Cálculo con MatLab

Curso 2010/2011

Práctica 1

Introducción

Las páginas siguientes constituyen una somera introducción a **MatLab**, con el fin concreto para el que se han elaborado: utilizar este programa como apoyo a las clases de **Cálculo** de primer curso del grado de Informática de la UDC.

MatLab es un paquete de software, comercializado por Math. Works Inc., ampliamente difundido en el campo de la docencia e investigación en numerosas universidades, porque es un programa --de alto nivel-- de cálculo científico, fácil de utilizar y con gran potencial desde el punto de vista computacional y gráfico.

El nombre **MatLab** viene de **Mat**rix **Lab**oratory puesto que **MatLab** fue ideado para proporcionar un fácil manejo de matrices. De hecho, es fundamentalmente un programa de cálculo matricial. No obstante, **MatLab** es también una potente herramienta tanto para resolver problemas elementales (como caso particular de las matrices, trabaja con vectores y, de nuevo como caso particular de éstos, con números –reales y complejos--, así como con otras estructuras), como otros de gran complejidad, usándolo como lenguaje de programación.

Sin embargo, el propósito de estas notas es mucho más limitado. Para un estudio más profundo remitimos a la amplia bibliografía existente y a la magnífica ayuda de **MatLab**.

No obstante, gracias a las funciones predefinidas en **MatLab**, se pueden calcular de forma sencilla límites, derivadas y/o integrales de funciones, se pueden resolver ecuaciones (numéricas y/o diferenciales), representar gráficamente curvas planas y superficies, calcular sumas de series, ... todo ello ejecutando sencillas órdenes.

En lo que sigue, se introducirán algunos de los comandos que necesitaremos a lo largo del curso de **Cálculo**, acompañados de ejemplos de aplicación.

Estos apuntes están redactados de modo que sean auto-contenidos: para que puedas desenvolverte autónomamente, si sigues las sugerencias incluidas en los mismos. Por ello, es aconsejable practicar todos los ejemplos y realizar los ejercicios que se proponen, pues la forma de aprender a utilizar cualquier programa es manejándolo e intentando saber cómo hacer aquello que se precisa.

1. Primer contacto con MatLab.

El programa está instalado bajo plataforma Windows. Para acceder al programa es necesario conectarse al equipo como usuario. Para ello en la pantalla de entrada debes teclear

usuario@windows.cc.fic.udc.es

donde, en lugar de la palabra usuario, debes introducir tu *login*. El sistema te pedirá la clave personal (*password*). Una vez hecho eso la pantalla que aparecerá es la habitual del entorno Windows, con el escritorio en el que figuran varios iconos de otras tantas aplicaciones. Si el icono de MatLab no figura en el escritorio deberás acceder al programa, como es habitual en Windows, a través del botón Inicio situado en la esquina inferior izquierda de la pantalla.

La versión con la que trabajaremos en esta asignatura es R2010b, también llamada MatLab 7.11.0.584 (de 16 de agosto de 2010).

Cuando se accede al programa haciendo doble clic con el ratón en el icono que aparece en el escritorio (o buscando el programa en el botón Inicio), la pantalla que se muestra ante nosotros consta de tres subventanas. Si la apariencia en tu equipo no es la que se indica aquí, basta acceder en el menú del programa a Desktop y escoger la opción Desktop Layout y a continuación Default.

La primera subventana en la que nos fijaremos, porque es en donde trabajaremos habitualmente, es la situada en la parte central de la pantalla: es la ***ventana de comandos***, (command window), a su derecha se hallan la ventana ***historial de comandos*** (command history) que se encuentra en la mitad inferior y la ventana ***área de trabajo*** (Workspace). La ventana ***directorio actual*** o directorio activo (current folder) se encuentra a la izquierda de la ventana de comandos.

En la esquina inferior de la pantalla está el botón Start del que se despliega un menú con diversas opciones del programa.

Es en la **ventana de comandos** en donde escribiremos las órdenes que queremos ejecutar. Al iniciar una sesión de trabajo con MatLab observamos que el cursor parpadea a la derecha del **indicativo** del programa (prompt) que es el símbolo >>

Las órdenes las escribiremos a la derecha de este indicativo, es decir, en donde parpadea el cursor.

(A lo largo de este texto, en las órdenes que se incluyan como ejemplo, aparecerá el indicativo del programa >> que tú verás en pantalla, por lo que, obviamente, no lo debes teclear.)

