existiera se borraría toda su información. rt para leer el fichero en modo texto, es decir, legible

	wt para escribir un fichero en modo texto-
fclose(fid)	Cierra el fichero identificado con fid
Ejemplo**: fprintf**(fid, 'El valor es: %8.2f\n', y)	Escribe en el fichero identificado con fid una cadena de caracteres, el valor de la variable y según el formato indicado y finalmente salta de línea. La orden fprintf sólo se usara para crear ficheros con modo w.
Ejemplo: [A,contador]=**fscanf**(fid,'formato') Ejemplo: [A,contador]=**fscanf**(fid,'formato',[14 inf])	**fscanf** sirve para ver la información de tipo texto, es decir legible, que contiene el fichero identificado con fid. Esta información se almacena en un vector A y en contador se almacena el número total de datos que contiene el fichero. La opción [14 inf] estructura la variable A como una matriz de 14 filas hasta el final de los datos

SECCIÓN 8. Formato IEEE754 y errores

1. Un algoritmo es inestable cuando los errores de redondeo se acumulan degradando la exactitud del resultado final, a pesar de que en aritmética exacta el algoritmo sea correcto. La sucesión siguiente es un ejemplo de algoritmo numérico inestable,

$$x_{n+2} = \frac{13}{3}x_{n+1} - \frac{4}{3}x_n, tomando\ como\ valores\ iniciales x_0 = 1 y x_1 = \frac{1}{3}$$

Se puede comprobar que el término general de esta sucesión es igual a:

$$x'_n = \left(\frac{1}{3}\right)^n = \frac{1}{3}x'_{n-1}$$

Si suponemos que x'_n son los valores exactos de la sucesión se puede estimar el error de redondeo de x_n. Escribir un programa en MATLAB ***(.m)*** que resuelva esta sucesión y visualice en pantalla, desde *n* igual a 1 hasta 20 iteraciones, x_n, x'_n, el error absoluto $|x_n - x'_n|$y el error relativo $\left|\frac{x_n - x'_n}{x'_n}\right|$. Escribe el programa y los resultados que obtienes.

2. Matlab utilizan el formato IEEE754 en doble precisión para la representación de números reales:

a. Teniendo en cuenta que la mantisa es de 52 bits, corroborar que la precisión del computador es eps=2^{-52}
b. Realiza la siguiente operación a=1+2^{-52}, b=1-a. ¿Qué valor se obtiene?¿Por qué?
c. Realiza la siguiente operación a=1+2^{-53}, b=1-a. ¿Qué valor se obtiene?¿Por qué? ¿Qué tipo de error se produce y por qué?

d. Ejecuta el siguiente script. ¿Son correctos los resultados? Razona tu respuesta

```
k=1;
while ((1.0+2^(-k))>1.0)
        k=k+1;
end
fprintf(1,'El número de bits de la mantisa es %d\n', k-1);
fprintf(1,'El eps calculado es %e\n',  2^-(k-1));
```

e. ¿Qué tipo de errores se producen cuando se teclea las siguiente cantidades 10^308 y 10^309?

Números en binario

3. Crea la función dec2bin_entero que ante la entrada de un número natural devuelva un vector con el contenido de los números en binario. Usa las funciones internas *mod* y *floor*. Ejemplo:

```
>> dec2bin_entero(19)
ans =

  1   0   0   1  1
```

4. Crea la función bin2dec_entero que realice el funcionamiento inverso, dado un vector en binario devuelva su valor en decimal.

```
>> bin2dec_entero([1 0 0 1 1])
ans =

  19
```

5. Crea la función dec2bin_decimal que ante la entrada de un numero fraccionario (sin parte entera, es decir, decimal positivo y menor que cero) y devuelva un vector con el contenido de los números en binario. Ejemplo:

```
>> dec2bin_decimal(0.125)
ans =

  0   0   1
```

6 (Opcional). Crea la función dec_to_ieee754 que implemente la conversión al formato IEEE754 en simple precisión y normalización 1.m. La función tendrá la siguiente cabecera [ieee, signo, mantisa, ex]=dec_ieee754(x) donde:
- ieee es el vector con el número codificado en dicho formato

- signo corresponde al bit de signo en dicho formato
- mantisa corresponde al vector mantisa en dicho formato
- exponente corresponde al campo exponente en dicho formato

```
>> [i,s,m,e] = dec_to_ieee754(-0.125)
i = 1  0  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
s =  1
m =  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
e =  0  1  1  1  1  1  0  0
```

Esta función debe llamar a las funciones que previamente has creado: dec2b_decimal, dec2bin_entero.m

7 (Opcional). Crear la función análoga ieee754_to_dec que dado un número en formato IEEE754 en simple precisión devuelva el número convertido en decimal. Llama a tu finción bin2dec_entero.m

SECCIÓN 9. Generación de números aleatorios.

