PRÁCTICAS CON MATLAB. ÁLGEBRA
HOJA 1.- INTRODUCCIÓN A MATLAB. MATRICES Y VECTORES. SISTEMAS DE ECUACIONES LINEALES

Introducción a MatLab

MATLAB es un programa de cálculo científico de gran versatilidad y facilidad de uso con un gran número de herramientas orientadas a una amplia diversidad de aplicaciones. El nombre de MATLAB proviene de las iniciales de MATrix LABoratory. La estructura de las instrucciones en MATLAB es muy parecida a la sintaxis de las expresiones algebraicas y las operaciones comúnmente utilizadas en Álgebra Lineal.

En MATLAB tecleando la instrucción help podemos visualizar un buen número de directorios que contienen los nombres de las instrucciones. Escribiendo help nom, donde nom es el nombre de una instrucción, nos proporciona información de la instrucción específica cuyo nombre(nom) hemos escrito. Una vez hayamos entrado en MATLAB veremos aparecer su logotipo >>, esto nos indica que MATLAB está esperando una instrucción. Mencionemos algunas características del MATLAB sobre las cuales tenemos que estar informados antes de comenzar a utilizarlo:

- Todas las instrucciones del MATLAB se escriben en letras minúsculas.
- Después de escribir el nombre de una instrucción hemos de pulsar ENTER.
- MATLAB guarda un número determinado de instrucciones, las más recientes, en un dispositivo de "almacenamiento", dichas instrucciones se pueden recuperar utilizando la flecha hacia arriba y hacia abajo.

El entorno de trabajo de MATLAB ha mejorado mucho a partir de la versión 6.0, haciéndose mucho más gráfico e intuitivo, similar al de otras aplicaciones profesionales de Windows. Algunos de los componentes más importantes del entorno de trabajo de MATLAB 6.5 son los siguientes:

1. La ventana de comandos (Command Window),
2. La ventana histórica de comandos (Command History),
3. El espacio de trabajo (Workspace),
4. El directorio actual (Current Directory),
5. La ventana de ayuda (Help)

Modo numérico y modo simbólico

Lo primero que debemos diferenciar en el uso de MATLAB es cuándo estamos trabajando en forma numérica o en forma simbólica. La primera conlleva que las operaciones aritméticas que realizamos se realicen tratando los números con precisión finita (con decimales), luego todas las operaciones

realizadas serán aproximaciones con un cierto número de decimales exactos, que dependerá de la precisión que queramos obtener. Cuando trabajamos en simbólico, el ordenador simula una aritmética de precisión infinita, como si todos los números reales pudiesen ser concebidos simbólicamente por el ordenador. Las expresiones simbólicas no son más que cadenas de caracteres que representan números, funciones, variables, etc. Así, podemos manipular expresiones algebraicas que involucren estas variables, parámetros o funciones.

MATLAB, por defecto, trabaja en numérico y el uso de cualquier variable o función simbólica debemos indicárselo expresamente. Por ejemplo, si queremos utilizar las constantes reales a, b y c en forma simbólica para realizar operaciones algebraicas debemos indicarlo mediante:

```
>> syms a b c real
>> x=a^2+b^2+c^2
```

Es frecuente también manipular números de manera algebraica, para ello debemos transformarlos previamente a modo simbólico mediante la función `sym`:

```
>> b=0.5
>> b=sym(b)
```

Si queremos pasar b a modo numérico de nuevo, debemos teclear:

```
>> b=double(b)
```

Matrices y vectores

Es importante entender que el carácter (numérico o simbólico) de los argumentos de las funciones que podemos utilizar con MATLAB depende de cómo están definidas internamente. Así, una función puede admitir argumentos solo numéricos, solo simbólicos o ambos. El siguiente cuadro recoge algunas de las funciones que vamos a utilizar en esta práctica junto con el tipo de argumentos que admite:

FUNCIÓN	SYM	DOUBLE	DESCRIPCIÓN
`rank(A)`	x	x	calcula el rango de A
`det(A)`	x	x	calcula el determinante de A
`colspace(A)`	x		devuelve una base de la imagen de A
`null(A)`	x	x	devuelve una base del núcleo de A
`inv(A)`	x	x	calcula la inversa de A
`[L,U,P]=lu(A)`		x	calcula la factorización LU de A
`factor(ex)`	x		factoriza un expresion ex
`expand(ex)`	x		desarrolla un expresion ex
`solve('ecu','var')`	x		resuelve ecuaciones

En MatLab, se puede definir una matriz $A = (a_{ij})$ mediante el uso del corchetes del siguiente modo:

>> $A = [a_{11}\ ,\ a_{12}\ ,\ \ldots\ ,\ a_{1n}\ ;\ a_{21}\ ,\ a_{22}\ ,\ \ldots\ ,\ a_{2n}\ ;\ \ldots\ ;\ a_{m1}\ ,\ a_{m2}\ ,\ \ldots\ ,\ a_{mn}]$

Es decir, cada fila se separa por puntos y comas, y los elementos de una misma fila se separan por comas (o espacios en blanco).

Ejercicio 1. *Introduce las siguientes matrices en forma simbólica:*

$$A = \begin{pmatrix} 1 & -2 & 0 \\ 4 & -1 & 3 \\ 5 & 0 & 6 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 2 & -3 \\ -1 & 4 & 5 \\ 2 & 0 & 7 \end{pmatrix}, \quad C = \begin{pmatrix} -2 & 3 & 0 \\ 1 & -1 & 1 \end{pmatrix}$$

A continuación, realiza las siguientes operaciones:

(a) $A + B$

(b) $A + C$ *¿Qué ocurre?*

(c) $A \cdot B$ *y* $B \cdot A$ *¿Coinciden?*

(d) $C \cdot A$ *y* $A \cdot C$ *¿Qué ocurre?*

(e) $(A + B)^T$ *y* $A^T + B^T$. *¿Qué puedes afirmar?*

(f) $(A \cdot B)^T$ *y* $B^T \cdot A^T$. *¿Qué puedes afirmar?*

(g) $|A \cdot B|$ *y* $|A| \cdot |B|$.

Ejercicio 2. *Dada la matriz* $E = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$, *obtener su matriz inversa y comprobar que* $E^{-10} = (E^{-1})^{10}$.

Ejercicio 3. *Dada la matriz* $G = \begin{pmatrix} 1 & -2 & 3 \\ 4 & -5 & 6 \\ 7 & -8 & 10 \end{pmatrix}$.

(a) ¿Tiene inversa?

(b) En caso afirmativo, calcularla y verificar el resultado.

(c) Definir $H = 2G - G^{-1}$. *Calcular* $(G \cdot H)^{-1}$.

Ejercicio 4. *Ejecuta las siguientes sentencias y trata de inferir qué realiza cada una de ellas:*

```
>> eye(10)
```

```
>> diag([1,2,3,4,5,6,7,9])
```

```
>> ones(3,4)

>> zeros(10,3)

>> diag([1 -1 2;2 -1 3;3 -1 3])
```

Ejercicio 5. *Dada la matriz* $M = \begin{pmatrix} 1 & 1 & 1 \\ a & b & c \\ a^2 & b^2 & c^2 \end{pmatrix}$, *donde* a, b *y* c *son números reales. Demuestra que* $|M| = (a-b)(a-c)(c-b)$.