Para ejecutar una instrucción en la ventana de comandos, una vez que se haya escrito correctamente, basta pulsar la tecla **Intro** (es el retorno de carro). El programa responde en la línea siguiente y el resultado lo guarda en una variable con el nombre **ans** (proviene de *answer*). Por ejemplo:

```
>> 2+3

ans =

      5

>>
```

Antes de continuar practicando con **MatLab**, escribiremos una orden que nos garantizará que nuestra sesión de trabajo se grabará en un archivo de texto; así, no sólo se grabarán las órdenes que introduzcamos, también lo harán las respuestas del programa. El nombre que le daremos a este archivo es `Primera_sesión` y la orden es

```
>> diary Primera_sesion
```

Al final de la sesión de trabajo y, antes de salir de **MatLab**, bastará teclear (¡no lo hagas ahora!)

```
>> diary off
```

y con ello se cerrará el archivo, guardando nuestro trabajo.

Este archivo que aparecerá en la subventana de directorio actual (`current holder`) se puede abrir con un editor de textos, lo que nos permite incluir en él comentarios, por ejemplo. Para ver su contenido en la ventana de comandos (cuando queramos acceder a él, con posterioridad a su grabación), bastará teclear

```
>> type Primera_sesion
```

Por otra parte, para salir del programa y cerrar una sesión de trabajo --la hayamos grabado o no--, basta teclear en la ventana de comandos (¡no lo hagas ahora!) **`quit`** o bien en el menú elegir **File** y escoger la última opción **Exit MatLab**, o simplemente cerrar la ventana del programa.

Aún cuando no grabemos el trabajo de una sesión, los comandos utilizados se guardan en la ventana `command history`, por lo que son recuperables cuando volvamos a conectarnos a **MatLab**. Para recuperarlos basta hacer un doble clic sobre aquellos que nos interesen. Al situarnos sobre uno de esos comandos, si utilizamos el botón derecho del ratón se abren opciones que actúan sobre ese comando.

Pero, volvamos a la ventana de comandos para seguir ejecutando órdenes.

Es posible ejecutar una orden sin que se muestre el resultado en pantalla, para ello la orden debe finalizar con un punto y coma (;). Por ejemplo, a la orden

```
>> y = 3*7;
```

el programa responde con una línea en la que sólo aparece el indicativo del programa. ¿Cómo saber entonces si el programa ha efectuado el producto pedido? Como al producto le hemos llamado y, basta preguntar por el valor de y

```
>> y
```

y el programa entonces muestra en pantalla:

```
y =
   21

>>
```

También se pueden incluir comentarios en las líneas de ejecución de la ventana de comandos, con el fin de hacer más legible nuestro trabajo. Para ello, los comentarios deben ir precedidos por el símbolo % pues MatLab ignora todo lo que se escribe a la derecha de %.

Por ejemplo,

```
>>  B = 15 , H = 3  % B denota la base y H la altura
```

Como se puede observar en la línea anterior, se pueden introducir varias órdenes en una misma línea, separadas por comas (o puntos y comas).

Si una orden ocupa más de una línea basta escribir tres puntos seguidos y continuar su escritura en la siguiente línea. Por ejemplo,

```
>> 30/5+12/6+15/3+125/5+40/8+36/2+45/9 ...

-25+24/12
```

La respuesta es:

```
ans =

     43

>>
```

Para desplazarnos por la ventana de comandos podemos emplear las flechas de izquierda y derecha, para movernos dentro de una misma línea. Además, como sólo está activa la última línea, se pueden utilizar las flechas (hacia arriba o hacia abajo) para recuperar líneas ya escritas, bien con la intención de modificarlas si contienen errores de sintaxis o bien para ejecutarlas

de nuevo.

Por otra parte, disponemos de las combinaciones de teclas **Ctrl+D** para borrar un carácter situado a la derecha del cursor, **Ctrl+E**, para ir al final de la línea en la que está situado el cursor (también se puede utilizar la tecla **Fin**) y **Ctrl+A** para ir al principio de la misma (o con la tecla **Inicio**).

Para dejar en blanco la ventana de comandos, eliminando todas las órdenes introducidas en la sesión de trabajo, basta teclear `clc` (proviene de *clear console*: limpiar la consola). Para dejar la pantalla en blanco (sin borrar lo anterior) teclearemos `home`. En ambos casos se puede observar que todos los comandos que desaparecen de la `command window` siguen estando en la `command history`.

2. Ayuda de MatLab

Ya hemos visto en el apartado anterior que en la esquina inferior de la pantalla se sitúa **Start**, botón similar al botón **Inicio** de Windows. Una de las opciones del menú contextual que se abre en ese botón es **help** que nos introduce en la ayuda de **Matlab**. Otra forma alternativa es escoger en el menú la última opción.

