

3.1. Introducción a MATLAB

MATLAB (abreviatura de "MATrix LABoratory", laboratorio de matrices en inglés) es un entorno de desarrollo interactivo y un lenguaje de programación, creado por la empresa Mathworks, ampliamente utilizado en matemáticas, física, ingeniería y otros campos técnicos que ofrece una amplia gama de herramientas para el análisis numérico, la visualización de datos y la programación. Su sintaxis es similar a los lenguajes tradicionales como C o Python, pero a mayor nivel. MATLAB está optimizado para el cálculo matricial, lo que lo convierte en una herramienta poderosa para resolver problemas matemáticos y científicos de manera eficiente. Además, MATLAB cuenta con una extensa colección de bibliotecas y funciones especializadas que facilitan la implementación de algoritmos avanzados en áreas como procesamiento de señales, procesamiento de imágenes, control automático, entre otros.

Para aprender Física Computacional, MATLAB es una herramienta poderosa, ya que tiene capacidad para realizar cálculos numéricos y simbólicos, un potente paquete de visualización y, es fácil de usar. Durante esta parte de la asignatura no vamos a conformarnos con aprender a usar muchas de las numerosas funciones que forman parte de este lenguaje de programación para cálculo simbólico y numérico, sino que aprenderemos a programar nuestras propias funciones. El objetivo central de esta parte de la asignatura es que cada uno de nosotros aprenda a generar su propia biblioteca de funciones, orientadas a resolver los problemas más o menos avanzados que se va encontrando a medida que progresa en sus estudios.

Para conseguir este fin debemos cubrir tres etapas: familiarizarnos con la sintaxis, aprender a usar la amplia ayuda, seguir los cursos online y resolver sus asignaciones. Una vez hecho esto ya estaremos en condiciones de programar en MATLAB.

En este tema nos familiarizaremos con el entorno de MATLAB y de las peculiaridades de la sintaxis de este lenguaje de programación, como por ejemplo el uso de "=" para hacer asignaciones, o el uso de un carácter especial para indicar el final de una instrucción: ";" dependiendo de que queramos, o no, visualizar el output producido por dicha instrucción.

Como es habitual en otros sistemas de álgebra computacional, en MATLAB se suele trabajar por medio de una interfaz gráfica, con información sobre las diversas (muy numerosas) funciones que ya vienen programadas en MATLAB. El manual al que nos referimos está disponible en el menú de ayuda.

Este manual es extremadamente útil, en él se explica la finalidad y sintaxis de todas las funciones del lenguaje MATLAB, además tiene buscadores que nos permiten localizar rápidamente cualquier función. El lenguaje MATLAB está provisto de un número enorme de funciones; ningún programador, por experto que sea, las conoce todas, de forma que el uso del centro de ayuda de Mathworks para consultar sintaxis u opciones de funciones ya conocidas, o también para localizar otras funciones que no conocíamos, es algo rutinario para todos los programadores y usuarios de este lenguaje. Por tanto es conveniente que desde el primer momento nos acostumbremos a consultar las herramientas de ayuda disponibles.

En particular, a lo largo de estas notas sobre MATLAB se hace referencia a diversos comandos o funciones del MATLAB, indicando su finalidad y (en algunas ocasiones) la sintaxis correspondiente. En todos estos casos se sobreentiende que la información aportada en estos apuntes debe complementarse con la información disponible en el manual, donde la sintaxis de estos comandos o funciones del MATLAB viene descrita con todo detalle, junto con las diversas opciones disponibles y algunos ejemplos de uso.

3.2. ¿Por qué MATLAB?

Si buscamos un sistema de álgebra computacional disponible en todos los sistemas operativos habituales, con capacidad para cálculo tanto simbólico como numérico, con un potente motor de representaciones gráficas, y disponible para la comunidad universitaria, la mejor opción es sin lugar a dudas MATLAB.

MATLAB es un *sistema de álgebra computacional* para la manipulación de expresiones matemáticas tanto simbólicas como numéricas, con capacidad para representaciones gráficas de funciones y datos en dos y tres dimensiones. MATLAB produce resultados con alta precisión usando expresiones exactas en el cálculo simbólico y representaciones con aritmética de coma flotante de precisión arbitraria en el cálculo numérico.

MATLAB incluye herramientas para todas las operaciones de cálculo habituales: operaciones con vectores y matrices, números complejos, diferenciación, integración, desarrollos de Taylor y Fourier, transformadas integrales (Laplace, Fourier), resolución de ecuaciones algebraicas por métodos analíticos y trascendentes por métodos numéricos, así como diversas herramientas para ecuaciones diferenciales ordinarias, sistemas de ecuaciones lineales, etc.

MATLAB dispone, además, de numerosos paquetes de herramientas (Toolboxes) que le añaden funcionalidades especializadas en multitud de campos de la física y la ingeniería.

MATLAB permite **llamar funciones y subrutinas escritas en C** (o Fortran) lo que nos permitirá trabajar con ambos lenguajes de manera simultánea.

MATLAB está disponible para toda la comunidad de la UNED a través de una licencia *Campus Wide* y se puede instalar sobre varios sistemas incluyendo Windows, Linux y MacOS X.

Actualmente existen otros sistemas de álgebra computacional que incluyen todas las características anteriores como MATHEMATICA o MAPLE, pero con el inconveniente de no estar disponibles para la UNED. El sistema gratuito de código abierto más próximo a MATLAB es el conocido OCTAVE, muy parecido en su funcionamiento a MATLAB. El OCTAVE es un sistema potente y parecido a MATLAB en todo lo referente a cálculo numérico y capacidad gráfica, también está disponible para todos los sistemas operativos habituales, pero no tiene capacidad para cálculo simbólico ni dispone de la documentación masiva que tiene MATLAB.

A continuación incluimos algunos enlaces con información útil sobre sistemas de álgebra computacional en general y sobre MATLAB en particular:

- Página central de MATLAB y documentación

<https://es.mathworks.com/products/matlab.html>

https://es.mathworks.com/help/?s_tid=gn_supp

- Sistemas de álgebra computacional

https://en.wikipedia.org/wiki/Computer_algebra_system

https://en.wikipedia.org/wiki/List_of_computer_algebra_systems

3.2.1. Descarga e Instalación

La instalación del programa MATLAB es muy sencilla y solo hay que seguir el enlace que se proporciona en el curso virtual <https://www.uned.es/universidad/campus/estudiantes/descargasoftware/Licencia-Campus-Wide-MATLAB.html> que permite la instalación del software independientemente del sistema operativo en el que se trabaje. Cuando se proceda a la instalación se podrán descargar los paquetes de herramientas que se quiera, teniendo en cuenta que pueden ocupar mucho espacio en el disco duro. En el siguiente listado se encuentran los paquetes indispensables para la asignatura:

1. Symbolic Math Toolbox (para Cálculo Simbólico)
2. Curve Fitting Toolbox (para Ajuste de Curvas)

Existen otros paquetes como "MATLAB Coder App" (que permite ir migrando de C a MATLAB) que pueden ser útiles para la realización del curso. En cualquier caso, la instalación de paquetes adicionales puede realizarse desde la pestaña "APPS", "Get More Apps".

El programa de instalación se encargará automáticamente de revisar qué bibliotecas adicionales necesitamos para instalar el MATLAB, nos informará de toda la lista de paquetes que vamos a instalar, las descargará de Internet y las instalará. Finalizado este proceso (unos pocos minutos si nuestra conexión de red es razonablemente rápida) ya estaremos en condiciones de empezar a trabajar con MATLAB.

Como se indica en el curso virtual, si su ordenador es antiguo o dispone de poca capacidad, recomendamos encarecidamente el uso de MATLAB online: <https://es.mathworks.com/products/matlab-online.html> a través de un navegador de internet.

3.2.2. Interfaz gráfica (*front end*)

MATLAB se presenta al usuario mediante una interfaz como la mostrada en la Figura 3.1. Como se puede observar, la interfaz por defecto consta de varias ventanas que incluyen una *ventana de comandos o command window*, que es donde introduciremos las operaciones básicas y que sirve para ejecutar los programas que crearemos, una *ventana de edición (editor)*, donde escribiremos nuestros algoritmos y dos ventanas que muestran el *workspace o "entorno de trabajo"* y un explorador de archivos. Hablaremos del "workspace" más adelante.

Si la ubicación de las ventanas integradas es diferente, se puede volver a ésta usando las opciones del menu **Layout** dentro de la pestaña **Home**. El Layout se puede modificar al gusto del usuario.