Ejercicio 6. *Calcular el valor del parámetro real* a *para que la siguiente matriz sea singular (no regular):*

$$K = \begin{pmatrix} a+1 & a & a & a \\ a & a+1 & a & a \\ a & a & a+1 & a \\ a & a & a & a+1 \end{pmatrix}$$

En estos primeros ejercicios hemos visto cómo se puede utilizar la ventana "Command Window" de MatLab para realizar cálculos concretos. Hay que reseñar que MatLab no permite guardar los resultados obtenidos en esta ventana. Si tratamos de guardar la sesión, MatLab únicamente guardará las variables de entorno (las que aparecen en "Workspace"). La razón de que no se pueda salvar toda esta información es que el programa MatLab tiene una filosofía orientada a trabajar con **scripts**.

¿Qué es un script?

Los scripts son ficheros externos planos con extensión .m que contienen una secuencia de sentencias MatLab. Tecleando simplemente el nombre del fichero sin la extensión .m en la ventana "Command Window", el programa leerá y ejecutará secuencialmente todas las sentencias que aparezcan en el fichero. Además, desde un script se puede llamar a otros scripts, dotando así a MatLab de una versatilidad muy práctica. MatLab admite sentencias típicas en programación que te permiten crear bucles, bifurcaciones, etc, lo que hace de MatLab una potente herramienta de *programación*.

Es importante asegurarnos que el fichero .m al que llamamos se encuentre dentro de la carpeta de trabajo que tengamos activa en el **"Current Directory"**; así, por ejemplo, si queremos leer un fichero llamado "scriptmatlab.m" debemos activar la carpeta donde se encuentre el fichero y teclear:

```
>> scriptmatlab
```

Como ejemplo, el siguiente script resuelve con detalle el Ejercicio 3:

```
%RESOLUCIÓN DEL EJERCICIO 3
disp('EJERCICIO 3')
```

```
G=sym([1 -2 3;4 -5 6;7 -8 10]);
disp('Apartado (a)')
disp('El determinante de G es: ')
detG=det(G)
disp('por tanto G es invertible')

disp('Apartado (b)')
pause
disp('La inversa de G es: ')
invG=inv(G)
disp('Calculamos invG*G y obtenemos')
G*invG
disp('Lo que prueba que invG es efectivamente la inversa de G')

disp('Apartado (c)')
pause
disp('Calculamos H: ')
H=2*G-inv(G)
disp('Finalmente calculamos (G*H)^(-1) y obtenemos: ')
inv(G*H)
```

Llámalo ejercicio3.m y ejecútalo en la pantalla "Command Windows" . Es fácilmente deducible lo que hace cada sentencia dentro del script, en particular, es importante obserbar que cuando al final de una sentencia se pone el símbolo de punto y coma ";", entonces no se muestra el resultado por pantalla.

Ejercicio 7. *Introducir las siguientes matrices mediante la lectura del fichero "datosmatrices.m":*

$$A = \begin{pmatrix} 1 & 2 & 3 \\ -1 & 3 & 7 \end{pmatrix} \quad B = \begin{pmatrix} 0 & 2 & 3 & 4 & 0 \\ -1 & 2 & 5 & 7 & 8 \\ 0 & 1 & 1 & 0 & 4 \\ 1 & 1 & -1 & -3 & -4 \end{pmatrix}$$

$$C = \begin{pmatrix} 2 & 1 & 6 & 4 \\ 3 & 5 & 8 & 6 \\ 4 & 7 & 6 & 4 \\ 76 & 45 & 8 & 11 \end{pmatrix} \quad D = \begin{pmatrix} -5 & 1 & 0 & 7 & 0 \\ 3 & 34 & 3 & 2 & 0 \\ 0 & 6 & 7 & -8 & 11 \\ 9 & 6 & 5 & 3 & 0 \\ 1 & 3 & -2 & 0 & 2 \\ 0 & 7 & 9 & 3 & 8 \end{pmatrix}$$

$$E = \begin{pmatrix} 1 & -1 & 3 & 2 & -2 \\ -6 & 36 & -28 & 28 & 2 \\ -1 & 7 & -5 & 6 & 0 \\ 4 & -16 & 16 & -8 & -4 \end{pmatrix} \qquad F = \begin{pmatrix} 0 & 1 & 2 & -2 & -1 \\ 4 & -2 & -1 & 0 & 1 \\ 2 & 2 & -1 & 3 & -3 \\ 2 & 1 & -1 & -4 & -1 \\ 8 & 2 & -1 & -1 & -4 \end{pmatrix}$$

Ejercicio 8 (Uso de los dos puntos)**.** *Teclear las siguientes sentencias y tratar de describir los resultados obtenidos:*

```
>> u=2:100
>> v=-10:10
>> w=0:2:20
>> u(5)
>> v([5,10,15])
>> w(2:2:6)
>> A(1,2)
>> D(:,[1,3])
>> F(2:5,:)
```

Ejercicio 9. *Ejecutar las siguientes sentencias y tratar de describir qué devuelve cada una de ellas:*

```
>> length(w)
>> size(B)
>> size(B,1)
>> size(B,2)
>> [B,E]
>> [B;D]
```

Ejercicio 10. *Calcular la factorización LU, y en su caso la matriz P de permutaciones, de la matriz*

$$R = \begin{pmatrix} 3 & -1 & 0 & 5 \\ -3 & 1 & 2 & 3 \\ 0 & 1 & -1 & 0 \\ 2 & 0 & 2 & 1 \end{pmatrix}.$$

Comprobar que los resultados son correctos.

Resolución de sistemas de ecuaciones lineales

Para resolver **sistemas de ecuaciones lineales** de la forma $AX = B$, donde A es la matriz de coeficientes del sistema y B el término independiente, debemos proceder de la forma siguiente:

(1) Definimos la matriz de coeficientes A y el término independiente B.

(2) Comprobamos si el sistema es compatible (mediante Rouche-Frobenius).

(3) Calculamos una solución particular mediante la sentencia `A\B`.

(4) Hallamos la solución general en forma paramétrica teniendo en cuenta que ésta es suma de la solución particular anterior y de cualquier solución del sistema homogéneo $AX = 0$ (que es una combinación lineal de las columnas de la matriz `null(A)`).

Ejemplo 1. *Realizar un script que resuelva el siguiente sistema de ecuaciones lineales:*

$$\begin{cases} x_1 - x_2 + 2x_3 + 3x_4 + x_5 = -2 \\ 3x_1 + x_2 + x_4 + x_5 = 4 \\ x_2 - x_3 + x_5 = 2 \end{cases}$$

Solución.-

```
%SOLUCION DEL EJEMPLO 1
disp('EJEMPLO 1')
A=sym([ 1, -1,  2, 3, 1; 3,  1,  0, 1, 1; 0,  1, -1, 0, 1]);
B=sym([-2 4 2]');
disp('El rango de la matriz de sistema es:')
rank(A)
disp('El rango de la matriz ampliada es:')
rank([A B])
disp('Por tanto el sistema es compatible indeterminado')
disp('Una solución particular está dada por:')
X0=A\B
syms a1 a2 real; %parámetros de la solución
par=[a1;a2];
disp('La solución general del sistema es: ')
X=X0+null(A)*par
disp('donde a1 y a2 son parámetros reales')
```

Ejercicio 11. *Realizar 3 scripts que resuelvan los siguientes sistemas de ecuaciones lineales:*