También desde la ventana de comandos se puede acceder a la ayuda de dos formas: bien directamente desde el enlace que aparece en la línea anterior a la primera línea de comandos, bien tecleando simplemente la orden

```
>> help
```

y observaremos cómo aparecen enlaces desde los que podremos acceder a una información más específica.

Si deseamos obtener información sobre alguna de las funciones definidas por **MatLab**, la orden debe ir acompañada por el nombre de la función de la que queremos obtener información. Es decir, la sintaxis

```
>> help nombre
```

proporciona información sobre la entidad *nombre*. Por ejemplo

```
>> help plot
```

proporciona información sobre la orden **plot**. Esa información la descarga en la ventana de comandos y para verla de forma completa hay que desplazarse verticalmente por esa ventana.

Otra forma de obtener información sobre **plot** es teclear:

```
>> helpwin plot
```

Ahora la información sobre **plot** aparece en una ventana, independiente de la ventana de comandos, por la que se puede navegar.

Para obtener ayuda sobre un comando también se puede resaltar el nombre del comando una vez escrito en la ventana de comandos y utilizar después el botón derecho del ratón. Además, si tecleamos las primeras letras del nombre de un comando y pulsamos la tecla del tabulador aparece un menú de opciones para completar el comando buscado. Pruébalo tecleando

```
>> su
```

y, a continuación, pulsa la tecla del tabulador para ver las opciones disponibles: **sub2ind**, **subaxes**, **subclust**, etc.

En este apartado dedicado a la ayuda de **MatLab**, no podemos olvidarnos de mencionar las numerosas *demos* que posee. Para acceder a ellas elegiremos en el menú contextual de **help** o bien eligiendo en el botón **Start** la última opción, o bien, a través de la ventana de comandos en el enlace que aparece en la parte superior. Otra forma es --desde el navegador de la ayuda obtenida al invocarla en la ventana de comandos-- elegir la pestaña **Demos** en el menú de la ayuda. Navega por las distintas demos para acostumbrarte a manejar el programa.

El programa con el que trabajamos posee órdenes que están a nuestra disposición en el momento de abrirlo. Sin embargo, otras órdenes son específicas para ciertos campos de la ciencia que pueden (o no) venir incluidas en el paquete instalado. En particular, **MatLab** posee comandos específicos para aplicar, entre otras disciplinas, a Estadística o Cálculo diferencial e integral, por ejemplo; así como en áreas de la técnica como Redes de Neuronas, Bioinformática, Procesamiento de la señal, Lógica difusa, etc. Esos comandos, que guardan relación temática, se reúnen en lo que se conoce como los **toolboxes** (literalmente cajas de herramientas) o librerías especializadas en ciertas parcelas científicas. En las prácticas de **Cálculo** usaremos comandos del toolbox de Cálculo simbólico (**Symbolic Math Toolbox**).

Ejercicio 1

Navega por la ayuda en busca de la librería **Symbolic Math** *y busca información sobre un comando llamado* **diff**. *Navega por las demos y encuentra en* **Contents** *la demo:* **Vibrating logo**

3. Usando MatLab como una calculadora

Como ya se ha visto, **MatLab** para la suma emplea el signo más (**+**), para multiplicar usa el asterisco (*****), para la resta utiliza el signo menos (**-**) y para el cociente la barra de fracción (**/**). Para la potenciación emplea el circunflejo (**^**).

El orden de prioridad de las operaciones es el mismo que en las calculadoras: las expresiones se evalúan de izquierda a derecha; la potencia tiene el orden de prioridad más alto, seguido del producto y división (ambas tienen la misma prioridad) y por último están la suma y la resta (con igual prioridad entre ellas). Si se desea alterar este orden se deben introducir paréntesis y, en ese caso, el programa realiza los cálculos empezando desde el paréntesis más interno al más externo. Veamos algunos ejemplos, con sus respuestas correspondientes:

```
>> 8+4/2-1

ans =
        9

>>(8+4)/2-1

ans =
        5

>>((8+4)*(2-6)-10)/(3-1)

ans =
      -29

>>3*4^2-1

ans =
      47

>>(3*4)^2-1

ans =
       143
>>
```

Ejercicio 2

Calcula: a) $5^2 + 7 - \frac{26}{2}$

b) $\frac{2^3 - 7 + 11}{3 \cdot 2^2}$

Si observamos las respuestas a los cálculos realizados, vemos que, hasta el momento, todas son valores enteros y el resultado de los cálculos es correcto (exacto).

