INFORMÁTICA Y PROGRAMACIÓN

Grado en Ingeniería Química

GUION DE PRÁCTICAS
2020/21

22 de mayo de 2021

Modesto Castrillón Santana, Abraham Rodríguez Rodríguez
Universidad de Las Palmas de Gran Canaria
Escuela de Ingenierías Industriales y Civiles

Esta obra está bajo licencia de Creative Commons Reconocimiento - No Comercial 4.0 Internacional

Índice general

1. Entorno de trabajo. Uso básico de Matlab como herramienta de cálculo **1**

1.1. Introducción 1

1.2. Primeros pasos con Matlab 3

1.2.1. Interfaz básica 4

1.2.2. Operaciones básicas 5

1.2.3. Variables 6

1.2.4. Funciones matemáticas 8

1.3. Cuenta Mathworks y Matlab online 10

1.4. Ejercicios propuestos como actividad independiente 11

1.5. Tarea 12

2. Matlab como calculadora matricial **15**

2.1. Introducción 15

2.2. Vectores y matrices 15

2.3. Operador de rangos 20

2.4. Editor de programas de Matlab 22

2.5. Ejercicios 24

2.5.1. De apoyo durante la sesión 24

2.5.2. Propuestos como actividad independiente 25

2.6. Tarea 26

3. Matlab como herramienta de visualización gráfica **27**

3.1. Introducción 27

3.2. Gráficos de bajo nivel 27

3.3. Gráficos de alto nivel 29

3.3.1. Gráficos 2D 29

3.3.2. Gráficos 3D 38

3.4. Gráficos estadísticos 39
3.5. Editor de programas de Matlab 41
3.6. Ejercicios 42
3.6.1. De apoyo durante la sesión 42
3.6.2. Propuestos como actividad independiente 43
3.7. Tarea 45

4. Programación básica **47**
4.1. Introducción 47
4.2. Lectura de datos 47
4.3. Control del formato de salida por defecto 48
4.4. Escritura de datos 49
4.4.1. Particularidades con vectores y matrices 51
4.4.2. Equivalencia entre *disp* y *fprintf* 52
4.5. Operaciones básicas 53
4.6. Ejercicios 53
4.6.1. De apoyo durante la sesión 53
4.6.2. Propuestos como actividad independiente 54
4.7. Tarea 55

5. Sentencias de control condicionales **57**
5.1. Introducción 57
5.2. Sentencia condicional 57
5.3. Ejercicios 60
5.3.1. De apoyo durante la sesión 60
5.3.2. Propuestos como actividad independiente 61
5.4. Tarea 62

6. Sentencias de control repetitivas **63**
6.1. Introducción 63
6.2. Sentencia repetitiva 63
6.3. Anidación de sentencias 65
6.3.1. Anidando repetitivas y condicionales 65
6.3.2. Anidando repetitivas 66
6.4. Estructura de acumulador 67
6.5. Algunos extras 68
6.5.1. Parte entera y resto 69

6.6. Ejercicios 70
6.6.1. De apoyo durante la sesión 70
6.6.2. Propuestos como actividad independiente 71
6.7. Tarea 73

7. Estructuras de datos dimensionadas: Vectores 77
7.1. Introducción 77
7.1.1. Creación/inicialización 77
7.1.2. Acceso 78
7.2. Ejercicios semana I 79
7.2.1. De apoyo durante la sesión 79
7.2.2. Propuestos como actividad independiente 80
7.3. Depuración 81
7.4. Ejercicios semana II 84
7.4.1. De apoyo durante la sesión 84
7.4.2. Propuestos como actividad independiente 87
7.5. Tarea 89

8. Estructuras de datos dimensionadas: Matrices 91
8.1. Introducción 91
8.2. Creación/inicialización 91
8.2.1. Acceso 92
8.3. Ejercicios 93
8.3.1. De apoyo durante la sesión 93
8.3.2. Propuestos como actividad independiente 94
8.4. Tarea 95

9. Estructuras de datos dimensionadas: Cadenas de caracteres 97
9.1. Introducción 97
9.2. Creación/inicialización 97
9.3. Trabajando con cadenas de caracteres 98
9.4. Ejercicios 100
9.4.1. De apoyo durante la sesión 100
9.4.2. Propuestos como actividad independiente 100
9.5. Tarea 101

10. Acceso a datos en archivo **103**
10.1. Introducción . . . 103
10.2. Manejo de ficheros a bajo nivel . . . 103
10.3. Ejercicios semana I . . . 108
10.3.1. De apoyo durante la sesión . . . 108
10.3.2. Propuestos como actividad independiente . . . 109
10.4. Manejo de archivos a alto nivel . . . 110
10.4.1. Datos numéricos . . . 110
10.4.2. Datos no únicamente numéricos . . . 112
10.4.3. Repositorios de datos . . . 119
10.5. Ejercicios semana II . . . 120
10.5.1. De apoyo durante la sesión . . . 120
10.5.2. Propuestos como actividad independiente . . . 121
10.6. Tarea . . . 121

11. Programación estructurada **123**
11.1. Introducción . . . 123
11.2. Definición de funciones . . . 125
11.3. Visualizando una matriz . . . 126
11.4. Ejercicios . . . 129
11.4.1. De apoyo durante la sesión . . . 129
11.4.2. Propuestos como actividad independiente . . . 131
11.5. Tarea . . . 132

12. Utilidades para la ingeniería **133**
12.1. Introducción . . . 133
12.2. Interpolación . . . 133
12.3. Polinomios . . . 134
12.4. Búsqueda de ceros de una función . . . 135
12.5. Resolución de sistemas de ecuaciones lineales . . . 136
12.6. Integración de funciones . . . 137
12.7. Ecuaciones diferenciales ordinarias . . . 138
12.8. Computación simbólica . . . 138
12.9. Ajuste por mínimos cuadrados . . . 139
12.10. Simulación con imágenes no estáticas . . . 140
12.10.1. Puntos aleatorios . . . 140

12.10.2 Caída por gravedad 143

Práctica 1

Entorno de trabajo. Uso básico de Matlab como herramienta de cálculo

1.1. Introducción

Primera sesión práctica de la asignatura de Informática y Programación del Grado en Ingeniería Química de la ULPGC. Con el apoyo de la biliografía básica de la asignatura Castrillón Santana et al. [2011], las sesiones prácticas profundizan en el objetivo de dotarte de habilidades en un entorno real informático. Nuestro entorno de trabajo lo conforman ordenadores conectados a la red. Una red de ordenadores se compone de dos o más ordenadores interconectados entre sí, con capacidad para comunicarse a través de un enlace de datos determinado. La complejidad de la red depende del número de ordenadores que conecte. Internet, también muchas veces denominada *la Red*, es una red de redes a nivel mundial. Internet consiste en un conjunto heterogéneo de redes de ordenadores interconectadas, de forma análoga a como las redes tradicionales de telefonía conmutada se encuentran conectadas entre sí a lo largo de todo el mundo. De la misma forma que es posible realizar una llamada telefónica internacional a cualquier país, también es posible acceder a cualquier información disponible, a través de alguna de las redes interconectadas a Internet, independientemente del país del mundo donde se encuentre.

Para poder ser identificado en Internet, cada ordenador conectado debe tener un "número identificativo" que se denomina dirección IP. La dirección IP está formada por cuatro números, todos ellos entre 1 y 255, separados por puntos. Por ejemplo la dirección IP del ordenador que tiene la página web de la ULPGC, www.ulpgc.es, es: *193.145.138.32* (lo puedes averiguar desde una terminal o símbolo del sistema, en Windows puedes lanzarlo buscando *cmd*, en la que tecleas *ping www.ulpgc.es*). Sin embargo, para evitar tener que memorizar

números al querernos conectar, a cada ordenador con dirección IP se le asigna un nombre o alias. En un navegador web, llegaremos a la misma página web tecleando en la zona de dirección www.ulpgc.es, ulpgc.es o *193.145.138.32*. En la consola de Windows, el comando *nslookup* te permite ver las equivalencias de dirección IP y nombre, y viceversa, con el Sistema de Nombres de Dominio ("Domain Name System" o DNS) establecido, p.e. *nslookup www.ulpgc.es* o *nslookup 193.145.138.32*.

Como bien sabes a través de la red puedes acceder a la enorme y creciente cantidad de contenidos:

- páginas hipertexto,
- documentos,
- imágenes,
- música,
- películas,
- software,
- bases de datos,
- y en general, todo tipo de contenido.

Mediante el uso de navegadores es posible acceder tanto a cualquier página web disponible en Internet, así como a otros documentos a través de los hiperenlaces contenidos en los distintos documentos a los que se acceden. Es preciso aquí aclarar que estos documentos enlazados pueden encontrarse en cualquier punto de la red independientemente de donde se encuentre situada físicamente la página de partida. A este proceso de consulta y acceso a través del enorme volumen de páginas web hiperenlazadas, que hay actualmente en la Red es lo que se conoce normalmente como navegar por la red o simplemente navegar.

Se define una dirección web o url (de la siglas e inglés de "uniform resource locator") como la dirección donde se encuentra una página web o documento hipertexto en la Red. La web *https://www2.ulpgc.es/plan-estudio/4026* permite acceder a la información disponible en el servidor web de la ULPGC (www. ulpgc.es), acerca del Grado en Ingeniería Química. Una dirección web está dividida habitualmente en tres partes:

- El prefijo que indica el protocolo (p.e. *HyperText Transfer Protocol*, *File Transfer Protocol*, etc.) y aparece en la url al comienzo en este caso como: http://

- www.ulpgc.es el nombre o alias de la computadora, servidor, máquina u ordenador donde se encuentra el documento.

- /plan-estudio/4026 describe dónde, dentro de la máquina, se encuentra el fichero que se está visualizando.

La parte de la dirección web correspondiente a la máquina, sigue un sistema de nombrado, el ya mencionado Sistema de Nombres de Dominio . Éste consiste en formar el nombre de un computador mediante varias palabras concatenadas mediante el símbolo ".". Cada una de estas palabras especifica el nombre de una máquina dentro de diferentes dominios de máquinas a diferentes niveles. Así comenzando de izquierda a derecha, la primera palabra identifica la máquina, la segunda indica el dominio de máquinas donde esta se encuentra, la tercera indicará un dominio de máquinas que incluya al dominio precedente. Esta concatenación de dominios continúa hasta llegar a la palabra más a la derecha que indica normalmente el dominio de máquinas de un país – España (.es), Francia (.fr), Alemania (.de), Reino Unido (.uk), Portugal (.pt), Suecia (.se), etc. -, o a qué tipo de organización pertenecen dichas máquinas – Instituciones Educativas (.edu), Negocios (.com), Organizaciones (.org), Entidades Internacionales (.int), etc.

1.2. PRIMEROS PASOS CON MATLAB

Tras una breve introducción a internet, nos centramos en el paquete software que se utilizará durante el curso. MATLAB (abreviatura de "MATrix LABoratory", que podemos traducir por laboratorio de matrices) es un software matemático que ofrece un entorno de desarrollo integrado con su propio lenguaje de programación, denominado lenguaje M. El núcleo de Matlab está basado en un intérprete que realiza cálculos numéricos con vectores y matrices, de ahí su nombre, permitiendo trabajar además con números escalares, tanto reales como complejos, cadenas de caracteres y otras estructuras de datos más elaboradas.

Matlab está estructurado en torno a una biblioteca o librería de funciones elementales sobre la que se han ido añadiendo otras especializadas para diferentes ámbitos de aplicación. Son las denominadas *toolboxes*, y se van extendiendo en número y funcionalidad con cada nueva versión. Nos centraremos en los aspectos básicos de Matlab, puesto que nos interesa utilizarlo como lenguaje de programación.

Octave[1] es una alternativa de software libre, incorporando la mayor parte de sus capacidades. En este documento se hace referencia a Matlab en las explicaciones por simplicidad, aunque en su gran mayoría son trasladables directamente a Octave.

[1] http://www.gnu.org/software/octave/

1.2.1. Interfaz básica

Es posible iniciar Matlab como cualquier otra aplicación, haciendo doble clic en el icono correspondiente en el escritorio. Al iniciar el programa, se abre una ventana similar a la mostrada en la Fig. 1.1, aunque su aspecto puede variar dependiendo de la versión del programa y de la personalización del usuario. Se puede recuperar la vista original, en cualquier momento, con la opción *Layout/Default*, en la pestaña *Home*.

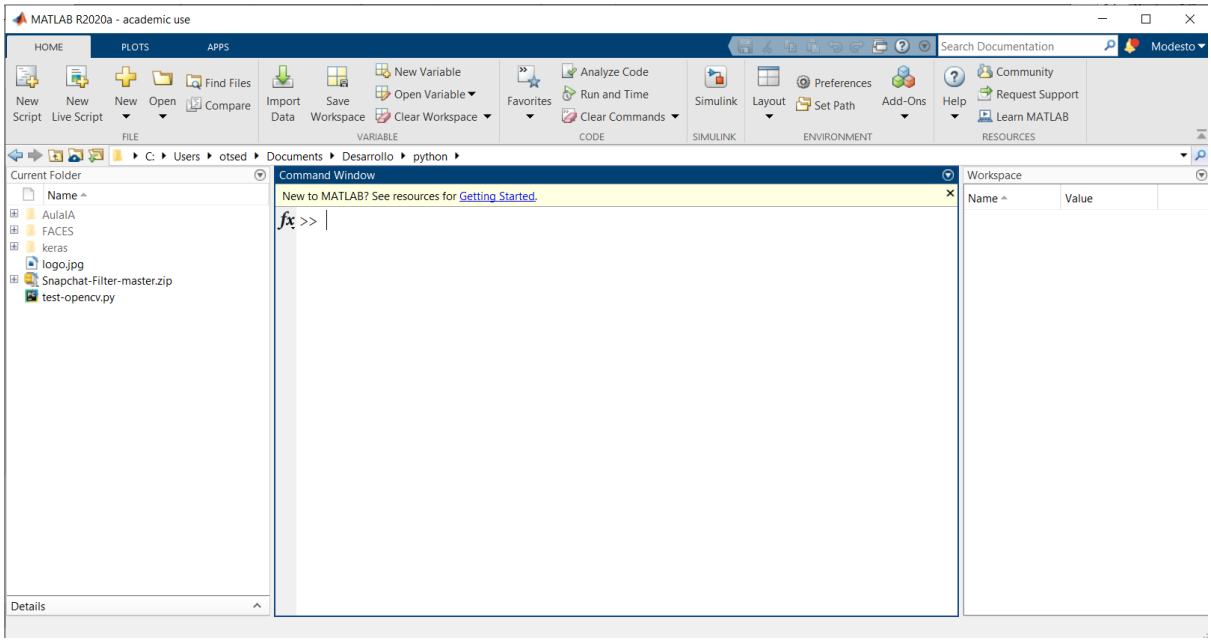


Figura 1.1: Interfaz por defecto de Matlab

Sobre la interfaz de la aplicación, se aprecian distintas zonas claramente diferenciadas:

- La parte superior, donde se encuentran los diversos menús y barras de botones.
- La parte central, denominada ventana de comandos o *Command Window*, donde se pueden ejecutar comandos de Matlab a través del *prompt* del intérprete (símbolo >>).
- Las zonas laterales, que se estructuran a su vez en diferentes paneles:
    - A la izquierda el directorio actual o *Current Directory*, que lista los ficheros del directorio donde se encuentra el programa en esos momentos.
    - A la derecha el espacio de trabajo o *Workspace*, que recoge las variables existentes en memoria, es decir, que están en uso.

El directorio actual o de trabajo es la carpeta donde Matlab busca en primer lugar los comandos que introduce el usuario. En la misma Fig. 1.1 y destacado en la Fig. 1.2, se puede leer en la parte superior de la ventana el nombre de la carpeta actual o directorio de trabajo (en el ejemplo *C:/Users/otsed/Documents/Desarrollo/python*). El directorio actual es modificable, siendo posible tanto recuperar una carpeta recientemente utilizada, como buscar una de forma manual utilizando los botones situados inmediatamente a la derecha.

Otra forma de identificar el directorio actual es tecleando en la ventana de comandos pwd (equivale a *Print Working Directory*). Si se desea modificar manualmente el directorio actual entonces debe utilizarse el comando cd (equivale a *Change Directory*).

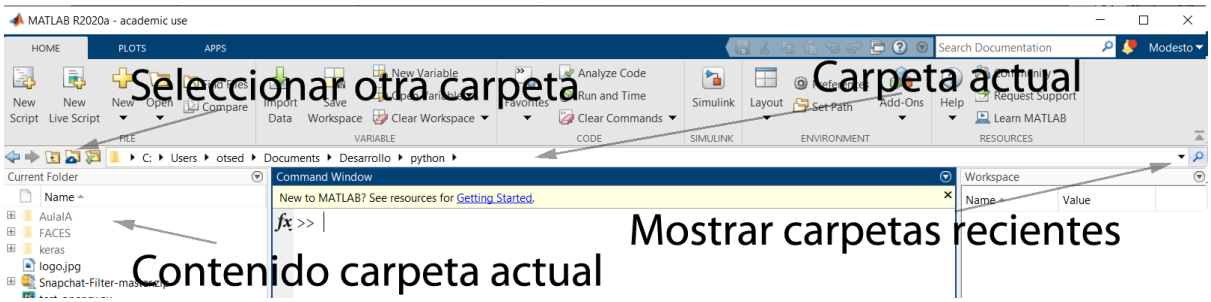


Figura 1.2: Zonas resaltadas de la interfaz

El asistente de ayuda es un elemento muy valioso en Matlab. Se suele utilizar para recordar la descripción de algunas funciones o los parámetros que estas necesitan para poder operar. Se puede acceder a la ayuda de dos formas distintas:

1. A través del botón del menú de ayuda de la ventana principal, *Help*.

2. Por medio del comando *help* a través de la ventana de comandos, tecleando help y el comando del que se desee obtener información. Por ejemplo, la secuencia *help cd* proporciona la información pertinente al comando *cd*.

1.2.2. Operaciones básicas

Desde la ventana de comandos, tanto en Matlab como Octave, podemos realizar operaciones aritméticas. Podemos por ejemplo teclear:

```
>>2+2
ans =
    4
```

Con lo que, utilizando cualquier operador aritmético, podremos realizar todas las operaciones y con el número de operadores que creamos necesarias.

```
>>2*3+4
ans =
    10
```

Los principales operadores aritméticos disponibles son:

+ Suma
− Resta
∗ Multiplicación
/ División
^ Potencia
() Paréntesis

El uso de paréntesis permite establecer la prioridad en aquellas operaciones que combinen distintos operadores. Comprueba la diferencia entre $2*3+3$ y $2*(3+3)$

1.2.3. Variables

Las variables son estructuras de datos empleadas en programación para almacenar un valor que puede cambiar. Si bien conceptualmente se asemejan al concepto matemático de variable, podemos hacer un símil con una caja, que ofrece un espacio para almacenar un contenido, pudiendo alterarse su contenido en cualquier momento. Una variable podrá por ejemplo almacenar datos de entrada, intermedios, o resultados. Estas estructuras se corresponden con una zona de memoria del ordenador donde durante la ejecución de un programa se almacena el contenido de la variable. Las variables se caracterizan por tener un nombre, una dirección, un tipo y un valor.

- **Nombre**: Identificador o alias por el cual referenciamos la variable dentro del programa. A bajo nivel, nivel de la máquina, el nombre se traduce a una dirección de memoria. Un nombre nunca debe coincidir con ninguna palabra clave o reservada del lenguaje de programación que estemos utilizando.

- **Dirección**: Dirección de la memoria del ordenador donde está localizada la variable.

- **Valor**: El valor o contenido que contiene la variable en un momento dado.

- **Tipo**: El conjunto de datos que puede almacenar o contener la variable. El tipo de una variable fija su tamaño en memoria (bytes) y el tipo de operaciones que podemos hacer con ella. Los tipos básicos habituales son: entera, real, carácter, cadena, etc.

Si tecleas de nuevo en la línea de comando una operación:

```
>>2*32
ans =
    64
```

Puedes observar que en la segunda línea aparece *ans = 64*, textitans es una variable del entorno o espacio de trabajo que almacena la última operación realizada. Como ocurre con cualquier variable, puedes hacer uso de ella en otra operación, donde se recupera su valor para llevarla a cabo:

```
>>2*ans
ans =
    128
```

Tenemos de esta forma un nuevo valor almacenado en la variable ans. Es más interesante y potente definir nuestras propias variables y asignarles el contenido que nos interese. Para definir una variable x con el valor 10:

```
>>x=10
x =
    10
```

Tecleando

```
>>x*x
x =
    100
```

Si deseamos que esa operación se guarde en otra variable y no simplemente en ans

```
>>xcuad=x*x
xcuad =
    100
```

Si queremos evitar que cada vez que se realiza una operación se muestre el resultado, basta con añadir un ; al final del comando

```
>>xcuad=x*x;
```

En cualquier momento, tecleando en la línea de comando una variable del espacio de trabajo, recuperamos su valor actual.

```
>>xcuad
xcuad =
    100
```

El conjunto de variables existentes en un momento se puede ver a través de la pestaña *Workspace*, o tecleando en la ventana de comandos *who*. Por ilustrar con un ejemplo, si Usain Bolt tiene el récord mundial de los 200 metros lisos en 19.19 segundos, la velocidad promedio

en km/h podemos resolverla de la siguiente forma, recordando la expresión del cálculo de la velocidad v=e/t

```
>>e=200;
>>t=19.19;
>>e_km=e/1000;
>>t_horas=t/3600;
>>v=e_km/t_horas
v =
    37.5195
```

1.2.4. Funciones matemáticas

Tanto Matlab como Octave proporcionan una colección de funciones matemáticas elementales. Por ejemplo:

```
>>y=exp(0)
y =
    1
```

Que obtiene el exponencial del valor pasado entre paréntesis. En el caso de pasar como argumento una variable, se obtiene el resultado de aplicar la función al valor contenido en dicha variable. Un ejemplo con la función *sqrt* que calcula la raíz cuadrada:

```
>> sqrt(4)
ans =
     2

>> x=16;
>> sqrt(x)
ans =
     4
```

Un reducido listado de funciones habituales:

sin y **sind** seno de un ángulo indicado respectivamente en radianes o grados
cos y **cosd** coseno de un ángulo indicado respectivamente en radianes o grados
tan y **tand** tangente de un ángulo indicado respectivamente en radianes o grados
sec y **secd** cosecante
cot y **cotd** cotangente
exp exponencial
log logaritmo natural
sqrt raíz cuadrada
abs valor absoluto
round proporciona el entero más cercano

fix devuelve el entero más cercano

floor devuelve el entero menor más cercano

Toda función puede incluirse en cualquier expresión a través de la línea de comando

```
>> x=pi
x =
    3.1416

>> r=cos(x)+sin(x*2)
r =
   -1.0000
```

Incluimos una nota final sobre la posibilidad de trabajar en Matlab con números complejos, contando con las funciones *real*, *imag*, *abs* y *angle* que respectivamente devuelven su parte real, su parte imaginaria, su módulo y su fase o argumento.

```
>> n=2-3i
n =
   2.0000 - 3.0000i

>> real(n)
ans =
     2

>> imag(n)
ans =
    -3

>> abs(n)
ans =
    3.6056

>> angle(n)
ans =
   -0.9828

>> radtodeg(angle(n))
ans =
  -56.3099
```

Observa el comando final que muestra el uso anidado de funciones. La función *angle* devuelve la fase en radianes, si a dicho resultado le aplicamos *radtodeg* obtenemos su equivalencia en grados.

Recordarte que cuentas con el comando *help* para obtener información de uso de los comandos y funciones de Matlab.

```
>> help sqrt
sqrt - Square root
```

```
This MATLAB function returns the square root of each element of the array X.

B = sqrt(X)

See also nthroot, realsqrt, sqrtm

Documentation for sqrt
Other functions named sqrt
```

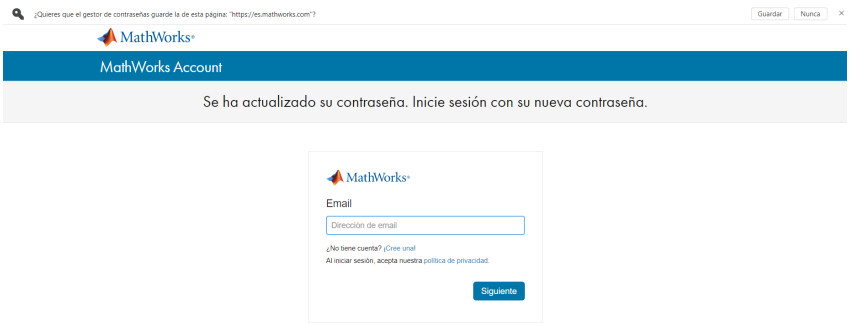


Figura 1.3: Pantalla de acceso a https://matlab.mathworks.com

1.3. Cuenta Mathworks y Matlab online

Si bien contamos con Matlab instalado en los equipos del laboratorio, es aconsejable disponer de un mecanismo para ejecutarlo desde casa. La ULPGC cuenta con una licencia Campus que permite la instalación en los ordenadores personales de la comunidad universitaria. Previamente es necesario crear una cuenta Mathworks. Para llevar a cabo la creación de dicha cuenta, y posterior instalación de Matlab, sugerir hacer uso de la guía editada por el Servicio de Informática para tal fin: Guía de instalación ULPGC[2].

También es posible acceder a Matlab online. Una vez creada la cuenta podemos acceder a nuestra espacio personal a través de la web[3], que antes de permitirme hacer uso de Matlab online, requerirá la verificación de mis credenciales.

La figura 1.3 muestra la pantalla de acceso a *Matlab account*. Si no te has registrado previamente, debes realizar el proceso haciendo uso del proceso definido en la mencionada anteriormente guía editada por la ULPGC. En mi caso, tras acceder llegué a una pantalla como la mostrada en la figura 1.4. Para poder descargar o incluso utilizar *Matlab online*, debes

[2]https://si.ulpgc.es/sites/default/files/ficheros-si/manuales-y-ayudas/instrucciones_para_instalar_matlab.pdf
[3]https://matlab.mathworks.com

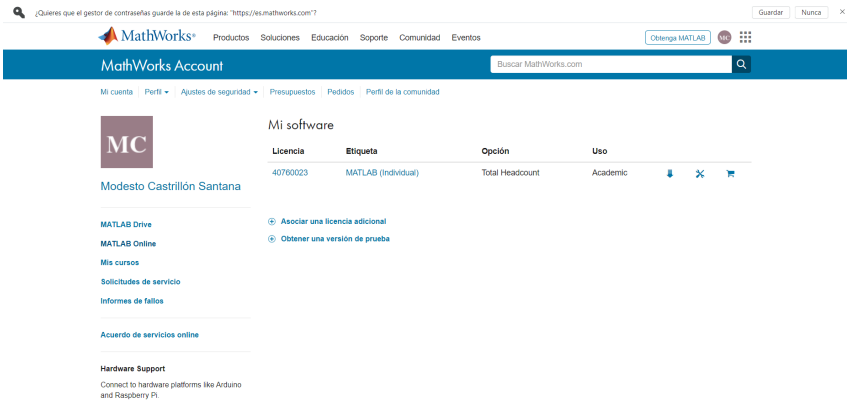


Figura 1.4: Pantalla tras acceder a mi cuenta Mathworks

Figura 1.5: Acceso a la verificación

verificar tus credenciales de la universidad, durante el proceso accederás a una página como la de la figura 1.5, que posteriormente requerirá introducir tus credenciales de la ULPGC, figura 1.6. Una vez verificado, deberías tener acceso tanto a la descarga, como a la versión online de Matlab, con la interfaz mostrada en la figura 1.7 similar a la vista en secciones previas.

1.4. EJERCICIOS PROPUESTOS COMO ACTIVIDAD INDEPENDIENTE

1. ¿Cómo calcularías en lenguaje m la expresión $\frac{1}{2}+\frac{3}{8}$?

2. ¿Utiliza la ventana de comandos de Matlab para calcular $(2+3)^2+3\left(\frac{1+4}{5-3}\right)$?

3. ¿Por qué los resultado de *cos(0)* y *cosd(0)* son idénticos, y no lo son los resultado de

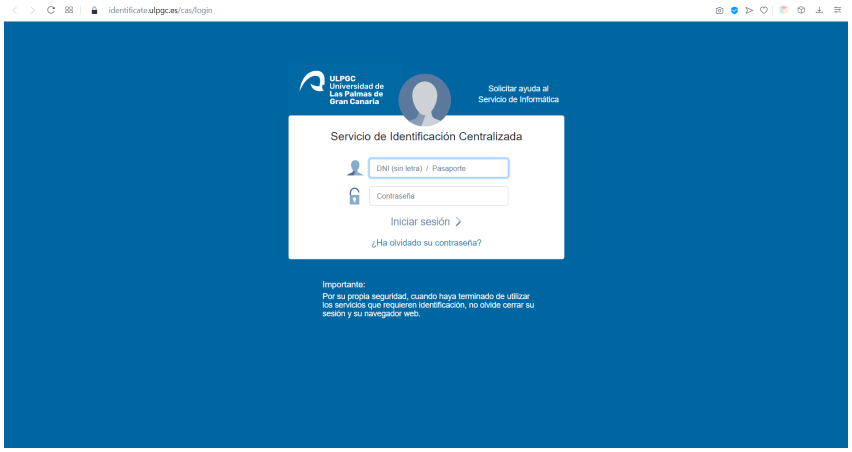


Figura 1.6: Pantalla que nos solicita identificador y clave de la ULPGC

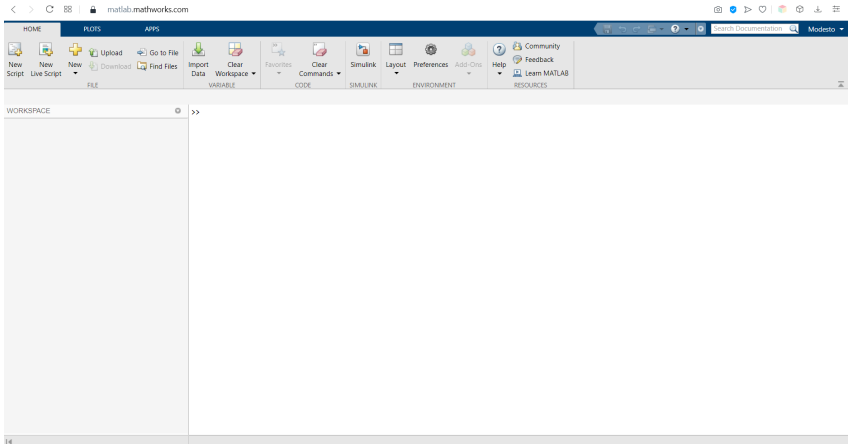


Figura 1.7: Interfaz en Matlab online

cos(90) y *cosd(90)*?

4. ¿Por qué el resultado de *floor(2.4)* es 2 y el de *floor(-2.4)* es -3?

5. ¿Qué valor se almacena en **c** tras ejecutar las siguientes líneas?

```
>> a=10;
>> b=10^a;
>> c=a+sqrt(b)
```

1.5. Tarea

La evaluación de esta práctica, se realiza a través de un cuestionario sobre el contenido de este documento y las actividades realizadas durante su correspondiente sesión práctica.

El cuestionario estará disponible para su resolución el viernes 12 de febrero de 12 a 18 horas a través del campus virtual.

BIBLIOGRAFÍA

Modesto Fdo. Castrillón Santana, Antonio Carlos Domínguez Brito, Santiago Candela Solá, Luis Jesús Doreste Blanco, David Freire Obregón, Agustín Javier Salgado de la Nuez, Sunil Lalchand Khemchandani, and José Daniel Hernández Sosa. *Fundamentos de informática y programación para ingenieros. Ejercicios resueltos para C y Matlab.* Paraninfo, 2011.