3.3. Sintaxis de MATLAB

3.3.1. Primera sesión con MATLAB

Para esta primera sesión utilizaremos MATLAB en línea de comandos. Como hemos comentado antes la forma habitual de trabajar es por medio del interfaz de la Figura 3.1, pero también es interesante observar, al menos una vez, cómo funciona este programa al nivel más

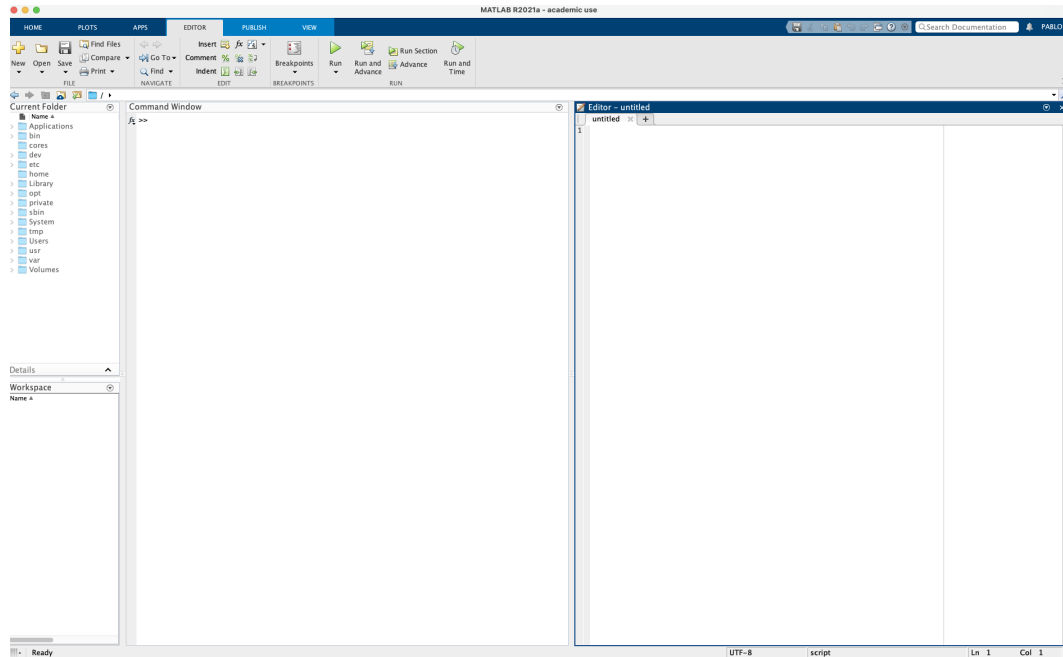


Figura 3.1: Pantalla de inicio de MATLAB.

básico. sin interfaz gráfica, donde solo se interactúa con la línea de comandos. Para ello en Linux o MACOS basta con abrir un terminal y escribir `"matlab -nodesktop -nosplash"`. Este comando convertirá nuestro terminal en una línea de comandos de MATLAB. Para volver al terminal bastará con escribir `» exit`. En Windows, se puede lanzar un comando similar desde las propiedades de inicio del programa (https://es.mathworks.com/help/matlab/matlab_env/startup-options.html).

En cualquier caso, las instrucciones que incluimos a continuación nos servirán para comenzar a familiarizarnos con la sintaxis de este lenguaje, independientemente del modo en el que hayamos abierto la línea o ventana de comandos. Se puede observar que en la línea de comandos aparece el símbolo `»`, llamado *prompt*, esperando nuestra primera acción. Si queremos calcular una simple suma tecleamos la operación deseada y una pulsación de la tecla de retorno `enter`.

```
» 2.5+4
```

a lo que MATLAB da la primera respuesta en la forma

```
ans=
```

```
6.5
```

```
»
```

donde (`ans`) es la abreviatura de *answer*. A continuación del `ans` observamos `»`, indicando que MATLAB espera una nueva instrucción.

Recordamos que en la notación anglosajona, la que sigue el MATLAB, la separación entre la parte entera y la parte decimal de un número se indica mediante un punto (`.`), mientras que la coma (`,`) se reserva para separar elementos de una lista (componentes de un vector, argumentos de una función, etc.), de tal forma que la siguiente acción da un resultado diferente

```
» 2,5+4
```

```
ans= 2
```

```
ans= 9
```

donde MATLAB ha interpretado que tenemos un vector de 2 elementos donde el primero es 2 y el segundo $5+4=9$.

Nuestra siguiente operación consistirá en asignar el valor 125 a la variable x y 46 a la variable y , solicitando luego su producto,

```
>> x=125; y=46; x*y
ans= 5750
```

Como vemos, el MATLAB ha notificado al usuario solamente el último de los cálculos que ha realizado, en este caso, el producto $x \cdot y$ con valor 5750. Esto se debe a que hemos hecho uso del (;) al final de cada una de las asignaciones. El (;) evita la salida del cálculo realizados que queda guardado en el *workspace*. El punto y coma actúa también como un separador cuando escribimos varias instrucciones en una misma línea.

3.3.2. Workspace

El Workspace (espacio de trabajo) es el conjunto de variables almacenadas en un momento dado en la memoria del MATLAB. Como se ha visto, si no se asigna el resultado a una variable específica, usando el símbolo (=), MATLAB asigna la operación a la variable "ans". Las variables creadas desde la línea de comandos o desde un script sin usar la definición **function** pertenecen al Base Workspace que se puede inspeccionar en la ventana correspondiente o usando los comandos `>> who` o `>> whos`. Estas variables permanecen en el Base Workspace cuando se termina la ejecución del script y se mantienen allí durante toda la sesión de trabajo o hasta que se borren, mediante el comando `>> clear`.

Cuando se usa una función **function**, las variables creadas en el espacio de trabajo de dicha función, que es independiente del espacio de trabajo base. Para compartir las variables entre una función **function** y el Base Workspace se puede utilizar el comando `>> global` seguido del nombre de las variables a compartir, como se verá más adelante.

3.3.3. Operaciones numéricas y reglas de prioridad

Las operaciones aritméticas en MATLAB se pueden realizar usando los símbolos siguientes:

- Suma (+)
- Resta (-)
- Multiplicación (*)
- División (/)
- Potencias (^)

En MATLAB existen unas reglas de prioridad para realizar las diferentes operaciones aritméticas. Por ejemplo, no es lo mismo

`>> 2+3*4`, `ans= 14` que `>> (2+3)*4`, `ans= 20`. Como se observa en este ejemplo, la multiplicación tiene prioridad (se ejecuta antes) que la suma. Así solo es posible cambiar el orden de prioridad de las operaciones usando paréntesis, que se ejecutarán en general de izquierda a derecha y de dentro a fuera `>> 2*((2+3)*4)`, `ans= 40`. Un resumen de las reglas de prioridad es: Potencias- Multiplicación y división- Sumas y restas.

3.4. Trabajando con MATLAB

Como se explica en el curso virtual, una de las mejores formas de aprender MATLAB es seguir los cursos on-line de los que dispone el propio programa que se deben complementar con estos apuntes. Especialmente útiles para iniciarse con MATLAB son los cursos:

- Curso de iniciación a MATLAB: <https://matlabacademy.mathworks.com/es/details/matlab-onramp/gettingstarted>.
- Curso de fundamentos de MATLAB: <https://matlabacademy.mathworks.com/es/details/matlab-fundamentals/mlbe>.
- Curso de Álgebra usando MATLAB: <https://matlabacademy.mathworks.com/es/details/introduction-to-linear-algebra-with-matlab/linalg>

Es fundamental destacar que el comando `>> help` seguido del nombre de la función que queramos ejecutar (si existe ya y tiene disponible la ayuda) proporciona la documentación necesaria para su ejecución.

3.4.1. Funciones y Scripts

MATLAB ofrece un potente lenguaje de programación que va mucho más allá del uso de la línea de comandos. Para explotar este lenguaje se puede escribir una serie de comandos en un archivo y luego ejecutarlo como si se llamara a cualquier función de MATLAB. Se puede usar el editor de MATLAB o cualquier otro editor de texto para crear nuestros propios archivos de funciones. Existen dos tipos de archivos que podemos usar para este fin:

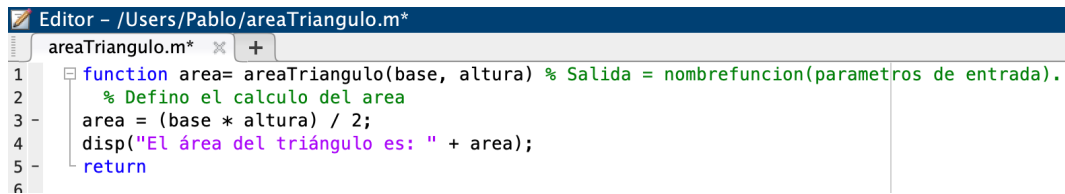
- **Scripts:** Los scripts son archivos con extensión ".m" que contienen una serie de comandos de MATLAB que se ejecutan secuencialmente. Son útiles para tareas simples o para ejecutar código de forma interactiva. Los scripts no pueden recibir entradas ni producir salidas de forma explícita, pero pueden modificar variables en el workspace.
- **Funciones:** Las funciones en MATLAB son bloques de código reutilizables que encapsulan una tarea específica. Permiten organizar el código, mejorar la legibilidad y facilitar el mantenimiento. Las funciones pueden recibir entradas (argumentos) y producir salidas (resultados). Se definen con la palabra clave `function` y se guardan, al igual que los scripts, en archivos ".m".