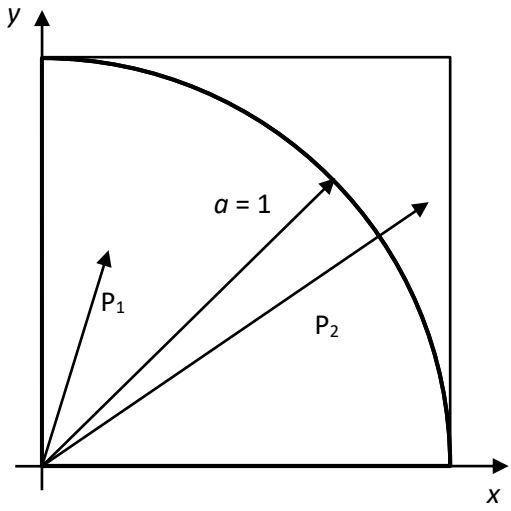
Copia el siguiente código de *Matlab* en un fichero *.m*, ejecútalo y comprueba que entiendes lo que hace. Puedes usar el comando *help* de *Matlab* para tener más información sobre la función *rand* y la función *randn.*

```
%%%%script para el uso de números aleatorios con la función rand%%%%
%Guardamos el estado actual del generador de números aleatorios esto nos permitirá
%reproducir la misma secuencia de números aleatorios siempre que queramos
s=rng;
%Generamos  un vector de cinco números aleatorios en el intervalo (0,1)
s1=rand(1,5)
%Generamos un nuevo vector de números aleatorios, deben ser distintos de
%los obtenidos en s1
s2=rand(1,5)
%Volvemos a reiniciar el generador de números aleatorios
rng(s);
%Si generamos un nuevo vector de números aleatorios, será igual que s1
s3=rand(1,5)

%Podemos reinicializar el generador de números aleatorios, empleando como semilla un valor
%entero cualquiera comprendido entre (0 y 2^32-1), es decir utiliza un registro de 32 bits para
%inicializar el generador.
%Ejemplo: reiniciamos el generador con semillas (0, 5, 10...25) y generamos una matriz de
%números aleatorios (cada fila está generada con una semilla distinta y contiene cinco números
%aleatorios.
aleatoria=[ ]
for i=0:5:25
   rng(i)
   aleatoria=[aleatoria; rand(1,5)]
end
```

```
%Si reiniciamos empleando cualquiera de las semillas anteriores (i), podremos reproducir los
%números de la fila correspondiente de la matriz aleatoria. Podemos así reproducir los
%correspondientes a la cuarta fila
rng(15)
fila4=rand(1,5)

%Si queremos cambiar la secuencia de números aleatorios ahora y que ya no siga la de la semilla
%15, podemos escribir
rng('shuffle');
```

1. Emplea el comando *rand* para generar un vector (fila o columna) de 10000 números aleatorios. Emplea el comando *randn* para generar otro vector de 10000 números aleatorios. Mediante el uso del comando *hist* dibuja los histogramas de los vectores generados (usa *subplot* para tener en el mismo gráfico los dos histogramas). Explica los resultados que observas.

2. Vamos a emplear el generador de números aleatorios de *Matlab* para calcular el número π. Para ello vamos a construir un cuadrado de lado la unidad (observa la siguiente figura). En dicho cuadro se describe el cuarto de circunferencia, como se observa en la figura, donde se cumple $\sqrt{x^2 + y^2} = 1$. Ahora vamos a generar números aleatorios *x* e *y* dentro de dicho cuadrado de lado unidad usando el comando *rand*.



Si el punto aleatorio generado $P_i(x,y)$ está en el interior de la circunferencia se debe satisfacer que $\sqrt{x_i^2 + y_i^2} \leq 1$, si por el contrario está fuera $\sqrt{x_i^2 + y_i^2} > 1$. Si generamos suficientes puntos aleatorios, el ratio entre el número puntos en el interior del cuarto de circunferencia y el número total de puntos generados corresponderá al ratio entre el área del cuarto de círculo y el área total del cuadrado. Es decir:

$$\frac{nº\ puntos\ dentro\ del\ cuarto\ de\ círculo}{nº\ de\ puntos\ totales\ generados} = \frac{área\ del\ cuarto\ de\ círculo}{área\ del\ cuadrado} = \frac{\frac{\pi \cdot a^2}{4}}{a^2} = \frac{\pi}{4}$$

Luego obtenemos el número π como:

$$\pi = 4\ \frac{nº\ puntos\ dentro\ del\ cuarto\ de\ círculo}{nº\ de\ puntos\ totales\ generados}.$$

Realiza una función en *Matlab* con el nombre *calculopi.m* que aproxime iterativamente el valor de π, empleando el sistema descrito que contemple los siguientes pasos:

- Genera un punto aleatorio $P_i(x,y)$.
- Comprobar si está dentro del cuarto de círculo o no.
- Estimar el valor de π con los puntos calculados hasta esa iteración.