Práctica 2

Matlab como calculadora matricial

2.1. INTRODUCCIÓN

Tras los primeros pasos utilizando Matlab cubiertos en la primera sesión práctica, en esta nueva sesión se enumeran de forma creciente diversas posibilidades disponibles en Matlab para el cálculo matricial.

2.2. VECTORES Y MATRICES

Las variables utilizadas en los ejemplos previos tienen una sola dimensión, es decir, cada variable contiene un único valor. Recordemos la asignación de un valor a una variable:

```
>>x=2
x =
     2
```

Sin embargo, tanto Matlab como Octave están diseñados para facilitar la realización de operaciones de muy diverso tipo con vectores y matrices. La definición de una variable de tipo vector se realiza de forma similar a como hemos visto hasta ahora con las variables, con la salvedad de requerir el uso de corchetes para delimitar los valores que debe contener la variable. El ejemplo con un vector de tres elementos:

```
>>v1=[3 4 5]
v1 =
    3 4 5
```

Realmente. para Matlab el vector **v1** es una matriz con una única fila, y también la variable **x** del ejemplo previo también es una matriz, con la singularidad de tener una fila y una columna. La función *size* permite conocer las dimensiones de una variable, indicando

en primer lugar el número de filas. Esta información también está presente en la zona del *Workspace* par alas distintas variables:

```
>> x=2
x =
     2

>> v1=[3 4 5]
v1 =
     3     4     5

>> size(x)
ans =
     1     1

>> size(v1)
ans =
     1     3
```

Como puedes apreciar, en ambos casos se indican dos dimensiones, correspondientes a filas y columnas. Para especificar una matriz con más de una fila, también se hace uso corchetes, si bien el final de cada fila se marca con el punto y coma (;). Debes cuidar que haya coherencia en los tamaños de las respectivas filas, o puede aparecer errores.

```
>> m1=[ 0 4 5 ; 2 3 7 ]
m1 =
    0 4 5
    2 3 7
```

En resumen, tanto **v1** y **m1** son variables, pero con la particularidad de contener varios valores bajo un único nombre. Esta característica encaja muy bien con las sentencias repetitivas que conoceremos en un futuro no muy lejano.

Llegados a este punto, sabemos que se pueden almacenar varios valores en una variable, pero en ocasiones nos interesará operar con valores concretos del vector o matriz. Si queremos acceder al valor de una posición específica, por ejemplo, el tercer elemento del vector **v1**, y operar:

```
>> v1(3)
ans =
    5
>> z=v1(3)*3
z =
    15
```

O el elemento de la matriz **m1** situado en la segunda columna de la segunda fila, par alo que necesitaremos indicar dos índices, uno para filas y otro para columnas, con el objetivo quizás de utilizarlo en una operación que nos interese:

```
>> m1(2,2)
ans =
    3
>> z2=m1(2,2)*v1(1)
z =
     9
```

La ya mencionada función *size* junto con la función *length* permiten conocer las dimensiones de vectores y matrices. La segunda de ellas, para un vector indica su longitud, y para matriz indicará el tamaño de la dimensión mayor (en realidad viene a ser lo mismo que para un vector).

Otra interesante opción es la posibilidad de acceder a un rango de las posiciones de un vector o matriz, por ejemplo, si queremos ver no todos los valores del vector **v1**, sino los que están entre las posiciones 1 y 2, se hace uso del operador dos puntos (:), y mostramos el cálculo de una función sobre esos valores concretos:

```
>> v1(1:2)
ans =
    3 4
>> sqrt(v1(1:2))
ans =
    1.7321      2.0000
```

El operador, en este caso (:), viene acompañado de una posición inicial a mostrar y una final. Si la diferencia entre ellas fuese mayor que uno, se mostrarán también los valores de las posiciones intermedias.

Este tipo de operaciones son extensibles a una matriz. Definamos primera la matriz **m**:

```
>>m=[1 2 3; 4 5 6; 7 8 9]
m =
    1   2   3
    4   5   6
    7   8   9
```

Podemos, por ejemplo, seleccionar los elementos situados entre las columnas 1 y 3 de la fila 1:

```
>>m(1,1:3)
ans =
    1   2   3
```

El esquema de selección puede extenderse a cualquier índice de la matriz, en este nuevo ejemplo lo aplicamos tanto a filas como columnas:

```
>>m(1:2,1:2)
ans =
    1   2
    4   5
```

La matriz **m** tiene tres filas y tres columnas, intentar acceder a un valor fuera del rango de una matriz produce un error:

```
>>a = m(4,4)
    ??? Attempted to access m(4,4); index out of bounds because size(m)=[3,3].
```

Sin embargo, podemos modificar dinámicamente las dimensiones de una matriz. Matlab extenderá la matriz lo justo, rellenando con ceros los elementos no definidos.

```
>>m(6,5) = -1
m =
    1  2  3  0  0
    4  5  6  0  0
    7  8  9  0  0
    0  0  0  0  0
    0  0  0  0  0
    0  0  0  0 -1
```

Las operaciones aritméticas descritas en la primera sesión de práctica, se pueden realizar también con vectores y matrices, no solo accediendo a valores determinados, como se ha mostrado en alguno de los ejemplos previos, sino también de forma general con todos los valores contenidos en un vector o una matriz. Por ejemplo:

```
>>v1(2)*3
ans =
    12

>>m1(2,2)*1.5
ans =
    4.5000

>>10*m1
ans =
    0     40     50
    20    30     70
```

Las operaciones aritméticas pueden de igual forma involucrar varios vectores y matrices. Si bien, debes recordar las particularidades del producto matricial. Si por ejemplo intentamos realizar el producto de dos vectores:

```
>>v1*v1
     ??? Error using ==> mtimes Inner matrix dimensions must agree.
```

Ocurre un error. Se debe a que intentamos multiplicar dos vectores, ambos con tres columnas, que es equivalente a intentar multiplicar dos matrices de una fila y tres columnas. No es posible realizarlo ya que el número de columnas del primer operando debe coincidir con el de filas del segundo. Podríamos, sin embargo, multiplicar v1 por su traspuesta.

```
>>v1*v1'
ans =
```

```
    50
```

El operador (‘) es una abreviatura de la función *ctraspose()* que calcula la conjugada traspuesta de la matriz. Si sólo queremos obtener la traspuesta usaríamos (.’) que es la abreviatura de la función *traspose()*. Obviamente, si la matriz es real ambas funciones devuelven el mismo resultado.

Si en lugar de querer realizar el producto matricial, se pretende realizar el producto elemento a elemento. Basta con anteceder el operador producto con un punto (.).

```
>>v1.*v1
ans =
    9 16 25
```

Obteniendo de esta forma, el cuadrado de cada elemento del vector. Con matrices la operativa es similar:

```
>>m1*m1
??? Error using ==> mtimes Inner matrix dimensions must agree.
>>m1*m1'
ans =
    41 47
    47 62
>>m1.*m1
ans =
    0 16 25
    4 9 49
```

Teniendo en mente un esbozo de las operaciones matriciales disponibles, disponemos de una herramienta para resolver sistemas de ecuaciones. A modo de ejemplo, partamos del siguiente sistema de ecuaciones:

$$\begin{aligned} 2x + 5z &= 100 \\ 3x + 5y + 9z &= 251 \\ 1x + 5y + 7z &= 301 \end{aligned} \tag{2.1}$$

Podemos considerar la ecuación matricial $A \times C = B$, donde es A la matriz de coeficientes, C la de incógnitas y B la de términos independientes. Resolvemos definiendo los valores de las respectivas variables conocidas, y despejando nuestra variable incógnita $C = A^{-1} \times B$:

```
>>A = [ 2 0 5; 3 5 9; 1 5 7]
>>B = [ 100 ; 251; 301 ]
>>C = inv (A)* B
C =
    -75.0000
```

```
    5.2000
    50.0000
```

En la resolución del sistema de ecuaciones, se hace uso de una función incluida tanto en Matlab como Octave que es *inv*, la cual nos devuelve la inversa de la matriz indicada entre paréntesis. Un sistema de ecuaciones tiene solución siempre que el determinante de la matriz A no sea nulo. Aunque hemos obtenido solución, podemos comprobarlo:

```
>>det(A)
ans =
    30.0000
```

De igual forma a como hemos usado la función *inv* puede aplicarse sobre matrices el repertorio de funciones mencionadas durante la primera sesión de práctica. Como norma general, toda función de Matlab puede aplicarse a vectores y matrices, obteniendo como resultado, el correspondiente a aplicar la función a cada elemento de la matriz.

```
>>sqrt(ml)
ans =
         0    2.0000    2.2361
    1.4142    1.7321    2.6458

>>sqrt(4)
ans =
    2

>>log(v1*2)
ans =
    1.7918    2.0794    2.3026

>>sqrt(1:10)
ans =
    1.0000    1.4142    1.7321    2.0000    2.2361    2.4495    2.6458    2.8284    3.0000    3.1623

>>log(0:0.1:1)
ans =
    -Inf   -2.3026   -1.6094   -1.2040   -0.9163   -0.6931   -0.5108   -0.3567   -0.2231   -0.1054
               0
```

2.3. Operador de rangos

Quizás hayas observado algo novedoso en las dos últimas llamadas a función del ejemplo previo. En ocasiones, nos interesa almacenar valores en una variable, existiendo un patrón en dichos valores. Por ejemplo, podemos necesitar definir un rango de valores equidistantes, y almacenarlos en un vector. Matlab ofrece utilidades para tal fin. Si queremos rellenar un

vector con valores entre 0.5 y -0.5 con saltos de una décima, puede realizarse manualmente, o con atajos:

```
>>v2=[0.5:-0.1:-0.5]
v2 =
    Columns 1 through 6
     0.5000    0.4000    0.3000    0.2000    0.1000         0

    Columns 7 through 11

    -0.1000   -0.2000   -0.3000   -0.4000   -0.5000
```

Unos párrafos antes hemos usado el operador (:) para referirnos a posiciones dentro del vector o matriz. Aquí se hace algo similar, se definen los valores inicial, final, además del paso o salto (incremento o decremento), por medio del operador (:), produciendo un vector que contiene los valores desde el valor inicial al final, con valores intermedios en base al paso definido. Existen dos formas:

$$V = a : b$$
$$V = a : p : b$$

En el primer caso, el paso entre los valores a y b es la unidad, y por tanto suele utilizarse cuando se pretende generar un vector conteniendo número enteros. En el caso más general, se suele utilizar la segunda versión, donde p define el paso. La secuencia generada por esta es:

$$a, a+p, a+2p, \ldots, a+(n-1)p$$

El primer elemento generado tiene por valor a (que podemos escribir como: $a+0p$), el segundo elemento es $a+p$, y así sucesivamente. El elemento enésimo es: $a+(n-1)p$

En el caso de necesitar un vector de valores con N elementos tal que el valor inicial sea a, y el enésimo tenga el valor b, dados a, b y N se debe determinar el valor del salto, p, para poder completar el comando solicitado,

$$a, a+p, a+2p, \ldots, b$$

de forma que el valor enésimo valga b. Resolvemos con $p = (b-a)/(N-1)$ o haciendo uso del comando *linspace(1,b,N)*, que simplifica la operación, además de asegurar que se incluyen los dos valores extremos:

```
>> a=linspace(0,10,10)
```

```
a =
         0    1.1111    2.2222    3.3333    4.4444    5.5556    6.6667    7.7778    8.8889   10.0000

>> a=linspace(0,10,9)
a =
         0    1.2500    2.5000    3.7500    5.0000    6.2500    7.5000    8.7500   10.0000

>> a=linspace(0,10,11)
a =
     0     1     2     3     4     5     6     7     8     9    10

>> a=linspace(0,10,3)
a =
     0     5    10
```

2.4. EDITOR DE PROGRAMAS DE MATLAB

Hasta este momento hemos utilizado la ventana de comandos para indicar los diferentes comandos a ejecutar, sin embargo, es posible hacer uso de un fichero que almacene toda la serie de comandos, para nuestra comodidad en el caso de querer recuperarlos posteriormente, son los denominados fichero *.m*. Si estamos interesados en crear un fichero .m que contenga una serie de comandos, debe crearse con cualquier editor (bloc de notas, Wordpad, Notepad++, etc.) un fichero de extensión .m. La interfaz de Matlab también proporciona su propio editor, lo que resulta muy práctico. Para abrir el editor de Matlab seleccionamos *File->New M File* y aparecerá la ventana de editor como se observa en las Fig. 2.1 y 2.2. Una vez abierto, tecleamos en el fichero los comandos a ejecutar, lo guardamos en disco. La ejecución de un fichero o programa .m es muy sencilla, basta con asegurarse que está en nuestra carpeta actual (*current directory*), y escribir su nombre (sin extensión) en la consola de comandos de Matlab.

A modo ilustrativo incluimos un ejemplo que crea una matriz y la visualiza a continuación como si fuera una imagen, ver listado 2.1. El listado contiene varios comandos nuevos. En primer lugar *clear*, siendo aconsejable en todo programa, dado que limpia la memoria, eliminando toda variable existente que pueda interferir en la nueva ejecución. La función *zeros* crea una matriz rellena de ceros, en el caso del ejemplo, la matriz tiene tres dimensiones indicando entre paréntesis respectivamente el número de filas, columnas y planos. La tercera dimensión se corresponde con los planos RGB (del inglés *red*, *green* y *blue*) que posteriormente interpreta la función de visualización de imagen *imshow*. El resto de líneas del programa simplemente sobreescribe los valores de algunas posiciones de la matriz inicializada a cero. Los valores de cada posición de la matriz, que se interpretará como píxel de la imagen, deben

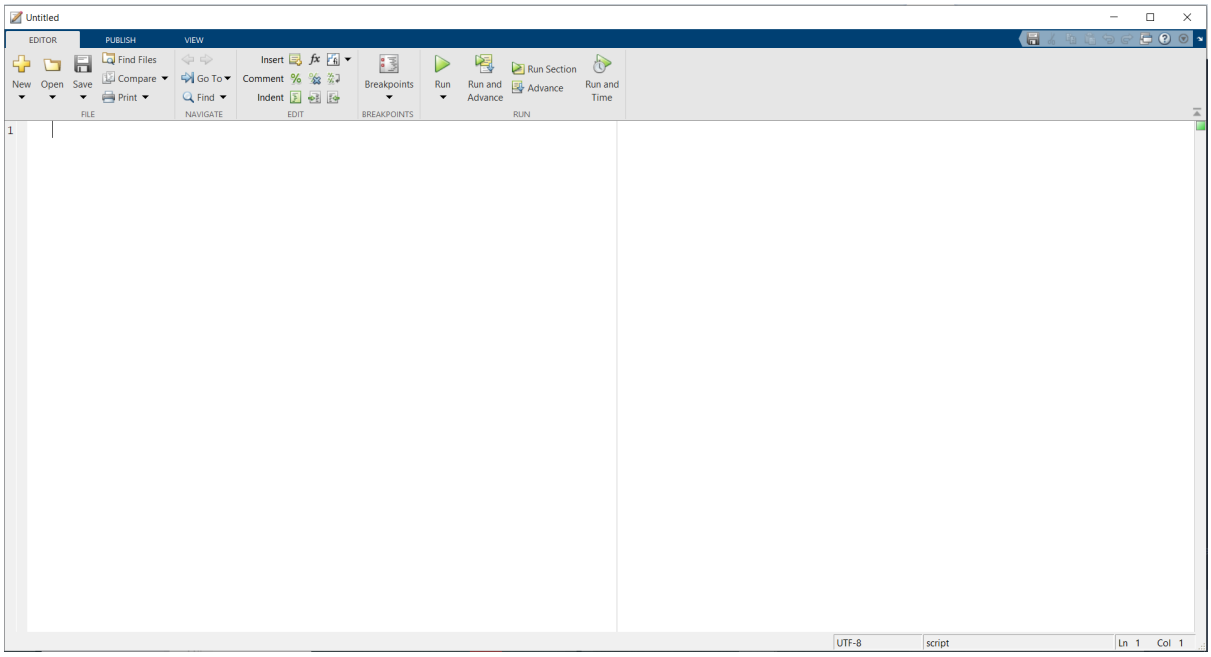


Figura 2.1: Interfaz del editor de código de Matlab.

estar entre 0 y 1 para su correcta especificación como colores por la mencionada función *imshow*.

Listado 2.1: Un programa m que crea una imagen RGB

```
clear;
a=zeros(500,500,3);
a(:,1:100,1)=0;
a(:,100:200,1)=0.25; a(:,200:300,1)=0.5;
a(:,300:400,1)=0.75;
a(:,400:500,1)=1;
imshow(a)
```

La gran ventaja del uso de un fichero es que podemos lanzar una secuencia de comandos mucho más fácilmente, y su modificación es sencilla para obtener otro resultado. ¿Te atreverías a modificar el programa para dibujar algo en esa matriz? Por ejemplo, un recuadro, o incluso una composición al estilo de Piet Mondrian[1]. Como propuesta final, en lugar de usar valores fijos para el color, te animo a utilizar un valor aleatorio entre 0 y 1, que se obtiene con una llamada a la función *rand()*.

[1] https://es.wikipedia.org/wiki/Piet_Mondrian

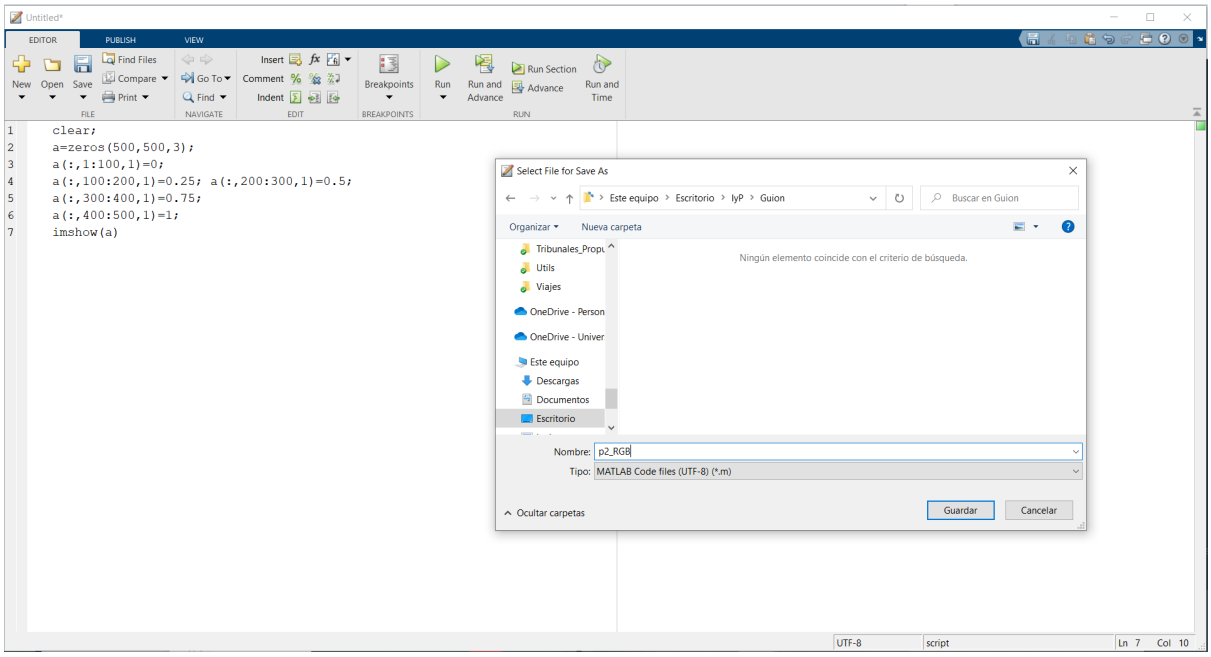


Figura 2.2: Programa m.

2.5. EJERCICIOS

2.5.1. De apoyo durante la sesión

1. Utilizando el operador de rangos, crea un vector con 10 valores equiespaciados, partiendo del valor 1, y finalizando en el valor 10.

2. Modifica el código para crear el vector con 20 valores.

3. Modifica el código para que se almacenen en el vector exactamente **n** valores, siendo **n** una variable previamente creada.

4. Modifica el código anterior para que los valores inicial y final no sean 1 y 10, sino que estén almacenados en dos variables, respectivamente **x1** y **x2**.

5. Si suponemos que nos encontramos en el suelo y queremos lanzar en vertical un objeto con una velocidad inicial, podemos hacer uso de la expresión del movimiento rectilíneo uniformemente acelerado, $v = v0 + at$, para describir el movimiento. Donde **v** es la velocidad, **v0**, la velocidad inicial, **t** un instante de tiempo, y a la aceleración, que en este escenario es la producida por la gravedad, aproximadamente $-9{,}8m/s^2$. Indica los comandos necesarios para calcular, dada una velocidad inicial, el valor de **t** cuando el objeto alcanza su máxima altura (¿Cuánto vale **v** en ese momento?)

6. Modifica el código anterior, para una vez calculado el instante **t** en el que el objeto alcanza su máxima altura, almacene en un vector **n** valores de **v**, equiespaciados en el tiempo, entre 0 y dicho valor de **t**.

2.5.2. Propuestos como actividad independiente

1. Escribe los respectivos comandos Matlab para generar cada uno de los siguientes vectores:

   a) $v = \begin{bmatrix} 1 & 2 & -1 & 3 \end{bmatrix}$

   b) $w = \begin{bmatrix} 1 \\ 2 \\ -3 \end{bmatrix}$

   c) Un vector fila, **a**, que contenga 20 valores comenzando en 10 y acabando en 100 (ambos inclusive).

   d) Un vector columna, **b**, que contenga los números impares entre 1 y 1000.

2. Si *x=0:2:20*, escribe el código de Matlab necesario para almacenar en la variable **rc** la raíz cuadrada de cada celda de **x**.

3. Si *y=linspace(0,2*pi,1000)*, ¿Qué valor tiene la celda 50 de **y**? ¿Cuál es la longitud del vector **y**?

4. Si $k = 0:100, p = 0.5\hat{}k$ ¿Tiene **p** 101 celdas? Si las tiene, ¿qué valor contiene la celda número 101?

5. Guarda en la variable **q** un vector de 10 celdas con valores aleatorios en el intervalo $(0,1)$.

6. Guarda en la variable **N** una matriz 4×4 con números enteros aleatorios en el intervalo $(-3,7)$.

7. Dadas las matrices $A = \begin{pmatrix} 2 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{pmatrix}$ y $B = \begin{pmatrix} 1 & 1 & 1 & 2 \\ 2 & 0 & 1 & 3 \\ 1 & 1 & 0 & 1 \end{pmatrix}$, calcula:

   a) Traspuesta de **A**.

   b) Determinante de **A**.

   c) Inversa de **C**, siendo **C** la matriz **B** sin la columna 4.

d) $A \times B$ y $A. \times C$.

e) $F \times E'$ siendo **F** la submatriz de **A** formada por las filas 1ª y 3ª y columnas 1ª y 3ª; **E** la submatriz de B formada por las filas 2ª y 3ª y columnas 2ª y 3ª.

8. El comando *magic(n)* genera una 'matriz mágica' $n \times n$ con números enteros (de 1 a n^2) con la propiedad de que las sumas por filas, columnas y diagonales (principal y secundaria) son iguales.

   a) Genera la matriz mágica 6×6 y guárdala en la variable **M**.

   b) Calcula y muestra la suma de la primera fila y de la diagonal principal.

   c) Calcula y muestra el producto $M1 \times M2$ siendo M1 la submatriz de M formada por filas 1, 2 y 5, columnas 3, 4 y 6 y **M2** la submatriz de **M** formada por filas 2, 4 y 6, columnas 1, 3 y 5.

   d) Calcula y muestra el producto $u1. \times u2$ siendo **u1** el vector fila que se obtenga de la fila 4 de **M** y **u2** el vector columna que se obtenga de la columna 2 de **M**.

2.6. TAREA

La evaluación de esta práctica, se realiza a través de un cuestionario sobre el contenido de este documento y las actividades realizadas durante su correspondiente sesión práctica. El cuestionario estará disponible para su resolución el viernes 19 de febrero de 12 a 18 horas a través del campus virtual.

Práctica 3

Matlab como herramienta de visualización gráfica

3.1. Introducción

En esta práctica se introducen las posibilidades gráficas de Matlab, también presentes en Octave. Dentro del repertorio de funciones gráficas disponibles podemos distinguir entre bajo y alto nivel. Las de bajo nivel (*line*, *patch*, etc.) permiten dibujar gráficos elementales como líneas, polígonos, etc., o gráficos más elaborados a partir de varias llamadas combinando los mencionados gráficos elementales. Por el contrario, las funciones de alto nivel (*plot*, *plot3*, *surf*, *mesh*, etc.) permiten representar gráficamente funciones o conjuntos de datos complejos (en forma de vectores o matrices de coordenadas x, y y z) con un único comando.

3.2. Gráficos de bajo nivel

El comando *line* permite dibujar líneas entre sucesivos pares de puntos del plano cartesiano. La función requiere como parámetros dos vectores: el primero con las coordenadas x, y el segundo con las correspondientes coordenadas y de los puntos a unir con líneas. A modo de ejemplo, para dibujar una línea que une el punto $(0,0)$ con el $(2,1)$ escribiremos el código a continuación, con el resultado de la figura 3.1:

```
>>line([0 2],[0 1]);
```

Proporcionando más de dos pares de puntos, se dibuja una línea poligonal, ya que une cada par (x, y) con el anterior. El ejemplo del siguiente listado almacena en dos vectores, **x** e **y**, con el resultado de la figura 3.2.

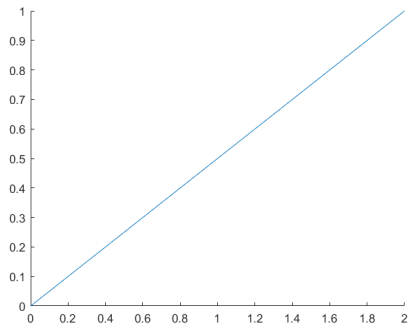


Figura 3.1: Resultado del comando *line*.

```
>> x = [1 2 3 4];
>> y = [2 3  0 1];
>> line(x,y)
```

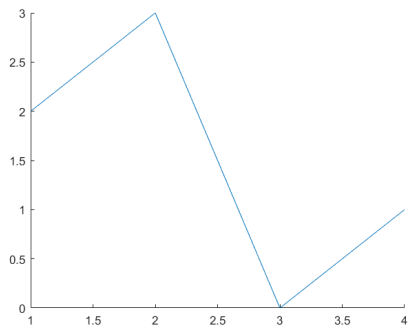


Figura 3.2: Resultado del comando *line* con varios pares de puntos.

Si se quieren especificar segmentos independientes, es necesario separar las coordenadas por filas, uniéndose los puntos por columnas.

```
>> x = [1 2 ; 3 4];
>> y = [2 3 ; 0 1];
>> line(x,y)
```

Para dibujar polígonos se utiliza el comando *patch*. El siguiente ejemplo dibuja un triángulo azul (indicado por *'b'* de *blue*), ver la figura 3.3:

```
>> x1 = [0 1 0];
>> y1 = [0 0 2];
>> patch(x1,y1,'b')
```

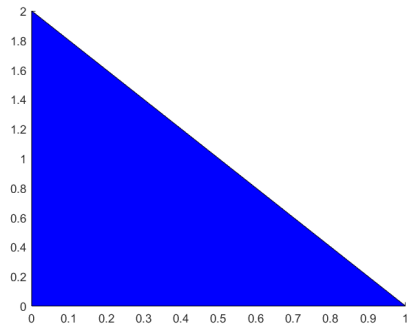


Figura 3.3: Resultado del comando *patch*.

3.3. GRÁFICOS DE ALTO NIVEL

3.3.1. Gráficos 2D

En la práctica anterior aprendimos a definir en Matlab rangos de valores con el operador (:), viendo ejemplos para rellenar los elementos de un vector. Si creamos un vector **x** con 180 valores equiespaciados de la función seno en el rango $[0, 6 \cdot \pi]$, y a continuación almacenamos en otro vector, **v1**, el seno de cada uno de esos valores. Una posible solución sería aplicar $(x2 - x1)/(n - 1)$ donde $x1$, $x2$ y n serían respectivamente 0, $6 \times pi$ y 180:

```
>> x = [0:6*pi/179:6* pi];
>> v1 = sin(x);
```

Una alternativa más cómoda que realiza la misma acción es el, ya mencionado durante la sesión anterior, comando *linspace*. Para el ejemplo que manejamos sería:

```
>> x = linspace(0,6*pi,180);
>> v1 = sin(x);
```

Una vez almacenados los valores en el vector **v1**, es posible realizar una representación gráfica de dichos valores en el plano cartesiano. Para ello disponemos del comando o función *plot*, que produce una representación de pares de puntos en el plano cartesiano (ordenadas frente a abscisas). Para realizarlo con el vector **v1** que acabamos de definir, teclea:

```
>> plot(v1);
```

El resultado se muestra en la figura 3.4. El comando *plot* dibuja pares de puntos, por lo que espera como argumentos tanto un vector con la coordenada x de cada par de puntos a representar, como otro vector con la correspondiente coordenada y. Dado que se indicó un único vector (*plot(v1)*), el comando *plot* asocia la posición del vector con el eje x y los valores contenidos en el vector **v1** con la y, en este caso el seno calculado. Claramente, la posición

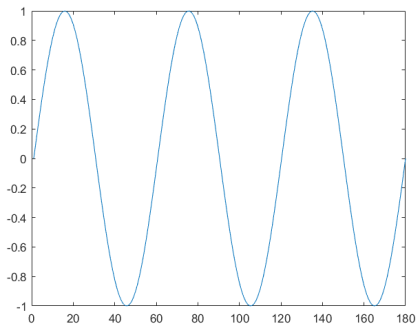


Figura 3.4: Representación de valores de la función seno en el rango $[0{,}6 \times pi]$.

no refleja el valor x para el que estamos obteniendo el seno y representando gráficamente. Sería adecuado proporcionar como primer parámetro de *plot* los valores del rango $[0,6 \cdot \pi]$ para los que hemos calculado el seno, valores contenidos en el vector **x**:

```
>> plot(x,v1);
```

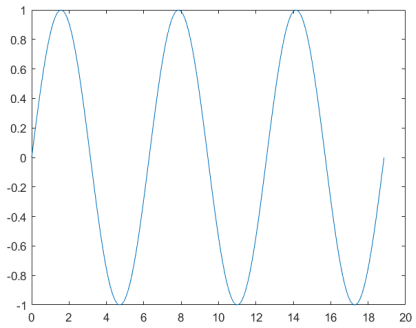


Figura 3.5: Representación de la función seno en el rango $[0{,}6 \cdot pi]$.

El resultado, ver figura 3.5, es similar excepto por las etiquetas del eje x. En resumen, el primer parámetro que se le proporciona al comando *plot* es el rango de valores para el que hemos calculado el seno, mientras que el segundo es el vector que contiene los valores resultantes, la $f(x)$. Ambos vectores deben tener la misma dimensión, o se indicará la existencia de un error. Si redefinimos los valores del vector **x** y lanzamos de nuevo el comando *plot*, obtendremos distintas gráficas, por ejemplo pintemos hasta $10 \cdot \pi$, ver figura 3.6.

```
>> x=(0:10*pi/179:10*pi);
>> plot(x,sin(x));
```

En las figuras 3.4-3.6 es posible observar el carácter periódico de la función seno. ¿Qué pasaría si cambiamos el orden de los parámetros en la llamada a *plot*? ¿No te lo imaginas? Prueba.

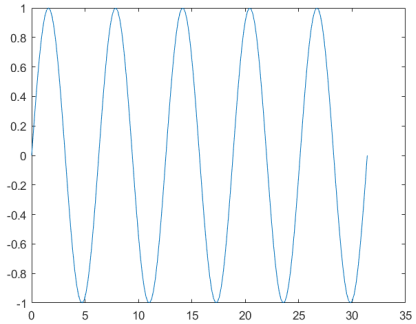


Figura 3.6: Representación de la función seno en el rango $[0,10 \times pi]$.