Una vez escritos, tanto las funciones como los scripts se pueden ejecutar escribiendo su nombre como si fuera un comando en la ventana de comandos o haciendo clic en el botón "Run" en el Editor de MATLAB. Para la primera opción hay que asegurarse de estar en el mismo directorio que el archivo que se quiera ejecutar, el *directorio de trabajo* (`>> pwd`). Las funciones se pueden ejecutar igualmente mediante la función (`>> feval`) usando los argumentos de entrada. Veamos un ejemplo:

Se quiere programar una función que devuelva el área de cualquier triángulo de altura y base dadas. Para ello escribimos una función como la que aparece en la imagen [3.2](#).

Para escribir esta función hemos seguido los siguientes pasos:

1. Declarar que es una función con la palabra "function". Posteriormente se escribe la salida deseada y el nombre de la misma y los parámetros de entrada.



```

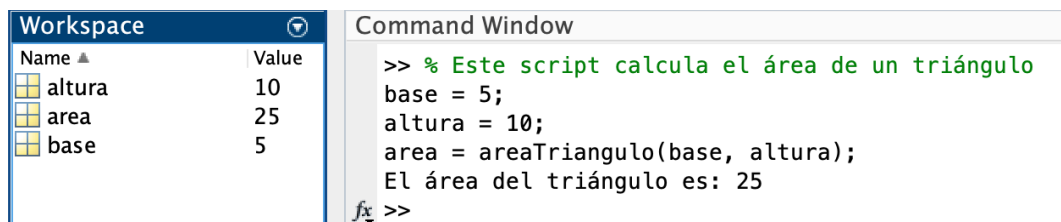
Editor - /Users/Pablo/areaTriangulo.m*
areaTriangulo.m*
1 function area= areaTriangulo(base, altura) % Salida = nombrefuncion(parametros de entrada).
2 % Defino el calculo del area
3 area = (base * altura) / 2;
4 disp("El área del triángulo es: " + area);
5 return
6

```

Figura 3.2: Función para calcular el area de un triángulo.

2. Se define como se calcula la salida en función de los parámetros de entrada.
3. Se salva el archivo con el nombre deseado y extensión ".m".
4. En este caso vamos a pedir que se devuelva el resultado en pantalla, con el comando "disp" equivalente al comando "print" en C.
5. Observe que el comentario en MATLAB es %, el equivalente a (#) en C.

Para obtener el area de un Triángulo de base 5 y altura 10 usando la función que hemos escrito se pueden seguir ahora varias aproximaciones. La primera sería escribir una secuencia de comandos como la que se muestra en la Figura 3.3. Como se puede ver, hemos asignado los valores deseados a las variables *base* y *altura* y nos ha devuelto el valor de la variable *area*, como se puede ver en la ventana del workspace.



Name	Value
altura	10
area	25
base	5

```

>> % Este script calcula el área de un triángulo
base = 5;
altura = 10;
area = areaTriangulo(base, altura);
El área del triángulo es: 25
fx >>

```

Figura 3.3: Llamada a la función para calcular el area de un triángulo.

Como ejercicio compruebe que los nombres entre los diferentes workspaces son independientes y que por lo tanto el resultado sería similar si usáramos:

```
>> x=5; y =10; area2=areaTriangulo(x,y);
```

Hay que evitar duplicar nombres de funciones, ya que MATLAB ejecuta solamente aquella que aparece primero en la ruta de búsqueda. Para comprobar si existen funciones similares se puede usar la instrucción `>> which areaTriangulo -all`, que en este caso devolverá la ruta donde se encuentra el archivo llamado 'areaTriangulo.m'.

Finalmente, pruebe a guardar las instrucciones de la Figura 3.3 en un archivo ".m" usando el editor y después ejecútelo escribiendo su nombre (sin la extensión) en la línea de comandos o pulsando el botón *Run* de la pestaña del editor.

Tipo de Dato	Descripción	Ejemplo
Numérico	- Enteros: 'int8', 'int16', 'int32', 'int64' - Reales: 'float32', 'float64' - Complejos: 'complex64', 'complex128'	'numero = 10' (entero) 'pi = 3.14159' 'z = 1 + 2i'
Cadena	Almacena texto	'texto = "Hola mundo"'
Lógica	Almacena valores booleanos (Verdadero/Falso)	'bandera = true'
Celda	Almacena una colección de datos heterogéneos	'datos = {"Juan", 25, true}'
Estructura	Almacena una colección de datos relacionados con nombres	'persona = struct("nombre", "Juan", "edad", 25)'
Matriz	Almacena una colección de datos del mismo tipo organizados en filas y columnas	'matriz = [1 2 3; 4 5 6]'

Tabla 3.1: Tabla de Tipos de Variables en MATLAB

3.4.2. Variables

Como hemos visto anteriormente, en MATLAB, las variables se declaran simplemente asignándoles un valor. No es necesario especificar el tipo de dato de la variable, ya que MATLAB lo infiere automáticamente.

Por ejemplo si escribimos `>> num= 10;` MATLAB entiende directamente que `num` es una variable numérica, mientras que si escribimos `>> texto = 'Hola mundo'; 0 >> caso = true;`, MATLAB asigna la variable `texto` como una cadena de caracteres y `caso` como una variable lógica. Se puede ver el tipo de variables asignadas usando el comando `>> whos`, como hemos comentado anteriormente.

MATLAB tiene una variedad de tipos de variables para almacenar diferentes tipos de datos. Su uso se define en la Tabla [3.4.2](#). Como reglas generales:

- Cualquier tipo de dato numérico puede ser utilizado para crear matrices.
- Los nombres de las variables deben empezar por una letra y pueden contener letras, números y guiones bajos.
- MATLAB distingue entre mayúsculas y minúsculas en los nombres de las variables.

Es importante elegir el tipo de dato adecuado para cada variable, ya que esto puede afectar al rendimiento y la precisión de los cálculos. Para almacenar un número entero, como la edad de una persona, se puede usar una variable de tipo `int32`. Para almacenar un nombre, se puede usar una variable de tipo `char`. Para almacenar un valor booleano, como si una persona está casada o no, se puede usar una variable de tipo `logical`. Para almacenar una colección de datos heterogéneos, como una lista de nombres y edades, se puede usar una variable de tipo `cell`. La elección del tipo de dato adecuado para cada variable es una parte importante de la programación en MATLAB.

3.4.3. Control del cálculo y memoria

Como otras características importantes, desde la línea de comandos de MATLAB se puede interrumpir en cualquier momento el cálculo de un script o de una orden, evitando así demoras si el usuario decide que la ejecución tarda demasiado usando la combinación de teclado `[Control] + [C]`. Como hemos dicho antes MATLAB también permite liberar memoria, aplicando el comando `>> clear` o `>> clearvars`.

Función	Descripción
sqrt	Raíz cuadrada
exp	Exponencial
log	Logaritmo natural
sin	Seno
cos	Coseno
tan	Tangente
atan	Arcotangente
round	redondeo hacia el entero más próximo
conj	Complejo conjugado

Tabla 3.2: Algunas funciones predefinidas en MATLAB

Constante	Valor
π , pi	3.141592653589793
e , e	2.718281828459045
i , i	Unidad imaginaria
∞ , inf	Infinito positivo
$-\infty$, -inf	Infinito negativo
NaN, NAN	Sin dato

Tabla 3.3: Algunas constantes predefinidas en MATLAB

3.4.4. Funciones y Constantes Básicas de MATLAB

MATLAB es un entorno de software de mucha potencia y como tal ofrece predefinidas una amplia gama de funciones y herramientas para realizar tareas básicas y avanzadas, como las que se muestran en la Tabla 3.2, entre muchas otras. MATLAB también posee una serie de constantes predefinidas útiles para el usuario (Tabla 3.3).