A diferencia de otros programas de cálculo simbólico (como **Maple**, por ejemplo), cuando el resultado de un cálculo no es un número entero de menos de 10 cifras, **MatLab**, en su modo de trabajo por defecto, no responde con el

valor racional del cálculo sino que responde, en general, con una aproximación en notación decimal. Por ejemplo, si ejecutamos la orden

```
>> 1+2/3
```

la respuesta que cabría esperar es `5/3`, sin embargo es:

```
ans =

       1.6667

>>
```

Esto sólo significa que, en su modo de trabajo por defecto, **MatLab** proporciona, en principio, una aproximación con un número finito de decimales (*precisión finita*).

Como veremos más adelante, existe en **MatLab** un *formato* en el que se puede trabajar con el resultado expresado en forma de fracción (`5/3` en el ejemplo anterior), en lugar de la aproximación decimal obtenida.

Ahora bien, debido a que las máquinas poseen recursos limitados, no pueden representar de modo exacto el infinito conjunto de los números reales: se dice que las máquinas trabajan en *aritmética finita*. De hecho, sólo manejan con exactitud un conjunto finito de números, pues el resto de los números reales se "redondea" dando lugar a otro número que es una aproximación suya. Esto se hace siguiendo ciertos estándares. Veamos todo esto con algo más de detalle.

Como es sabido, todo número real no nulo, **x**, se puede escribir de forma única en **notación científica normalizada** (**NCN**) como

$\mathbf{x} = (-1)^s \times 0.d_1 d_2 \ldots d_p \ldots \times 10^e$ donde d_1, d_2, ..., d_p,… son números enteros no negativos, con $d_1 \neq 0$, $s \in \{0,1\}$ y e es un número entero.

Por ejemplo:

x = `0.001234` en **NCN** es: $\mathbf{x} = (-1)^0 \times 0.1234 \times 10^{-2}$

y = `-123.45` en **NCN** es: $\mathbf{y} = (-1)^1 \times 0.12345 \times 10^3$

z = `-0.5699999…` en **NCN** es: $\mathbf{z} = (-1)^1 \times 0.56999\ldots \times 10^0$

Un **formato en coma flotante** no es otra cosa que una representación normalizada de los números con una limitación de dígitos en la mantisa --que ahora está entre 1 y 10 (éste último excluido)-- y limitación en los valores del exponente.
Por ejemplo, en el formato en coma flotante con 4 dígitos en la mantisa y exponente en el intervalo [-4,5], el valor de **x** se representa de forma exacta como

$$\mathbf{x} = (+1) \times 1.234 \times 10^{-3}$$

Si la máquina con la que estuviésemos trabajando operase en este formato almacenaría el valor de **x** con exactitud.

Sin embargo, si la máquina trabaja con aritmética finita de 3 dígitos en la mantisa y exponente en el intervalo [-9,9], el número $\mathbf{x} = 0.001234$ no es almacenable de forma exacta y debe redondearlo, recortando la mantisa y almacenando, en lugar de **x**, una aproximación suya:

$$\mathbf{x}_{\mathbf{aprox}} = (+1) \times 1.23 \times 10^{-3} = 0.00123$$

Como se observa se tiene un error al manejar el valor de **x** en esa máquina y ese error se propagará en aquellos cálculos en los que intervenga **x**, ya que, en su lugar, la máquina operará con $\mathbf{x}_{\mathbf{aprox}}$.

Para más información sobre la representación y almacenamiento en formato en coma flotante, recomendamos el archivo de texto que se puede descargar en el enlace

http://steve.hollasch.net/cgindex/coding/ieeefloat.html

Analicemos con algo más de detalle cómo se produce el redondeo en el formato en coma flotante que utiliza **MatLab**.

Revisando la orden dada antes

```
>> 1+2/3
```

y su respuesta

```
ans =

    1.6667

>>
```

observamos que, aunque el valor exacto de 5/3 = $1.\overline{6}$ --número decimal periódico puro, de período 6--, el programa responde con un número decimal de cinco cifras, cuatro de ellas decimales y la última (7) diferente de la que ocupa la misma posición en el número original (6). Esto se debe a que, **para la representación externa**, el programa redondea a un número decimal con un número prefijado de cifras decimales --en este caso, como se observa, ese número es 4--.

Además trunca la cantidad original siguiendo un criterio que tiene en cuenta la primera cifra decimal que desprecia. Compara el resultado anterior con la respuesta a la orden

```
>> 1/3
```

¿qué diferencias se observan entre el redondeo realizado para `5/3` y el realizado para `1/3`?

¿Qué similitudes se observan en el formato de las dos expresiones decimales obtenidas?