Llegados a este punto, seguramente habrás observado la barra de menú visible en la ventana que contiene a la figura. En ella existen botones que permiten hacer zoom, rotar la imagen, y guardarla si fuese necesario (quizás para un informe o memoria). Pero antes de guardarla, observa que hay muy poca información contenida en esta gráfica, ni tan siquiera al verla podemos identificar lo que representa cada eje. Dicha información puede añadirse de forma interactiva a través de la interfaz, o personalizarse usando parámetros de la gráfica una vez dibujada como veremos posteriormente.

Antes, jugaremos con la configuración de l curva representada. Teclea por ejemplo:

```
>> plot(x,sin(x),'r');
```

¿Has apreciado el cambio de color? Teclea ahora:

```
>> plot(x,sin(x),'b');
```

En este caso ha sido azul el color utilizado para la gráfica. Con ese tercer parámetro del comando *plot* podemos justamente indicar el color de la gráfica, muy útil si tenemos más de una gráfica. Habitualmente es la inicial de su nombre en inglés. Otros colores disponibles son:

b azul (*blue*)
g verde (*green*)
r rojo (*red*)
c cian (*cyan*)
m magenta (*magenta*)
y amarillo (*yellow*)
k negro (*black*)

Para utilizar otro color diferente a los mencionados, debes hacer uso de la representación en el espacio RGB (rojo, verde y azul) indicando para cada una de las tres componentes un valor entre 0 y 1, el resultado del siguiente código se muestra en la figura 3.7:

```
>> plot(x,sin(x),'Color',[1 0.5 0.5])
```

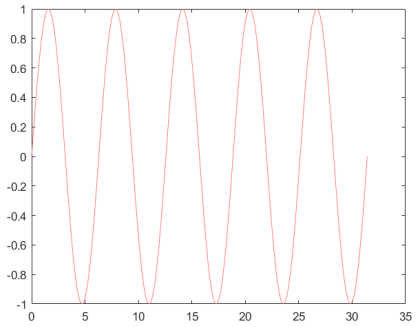


Figura 3.7: Modificando la propiedad del color de representación.

Además del color se puede definir la marca usada para pintar cada par (x, y), el siguiente códig, cuyo resultadoo se muestra en la figura 3.8:

```
>> plot(x,sin(x),'r*');
```

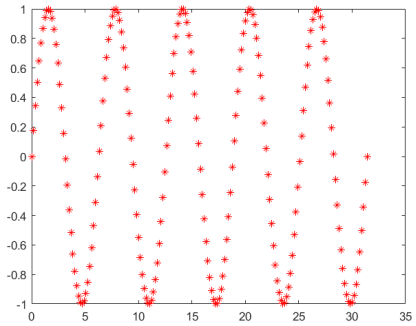


Figura 3.8: Configurando la marca del comando *plot*.

Otras marcas disponibles son:

. punto
o círculo
x equis
$+$ símbolo más
$*$ estrella
s cuadrado
d diamante
v triángulo (abajo)

^ triángulo (arriba)
< triángulo (izquierda)
> triángulo (derecha)
p pentagrama
h hexagrama

Un tercer carácter permite establecer el estilo de las líneas que unen los pares (x, y):

- línea sólida
: punteada
-. líneas y puntos
– a trozos
(ausente) sin línea

Combinando las opciones se obtienen distintos resultados, ver la figura 3.9:

```
>> plot(x,sin(x),'bs-');
```

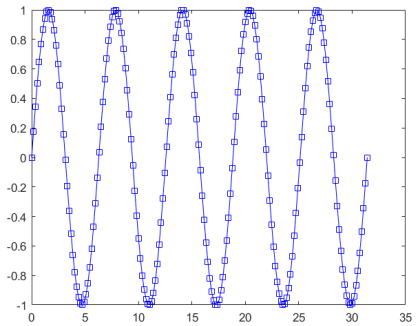


Figura 3.9: Configurando la marca del comando y el estilo de línea con *plot*.

Recuerda que estamos pintando el valor del seno en un rango de valores en radianes. Es un dato que no se ve reflejado en la gráfica. Para que la gráfica sea completa, debe aportarse siempre información sobre su contenido, podemos añadir etiquetas a cada eje:

```
>> xlabel('Radianes');
>> ylabel('Valor del seno');
```

E incluso un título a la gráfica, ver figura 3.10:

```
>> title('Valores del seno entre 0 y 10*\pi');
```

Una vez introducidos los valores, el comando *plot* decide de forma automática el rango de valores que muestran los ejes x e y. Para modificarlo según nos interese, se hace uso del comando *axis* definiendo un rango concreto del plano cartesiano. El ejemplo de la figura 3.11:

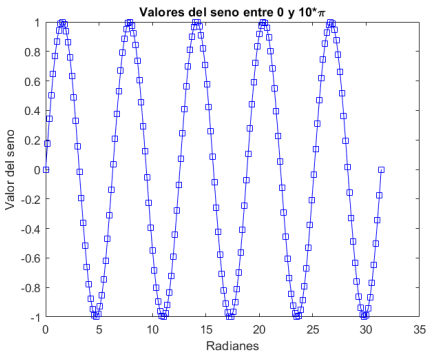


Figura 3.10: Incluyendo título y etiquetas de cada eje.

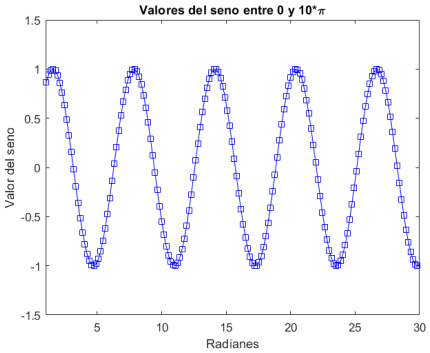


Figura 3.11: Modificando el rango de visualización con *axis*.

```
>> axis([1 30 -1.5 1.5]);
```

También es interesante el comando *grid* que añade una rejilla que sirve como referencia a quien ve la imagen, ver la figura 3.12.

```
>> grid
```

En cualquier momento es posible guardar la imagen con el comando *saveas*:

```
>> saveas(gcf,'Miimagen','png');
```

El primer parámetro identifica la imagen a guardar, en este caso la que esté activa en ese momento, el segundo especifica el nombre que se le dará al fichero, y el tercero establece el formato de la imagen. En el ejemplo se ha escogido *png*, que es un formato de imagen sin pérdida de calidad, pero hay otros como *jpeg* que se escoge usando como parámetro *p'jpg'*.

```
>> saveas(gcf,'Miimagen2','jpg');
```

Para representar dos gráficas en una ventana, basta con añadir otra terna de argumentos al comando *plot*, ver figura 3.13.

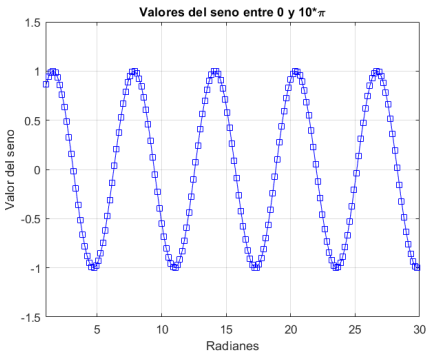


Figura 3.12: La propiedad *grid*.

```
>> plot(x,sin(x),'bs-',x,cos(x),'ro-');
```

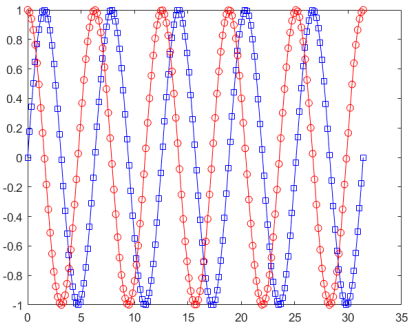


Figura 3.13: Dos curvas en una gráfica.

Añadir más gráficas requiere mostrar la leyenda para aclarar lo que contiene la imagen resultante, ver figura 3.14.

```
>> x=(0:10*pi/179:10*pi);
>> plot(x,sin(x),'bs-',x,cos(x),'ro-',x,sin(x)-cos(x),'k+-');
>> legend('Seno','Coseno','Seno-Coseno');
```

Una alternativa para dibujar varias gráficas sobre una misma figura es emplear el comando *hold*. Tras activarlo tecleando *hold on*, todas las llamadas al comando *plot* se realizarán sobre la misma ventana. Esta opción se desactiva introduciendo *hold off*. Ver ejemplo y su resultado en la figura 3.15.

```
>> x=0:10*pi/179:10*pi;
>> plot(x,sin(x),'bs-');
>> hold on;
>> plot(x,cos(x),'k+-');
>> hold off;
```

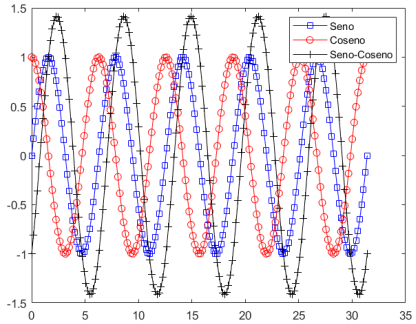


Figura 3.14: Varias curvas y la leyenda.

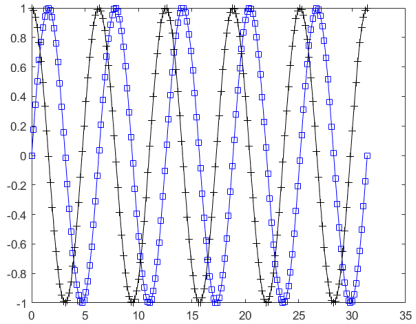


Figura 3.15: Varias curvas con *hold*.

El comando *subplot(m,n,p)* permite dividir el área de dibujo en una matriz de $m \times n$ rectángulos. Con el parámetro p seleccionamos la celda concreta que dibujamos a continuación, con el ya conocido comando *plot*. Indicar que p se corresponde a los rectángulos de dibujo por filas. Es decir p valdrá de 1 a n para la primera fila, de $n+1$ a $2 \times n$ para la segunda fila y así sucesivamente. Recuperamos nuestro ejemplo de función seno y coseno, cuyo resultado se ilustra en la figura 3.16.

```
>> v=(0:6*pi/179:6*pi);
>> subplot(1,2,1);
>> plot(v,sin(v));
>> subplot(1,2,2);
>> plot(v,cos(v));
```

En algunas ocasiones nos puede interesar modificar las etiquetas que se muestran en cada eje. Si quisieras modificar el número de ellas tras el *plot* teclea:

```
>> set(gca,'XTick',1:10:31);
```

para mostrar solo una de cada diez. Si quisieras modificar la etiqueta propiamente, como por ejemplo hacer uso de etiquetas no numéricas como los nombres de los meses:

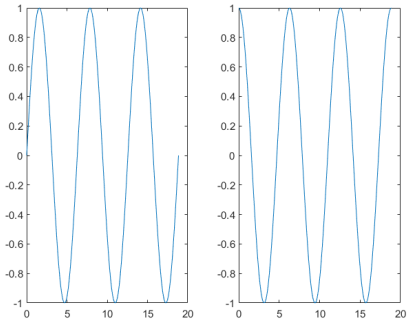


Figura 3.16: Utilizando el comando *subplot*.

```
>> set(gca,'XTickLabel',['Ene';'Feb';'Mar';'Apr';'May';'Jun';'Jul';'Ago'; 'Sep';'Oct';'Nov';'Dic']);
```

Si las etiquetas se refirieran a años podrías usar algo como:

```
>> set(gca,'XTickLabel',2000:2009);
```

Como habrás observado *XTickLabel* se aplica a las etiquetas del eje x, *YTickLabel* se aplica a las del eje y. Ten en cuenta que se debe indicar un número de etiquetas coincidente con las que muestra el *plot* de forma automática, o las fijadas con *XTick*.

En caso de querer representar sólo los puntos, y no unirlo con el previo, podemos emplear *stem*, cuyo resultado se muestra en la figura 3.17.

```
>> x=0:10*pi/59:10*pi;
>> stem(x,sin(x))
```

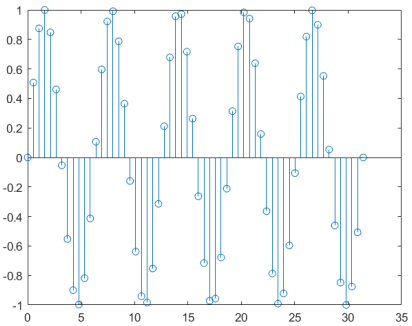


Figura 3.17: El comando *stem*.

Como nota fila, señalar que si necesitamos trabajar con coordenadas polares en lugar de cartesianas, hacemos uso de *cart2pol*, que realiza la correspondiente transformación de coordenadas cartesianas a polares dados los vectores con las coordenadas x e y.

3.3.2. Gráficos 3D

Matlab también cuenta con funciones para realizar gráficos en tres dimensiones, tanto curvas como superficies. Lo ilustramos almacenando en el vector **t** valores en el rango $[0, 10 \cdot \pi]$ con un cierto incremento arbitrario:

```
>> t = 0:pi/50:10*pi;
```

Obtenemos a partir de los valores de **t**, los valores de nuestra curva considerando para la x el seno, para la y el coseno, e igualando la tercera dimensión, z, al valor de **t**.

```
>> plot3(sin(t),cos(t),t);
```

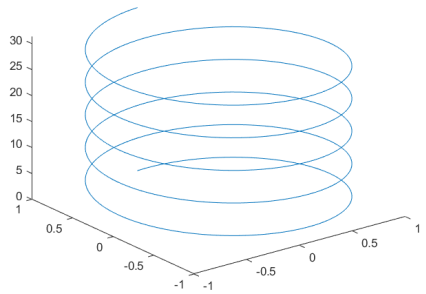


Figura 3.18: Dibujo de una espiral con *plot3*.

El resultado se muestra en la figura 3.18. Otras definiciones para la x, y y z producirán lógicamente figuras diferentes.

Si disponemos de la definición analítica de una superficie tridimensional, como por ejemplo $z = x^2 - y^2$, es posible obtener su representación gráfica. Como primer paso almacenamos en los vectores **x** e **y** los valores del intervalo que queremos representar, en nuestro ejemplo escogemos el rango $[-2, 2]$ tecleando:

```
>> [x,y]=meshgrid(-2:0.2:2,-2:0.2:2)
```

Introduciendo en la ventana de comandos **x** e **y** para ver el contenido, observarás que se crean dos matrices, **x** e **y** en las que se almacenan repetidas veces valores en el intervalo en dicho rango. Para evaluar dichos valores en una función descrita de forma analítica usamos el comando *surf* cuya salida ilustra la figura 3.19.

```
>> surf(x,y,x.^2-y.^2)
```

Recuerda que el operador (.) indica que se realiza la operación por elemento y no considerando el producto matricial. Tecleando *colorbar* aparece una barra indicando el color utilizado para los distintos valores de z.

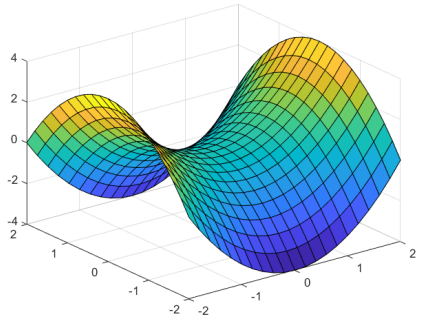


Figura 3.19: Resultado del comando *surf*.

3.4. GRÁFICOS ESTADÍSTICOS

Matlab también dispone de diversas utilidades que facilitan la realización de gráficos variados para la representación de datos, como pueden ser los diagramas de sectores (*pie*, *pie3*), diagramas de Pareto (*pareto*), o diagramas de barras (*bar*, *barh*, *bar3*, *bar3h*).

Un ejemplo básico de diagrama de sectores, cuyo resultado se ilustra en la figura 3.20:

```
>> x=[26 15 17 2];
>> leyenda={'A'; 'B'; 'C'; 'D'};
>> pie(x), legend(leyenda);
```

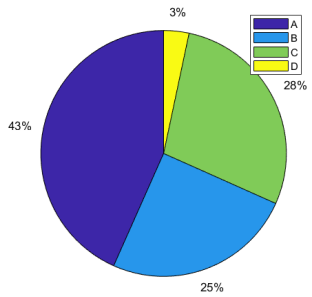


Figura 3.20: Diagrama de sectores.

Incluimos también ejemplos de uso de los comandos *bar* y *stairs* para la representación de diagramas de barras, mostrados en la figura 3.21:

```
>> subplot(1,3,1)
>> x=[9 7 6 2 4 5 1];
>> bar(x)
>> subplot(1,3,2)
>> stairs(x);
>> subplot(1,3,3)
>> dat=[ 2 3 1 4 5 ; 4 2 5 1 2];
```

```
>> bar(dat)
```

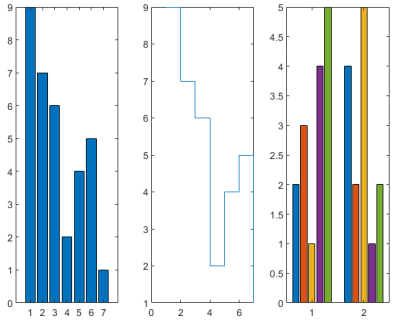


Figura 3.21: Resultados con *bar* y *stairs*.

El comando *bar* admite entrada en forma matricial donde los valores de la misma columna pueden visualizarse de forma adyacente o apilada, ver la figura 3.22

```
fil=3;
col=5;
%Mtriz con fil*col elementos enteros en el rango (0,10)
datos=randi(10,fil,col);

subplot(1,2,1);
bar(dado);

subplot(1,2,2);
bar(datos,'stacked');
```

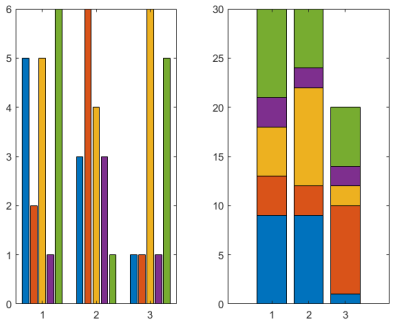


Figura 3.22: Resultados con *bar* con una matriz como entrada.

Existen variantes tridimensionales., en este caso, el comando *bar3*, permite la creación de diagramas de barras en tres dimensiones. Ambas salidas se muestran en la figura 3.23.

```
close all %Cierra ventanas gráficas abiertas
```

```
%Modifica el color de fondo de la ventana
set(gcf,'Color','w');

n=5;
%Vector con n elementos enteros en el rango (1,6)
dado=randi([1 6],1,n);

subplot(1,2,1);
bar(dado);

subplot(1,2,2);
bar3(1:n,dado,'r');
```

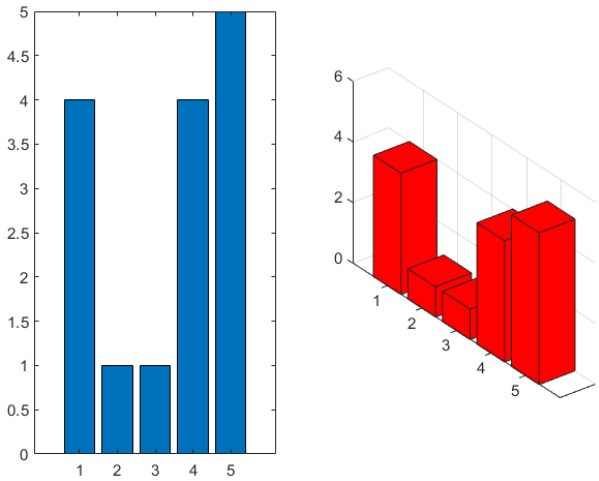


Figura 3.23: Resultados con *bar* y *bar3*.

Para representar medias y desviaciones se emplea habitualmente *errorbar*. La fig. 3.24 muestra el resultado partiendo de una matriz que contiene las notas de tres estudiantes en seis asignaturas:

```
>> notas=[8 7 7 ; 5 6 7; 3 4 6 ; 7 7 8; 9 8 7 ; 5 8 9]
>> media=mean(notas,2)
>> desv=std(notas,1,2);
>> errorbar(media,desv,'xr')
```

3.5. EDITOR DE PROGRAMAS DE MATLAB

Si bien se comenta en el guion de la práctica anterior, reiterar que tanto Matlab como Octave permiten indicar una secuencia de instrucciones o comandos tanto a través de la ventana de comandos como por medio de un fichero .m.

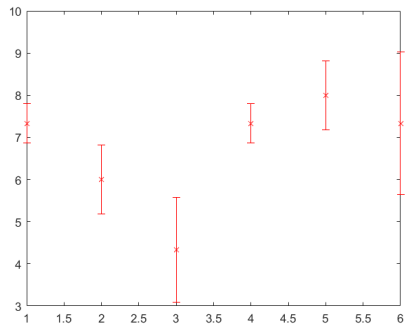


Figura 3.24: Resultados de y *errorbar*.

Para elaborar las gráficas anteriores hemos necesitado varios comandos de Matlab. Un ejemplo de fichero .m que genere la función seno con todos sus parámetros podría ser:

```
clear;
close all;
v=[0:6*pi/359:6*pi];
plot(v,sin(v),'bd--');
xlabel('Radianes');
ylabel('Seno');
title('Valores de seno entre 0 y 6*pi');
```

La mencionada gran ventaja del uso de un fichero es que podemos lanzar una secuencia de comandos mucho más fácilmente, y su modificación es más cómoda para obtener otras gráficas.

3.6. EJERCICIOS

3.6.1. De apoyo durante la sesión

1. Escribe en diferentes ficheros .m los comandos necesarios para representar gráficamente las siguientes funciones 2y 3D, en el rango y con el paso indicado. Emplea diferentes colores, marcas y trazos.

   a) $f(x) = \frac{x+1}{x^2-5x+6}$ $\quad -2 \leq x \leq 8$ con paso 0.02

   b) $f(x) = \frac{1}{e^{x/10}} sen(x)$ $\quad -20 \leq x \leq 20$ con paso 0.1

   c) $f(x) = sen(x) - \frac{3}{2} cos\left(\frac{x}{4}\right)$ $\quad -10 \cdot \pi \leq x \leq 10 \cdot \pi$ con paso 0.01

   d) $z = e^{-x^2} + e^{-y^2} + \frac{x^2}{10} + \frac{y^2}{10} + sen\left(\sqrt{x^2+y^2}\right)$ $\quad -4 \leq x, y \leq 4$ con paso: 0.1

2. Considera el último ejercicio de apoyo de la práctica anterior, apartado 2.5.1, donde definida una velocidad vertical inicial, $v0$, se calculaba el valor de t al alcanzar el punto

de altura máxima, donde $v = 0$, y se solicitaba que se creara un vector de n elementos de los valores de la velocidad en n instantes equiespaciados entre 0 y dicho valor t. Dibuja haciendo uso del comando *plot*, dicha colección de pares de puntos que relacionan tiempo y velocidad. Haz uso de todos los elementos necesarios para que la gráfica sea lo suficientemente informativa, p.e. usa los comando *xlabel*, *ylabel*, *legend*, etc.

3. Modifica el código anterior para representar en una misma gráfica las curvas obtenidas con dos valores distintos de velocidad vertical inicial, $v0$.

4. Combina los comandos *plot* y *subplot* para mostrar los resultados para 4 valores de velocidad vertical inicial, $v0$, visualizando cada curva resultante en una gráfica distinta.

5. En las primeras 500 cifras decimales del número π, el número de veces que aparece cada cifra (0-9) es la siguiente (formato *cifra:apariciones*):

$$0:45,\ 1:59,\ 2:54,\ 3:50,\ 4:53,\ 5:50,\ 6:48,\ 7:36,\ 8:53,\ 9:52$$

Define un vector con dichos valores, generando un diagrama de sectores en 3 dimensiones (o barras) con su correspondiente leyenda ¿Qué número se repite más? ¿Cuál es el que se repite menos?

3.6.2. Propuestos como actividad independiente

1. Escribe en diferentes ficheros .m los comandos necesarios para representar gráficamente las siguientes funciones 2y 3D, en el rango y con el paso indicado. Emplea diferentes colores, marcas y trazos.

a)	$f(x) = x^2 + y^2$	$-2 \le x, y \le 2$	con paso 0.2
b)	$f(x) = (x-1)^2 + (y-1)^2$	$-2 \le x, y \le 4$	con paso 0.1
c)	$f(x) = x^3 - x + 1$	$-5 \le x \le 5$	con paso 0.1
d)	$f(x) = x \cdot e^{-x^2}$	$-3 \le x \le 3$	con paso 0.02
e)	$f(x) = \frac{1}{1+x^2}$	$-10 \le x \le 10$	con paso 0.05
f)	$f(x) = x \cdot sen(x)$	$-12 \cdot \pi \le x \le 12 \cdot \pi$	con 500 puntos
g)	$f(x) = sen(e^x)$	$-\pi \le x \le \pi$	con 300 puntos
h)	$z = e^{-x^2-y^2} \cdot cos(3x^2 + 3y^2)$	$-2 \le x, y \le 2$	con paso: 0.1
i)	$z = \frac{cos(\frac{(x^2+y^2)}{4})}{3+x^2+y*2}$	$-5 \le x, y \le 5$	con paso: 0.2

2. Usa comandos *subplot* para representar las gráficas c-f del ejercicio anterior. Usa un color y trazo diferente para cada gráfica.

3. Representa en la misma gráfica y con un único comando *plot* las funciones $f(x) = e^{-x^2}$ y $g(x) = 2 \cdot e^{-x^2}$ en el rango $-3 \leq x \leq 3$ con 500 puntos. La primera función debe pintarse en rojo con trazo continuo y la segunda en negro con trazo discontinuo.

4. Considerando los pares de puntos del plano cartesiano $(1,1)(2,3)(-7,2)(0,4)$, representa los puntos en una gráfica como pentagramas rojos, incluyendo una recta que una los puntos $(-7,2)(2,3)$. Sugerencia, el comando *hold on*.

5. Para representar una curva definida en forma paramétrica, donde $x(t) = f(t)$ e $y(t) = g(t)$, se genera un vector, p.e. **t**, con el comando *linspace(t_{min},t_{max},n)*, representando posteriormente con el comando *plot(f(t),g(t))*. Representa la curva $x(t) = sen(2 \cdot t) + sen(t)$, $y(t) = -cos(2 \cdot t) - cos(t)$ en el rango $-\pi \leq t \leq \pi$.

6. Para representar una curva en coordenadas polares, donde $r = f(\alpha)$ se genera el vector, p.e. **a**, con el comando *linspace(a_{min},a_{max},n)* representando con el comando *polar(a,$f(\alpha)$)*. Representa en coordenadas polares $r = \frac{sen(\alpha) \cdot \sqrt{abs(cos(\alpha))}}{1.4 + sen(\alpha)} - 2 \cdot sen(\alpha) + 2$ en el rango $0 \leq \alpha \leq 2 \cdot \pi$.

7. El hipotético número de personas que aprobaron una cierta oposición por comunidad autónoma, está reflejado en la tabla 7. Escriba un fichero .m con las órdenes necesarias para crear una gráfica de sectores (en 3 dimensiones) de los datos estadísticos anteriores y que contenga las leyendas de las correspondientes comunidades autónomas.

Comunidad autónoma	Nº de aprobados
Canarias	181
País Vasco	102
Madrid	98
Cataluña	53
Asturias	49
Extremadura	23
Andalucía	8
Otras	6
Total	520

3.7. TAREA

La evaluación de esta práctica, se realiza a través de un cuestionario sobre el contenido de este documento y las actividades realizadas durante su correspondiente sesión práctica. El cuestionario estará disponible para su resolución el viernes 26 de febrero de 12 a 18 horas a través del campus virtual.

Práctica 4

Programación básica

4.1. INTRODUCCIÓN

El objetivo de esta práctica es adquirir destrezas en la elaboración de programas básicos que incluyen únicamente sentencias secuenciales. La estructura más habitual de uno de estos programas básicos, parte de una lectura de datos de entrada, para realizar con ellos unos cálculos, y finalmente mostrar sus resultados correspondientes por pantalla. Dado que ya se ha experimentado con la realización de operaciones, el siguiente paso es conocer los comandos para realizar tanto la lectura de datos, como su escritura por pantalla.

4.2. LECTURA DE DATOS

Tanto Matlab como Octave permiten obtener valores de teclado para las variables a través del comando *input*. La llamada a *input* requiere un parámetro, indicado entre comillas simples, que contiene el mensaje a mostrar por pantalla al requerir el dato. Al ejecutarse, el comando detiene la ejecución del programa hasta que el usuario teclea y pulsa *enter*/*return*. El valor tecleado se asigna a la variable indicada a la derecha de la expresión de asignación, apareciendo desde ese momento dicha variable en el espacio de trabajo. Las siguientes líneas muestran varios ejemplos de lectura de datos:

```
>> x=input('Introduce el valor de la variable x ')
Introduce el valor de la variable x 23
x =
    23

>> y=input('Introduce el valor de la variable y ');
Introduce el valor de la variable y 45
```

Observa el espacio que se coloca al final del mensaje, permite evitar que el dato introducido aparezca pegado como en el siguiente caso:

```
>> z=input('Introduzca el dato');
Introduzca el dato23
```

Es posible usando el código *\n* forzar a que se recoja el dato en la línea siguiente:

```
>> p=input('Introduzca el dato\n');
Introduzca el dato
12
```

De igual forma puede leerse un vector, una matriz, valores reales, etc., teniendo presente hacer uso de corchetes en los primeros dos casos

```
>> vec=input('Introduce un vector\n')
Introduce un vector
[2 3.4 5 1]

vec =     2.0000     3.4000     5.0000     1.0000

>> M=input('Introduce una matriz\n')
Introduce una matriz
[2 3 1 ; 2 3 4]

M =
     2     3     1
     2     3     4
```

El comando *input*, al igual que el resto de comandos, puede incluirse en un programa .m, para leer los valores necesarios para el cálculo posterior.

4.3. CONTROL DEL FORMATO DE SALIDA POR DEFECTO

Para controlar el formato de los resultados que Matlab muestra por pantalla se puede usar el comando *format*. Es importante observar que este comando afecta exclusivamente al formato de salida por pantalla, ya que Matlab realizará siempre los cálculos con la precisión adecuada al tipo de los operandos. El comando *format* puede recibir los siguientes argumentos:

short: coma fija con 4 decimales (por defecto)
long: coma fija con 15 decimales
hex: cifras hexadecimales
bank: números con dos cifras decimales
short e: notación científica con 4 decimales

short g: notación científica o decimal, dependiendo del valor

long e: notación científica con 15 decimales

long g: notación científica o decimal, dependiendo del valor

rational: expresa los números racionales como cocientes de enteros

compact: produce una salida más compacta eliminando los saltos de línea que se intercalan entre el nombre de la variable y el contenido de la misma.

loose: tiene el efecto opuesto a *compact* (opción por defecto).

Algunos ejemplo ilustrativos del uso de *format*s:

```
>> format long e
>> n = 23.45678
n =
    2.345678000000000e+001
>> format compact
>> n
n =
    2.345678000000000e+001
>> format short e
>> n
n =
  2.3457e+001
>> format
>> n
n =
   23.4568
```

Observar que invocar el comando *format* sin argumentos es equivalente a establecer el formato por defecto (*format short; format loose*).