(1)
$$\begin{cases} 6x_1 + x_3 + 10x_4 + 3x_5 - 3x_6 = 1 \\ 3x_1 - 3x_2 + 9x_3 + 7x_4 + 9x_5 + 13x_6 = -1 \\ -x_1 + 2x_2 - 6x_3 - x_4 - 5x_5 - 12x_6 = 2 \\ x_1 + 5x_2 - 8x_3 - 2x_4 - 5x_5 - 5x_6 = 1 \\ 5x_1 - x_2 + x_3 + 10x_4 + 3x_5 - 6x_6 = -1 \\ -3x_1 - 6x_2 + 12x_3 + 5x_4 + 9x_5 + 7x_6 = 5 \end{cases}$$

(2)
$$\begin{cases} -3x_1 + x_2 + 11x_3 - 5x_4 + 2x_6 - 2x_7 + 2x_8 = 1 \\ 2x_1 + x_2 + 6x_3 - 2x_4 - x_5 - 2x_7 + 2x_8 = 2 \\ 4x_1 + 5x_2 + 3x_3 - 2x_4 + x_5 - 4x_6 - 8x_7 + 3x_8 = -1 \\ -x_2 - 4x_3 + 2x_4 + 2x_7 - x_8 = 3 \end{cases}$$

(3)
$$\begin{cases} 4x_1 + 4x_2 - 4x_3 + 2x_4 + x_5 - 3x_6 + 2x_7 = -13 \\ 7x_1 + 7x_2 - 7x_3 + x_6 + 6x_7 = 16 \\ -5x_1 + x_2 - x_3 - 4x_4 + 5x_5 - x_6 - x_7 = 23 \\ -x_1 - 5x_2 + 5x_3 + 4x_4 - 5x_5 - x_6 - 3x_7 = -35 \\ -x_1 - 2x_4 + 4x_5 - x_7 = 10 \end{cases}$$

PRÁCTICAS CON MATLAB. ÁLGEBRA
HOJA 2.- GRÁFICOS 2D

Comando básico `plot`

La función básica para representar funciones en dos dimensiones es `plot`. Después de ver cómo funciona este programa, a nadie le puede resultar extraño que los gráficos 2-D de MATLAB estén fundamentalmente orientados a la representación gráfica de vectores (y matrices). En el caso más sencillo los argumentos básicos de la función plot son vectores. Cuando una matriz aparezca como argumento, se considerará como un conjunto de vectores columna (en algunos casos también de vectores fila). En el siguiente cuadro presentamos algunos comandos básicos para comenzar.

`plot()`	crea un gráfico a partir de vectores y/o columnas de matrices, con escalas lineales sobre ambos ejes.
`title('título')`	añade un título al gráfico.
`xlabel('tal')`	añade una etiqueta al eje de abscisas. Con `xlabel off` desaparece.
`ylabel('cual')`	añade una etiqueta al eje de ordenadas. Con `ylabel off` desaparece.
`text(x,y,'texto')`	introduce 'texto' en el lugar especificado por las coordenadas x e y. Si x e y son vectores, el texto se repite por cada par de elementos. Si texto es también un vector de cadenas de texto de la misma dimensión, cada elemento se escribe en las coordenadas correspondientes.
`gtext('texto')`	introduce texto con ayuda del ratón: el cursor cambia de forma y se espera un clic para introducir el texto en esa posición.
`legend()`	define rótulos para las distintas líneas o ejes utilizados en la figura. Para más detalle, consultar el Help.
`grid`	activa la inclusión de una cuadrícula en el dibujo. Con `grid off` desaparece la cuadrícula.
`figure(n)`	hace que la ventana n pase a ser la ventana o figura activa. Si dicha ventana no existe, se crea una nueva ventana con el número n. La función `close` cierra la figura activa, mientras que `close(n)` cierra la ventana o figura número n.

MATLAB utiliza un tipo especial de ventanas para realizar las operaciones gráficas. Ciertos comandos abren una ventana nueva y otros dibujan sobre la ventana activa, bien sustituyendo lo que hubiera en ella, bien añadiendo nuevos elementos gráficos a un dibujo anterior. Para analizar los recursos gráficos que tiene el MATLAB, vamos a practicar tecleando las funciones más usuales.

Ejercicio 12. *Ejecutar las siguientes sentencias:*

```
>> X=(-pi:0.1:pi)'; plot(X,sin(X))
>> plot(X,sin(X),X,cos(X))
>> title('GRÁFICOS SENO Y COSENO'), figure(gcf)
>> xlabel('Eje x'), figure(gcf)
>> ylabel('Eje y'), figure(gcf)
>> figure (también figure(2))
>> plot(X,power(X,2),'m:.'), figure(gcf)
>> plot(X,power(X,2),'r.'), figure(gcf)
>> plot(X,power(X,2),'r.',X,sin(X),'bo'), figure(gcf)
>> close(2)
```

Para la función `plot` tenemos las siguientes opciones:

Símbolo	**Color**	**Símbolo**	**Marcadores (markers)**
y	yellow	.	puntos
m	magenta	o	círculos
c	cyan	x	marcas en x
r	red	+	marcas en +
g	green	*	marcas en *
b	blue	s	marcas cuadradas (square)
w	white	d	marcas en diamante (diamond)
k	black	^	triángulo apuntando arriba
		v	triángulo apuntando abajo
Símbolo	**Estilo de línea**	>	triángulo apuntando a la dcha
-	líneas continuas	<	triángulo apuntando a la izda
:	líneas a puntos	p	estrella de 5 puntas
-.	líneas a barra-punto	h	estrella se seis puntas
–	líneas a trazos		

La función plot vista anteriormente dibuja vectores. Si se quiere dibujar una función, antes de ser pasada a plot debe ser convertida en un vector de valores. Esto tiene algunos inconvenientes, por

ejemplo, el que "a priori" es difícil predecir en que zonas la función varía más rápidamente y habría por ello que reducir el espaciado entre los valores en el eje de abscisas. Este inconveniente lo soluciona el comando `fplot`. A continuación mostramos cómo puede utilizarse `plot` también con matrices como argumentos, la función `fplot` y el comando `subplot` para dibujar varias gráficas.

`plot(A)`	dibuja una línea por cada columna de A en ordenadas, frente al índice de los elementos en abscisas
`plot(x,A)`	dibuja las columnas (o filas) de A en ordenadas frente al vector x en abscisas. Las dimensiones de A y x deben ser coherentes: si la matriz A es cuadrada se dibujan las columnas, pero si no lo es y la dimensión de las filas coincide con la de x, se dibujan las filas
`fplot(f,[a,b],opciones)`	admite como primer argumento un nombre de función, como segundo el intervalo donde se quiere representar la función y como tercer argumento, el estilo del gráfico.
`subplot(m,n,i)`	divide la ventana gráfica en m particiones horizontales y n verticales, con objeto de representar múltiples gráficos en ella. Cada una de estas subventanas tiene sus propios ejes, aunque otras propiedades son comunes a toda la figura. El argumento i es la subdivisión que se convierte en activa.
`hold`	sirve para superponer nuevas curvas a un gráfico ya existente, sin destruirlo o sin abrir una nueva ventana. Se utilizan para ello los comandos `hold on` y `hold off`. El primero de ellos hace que los gráficos sucesivos respeten los que ya se han dibujado en la figura (es posible que haya que modificar la escala de los ejes); el comando hold off deshace el efecto de hold on.