La respuesta a esta última cuestión es sencilla. Ambas constan de un total de cinco dígitos uno en la parte entera y cuatro en la parte decimal. Formalizando esto, se dice que **MatLab** emplea, por defecto, como formato de representación, el **formato short**, que se caracteriza porque el resultado numérico se expresa con 3 cifras (como máximo) en la parte entera y 4 en la parte decimal.

Por otra parte, cuando el número es entero (o decimal) con 10 o más dígitos, **MatLab** emplea, para representar el número en pantalla, la notación científica en la que utiliza la letra **e** para indicar el exponente (positivo o negativo) con base 10, que actúa como factor del número decimal que le antecede. Veamos algunos ejemplos:

`2.0051e-001` es el número $2.0051 \times 10^{-1} = 0.20051$

mientras que

`2.0051e001` es el número $2.0051 \times 10^{1} = 20.051$

Si tecleamos

```
>>  1/23456
```

obtenemos como respuesta

```
ans =

    4.2633e-005
 >>
```

Es decir, el número $\frac{1}{23456}$ se aproxima por:

$4.2633 \times 10^{-5} = 0.000042633$

en el **formato short**.

Existen otros formatos en **MatLab** para representar las respuestas, entre ellos destacamos:

- **`format rat`**: la respuesta se representa en forma de fracción (cociente de dos números enteros)

- **`format long`**: es un formato de representación decimal que emplea 2 cifras en la parte entera y 15 en la parte decimal.

Los dos siguientes formatos incorporan el valor del exponente al normalizar la representación del número real introducido por el teclado:

- **`format short e`**: se caracteriza porque la respuesta tiene una cifra en la parte entera, 4 en la parte decimal y 3 cifras en el exponente.

- **`format long e`**: utiliza 1 cifra en la parte entera, 15 en la parte decimal y 3 en el exponente

Además el **`format bank`** utiliza dos dígitos decimales. Nótese la dferencia entre la representación de 51,2 en formato short y en formato bank

Se puede cambiar de un formato a otro invocando su nombre en la línea de comandos.

Ejercicio 3

Representa en los distintos formatos los siguientes números:

a) $\frac{2}{3}$ *c)* $\frac{2000}{3}$ *e) 1'333*

b) $\frac{20}{3}$ *d)* $\frac{20000000000}{3}$

Ya se ha comentado en líneas precedentes en qué consiste, en líneas generales, la representación en coma flotante. Pues bien, en la mayoría de los procesadores actuales el tipo de almacenamiento de los datos viene especificado por la representación en formato en coma flotante que sigue el **estándar IEEE 754**. En este formato existen dos tipos de almacenamiento de los datos: en simple y doble precisión

Independientemente del formato de representación en pantalla, en MatLab los números se almacenan internamente en doble precisión, guardando cada dato en ocho bytes (64 bits), lo que se traduce en unas 15 cifras decimales aproximadamente, al convertirlo al sistema decimal (**`format long`**). Por eso se dice que, *grosso modo*, en el formato en coma flotante en doble precisión, los números se representan con una precisión finita de 16 cifras decimales. Eso significa que sólo se representan números reales en un determinado rango: entre 10^{-308} y $2 \cdot 10^{308}$ aproximadamente.
Recalquemos que, **aunque cambiemos de formato, lo único que cambia es la apariencia de la respuesta, no la precisión.**

A cualquier número menor que el menor representado (llamado **realmin**) el programa lo considera como cero o como un número no normalizado (es lo que se conoce como un ***error de underflow***). A cualquier número de una magnitud mayor que el mayor número representable[1] (**realmax**) el programa lo considera de magnitud infinita (**Inf** o **inf**), esto es lo que se conoce como un ***error de overflow***.

```
 >> realmax

ans=
        1.7977e+308

>> 2*realmax

ans =

    Inf
>>
```

No se debe confundir la respuesta anterior con la siguiente respuesta a la orden[2]

```
>> 0/0

ans =

      NaN
>>
```

NaN es la abreviatura de ***Not a Number*** que representa una indeterminación ($0/0$, $\infty-\infty$, por ejemplo).

4. Manipulación de expresiones simbólicas

4.1. Variables

Una variable es una etiqueta que identifica una porción de memoria. Para asignar el nombre de una variable a un ente (un número, un vector, una matriz o un carácter) se debe utilizar el operador de asignación **=**

En los dos subapartados siguientes trataremos dos tipos de variables, las numéricas y las simbólicas.