4.4. ESCRITURA DE DATOS

Los comandos más habituales para la salida de datos por pantalla son *disp* y *fprintf*. Los siguientes ejemplos muestran la salida del mensaje *Hola* por pantalla, con ambos comandos para mostrar sus diferencias.

```
>> disp('Hola')
Hola
>> fprintf('Hola')
Hola>>
```

El uso del comando *disp* incluye el final de línea que con *fprintf* debe indicarse de forma explícita haciendo uso del código *\n*. Con el comando *disp* podemos mostrar también el contenido de una variable o el resultado de una operación

```
>> a=23;
>> disp(a);
    23
>> disp(2*a);
    46
```

Por contra, el comando *fprintf* facilita incluir variables en el mensaje a mostrar. Los siguientes comandos se han introducido en el fichero *doble.m* realizando el cálculo del doble de un valor almacenado en **a**.

```
a=input('Introduzca un valor entero');
fprintf('El doble de %d es %d\n',a,2*a);
```

La ejecución desde la ventana de comandos mediante la llamada a *doble*:

```
>> doble
Introduzca un valor entero 23
El doble de 23 es 46
```

El comando *fprintf* indica la cadena a mostrar, entre comillas, marcando con códigos de control donde colocar el contenido de las variables indicadas a continuación del mensaje a presentar por pantalla. Los códigos de control más habituales son los siguientes:

%d para expresar el dato en formato entero (sin decimales)
%f es para reales en formato de coma fija
%g es para reales en formato de coma flotante
%e es para presentar en notación científica
%c es para mostrar un carácter
%s es para mostrar una cadena de caracteres

En ocasiones nos puede interesar establecer un número determinado de cifras a mostrar. En el siguiente ejemplo, se limita el número de cifras decimales a mostrar:

```
>> fprintf('La raíz de 2 es %f\n',sqrt(2));
La raíz de 2 es 1.414214
>> fprintf('La raíz de 2 es %0.1f\n',sqrt(2));
La raíz de 2 es 1.4
>> fprintf('La raíz de 2 es %0.2f\n',sqrt(2));
La raíz de 2 es 1.41
```

Varios ejemplos mostrando el valor de *pi*:

```
>> fprintf(' %f \n', pi)
 3.141593
>> fprintf('%.2f \n', pi)
3.14
>> fprintf('%+.2f \n', pi)
+3.14
```

```
>> fprintf('%12.2f \n', pi)
        3.14
>> fprintf('%012.2f \n', pi)
000000003.14
>> fprintf('%e \n', pi)
3.141593e+00
>> fprintf('%E \n', pi)
3.141593E+00
>> fprintf('%.3e \n', pi)
3.142e+00
```

Es posible también controlar estos aspectos del formato mediante argumentos usando * en la cadena de formato para indicar que ese dato hay que tomarlo del siguiente argumento:

```
>> longitud = 8; decimales = 3; numero = 23.9876E3;
>> fprintf('%*.*f\n', longitud, decimales, numero);
23987.600
>> numero = 1.4

numero =
    1.4000

>> numero = 1.4;
>> fprintf('%*.*f\n', longitud, decimales, numero);
   1.400
```

Observa como indicar el número de dígitos del número (**longitud**, en el ejemplo) solo tiene efecto si es posible presentar el número con menos caracteres. En ese caso, se añaden espacios a la izquierda del número hasta completar la longitud indicada en el formato.

4.4.1. Particularidades con vectores y matrices

Un aspecto singular del comando *fprintf* en Matlab es cómo opera sobre vectores y matrices. Observa las diferencias en las salidas proporcionadas por los siguientes comandos:

```
>> vector = [0:10:100];
>> fprintf('%4d\n', vector);
   0
  10
20
30
40
50
60
70
80
90
100
>> fprintf('%4d ', vector);
   0   10   20   30   40   50   60   70   80   90  100 >>
>> matriz = [ 1 2 3; 4 5 6; 7 8 9]
```

```
matriz =
     1     2     3
4     5     6
7     8     9
>> fprintf('%4d\n', matriz);
   1
4
7
2
5
8
3
6
9
>> fprintf('%4d %4d %4d\n', matriz);
   1    4    7
2    5    8
3    6    9
>> fprintf('%4d ', matriz);
   1    4    7    2    5    8    3    6    9 >>
```

Aprovechamos el siguiente ejemplo para mostrar el modo de presentar elementos de un vector, su longitud, o el resultados de aplicar la función *sort* para ordenar el vector en sentido ascendente o descendente, o *sum* para mostrar la suma de sus elementos

```
v1=[5 8 9 0 -3];
v2=[4 -1 0 0 2];
disp(v1);
disp(v2);

disp('Tercera posición de v1');
disp(v1(3));
disp(Dimensión de v2');
disp(length(v2));
disp('v1 ordenado de menor a mayor');
disp(sort(v1));
disp('v1 ordenado de mayor a menor');
disp(-sort(-v1));
disp('muestra la suma de los elementos de v1');
disp(sum(v1));
```

4.4.2. Equivalencia entre *disp* y *fprintf*

Existe la posibilidad de usar el comando *disp* de forma similar al comando *fprintf*, pero necesitamos hacer uso del comando *sprintf*, de similar sintaxis a *fprintf* pero que escribe en una variable cadena en lugar de en pantalla. Esa cadena se la suministramos a *disp* como argumento. Si bien puede resultar más engorroso, para obtener una salida similar en nuestro archivo *doble.m*:

```
a=input('Introduzca un valor entero ');
disp(sprintf('El doble de %d es %d\n',a,2*a));
```

La ejecución produciría:

```
>> doble
Introduzca un valor entero 14
El doble de 14 es 28
```

Finalmente, podemos invocar el comando *type* seguido del nombre de un fichero de texto para imprimir por pantalla el contenido del fichero.

```
>> type doble.m
a = input('Introduzca un valor entero: ');
salida = sprintf('El doble de %d es %d\n', a, 2*a));
disp(salida)
```

4.5. Operaciones básicas

En sesiones previas se han tratado operaciones entre contenidos de variables de diverso tipo. En este apartado se incluyen dos versiones que tras leer un valor numérico, calculan su doble y su raíz cuadrada. La primera almacena los cálculos en otra variable

```
x=input('introduzca un número ');
doble=2*x;
raiz=sqrt(x);
fprintf('El doble de %d es %d y su raíz cuadrada %f\n',x,doble,raiz);
```

La segunda versión no utiliza variables para almacenar, sino que realiza el cálculo directamente al mostrar, reduciendo por tanto el uso de variables

```
x=input('introduzca un número ');
fprintf('El doble de %d es %d y su raíz cuadrada %f\n',x, 2*x, sqrt(x));
```

4.6. Ejercicios

4.6.1. De apoyo durante la sesión

1. Realiza un programa que lea un número por teclado y lo imprima en pantalla (en este orden) como entero, como número real con tres cifras decimales o en notación científica con tres cifras decimales.

2. Realiza un programa que lea por teclado un valor de temperatura en ºF y lo convierta a ºC. La temperatura en ºC se debe mostrar con dos decimales.

3. Realiza un programa que lea la duración de un proceso que viene dado en horas, minutos y segundos y devuelva por pantalla la duración total en segundos.

4. Realiza un programa que lea el tiempo (en segundos) y la distancia (en pulgadas) en los que una bicicleta ha realizado un determinado recorrido y muestre por pantalla la velocidad expresada en kilómetros/hora sin decimales.

5. Realiza un programa que tras leer el volumen de soluto y de la disolución, ambos en mililitros, muestre por pantalla el porcentaje volumen-volumen[1] con un decimal.

6. Tras conocer el número de moles, volumen (l) y temperatura (K) de un gas ideal, realiza un programa que calcule la presión absoluta (atm) con tres decimales[2].

7. La ley de gravitación universal describe la interacción gravitatoria entre cuerpos con masa[3]. Realiza un programa que tras obtener las masas (kg), y la distancia (m) de separación entre dos cuerpos, calcule la fuerza ejercida entre ambos y la muestre con diez cifras decimales.

8. Conociendo que la fórmula para calcular el pH de una solución acuosa es el opuesto del logaritmo en base 10 de la actividad de los iones de hidrógeno $pH = -log_{10}[aH+]$. Realiza un programa que conociendo que el ácido clorhídrico es un ácido fuerte, tras leer un valor de concentración molar, calcule su pH.

4.6.2. Propuestos como actividad independiente

1. Escribe un programa que tras leer un valor en grados calcule su coseno con la función *cos*.

2. Define dos variables **pvp** y **desc** que contendrán respectivamente el precio de un producto y un descuento expresado en %. Calcular el precio después de aplicado el descuento.

3. El supermercado X tiene un determinado producto con una oferta 3×2. Definir dos variables **pvp1** y **pvp3x2** que contendrán respectivamente el precio de una unidad, y el precio en oferta de tres unidades. Calcular la diferencia de precio por unidad comprando una única unidad o tres unidades aplicando la oferta.

[1] https://es.wikipedia.org/wiki/Concentración
[2] https://es.wikipedia.org/wiki/Ley_de_los_gases_ideales
[3] https://es.wikipedia.org/wiki/Ley_de_gravitación_universal

4. Define dos variables **pvp** e **imp** que contendrán respectivamente el precio de un producto y un impuesto en %. Calcular el precio después de aplicar el impuesto.

5. Si la edad de un padre actualmente es cinco veces la suma de las edades de sus hijos, y ocho veces la edad del mayor. Sabiendo que dentro de 22 años la suma de las edades de los hijos será sólo 10 años menor que la edad del padre, plantea el sistema de ecuaciones y resuelve con Matlab para obtener las edades actuales de padre e hijos.

6. En una granja con cerdos, vacas y gallinas, el número total de animales es de 450, siendo el de vacas cuatro veces el de cerdos. Si sabemos además que el número de total de patas de animales es de 1200, plantea el sistema de ecuaciones y resuelve con Matlab.

7. Conocidas la nota de teoría y práctica de un estudiante, además de los pesos de cada parte en la evaluación final de la asignatura, realiza un programa que determine la calificación final con un decimal.

8. Conociendo que la presión se define como la fuerza normal ejercida por unidad de área, realiza un programa que dadas la presión ($Pa = N/m^2$) y el área (m^2) sobre la que actúa, calcule y muestre la fuerza normal (N) ejercida con tres decimales.

9. Leer las variables **m** y **a** conteniendo respectivamente la masa y la aceleración de un cuerpo. Calcula la fuerza siguiendo la segunda ley de Newton $F = m \times a$.

4.7. TAREA

La evaluación de esta práctica, se realiza a través de un cuestionario sobre el contenido de este documento y las actividades realizadas durante su correspondiente sesión práctica. El cuestionario estará disponible para su resolución el viernes 5 de marzo de 12 a 18 horas a través del campus virtual.

Práctica 5

Sentencias de control condicionales

5.1. Introducción

Una vez realizada una colección de ejercicios con sentencias secuenciales y las utilidades para leer y mostrar datos, el objetivo de esta nueva práctica es la familiarización con las sentencias de control condicionales.

5.2. Sentencia condicional

La sentencia condicional en lenguaje m hace uso de la palabra reservada *if* con la sintaxis dependiente de si se requiere un condicional simple o doble en sus diferentes variantes:

```
% Condicional simple
if <predicado>
 <acciones_if>
end

% Condicional doble
if <predicado>
 <acciones_if>
else
 <acciones_else>
end

% Condicional doble con elseif
if <predicado>
 <acciones_if>
elseif ~<predicado>
 <acciones_else>
end
```

La siguiente muestra de código, ilustra el uso del condicional simple, incluso anidado, con diversos ejemplos con variadas condiciones:

```
x=input('Introduce el valor de la x\n');%Lee un valor

%Escribe si el valor es mayor que 10
if x>10
    fprintf('El valor %d es mayor que diez\n',x);
end

%Escribe si el valor es igual a 10
if x==10
    fprintf('El valor %d es igual a diez\n',x);
end

%Escribe si el valor es múltiplo de 10, a través del resto
%de la división entera entre 10
if mod(x,10)==0
    fprintf('El valor %d es múltiplo de diez\n',x);
end

%Escribe si el valor es distinto a 10
if x~=10
    fprintf('El valor %d es distinto a diez\n',x);
end

%Condición doble escribe si el valor es mayor o igual que 0
%y menor o % igual que 10
if x>=0 && x<=10
    fprintf('El valor %d está entre cero y diez\n',x);
end

%Alternativa al anterior con dos sentencias if
if x>=0
    if x<=10
        fprintf('El valor %d está entre cero y diez\n',x);
    end
end
```

Para ilustrar un ejemplo de aplicación de un condicional doble, resolvemos el siguiente sistema de ecuaciones lineal

$$\begin{cases} 5x+7y-2z=1 \\ 10x-8y-3x=6 \\ 3x+2y+4z=11 \end{cases}$$

Resolviendo $A \times X = B$, tras verificar si es resoluble comprobando que el determinante de A no sea nulo.

```
clc;
clear all;
A=[5 7 -2;10 -8 -3;3 2 4];
display('Determinante de A');
```

```
disp(det(A));
if det(A)~=0
  B=[1 6 11];
  B=B';
  display('Valores x,y,z del sistema');
  disp(inv(A)*B);
else
  disp('No hay solución');
end
```

Una particularidad del lenguaje m es la existencia de la variante *elseif*, que en el caso de los condicionales dobles, permite en ciertos casos redactar un código más compacto, dado que esta variante no requiere un *end* al final de la sentencia, mientras que si utilizáramos la combinación *else if*, sería necesario el *end* para cerrar la nueva sentencia *if* introducida. En el siguiente ejemplo pueden apreciarse condicionales dobles con ambas variantes, mostrando la forma en la que la variante *elseif* queda embebida dentro de la sentencia condicional *if* que le precede:

```
%Usando elseif
if( (a>0 || b>c) && (d>a || d<5) )
    a=0;
    d=b+c;
    c=a-b;
elseif (c>0)
        d=-d;
    else
        d=2*c;
end

%Usando else if
if( (a>0 || b>c) && (d>a || d<5) )
    a=0;
    d=b+c;
    c=a-b;
else
    if (c>0)
        d=-d;
    else
        d=2*c;
    end
end
```

Para finalizar, mostramos el condicional múltiple, que se describe a través del comando *switch*, que facilita especificar múltiples condiciones. La sintaxis es la que se muestra a continuación:

```
switch (<expEntera>)
 case <val1>
  <acciones1>
 case <val2>
```

```
  <acciones2>
 case <val3>
  <acciones3>
 ...
 otherwise
  <accionesOther>
end
```

En el siguiente fragmento de código se presenta un ejemplo concreto de condicional múltiple. Tras leer la cantidad de unidades adquiridas de un producto se aplica un precio en función de dicho número.

```
cantidad=input('Introduzca la cantidad: ');

switch(cantidad)
 case 1
  precio=100;
 case 2
  precio=95;
 case 3
  precio=90;
 otherwise
  precio=85;
end

fprintf('El precio es %d\n', precio);
```

5.3. EJERCICIOS

5.3.1. De apoyo durante la sesión

1. Realiza un programa que tras leer los tres coeficientes de una ecuación de segundo grado, $ax^2 + bx + c = 0$, muestre por pantalla el mensaje *Con raíces reales* en caso de que así sea.

2. Realiza un programa que tras leer los tres coeficientes de una ecuación de segundo grado, $ax^2 + bx + c = 0$, muestre por pantalla el mensaje *Con raíces reales* o *Sin raíces reales* según el caso.

3. Realiza un programa que tras leer los tres coeficientes de una ecuación de segundo grado, $ax^2 + bx + c = 0$, muestre por pantalla con tres decimales las raíces reales si existen. En el hipotético caso de que las raíces o soluciones de la ecuación fueran imaginarias debe mostrar el mensaje *Raíces imaginarias*.

4. Desde el punto de vista del diseño de una estrategia comercial, la edad es un factor esencial para determinar el público al que se venderá un producto. Aunque no existe una escala definida, tomaremos como referencia la propuesta del siguiente enlace[1].

   a) Niños (de 0-14 años)
   b) Adolescentes (15-19 años)
   c) Jóvenes (20-29 años)
   d) Adultos (30-40 años)
   e) Adultos B (41-59 años)
   f) Ancianos (60 o más)

   Realiza un programa que tras leer el año de nacimiento de una persona, determine el grupo de edad al que pertenece la persona: Niños, Adolescentes, Jóvenes, Adultos, Adultos B o Ancianos.

5. Escribe un programa, que dada una fecha (dos variables contendido día y mes en formato numérico), muestre por pantalla el mensaje *Fecha correcta* o *Fecha incorrecta*, según si es válida o no. Nota: No es necesario considerar años bisiestos.

5.3.2. Propuestos como actividad independiente

1. Realiza un programa que lea un número entero y escriba por pantalla un 1 si el número es negativo y múltiplo de 3, y 0 en caso contrario.

2. Escribe un programa que lea un número y determine si es entero o real.

3. Escribe un programa que lea dos números, y determine si el segundo es múltiplo del primero.

4. Realiza un programa que lea tres números enteros y muestre por pantalla el mensaje *Cumple* si el producto de dos de ellos es igual al tercero, o el mensaje *No cumple* en caso contrario.

5. Realiza un programa que tras leer cinco valores de temperatura, y la media prevista para el mes, indique los valores que han superado el valor de la media.

6. La ecuación general o implícita de una recta en el plano XY es $r(x,y) = ax + by + c$ de coordenadas (A,B) puede estar situado: a) Sobre la recta, si $r(A,B) = 0$, a un lado de la recta si $r(A,B) > 0$, y c) al otro lado de la recta si $r(A,B) < 0$, Escribe un programa

[1] https://mglobalmarketing.es/blog/marketing-para-empresas-como-se-define-el-target/

en el que se solicite un vector, **r** (de 3 celdas para introducir a,b,c), luego solicite las coordenadas (A,B) del punto y muestre por pantalla con una frase la situación del punto respecto a la recta.

7. La lipoproteína de alta densidad o HDL es considerada buena para el cuerpo al extraer colesterol del flujo sanguíneo. Realizar un programa que indique si el nivel de HDL indica un riesgo mayor (< 40), normal, o muy bajo (≥ 60) de tener una enfermedad cardíaca.

8. Realiza un programa que calcule el índice de masa corporal según la expresión que relaciona el peso en kilogramos y la altura en metros, e indique si la persona tiene infrapeso ($IMC < 18{,}5$), es normal, tiene sobrepeso ($IMC \geq 25$) o es obesa ($IMC \geq 130$). El programa debe controlar la coherencia de los valores introducidos.

$$IMC = \frac{peso}{altura^2}$$

9. Diversos estudios apuntan a que el nivel normal de glucosa en sangre antes de irse a dormir debe estar entre 100 y 140 mg/dl. Hacer un programa que lea la concentración de glucosa en sangre en mg/dl y determine si el dato indica hipoglucemia, normal o hiperglucemia.

10. Realiza un programa, que dada una fecha del siglo XXI (representada por el día, el mes y el año en formato numérico), muestre la fecha del día anterior. Debe mostrar error en caso de la fecha no ser correcta.

5.4. TAREA

La evaluación de esta práctica, se realiza a través de un cuestionario sobre el contenido de este documento y las actividades realizadas durante su correspondiente sesión práctica. El cuestionario estará disponible para su resolución el viernes 12 de marzo a las 12 horas, al sábado 13 de marzo a las 12 horas, a través del campus virtual.

Práctica 6

Sentencias de control repetitivas

6.1. INTRODUCCIÓN

Tras el trabajo realizado en la práctica precedente con las sentencias condicionales, el objetivo de esta práctica es la familiarización con las sentencias de control repetitivas.

6.2. SENTENCIA REPETITIVA

La sentencia repetitiva *while* en Matlab y Octave, tiene la siguiente estructura:

```
while <predicado>
    <acciones>
end
```

Como alternativa se cuenta con la sentencia *for*, donde en la cabecera se indica en primer lugar el valor inicial de la variable iteradora, a continuación, el paso de dicha variable, finalizando con el último valor que dicha variable puede alcanzar.

```
for <variable>=<valor_inicial>:<paso>:<valor_final>
    <acciones>
end
```

Mostramos a continuación varios ejemplos, a modo ilustrativo con ambas sentencias. El primero de ellos muestra **n** veces la misma frase:

```
n=input('Introduce el valor de n\n');

%Escribe n veces la frase con la sentencia for
for i=1:n
    fprintf('Tengo que hacer más ejercicios\n');
end
```

```
%Escribe n veces la frase con la sentencia while
i=1;
while i<=n
    fprintf('Tengo que hacer más ejercicios\n');
    i=i+1;
end
```

Para este caso concreto, es indiferente si la cuenta del bucle se realiza hacia adelante o hacia atrás, el efecto resultante en pantalla es el mismo:

```
n=input('Introduce el valor de n\n');

%Escribe n veces la frase con la sentencia for
for i=n:-1:1
    fprintf('Tengo que hacer más ejercicios\n');
end

%Escribe n veces la frase con la sentencia while
i=n;
while i>=1
    fprintf('Tengo que hacer más ejercicios\n');
    i=i-1;
end
```

En ocasiones, la acción repetitiva a realizar depende del valor de la variable contadora, en el siguiente ejemplo, mostramos el valor de dicha variable contadora en cada iteración:

```
n=input('Introduce el valor de n\n');

%Muestra los números enteros de 1 a n usando la sentencia for
for i=1:n
    fprintf('%d\n',i);
end

%Muestra los números de 1 a n usando la sentencia while
i=1;
while i<=n
    fprintf('%d\n',i);
    i=i+1;
end
```

Si la cuenta se realizara al revés, ahora sí que obtenemos un resultado diferente por pantalla:

```
n=input('Introduce el valor de n\n');

%Muestra los números de n a 1 usando la sentencia for
for i=n:-1:1
    fprintf('%d\n',i);
end

%Muestra los números de n a 1 usando la sentencia while
i=n;
```

```
while i>=1
    fprintf('%d\n',i);
    i=i-1;
end
```

En los siguientes ejemplos veremos que es posible hacer uso de un paso distinto al empleado en los ejemplos previos, 1 o −1, según nuestros intereses:

```
n=input('Introduce el valor de n\n');

%Muestra los números de 1 a n de tres en tres usando la sentencia for
for i=1:3:n
    fprintf('%d\n',i);
end

%Muestra los números de 1 a n de tres en tres usando la sentencia while
i=1;
while i<=n
    fprintf('%d\n',i);
    i=i+3;
end
```

6.3. ANIDACIÓN DE SENTENCIAS

6.3.1. Anidando repetitivas y condicionales

De la misma forma que los condicionales pueden anidarse, es posible combinar varios bucles, y por supuesto condicionales y bucles. Un ejemplo de combinación de repetitiva con condicional es el siguiente ejemplo que muestra, con ambas sentencias, los múltiplos de 7 en el rango $[1, n]$:

```
n=input('Introduce el valor de n\n');

%Muestra múltiplos de 7 de 1 a n usando la sentencia for
for i=1:n
    if mod(i,7)==0
        fprintf('%d\n',i);
    end
end

%Muestra múltiplos de 7 de 1 a n usando la sentencia while
i=1;
while i<=n
    if mod(i,7)==0
        fprintf('%d\n',i);
    end
    i=i+1;
end
```

Al combinar repetitivas con condicionales, puede ser de interés contabilizar el número de veces que se cumple una condición, por ejemplo, modificando el ejemplo anterior para mostrar no los múltiplos de 7, sino el número total de ellos en el rango $[1, n]$:

```
n=input('Introduce el valor de n\n');

%Muestra y cuenta múltiplos de 7 de 1 a n usando la sentencia for
c=0;
for i=1:n
    if mod(i,7)==0
        fprintf(' %d\n',i);
        c=c+1;
    end
end
fprintf('Encontrados %d múltiplos de 7\n',c')

%Muestra y cuenta múltiplos de 7 de 1 a n usando la sentencia while
c=0;
i=1;
while i<=n
    if mod(i,7)==0
        fprintf(' %d\n',i);
        c=c+1;
    end
    i=i+1;
end
fprintf('Encontrados %d múltiplos de 7\n',c')
```

En este caso hay dos variables contadoras, **i** cuenta la iteración actual, mientras que **c** cuenta el número de valores que cumplen la condición.

6.3.2. Anidando repetitivas

También es posible la anidación de sentencias repetitivas. Partiendo de un bucle sencillo con la sentencia *for*, que provoca la repetición de una acción:

```
for i=1:4
    fprintf('Hola');
end
```

La salida correspondiente por pantalla sería:

```
HolaHolaHolaHola>>
```

SI añadimos un espacio en blanco al final de la cadena, se modifica de la siguiente manera:

```
for i=1:4
    fprintf('Hola ');
end
```

```
Hola Hola Hola Hola >>
```

Si en su lugar introducimos el código *\n* que se corresponde al salto de línea:

```
for i=1:4
    fprintf('Hola');
    fprintf('\n');
end
```

¿Qué ocurre si introducimos otro bucle interno?

```
for i=1:4
    for j=1:2
        fprintf('Hola ');
    end
    fprintf('\n');
end
```

La salida por pantalla cambia significativa. Si analizamos el bucle interior, debe mostrar dos veces la cadena *Hola* con sus correspondientes espacios en blanco. El bucle externo, repite cuatro veces lo que contiene, es decir, el bucle que escribe dos veces *Hola*, y un salto de línea. En definitiva, se repite cuatro veces la línea que contiene dos veces *Hola*

```
Hola Hola
Hola Hola
Hola Hola
Hola Hola
```

En lugar de mostrar *Hola*, veamos lo que ocurre al mostrar el contenido de las variables contadoras de cada bucle:

```
for i=1:4
    for j=1:2
        fprintf('%d ',j);
    end
    fprintf('\n');
end
```

Obtenemos las cuatro repeticiones de los dos valores de **j**.

```
1 2
1 2
1 2
1 2
```

6.4. ESTRUCTURA DE ACUMULADOR

Se incluye en este apartado la estructura necesaria para acumular operaciones en una variable, mostrando tanto el uso de sumas como productos:

```
%Suma los números de 1 a 10 utilizando for
s=0;
for i=1:10
    s=s+i;
end
fprintf('El sumatorio es %d\n',s)
%Suma los números de 1 a 10 utilizando while
s=0;
i=1;
while i<=10
    s=s+i;
    i=i+1;
end
fprintf('El sumatorio es %d\n',s)
O productos

%Multiplica los números de 1 a 10 utilizando for
s=1;
for i=1:10
    s=s*i;
end
fprintf('El factorial es %d\n',s)
% Multiplica los números de 1 a 10 utilizando while
s=1;
i=1;
while i<=10
    s=s*i;
i=i+1;
end
fprintf('El factorial es %d\n',s)
```

6.5. ALGUNOS EXTRAS

Destacar que es posible definir un bucle infinito, que no finaliza nunca, fácilmente con la sentencia *while*:

```
while 1
    ...
end
```

Existen dos comandos, *break* y *continue*, que dan posibilidades añadidas durante la ejecución de una iteración. El primero de ellos fuerza la finalización del bucle, incluso si fuera un bucle infinito, continuando la ejecución del programa tras la sentencia *end* que cierra dicho bucle:

```
y = [-2 -4 0 -4 3 7];
for i = 1 : length(y)
```

```
    if y(i) > 0
        break
    end
    z = y(i) + 6;
    disp(z)
end
```

Por otro lado, *continue*, salta a la siguiente iteración del bucle

```
for i = 1:7
  if (i == 6 || i == 7)
    continue
  end
  disp(i)
  beep
  pause(1)
end
```

6.5.1. Parte entera y resto

De cara a tener elementos para alguno de los ejercicios propuestos, conocida slas funciones *mod* o *rem* para obtener el resto de la división entera, debemos tener presente que si necesitamos obtener las unidades de cualquier número entero, bastaría con almacenar en una variable el resto de la división entre 10.

```
uni=mod(n,10);
```

Por otro lado, recordar que las funciones *fix*, *floor* y *round* nos dan respectivamente la parte entera de un número, el entero más próximo menor, y su redondeo, o entero más próximo. Observa en el ejemplo las diferencias:

```
>> 10/6
ans =   1.6667
>> fix(10/6)
ans =   1
>> floor(10/6)
ans =   1
>> round(10/6)
ans =   2
>> -10/6
ans = -1.6667
>> fix(-10/6)
ans = -1
>> floor(-10/6)
ans = -2
>> round(-10/6)
ans = -2
```

6.6. EJERCICIOS

6.6.1. De apoyo durante la sesión

Semana I

1. Realiza un programa que escriba n veces *Buenos días* en n líneas distintas.

2. Modifica el programa anterior numerando cada línea, a modo de ejemplo, la tercera sería:

```
3.- Buenos días
```

3. Realiza un programa que, tras leer n, escriba n veces el carácter − en la misma línea. A modo de ejemplo, si el programa leyera 5, debe mostrar por pantalla:

```
-----
```

4. Modifica el programa anterior para que en las posiciones que sean múltiplos de 4 muestre el carácter |. Por ejemplo, si el número fuera 8 debe mostrar:

```
---|---|
```

5. Modifica el programa anterior para que muestre tantas líneas como el valor de n, mostrando en cada línea tantos caracteres como el índice de la línea. A modo de ejemplo, si el valor introducido fuera 5, el programa debe mostrar:

```
-
--
---
---|
---|-
```

6. Modifica el programa anterior para que añada una última línea mostrando el número de veces que se escribe el carácter |. Por ejemplo, si el número fuera 5 debe mostrar:

```
-
--
---
---|
---|-
2
```

Semana II

1. Realiza un programa que, tras leer un número natural, n, mayor que cero, muestre por pantalla $\sum_{i=1}^{n} i^2$.

2. Realiza un programa que, tras leer un número natural, n, mayor que cero, muestre por pantalla $\sum_{i=1}^{n} i^2$, y la cantidad de números naturales menores que no son divisores.

3. Realiza un programa que tras leer un número natural, n, mayor que cero, muestre por pantalla con tres decimales el resultado de $\prod_{i=1}^{n} i^2 + \sum_{i=1}^{n} i^2$

4. Modifica el programa anterior para que tras leer dos números naturales, n y m, mayores que cero, muestre por pantalla con un decimal el resultado de $\frac{\prod_{i=1}^{n} i^2 + \sum_{i=1}^{n} i^2}{\prod_{i=1}^{m} i^2 + \sum_{i=1}^{m} i^2}$

5. Escribe un programa que, tras leer un número natural, indique por pantalla el número de cifras pares que contiene.

6.6.2. Propuestos como actividad independiente

1. Realiza un programa que lea dos números enteros y escriba por pantalla los números impares que hay entre ellos de mayor a menor.

2. Realiza un programa que lea un número natural y muestre por pantalla en orden decreciente todos los números naturales pares menores que dicho número. En caso de no ser un número natural debe indicar *Error*.