La función deberá admitir como entrada el número máximo de puntos aleatorios generados y deberá representar los sucesivos valores de π obtenidos en cada iteración. Emplea el programa realizado para obtener una estimación de π empleando 2000 iteraciones. Representa también los puntos $P_i(x,y)$ mediante el uso de la opción de *axis square*.

3. A partir de rand, se pueden generar números aleatorios en cualquier intervalo que se desee.

Por ejemplo entre dos números enteros a y b:

```
>> r = a + (b-a).*rand(100,1); %genera 100 números aleatorios en el intervalo [a b].
```

Además, es posible generar números aleatorios enteros.

```
>> r = ceil(n.*rand(100,1)); %genera 100 enteros aleatorios comprendidos entre 1 y n.
```

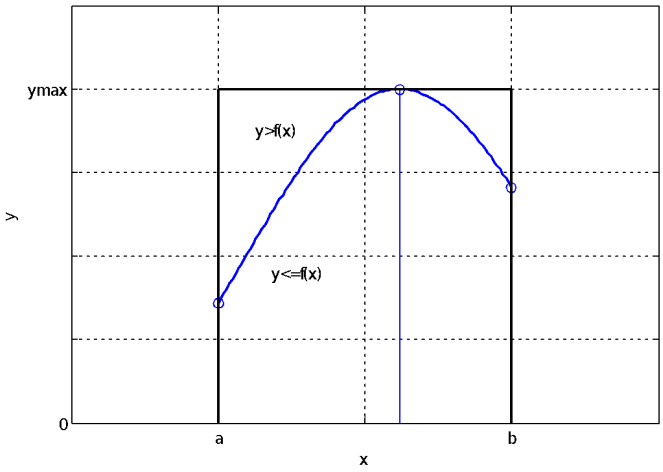
Escribe una función que simule un dado, es decir, genere aleatoriamente los números 1, 2, 3, 4, 5 y 6; y compruebe que la probabilidad de obtener un número par es 0.5, empleando la ley de los grandes números:

$$P = Probabilidad = \frac{nº\ de\ aciertos}{nº\ de\ intentos}$$

El programa deberá admitir como variable de entrada el número de intentos y devolver como variable de salida la probabilidad obtenida de sacar un número par (nota: para obtener la probabilidad en % multiplicar el resultado por 100).

4 (ejercicio complementario a). La probabilidad de sacar 2 veces cara al tirar una moneda al aire es P(cara)·P(cara)=(1/2)·(1/2)=1/4. Escribe un programa que simule el lanzamiento de una moneda y compruebe, mediante repetición de lanzamientos, el valor de la probabilidad de obtener dos veces seguidas cara.

5 (ejercicio complementario). Los números aleatorios pueden emplearse para calcular integrales definidas y en particular para calcular áreas y volúmenes. El procedimiento es parecido al que hemos visto antes para obtener el número π (ejercicio 2).



Para el caso del cálculo del área entre una curva y el eje de coordenadas *x*, podemos generar puntos aleatorios *P(x,y)* en el intervalo de integración [a,b] en la coordenada *x*; y en el intervalo [0, y_{max}] en la coordenada *y* (ver la figura adjunta). Donde y_{max} representa el valor máximo que toma la función que queremos integrar en el intervalo [a,b].

Al igual que el ejercicio 2, los puntos P(x,y) generados de forma aleatoria entre [a,b] para *x* y entre [0,y_{max}] para *y*, podrán estar por debajo (y <= f(x)) o por arriba de la curva (y > f(x)). Para un número muy elevado de puntos, la razón entre todos los puntos que caen por debajo de la curva y el número total de puntos corresponderá con la razón entre el área de la curva y el área total del rectángulo determinado por [a,b] y [0,ymax], es decir:

$$\frac{nº\ puntos\ y \le f(x)}{nº\ puntos\ totales} = \frac{\int_a^b f(x)dx}{(b-a)\cdot y_{max}} \to \int_a^b f(x)dx$$
$$= (b-a)y_{max}\frac{nº\ puntos\ con\ y \le f(x)}{nº\ puntos\ totales}$$

Escribe un programa basado en la descripción anterior que calcule el área de la función 1/x en el intervalo [1, 2], es decir:

$$\int_1^2 \frac{1}{x}dx$$

El resultado analítico de dicha integral es *log(2)*. Comprueba que la precisión obtenida mejora con el número de puntos aleatorios generados para obtener la integral.