Ejercicio 13. *Ejecutar las siguientes sentencias:*

```
>> X=(-pi/2:0.1:pi/2)';
>> A=[sin(X),cos(X),exp(X)];
>> figure
```

```
>> plot(X,A)

>> hold on

>> plot(X,power(X,2),'m.'), figure(gcf)

>> hold off;

>> f=inline('4*t^3-5*t^2+3')

>> fplot(f,[-20,20],'r'), figure(gcf)

>> [2,3,4]*[2,3,4] (¿Por qué da error?)

>> [2,3,4].*[2,3,4] (¿Cuál es el resultado si ponemos el punto?)

>> Y=sin(X); Z=cos(X); W=exp(-X*.1).*Y; V=Y.*Z;

>> figure

>> subplot(2,2,1), plot(X,Y), figure(gcf)

>> subplot(2,2,2), plot(X,Z), figure(gcf)

>> subplot(2,2,3), plot(X,W), figure(gcf)

>> subplot(2,2,4), plot(X,V), figure(gcf)
```

Veamos ahora algunas opciones para los ejes. El comando básico es el comando `axis`. Por defecto, MATLAB ajusta la escala de cada uno de los ejes de modo que varíe entre el mínimo y el máximo valor de los vectores a representar. Este es el llamado modo "auto", o modo automático.

`axis([xmin, xmax, ymin, ymax])`	define de modo explícito los valores máximo y mínimo según cada eje.
`axis('auto')`	devuelve el escalado de los ejes al valor por defecto o automático.
`v=axis`	devuelve un vector v con los valores [xmin, xmax, ymin,ymax]
`axis('ij')`	utiliza ejes de pantalla, con el origen en la esquina superior izda. y el eje j en dirección vertical descendente.
`axis('xy')`	utiliza ejes cartesianos normales, con el origen en la esquina inferior izda. y el eje y vertical ascendente.
`axis('auto x')`	utiliza el escalado automático sólo en dirección x.
`axis('auto xy')`	utiliza el escalado automático sólo en direcciones x, y.
`axis(axis)`	mantiene los ejes en sus actuales valores, de cara a posibles nuevas gráficas añadidas con hold on.

`axis('tight')`	establece los mismos límites para los ejes que para los datos, esto es, "ajusta" la gráfica al máximo.
`axis('equal')`	el escalado es igual en ambos ejes.
`axis('square')`	la ventana será cuadrada.
`axis('image')`	la ventana tendrá las proporciones de la imagen que se desea representar en ella.
`axis('normal')`	elimina las restricciones introducidas por 'equal' y 'square'.
`axis('off')`	elimina las etiquetas, los números y los ejes.
`axis('on')`	restituye las etiquetas, los números y los ejes.

Ejercicio 14. *Ejecutar las siguientes sentencias:*

```
>> X=(-4*pi:pi/20:4*pi)';
>> figure
>> plot(X,sin(X),'r',X,cos(X),'g')
>> title('Función seno(x) -en rojo- y función coseno(x) -en verde-')
>> xlabel('ángulo en radianes'), figure(gcf)
>> ylabel('valor de la función trigonométrica'), figure(gcf)
>> axis([-12,12,-1.5,1.5]), figure(gcf)
>> axis('equal'), figure(gcf)
>> axis('normal'), figure(gcf)
>> axis('square'), figure(gcf)
>> axis('off'), figure(gcf)
>> axis('on'), figure(gcf)
>> axis('on'), grid, figure(gcf)
```

Aunque disponemos de sentencias para modificar todas las opciones de los gráficos, en la práctica, puede resultar mucho más cómodo y rápido manipular el menú desplegado en la ventana gráfica.

Cómo dibujar una circunferencia en forma paramétrica

Sabemos que la ecuación implícita de una circunferencia de radio r en $\mathbb{R}^2$ viene dada por $x^2+y^2=r^2$. Para dibujar una circunferencia de radio r en MatLab es conveniente parametrizarla en la forma:

$$x = r\cos(t); \quad y = r\,\mathrm{sen}(t), \quad t \in [0, 2\pi].$$

Veamos un ejemplo que dibuja una circunferencia de radio 2.

```
>> t=0:0.01:2*pi; r=2;
>> x=r*cos(t); y=r*sin(t);
>> plot(x,y);
>> figure(gcf); comet(x,y);
```

Obsérvese que la última sentencia `comet` hace lo mismo que plot, pero generando una pequeña animación del trazo de la circunferencia. Además, da la sensación que se ha dibujado una elipse en lugar de una circunferencia. Esto es debido al escalado de los ejes, y se puede solucionar utilizando la misma escala para los dos ejes mediante la sentencia

```
>> axis('equal')
```

Ejercicio 15. *Realizar un script que, en un mismo gráfico y con un escalado igual de ejes, realice lo siguiente:*

(a) Dibuje en azul una circunferencia centrada en el punto $(1,1)$ *y de radio 1.*

(b) Dibuje en verde un cuadrado de vértices $\{(-1,1),(1,1),(1,-1),(-1,-1)\}$.

Scripts y programación en MatLab

Los archivos de funciones scripts hacen que MATLAB tenga capacidad de crecimiento. Se puede crear funciones específicas para un problema concreto, y, una vez programadas tendrán el mismo rango que las demás funciones del sistema. Para poder realizar una función, en primer lugar debemos, editarla (MATLAB trae consigo un editor incorporado) y guardarla como un fichero con extensión .m. El nombre del fichero **tiene que ser el mismo que el nombre de la función que hemos definido**.

Un aspecto importante es que las variables definidas dentro de una función son variables locales, en el sentido de que son inaccesibles desde otras partes del programa y en el que no interfieren con variables del mismo nombre definidas en otras funciones o partes del programa. Se puede decir que pertenecen al propio espacio de trabajo de la función y no son vistas desde otros espacios de trabajo. Un ejemplo de una función sencilla llamada "cuadrado" es la siguiente, la cual, si le pasamos como argumento un número a, devuelve su cuadrado ac:

```
function ac=cuadrado(a)
ac=a*a;
```

Presentemos algunos bucles y bifurcaciones necesarios para programar en MatLab. En su forma más simple, la sentencia **if** se escribe en la forma siguiente:

```
if condición
sentencias;
end
```

Existe también la bifurcación múltiple, en la que pueden concatenarse tantas condiciones como se desee, y que tiene la forma:

```
if condición1
sentencias bloque1;
elseif condición2
sentencias bloque2;
elseif condición3
sentencias bloque3;
else (opcion por defecto para cuando no se cumplan las condiciones 1,2,3)
sentencias bloque4;
end
```

donde la opción por defecto `else` puede ser omitida: si no está presente no se hace nada en caso de que no se cumpla ninguna de las condiciones que se han chequeado.

La sentencia `for` repite un conjunto de sentencias un número predeterminado de veces. La siguiente construcción ejecuta sentencias con valores de i de 1 a n, variando de uno en uno:

```
for i=1:n
sentencias;
end
```

o bien,

```
for i=vectorValores
sentencias;
end
```

donde `vectorValores` es un vector con los distintos valores que recorre la variable i. Se pueden también realizar bucles anidados utilizando varias sentencias `for`.

Ejercicio 16. *Crear una función script que se llame "plotfiguras" y que represente gráficamente (en un mismo gráfico) dos polígonos, el primero en azul y el segundo en verde. Los polígonos se pasarán como argumentos de la función "plotfiguras" en forma de una matriz. Ajustar que el escalado de los ejes sea el mismo y que la ventana emergente sea cuadrada.*

PRÁCTICAS CON MATLAB. ÁLGEBRA
HOJA 3.- TRANSFORMACIONES GEOMÉTRICAS LINEALES 2D

El objetivo de esta práctica es mostrar la interpretación geométrica de una transformación (aplicación) lineal invertible $T : \mathbb{R}^2 \to \mathbb{R}^2$ en el plano. Sabemos que una transformación (o endomorfismo) se identifica con una matriz cuadrada cuando trabajamos en coordenadas, por lo que trabajaremos directamente con las matrices de transformación.