3. Escribe un programa que pida un número, n, y devuelva la tabla de multiplicar de ese número. Por ejemplo, si insertamos un 8 que devuelva lo siguiente:

```
8 por 1 igual a 8
8 por 2 igual a 16
8 por 3 igual a 24
8 por 4 igual a 32
8 por 5 igual a 40
8 por 6 igual a 48
8 por 7 igual a 56
8 por 8 igual a 64
8 por 9 igual a 72
8 por 10 igual a 80
```

4. Escribe un programa que lea un número entero, dibujando si es par con el propio número un cuadrado, y si fuera impar un triángulo. Los siguientes ejemplos muestran el resultado para el 5 y el 4:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
4 4 4 4
4 4 4 4
4 4 4 4
4 4 4 4
```

5. Realiza un programa que lea un número y muestre sus divisores pares.

6. Escribe un programa que solicite un número, n, y muestre todos sus divisores pares. Si no se encuentran divisores pares, se debe mostrar la frase *No hay divisores pares*.

7. Realiza un programa que lea un número y muestre sus divisores múltiplos de 5, además del número total de divisores.

8. Realiza un programa que lea números de forma indefinida hasta que se introduzca el 0, mostrando la suma de los valores introducidos.

9. Escribe un programa que tras leer el número de valores a leer a continuación, los lea, mostrando la media de dichos números con 2 decimales. No utilizar la función *mean* de Matlab.

10. Escribe un programa que lea números enteros hasta introducir el número -999. Luego debe mostrar la media con 3 decimales de los números introducidos sin incluir el -999. No utilizar la función *mean* de Matlab.

11. Escribe un programa que tras leer un número, indique por pantalla el número de cifras pares e impares que contiene.

12. Realiza un programa que determine si un número de tres cifras tiene la particularidad de ser un número de Armstrong. Se dice que un número de k dígitos es de Armstrong si la suma de cada dígito que lo compone elevado a k es igual a dicho número. Por ejemplo 153 es un número de Armstrong porque 13 + 53 +33 = 153. El programa devolverá *Sí* para indicar que la entrada es un número de Armstrong y *No* si no lo es. Para saber más de este tipo de números puedes visitar el enlace[1].

[1] http://en.wikipedia.org/wiki/Narcissistic_number

13. Escribe un programa que muestre por pantalla los n primeros valores de la sucesión $A_i = A_{i-1} \times i$ siendo $A_1 = 1$.

14. Realiza un programa que muestre únicamente el elemento n-ésimo de la sucesión de Fibonacci. Recuerda que en dicha sucesión $A_i = A_{i-1} + A_{i-2}$ siendo $A_1 = A_2 = 1$.

15. Escribe un programa que genere el enésimo valor de la sucesión $A_i = A_{i-1} + i^2$ siendo $A_1 = 1$.

16. El número personal está basado en la suma de los dígitos de la fecha de nacimiento, de esta forma, una persona nacida el 14/11/1975 realizaría el cálculo de la siguiente manera: $(1+4)+(1+1)+(1+9+7+5) = 29$ debiendo quedarnos con solo una cifra final $2+9 = 11$, $1+1 = 2$

17. Escribir un programa que tras leer la fecha de nacimiento de una persona calcule su número personal.

18. La suma de una serie Geométrica viene dada por: $r + r^2 + r^3 + ... + r^n$. Escribe un programa que solicite dos números r, n, y calcule y muestre dicha suma.

19. Escribe un programa que lea números ininterrumpidamente hasta que se introduzca un cero. Una vez que se acabe de introducir números, se debe mostrar la media aritmética y la media geométrica de los números introducidos. Notas: la media geométrica de n números $x_1, x_2, \ldots x_n$ tiene la siguiente expresión: $\sqrt[n]{x_1 \cdot x_2 \cdot \ldots x_n}$. La función *nthroot(a,n)* permite calcular la raíz n del número a.

6.7. TAREA

La evaluación de esta práctica, se realiza a través de un cuestionario sobre el contenido de este documento y las actividades realizadas durante su correspondiente sesión práctica. El cuestionario estará disponible para su resolución desde el viernes 26 de marzo a las 12 horas, al sábado 27 de marzo a las 12 horas, a través del campus virtual.

Ejercicio compendio prácticas 1-6

Las compañías eléctricas ofrecen a sus abonados distintos planes de energía en lo que establecen precios de consumo fijos durante la jornada o variables en distintas franjas horarias. Este ejercicio consiste en valorar cuál de los planes que ofrece una compañía imaginaria interesaría contratar para una familia también imaginaria.

La compañía X ofrece dos planes de energía, tal como se muestra en la siguiente tabla:

Plan	**Precio por kWh consumido**	**Horario**
Fijo	.5€	1-24
Variable por franjas	.3€ valle .65€ pico	0-12 valle 12-24 pico

Para simular el consumo para un hogar concreto, se genera un consumo aleatorio para cada hora de un día tipo de entre 0 y 2kWh.

A partir de los datos anteriores, tendremos que decidir cuál de los dos planes conviene calculando el coste diario para ambos planes y mostrando una gráfica con el coste de dichos planes para cada hora. Las siguientes líneas proponen soluciones para las distintas etapas involucradas en la resolución.

Simular el consumo entre 0 y 2kWh:

```
consumo = 2*rand(1,24);
```

Crear un vector con las horas de una jornada:

```
horas = 1:24
```

Inicializar variables con los precios de los planes de energía:

```
precioPlanFijo = 0.5;
precioPlanVariableValle = 0.3;
precioPlanVariablePico = 0.65;
```

Calcular los costes por hora y total diario para el plan fijo:

```
costePlanFijo = precioPlanFijo * consumo;
costePlanFijoDiario = sum(costePlanFijo)
```

Mostrar un mensaje con una precisión de dos decimales:

```
fprintf('El coste diario para el plan de precio fijo es de %.2f \n',costePlanFijoDiario);
```

Calcular los coste por hora y total diario para el plan variable:

```
costePlanVariable(1:11) = precioPlanVariableValle * consumo;
costePlanVariable(12:24) = precioPlanVariablePico * consumo;
costePlanVariableDiario = sum(costePlanVariable);
```

Mostrar un mensaje con una precisión de dos decimales:

```
fprintf('El coste diario para el plan de precio fijo es de %.2f \n',costePlanVariableDiario);
```

Mostrar un mensaje indicando la opción más rentable:

```
if (costePlanFijoDiario < CostePlanVariableDiario)
    disp('El plan fijo es mas rentable que el variable por franjas');
else
    disp('El plan variable por franjas es más rentable que el plan de precio fijo');
end
```

Mostrar las gráficas de costes por hora:

```
hold on
plot(horas,costePlanFijo,'k-');
plot(horas,costePlanVariable,'r-');
legend('Plan coste fijo','Plan coste variable por franjas horarias');
title('Diagrama comparativo de planes de energía')
hold off
```

¿Cómo modificarías el código anterior para ejecutar 10 veces la simulación?

Práctica 7

Estructuras de datos dimensionadas: Vectores

7.1. INTRODUCCIÓN

En las próximas tres prácticas se aborda, por medio de sentencias de control, el procesamiento de datos contenidos en conjuntos dimensionados homogéneos como son vectores, matrices y cadenas de caracteres. En esta primera práctica de la serie se trata el manejo de vectores.

7.1.1. Creación/inicialización

Un vector puede crearse, bien de forma directa en el código, o a través del uso de la ya conocida sentencia *input*, que permite que sea el usuario quien defina los valores contenidos en el vector. Las siguientes líneas sirven de recordatorio y ejemplo sobre alternativas disponibles para la inicialización de este tipo de variables:

```
%Definiendo el contenido de un vector
v1=[2 3 4 1 6];

%Estableciendo un rango para definir los elementos de un vector
v2=[1:0.5:30];

%10 valores enteros en el rango (0,100)
v3=randi([0 100],1,10);

%Definiendo elemento a elemento el contenido del vector
v4(1)=3;
v4(2)=4;
v4(3)=9;
```

```
%Leyendo cada elemento por teclado
v5(1)=input('Introduce un elemento del vector ');
v5(2)=input('Introduce un elemento del vector ');
v5(3)=input('Introduce un elemento del vector ');
v5(4)=input('Introduce un elemento del vector ');
v5(5)=input('Introduce un elemento del vector ');

%Leyendo cada elemento de teclado haciendo uso de un bucle
for i=1:5
    v6(i)=input('Introduce un elemento del vector ');
end

%Leyendo el vector de golpe (requiere introducir corchetes)
v7=input('Introduce el vector ');
```

7.1.2. Acceso

Una vez que los datos están cargados en variables del espacio de trabajo, será posible acceder a los mismos y analizarlos para diversos fines. Se muestran a continuación de modo ilustrativo algunas variantes de acceso al contenido de un vector, básicamente mostrar su contenido elemento a elemento (desde el inicio o desde el final), o comprobar si hay valores contenidos en el vector que cumplen una condición. Se muestra un primer ejemplo que cuenta en la variable **mayorquecero** el número de valores contenidos en el vector mayores que cero. Posteriormente, en el último bucle utilizamos dos variables, **npares** y **mediapares** para respectivamente contar, y calcular la media de los valores pares presentes en el vector. Además se crean los vectores **w** y **wp** que almacenan respectivamente dichos valores pares y sus posiciones, es decir, todos aquellos que cumplen la condición de ser par.

```
v=[2 3 4 1 6]; %Ejemplo de inicialización del vector

%Escribimos los datos del vector de izquierda a derecha. El uso %de los bucles facilita
for i=1:5
    fprintf('El valor en la posición %d del vector es %d\n', i, v(i));
end

fprintf('\n');

%Escribimos los datos del vector de derecha a izquierda, es %decir, al revés
for i=5:-1:1
    fprintf('El valor en la posición %d del vector es %d\n', i, v(i));
end

fprintf('\n');

%Variable que cuenta el número de valores mayores que 0 presentes en el vector
mayorquecero=0;
%Variable que acumula los mayores que 0
```

```
suma=0;
for i=1:5
    if v(i)>0
        mayorquecero = mayorquecero + 1;
        suma = suma + v(i);
    end
end
fprintf('El número de valores mayores que cero es %d\n', mayorquecero);

%Contabiliza y guarda los valores pares en otro vector
npares=0;
mediapares=0;
for i=1:5
    if mod(v(i),2)==0
        npares = npares + 1;
        mediapares = mediapares + v(i);
        w(npares)=v(i);
        wp(npares)=i;
    end
end
disp(mediapares/npares);
disp(w);
disp(wp);
```

7.2. EJERCICIOS SEMANA I

7.2.1. De apoyo durante la sesión

1. Según el trabajo *Irrigation Water Quality Standards and Salinity Management Strategies, Texas Agricultural Extension Service, The Texas A&M University System* (1996)[1] , para el cultivo de la papa, el valor máximo de concentración de cloruros en la solución del suelo, a partir de los cuales aparecen síntomas de toxicidad en los cultivos es de 525 mg/l. Realiza un programa que, tras leer, el número n de medidas de concentración de cloruros a leer, lea y almacene dichas medidas en un vector, indicando posteriormente el número de ellas que superan el valor de toxicidad.

2. Modifica el programa anterior para que además indique la media de los valores que cumplen la condición de superar el valor de toxicidad.

3. Modifica el programa anterior para que además muestre el valor mínimo de todas las medidas contenidas en el vector.

[1] https://cdn-ext.agnet.tamu.edu/wp-content/uploads/2019/03/EB-1667-irrigation-water-quality-standards-and-salinity-management-strategies-1.pdf

4. Modifica el programa anterior para que además muestre tanto el valor mínimo como su posición, en caso de contar con la presencia de dicho valor mínimo en varias posiciones, se indicará únicamente la última.

5. Modifica el programa del ejercicio 1 para que almacene los valores que cumplen dicha condición en un nuevo vector, mostrando su contenido por pantalla.

6. Modifica el programa del ejercicio anterior para que además determine si los elementos del vector resultante están *ordenados de forma ascendente*, *ordenados de forma descendente* o *desordenados* (mostrar dichos mensajes por pantalla en cada caso).

7.2.2. Propuestos como actividad independiente

1. Realiza un programa que lea 10 números enteros, los guarde en un vector, y posteriormente muestre todos los impares en sentido contrario (es decir desde el último introducido al primero). El programa debe devolver 0 en caso de que no haya ningún valor impar en el vector.

2. Realiza un programa que lea 10 números enteros, los guarde en un vector, y posteriormente muestre todos los múltiplos de 3, pero en sentido contrario (es decir desde el último introducido al primero). El programa debe devolver 0 en caso de que no haya ningún múltiplo en el vector.

3. Escribe un programa que lea 10 números enteros, los guarde en un vector. Posteriormente leeremos un valor a buscar en el vector, y almacenaremos en un nuevo vector las posiciones en las que encontremos dicho valor. La salida del programa consistirá en la impresión por pantalla de los elementos del vector resultante, o 0 en caso de no haberse encontrado dicho valor en el vector.

4. Realiza un programa que lea por teclado 10 números enteros, los guarde en un vector y a continuación imprima el mayor valor par y el mayor valor impar de todos ellos y su posiciones respectivas.

5. Escribe un programa que lea 10 números enteros, los guarde en un vector. Posteriormente leerá un valor a buscar en el vector, y almacenará en un nuevo vector las posiciones en las que no se encuentre dicho valor. La salida del programa consistirá en la impresión por pantalla de los elementos del vector resultante, ó 0 en caso de tener un vector resultante vacío.

6. En un vector **T**, guardamos los valores de las temperaturas de ebullición de diferentes líquidos. Escribe un programa que calcule las temperaturas mínima, máxima y media de **T**, mostrando por pantalla dichos valores.

7. Realiza un programa que tras leer un número entero, almacene en un vector todas sus cifras y las muestre.

8. Realiza un programa que primero lea por teclado un vector de 10 elementos enteros comprendidos en el intervalo [0,10). A continuación, el programa deberá generar el histograma de ese vector para un tamaño de agrupación de 2. Es decir, el primer elemento del vector **histograma** contendrá el número de veces que aparece un número en el intervalo [0,2) en el vector original; el segundo elemento indicará el número de veces que aparece un número en el intervalo [2,4) en el vector original y así sucesivamente hasta el intervalo [8,10). El programa deberá generar como salida el contenido del vector **histograma**.

9. Realiza un programa que indique si un vector para el que previamente se han leído su tamaño y elementos, está ordenado de mayor a menor. En tal caso, deberá imprimir *ordenado* en pantalla y *desordenado* en otro caso.

10. Realiza un programa que tras leer dos vectores de 5 elementos ordenados de menor a mayor, combine ambos en un nuevo vector que también deberá estar ordenado de menor a mayor.

7.3. DEPURACIÓN

Matlab incorpora diferentes herramientas de apoyo al programador. Así, por ejemplo, el editor realiza detección de errores (marcas en rojo) y funciones de autocompletado y ayuda en línea a medida que se va tecleando el código. En algunos casos, se realizan sugerencias para mejorar la eficiencia de los programas, de manera que la propia interfaz se encarga de implementarlas, si el usuario así lo desea.

Matlab ofrece también herramientas de depuración similares a las presentes en la mayoría de los entornos de programación. De esta forma, se puede realizar una ejecución de los programas paso a paso, visualizando además los contenidos de las variables empleadas, permitiendo al programador identificar errores de programación. Para mostrar el modo de realizar la depuración, emplearemos el ejemplo del cálculo del sumatorio de los n valores contenidos en un vector, con el siguiente código:

```
clear;clc;
n=input('Introduce el número de elementos ');
for i=1:n
    v(i)=input('Introduce el valor ');
end
suma=0;
for i=1:n
    suma=suma+v(i);
end
fprintf('El resultado del sumatorio es %d\n',suma);
```

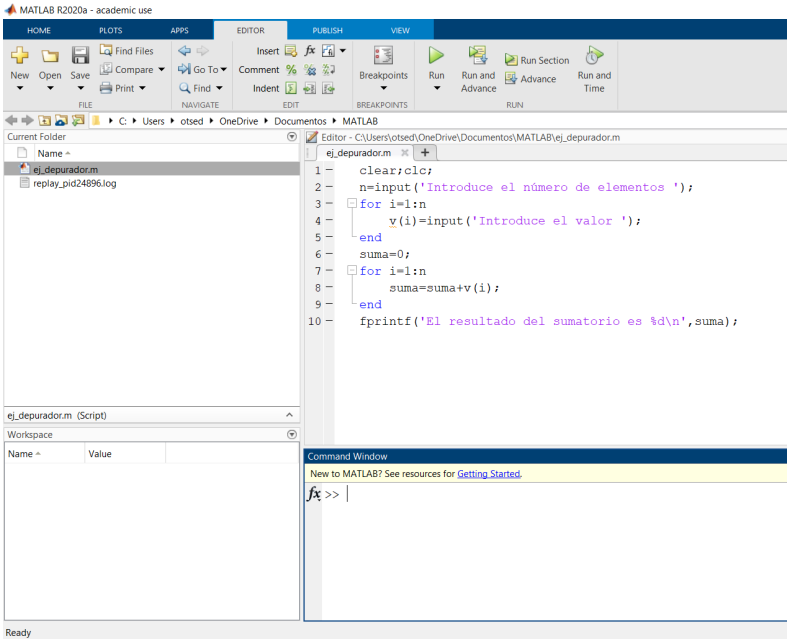


Figura 7.1: Programa m introducido en el editor

Un mecanismo más burdo de depuración es la omisión del carácter ; al final de cada operación que se realiza en el programa, lo cual provoca la escritura del resultado de la operación o contenido de la variable indicada. Sin embargo existen otras herramientas que utilizaremos durante la depuración, como las siguientes:

- Puntos de ruptura
- Ejecución paso a paso
- Inspección de variables

Para mostrar las opciones de depuración, lo primero que debemos realizar es introducir el código del programa a depurar, si no lo tenemos ya, utilizando el editor de ficheros M, escogiendo desde la pestaña *Home* la opción *New Script*, ver figura 7.1.

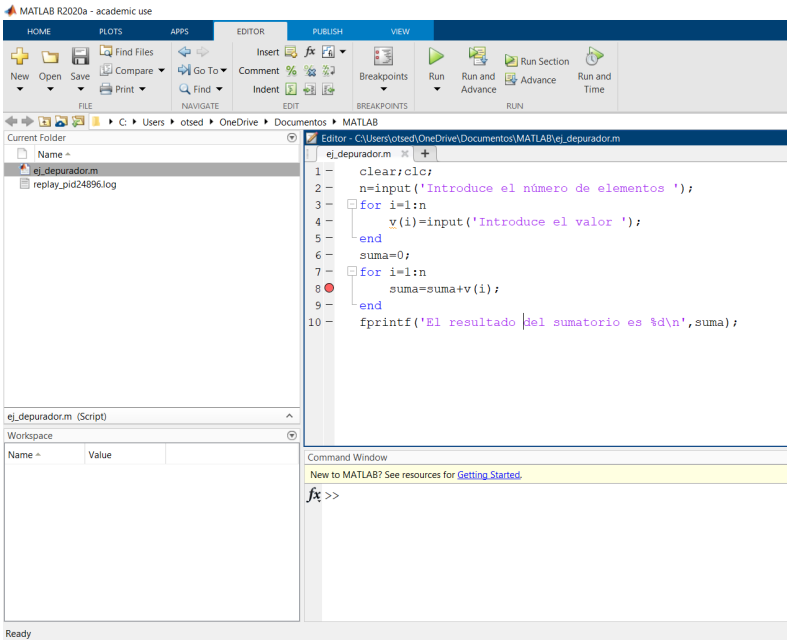


Figura 7.2: Interfaz con un punto de ruptura activado en la línea 8.

A partir de este momento, podemos ejecutar el programa. Para poder hacer uso de la posibilidades de depuración, antes es aconsejable fijar al menos un punto de ruptura o parada, para detener en esa línea en concreto la ejecución del programa. Con un punto de ruptura especificamos una línea del programa donde queremos que se detenga la ejecución. Colocar un punto de ruptura es sencillo. Colocamos el cursor en la línea del programa donde queremos colocarlo, y desde de la pestaña *Editor* seleccionamos *Breakpoints->Set/Clear* (tecla rápida F12). Observa que aparece un punto rojo en el margen izquierdo justo al lado del número de línea, ver figura 7.2. Si repetimos la acción en la misma línea el punto desaparece. Es además posible activarlo o desactivarlo con el ratón haciendo clic en la zona del margen izquierdo donde has visto que aparece el punto de ruptura.

Al lanzar ahora la ejecución, una vez que se lee el dato, se observa que en la ventana de comandos aparece $K >>$ y que en el editor aparece una flecha verde en la línea 8. La ejecución se ha detenido en esta línea, ver figura 7.3.

Para avanzar a la siguiente instrucción, en la barra de menú escogemos *Step* (tecla rápida F10), acción que ejecutará la instrucción o comando de la línea marcada por la flecha verde. En esta caso se modifica el valor de la variable **suma**. Su valor se puede consultar colocando el cursos sobre la mención a la variable en el código, o a través del espacio de trabajo. Cada vez que se pulsa *Step*, y la flecha verde avanza en el editor a la siguiente instrucción a ejecutar. Si escogemos en el menú *Continue* la ejecución prosigue hasta llegar a un punto de parada. En nuestro caso a la segunda iteración del bucle, de nuevo en la línea 8, ver figura 7.4. Colocar

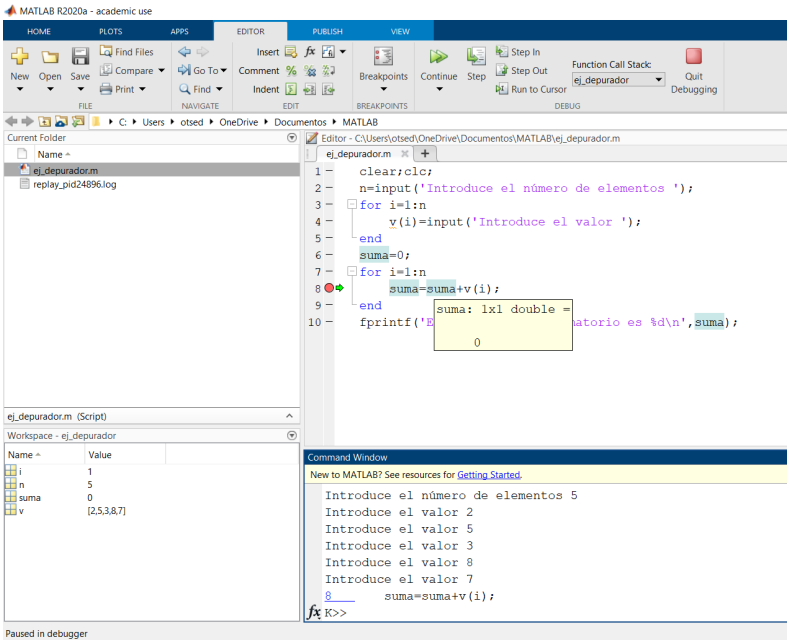


Figura 7.3: Interfaz detenida en la línea 8 en la primera iteración del bucle

el cursos sobre la variable **suma** nos muestra su valor actual.

Además de consultar el valor de variables a través del espacio de trabajo o colocando el cursos sobre sus menciones en el código, es posible consultar en la ventana de comandos el contenido de cualquier variable (tecleando por ejemplo a en nuestro caso), o realizar las operaciones que consideremos.

Si no deseamos seguir ejecutando paso a paso, siempre es posible desactivar los puntos de parada, y escoger *Continue* para que el programa prosiga su ejecución sin interrupción. Si por el contrario queremos detener la depuración, se debe escoger en el menú la opción *Quit Debugging*, situada en el extremo derecho.

Para el caso en que una instrucción sea una función o procedimiento contenido en otro fichero m, se hace uso respectivamente de los comandos *Step In* o *Step Out* para entrar en dicho fichero, y continuar la depuración en él, o salir del mismo y continuar la depuración en el fichero que lo llamó.

7.4. EJERCICIOS SEMANA II

7.4.1. De apoyo durante la sesión

1. El siguiente programa no resuelve correctamente contar el número de valores positivos y negativos contenidos en un vector. ¿Cómo lo arreglarías?

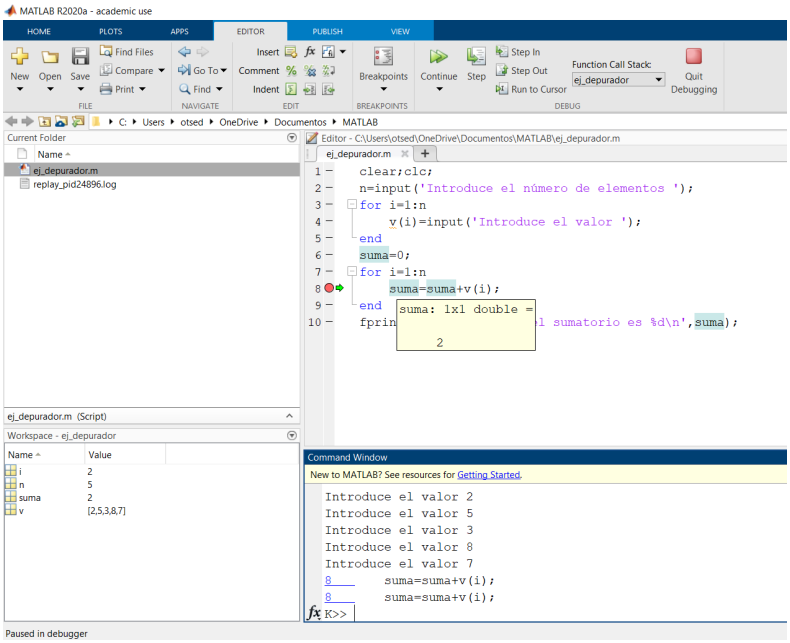


Figura 7.4: Interfaz detenida en la línea 8 en la segunda iteración del bucle

```
clear;clc;
n=input('Introduce el número de elementos ');
for i=1:n
    v(i)=input('Introduce el valor ');
end
np=0;
nn=0;
for i=1:n
    if i>0
        np=np+1;
    else
        if i<0
            nn=nn+1;
        end
    end
end
fprintf('El número de positivos es %d, y el de negativos %d\n',nn,np);
```

2. El siguiente programa no es capaz de determinar el último valor positivo contenido en un vector ¿Cómo lo arreglarías?

```
clear;clc;
n=input('Introduce el número de elementos ');
for i=1:n
    v(i)=input('Introduce el valor ');
end
for i=1:n
    if v(i)>0
```

```
        positivo=v(i);
    else
        positivo=0;
    end
end
fprintf('El último valor positivo es %d\n',positivo);
```

3. El siguiente programa no resuelve la tarea planteada de mostrar la media de los valores pares contenidos en un vector. ¿Cómo lo arreglarías?

```
clear all;
clc;

n = input('Introducir un número: ');

for i=1:n
    v(i)=input('Valor ');
end

c=1;
np=np+1;
for i=1:n
    if mod(I,2)
       c=c+I;
       np=np+1;
    end
end
fprintf '%g\n',c/v(i));
```

4. El siguiente programa no resuelve la tarea planteada de mostrar los n primeros valores pares de la serie de Fibonacci. Recuerda que en esta serie $A_i = A_{i-1} + A_{i-2}$, siendo $A_1 = A_2 = 1$.

```
clear all;
clc;
n = input('Introducir un número: ');
f(1)=1;
f(2)=1;
i=1;
j=0;
while i<=n
    f(i)=f(i-1)+f(i-2);
    fprintf('%d \n',f(i))
    i=i+1;
    j=j-1;
end
```

7.4.2. Propuestos como actividad independiente

1. El siguiente programa no resuelve la tarea planteada de mostrar los n primeros valores pares de la serie de Fibonacci. Recuerda que en esta serie $A_i = A_{i-1} + A_{i-2}$, siendo $A_1 = A_2 = 1$.

```
clear all;
clc;

fpp=1;
fp=1;
fprintf('%d %d ',fpp,fp);
for i=1:10
    f=fp+fpp;
    fprintf('%d ',f);
    fp=fpp;
    fpp=fp;
end
```

2. Emplear las herramientas de depuración para corregir *el fallo* que impide que el siguiente programa pueda resolver correctamente el cálculo de una división entera mediante restas.

```
% División por restas
% Obtenemos los datos
dividendo = input('Dividendo: ');
divisor = input('Divisor: ');

% Calculamos la división
resto = dividendo;
cociente = 0;
while (resto > divisor)
    resto = resto - divisor;
    cociente = cociente + 1;
end

% Mostramos el resultado
fprintf('%d dividido por %d es %d con un resto de %d\n', ...
    dividendo, divisor, cociente, resto);

% Comprobación de la división
c_dividendo = cociente * divisor + resto;
if (dividendo ~= c_dividendo)
    fprintf('Algo fue mal: %d ~= %d\n', dividendo, c_dividendo);
else
    fprintf('Se ha comprobado el resultado\n');
end
```

NOTA: Este código *parece* ser correcto en algunos casos, p.e. 10/3, pero no lo es.

3. El siguiente programa implementa un cronómetro (acelerado x 10) de cuenta atrás a partir de un tiempo inicial que el usuario especifica en horas minutos y segundos. Es evidente que esta versión no funciona bien, como se puede verificar fácilmente ejecutando el código. Observe que puede ralentizarse la cuenta atrás incrementando el retardo de *pause()*.

```
%Pedimos el valor inicial del contador
horas = input('Horas: ');
minutos = input('Minutos: ');
segundos = input('Segundos: ');

%Realizamos la cuenta atrás 10 segundos
seg_rest=10;
for h = horas:-1:0
    for m = minutos:-1:0
        for s = segundos:-1:0
            %Imprimimos el valor del contador
            fprintf('%02d:%02d:%02d\n',h, m, s)

            %Esperamos una décima de segundo para ver algo
            pause(0.1);

            %Borramos los ocho caracteres del contador
            %salvo si hemos llegado al final
            if (h==0 && m==0 && s==0)
                return;
            end

            %si ya hemos contado 10 segundos, salimos
            seg_rest = seg_rest-1;
            if (seg_rest==0)
                return;
            end
        end
    end
end
```

4. El siguiente programa solicita un número positivo al usuario y debe indicar si es capicúa o no.