4.1.1. Variables numéricas

[1] El valor de **realmax** corresponde aproximadamente a 2^{1024} (las computadoras realizan sus cálculos en aritmética binaria)

[2] En versiones anteriores aparecía además un aviso: `Warning: Divide by zero.`

Para almacenar los resultados de los cálculos se emplean variables. De hecho, el programa ha respondido siempre con un nombre de variable antes del resultado (**ans**). Es decir, tras cada cálculo realizado el valor de **ans** ha cambiado (observa en la ventana `workspace` el valor actúal de **ans**). Por esta razón, es aconsejable asignar nombres a los resultados u operaciones con los que nos interese trabajar posteriormente.

Cuando definamos una variable, su nombre debe empezar por una letra y puede contener letras, números y guión subrayado. No se permiten espacios y debe contener 63 caracteres como máximo[3].
Así, la primera de las siguientes órdenes es correcta, mientras que la segunda contiene un nombre incorrecto para la variable:

```
>> altura = 15

>> 25dias = 25
```

El programa responde a esa orden advirtiéndonos del error:

```
??? 25dias=25
    |
Error: Unexpected MATLAB operator.
```

Por otra parte, si asignamos un nuevo valor a una variable ya existente, MatLab cambia su valor y, al acceder a ella, el programa guarda ese último valor. Teclea la orden siguiente para comprobarlo:

```
>> altura = 20
```

Para conocer el valor de una variable basta invocar su nombre. Por ejemplo,

```
>> altura
```

y la respuesta del programa es:

```
altura =
          20

>>
```

Obsérvese que MatLab distingue entre mayúsculas y minúsculas, por lo que si introducimos la línea

```
>> Altura
```

la respuesta es:

```
??? Undefined function or variable 'Altura'.
```

[3] Tecleando el comando **namelengthmax** el programa responde con el número máximo de caracteres

Un apunte más sobre cambios en el valor de las variables. Si ejecutamos las órdenes

```
>> y = 5^2;

>> x = 3;

>> z = x-y
```

obtenemos como respuesta:

```
z = -22

>>
```

y si ahora cambiamos el valor de la variable x introduciendo, por ejemplo, la orden

```
>> x = 7
```

entonces este cambio no altera el valor de la variable z salvo que se vuelva a ejecutar la orden que define z. Compruébalo tecleando la siguiente secuencia de órdenes:

```
>> z

>> z = x - y
```

Además de las variables numéricas que podemos definir, MatLab tiene asignados valores por defecto a ciertas variables. A lo largo del curso utilizaremos, entre otras, las siguientes:

```
>> i

>> j

>> pi
```

Las dos primeras órdenes son equivalentes ya que ambas representan la unidad imaginaria de los números complejos. La tercera representa el número π, que es la razón entre la longitud de una circunferencia y su diámetro.

Aunque es posible utilizar esos nombres para definir nuestras propias variables, no es aconsejable, por lo que los consideraremos como *palabras reservadas* del programa.

4.1.2. Variables simbólicas

Además de las variables numéricas existen las variables simbólicas.

La capacidad que **MatLab** posee de trabajar con variables simbólicas permite manejar ecuaciones de manera simbólica y resolverlas después reemplazando por un valor cada una de las distintas variables que aparezcan en la ecuación. También permite resolver, por ejemplo, integrales definidas o realizar cálculos de primitivas de manera analítica.

Las variables simbólicas son cadenas de caracteres que pueden representar números, funciones, expresiones o variables.

4.1.2.1. Cadenas de caracteres

MatLab puede definir variables que contengan cadenas de caracteres. Las cadenas de texto van entre apóstrofes. Los caracteres de una cadena se almacenan en un vector, con un carácter por elemento.

Es habitual convertir los valores numéricos en cadenas de caracteres para poder imprimirlas como títulos en las gráficas. Por ejemplo,

```
>> fahr = 70, grd = (fahr-32)/1.8;

>> title(['Temp. Amb: ', num2str(grd), ' gradoscentigrados'])
```

Existen funciones que actúan sobre cadenas de caracteres en **toolbox\matlab\strfun**

Ejercicio 4

Utiliza la ayuda de **MatLab** *para obtener información sobre* **num2str**

En **MatLab**, por defecto, todas las variables son numéricas así que para definir un objeto simbólico, el programa dispone del comando **sym**. Por ejemplo, la orden

```
>> x = sym('x')
```

declara x como una variable simbólica.

Para definir varias variables simbólicas conjuntamente, debemos utilizar **syms**. Por ejemplo,

```
>> syms a b  c
```

declara a, b y c como variables simbólicas. Observa la diferencia entre la orden anterior y la siguiente:

```
>> syms a, b, c
```

¿Cuál es el efecto de las comas intercaladas entre las letras?