Para realizar un estudio visual de cómo actúan las transformaciones geométricas lineales utilizaremos la función script auxiliar "plotfiguras" que viene definida por el script:

```
function grafico=plotfiguras(Figura1,Figura2)
hold off;
plot(Figura1(1,:),Figura1(2,:),'b');
hold on;
plot(Figura2(1,:),Figura2(2,:),'g');
axis('equal');
```

En segundo lugar definimos un cuadrado y un triángulo mediante matrices cuyas columnas contengan las coordenadas de los vértices.

```
>> cuadrado=[0 0;0 1; 1 1; 1 0; 0 0]';
```

```
>> triangulo=[0 0; 1/2 1; 1 0; 0 0]';
```

Transformaciones geométricas lineales elementales

Simetrías

Si fijamos una recta en el plano que pase por el origen, existe una transfomación lineal que reflecta cualquier vector respecto de esa recta. Las simetrías básicas son las siguientes:

Respecto del eje x: Vienen dadas por la matriz $T_1 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

```
>> T1=[1 0;0 -1]
```

```
>> plotfiguras(cuadrado,T1*cuadrado)
```

Respecto del eje y: Vienen dadas por la matriz $T_2 = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$

```
>> T2=[-1 0;0 1]
>> plotfiguras(cuadrado,T2*cuadrado)
```

Respecto del origen: Vienen dadas por la matriz $T_3 = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$

```
>> T3=[-1 0;0 -1]
>> plotfiguras(cuadrado,T3*cuadrado)
```

Respecto del eje y=x: Vienen dadas por la matriz $T_4 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

```
>> T4=[0 1;1 0];
>> plotfiguras(triangulo,T4*cuadrado)
>> plot([0 1],[0 1],'r');
```

Nota 2. *Obsérvese que $T_3 = T_1 \cdot T_2 = T_2 \cdot T_1$, esto se interpreta geométricamente afirmando que la simetría respecto del origen es la composición de una simetría respecto del eje x con una simetría respecto del eje y o viceversa.*

Escalados

Un *escalado* consiste, de manera informal, en dilatar o comprimir una figura. Los escalados elementales son los siguientes:

Escalado a lo largo del eje x: La matriz de la transformación es $T_5 = \begin{pmatrix} k & 0 \\ 0 & 1 \end{pmatrix}$. En el caso de que $0 < k < 1$, diremos que el escalado es una *compresión*, y si $1 < k$ diremos que es una *dilatación* en la dirección del eje x.

```
>> T5=[3/2 0;0 1]
>> plotfiguras(cuadrado,T5*cuadrado)
```

Escalado a lo largo del eje y: La matriz de la transformación es $T_6 = \begin{pmatrix} 1 & 0 \\ 0 & k \end{pmatrix}$. De la misma forma, en el caso de que $0 < k < 1$, diremos que el escalado es una *compresión*, y si $1 < k$ diremos que es una *dilatación* en la dirección del eje y.

```
>> T6=[1 0;0 1/2]
>> plotfiguras(cuadrado,T6*cuadrado)
```

Afilamientos

Un *afilamiento* es una transformación lineal que deja fijos todos los puntos sobre una recta r y los demás puntos son desplazados paralelamente a esta recta en una distancia proporcional a la distancia entre el punto y la recta r. Estudiemos a continuación los afinamientos elementales:

Afilamiento en la dirección del eje x: La matriz de la transformación es $T_7 = \begin{pmatrix} 1 & k \\ 0 & 1 \end{pmatrix}$. En este caso la recta de puntos fijos es el eje x:

```
>> T7=[1 2;0 1]
>> plotfiguras(cuadrado,T7*cuadrado)
```

Afilamiento en la dirección del eje y: La matriz de la transformación es $T_7 = \begin{pmatrix} 1 & 0 \\ k & 1 \end{pmatrix}$. En este caso la recta de puntos fijos es el eje y:

```
>> T8=[1 0;2 1]
>> plotfiguras(cuadrado,T8*cuadrado)
```

Observemos que cualquier *matriz elemental* que proviene de una transformación elemental se corresponde con una trasformación geométrica elemental o una composición de ellas. En efecto:

(1) Intercambiar dos filas: Se corresponde con una *simetría respecto al eje $y = x$.*

(2) Multiplicar una fila por un $\alpha \neq 0$: Si $\alpha > 0$ se corresponde con un *escalado*. Si $\alpha < 0$, entonces es una composición de una *simetría (respecto de un eje coordenado)* y un *escalado*.

(3) Sumar a una fila una combinación lineal de otra: Se corresponde con un *afilamiento*.

Es sabido por teoría, que cualquier transformación (matriz) invertible es producto de matrices elementales. Así pues, como todas las matrices elementales son composición de transformaciones geométricas elementales, podemos establecer el siguiente resultado.

Proposición 3. *Toda transformación (matriz) invertible es composición (producto) de transformaciones geométricas elementales.*

Otras Transformaciones geométricas lineales de interés

Rotaciones

La matriz de una transformación geométrica que se corresponde con una rotación de ángulo α viene dada por

$$R_\alpha = \begin{pmatrix} \cos\alpha & -\operatorname{sen}\alpha \\ \operatorname{sen}\alpha & \cos\alpha \end{pmatrix}, \quad \alpha \in \mathbb{R}.$$

Obsérvese que esta transformación es invertible, ya que $\det(R_\alpha) = \cos^2\alpha + \text{sen}^2\,\alpha = 1$.

Para definir de forma general esta matriz con MatLab podemos teclear el siguiente código:

```
>> syms alpha real;
>> R=[cos(alpha) -sin(alpha);sin(alpha) cos(alpha)]
```

Pongamos, por ejemplo, que queremos hacer una rotación de ángulo $\frac{\pi}{3}$, entonces tecleamos:

```
>> alpha=pi/3; R1=eval(R)
>> plotfiguras(cuadrado,R1*cuadrado)
```

Proyección ortogonal

Sea r una recta vectorial (pasando por el origen) y u un vector (columna) en la dirección de r. Entonces la matriz de la proyección ortogonal sobre r de cualquier vector de $\mathbb{R}^2$ viene dada por:

$$P_r = \frac{1}{\beta} uu^T, \qquad \beta = u^T u. \tag{1}$$

En particular, si $\beta = 1$ (u es un vector unitario), podemos calcular en MatLab la expresión general de la matriz asociada a la proyección ortogonal siguiendo los siguientes pasos:

```
>> syms alpha real
>> u=[cos(alpha) sin(alpha)]'
>> P=u*u'
```

Ejercicio 17. *Realizar un script que calcule:*

(a) La matriz asociada a la proyección ortogonal sobre la recta $y + 2x = 0$.

(b) La proyección del vector $u = (1, 1)$ *sobre esta recta.*

Hay una fórmula general que nos permite construir la matriz proyección ortogonal sobre cualquier subespacio vectorial de $\mathbb{R}^n \equiv \mathbb{R}^{n\times 1}$. Consideremos una matriz $M \in \mathbb{R}^{n\times r}$ tal que $\text{rg}(M) = r$ (rango máximo por columnas) y sea F el subespacio generado por las columnas de M (esto es, $F = Im(M)$), entonces **la matriz asociada a la proyección ortogonal sobre el subespacio** F es

$$P_F = M(M^T M)^{-1} M^T.$$

Obsérvese que en el caso particular de un subespacio de dimensión 1 (recta) se obtiene la fórmula (1).