```
clear all;
clc;

k = input('Introducir un número: '');
nc=0;
while n>=0
    nc=nc+1;
    cif(nc)=mod(n,10);
    n=floor(n/10);
end
c=0;
```

```
for i=1:nc
    if cif(i)==cif(nc-i+1)
       c=c+1;
    end
end
if c==floor(nc/2)
    fprintf('Capicúa\n');
else
    fprintf('No es capicúa\n');
end
```

7.5. TAREA

La evaluación de esta práctica, se realiza a través de un cuestionario sobre el contenido de este documento y las actividades realizadas durante su correspondiente sesión práctica. El cuestionario estará disponible para su resolución el viernes 169 de abril de 12 a 18 horas a través del campus virtual.

Práctica 8

Estructuras de datos dimensionadas: Matrices

8.1. INTRODUCCIÓN

Esta nueva sesión prosigue en la línea de conocer el manejo de datos contenidos en conjuntos dimensionados homogéneos, centrando esta práctica el uso de matrices.

8.2. CREACIÓN/INICIALIZACIÓN

Mantenemos una descripción con una estructura similar a la previa, recordando en primer término alternativas para la creación o inicialización de matrices:

```
%Definiendo una matriz, separando cada fila con ;
m1=[1 3 2; 2 3 5];

%Leyendo los valores de golpe. Necesario introducir los corchetes y separar las filas con ;
m2=input('Introduce la matriz ');

%Leyendo los valores de una matriz 2x2 uno a uno
m3(1,1)=input('Introduce un elemento de la matriz ');
m3(1,2)=input('Introduce un elemento de la matriz ');
m3(2,1)=input('Introduce un elemento de la matriz ');
m3(2,2)=input('Introduce un elemento de la matriz ');

%Leyendo los valores de una matriz 3x3 uno a uno por filas
%con un doble bucle
for fila=1:3
    for columna=1:3
        m4(fila,columna)=input('Introduce un elemento de la matriz ');
    end
end
```

```
%Cuatro filas de seis columnas de valores enteros en el rango (0,100)
m5=randi([0 100],4,6);
```

8.2.1. Acceso

De forma análoga a los ejemplos mostrados para vectores, es posible acceder a los valores contenidos en un vector para diversos fines, como puede ser mostrarlos, contar los que cumplen una condición, etc.:

```
m=[2 3 0; 4 1 6]; %Ejemplo de inicialización de una matriz 2x3

%Escribimos los datos de la matriz por filas
for i=1:2
    for j=1:3
        fprintf('El valor en la posición %d, %d de la matriz es %d\n',i,j,m(i,j));
    end
end

fprintf('\n'); %Salto de línea para separar la salida

%Escribimos los datos de la matriz por columnas
for j=1:3
    for i=1:2
        fprintf('El valor en la posición %d, %d de la matriz es %d\n',i,j,m(i,j));
    end
end

fprintf('\n'); %Salto de línea para separar la salida

%Contamos el número de valores mayores que 0
mayorquecero=0;
for j=1:3
    for i=1:2
        if m(i,j)>0
            mayorquecero = mayorquecero + 1;
        end
    end
end

fprintf('El número de valores mayores que cero es %d\n', mayorquecero);

%Almacenamos los valores impares en un vector, además de sus filas y columnas
%Calcula también su media
nimpares=0;
suma=0;
for j=1:3
    for i=1:2
        if mod(m(i,j),2)~=0
            nimpares = nimpares + 1;
            suma = suma + m(i,j);
```

```
            v(nimpares)=m(i,j);
            fil(nimpares)=i;
            col(nimpares)=j;
        end
    end
end
disp(v)
disp(fil)
disp(col)
disp(suma/nimpares)
```

8.3. EJERCICIOS

8.3.1. De apoyo durante la sesión

1. Realiza un programa que lea por teclado una matriz de valores numéricos de tamaño $n \times n$ (preguntar el valor de n al usuario previamente), y posteriormente indique el número de ellos que son múltiplos de tres.

2. Modifica el programa anterior para almacenar los valores que cumplen la condición en un nuevo vector, mostrando el número de valores múltiplos de tres, el vector y su media con un decimal.

3. Modifica el programa del ejercicio 1 para almacenar en un vector el número de valores que cumplen la condición de cada fila. El programa debe mostrar el número de valores múltiplos de tres, y el vector con el número de valores que cumplen dicha condición por fila.

4. Modifica el programa del ejercicio 1 para almacenar en otro vector la media de los valores que cumplen la condición de cada fila. El programa debe mostrar el número de valores múltiplos de tres, y el contenido del nuevo vector, mostrando cada media con un decimal.

5. Modifica el programa del ejercicio 3 para que además de mostrar el número de valores múltiplos de tres, y el vector con el número de valores que cumplen dicha condición por fila, indique la fila o filas (si en varias aparece el valor máximo) que contiene más valores que cumplen la condición.

6. Modifica el programa del ejercicio anterior para que ponga a cero la fila o filas que contiene más valores que cumplen la condición. El programa debe mostrar por tanto

el número de valores múltiplos de tres, y el vector con el número de valores que cumplen dicha condición por fila, la fila o filas que contiene más valores que cumplen la condición, y la matriz tras haber puesto a cero los valores de las mencionada fila o filas.

8.3.2. Propuestos como actividad independiente

1. Realizar un programa que lea por teclado una matriz de tamaño nxn (preguntar el valor de n al usuario previamente), y posteriormente muestre la suma de todos los elementos de la matriz.

2. Realiza un programa que lea por teclado una matriz de tamaño $n \times n$, y posteriormente muestre la media aritmética de los elementos de la matriz.

3. Escribe un programa que lea una matriz de tamaño $n \times m$ y genere otra matriz de dimensión $(n-1) \times (m-1)$ mediante la eliminación de una fila y de una columna indicadas por el usuario. Muestra por pantalla la matriz resultante.

4. Escribe un programa que lea una matriz de tamaño $n \times m$ y genere otra matriz de dimensión $(n+1) \times (m+1)$ mediante la incorporación de una fila y de una columna, inicializadas a 0, en las posiciones indicadas por el usuario. Muestra por pantalla la matriz resultante.

5. Escribe un programa que genere una matriz cuadrada $n \times n$ y posteriormente los normalice en el rango (0,1) con la expresión $\frac{(a-min)}{(max-min)}$, donde a se refiere a cada valor de la matriz.

6. Realiza un programa una vez leída una matriz cuadrada de n×n elementos, muestre por pantalla los elementos de la matriz triangular inferior derecha. Por ejemplo, para la siguiente matriz:

```
1 2 3
4 5 6
7 8 9
```

el resultado sería:

```
3
5 6
7 8 9
```

7. Realiza un algoritmo una vez leída una matriz cuadrada de $n \times n$ elementos, escriba los elementos de la matriz triangular superior. Por ejemplo, para la siguiente matriz:

```
1  2  3
4  5  6
7  8  9
```

el resultado sería:

```
1  2  3  5  6  9
```

8. Realiza un programa que calcule el producto de una matriz por un vector.

8.4. TAREA

La evaluación de esta práctica, se realiza a través de un cuestionario sobre el contenido de este documento y las actividades realizadas durante su correspondiente sesión práctica. El cuestionario estará disponible para su resolución el viernes 23 de abril de 12 a 18 horas a través del campus virtual.

Práctica 9

Estructuras de datos dimensionadas: Cadenas de caracteres

9.1. INTRODUCCIÓN

En esta práctica se prosigue con la introducción al manejo de datos contenidos en conjuntos dimensionados homogéneos, revisando en esta ocasión el uso de variables de tipo cadena.

9.2. CREACIÓN/INICIALIZACIÓN

La creación o lectura de una variable de tipo cadena de caracteres, requiere del uso de comillas simples. Sin embargo, al utilizar el comando *input*, es posible indicar de forma expresa que se espera una variable de dicho tipo, con el argumento *'s'*, ahorrando al usuario tener que introducir las mencionadas comillas. Los siguientes ejemplos, muestran diversas formas de inicialización y muestra del contenidos de variables de tipo cadenas de caracteres.

```
%Inicializando una variable cadena
cad1='Hola mundo';
%Se muestra el contenido
disp(cad1)

%Lectura por teclado de una variable de tipo cadena (requiere las comillas)
cad2=input('Introduce una cadena ');
%Muestra el contenido como cadena de caracteres
fprintf('Acabo de leer %s\n',cad2);
%Muestra el código ASCII de cada carácter de la cadena
fprintf(' %d ',cad2);
fprintf('\n');
```

```
%Lectura por teclado de una variable de tipo cadena especificando que lo es
cad3=input('Introduce una cadena ','s');
%Muestra un carácter concreto
fprintf('El segundo carácter de la cadena es %s\n',cad3(2))

%Crea una cadena con DIMENSIÓN variable y caracteres aleatorios
nletras=randi(20);
%Obtiene códigos ASCII de mayúsculas
codes=randi( [double('A') double ('Z')] , 1, nletras);
cad4=char(codes);
disp(cad4)
```

9.3. TRABAJANDO CON CADENAS DE CARACTERES

El manejo de cadenas de caracteres tiene particularidades dado que existe un repertorio de funciones especializadas que facilitan su uso. Destacar Entre ellas las siguientes:

- *length(c)* o *size(c,2)* retornan el tamaño de la cadena, de forma similar a un vector, o *size()* para una matriz.
- *ischar(c)* devuelve verdadero o 1, si **c** es una cadena de caracteres, falso o 0 en caso contrario.
- *strcmp(c1,c2)* compara dos cadenas. Si las cadenas son iguales devuelve verdadero o 1, y si no lo son, devuelve falso o 0.
- *strfind(c1,c2)* devuelve un vector con las posiciones donde aparece **c2** en *c1*. En caso de no estar presente, el vector obtenido estará vacío.
- *int2str(v)* convierte un número entero en cadena de caracteres.
- *num2str(x)* convierte un número real **x** en cadena de caracteres.
- *str2double(str)* convierte una cadena representando un número real, en el número real correspondiente.
- *double(c)* convierte el carácter c en su correspondiente código ASCII.
- *char(v)* convierte un vector de códigos ASCII v en una cadena de caracteres.

A continuación, mostramos algunos ejemplos utilizando estas funciones.

Es importante tener presente que los códigos ASCII de los caracteres están ordenados de forma similar a la ordenación alfabética.

```
c= input('Introducir texto: ', 's');

%Escribimos los caracteres de la cadena de izquierda a derecha
for i=1:length(c)
    fprintf('El carácter en la posición %d de la cadena es %s\n',i,c(i));
end

fprintf('\n');

%Escribimos los caracteres de la cadena de derecha a izquierda
for i= length(c):-1:1
    fprintf('El carácter en la posición %d de la cadena es %s\n',i,c(i));
end

%Contamos el número de espacios en blanco de la cadena
blancos=0;
for i=1: length(c)
    if c(i)==' '
        blancos=blancos+1;
        pos(blancos)=i;
    end
end
fprintf('El número de espacios en blanco en la cadena es %d\n',blancos);
%Se muestran las posiciones
disp(pos)

%Contamos el número de espacios en blanco con strfind
posblancos=strfind(c,' ');
%Se muestran las posiciones
disp(posblancos)
fprintf('El número de espacios en blanco en la cadena es %d\n',length(posblancos));

%Guardamos en otra cadena todo lo que no sea espacio en blanco
noblancos=0;
for i=1: length(c)
    if c(i)~=' '
        noblancos=noblancos+1;
        c2(nobalncos)=c(i);
    end
end
fprintf('La cadena resultante es %s\n',c2);

%Almacenamos la letra a en una variable
c='a';
%Mostramos su código ASCII
vc=double(c);
fprintf('El código ASCII de %c es %d\n',c,vc);

%Diferencias de formato al escribir un carácter con fprintf
c2='abcdefghijk';
for i=1: length(c2)
    fprintf('El carácter %c en la posición %d tiene el código %d\n',c2(i),i, c2(i));
end
```

9.4. EJERCICIOS

9.4.1. De apoyo durante la sesión

1. Realiza un programa que lea una cadena por teclado y posteriormente cuente el número de signos de interrogación "¿" o "?" que contiene, mostrando por pantalla respectivamente el número de símbolos de apertura y cierre.

2. Modifica el programa anterior para que el programa confirme que la coherencia o no en el número de signos de apertura y cierre. El programa mostrará además del número de signos de apertura y cierre, la cadena "No coincide" o "Coincide" en cada caso.

3. Modifica el programa del primer ejercicio para que, tras leer una cadena o texto, muestre por pantalla cuatro valores:

   - el número de signos de interrogación (sin distinguir entre apertura y cierre),
   - el número de signos de exclamación (sin distinguir entre apertura y cierre),
   - el número de espacios en blanco,
   - y el número de signos de puntuación (“.” “,” “;”)

4. Asumiendo que una cadena o texto únicamente contiene caracteres alfabéticos, espacios en blanco y signos de interrogación, exclamación y puntuación, realiza un programa que, tras leer una cadena, indique el número de palabras que contiene. Por simplicidad el primer carácter de toda cadena será alfabético, y el último no lo será.

9.4.2. Propuestos como actividad independiente

1. Escribe un programa que muestre el código ASCII de cada carácter presente en LA cadena.

2. Realiza un programa que lea una cadena por teclado y posteriormente muestre por pantalla el número de letras mayúsculas que contiene. Por simplicidad, suponga que la cadena no contiene la letra ñ, ni vocales con tilde.

3. Escribe un programa que lea por teclado una cadena de caracteres y un carácter a buscar en la cadena y devuelva un vector con las posiciones en las que se ha encontrado, o -1 en otro caso. Evidentemente, la longitud del vector indicará cuántas veces se ha encontrado el carácter en la cadena. La salida del programa consistirá en la impresión por pantalla de los elementos del vector resultante separados por un espacio.

4. Realiza un programa que lea un párrafo, la separe por palabras y muestre por pantalla dicha secuencia de palabras. Las palabras podrán estar separadas por espacios o por signos de puntuación. Por simplicidad, suponga que la frase a analizar no contiene la letra ñ, ni vocales con tilde.

5. Realiza un programa que lea una cadena por teclado y muestre por pantalla una cadena donde cada vocal se ha sustituido según el siguiente código: a por 1, e por 2, i por 3, o por 4, u por 5.

6. Realiza un programa que lea una cadena y muestre por pantalla los caracteres que aparecen repetidos y cuántas veces aparece cada uno.

7. Escribe un programa, que tras leer un numero real, indique el número de cifras enteras que contiene (equivale a indicar la posición del punto decimal).

8. Escribe un programa que tras leer dos cadenas, almacene en una tercera la concatenación de ambas.

9.5. TAREA

La evaluación de esta práctica, se realiza a través de un cuestionario sobre el contenido de este documento y las actividades realizadas durante su correspondiente sesión práctica. El cuestionario estará disponible para su resolución el viernes 30 de abril de 12 a 18 horas a través del campus virtual.

Práctica 10

Acceso a datos en archivo

10.1. Introducción

En esta práctica se aborda el manejo de archivos o ficheros, que facilita el trabajo con mayores volúmenes de datos. Podemos ilustrarlo con el ejemplo de un sensor situado en una localización remota de difícil acceso, que adquiere diversas medidas y las almacena de forma automática durante un largo período de tiempo. Una vez recuperados los datos, su introducción manual resultaría harto costosa, veremos la forma de automatizarlo, centrándonos en ficheros de texto.

En prácticas anteriores hemos trabajado con datos introducidos manualmente uno a uno, y almacenados en variables, vectores o matrices, llegando incluso a dibujar dichos datos comandos como la sentencia *plot*. Si necesitáramos procesar o representar un conjunto mayor de datos, la introducción a mano de los datos una y otra vez resulta un trabajo costoso y poco aprovechado. Para facilitar el manejo y reutilización de grandes cantidades de datos, se hace uso de los ficheros de datos en los que se almacena información de modo que se pueda utilizar en un programa. A continuación, resumiremos las funciones habituales de acceso a ficheros, abordando en primer término las que permiten un mayor control, de bajo nivel, y en una segunda sesión las más elaboradas y cómodas, las de alto nivel.

10.2. Manejo de ficheros a bajo nivel

Para describir las funciones básicas para el manejo de archivos a bajo nivel, en primqe término se crea en primer lugar un fichero de datos utilizando por ejemplo el bloc de notas. Utilizando el editor, se escriben las fechas de nacimiento de cinco personas, una por línea del fichero, separando con espacios día, mes y año, por ejemplo:

```
1 8 1998
23 4 1997
14 2 1998
30 1 1997
22 9 1997
```

Tras grabar el archivo a disco con el nombre *Fechas.txt*, a continuación se analiza lo necesario para leer esos datos desde Matlab. Las dos funciones básicas de bajo nivel para gestionar el acceso a un un archivo son abrir y cerrar (igual que como haces con un archivo de otro tipo en cualquier aplicación). Antes de comenzar a utilizar un archivo, es decir, antes de poder acceder a sus datos, el archivo debe abrirse, y al concluir su uso, debe cerrarse. Para abrir un archivo debemos especificar lógicamente su nombre y localización en el disco. En el momento de apertura debemos además indicar el tipo de operación que realizaremos con el archivo: leer, escribir, o anexar.

El modo *leer* permitirá únicamente leer datos del archivo, no escribir en él. De este modo se evita que se pueda modificar por error el contenido del archivo, con lo que se evitan percances. El modo *escribir* borra todo el contenido del fichero, si existe, o lo crea en caso contrario, permitiendo que se escriban datos en el archivo. Finalmente, el modo *anexar* permite añadir nuevo contenido al ya existente.

Al realizar una operación de apertura de un archivo se asigna el resultado de la acción a un identificador o variable, este identificador tendrá un valor negativo si hubiera fallado la operación, sirviendo en caso contrario como alias del fichero abierto. Para el caso concreto de apertura en modo lectura del archivo *Fechas.txt* el programa m debería contener:

```
f=fopen('Fechas.txt','r'); %Abre el archivo en modo lectura
```

A partir de este momento, si **f** es distinto de -1, significa que se dispone un canal de comunicación abierto para leer directamente datos del archivo*Fechas.txt*. Para cerrarlo tras finalizar su uso utilizaremos:

```
fclose(f);
```

Los comandos de apertura del fichero en otros modos:

```
f=fopen('Fechas.txt','w'); %Abre el archivo en modo escritura
f=fopen('Fechas.txt','a'); %Abre el archivo en modo anexar
```

Una vez abierto el canal de comunicación con un archivo secuencial, la sentencia básica para leer desde un fichero almacenando en una variable:

```
nombre_variable=fscanf(f,'%d',1);
```

Esta sentencia lee desde el archivo con alias **f** (previamente abierto utilizando *fopen*) datos del tipo entero. Para el caso concreto del archivo *Fechas.txt* recuerda que aparecen

en cada línea los valores numéricos correspondientes a cinco fechas de nacimiento, en el siguiente orden: día, mes y año separados por espacios. El programa completo resultante para este ejemplo sería:

```
clear

f=fopen('Fechas.txt','r');

%Si se pudo abrir el fichero f será mayor que -1
if f>-1
    %Leemos la fecha de nacimiento (tres datos por línea) de los 5 estudiantes
    for i=1:5
        dia(i)=fscanf(f,'%d',1);
        mes(i)=fscanf(f,'%d',1);
        anho(i)=fscanf(f,'%d',1);
    end
    %Cerramos el fichero
    fclose(f);
end
```

Tras ejecutar, tendremos en memoria las distintas fechas de nacimiento almacenadas en los vectores **dia**, **mes** y **anho**. En el caso de no conocer a priori el número de filas del archivo, debemos hacer uso de la función *feof* para comprobar si la lectura ha llegado al final del archivo, para en ese caso detenerla. Se ilustra con el archivo *Fechas.txt* empleando un bucle *while* para un número indeterminado de fechas de nacimiento.

```
clear

f=fopen('Fechas.txt','r');
%Si se pudo abrir el fichero f será distinta de -1
if f>-1
    i = 0;
    %Mientras no hayamos alcanzado el final del fichero
    while ~feof(f)
       %Intenta leer el primer valor de la línea
        tmp = fscanf(f,'%d',1);
        %Comprueba que se leyó algo, que tmp no está vacía con lo que hay una nueva línea
        if ~isempty(tmp)
            %Una fecha de nacimiento más ha sido leída
            i = i+1;
            %Almacenamos el día, que ya estaba leído en tmp
            dia(i)=tmp;
            %Leemos mes y anualidad de la fecha
            mes(i)=fscanf(f,'%d',1);
            anho(i)=fscanf(f,'%d',1);
        end
    end

    %Cierra el fichero
    fclose(f);
end
```

Una alternativa, necesaria cuando el fichero no contiene únicamente información numérica, consiste en leer la línea completa como una cadena de caracteres, y no sus valores uno a uno por separado. Para el caso del fichero ejemplo *Fechas.txt*:

```
clear

f=fopen('Fechas.txt','r');
%Si se pudo abrir el fichero f será distinta de -1
if f>-1
    i = 0;
   %Mientras no hayamos alcanzado el final del fichero
    while ~feof(f)
        %Leemos la línea completa
        tmp = fgetl(f);
        %Comprueba que se leyó algo, que tmp no está vacía con lo que hay una nueva línea
        if ~isempty(tmp)
            %Una fecha de nacimiento más ha sido leída
            i = i+1;
            %Localizamos los espacios en la cadena leída
            espacios=findstr(tmp,' ');
            %Guardamos el día, txt hasta primer espacio, convirtiendo a número
            dia(i)=str2num(tmp(1:espacios(1)-1));
            %El mes entre el primer y segundo espacio
            mes(i)=str2num(tmp(espacios(1)+1:espacios(2)-1));
        %La anualidad entre segundo espacio y el final
            anho(i)=str2num(tmp(espacios(2)+1:end));
        end
    end

    %Cerramos el fichero
    fclose(f);
end
```

Los ejemplos previos consideran únicamente ficheros con valores numéricos. En el archivo *Nombres.txt* se encuentran los nombres de cinco estudiantes:

```
Paula María Ruiz Rodríguez
Daniel García Quintana
Francisco Kevin Guerra Falcón
Mónica Mendoza Aguiar
Raul Pérez Ríos
```

Para leer la línea completa empleamos la función *fgetl* que guarda el contenido de una línea como cadena. El siguiente ejemplo, lee los nombres, distinguiendo entre el nombre y los apellidos al mostrarlo:

```
clear

f=fopen('Nombres.txt','r');
%Si se pudo abrir el fichero f será distinta de -1
if f>-1
    %Mientras no hayamos alcanzado el final del fichero
```

```
    while ~feof(f)
        %Leemos la línea completa
        NombreCompleto = fgetl(f);
        %Comprobamos que no está vacía la lectura
        if ~isempty(NombreCompleto)
            %Localiza  espacios en la cadena leída
            espacios=findstr(NombreCompleto,' ');
            %Separa nombre y apellidos, asume tres espacios para nombres compuestos
            %y sólo dos para nombres no compuestos
            if length(espacios)==3
                fprintf('Buenos días, he identificado que su nombre es %s, y sus apellidos %s\n',
                    NombreCompleto(1:espacios(2)-1) , NombreCompleto(espacios(2)+1:end))
            else
                fprintf('Buenos días, he identificado que su nombre es %s, y sus apellidos %s\n',
                    NombreCompleto(1:espacios(1)-1) , NombreCompleto(espacios(1)+1:end))
            end
        end
    end

    %Cerramos el fichero
    fclose(f);
end
```

Tras estos ejemplos observa que, si necesitas convertir una cadena a entero, debe usarse la función *str2num*.

Llegados a este punto, tendrías ya el conocimiento básico para leer un fichero secuencial de datos. Para realizar escritura a fichero, las acciones son similares, pero esta vez en modo escritura:

```
f=fopen('Nombre.txt','w'); %Abre el fichero en modo escribir
```

Empleando para cerrar el canal el mismo comando *fclose(f)*. Nos resta señalar ejemplos de escritura a fichero. Realmente es similar a la utilizada para escribir en pantalla, solo que en este caso se le indica además el identificador del fichero en el que queremos escribir un valor. Por ejemplo:

```
fprintf(f,'Hola'); %escribe la cadena Hola en el fichero identificado por f
x=3;
fprintf(f,'%d',x); %escribe el valor de x en formato entero en f fprintf(f,'%f',x); %escribe el valor de
    x en formato real en f

v=[1 3 2 5 6];
fprintf(f,'%d ',v); %escribe los valores de v en f, separándolos con
%un espacio en blanco
```

Un ejemplo muy sencillo que escribe valores a archivo:

```
f=fopen('Datosfuncion.txt','w'); %Abre el fichero en modo escribir
%Si pudo abrir el fichero sin error
if f>-1
 for x=-2:0.1:2
```

```
  %Escribe a fichero x y f(x)
  fprintf(f,'%f %f\n',x,x*x);
end
 fclose(f);
end
```

10.3. EJERCICIOS SEMANA I

10.3.1. De apoyo durante la sesión

Partiendo del archivo de datos denominado *Votacion_esp.csv*:

```
9 7 16 13 20
1 13 9 4 20
1 20 13 12 10
14 7 13 9 16
13 1 16 9 15
7 16 9 12 13
9 13 7 2 11
8 7 16 13 9
7 16 13 9 20
12 3 7 15 13
9 20 1 12 16
12 13 1 7 15
9 3 16 7 13
20 10 13 21 12
1 7 18 2 17
17 4 10 21 9
13 1 12 8 10
13 7 12 1 9
21 15 18 1 5
9 10 11 12 17
```

1. Tras crear el archivo *Votacion_esp.csv* con tu herramienta preferida, escribe un programa que lea el primer valor contenido en el archivo y lo muestre por pantalla.

2. Modifica el programa anterior para que lea todos los valores de la primera fila, almacenándolos en un vector.

3. Modifica el programa anterior para que lea todos los valores del archivo, almacenándolos en una matriz.

4. Sabiendo que los números que aparecen en cada columna del archivo están en el rango $(1,21)$, cuenta el número de apariciones de cada valor en la primera columna.

5. Cada fila del archivo *Votacion_esp.csv* se corresponde con una votación anónima, donde una persona indica su elección entre 21 propuestas. La primera columna se corresponde con su primera elección, la segunda elección, y así sucesivamente. De cara a calcular el resultado de todas las votaciones, se asume que cada propuesta recibe cinco puntos por aparecer en la primer columna, cuatro en al segunda, y así sucesivamente. Con los datos del archivo, escribe un programa que muestre por pantalla las tres propuestas con mayor puntuación.

10.3.2. Propuestos como actividad independiente

Dado el contenido del archivo *BladeRunner.txt*:

```
Yo he visto cosas que vosotros no creeríais. Atacar naves en llamas más allá de Orión. He visto rayos C
    brillar en la oscuridad cerca de la Puerta de Tannhauser. Todos esos momentos se perderán en el
    tiempo, como lágrimas en la lluvia. Es hora de morir.
```

1. Escribe un programa que cuente las letras presentes en el fichero *BladeRunner.txt*. NOTA: recordar que la función *fgetl(f)* lee una línea de texto completa del fichero.

2. Escribe un programa que cuente las letras mayúsculas presentes en el fichero *BladeRunner.txt*.

3. Escribe un programa que cuente las frases presentes en el fichero *BladeRunner.txt*. (se asume finalizadas por punto, cierre de exclamación o interrogación).

4. Escribe un programa que calcule la media de palabras por frase presente en el fichero *BladeRunner.txt*.

5. Escribe un programa que lea el fichero *BladeRunner.txt*, y convierta todo el texto a mayúsculas escribiendo el resultado en el fichero *BladeRunnermayusculas.txt*.

6. Escribe un programa que lea el fichero *BladeRunner.txt* y cuente el número de veces que se aparece una letra indicada por el usuario.

7. Crea un programa que añada al fichero *datsen*.txt los datos de la función seno en el intervalo $[\pi, 2 \times \pi]$ con un incremento de 0.1.

8. Escribe un programa que genere como resultado un fichero llamado *datos.txt* que contendrá un número n de valores equiespaciados para la función $f(x) = sin(x) \times cos(x)$ en el rango $(xmin, xmax)$ especificado por el usuario.

9. Escribir un programa que lea el fichero *textoI.txt* que contiene líneas de texto en minúsculas (no incluye ni acentos ni ñ) y convierta las letras "i" minúsculas (i) y mayúsculas (I) por el carácter uno (1), escribiendo el resultado en el fichero *textoO.txt*. NOTA: recordar que la función *fgetl()* lee una línea de texto completa del fichero.

10. Desarrollar un programa que lea un texto de un fichero y cuente el número de palabras que contiene. En cada línea, las palabras están separadas por espacios.

11. Desarrollar un programa que lea un texto de un fichero y cuente el número de veces que se repite una letra indicada por el usuario.

10.4. MANEJO DE ARCHIVOS A ALTO NIVEL

Los comandos de manejo a bajo nivel son muy flexibles, permitiendo realizar cualquier operación con ficheros, si bien hay situaciones en las que no simplifican el proceso con datos almacenados de forma tabulada. El formato de ficheros csv (del inglés *comma-separated values*) es un formato sencillo que supone un estándar de facto para la distribución de datos en archivos de texto. Repositorios de datos gubernamentales, y software de hojas de cálculo suelen facilitar la exportación a este formato entre otros.

10.4.1. Datos numéricos

Un ejemplo de achivo csv con únicamente datos numéricos es el que se muestra a continuación, al que en los siguientes ejemplos se hace referencia como *Votacion.csv*:

```
9;7;16;13;20
1;13;9;4;20
1;20;13;12;10
14;7;13;9;16
13;1;16;9;15
7;16;9;12;13
9;13;7;2;11
8;7;16;13;9
7;16;13;9;20
12;3;7;15;13
9;20;1;12;16
12;13;1;7;15
9;3;16;7;13
20;10;13;21;12
1;7;18;2;17
17;4;10;21;9
13;1;12;8;10
13;7;12;1;9
21;15;18;1;5
```

```
9;10;11;12;17
```

En el caso de contener únicamente datos numéricos con un separador claro, los comandos *dlmread* y *csvread()* facilitan la lectura de los mismos. El primero de ellos trabaja con cualquier separador, siendo generalmente capaz de detectarlo automáticamente, mientras que el segundo requiere que el separador sea la coma (','). Por dicho motivo al ser un caso especial del primero, sugerir el uso habitual de *dlmread* . El siguiente ejemplo, representa gráficamente en un diagrama de barras, tras leer los datos contenidos en el archivo con *dlmread*, los valores de la segunda columna del fichero de texto, con el resultado de la figura 10.1.

```
clear

M=dlmread('Votacion.csv');

[fil,col]=size(M); % Obtiene dimensiones de la matriz
fprintf('Matriz de datos de dimensión %dx%d\n',fil,col);

bar(M(:,2))
```

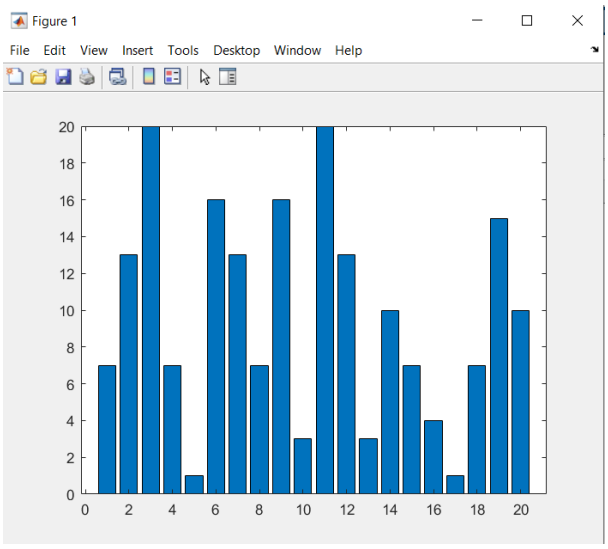


Figura 10.1: Gráfica resultante tras cargar el archivo *Votacion.csv*

Puede darse el caso de que no interesen todos los datos presentes en el archivo. Esta situación puede darse por ejemplo si la primera línea del fichero fuera una cabecera conteniendo etiquetas (no numéricas) que describen los datos. Si el archivo fuera:

```
P1;P2;P3;P4;P5
9;7;16;13;20
1;13;9;4;20
1;20;13;12;10
14;7;13;9;16
13;1;16;9;15
```

```
7;16;9;12;13
9;13;7;2;11
8;7;16;13;9
7;16;13;9;20
12;3;7;15;13
9;20;1;12;16
12;13;1;7;15
9;3;16;7;13
20;10;13;21;12
1;7;18;2;17
17;4;10;21;9
13;1;12;8;10
13;7;12;1;9
21;15;18;1;5
9;10;11;12;17
```

Para estos casos es posible indicarle al comando *dlmread* el desplazamiento tanto de filas como columnas, requiriendo también que sea indicado en carácter separador, en este caso (';'). El resultado sería idéntico al del archivo anterior tecleando:

```
clear

M=dlmread('Votacion_header.csv',';',1,0);

[fil,col]=size(M);
fprintf('Matriz de datos de dimensión %dx%d\n',fil,col);

bar(M(:,2))
```

donde se ha indicado que el separador entre datos es el carácter (';'), y que se debe aplicar un desplazamiento de una fila, y de ninguna columna.

Si por el contrario nuestra objetivo es escribir datos contenidos en una variable a un archivo csv, en la documentación actual de Matlab se recomienda el comando *writematrix* en lugar de *dlmwrite*, por ejemplo:

```
writematrix(M(5:end,2:3), 'Votacion_sub.csv');
```

que almacena en el archivo *Votacion_sub.csv* los datos de las columnas segunda y tercera a partir de la quinta fila.

10.4.2. Datos no únicamente numéricos

Los comandos mencionados anteriormente no resultan prácticos en situaciones en las que el archivo de datos contiene datos no numéricos, fuera de la cabecera. Las versiones actuales de Matlab ofrecen comandos cono *readtable*, *xlsread* o *importdata* que podrían requerir posterior tratamiento de cadenas para aquellas columnas concretas con datos no numéricos. Una alternativa es hacer uso de las funciones de bajo nivel, descritas en la primera

sección del capítulo. A continuación mostraremos casos reales de uso de los dos primeros comandos, con archivos obtenidos de repositorios institucionales.

Para ilustrar el uso del comando *readtable* se hace uso de datos obtenidos a partir de la Iniciativa Aporta del gobierno de España[1]. Entre todo el catálogo de datos disponibles, el ejemplo que se utiliza a continuación, utiliza los datos *Evolución de enfermedad por el coronavirus (COVID-19)*[2]. Tras descargar el archivo correspondiente a la *Serie histórica de casos por Comunidad Autónoma*, se tiene acceso al archivo *casos_diagnostico_ccaa.csv* desde nuestro equipo. Con el siguiente código se cargan los datos, mostrando el tamaño y el contenido de las cinco primeras filas. Debes tener presente que el archivo csv debe estar en la misma carpeta que el programa .m, o en su defecto indicar la ruta de acceso en la llamada al comando *readtable*.