3.4.5. Operaciones aritméticas

Como se ha comentado anteriormente, los operadores aritméticos más comunes son suma (+), resta (−), multiplicación (*), división (/), y potencia (^). MATLAB por defecto devuelve los resultados de forma exacta, sin aproximaciones ni cálculos numéricos. Así podemos obtener resultados sencillos de expresiones complicadas como:

$$P = \left(\frac{2^5}{1 + \frac{1}{(2/3)^3 + (3/2)^2}} \right)^{-3}$$

en forma exacta

```
>> (2^5/(1+1/((2/3)^3+(3/2)^2)))^(-3);
>> P= 8.2442e-05
```

Observe que el operador `e-05` es el equivalente a 10^{-5} . MATLAB permite variar la precisión de esta operación escribiendo los comandos `>> format long`, `>> format short`. Entre los diferentes formatos que MATLAB ofrece para representar números, se encuentran el Decimal, Binario, Hexadecimal, etc... Como ejemplo, consideremos el número decimal 100. Este número puede ser representado en diferentes formatos en MATLAB:

```
% Representación de un número en diferentes formatos
numero_decimal = 100;
numero_binario = dec2bin(numero_decimal);
numero_hexadecimal = dec2hex(numero_decimal);
numero_octal = dec2oct(numero_decimal);

% Conversión entre bases
binario = '1100100';
decimal = bin2dec(binario);

% Mostrar resultados
disp('**Representación de un número en diferentes formatos:**');
disp(['Decimal: ', num2str(numero_decimal)]);
disp(['Binario: ', numero_binario]);
disp(['Hexadecimal: ', numero_hexadecimal]);
disp(['Octal: ', numero_octal]);

disp('**Conversión entre bases:**');
disp(['Binario: ', binario]);
disp(['Decimal equivalente: ', num2str(decimal)]);
```

3.4.6. Teoría elemental de números con MATLAB

MATLAB tiene una serie de comandos que resultan muy útiles para verificar algunos teoremas de la teoría de números, que además nos sirven para poner en práctica parte de lo aprendido hasta ahora. Tenemos `>> isprime(n)`, que verifica si el número n dado es primo o no, donde las respuestas pueden ser `true` o `false` (o su equivalente en 1 y 0 como variable lógica). En este sentido conviene recordar que por convención se considera que la unidad no es un número primo. Otras funciones interesantes son `>> primes(n)`, que retorna todos los números primos menores que el número n dado, `>> gcd(n_1, n_2)`, que da el máximo común divisor de los dos números dados n_1 y n_2 , o `>> factor (n)`, que factoriza el número n dado en producto de sus factores primos elevados a las potencias correspondientes.

3.5. Aprendiendo MATLAB

3.5.1. Niveles de uso de MATLAB

Cualquier sistema de álgebra computacional, como el MATLAB, se puede usar a distintos niveles, tal y como resumimos a continuación.

En nuestro caso el nivel más básico consiste en abrir una sesión de MATLAB, escribir en la línea de comandos algunas expresiones y posteriormente manipularlas por medio de las funciones que escribiremos en el editor. Este nivel de *usuario básico* nos ofrece una capacidad de computación similar a la de una calculadora científica muy avanzada, con capacidad para hacer gráficas y con la posibilidad adicional de guardar la sesión de trabajo en un archivo, lo que nos permite volver a abrir en otro momento nuestra sesión de trabajo y tomarla como punto de partida para cálculos posteriores. En este sentido tenemos el comando `>> save` que nos permite guardar el *workspace* entero o las variables deseadas. Para acceder a este nivel

de uso basta, por tanto, con tener el programa instalado en el ordenador, escribir alguna expresión y posteriormente explorar un poco las posibilidades de cálculo, representación gráfica y manipulación de expresiones matemáticas que nos ofrece MATLAB, sin necesidad de tener conocimientos previos sobre este lenguaje de programación. Por supuesto que todos hemos sido *usuarios básicos* cuando hemos empezado a usar este lenguaje, no hay nada malo en ello.

El siguiente nivel, que podríamos denominar de *usuario medio*, consiste en no limitarse a manipular expresiones por medio de los comandos y funciones disponibles en MATLAB, sino en entender y aplicar los comandos o funciones que proporciona el lenguaje MATLAB. Al cabo de unas pocas sesiones de trabajo, cualquier usuario básico con un poco de curiosidad se convierte sin darse cuenta, de manera natural, en un usuario medio. Para ello basta con plantearse los problemas que se quieren resolver y ponerse a trabajar sobre el programa. Una forma muy práctica de ir aprendiendo es la siguiente:

- Sin necesidad de tener conocimientos previos de MATLAB seleccionamos alguna operación que nos interese de las disponibles en el **listado de funciones predefinidas** <https://es.mathworks.com/help/matlab/referencelist.html>. Por ejemplo, si nos interesa factorizar polinomios buscamos en la ayuda y encontramos el comando `>> roots`.
- Lo primero que podemos hacer para ver cómo se usa este comando es visualizar un ejemplo de uso. Para ello podemos ejecutar el comando `>> help roots` en la línea de comandos y leer la ayuda e incluso pinchar en la referencia a la documentación que nos redireccionará a la página de ayuda con múltiples ejemplos sencillo de uso de esta función.
- Explorando un poco la documentación de ayuda localizaremos inmediatamente otros comandos adicionales (en la parte de abajo: *See Also*), relacionados con el comando que habíamos seleccionado al principio. De esta forma aprenderemos rápidamente, en unas pocas sesiones de trabajo, a usar otras muchas funciones del MATLAB y opciones de uso de estas.
- Evidentemente la mejor forma de aprender MATLAB es usándolo (**y completando los extensos tutoriales disponibles en el curso virtual!**). Para que esta tarea no nos resulte demasiado ardua lo que debemos hacer es usarlo para resolver problemas que nos parezcan interesantes, explorando en la documentación qué herramientas nos ofrece este lenguaje de programación para resolver el problema en que estemos trabajando.

Estos son los pasos que hemos seguido todos para pasar de *usuarios básicos* a *usuarios medios*.

El siguiente nivel de uso de MATLAB, y objetivo de esta parte de la asignatura, es el de convertirse en *usuarios avanzados*. Ser un usuario avanzado no consiste en saberse de memoria cientos de comandos de MATLAB, sino en saber programar funciones propias usando este lenguaje de programación. Para ello es imprescindible tener algunos conocimientos básicos sobre sintaxis y funciones del MATLAB y sobre todo hay que saber buscar la información necesaria para localizar las herramientas de este lenguaje que podemos necesitar en un problema concreto. Como decíamos antes, ningún usuario de MATLAB conoce todas las funciones disponibles en este lenguaje, lo que caracteriza a un usuario avanzado es que es capaz de localizar las funciones que necesita usar, y es capaz de escribir sus propias funciones cuando lo que quiere hacer no está programado en ninguna función o comando propio del lenguaje MATLAB.

3.5.2. Comandos e instrucciones de uso frecuente

En los ejemplos disponibles en los apuntes y en los cursos virtuales de *MathWorks* pueden encontrarse diversos ejemplos de uso de bucles y otras instrucciones empleadas frecuentemente en programación con MATLAB. A modo de introducción, a continuación mencionamos un breve listado de comandos e instrucciones de uso frecuente. Por supuesto que es en la ayuda del MATLAB donde debe consultarse la información completa sobre uso, sintaxis, opciones, ejemplos, etc. acerca de estas instrucciones:

■ Operaciones básicas

- Para las operaciones básicas se emplea: +, -, *, /. Para la operación “elevar a una potencia” se emplea el símbolo ^. Para las funciones matemáticas habituales MATLAB cuenta con las funciones habituales `sin`, `cos`, `log`, etc. En la ayuda de MATLAB podremos encontrar sin ninguna dificultad cualquiera de estas funciones a medida que nos vayan haciendo falta. Para las operaciones matriciales se emplea `(.)` si se opera elemento a elemento (las matrices que multiplicamos deben tener dimensiones compatibles, de lo contrario MATLAB no podrá realizar el producto y nos informará de un error).

■ Asignaciones

- MATLAB, al igual que la inmensa mayoría de los lenguajes de programación, usa el símbolo '=' para asignar un valor a una variable. El símbolo "==" en MATLAB se emplea para operaciones lógicas que se verán más adelante.

■ Bucles

En programación los bucles son una de las instrucciones más habituales y útiles. En MATLAB para hacer un bucle hacemos, en general, lo siguiente:

```
for índice = valor inicial : incremento : valor final ; instrucciones end
```

Por ejemplo, el siguiente bucle escribe los primeros 50 múltiplos de 3:

```
for i=1:1:50; disp(3*i); end
```

En la variable `i` puede omitir el incremento y MATLAB asume que es la unidad:

```
for i=1:50; disp(3*i); end
```

El mismo resultado se obtiene con este otro bucle:

```
for i=3:3:150; disp(i); end
```

Aparte del valor final la instrucción que define cuándo debemos interrumpir la ejecución del bucle también puede programarse por medio de una condición, empleando para ello la instrucción `while`, o, como en todos los lenguajes de programación MATLAB, usando la instrucción `if`, con la sintaxis habitual `if ...else ...elseif ...end`, etc.