Ejercicio 18. *Realizar una función script que se denomine "proyeccion", que tenga como argumento una matriz A cualquiera, y que devuelva la matriz proyección del subespacio generado por las columnas de A.*

Transformación de Householder

Una transformación de Householder en $\mathbb{R}^2$ es una transformación lineal que consiste en una simetría respecto a una recta cualquiera que pase por el origen. Sea u un vector unitario en $\mathbb{R}^2$, entonces la matriz de la simetría de Householder respecto de la recta r ortogonal a u viene dada por $H = I - 2uu^T$ siendo u un vector columna.

La expresión general de esta matriz podemos generarla con MatLab mediante:

```
>> syms alpha real
>> u=[cos(alpha) sin(alpha)]'
>> v=[-sin(alpha) cos(alpha)]'
>> H=eye(2)-2*u*u'
```

En particular, si tomamos $\alpha = \frac{\pi}{3}$ obtenemos:

```
>> alpha=pi/3; H1=eval(H); u1=eval(u); v1=eval(v);
>> plotfiguras(H1,H1*cuadrado)
>> compass([u1(1),v1(1)],[u1(2),v1(2)])
```

Ejercicio 19. *Realizar un script que devuelva la matriz de la transformación de Householder correspondiente a una simetría respecto de la recta* $y + x = 0$.

Nota 4. *La transformación de Householder se extiende directamente a* $\mathbb{R}^n$ *mediante la misma fórmula. En ese caso la interpretación geométrica es una simetría en* $\mathbb{R}^n$ *respecto del hiperplano ortogonal al vector* $u \in \mathbb{R}^n$.

PRÁCTICAS CON MATLAB. ÁLGEBRA
HOJA 4.- COORDENADAS HOMOGÉNEAS. TRANSFORMACIONES GEOMÉTRICAS 2D

El objetivo de esta práctica es conocer y utilizar las coordenadas homogéneas en dos dimensiones. En la práctica anterior estudiamos algunos ejemplos de transformaciones geométricas lineales, pero hay transformaciones geométricas importantes que no son lineales y que no se pueden representar por una matriz 2×2, como por ejemplo las traslaciones.

Coordenadas homogéneas

Sea $u \in \mathbb{R}^2$ un vector fijo. En geometría se entiende por *traslación* en la dirección de u a la aplicación definida por

$$\begin{aligned} T_u : \mathbb{R}^2 &\longrightarrow \mathbb{R}^2 \\ v &\rightsquigarrow u + v, \end{aligned}$$

esto es, si consideramos $\mathbb{R}^2$ como un espacio de puntos (x, y), la aplicación T_u desplaza cada punto en la dirección de u a una distancia igual al módulo (a la norma) de ese mismo u.

En coordenadas, si $u = \begin{pmatrix} a \\ b \end{pmatrix}$, $v = \begin{pmatrix} x \\ y \end{pmatrix}$ y $T_u(v) = \begin{pmatrix} x' \\ y' \end{pmatrix}$ entonces

$$\begin{cases} x' &= x + a \\ y' &= y + b. \end{cases} \tag{2}$$

Es fácil probar que esta aplicación no es lineal y, por tanto, no se puede expresar en coordenadas mediante una matriz 2×2. En la práctica anterior se mostró que una de las ventajas de entender las transformaciones lineales como matrices es que la composición de aplicaciones lineales se interpreta como un producto de matrices. Este hecho supone una ventaja clara a nivel computacional, y es que para componer dos transformaciones geométricas lineales basta con multiplicar sus matrices asociadas. La pregunta que nos hacemos es ¿podemos representar de alguna manera también las traslaciones como matrices? La respuesta es afirmativa, pero debemos incrementar la dimensión de las matrices a 3×3 y necesitamos introducir lo que se conoce como *coordenadas homogéneas* y que describimos de manera informal a continuación:

- Un punto (x, y) en $\mathbb{R}^2$ se representa en coordenadas homogéneas de la forma $(hx, hy, h) \in \mathbb{R}^3$ para cualquier $h \in \mathbb{R}$.
- Esto significa que un mismo punto tiene **infinitas** representaciones en coordenadas homogéneas.
- Ejemplo: el punto $(4, 6)$ puede expresarse como $(4, 6, 1), (8, 12, 2), (2, 3, 1/2), (8/3, 4, 2/3)$, etc.
- Lo habitual es tomar $h = 1$, con lo que el punto (x, y) pasa a $(x, y, 1)$.

- Recíprocamente, un punto (a, b, c) en $\mathbb{R}^3$ en coordenadas homogéneas representa al punto $(a/c, b/c)$.

Para comodidad, comenzamos definiendo dos funciones inline (una inversa de la otra) para pasar a coordenadas homogéneas y viceversa.

Paso a coordenadas homogéneas: Definimos una función script *homo* : $\mathbb{R}^{2\times k} \to \mathbb{R}^{3\times k}$ que pasa una matriz de k vectores en $\mathbb{R}^2$ (vista como vectores columnas) a una matriz de k vectores en coordenadas homógeneas (también por columnas) mediante el siguiente código:

```
function Ch=homo(C)
n=size(C,2);        %número de columnas de C
vunos=ones(1,n);    %construye un vector fila de n unos
Ch=[C;vunos];
```

Inversa de *homo*: Definimos una función script *invhomo* : $\mathbb{R}^{3\times k} \to \mathbb{R}^{2\times k}$ que pasa una matriz de k vectores en coordenadas homogéneas a una matriz de k vectores en coordenadas normales:

```
function C=invhomo(Ch)
fila3=Ch(3,:);   %Obtiene la última fila de Ch
D=diag(fila3);   %Construye una matriz diagonal con la fila3
C=Ch*inv(D);     %Divide cada columna de Ch entre la última coordenada
C(3,:)=[];                  %Borra la última fila de unos.
```

Transformaciones geométricas

Traslaciones

Mostremos cómo se puede escribir en forma matricial una traslación haciendo uso de las ecuaciones (2) y de las coordenadas homogéneas:

$$\begin{cases} x' &= x + a \\ y' &= y + b. \end{cases} \quad \Leftrightarrow \quad \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Ejemplo 5. *Hallar la matriz de la traslación asociada al vector $u = (1, 1)$ y representar gráficamente el resultado aplicado a un hexágono de vértices*

$$\left\{(-1,0), (-1/2, \sqrt(3)/2), (1/2, \sqrt(3)/2), (1,0), (1/2, -\sqrt(3)/2), (-1/2, -\sqrt(3)/2)\right\}.$$

Solución.- Definimos el hexágono por medio de una matriz cuyas columnas son los vértices y definimos la matriz de traslación asociada al vector $u = (1, 1)$:

```
>> hex=[-1 0;-1/2 sqrt(3)/2;1/2 sqrt(3)/2;1 0;1/2 -sqrt(3)/2;-1/2 -sqrt(3)/2;
   -1 0]'

>> Tu=[1 0 1;0 1 1;0 0 1]
```

A continuación, pasamos los vértices del hexágono a coordenadas homogéneas mediante la función H, aplicamos la traslación multiplicando por Tu y pasamos de nuevo a coordenadas cartesianas usuales:

```
>> Hhex=H(hex)

>> TuHhex=Tu*Hhex

>> Tuhex=Hinv(TuHhex);
```

Finalmente representamos el resultado mediante la función script "plotfiguras.m":

```
>> plotfiguras(hex,Tuhex)
```

Transformaciones geométricas afines

Entendemos por una *transformación (geométrica) afín* el resultado de componer una traslación y una transformación lineal. En el caso particular en que la transformación lineal es una rotación se denomina *movimiento rígido*, y tiene importantes aplicaciones en robótica.