```
ccaa = readtable('casos_diagnostico_ccaa.csv');
%Dimensiones
size(ccaa)
%Muestra contenido primeras filas
disp(ccaa(1:5,:))
```

Como se aprecia en la figura 10.2 cada columna tiene un nombre asignado, y unos datos, entre los que se observa una primera columna con texto, una segunda con un formato de fecha, y el resto con valores numéricos.

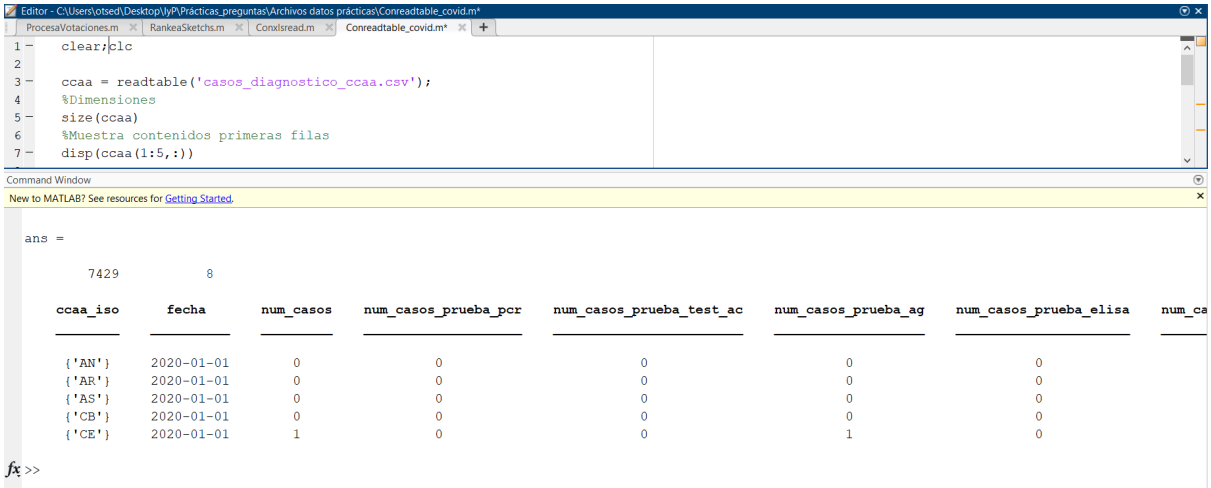


Figura 10.2: Mostrando el tamaño y primeras filas de *casos_diagnostico_ccaa.csv*

Una posibilidad es extraer todos los datos a partir de la tercera columna, excluyendo la primera fila, y salvarlos en el archivo *cca_covid_num.csv*:

[1] https://datos.gob.es/es
[2] https://datos.gob.es/es/catalogo/e05070101-evolucion-de-enfermedad-por-el-coronavirus-covid-19

```
ccaa = readtable('casos_diagnostico_ccaa.csv');
%Extrae sin primera línea, a partir de tercera columna
ccaa_num=ccaa{2:end,[3:end]};
writematrix(ccaa_num, 'cca_covid_num.csv');
```

Sin embargo, son datos en los que se mezclan los de todas las comunidades autónomas. De cara a limitarlos a una sola, dado que la primera columna contiene las siglas de cada comunidad autónoma, y conociendo que *CN* es el identificador de Canarias, el siguiente código primero determina las filas que se corresponden a Canarias, extrayendo posteriormente la subtabla correspondiente, y mostrando con un *plot* el contenido de la tercera columna, que se corresponde al número de casos.

```
ccaa = readtable('casos_diagnostico_ccaa.csv');

%Filas de una determinada comunidad
filas_CN=strcmp(ccaa.ccaa_iso, 'CN');
%Extrae una tabla con datos de determinadas filas
can_dat=ccaa(filas_CN, :);
%Gráfica con los datos de la tercera columna
plot(can_dat{2:end,3})
```

Para que aparezca en el eje x la fecha basta con darle como entrada al comando *plot* el contenido de la segunda columna de la tabla, la que se corresponde con las fechas:

```
ccaa = readtable('casos_diagnostico_ccaa.csv');

%Filas de una determinada comunidad
filas_CN=strcmp(ccaa.ccaa_iso, 'CN');
%Extrae una tabla con datos de determinadas filas
can_dat=ccaa(filas_CN, :);
%Gráfica con los datos de la tercera columna
plot(can_dat{2:end,2}, can_dat{2:end,3})
```

Por último, el siguiente código selecciona las filas correspondientes a un rango de fechas concreto, haciendo uso de los comandos *datetime* e *isbetween*, para finalmente obtener la gráfica mostrando el contenido de la mencionada tercera columna en dicho período temporal, con el resultado mostrado en la figura 10.3:

```
ccaa = readtable('casos_diagnostico_ccaa.csv');

%Filas de una determinada comunidad
filas_CN=strcmp(ccaa.ccaa_iso, 'CN');
%Extrae una tabla con datos de determinadas filas
can_dat=ccaa(filas_CN, :);

%Datos en un rango de fechas
%Definimos el rango
inicio=datetime(2020,1,1);
fin=datetime(2020,8,31);
```

```
%Determinamos las filas (mirando segunda columna) en dicho rango
filas_fechas=isbetween(can_dat.fecha,inicio,fin);
%Gráfica con los datos del rango de fechas escogido
plot(can_dat{filas_fechas,2}, can_dat{filas_fechas,3})
```

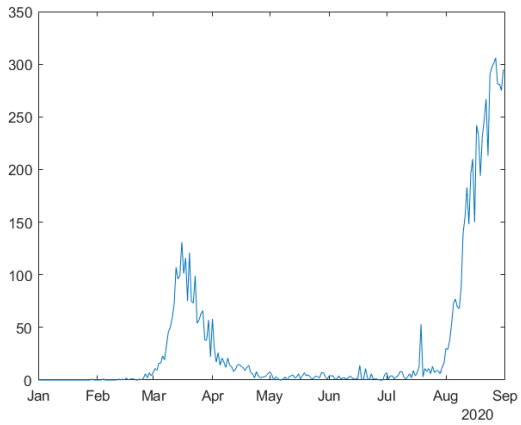


Figura 10.3: Número de casos en Canarias en un rango de fechas

Para ilustrar el comando *xlsread*, en las siguientes líneas se hace uso de datos disponibles a través de la Red de Control y Vigilancia de la Calidad del Aire de Canarias[3]. Una vez en la página, en el ejemplo que se muestra a continuación, por medio de la opción de menú *Datos->Descargas* se descargan los datos correspondientes a 2019. La descarga se corresponde con un archivo comprimido, que debe descomprimirse para poder trabajar con él. Tras la descompresión, uno de los archivos está en formato Excel (*.xlsx*). La apertura del archivo con la citada aplicación, permite ver la existencia de un número hojas, correspondientes a cada estación de la red. Para acceder a los datos presentes en el citado archivo *Datos 2019.xlsl*, la llamada:

```
[num,txt,raw]=xlsread('Datos 2019.xlsx');
```

obtiene únicamente la primera hoja del libro Excel. Es posible especificar la hoja concreta que se quiere leer especificando su nombre, recomendando obtener los nombres de las hojas presentes en el libro Excel con el comando *sheetnames*:

```
hojas = sheetnames('Datos 2019.xlsx')
```

que para este caso concreto, proporciona como salida:

```
hojas =
  60x1 string array
    "Casa Cuna"
```

[3]https://www3.gobiernodecanarias.org/medioambiente/calidaddelaire/inicio.do

```
"Vuelta Los Pájaros-Sta Cruz TF"
"Depósito de Tristán-Sta Cruz TF"
"García Escámez-Sta Cruz TF"
"Parque La Granja-Sta Cruz TF"
"San Nicolás"
"Mercado Central"
"Nestor Alamo"
"Tome Cano"
"Parque San Juan-Telde"
"La Hidalgo-Arafo"
"Balsa de Zamora-Los Realejos"
"Ciudad Deportiva-Arrecife"
"Agencia Tributaria"
"Bomberos"
"Comandancia Naval"
"Comisaria"
"Tena Artigas-Sta Cruz de TF"
"Polideportivo Afonso-Arucas"
"Ed Polivalente"
"Casa Palacio-Pto del Rosario"
"San Antonio-Brña Baja"
"Piscina Municipal-Sta Cruz TF"
"Tefía-Pto del Rosario"
"Las Balsas-S.Andrés y Sauces"
"Echedo-Valverde"
"Tio Pino-Sta Cruz de TF"
"Residencia Escolar-La Gomera"
"San Lázaro"
"Palmetum-Sta Cruz de TF"
"Agüimes"
"Arinaga"
"Castillo del Romeral"
"Playa del Ingles"
"San Agustín"
"Camping Temisas-Sta Lucía de T."
"Barranco Hondo"
"Caletillas"
"Igueste"
"Depósito La Guancha-Candelaria"
"Buzanada"
"El Río "
"Galletas"
"Granadilla"
"Médano"
"San Isidro"
"Tajao"
"Jinamar 3"
"Pedro Lezcano"
"La Loma-Telde"
"Centro de Arte"
"Parque de la Piedra"
"El Charco-Pto del Rosario"
"Arrecife"
"Costa Teguise"
```

```
    "Las Caletas-Teguise"
    "La Grama-Breña Alta"
    "El Pilar-Sta Cruz de La Palma"
    "Centro de Visitantes-SS Gomera"
    "Las Galanas-SS Gomera"
```

Una vez conocido el repertorio de estaciones disponible, en el siguiente código, se cargan los datos recogidos en la hoja correspondiente a una única estación, la estación denominada *Mercado Central* :

```
%Obtiene de una hoja concreta
[num, txt ,raw]=xlsread('Datos 2019.xlsx','Mercado Central');
size(num)
size(txt)
size(raw)
```

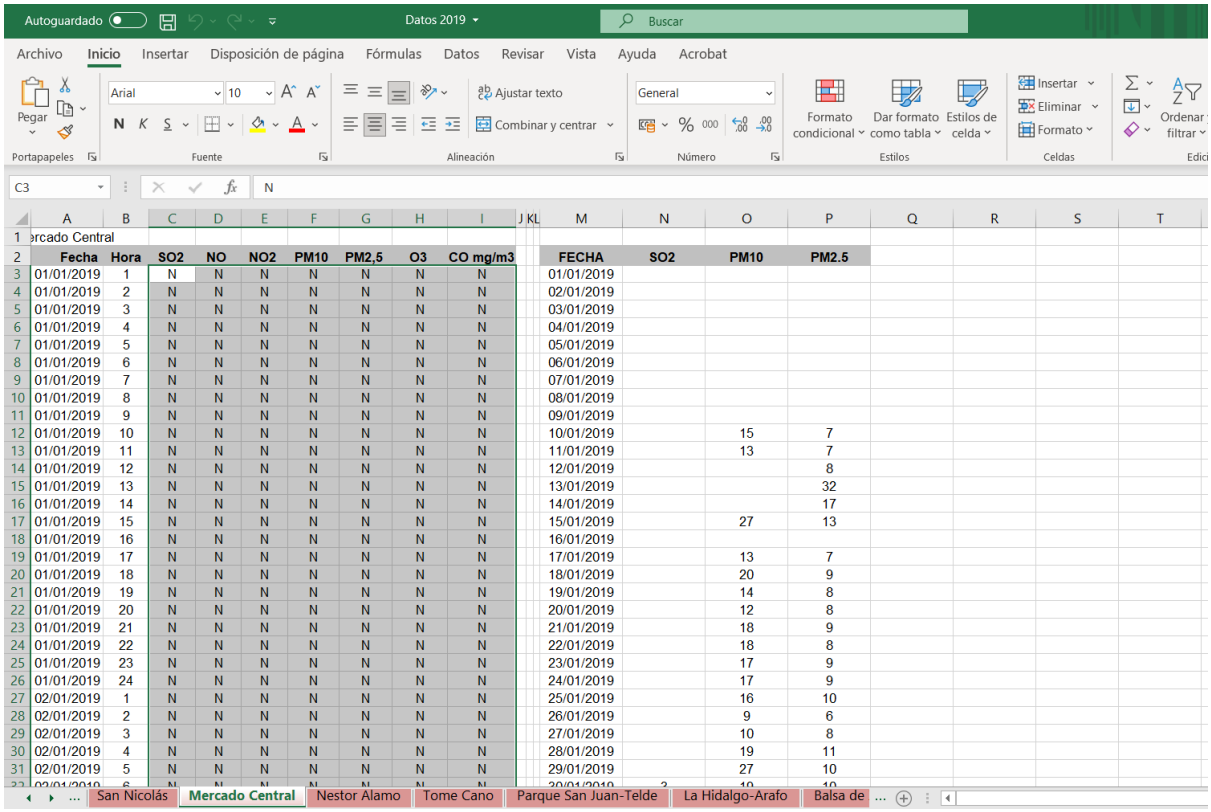
La llamada al comando *xlsread* en este caso devuelve tres variables, todas ellas matrices: **num**, **txt** y **raw**. Estas variables se corresponden respectivamente con los datos numéricos presentes en la hoja, los de tipo texto o cadenas de caracteres, y los datos completos en crudo (*raw* en inglés). Mostrar las dimensiones de las respectivas matrices devuelve:

```
ans =
        8760          15

ans =
        8762          16

ans =
        8762          16
```

Se observa que la matriz de datos numéricas contiene dos filas y una columna menos. Se corresponden a las dos primeras filas (sin números) y primera columna (sin números). Ver la hoja abierta en Excel en la figura 10.4.

La primera columna se corresponde con la fecha, y la segunda con la hora de adquisición del dato. Sin embargo, ninguna de las dos está en formato *datetime*, sino que la fecha es una cadena de caracteres, y la hora un valor numérico. Para poder acceder a rangos de fechas, como en el ejemplo previo con *readtable*, interesa convertir al mencionado formato. El siguiente código realiza la conversión, en primer término pasando el texto (desde la tercera fila, para excluir las dos primeras de cabecera) de la fecha al formato adecuado con el comando *datetime*, y posteriormente añadiendo la información de la hora de adquisición con el comando *hours*:

```
%Obtiene de una hoja concreta
[num, txt ,raw]=xlsread('Datos 2019.xlsx','Mercado Central');

%Convierte a formato de fecha, excluyendo las dos primeras filas
```



Fecha	Hora	SO2	NO	NO2	PM10	PM2,5	O3	CO mg/m3
01/01/2019	1	N	N	N	N	N	N	N
01/01/2019	2	N	N	N	N	N	N	N
01/01/2019	3	N	N	N	N	N	N	N
01/01/2019	4	N	N	N	N	N	N	N
01/01/2019	5	N	N	N	N	N	N	N
01/01/2019	6	N	N	N	N	N	N	N
01/01/2019	7	N	N	N	N	N	N	N
01/01/2019	8	N	N	N	N	N	N	N
01/01/2019	9	N	N	N	N	N	N	N
01/01/2019	10	N	N	N	N	N	N	N
01/01/2019	11	N	N	N	N	N	N	N
01/01/2019	12	N	N	N	N	N	N	N
01/01/2019	13	N	N	N	N	N	N	N
01/01/2019	14	N	N	N	N	N	N	N
01/01/2019	15	N	N	N	N	N	N	N
01/01/2019	16	N	N	N	N	N	N	N
01/01/2019	17	N	N	N	N	N	N	N
01/01/2019	18	N	N	N	N	N	N	N
01/01/2019	19	N	N	N	N	N	N	N
01/01/2019	20	N	N	N	N	N	N	N
01/01/2019	21	N	N	N	N	N	N	N
01/01/2019	22	N	N	N	N	N	N	N
01/01/2019	23	N	N	N	N	N	N	N
01/01/2019	24	N	N	N	N	N	N	N
02/01/2019	1	N	N	N	N	N	N	N
02/01/2019	2	N	N	N	N	N	N	N
02/01/2019	3	N	N	N	N	N	N	N
02/01/2019	4	N	N	N	N	N	N	N
02/01/2019	5	N	N	N	N	N	N	N

FECHA	SO2	PM10	PM2.5
01/01/2019			
02/01/2019			
03/01/2019			
04/01/2019			
05/01/2019			
06/01/2019			
07/01/2019			
08/01/2019			
09/01/2019			
10/01/2019		15	7
11/01/2019		13	7
12/01/2019			8
13/01/2019			32
14/01/2019			17
15/01/2019		27	13
16/01/2019			
17/01/2019		13	7
18/01/2019		20	9
19/01/2019		14	8
20/01/2019		12	8
21/01/2019		18	9
22/01/2019		18	8
23/01/2019		17	9
24/01/2019		17	9
25/01/2019		16	10
26/01/2019		9	6
27/01/2019		10	8
28/01/2019		19	11
29/01/2019		27	10

Figura 10.4: Hoja correspondiente al Mercado Central.

```
fechas=datetime(raw(3:end,1),'InputFormat','dd/MM/yyyy');
%Adjunta las horas (primera columna de num) a la fecha,
fechas=fechas(:)+hours(num(:,1));
disp(fechas(1:5))
```

La última línea muestra el contenido de las cinco primeras filas de la variable **fechas**, que justamente son las fechas y horas de las primeras adquisiciones de datos de 2019:

```
01-Jan-2019 01:00:00
01-Jan-2019 02:00:00
01-Jan-2019 03:00:00
01-Jan-2019 04:00:00
01-Jan-2019 05:00:00
```

Llegados a este punto, la variable **fechas** contiene las fechas y horarios de las tomas de datos, la variable **num**, las correspondientes medidas (si bien recordar que la primera columna es la hora). La medida concreta disponible en cada columna la podemos obtener por la segunda fila del archivo xlsl, que a través del programa m sería:

```
raw(2,:)
```

En el siguiente ejemplo, se realiza todo el proceso para centrarse en las medidas de la novena

columna, la que contiene $CO\ mg/m^3$. Teniendo presente que **num** tiene una columna menos, se deduce que los datos de la novena columna de la hoja, son los almacenados en la octava de **num**. De forma análoga al ejemplo mostrado con *readtable*, se define un rango de fechas que se almacena en las variables **inicio** y **fin** permitiendo establecer las filas de interés en dicho rango, que se almacenan en **filas_fechas**, extrayendo por comodidad las fechas concretas y medidas a las variables **fechas_aux** y **medidas_aux** para en primer término visualizar los datos, ver figura 10.5, y posteriormente obtener el valor máximo y el momento en que dicho valor fue obtenido.

```
clc; clear

%Obtiene de una hoja concreta
[num, txt ,raw]=xlsread('Datos 2019.xlsx','Mercado Central');

%Convierte a formato de fecha, excluyendo las dos primeras filas
fechas=datetime(raw(3:end,1),'InputFormat','dd/MM/yyyy');
%Adjunta las horas (primera columna de num) a la fecha,
fechas=fechas(:)+hours(num(:,1));

%Rango de fechas
inicio=datetime(2019,1,1);
fin=datetime(2019,1,31);
filas_fechas=isbetween(fechas,inicio,fin);

fechas_aux=fechas(filas_fechas);
medidas_aux=num(filas_fechas,8);
%Gráfica con los datos del rango de fechas escogido
plot(fechas_aux, medidas_aux)
title(char(raw(2,9))); %Cabecera novena columna de raw o txt

%Obtiene el máximo y su momento de captura
[maxval,idxmaxval]=max(medidas_aux);
fprintf('Valor máximo de %s medido fue %.2f tomado %s\n', char(raw(2,9)), maxval, char(fechas_aux(
    idxmaxval)))
```

Llegados a este punto, cuentas con las herramientas necesarias para calcular máximos, mínimos, medias, y cualquier estadística para el rango de fechas que determines.

10.4.3. Repositorios de datos

Se incluye un pequeño listado con algunos repositorios de datos de nuestro entorno:

- Iniciativa de datos abiertos del Gobierno de España[4]
- Canarias datos abiertos[5]

[4] https://datos.gob.es
[5] https://datos.canarias.es/portal/

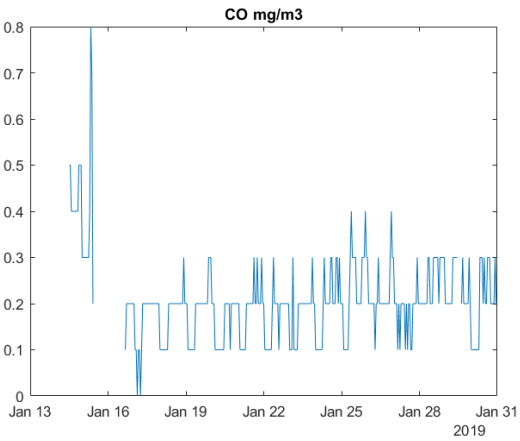


Figura 10.5: Datos del mes de enero.

- Catálogo de datos del ayuntamiento de Las Palmas de Gran Canaria[6]
- AEMET[7]

10.5. EJERCICIOS SEMANA II

10.5.1. De apoyo durante la sesión

1. Para la tarea se utilizarán datos obtenidos a través de su web[8] en la sección del menú *Datos->Descargar*, en concreto los datos de 2019. Escribe un programa que determine el máximo valor de *O3* para la estación denominada *Arinaga*.

2. Modifica el programa anterior para que además muestre la fecha y horario en que ocurre dicho máximo.

3. Escribe un programa que determine los valores máximo y mínimo por trimestre de *PM10* de la estación de *Arinaga*, indicando las respectivas fechas y horarios.

4. Escribe un programa que compare los datos denominados *PM10* de las estaciones de *Arinaga* y *San Agustín*, e indique. considerando los máximos por mes, en cuántas ocasiones el máximo valor de la primera estación supera al de la segunda.

[6] http://datosabiertos.laspalmasgc.es
[7] http://opendata.aemet.es
[8] http://www.gobiernodecanarias.org/medioambiente/calidaddelaire/inicio.do

10.5.2. Propuestos como actividad independiente

1. Descargar desde el repositorio conteniendo la *Evolución de enfermedad por el coronavirus (COVID-19)*[9], el archivo correspondiente a la *Serie histórica de casos por provincia*. Visualizar el número de casos diarios durante 2020 para la provincia canaria de tu elección.

2. Modifica el programa anterior para que visualice el acumulado por mes.

3. Modifica el programa anterior para que muestre los acumulados por mes de ambas provincias canarias. Determina el número de meses en que cada provincia supera a la otra,

10.6. TAREA

La evaluación de esta práctica, se realiza a través de un cuestionario sobre el contenido de este documento y las actividades realizadas durante su correspondiente sesión práctica. El cuestionario estará disponible para su resolución el viernes 14 de mayo de 12 a 18 horas a través del campus virtual.

[9] https://datos.gob.es/es/catalogo/e05070101-evolucion-de-enfermedad-por-el-coronavirus-covid-19

Práctica 11

Programación estructurada

11.1. INTRODUCCIÓN

En esta práctica se aborda el concepto de función como base de la programación estructurada o modular. Matlab permite la creación de funciones (subprogramas) propias facilitando la escritura de código más modular, evitando la utilización de un único programa que resta legibilidad. La utilización de funciones tiene dos ventajas:

- **Claridad**: Los programas escritos utilizando funciones son más claros de entender y de realizar. A continuación, se muestra el ejemplo del cálculo de las combinaciones de **n** elementos tomados de **m** en **m** sin y con funciones:

Listado 11.1: Solución sin utilizar funciones.

```
%Programa para calcular las combinaciones de n elementos tomados de m en m
n = input('Introducir n: ');
m = input('Introducir m: ');

%cálculo del factorial de n
factn = 1;
for i = 1:n
    factn = factn*i;
end

%cálculo del factorial de m
factm = 1;
for i = 1:m
    factm = factm*i;
end

%cálculo del factorial de n-m
factnm = 1;
for i = 1:n-m
    factnm = factnm*i;
```

```
end

% calcula las combinaciones
comb = factn/(factm*factnm);

% muestra el resultado
fprintf('Las combinaciones de %d elementos de %d en %d son: %d\n',n,m,m,comb);
```

Listado 11.2: Solución utilizando una función.

```
% Combinaciones de n elementos tomados de m en m utilizando una función
n = input('Introducir n: ');
m = input('Introducir m: ');

% calcula las combinaciones
comb = factorial(n)/(factorial(m)*factorial(n-m));

% muestra el resultado
fprintf('Las combinaciones de %d elementos de %d en %d son: %d\n',n,m,m,comb);
```

Listado 11.3: Código de la función.

```
function [f] = factorial(n)
% función para calcular el factorial del número n
facto = 1;
for i = 1:n
    facto = facto*i;
end
f = facto;
end
```

En la segunda solución se hace uso de dos ficheros .m, uno con la descripción de la función *factorial* que denominado *factorial.m* (debe coincidir nombre del fichero y de la función), y otro el fichero .m principal que lee los datos, calcula las combinaciones, y muestra el resultado.

- **Reutilización**: La resolución de multitud de problemas requiere resolver subproblemas que son comunes entre ellos. Haciendo uso de funciones para resolver estos subproblemas comunes, estas funciones se pueden utilizar en diversos problemas sin necesidad de reprogramarlos. Un nuevo ejemplo, para calcular las permutaciones sin repetición de **k** elementos tomados de un conjunto de **n** elementos, una posible solución en Matlab reutilizando la función *factorial* descrita anteriormente:

```
% Programa para calcular las permutaciones sin repetición de k elementos
% tomados de un conjunto de n usando una función=> n!(n-k)

n = input('Introducir n: ');
k = input('Introducir k: ');
```

```
% calcula las permutaciones
permu = factorial(n)/factorial(n-k);

% muestra el resultado
fprintf('Las permutaciones de %d elementos tomados de un conjunto de %d son: %d\n',k,n,permu);
```

11.2. DEFINICIÓN DE FUNCIONES

En lenguaje m las funciones son muy similares a un programa normal con la diferencia de que la función recibe unos argumentos de entrada o parámetros, y debe devolver uno o varios valores o argumentos de salida. En cuanto a la sintaxis, una función tiene ciertas diferencias con los programas que hemos realizado anteriormente, en concreto la primera línea del fichero m, que debe tener el mismo nombre que la función, especifica el nombre de la función, los argumentos de entrada, y los argumentos de salida, es decir, los valores que devuelve. A continuación, se muestra la sintaxis estándar de definición de una función, que podrás comparar con de la función *factorial* utilizada anteriormente.

```
function [ArgOut1 ArgOut2 ... ] = NombreFuncion(ArgIn1, ArgIn2, ...)

ArgOut1 = ...;
ArgOut2 = ...;
...
end
```

La primera línea es ahora una cabecera, y la última *end* (opcional). La cabecera se identifica por la palabra *function*. A continuación, y entre corchetes (obligatorio si son más d euno), se listan todos los argumentos de salida de la función. Tras el signo igual, aparece el nombre de la función (cualquiera pero teniendo en cuenta que no coincida con el nombre de alguna función ya existente). Por ejemplo, no debería asignar a una función el nombre *plot*, para evitar confusiones. Después del nombre y entre paréntesis se enumeran los argumentos que acepta como entrada la función y de cuyos valores dependerá el valor de los argumentos de salida.

A continuación, se muestra el ejemplo de una función que devuelve más de un valor a diferencia de la función *factorial* que solo devolvía uno. Así en el editor m tras elegir la creación del fichero m, tecleamos:

```
function [s,r]=sumaresta(x,y)
s=x+y;
r=x-y;
end
```

Observa que esta función devuelve más de un valor. Las variables de salida deben ser utilizadas dentro de la función para guardar en ellas los valores que nos interese devolver o retornar desde la función. De esta forma en **s** guardamos el resultado de la suma de los dos argumentos de entrada, y en **r** la sustracción de ambos. Guardamos el fichero m con el nombre *sumaresta.m* y para utilizarla escribe en la ventana de comandos:

```
sumaresta (2,4)
```

¿A qué se corresponde el valor devuelto? Para que el resultado se almacene directamente en una variable escribimos:

```
a = sumaresta(2,4);
```

La variable **a** almacena el resultado de la operación. Estamos almacenando el primero de los dos valores que devuelve la función. Para guardar ambos, teclea en la ventana de comandos:

```
[a,b]= sumaresta(2,4);
```

La llamada a una función puede combinarse con esquemas repetitivos para obtener resultados para distintos valores, como en el siguiente ejemplo:

```
for i=1:1000
    for j=1:2:500
        [a,b]= sumaresta (i,j)
    end
end
```

11.3. VISUALIZANDO UNA MATRIZ

De cara a los ejercicios propuestos, se recupera el ejemplo del listado 2.1 que brevemente ilustraba que los valores almacenados en una matriz pueden interpretarse como píxeles de una imagen. Partiendo de una matriz con valores en el rango $(0,1)$, el valor de cada elemento de la matriz se interpreta como el nivel de gris un píxel de la imagen. El valor mínimo, el 0, se corresponde al color negro, mientras que el mayor, el 1, al color blanco. Los valores intermedios se corresponden con diferentes niveles de gris. Para imágenes en color, como la del mencionado listado, cada píxel requiere tener tres valores asociados, que se correspondían con las componentes o planos RGB del píxel. Cada valor está comprendido entre 0 y 1, indicando un 0 ausencia de dicho color, y un 1 la cantidad máxima del mismo.

El comando *imshow* permite visualizar una matriz asumiendo que los valores que contiene se corresponden a píxeles. Si se inicializa a cero, con la función *zeros* obtenemos la imagen negra, ver figura 11.1:

```
a=zeros(500,500);
imshow(a)
```

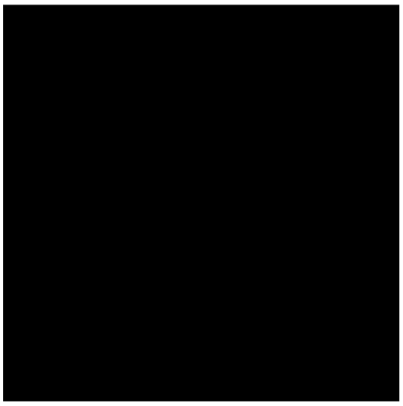


Figura 11.1: Visualizando una matriz inicializada a 0.

Con la función *rand*, se inicializa la matriz con valores entre 0 y 1, Al ser valores aleatorios, la imagen resultante es contendrá píxeles con diferentes niveles de gris entre el negro y el blanco, ver figura 11.2:

```
a=rand(500,500);
imshow(a)
```

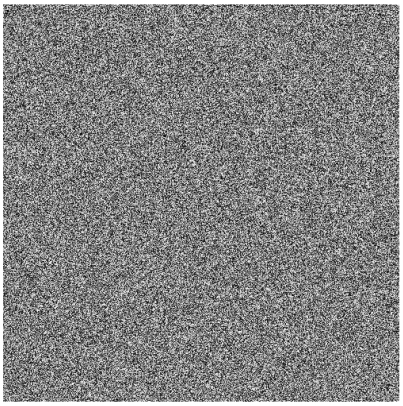


Figura 11.2: Visualizando una matriz inicializada de froma aleaoria.

El ooperador ':' permite asignar tonos a zonas concretas de la matriz, es decir de la imagen. El siguiente código crea unas franjas verticales grises, ver figura 11.3:

```
a(1:500,1:100)=0;
a(1:500,101:200)=0.25;
a(1:500,201:300)=0.5;
a(1:500,301:400)=0.75;
a(1:500,401:500)=1;
imshow(a)
```

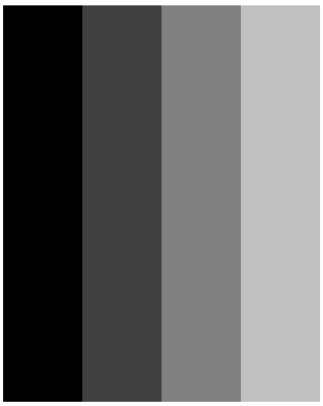


Figura 11.3: Barras verticales grises.

Para trabajar con matrices como imágenes en color es necesario considerar los tres planos RGB. Al definir la matriz, la tercera dimensión contiene tres posiciones correspondiendo respectivamente a los planos de color rojo, verde y azul. Para crear una imagen con colores aleatorios, ver figura 11.4:

```
a=rand(500,500,3);
imshow(a)
```

Para definir una imagen completamente roja, basta fijar el primer canal a 1 y el resto a 0, ver figura 11.5:

```
a=zeros(500,500,3);
a(:,:,1)=1;
imshow(a)
```

Para crear unas barras rojas similares al ejemplo previo con grises, ver figura 11.6:

```
a=zeros(500,500,3);
a(:,1:100,1)=0;
a(:,100:200,1)=0.25;
a(:,200:300,1)=0.5;
a(:,300:400,1)=0.75;
a(:,400:500,1)=1;
imshow(a)
```

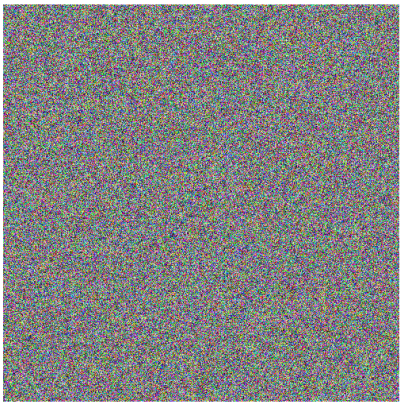


Figura 11.4: Imagen RGB aleatoria.

Figura 11.5: Imagen roja.

Rellenar una zona concreta con tonos verdes, ver figura 11.7:

```
a=zeros(500,500,3);
a(200:300,100:175,2)=1;
imshow(a)
```

11.4. EJERCICIOS

11.4.1. De apoyo durante la sesión

1. Escribe una función denominada *inimatrizg.m* que dados como argumentos de entrada la dimensión y un valor **v** en el rango $(0,1)$, devuelva como argumento de salida una

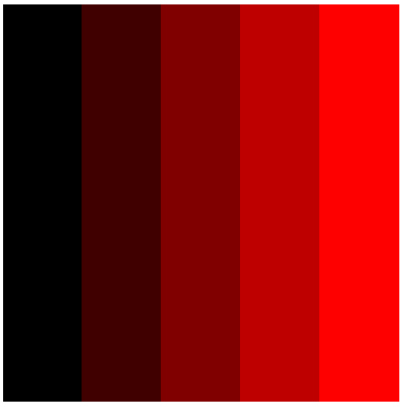


Figura 11.6: Tonos de rojo.

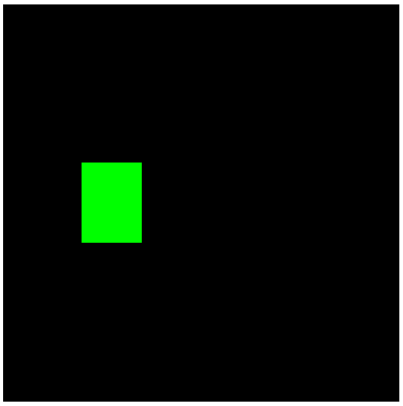


Figura 11.7: Recuadro verde sobre negro.

matriz $n \times n$ con todas sus posiciones inicializadas al valor de **v**. Llama a dicha función desde un programa principal que previamente ha leído los valores de **n** y **v**. El programa principal debe para finalizar mostrar el contenido de la matriz resultante. Puedes visualizarla con *imshow*.

2. Escribe una función denominada *inimatrizrgb.m* que, dados como argumentos de entrada la dimensión, y los valores **r**, **g** y **b** en el rango (0,1), devuelva como argumento de salida una matriz $n \times n$ con tres planos de profundidad, con todas sus posiciones inicializadas a dichos valores. Llama a dicha función desde un programa principal en el que se leen **n**, **r**, **g** y **b**. El programa principal debe para finalizar mostrar el contenido de la matriz resultante.

3. Escribe una función denominada *lineahrgb.m* que tomando como argumentos de entrada una matriz, un valor de fila, y los valores **r**, **g** y **b**, devuelva como argumento de salida una matriz en la que todos los valores de la fila indicada tengan asignados dichos valores **r**, **g** y **b**. Modifica el programa principal del ejercicio 2 para que, tras crear la matriz inicial con *inimatrizrgb.m*, llame a *lineahrgb.m*. Para finalizar mostrar el contenido de la matriz resultante.

4. Escribe una función denominada *rectrgb.m* que tomando como argumentos de entrada una matriz, una fila inicial y otra final, una columna inicial y otra final, y los valores **r**, **g** y **b**, devuelva como argumento de salida una matriz en la que todas las posiciones en dichos rangos de filas y columnas tengan asignados dichos valores **r**, **g** y **b**. Modifica el programa principal del ejercicio 2 para que, tras crear la matriz inicial con *inimatrizrgb.m*, llame a *rectrgb.m*. Para finalizar mostrar el contenido de la matriz resultante.

5. Desde un programa principal, haz uso de las funciones creadas anteriormente para crear una imagen inspirada en alguna de las pinturas de Piet Mondrian. Visualiza el resultado con **imshow**.

11.4.2. Propuestos como actividad independiente

1. Escribe una función denominada *raices* que acepte los tres coeficientes de un polinomio de segundo grado y devuelva las dos raíces si son reales. Si no son reales las raíces deben tener el valor *NaN*.

2. Escribe una función que acepte la parte real e imaginaria de dos números complejos A y B (en la forma *[reSuma,imSuma] = sumacompleja(realA, imaginariaA, realB, imaginariaB))* y devuelva la parte real e imaginaria de la suma de ambos números complejos.

3. Escribe una función que acepte la parte real e imaginaria de dos números complejos A y B (en la forma *[reProducto,imProducto] = productocomplejo(realA, imaginariaA, realB, imaginariaB)*) y devuelva la parte real e imaginaria del producto de ambos números complejos.

4. Escribe una función (en la forma *k = serie(realV,imaginariaV)*) acepte la parte real e imaginaria de un número complejo **V** que haciendo uso de las dos anteriores, realice un máximo de 100 iteraciones de la sucesión de números complejos:

$Z_i = Z_{i-1}^2 + V,\ con\ Z_0 = 0 + 0i$

Para un determinado valor de **V**, la función devuelve la iteración, **k**, en que el valor absoluto de Z_i es mayor que 2, o bien 100 cuando Z_i no supera dicho valor tras las 100 iteraciones.

Nota: Cada iteración, $Z_i = Z_{i-1}^2 + V$, exige el cálculo del cuadrado del número complejo Z_{i-1}. Para calcular el cuadrado harás uso de la función que realiza el producto de números complejos realizada en el ejercicio 3. El número complejo resultante de elevar al cuadrado Z_{i-1} se sumará al número complejo **V**, para lo que harás uso de la función de suma de complejos que hayas definido en el ejercicio 2.

5. Escribir un programa que tras definir el número **m** y **n** de valores equiespaciados respectivamente en el eje x e y del plano complejo ($-2{,}1 < x < 0{,}9$ y $-1{,}5 < y < 1{,}5$), cree una matriz **R** de dimensiones $m \times n$ donde el valor de la posición $(fila, columna)$ será el resultado de la llamada a la función del ejercicio 4 *serie(x(columna),y(fila))*. Una vez calculada la matriz, visualizarla con el comando *image(R)*, comando que permite interpretar cada posición de la matriz como un píxel de una imagen.

11.5. TAREA

La evaluación de esta práctica, se realiza a través de un cuestionario sobre el contenido de este documento y las actividades realizadas durante su correspondiente sesión práctica. El cuestionario estará disponible para su resolución el viernes21 de mayo de 12 a 18 horas a través del campus virtual.

Práctica 12

Utilidades para la ingeniería

12.1. Introducción

En esta práctica describimos de forma breve diversas utilidades que ofrece el entorno Matlab para resolver problemas habituales en ingeniería, haciendo uso del lenguaje m.

12.2. Interpolación

En ocasiones se dispone de valores, datos o medidas asociados a puntos o momentos concretos, y se hace necesario obtener valores en puntos intermedios. Por ejemplo, si contamos con un sensor que realiza mediciones de la temperatura, y, en distintos momentos del día, x, se contará con un nutrido conjunto de datos, pero no con la información continua para cualquier instante a los largo del día. Al disponer del repertorio de datos, puede realizarse una estimación de y para valores de x en los que no se tiene la medición real, por medio de una técnica de interpolación. La técnica más sencilla es la interpolación lineal, que asume que el valor de y entre dos puntos conocidos se puede aproximar por una línea recta. El comando *interpl*, interpola conocidos pares (x, y) los valores de y para valores en el rango de x:

```
%Pares de puntos disponibles
x = [1 2 3 4];
y = [1 4 9 16];

interp1(x,y,[1.5 2.5 3.5 4.5])
ans =
    2.5000    6.5000   12.5000       NaN
```

Con la interpolación lineal, interpolar fuera del rango conocido de **x** nos produce un valor erróneo *NaN*.

La interpolación lineal es la aplicada por defecto, siendo posible variantes como por ejemplo el *spline* cúbico, que además permite interpolar fuera del rango conocido.

```
interp1(x,y,[1.5 2.5 3.5 4.5],'spline')

ans =
    2.2500    6.2500   12.2500   20.2500
```

Un ejemplo final compara gráficamente la salida de ambas interpolaciones, lineal y basada en *spline*. ver figura 12.1, dados cinco pares de puntos de partida:

```
%Pares de puntos disponibles
x = [0 1 2 3 4];
y = [0 2 0 2 0];

xe=0:0.1:4.5;
ye=interp1(x,y,xe);
ye2=interp1(x,y,xe,'spline');

plot(xe,ye,'r-x',xe,ye2,'b-o')
legend('lineal','spline');
```

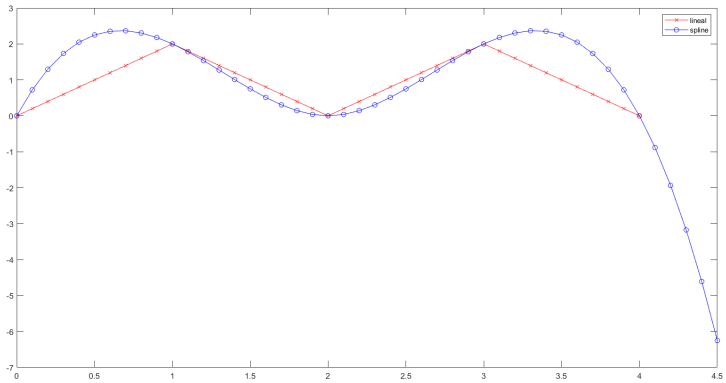


Figura 12.1: Comparativa de interpolación lineal y basada en *spline*.

12.3. Polinomios

Un polinomio puede evaluarse de forma directa para un determinado valor de **x**, o representarse como un vector **y** hacer uso de la función *polyval*:

```
%4*x^3 + 3*x^2 -2*x + 10 = 0
pol = [4 3 -2 10];
%Obtiene para un valor concreto de x
x=5;
polyval(pol, x)
```

```
ans =
   575
```

La representación como vector se adapta al uso de funciones para el cálculo de la derivada, integral o de ceros o raíces de una función polinomial:

```
%Derivada
dpol=polyder(pol)
%Integral (asume constante nula)
ipol=polyint(pol)
%raíces
roots(pol)

dpol =
    12     6    -2

ipol =
     1     1    -1    10     0

ans =
  -1.7997 + 0.0000i
   0.5248 + 1.0553i
   0.5248 - 1.0553i
```

12.4. BÚSQUEDA DE CEROS DE UNA FUNCIÓN

Los ceros, soluciones o raíces de una función, $f(x) = 0$, son puntos de corte con el eje x o abscisa. En el apartado previo se ha descrito su cálculo para funciones polinomiales, siendo necesario para funciones más generales el uso de la función *fzero*, que permite calcular las raíces de una función genérica. Dada una función $y = f(x)$ el objetivo es localizar una raíz de esta función, $f(x) = 0$, en el intervalo $x = [a, b]$. Para funciones sencillas basta con llamar a la mencionada función *fzero*:

```
x = fzero('sin(x)',3)
```

En la llamada a *fzero*, el segundo argumento indica un valor cercano a la posible raíz, dado que una función puede contar con muchos veros. Piensa por ejemplo en lo uqe ocurre con la función sinusoidal. Para funciones más elaboradas, la definición de la función puede realizarse a través de un fichero m, como se ha descrito en la práctica precedente:

```
function f = mifuncion (x)
f = x.^2-3;            %cálculo de la función en el valor x
```

O como función anónima (sin fichero m asociado), donde la definición de la función se realiza como si fuese una variable con la siguiente sintaxis:

```
nombre = @(variable) expresión;
```

Definiendo el *nombre* con el que se llama a la función, la *variable* de l que depende la función (si hubiera varias se separan con comas), y *expresión* establece la expresión de la función a definir. A modo de ejemplo, para definir la función $y = x^2 - 3$ es válido lo siguiente:

```
y=@(x) x.^2-3;
```

El operador .∧ se utiliza en caso necesario, como podrían serlo .∗ y ./. A partir de este momento, cualquier referencia a **y** es la función que hemos definido previamente.

Un ejemplo mínimo de uso del comando *fzero* definiendo una función de ambas formas

```
%definición en archivo
fa=@mifuncion;

%definición anónima
fb=@(x) x.^2-3;

fzero(fa,2)
fzero(fb,2)
```

12.5. Resolución de sistemas de ecuaciones lineales

Recordar la técnica de resolución de sistemas ecuaciones lineales, mencionada en la segunda práctica:

```
% 3.5*x + 2*y          = 5
% -1.5*x + 2.8*y + 1.9*z = -1
%        - 2.5*y +   3*z = 2

A = [3.5 2 0; -1.5 2.8 1.9; 0 -2.5 3];
B = [5; -1; 2];

x = inv(A)*B
```

Sin embargo, este método de resolución, requiere el cálculo de la matriz inversa, que es computacionalmente costoso, además de existir la posibilidad de no tener solución. Por ello es más aconsejable utilizar la función *linsolve* o la barra invertida.

```
>> x=A\B
x =
    1.4421
   -0.0236
    0.6470

>> x=linsolve(A,B)
```

```
x =
    1.4421
   -0.0236
    0.6470
```

En el caso de ser un sistema sin solución, Matlab proporciona una solución, incluyendo una advertencia sobre la posible singularidad de la matriz.

```
A = [3.5 2 0; -1.5 2.8 1.9; 0 -2.5 3];
B = [5; -1; 2];

x = inv(A)*B

Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  3.364312e-18.

x =
   1.0e+16 *

    1.8014
   -1.8014
   -1.8014
```

12.6. INTEGRACIÓN DE FUNCIONES

Para el cálculo de integrales definidas, se sugieren las funciones (*integral*, *integral2* e *integral3*), dependiendo de si se pretende calcular una integral simple, doble o triple:

```
%Función anónima y=x^2, integrando  en el rango (0,1)
f = @(x) x.^2;
a = 0;
b = 1;
res = integral(f,a,b)

%f(x,y)=y*sin(x)+x*cos(y)
f = @(x,y) y.*sin(x)+x.*cos(y);
x1=pi; x2=2*pi;
y1=0; y2=pi;
res2 = integral2(f, x1,x2,y1,y2)

%f(x,y,z)=y*sin(x)+z*cos(x)
f = @(x,y,z) y.*sin(x)+z.*cos(x);
x1=0; x2=pi;
y1=0; y2=1;
z1=-1; z2=1;
res3 = integral3(f, x1,x2,y2,y2,z1,z2)
```

12.7. ECUACIONES DIFERENCIALES ORDINARIAS

Matlab cuenta con varios comandos para resolver de forma numérica ecuaciones diferenciales. En este apartado se describe brevemente el uso del comando *ode45*.

Para resolver la ecuación diferencial ordinaria de primer orden $\frac{dy}{dt} = f(t,(y(t))$ en el intervalo $[a,b]$ con la condición inicial $y_{t_0} = y_0$, en primer término se define la función, a través de fichero o de forma anónima. Como ejemplo ilustrativo, se considera la Ley de Enfriamiento de Newton[1], $\frac{dT}{dt} = -k \cdot (T - T_m)$, siendo k el decaimiento, T_m la temperatura ambiente. El siguiente código, tras definir la función anónima, calcula el comportamiento con un determinado valor de k y temperatura inicial 50°C, para terminar visualizando la bajada de temperatura, ver figura 12.2:

```
dT = @(t,T) -0.02*(T-25);
t=0:0.5:120;
[t,T]=ode45(dT,t,50);
plot(t,T)
```

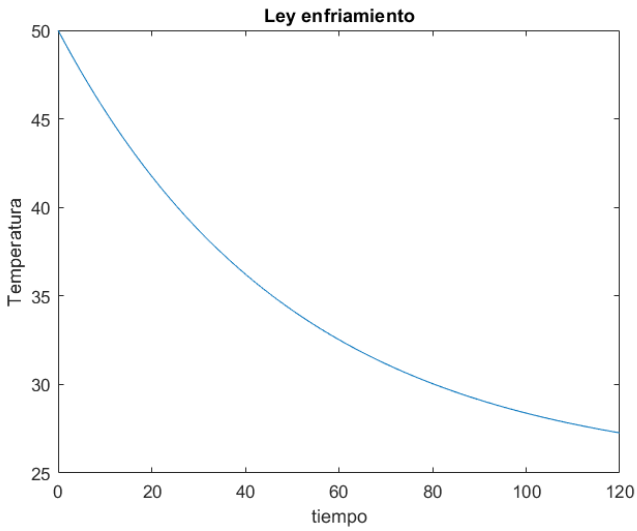


Figura 12.2: Aplicación concreta de la Ley de Enfriamiento.

12.8. COMPUTACIÓN SIMBÓLICA

Es posible la definición de símbolos para su posterior uso, no como variables, sino en cálculos simbólicos. Mostramos un ejemplo básico de cálculo de derivadas e integrales.

[1] https://es.wikipedia.org/wiki/Ley_del_enfriamiento_de_Newton

```
% Definición variables simbólicas
syms a b x d

% Obtención de la derivada
d = diff(a*x^2+b*x)

% Integrar para obtener la función original
int(d)

% Resolución de ecuaciones diferenciales "dsolve"
dsolve('D2y = 6*y - Dy','x')
```

12.9. AJUSTE POR MÍNIMOS CUADRADOS

En ocasiones se cuenta con un conjunto de medidas o valores redundantes, y se requiere ajustar una función que aproxime el sistema. Una técnica es hacer uso del criterio de que la suma del cuadrado de los residuos sea mínima.

Conocido un conjunto de n datos experimentales expresados como pares de puntos en el plano XY: $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, si el objetivo es ajustar a una recta, se conoce como recta mínimos cuadrados, partiendo su expresión $y = a \cdot x + b$, donde:

$$a = \frac{n \cdot (x_1 \cdot y_1 + x_2 \cdot y_2 + ... + x_n \cdot y_n) - (x_1 + x_2 + ... + x_n) \cdot (y_1 + y_2 + ... + y_n)}{n \cdot (x_1^2 + x_2^2 + ... + x_n^2) - (x_1^2 + x_2^2 + ... + x_n^2)}$$

$$b = \frac{(y_1 + y_2 + ... + y_n)}{n} - a \cdot \frac{(x_1 + x_2 + ... + x_n)}{n} \quad (12.1)$$

Matlab cuenta con el comando *polyfit* para ajustar los datos a un polígono, proporcionado como argumentos además de los pares de puntos, el grado del polígono que se requiere. El siguiente código ajusta a una línea, dado un conjunto de puntos inicial, ver figura 12.3:

```
%Datos disponibles
x=[1:10];
%y en principio es 2x, pero perturbamos levemente
pert=(rand(1,length(x))-0.5)*2;
y=2*x+pert;

%Ajuste de la forma y = ax + b
r=polyfit(x,y,1);

%r obtiene los coeficientes d ela recta
%Dibuja puntos de partida
plot(x,y,'r*')

%Puntos de la recta ajustada
```

```
xp=linspace(x(1),x(end),20);
yp=polyval(r,xp);
hold on
plot(xp,yp,'b')
```

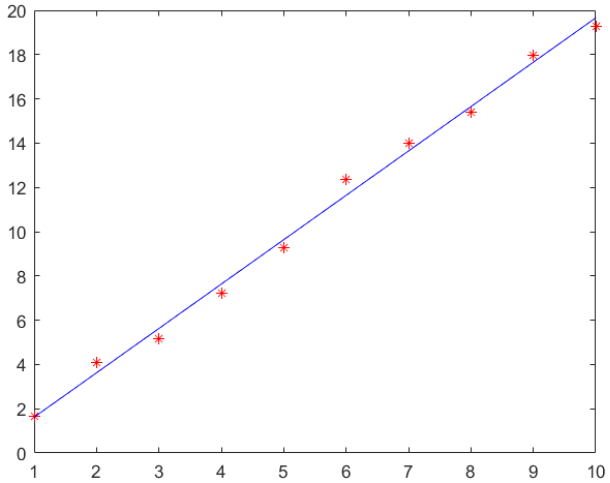


Figura 12.3: Ajuste por mínimos cuadrados.

12.10. SIMULACIÓN CON IMÁGENES NO ESTÁTICAS

Si bien no es la herramienta más adecuada para ello, es posible crear animaciones a partir de imágenes estáticas de forma similar a como se realiza en los dibujos animados o videojuegos, que aplicaremos para la visualización de un fenómeno cinemático.

12.10.1. Puntos aleatorios

Recuperando la imagen negra con un único plano, es decir sin color, el siguiente código pinta aleatoriamente puntos blancos en ella, usando un bucle y *rand* para decidir su localización antes de visualizar con *imshow*, ver figura 12.4:

```
%Imagen negra
a=zeros(400,400);
%Puntos aleatorios
for i=1:100 %Número de puntos
    %Posiciones aleatorias
    px=floor(rand()*size(a,2));
    py=floor(rand()*size(a,1));
    %punto blanco
    a(py,px)=1;
```

```
end
imshow(a);
```

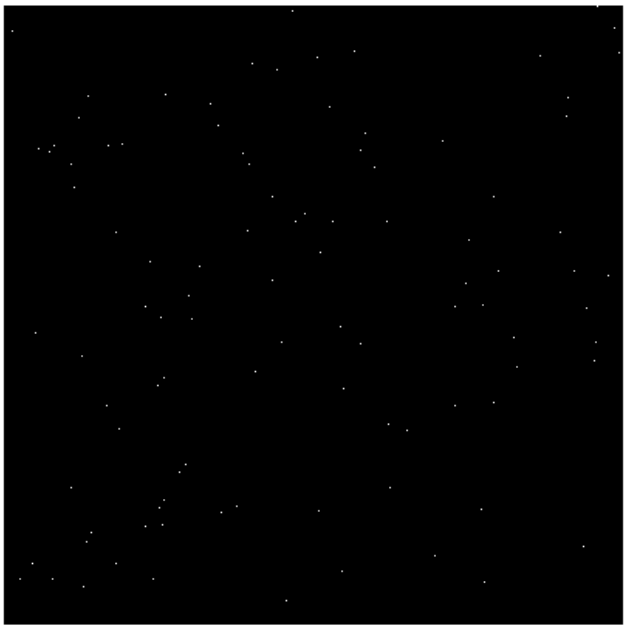


Figura 12.4: imagen negra ocn puntos aleatorios.

Es una imagen estática que no cambia. Sin embargo, es posible crear la ilusión de una animación. Un primer paso es dibujar los puntos de manera progresiva tras añadir cada punto. Para ello, la llamada a *imshow* se coloca dentro del bucle y se añade una pausa, comando *pause*, para que no sea un efecto excesivamente rápido:

```
%Imagen negra
a=zeros(400,400);
%Puntos aleatorios
for i=1:100 %Número de puntos
    %Posiciones aleatorias
    px=floor(rand()*size(a,2));
    py=floor(rand()*size(a,1));
    %punto blanco
    a(py,px)=1;
    imshow(a); %Mostramos la nueva imagen
    pause(0.1) %Hacemos una pausa para permitir que el ojo lo vea
end
```

En lugar de ir añadiendo punto a punto, una variante consiste en borrar la imagen en

cada iteración, es decir, que únicamente se vea el último punto dibujado. Para ello, basta con mover la inicialización de la imagen dentro del bucle.

```
%Puntos aleatorios
for i=1:100 %Número de puntos
    %Imagen negra
    a=zeros(400,400);
    %Posiciones aleatorias
    px=floor(rand()*size(a,2));
    py=floor(rand()*size(a,1));
    %punto blanco
    a(py,px)=1;
    imshow(a); %Mostramos la nueva imagen
    pause(0.1) %Hacemos una pausa para permitir que el ojo lo vea
end
```

Como los puntos pintados son pequeños, ocupan un píxel, no son fáciles de ver. Para hacer el efecto de dibujar puntos más gruesos, puede rellenarse una submatriz dentro de la matriz. El siguiente código rellena desde la posición $(1,1)$ hasta la $(10,10)$.

```
%Imagen negra
a=zeros(400,400);
%Rellenamos submatrix 10x10, el punto grueso
for i=1:10
    for j=1:10
        a(i,j)=1;
    end
end
imshow(a)
```

Una vez que está clara la forma de dibujar el punto grueso, es plausible dibujarlo no solo a partir del punto (1,1) de la imagen, sino en otras posiciones hacia la derecha: (1,21), (1, 41), (1,61), ..., (1,381):

```
%Imagen negra con tres planos (RGB)
a=zeros(400,400,3);
%Fijamos las dimensiones del punto blanco
salto=20;
%Cuenta las repeticiones de puntos gruesos
for nsaltosx=0:size(a,2)/salto
    %Rellenamos submatrix 10x10, el punto grueso
    for i=1:10
        for j=1:10
          %Va desplazando los puntos hacia la derecha
            a(i,j+salto*nsaltosx,1)=1;
        end
    end
end
imshow(a)
```

Puede utilizarse el operador dos puntos, con el que especificamos de forma alternativa el relleno de la submatriz 10×10, de una forma más compacta:

```
%Imagen negra con tres planos (RGB)
a=zeros(400,400,3);
%Fijamos las dimensiones del punto blanco
salto=20;
%Cuenta las repeticiones de puntos gruesos
for nsaltosx=0:size(a,2)/salto
    %Rellenamos submatrix 10x10, el punto grueso
    a(1:10,1+salto*nsaltosx:10+salto*nsaltosx,1)=1;
end
imshow(a)
```

Conocido el mecanismo para dibujar puntos más gruesos, recuperando el ejemplo de los puntos blancos que se dibujan en localizaciones aleatorias en la imagen, pero ahora con un grosor de 5.

```
%Grosor del punto
grosor=5;
%Puntos aleatorios
for i=1:100 %Número de puntos
    %Imagen negra
    a=zeros(400,400);
    %Posiciones aleatorias
    px=floor(rand()*size(a,2));
    py=floor(rand()*size(a,1));
    %punto blanco grueso
    for n=1:grosor
        for m=1:grosor
            %Que no se salga de la matriz
            if py>0 && py+n<=size(a,1) && px>0 && px+m<=size(a,2)
                a(py+n,px+m)=1;
            end
        end
    end
    imshow(a); %Mostramos la nueva imagen
    pause(0.1) %Hacemos una pausa para permitir que el ojo lo vea
end
```

12.10.2. Caída por gravedad

El siguiente objetivo será dar movimiento definido al punto. De forma aleatoria decidiremos una localización inicial en la parte superior de la imagen (fila 1) por donde aparecerá un punto, que a partir de ese momento caerá, un comportamiento cinemático producido por la fuerza de gravedad. Para ello se calcula su desplazamiento en el eje y, considerando una velocidad inicial nula. Una vez definido el escenario, la expresión que modela el espacio en función del tiempo según el movimiento rectilíneo uniformemente acelerado: $e(t) = e_0 + v_0 \cdot t + \frac{1}{2} a \cdot t^2$ Siendo en nuestro caso nulos los dos primeros términos, limitándose el desplazamiento al eje y. El código que se sugiere para repetir el proceso con diez puntos:

```
grosor=5; %Grosor del punto
%Para sucesivos puntos aleatorios que caen desde arriba
for i=1:10 %Repeticiones
    %Imagen negra
    im=zeros(400,400);
    %Cae desde arriba en un lugar aleatorio
    px=floor(rand()*size(im,1));
    py=1;
    %Inicializaciones
    t=0;
    a=98;
    p=0.1;

    %Mientras no llegue al suelo (parte inferior o última fila de la matriz)
    while py<size(im,1)
        %Actualiza la posición del punto a dibujar y tiempo transcurrido
        py=round(1/2*a*t*t);
        t=t+p;

        %Pinta en blanco el nuevo punto
        for n=1:grosor
            for m=1:grosor
                %Que no se salga de la matriz
                if py >0 && py+n<=size(im,1) && px>0 && px+m<=size(im,2)
                    im(py+n,px+m)=1;
                end
            end
        end
        imshow(im);%Mostramos la nueva imagen
        pause(p);
    end
end
```

Ha dibujado la estela, se evita borrando el punto previamente pintado, antes de actualizar su posición para el nuevo valor de **t**.

```
clear;
grosor=5; %Grosor del punto
%Para sucesivos puntos aleatorios que caen desde arriba
for i=1:10 %Repeticiones
    %Imagen negra
    im=zeros(400,400);
    %Cae desde arriba en un lugar aleatorio
    px=floor(rand()*size(im,1));
    py=1;
    %Inicializaciones
    t=0;
    a=98;
    p=0.1;

    %Mientras no llegue al suelo (parte inferior o última fila de la matriz)
    while py<size(im,1)
        %Borra previo pintando en negro
```

```
        if t>0 %No lo hace la primera vez, sólo cuando ya haya pintado algo
         %Pinta en negro el último punto dibujado
         for n=1:grosor
            for m=1:grosor
                %Que no se salga de la matriz
                if py >0 && py+n<=size(im,1) && px>0 && px+m<=size(im,2)
                    im(py+n,px+m)=0;
                end
            end
         end
        end
        %Actualiza la posición del punto a dibujar y tiempo transcurrido
        py=round(1/2*a*t*t);
       t=t+p;

        %Pinta en blanco el nuevo punto
        for n=1:grosor
            for m=1:grosor
                %Que no se salga de la matriz
                if py >0 && py+n<=size(im,1) && px>0 && px+m<=size(im,2)
                    im(py+n,px+m)=1;
                end
            end
        end
        imshow(im); %Mostramos la nueva imagen
        pause(p);
    end
end
```

Como remate final añadamos la reproducción de un sonido, una onda, para el momento en que la partícula llegue al suelo.

```
clear;
grosor=5; %Grosor del punto

%Sonido
N=100;
s=zeros(N,1);
for a=1:N
    s(a)=tan(a);
end
Fs=10000;

%Para sucesivos puntos aleatorios que caen desde arriba
for i=1:10 %Repeticiones
    %Imagen negra
    im=zeros(400,400);
    %Cae desde arriba en un lugar aleatorio
    px=floor(rand()*size(im,1));
    py=1;
    %Inicializaciones
    t=0;
    a=98;
    p=0.1;
```

```
    %Mientras no llegue al suelo (parte inferior o última fila de la matriz)
    while py<size(im,1)
        %Borra previo pintando en negro
        if t>0 %No lo hace la primera vez, sólo cuando ya haya pintado algo
         %Pinta en negro el último punto dibujado
         for n=1:grosor
            for m=1:grosor
                %Que no se salga de la matriz
                if py >0 && py+n<=size(im,1) && px>0 && px+m<=size(im,2)
                    im(py+n,px+m)=0;
                end
            end
         end
        end

        %Actualiza la posición del punto a dibujar y tiempo transcurrido
        py=round(1/2*a*t*t);
       t=t+p;

        %Pinta en blanco el nuevo punto
        for n=1:grosor
            for m=1:grosor
                %Que no se salga de la matriz
                if py >0 && py+n<=size(im,1) && px>0 && px+m<=size(im,2)
                    im(py+n,px+m)=1;
                end
            end
        end
        imshow(im);%Mostramos la nueva imagen
        pause(p);
    end
    %Lanza el sonido tras llegar al suelo
    soundsc(s,Fs)
end
```