3.5.3. ¿Cómo preparar esta parte de la asignatura?

La forma en que está diseñada esta parte de la asignatura es la siguiente. En estos apuntes *no* se reproduce la información disponible en los cursos on-line MATLAB. Estos apuntes contienen diversos ejemplos de programación de funciones, unas muy sencillas, otras un poco más complicadas, por medio de las cuales vamos a ir explorando algunas de las muchísimas posibilidades que nos ofrece este lenguaje de programación. En todo momento hemos intentado que los problemas matemáticos sobre los que versan estos ejemplos resulten interesantes y útiles. Por este motivo los ejemplos están relacionados con problemas de matemáticas de interés en física. **Sin embargo es necesario realizar los cursos on-line disponibles en Agora para un máximo aprovechamiento del curso.** La manera de trabajar con estos apuntes es la siguiente.

- De manera secuencial (uno a uno y en orden) vamos trabajando con las funciones programadas en los ejemplos.
- Leemos cuidadosamente los comentarios que aparecen en dichas funciones, donde se explica qué es lo que hace cada línea de código.
- Guardamos estas funciones en un archivo (*.m) y las llamamos bien desde el editor o desde la línea de comandos.
- En las funciones que aparecen como ejemplos en estos apuntes se usan diversos comandos o funciones propios del lenguaje MATLAB:
 - Es fundamental estudiar en la documentación del MATLAB la información disponible sobre estas funciones, no para aprendérselas de memoria, sino para entender qué es lo que hace cada línea de código de las funciones que ponemos como ejemplo y también para aprender qué otras opciones nos ofrece este lenguaje de programación.
- Para adquirir *experiencia* suficiente es fundamental recorrer todos los ejemplos disponibles en todos los apuntes, independientemente de que estén relacionados, o no, con los problemas propuestos con la PEC del año en curso.

3.5.4. Errores frecuentes

- Tanto si se trabaja con SO MS Windows o con sistemas tipo UNIX, como el Linux, cuando especificamos la ruta o path de un archivo en el disco en MATLAB hay que indicar los sub-directorios por medio del carácter empleado en UNIX: “/”, en lugar del *backslash* empleado en Windows.
- No se deben usar caracteres no estándar en los nombres de archivos y/o directorios, esto incluye
 - caracteres con tilde,
 - espacios,
 - letra “ñ”,
 - etc.

Aunque el sistema operativo nos permita generar nombres de archivo y/o directorios empleando caracteres no estándar, luego es una fuente de problemas para muchos programas. Por ejemplo, la instrucción `cd` que asigna el directorio de trabajo no funcionará si en el nombre del archivo o en el path existe algún carácter no estándar.

- Verificar muy cuidadosamente la sintaxis de las funciones que se programan. Cualquier error sintáctico, por pequeño que sea, convierte un código en algo totalmente incomprensible para el MATLAB. Los errores sintácticos más frecuentes son
 - Paréntesis sin cerrar o mal cerrados (por ejemplo, cerramos con “]” un paréntesis abierto con “(”, en cualquier lenguaje de programación estos caracteres tienen significados y finalidades distintos).
 - Falta una coma “,”, o se ha sustituido por otro signo de puntuación (. , ; , ...). Igual que antes, en cualquier lenguaje de programación los distintos signos de puntuación son caracteres reservados con significados y finalidades distintos.
 - “Hay una errota (falta una ltra, se ha sustituido una letri por utra, ...).” Aunque los seres humanos somos capaces de darnos cuenta instantáneamente de este tipo de errores, y podemos comprender sin dificultad el significado de una señal con ruido, como un texto lleno de errores, las máquinas no son tan flexibles. Un ordenador lee y ejecuta exactamente lo que pone en el código, si el código tiene un error el programa o bien no funcionará o, peor aún, funcionará haciendo algo diferente de lo que creemos que hace.
 - MATLAB proporciona el siguiente error: *Error: Invalid text character. Check for unsupported symbol, invisible character, or pasting of non-ASCII characters.* Este error es muy común cuando se copia/pega texto y puede indicar que hay un espacio al inicio del código pegado, que no debería estar en esa posición.

Todos estos errores son sencillos de cometer si se trabaja demasiado deprisa, y también es fácil localizarlos si se examina el código con atención. La claridad y limpieza del código es fundamental en este sentido. Como regla general, cuando algo que hayamos programado no funcione, o MATLAB se demore demasiado en generar una respuesta, lo primero que debemos sospechar es que quizá hay un error sintáctico en alguna parte.

3.6. Álgebra Computacional con MATLAB

En esta sección aprenderemos a utilizar MATLAB para operar con vectores y matrices, veremos algunos ejemplos con matrices de rotaciones y cálculo de autovalores y autovectores, y finalmente nos centraremos en el tema de la aplicación de cambios de base sobre vectores y matrices, lo cual es una operación muy habitual en álgebra especialmente cuando se estudia el tema de diagonalización de matrices.

3.6.1. Operaciones elementales con vectores y matrices

Además de expresiones y funciones escalares, con MATLAB también podemos manipular fácilmente expresiones que contengan vectores y matrices, de hecho está especialmente diseñado para ello. Para ello emplearemos *matrices* con varios elementos. Tenga en cuenta que

un vector es también una matriz en la que una de sus dos dimensiones es 1.

Por ejemplo, para asignar a la variable a un vector de 6 elementos (p. ej. A, B, C, D, E, F) la sintaxis es la siguiente:

```
>> a = [A, B, C, D, E, F];
```

Lógicamente MATLAB proporcionará un error si no hemos definido antes los valores de “A” a “F”. Un código funcional sería:

```
>> A=1; B=2; C=3; D=4; E=5; F=6;
```

```
>> a = [A, B, C, D, E, F]
```

que devolverá un vector de 1 fila y 6 columnas asignado a la variable a . Tenga en cuenta que la expresión $\gg a = [A \ B \ C \ D \ E \ F]$ es equivalente. Este mismo vector se puede escribir como vector columna usando (;) :

```
>> a = [A; B; C; D; E; F].
```

Para transformar a de vector fila a vector columna se puede usar el comando $\gg a'$ que nos permite transponer vectores y matrices.

Finalmente, para extraer elementos del vector la notación es “ $a(n)$ ”, siendo n el valor del índice cuyo elemento nos interesa (p. ej. $a(1)$ nos devuelve la variable A , en nuestro caso 1).

UNA CONSIDERACIÓN IMPORTANTE: En MATLAB, para aplicar operaciones aritméticas sobre cualquier vector o matriz deberemos tener en cuenta el tamaño del mismo y el resultado que queremos obtener.

Siguiendo el ejemplo anterior, si hacemos $\gg a + 1$ veremos que MATLAB suma 1 a cada elemento del vector. Algo parecido puede ocurrir cuando se usa $\gg \sin(a)$ donde MATLAB interpreta que queremos el seno de cada elemento del vector o cuando se multiplica un vector por un escalar, donde todos sus elementos serán multiplicados por el escalar, por ejemplo $\gg 2*a$.

Sin embargo, cuando se multiplica o divide una matriz, o un vector, por otro, MATLAB requiere que las dimensiones sean apropiadas. Por lo tanto, si hacemos $\gg a^2$ o $\gg a/a$, obtendremos un error, dado que MATLAB interpreta que queremos multiplicar un vector de dimensiones (1,6) por otro vector de dimensiones (1,6), lo cual es incompatible.

Para ejecutar una operación aritmética o aplicar una función *elemento a elemento* sobre un vector o matriz en MATLAB, es imprescindible usar el símbolo (.) en la operación. Por ejemplo, si queremos obtener cada elemento de a al cuadrado deberemos escribir:

```
>> a.^2, mientras que su división elemento a elemento será >> a./a
```

Supongamos ahora que definimos los vectores $\gg a = [1 \ 3 \ 5]$ y $\gg b = [2 \ 4 \ 6]$. Si queremos multiplicar $a * b$ *elemento a elemento* y guardarlo en un vector c deberemos escribir:

```
>> c = a.*b, que calcula el vector >> c = [2 12 30]
```

Por el contrario, si lo que queremos es calcular el producto escalar entre los vectores a y b deberemos usar el comando:

```
>> c = a*b', que calcula el escalar  $c = (1 * 2) + (3 * 4) + (5 * 6) = 44$ 
```

Observe que el comando $\gg c = a*b$ tampoco sería aplicable al tener ambos vectores dimensiones incompatibles. Como curiosidad, observe que el producto escalar podría también calcularse como $\gg c = \text{sum}(a.*b)$, donde *sum* suma todos los elementos de la multiplicación *elemento a elemento* entre los vectores *a* y *b*.