En coordenadas cartesianas usuales sabemos que una transformación geométrica lineal se representa por una matriz regular $T = \begin{pmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{pmatrix}$. Si pasamos a coordenadas homogéneas es fácil comprobar que

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \quad \Leftrightarrow \quad \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Por lo tanto la matriz asociada a la transformación lineal T en coordenadas homogéneas es

$$\tilde{T} = \begin{pmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Si consideramos una traslación T_u genérica, entonces podemos calcular la matriz A de la transformación afín asociada a la transformación lineal $\tilde{T}$ y la traslación T_u multiplicando (componiendo) ambas; así pues, la matriz general de una transformación geométrica afín es:

$$A = T_u \cdot \tilde{T} = \begin{pmatrix} t_{11} & t_{12} & a \\ t_{21} & t_{22} & b \\ 0 & 0 & 1 \end{pmatrix}.$$

Ejercicio 20. *Realizar un script que halle la matriz de la transformación afín (en coordenadas homogéneas) que se obtiene de una rotación de un ángulo de 60 grados, seguido de una traslación asociada al vector* $u = (1, 1)$. *Además, representar gráficamente el resultado aplicado al hexágono del Ejemplo 5.*

Ejercicio 21. *Realizar un script que:*

(a) Halle la matriz de la transformación afín (en coordenadas homogéneas) que se obtiene de realizar una simetría respecto de la recta $x + y = 2$.
Indicación: *Obsérvese que una simetría respecto de una recta que no pasa por el origen es la composición de una cierta simetría de una recta que pasa por el origen y ciertas traslaciones.*

(b) Represente gráficamente la circunferencia unidad centrada en el origen y su imagen mediante la transformación afín del apartado (a).

PRÁCTICAS CON MATLAB. ÁLGEBRA
HOJA 5.- DESCOMPOSICIÓN EN VALORES SINGULARES DE UNA MATRIZ

Valores singulares de una matriz cuadrada

En este epígrafe mostraremos la interpretación geométrica de los valores singulares de una matriz.

Para responder a la pregunta ¿qué son los valores singulares? podemos formular otra pregunta: ¿cuál es la imagen de la circunferencia unidad (la de centro el origen de coordenadas y radio 1) por una aplicación lineal?. Para aquellos que no lo saben vamos realizar un sencillo experimento con la ayuda de MatLab que nos ayudará a dar una respuesta. En primer lugar definimos una matriz A y una circunferencia de radio unidad, que denotamos por C:

```
>> A=[1/4 3/4;1 1/2]
>> t=0:0.01:2*pi;
>> x=cos(t); y=sin(t); C=[x;y];
```

Hallamos la imagen de la circunferencia por la aplicación lineal A y la dibujamos en verde, para lo cual utilizamos la función `plotfiguras` que definimos en la práctica anterior:

```
>> plotfiguras(C,A*C);
```

Se puede observar que la imagen de una circunferencia a través de la aplicación A es una elipse. Denotemos por s_1 y s_2 a las distancias de los semiejes de la elipse obtenida mediante la transformación lineal A. En nuestro caso particular, el teorema de descomposición en valores singulares (SVD) se puede enunciar geométricamente de la siguiente manera:

Teorema 6. *Para la matriz $A \in \mathbb{R}^{2\times 2}$ existe una base ortonormal v_1, v_2 tal que los vectores imagen Av_1 y Av_2 de esta base son justamente los semiejes de la elipse, en otras palabras,*

$$Av_1 = s_1 u_1, \qquad Av_2 = s_2 u_2, \tag{3}$$

para ciertos vectores unitarios y ortogonales u_1 y u_2 situados en la dirección de los ejes.

Obsérvese que las ecuaciones (3) son equivalentes a la ecuación que tenemos en teoría, en efecto:

$$\left(\begin{array}{c|c} Av_1 & Av_2 \end{array} \right) = \left(\begin{array}{c|c} s_1 u_1 & s_2 u_2 \end{array} \right) \Leftrightarrow$$

$$A \cdot \left(\begin{array}{c|c} v_1 & v_2 \end{array} \right) = \left(\begin{array}{c|c} u_1 & u_2 \end{array} \right) \cdot \begin{pmatrix} s_1 & 0 \\ 0 & s_2 \end{pmatrix} \Leftrightarrow$$

$$A \cdot V = U \cdot S \quad \Leftrightarrow \quad A = U \cdot S \cdot V^T,$$

que es la ecuación de descomposición de una matriz A en valores singulares.

Mostremos el teorema con MatLab. Hallamos ahora la descomposición en valores singulares de A, para ello utilizamos la función `svd` de la siguiente manera:

```
>> [U,S,V]=svd(A)
```

Las matrices U y V son matrices ortogonales y la matriz S es una matriz diagonal verificando $S = U^T AV$. A continuación graficamos los vectores v_1, v_2, s_1u_1 y s_2u_2:

```
>> E=A*V;
>> compass(V(1,:),V(2,:),'b')
>> compass(E(1,:),E(2,:),'r')
```

Concluimos que a cada matriz le podemos asociar sus valores singulares, pero puede haber, y en general hay, muchas matrices que produzcan la misma elipse como imagen de la circunferencia unidad. Todas ellas tendrán los mismos valores singulares. En ese caso se dice que son *ortogonalmente equivalentes*. En el caso de matrices con valores complejos se suelen denominar matrices *unitariamente equivalentes*, pero nosotros sólo trataremos el caso real.

Ejercicio 22. *Realizar el procedimiento descrito anteriormente para la matriz* $A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ *y comentar los resultados obtenidos.*

Norma matricial

Los valores singulares pueden caracterizarse también algebraicamente. En efecto, si s_1 es el mayor valor singular, su significado geométrico es la longitud del mayor de los semiejes de la elipse en que se convierte la circunferencia unidad por una transformación lineal. Ahora bien, en la circunferencia unidad están representados todos los vectores de longitud 1 del plano. Es decir, la circunferencia unidad es el lugar geométrico de los vectores de longitud, o norma euclídea, igual a 1. Simbólicamente

$$\text{Circunferencia unidad} = \left\{ v \in \mathbb{R}^2 : \|v\|_2 = 1 \right\}.$$

Entre todos estos vectores hay uno, v_1, tal que la longitud de Av_1 es el semieje mayor de la elipse. Y precisamente el hecho de que sea la longitud del semieje mayor significa, además, que los transformados de todos los demás vectores de la circunferencia unidad por A nunca tienen longitud mayor que s_1. Esto lo podemos escribir de la siguiente forma:

$$s_1 = \max_{\|v\|_2=1} \|Av\|_2.$$

Esta caracterización proporciona la condiciones para establecer una norma que se demuestran sin dificultad a partir de las propiedades del máx:

(1) Para cualquier matriz A, $s_1(A) \geq 0$ y $s_1(A) = 0$ si y solo si $A = 0$.

(2) Para cualquier matriz A y cualquier número α, $s_1(\alpha A) = |\alpha| s_1(A)$.

(3) Para cualesquiera matrices A y B, $s_1(A + B) \leq s_1(A) + s_1(B)$.