Ejercicio 5

Ejecuta las siguientes órdenes y observa las respuestas correspondientes

```
>> x = 10/7

>> y = sym('10/7')

>> double(y)

>> y+5

>> double(y)+5
```

¿ qué hace cada orden?

En ocasiones puede ser conveniente recordar qué variables hemos definido, para ello basta teclear

```
>> who
```

Esta orden responde con el nombre de todas las variables que hemos definido.

Si en lugar de **who** tecleamos

```
>> whos
```

la respuesta incluye también el tipo al que corresponde cada una de las variables definidas.

Para saber si ya hemos definido una determinada variable, `var`, podemos teclear

```
>> exist('var')
```

cuya respuesta es

```
ans =
       1

>>
```

si ya existe, y

```
ans =
       0

>>
```

en caso contrario.

Se puede observar que si la variable es numérica el resultado aparece afectado por una sangría, mientras que en las variables simbólicas no aparece tal sangrado. Esto puede ayudar a la identificación del tipo de variable, en la ventana de comandos.

Si has practicado reproduciendo los ejemplos incluidos en estos apuntes, teclea las siguientes órdenes,

```
>> clear x, y

>> clear y z

>> z
```

y observarás que para borrar una variable basta teclear el comando **clear** seguido del nombre de la variable (o variables, separados por espacios en blanco).

El comando **clear** borra todas las variables definidas previamente, si no va acompañado de ningún nombre.

A diferencia del comando **clc** visto en el primer apartado, el comando **clear** borra todas las variables del área de trabajo (workspace).

4.2. Expresiones simbólicas

Cuando en una expresión interviene una variable simbólica la expresión es simbólica (aunque también intervengan variables numéricas). Esto significa que podemos operar con variables simbólicas como si fuesen numéricas, para formar expresiones simbólicas. Por ejemplo,

```
>> syms a b c x

>> polinomio = a*x^2+b*x+c
```

define la expresión simbólica llamada polinomio.

Para conocer las variables simbólicas que hay en una expresión simbólica, se utiliza la orden **findsym**. Así, a la orden:

```
>> findsym(polinomio)
```

el programa responde:

```
ans =
a, b, c, x

>>
```

Si deseamos conocer el valor del polinomio anterior en `x = 1`, basta utilizar la orden **subs** así:

```
>> subs(polinomio, x, 1)
```

que se puede "traducir" a lenguaje natural como: sustituye en la variable llamada `polinomio` el valor de `x` por `1`.

Se pueden hacer varias substituciones de forma conjunta. Por ejemplo, si queremos asignar los valores `a = 1` y `b = 2`, la orden

```
>> subs(polinomio, [a,b], [1,2])
```

proporciona como respuesta:

```
polinomio =

x^2+2x+c

>>
```

Una orden equivalente a la anterior es:

```
>> subs(polinomio, {a,b}, {1,2})
```

También es válida:

```
>> subs(polinomio, {a b}, {1 2})
```

La orden **subs** permite obtener expresiones para substituciones concretas, pero si preguntamos al programa por el valor de la variable `polinomio` observaremos que la respuesta a la orden

```
>> polinomio
```

es:

```
ax^2+bx+c

>>
```

En un apartado posterior veremos cómo trabajar con los polinomios de un modo más eficiente que evite el uso de **subs** para obtener valores en puntos concretos.

A veces puede interesarnos extraer el numerador o el denominador de una expresión. A continuación se expone un ejemplo de cómo hacerlo:

```
>> syms x; y = (x^2-1)/(2*(x+4)^3)

>> numden(y)
```

```
ans =

x^2-1

>> [num, den]=numden(y)

num =

x^2-1

den = 2*(x+4)^3
```

Ahora podemos definir `f` como `1/y` (la inversa para la mutiplicación):

```
>> f = den/num

2*(x+4)^3/(x^2-1)

>>
```

Ejercicio 6

Después de haber ejecutado las órdenes anteriores, ¿cuáles son las variables actuales? ¿de qué tipo son?

En otro orden de cosas, algunos comandos pueden ser útiles para obtener una expresión simbólica con una apariencia más apropiada a nuestros propósitos. Por ejemplo, podemos simplificar una expresión simbólica con el comando **`simplify`**, podemos factorizar la expresión con **`factor`**, podemos desarrollar en sumas con **`expand`**, podemos simular la escritura matemática habitual con **`pretty`**. Veamos algunos ejemplos:

```
>> syms x a

>> formulita1 = 3*x^2+2*a*x+ 15*x^3*a^2+25*a, formulita2 = (x+a)^3,
formulita3= x^3+x^2+x+1, formulita4 = (a^3+3*a^2+3*a+1)/(a^2+2*a+1)

>> pretty(formulita1)

>> expand(formulita2)

>> factor(formulita3)

>>simplify(formulita4)
```

Algunos de estos comandos funcionan también con ecuaciones.