Definir matrices es equivalente en MATLAB a definir vectores, mediante la secuencia de construcción [elementos fila 1 ; elementos fila 2; ...]. Por ejemplo, definimos *M* como la siguiente la matriz (3×3):

```
 $\gg M = [3 \ -1 \ 0; \ 2 \ -3 \ 1; \ 4 \ 4 \ -2];$ 
```

```
M =
3  -1  0
2  -3  1
4   4 -2
```

MATLAB almacena las entradas M_{ij} de la matriz en la forma $M(i,j)$, p. ej. podemos comprobar que $M(2,3) = 1$.

Por medio de esta sintaxis podemos definir cualquier matriz a partir de sus *filas*. Si la información que tenemos sobre una matriz está dada *por columnas*, se puede igualmente introducir esta matriz por filas y posteriormente calcular la traspuesta por medio de $\gg M'$. Por ejemplo, vamos a definir la matriz $M^T = M'$

```
 $\gg MT = [3 \ -1 \ 0; \ 2 \ -3 \ 1; \ 4 \ 4 \ -2]'$ 
MT=
3    2    4
-1   -3    4
0     1   -2
```

El mismo operador que hemos usado antes para calcular el producto de dos vectores (*) es el que se usa para el producto de matrices y para el producto de matrices por vectores (que es un caso particular del producto de matrices). Por ejemplo, para calcular el resultado de aplicar *M* sobre el vector (x, y, z) hacemos

```
 $\gg M * [x, \ y, \ z]'$ 
ans =
3x - y
z - 3y + 2x
-2z + 4y + 4x
```

Por supuesto, para poder aplicar el operador de producto matricial * es necesario que las matrices o vectores sobre los que actúa tengan las dimensiones adecuadas, de lo contrario MATLAB nos informará sobre un error al intentar multiplicar matrices con dimensiones incompatibles. Veamos el mismo ejemplo aplicado mediante el uso del vector $[2 \ 1 \ 1]$. El código

```
 $\gg M * [2 \ 1 \ 1]'$ 
```

devuelve un vector de dimensiones adecuadas al producto $= (3, 3) * (3, 1) = (3, 1)$:

```
ans =
```

5
2
10

, mientras que el código:

```
>> M * [2 1 1] nos devuelve un error.
```

También podemos hacer el producto *elemento a elemento* entre un vector y una matriz si las dimensiones son adecuadas. Así:

```
>> M.* [2 1 1]
```

, devuelve el resultado de multiplicar cada elemento del vector en toda la fila de la matriz: MT=

```
6 -1 0
4 -3 1
8 4 -2
```

Si los elementos de la matriz siguen una forma funcional de su posición por filas y columnas, esto es, si conocemos una función $f(i, j)$ tal que asigne a cada entrada M_{ij} de la matriz su valor correspondiente, se pueden ensamblar matrices usando bucles. Por ejemplo, la matriz de Hilbert de dimensión n está dada por los valores

$$H_{ij} = (i + j - 1)^{-1},$$

con i y j entre 1 y la dimensión de la matriz, n . Construyamos ahora la matriz de Hilbert 4×4 , utilizamos un bucle:

```
% Definamos el tamaño de la matrix
m = 4;
% Creamos una matriz de ceros de dimensiones (MxM) que iremos rellenando
hilmat = zeros(m)
% Sobre cada elemento de la matriz aplicamos la fórmula de Hilbert
for i=1:m% itero en las filas
    for j=1:m% itero en las columnas

        % Definimos la formula de Hilbert
        den = 1/(i+j-1) ;

        % Aplicamos en cada elemento
        hilmat(i,j)= den

    end
end
```

Así, hemos podido obtener:

```
hilmat=
[ 1 1/2 1/3 1/4
  1/2 1/3 1/4 1/5
  1/3 1/4 1/5 1/6
  1/4 1/5 1/6 1/7 ]
```

Una de las características principales de las matrices de Hilbert es que aunque sus elementos son de orden unidad su determinante es sorprendentemente pequeño. Utilizamos el comando

`det` y confirmamos esta característica en nuestro caso,

```
>> det(hilmat)
ans = 1.6534e-07
```

Recordamos que los determinantes sólo están definidos para matrices cuadradas. Como iremos observando, la librería de funciones de MATLAB es inmensa y nuestro bucle anterior puede resumirse usando la función `H = hilb(n)`.

Otras construcciones que MATLAB admite de forma directa, son la matriz identidad `eye` (dimensión), la matriz nula `zeros` (número de filas, número de columnas), y cualquier matriz diagonal con el comando `diag`. Además tiene un comando específico para verificar si una lista de elementos construida es una matriz, `ismatrix(variable)`, con dos posibles outputs, `true` o `false`.

Retomamos ahora la matriz M definida numéricamente al comienzo de esta sección. Con las matrices se pueden realizar numerosas operaciones en MATLAB, por ejemplo operando en bloques podemos añadirle una fila nueva

```
>> M1 = [M; [0 2 0]]
M1=
 3  -1  0
 2  -3  1
 4   4 -2
 0   2  0
```

o una columna nueva:

```
>> M2 = [M [-1 2 2]']
M2 =
 3  -1  0  -1
 2  -3  1   2
 4   4 -2   2
```

Una de las operaciones más habituales con matrices es calcular la *matriz inversa*. En MATLAB esto se puede hacer o bien elevando la matriz a la potencia -1 o bien por medio del comando `inv`:

```
>> inv(M);
ans =
 -1   1  0,5
 -4   3  1,5
 -10  8  3,5
```

y se puede comprobar que M^{-1} ; produce el mismo resultado. Un ejercicio que puede hacerse con las operaciones definidas hasta ahora es comprobar el teorema de los determinantes

$$\det(A \cdot B) = \det(A) \det(B), \quad \det(A^n) = \det(A)^n.$$

3.6.2. Matrices Ortogonales, Rotaciones

La localización de un punto en el espacio euclídeo tridimensional suele darse en función del vector de posición en coordenadas cartesianas respecto al origen de coordenadas $\mathbf{r} = (x, y, z)$. Las operaciones habituales con vectores resultan extremadamente útiles cuando se interpretan de manera geométrica como operaciones sobre vectores de posición de puntos en un espacio. Por ejemplo, si desplazamos el vector \mathbf{r} anterior en la dirección de otro vector

$\mathbf{d} = (d_1, d_2, d_3)$, la posición final de \mathbf{r} tras este desplazamiento estará dada por la suma de vectores $\mathbf{r} + \mathbf{d}$.

La operación de *rotación* de un cierto ángulo respecto a un cierto eje, aplicada sobre un vector de posición dado, también puede describirse por medio de una operación matemática sencilla empleando matrices.

Como la rotación es una operación lineal, la transformación $\mathbf{r} \rightarrow \mathbf{r}'$ resultado de aplicar una rotación puede escribirse en forma matricial: $\mathbf{r}' = A \cdot \mathbf{r}$. Está claro que al aplicar una rotación sobre un vector la longitud del vector \mathbf{r} no cambia, lo que exige que la matriz A , con la que describimos esa rotación, sea *ortogonal*, es decir, la matriz traspuesta debe ser igual a la matriz inversa ($A^T = A^{-1}$).

El ejemplo más sencillo de matriz de rotación en 3 dimensiones es la matriz de rotación de ángulo ϕ alrededor del eje z

$$A = \begin{pmatrix} \cos \phi & \sin \phi & 0 \\ -\sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Aplicada sobre el vector \mathbf{r} esta matriz genera la transformación $\mathbf{r}' = A \cdot \mathbf{r}$, dada por:

$$\begin{aligned} x' &= x \cos \phi + y \sin \phi \\ y' &= -x \sin \phi + y \cos \phi \\ z' &= z \end{aligned}$$

que es el resultado de rotar el vector \mathbf{r} un ángulo ϕ respecto al eje z . Por ejemplo, si aplicamos esta matriz sobre el vector unitario $\mathbf{i} = (1, 0, 0)$ obtenemos¹ el vector $(\cos \phi, -\sin \phi, 0)$ si lo aplicamos sobre el vector unitario $\mathbf{j} = (0, 1, 0)$ obtenemos $(\sin \phi, \cos \phi, 0)$ y si lo aplicamos sobre $\mathbf{k} = (0, 0, 1)$ vemos que este vector permanece invariante, como cabía esperar.

La matriz de rotación A tiene varias propiedades interesantes. En primer lugar podemos comprobar que es ortogonal (su matriz traspuesta es también su inversa), una vez comprobado esto es inmediato darse cuenta de que la matriz inversa de A es igual a la matriz A cambiando ϕ por $-\phi$. Evidentemente, la operación inversa a aplicar una rotación de ángulo ϕ es aplicar una rotación de ángulo $-\phi$. Por último es inmediato observar que la matriz A se reduce a la matriz identidad cuando el ángulo de rotación ϕ es nulo, lógicamente. Estas propiedades son comunes a todas las matrices de rotación.