Las normas sirven para medir lo grande o pequeña que son las matrices y la distancia entre ellas. En otras palabras, el mayor valor singular de una matriz es una norma (conocida con el nombre de *norma de Hilbert, espectral o de operador*) y nos sirve para tener una idea de lo grande o pequeña que es una matriz, así como para establecer una distancia entre dos matrices que nos proporciona una medida de cuán "diferentes" son.

De la misma manera podemos definir

$$s_2 = \min_{\|v\|_2} \|Av\|_2,$$

pero s_2 no es una norma matricial aunque sí que tiene un signicado algebraico: lo grande o pequeño que sea este número nos da una idea de si las columnas de A son "muy linealmente independiente" o "muy poco linealmente independientes". En otras palabras, s_2 nos da una idea de lo cerca o lejos que está A de ser singular.

Ejercicio 23. *Analizar geométricamente la descomposición en valores singulares de las matrices:*

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1{,}1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}.$$

Aplicación: Tratamiento de imágenes

Imágenes en blanco y negro

En esta última parte, vamos a mostrar una aplicación sencilla de la descomposición en valores singulares para el procesamiento de imágenes digitales. En concreto, vamos a aprender un método de compresión de una foto en blanco y negro (y más tarde en color) basada en la descomposición SVD.

Desde el punto de vista matricial, lo que nos interesa de una imagen digital es que ésta se almacena en el ordenador como una matriz de "pixels", y cada entrada de la matriz (pixel) es un número que se corresponde con un tono de gris. Para obtener M tenemos que utilizar la función `imread` pasando como argumento el nombre de la imagen con su extensión, esto es:

```
>> M=imread('niebla.jpg');
>> size(M)
```

La matriz M de tamaño 492×648 contiene los datos necesarios para obtener la imagen de la foto. Se necesitan entonces 318816 números en punto flotante para conseguir la imagen. Si queremos recuperar la imagen a partir de la matriz utilizamos la siguientes sentencias:

```
>> imagesc(M)

>> colormap(gray)
```

¿Cómo podemos utilizar los valores singulares para obtener esta misma imagen con menos datos? La idea es la siguiente: De acuerdo con el Teorema SVD, exiten matrices ortogonales U y V tales que $M = U\Sigma V^T$, siendo U una matriz de tamaño 492×492, V de tamaño 648×648 y ambas ortogonales.

Lo mejor para mostrar cómo funciona el método es con matrices de dimensión más baja. Supongamos que M es 4×3, entonces:

$$\begin{aligned} M &= \left(\begin{array}{c|c|c|c} u_1 & u_2 & u_3 & u_4 \end{array} \right) \begin{pmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \\ 0 & 0 & 0 \end{pmatrix} \left(\begin{array}{c} v_1^T \\ \hline v_2^T \\ \hline v_3^T \end{array} \right) \\ &= \left(\begin{array}{c|c|c} u_1 & u_2 & u_3 \end{array} \right) \begin{pmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{pmatrix} \left(\begin{array}{c} v_1^T \\ \hline v_2^T \\ \hline v_3^T \end{array} \right) \\ &= \left(\begin{array}{c|c|c} s_1u_1 & s_2u_2 & s_3u_3 \end{array} \right) \left(\begin{array}{c} v_1^T \\ \hline v_2^T \\ \hline v_3^T \end{array} \right) \\ &= s_1u_1v_1^T + s_2u_2v_2^T + s_3u_3v_3^T. \end{aligned}$$

Las matrices $M_i = s_iu_iv_i^T$ se denominan *i-ésimo modo* de M. Así, $s_1u_1v_1^T$ es el primer modo de M, $s_2u_2v_2^T$ el segundo modo, etc, y obtenemos que

$$M = M_1 + M_2 + \cdots + M_r, \quad r = \mathrm{rg}(A).$$

El hecho importante para la compresión de datos es que en la suma anterior hay modos que son más dominantes que otros, esto es, "codifican" la información relevante o esencial de la matriz M. Concretamente los modos más dominantes son los asociados a los valores singulares mayores. Así pues podemos considerar para todo $1 \leq k \leq r$ las matrices

$$M(k) = M_1 + M_2 + \cdots + M_k,$$

y cuanto mayor sea k, mas "cerca" (respecto a la distancia definida en el epígrafe anterior) estará la matriz $M(k)$ de la matriz original M. Es claro que el número de datos para obtener $M(k)$ es inferior al de M a partir de un cierto valor crítico de k.

Realicemos el procedimiento utilizando MatLab aplicado a la imagen en blanco y negro que tenemos. En primer lugar calculamos la descomposición en valores singulares y, a continuación para un cierto k, calculamos la matriz formada por los k primeros modos:

```
>> [U,S,V]=svd(double(M));
```

```
>> k=10; Mk = U(:,1:k)*S(1:k,1:k)*V(:,1:k)';
>> Mk=uint8(Mk);
```

Finalmente, si deseamos mostrar la imagen codificada en la matriz Mk basta con teclear:

```
>> figure
>> imagesc(Mk)
>> colormap(gray)
```

y obtenemos una nueva imagen con una calidad inferior a la original. También podemos exportarla a un fichero mediante la sentencia `imwrite`, donde en el primer argumento se indica la matriz y en el segundo la ruta del fichero donde queremos exportar la imagen:

```
>> imwrite(Mk,'nieblak.jpg')
```

Obsérvese que la matriz Mk tiene la misma dimensión que M, entonces, ¿cómo hemos comprimido la información? En realidad la información de Mk queda codificada conociendo solamente los vectores u_i, v_i y en los escalares s_i, concretamente, el número de datos necesarios para codificar las entradas de la matriz Mk es:

$$\text{nº de datos} = m \times k + k \times n + k = (m + n + 1)k,$$

siendo $M \in \mathbb{R}^{m \times n}$.

Ejercicio 24. *Se pide:*

(a) Calcular el número de datos que tiene M y el número de datos que utilizamos para comprimir la imagen "niebla.jpg" cuando $k = 10$.

(b) Realizar el mismo procedimiento para valores de k mayores hasta obtener una imagen con una calidad satisfactoria.

(c) Hallar el valor de k a partir del cuál no ahorras memoria.

Ejercicio 25. *Realizar un script que automatice el proceso anterior de compresión de imágenes digitales en blanco y negro.*

Imágenes en color

Tratemos ahora imágenes en color. En primer lugar debemos conocer cómo está codificada matricialmente la información de una imagen en color, para ello tecleamos:

```
>> M=imread('cat.png');
```

```
>> size(M)
```

Podemos apreciar que devuelve un array de dimensión $373 \times 293 \times 3$, que viene a ser como tres matrices de dimensión 373×293, que se pueden obtener explícitamente como:

```
>> M1=M(:,:,1);

>> M2=M(:,:,2);

>> M3=M(:,:,3);
```

En este caso, a cada pixel le corresponde un color que viene determinado por tres números indicando una mezcla de tres tonalidades en el sistema RGB (rojo, verde y azul). Así por ejemplo, el color del pixel $(1,1)$ viene determinado por los dígitos de la entrada 11 de cada una de las tres matrices $M1, M2$ y $M3$. El procedimiento de compresión, en este caso, consiste en aplicar el algoritmo utilizado para imágenes en blanco y negro a cada una de estas tres matrices y finalmente construir la matriz Mk que se obtiene juntando de nuevo las tres matrices resultantes, que llamaremos Mk1, Mk2 y Mk3, en un array.

Ejercicio 26. *Elaborar un script que realice una compresión de la imagen digital 'cat.pn' para* $k = 10, 20, 50$ *y que muestre los resultados.*