Al operar con matrices de rotación hay un detalle importante que conviene aclarar. Dada una matriz de rotación se pueden hacer dos cosas diferentes: Por un lado podemos aplicar dicha matriz de rotación *sobre los vectores de la base*, generando de esta manera un *cambio de base*, o podemos aplicar esta matriz de rotación sobre los vectores del espacio, sin cambiar la base que teníamos. Esto último es lo que hemos hecho más arriba al aplicar la matriz A sobre el vector \mathbf{r} .

- En el primer caso, si generamos un cambio de base los vectores del espacio no cambian, pero sus componentes en la nueva base son diferentes a las componentes que tenían respecto de la base antigua. Para ver cuáles serían las componentes de los vectores respecto de la base nueva situémonos en el punto de vista de la base. Al aplicar el cambio de base esta gira, a medida que la base gira un cierto ángulo α (respecto a un cierto eje) desde el punto de referencia de la base veremos que todos los vectores

¹MATLAB es suficientemente listo como para entender que, si se aplica la matriz por la izquierda al vector, es porque éste se considera un vector columna a efectos de la operación de multiplicación; en sentido estricto, esta operación sería incorrecta, pero MATLAB traspone automáticamente el vector.

del espacio rotan un ángulo $-\alpha$ respecto del mismo eje. Por tanto, para calcular las componentes de estos vectores en la nueva base lo que tenemos que hacer es aplicar sobre dichos vectores la matriz de rotación de ángulo $-\alpha$ respecto al eje de rotación dado, es decir, la matriz inversa (o, equivalentemente, la transpuesta) de la matriz que hemos empleado para generar el cambio de base.

- En la segunda posibilidad todo es mucho más sencillo. La base del espacio no cambia y sencillamente aplicamos la matriz de rotación sobre los vectores del espacio, obteniendo al hacerlo las correspondientes coordenadas de dichos vectores rotados un ángulo α respecto al eje de rotación que sea, y referidos a la misma base que teníamos al principio.

Aunque hemos ilustrado el tema de las matrices de rotación con un caso en el espacio tridimensional habitual, la discusión anterior es válida independientemente de la dimensionalidad del espacio en que estemos operando, 2D, 3D, ..., n D.

Pasamos ahora a comprobar las propiedades de la transformación A por medio de MATLAB. En primer lugar dado que la matriz de rotación depende del ángulo de rotación ϕ , y éste puede tomar valores arbitrarios, vamos a definir la matriz $A(\phi)$ como una función de ϕ . Para ello, podemos escribir un script en el que nuestro ángulo phi esté predefinido (luego lo podremos cambiar):

```
clear;% Limpiemos el workspace antes de empezar, una buena práctica
phi=pi/2;% por ejemplo  $\phi = \pi/2$ 
A= [ cos(phi) sin(phi) 0 ;
    -sin(phi) cos(phi) 0;
    0 0 1]
```

Comprobamos ahora que esta matriz es ortogonal. Para ello, debemos evaluar su inversa y su transpuesta, que en MATLAB corresponden a los comandos `inv` y `'`, con el resultado esperado de ortogonalidad

```
>> inv(A) junto al comando >> A'
```

3.6.3. Autovalores y Autovectores

El cálculo de autovalores y autovectores es una de las tareas más frecuentes en la actividad de un físico, sea cual sea el campo en el que trabaje. Todos los paquetes de álgebra y cálculo incorporan herramientas para este tipo de operaciones, cuya eficacia está limitada siempre por la dimensión de la matriz con la que trabajemos, con matrices más o menos pequeñas esto siempre es muy fácil de hacer, pero con matrices muy grandes (p. ej. $10^6 \times 10^6$) esto se convierte en algo realmente difícil.

Tomemos como un primer ejemplo el cálculo de autovalores y autovectores de una matriz simétrica. Según la teoría de matrices todos sus autovalores deben ser reales y los autovectores mutuamente perpendiculares. Definimos la matriz bajo estudio:

```
>> A = [0 1 0; 1, 0, 0; 0, 0, 0]
```

Los autovalores son las raíces de la ecuación característica

$$\det(A - \lambda I) = 0$$

En MATLAB se puede resolver este problema de manera simbólica, como se verá más adelante, pero si queremos obtener los autovalores numéricos de A , bastaría con escribir `eig(A)`, que nos devuelve un vector con los 3 autovalores $\lambda = -1, \lambda = 1, \lambda = 0$.

```
clear;% Limpiemos el workspace
>> A = [0 1 0; 1, 0, 0; 0, 0, 0];
ans=
-1
0
1
```

Comprobamos que todos son reales, y al ser distintos, decimos que no existe degeneración². Para calcular los autovectores recordamos que para cada autovalor λ , el autovector correspondiente debe satisfacer

$$(A - \lambda I) \cdot v_\lambda = 0$$

es decir, $A \cdot v_\lambda = \lambda v_\lambda$.

Para calcular cada autovector en MATLAB basta pedir una salida adicional al comando `eig(A)`.

Por ejemplo:

```
>> [Autovector, Autoval] = eig(A)
```

nos devuelve 2 matrices,

```
Autovector=

-0,7071    0    0,7071
 0,7071    0    0,7071
 0         1         0

Autoval=

-1    0    0
 0    0    0
 0    0    1
```

donde cada autovalor se sitúa en la diagonal de la matrix `Autoval`.

Este resultado nos indica que el autovector del autovalor $\lambda = -1$ es la recta con vector director $v_{-1} = (-1/\sqrt{2}, 1/\sqrt{2}, 0)$, el autovector normalizado correspondiente al autovalor $+1$ es $v_{+1} = (1/\sqrt{2}, 1/\sqrt{2}, 0)$ y, finalmente para el tercer autovalor encontramos $v_0 = (0, 0, 1)$, que son las columnas correspondientes de la matrix `Autovector`. Observamos que estos tres subespacios propios son mutuamente ortogonales, tal y como esperábamos de una matriz real y simétrica con autovalores distintos.

²Decimos que existe degeneración de los autovalores cuando podemos tener dos autovectores linealmente independientes correspondientes al mismo autovalor. En Física habitualmente asociamos cada autovector con un estado físico diferente, por lo que un autovalor es degenerado cuando no sabemos diferenciar entre dos estados basándonos sólo en el valor numérico de ese autovalor repetido.

Una de las aplicaciones más importantes del cálculo de autovalores y autovectores es la diagonalización de matrices. En el caso anterior, dado que los tres autovectores son linealmente independientes, la matriz A admite una representación diagonal en la forma

$$A' = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Con ayuda de los autovectores es fácil determinar qué matriz P define este cambio de base, en la forma

$$A' = P^{-1} \cdot A \cdot P$$

La matriz P está dada por

$$P = (\mathbf{v}_{-1}, \mathbf{v}_1, \mathbf{v}_0)$$

es decir, está formada por los autovectores como columnas, siendo irrelevante la normalización de los autovectores, tal y como puede comprobarse fácilmente. Con MATLAB el cálculo quedaría como sigue. En primer lugar ensamblamos P a partir de la matriz `Autovec`:

```
>> P= [Autovec(:,1) Autovec(:,3) Autovec(:,2)]
```

Y posteriormente ejecutamos las operaciones de cambio de base:

```
Ap=
```

$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Otro cálculo interesante es verificar que el espacio generado por los tres autovectores linealmente independientes se corresponde con todo el espacio euclídeo de tres dimensiones; para ello basta demostrar, y se deja como ejercicio, que la matriz I definida como la expansión matricial en los tres autovectores normalizados

$$I = \mathbf{v}_{-1}\mathbf{v}_{-1}^T + \mathbf{v}_1\mathbf{v}_1^T + \mathbf{v}_0\mathbf{v}_0^T$$

es la matriz identidad 3×3 . Por tanto, para cualquier vector \mathbf{r} del espacio euclídeo, tenemos

$$\mathbf{r} = I \cdot \mathbf{r} = \mathbf{v}_{-1}\mathbf{v}_{-1}^T \cdot \mathbf{r} + \mathbf{v}_1\mathbf{v}_1^T \cdot \mathbf{r} + \mathbf{v}_0\mathbf{v}_0^T \cdot \mathbf{r}$$

que podemos escribir como el desarrollo del vector \mathbf{r} en la base formada por los tres autovectores normalizados

$$\mathbf{r} = x_{-1}\mathbf{v}_{-1} + x_1\mathbf{v}_1 + x_0\mathbf{v}_0$$

donde $x_i = \mathbf{v}_i^T \cdot \mathbf{r}$ es el producto escalar, en notación matricial.

Con estos ejemplos hemos visto todo lo necesario para empezar a manejar el lenguaje de programación MATLAB. La mejor forma de aprender cualquier lenguaje de programación es usándolo para resolver problemas concretos, de modo que en lo que sigue indicaremos algunos problemas especialmente representativos que pueden resolverse con MATLAB.

3.6.4. Cambios de Base

3.6.4.1. Aplicación de cambios de base sobre vectores

En un espacio vectorial un vector queda descrito por sus componentes respecto de una determinada base del espacio vectorial, p. ej. el vector $v = (1, 2, 3)$ es el que resulta de desplazarse una unidad en la dirección x , 2 en y y 3 en z . En el ejemplo anterior hemos empleado la *base canónica*, que es la formada por los vectores

$$\left. \begin{array}{l} e_1 = (1, 0) \\ e_2 = (0, 1) \end{array} \right\} \text{ en 2 dimensiones}$$

$$\left. \begin{array}{l} e_1 = (1, 0, 0) \\ e_2 = (0, 1, 0) \\ e_3 = (0, 0, 1) \end{array} \right\} \text{ en 3 dimensiones}$$

$$\left. \begin{array}{l} e_1 = (1, 0, 0, \dots, 0) \\ e_2 = (0, 1, 0, \dots, 0) \\ \dots \quad \dots \quad \dots \quad \dots \\ e_n = (0, 0, \dots, 0, 1) \end{array} \right\} \text{ en } n \text{ dimensiones}$$

Aparte de la base canónica cualquier conjunto de n vectores *linealmente independientes* forma una base válida del espacio. Aparece entonces la cuestión:

- ★ **dadas las componentes de un vector (que existe independientemente de la base) respecto de una base cualquiera ¿cómo se calculan las componentes de este vector respecto de otra base distinta?**

Como el objetivo de esta asignatura es solamente aprender a programar, damos a continuación las instrucciones detalladas sobre cómo se hacen cambios de base para el caso de dimensión n arbitraria, sin entrar en demostraciones. De todas formas, para resolver los ejercicios recomendamos comenzar con cosas más sencillas, resolviendo primero casos con $n = 2$, o $n = 3$ y, cuando eso ya esté superado, generalizándolos a dimensión arbitraria.

1. Cualquier vector v se expresa en términos de la base de partida como

$$v = v_1 e_1 + v_2 e_2 + \dots + v_n e_n = \sum_{i=1}^n v_i e_i = v_i e_i$$

donde los números v_i ($i = 1, \dots, n$) son las coordenadas de v respecto de la base formada por los e_i . En cálculo con matrices y vectores se usan constantemente expresiones como la anterior, en la que se hace una suma respecto de un índice (p. ej. i en la ecuación de arriba), para simplificar estas expresiones se suele usar el llamado:

- ★ **Convenio de suma de Einstein:**

Cuando en una expresión aparece un índice repetido la expresión representa la suma respecto de ese índice para todos los valores posibles del índice, p.ej.:

- Desarrollo en componentes de un vector

$$v = \sum_{i=1}^n v_i e_i = v_i e_i$$

- Producto escalar

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^n u_i v_i = u_i v_i$$

- Producto de dos matrices

$$(A \cdot B)_{ij} = \sum_{k=1}^n A_{ik} B_{kj} = A_{ik} B_{kj}$$

- Producto de muchas matrices

$$(A \cdot B \cdot C \cdot D)_{ij} = \sum_{\alpha=1}^n \sum_{\beta=1}^n \sum_{\gamma=1}^n A_{i\alpha} B_{\alpha\beta} C_{\beta\gamma} D_{\gamma j} = A_{i\alpha} B_{\alpha\beta} C_{\beta\gamma} D_{\gamma j}$$

- Traza de una matriz

$$\text{tr} A = \sum_{i=1}^n A_{ii} = A_{ii}$$

- El índice que aparece repetido se llama *índice mudo*, mientras que los demás índices que aparezcan en la expresión se llaman *índices libres*. Como la expresión con un índice mudo realmente representa la suma para todos los valores posibles del índice mudo, está claro que el resultado será el mismo independientemente de la letra con que designemos al índice mudo (y que no debe coincidir con ninguna de las empleadas para los índices libres). P. ej. $\mathbf{u} \cdot \mathbf{v} = u_i v_i = u_j v_j = u_\alpha v_\alpha = \dots$
- El convenio de suma de Einstein se usa con muchísima frecuencia para escribir expresiones de este tipo de una manera rápida, sin sumatorios. En lo sucesivo suponemos que se aplica este convenio a menos que se diga lo contrario.

2. Cuando se aplica un cambio de base, la base “nueva” está formada por los vectores $\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \dots, \mathbf{t}_n$, cuya expresión en términos de la base antigua suponemos conocida.
3. La matriz del cambio de base (C) es la matriz cuadrada formada por los *vectores columna* \mathbf{t}_i (con $i = 1, \dots, n$) expresados en términos de la base antigua

$$C = (\mathbf{t}_1 \quad \mathbf{t}_2 \quad \mathbf{t}_3 \quad \dots \quad \mathbf{t}_n)$$

Es muy fácil comprobar que cada uno de los vectores \mathbf{t}_i se obtiene al aplicar la matriz del cambio sobre la base de partida en la forma

$$\mathbf{t}_j = \mathbf{e}_i C_{ij}, \quad i, j = 1, \dots, n$$

por eso se define de esta forma la matriz del cambio C .

4. Si los vectores \mathbf{t}_i son linealmente independientes entonces C es invertible, de donde deducimos

$$\mathbf{e}_j = \mathbf{t}_i (C^{-1})_{ij}, \quad i, j = 1, \dots, n$$

donde C^{-1} es la inversa de la matriz C (si los \mathbf{t}_i no son linealmente independientes entonces no forman una base).

5. Sustituyendo esta última relación en

$$\mathbf{v} = v_i \mathbf{e}_i = \hat{v}_i \mathbf{t}_i$$

encontramos finalmente las relaciones

$$v_i = C_{ij} \hat{v}_j \quad \hat{v}_i = (C^{-1})_{ij} v_j$$

que indican cómo se relacionan las coordenadas de un vector expresadas en dos bases distintas.

3.6.5. Aplicación de cambios de base sobre aplicaciones lineales

De forma similar a como sucedía con los vectores, las aplicaciones lineales definidas sobre un espacio vectorial también se describen por medio de sus componentes respecto de una base del espacio. En el caso de las aplicaciones lineales estas componentes forman una matriz, cuyas componentes son las coordenadas de los vectores que resultan de aplicar la aplicación lineal sobre cada uno de los vectores de la base del espacio vectorial.

* ¿Cómo se calculan las componentes de una matriz respecto de la nueva base?

A partir de las relaciones anteriores es muy sencillo deducir la regla de transformación de las aplicaciones lineales.

1. Supongamos una aplicación lineal A , tal que aplicada sobre el vector \mathbf{u} nos da el vector \mathbf{v}

$$A\mathbf{u} = \mathbf{v}$$

2. Si escribimos esto en componentes respecto de la base de partida tendremos

$$A_{ij} u_j = v_i$$

donde A_{ij} es la matriz de la aplicación lineal A en la base de partida. Suponemos que la matriz A_{ij} es un dato que conocemos. Queremos calcular la forma de esta matriz en la base nueva.

3. La matriz de A en la base nueva (vamos a denotarla por \hat{A}_{ij}) cumplirá una expresión análoga a la anterior pero con los vectores \mathbf{u} y \mathbf{v} referidos a la base nueva, es decir

$$\hat{A}_{ij} \hat{u}_j = \hat{v}_i$$

4. Sustituyendo en esta ecuación la regla de transformación de los vectores deducimos directamente

$$A_{ij} = C_{ik} \hat{A}_{kl} (C^{-1})_{lj} \quad \hat{A}_{ij} = (C^{-1})_{ik} A_{kl} C_{lj}$$

3.7. Problemas

3.7.1. Problemas propuestos

1. Escriba una función que aplique cambios de base sobre vectores.
 - input: coordenadas del vector en la base estándar, vectores que forman la nueva base.
 - output: coordenadas del vector en la nueva base.
2. Escriba un programa que aplique cambios de base sobre matrices.
 - input: coordenadas de la matriz correspondiente a una aplicación lineal en la base estándar, vectores que forman la nueva base.
 - output: coordenadas de la matriz correspondiente a la aplicación lineal en la nueva base.
3. Invierta las anteriores relaciones para escribir una función que genere las componentes de vectores y matrices respecto de la base estándar a partir de sus componentes respecto a una base arbitraria.
 - input: coordenadas del vector o de la matriz respecto de una base arbitraria, vectores que forman la base arbitraria respecto de la base estándar.
 - output: coordenadas del vector o de la matriz en la base estándar.