Ejercicio 7

Utiliza los comandos de **MatLab** *para obtener las expresiones indicadas:*

a) el desarrollo de $(x^3+1)^2$ *b) la simplificación de* $\frac{x^3+x^2-x-1}{x-1}$

c) la factorización de x^3+x^2-x-1 *d) la expresión matemática habitual de* $(x^3+1)^2$

En general, las expresiones simbólicas con las que vamos a trabajar en la asignatura **Cálculo** serán las fórmulas que determinan una función real de variable real. Trabajaremos con funciones para calcular su valor en algunos puntos, de interés para nosotros por ser valores en los que se anule la función o por ser puntos críticos o puntos de inflexión, por ejemplo.

Algunas de las funciones con las que trabajaremos están predefinidas en **MatLab**, otras las definiremos nosotros. A estos aspectos dedicamos el apartado siguiente.

5. Funciones

MatLab posee un gran número de funciones matemáticas elementales predefinidas, que usaremos de modo habitual. Como en el lenguaje matemático, en **MatLab** las funciones se componen de un nombre seguido de un argumento situado entre paréntesis. Entre otras utilizaremos:

abs(argumento) calcula el valor absoluto de `argumento`
rats(argumento) aproxima `argumento` por una fracción
sqrt(argumento) calcula la raíz cuadrada de `argumento`
exp(argumento) calcula la exponencial (con base e) de `argumento`
log(argumento) calcula el logaritmo neperiano del número real no negativo `argumento`
log10(argumento) idem con el logaritmo en base 10
sin(argumento) calcula el seno de `argumento` medido en radianes
cos(argumento) calcula el coseno de `argumento` medido en radianes
tan(argumento) calcula la tangente de `argumento` medido en radianes
asin(argumento) calcula el arcoseno de `argumento`
acos(argumento) calcula el arcocoseno de `argumento`
atan(argumento) calcula el arcotangente de `argumento`, respondiendo con un ángulo en el intervalo $\left(-\frac{\pi}{2},\frac{\pi}{2}\right)$.

Existe una variante de esta función: **atan2**, que devuelve un ángulo en el intervalo $(-\pi,\pi)$.
Análogamente, se pueden calcular la secante (**sec**), el arcosecante (**asec**), la cosecante (**csc**), el arcocosecante (**acsc**), la cotangente (**cot**) o el arcocotangente (**acot**).

Obsérvese que en el nombre de cada una de las funciones anteriores sólo se emplean letras minúsculas; así que –como ya se ha comentado-- si tecleamos por ejemplo,

```
>> Sin(pi)
```

el programa nos advierte del error cometido con el mensaje:

```
??? Undefined command/function 'Sin'.
```

Para obtener una lista de las funciones elementales, predefinidas en **MatLab**, basta teclear:

```
>> help elfun
```

También se puede obtener información, sobre las funciones incorporadas en el programa, a través del navegador de la ayuda. Accederemos a enlaces que clasifican a las funciones por categoría (`Functions-by cathegory`) o por orden alfabético (`functions-alphabetycal list`).

Podemos asignar como argumento de cada una de las funciones un valor numérico, una variable o una expresión. En este último caso estamos componiendo una función con otra (u otras). Pero, para la composición, **MatLab** posee una orden explícita. Observa las siguientes líneas de comandos y sus respuestas:

```
>> syms x;
>> primera_funcion = x^2;
>> segunda_funcion = x^3+1;
>> f2_o_f1 = compose(segunda_funcion, primera_funcion)

   f2_o_f1 =

x^6+1

>> f1_o_f2 = compose(primera_funcion, segunda_funcion)

  f1_o_f2 =

(x^3+1)^2

>> pretty(expand(f1_o_f2))

ans =

x^6+2x^3+1

>>
```

de lo que se deduce que la composición de funciones, como ya sabemos, no es conmutativa.

También es posible calcular de forma rápida la expresión de la función inversa a una dada (que sea inyectiva!!), con el comando **`finverse`**, así:

```
>> finverse(log(x))
```

```
ans =

exp(x)

>>
```

En cambio, si pretendemos calcular la inversa de $y = x^2$, el programa indica que la inversa no es única (debemos restringirnos a un intervalo en el que la función de partida sea inyectiva, para garantizar la existencia de inversa).

```
>> finverse(primera_funcion)

Warning: finverse(x^2) is not unique.
> In sym.finverse at 48

ans =

x^(1/2)

>>
```

Aún así, el programa ha respondido con la expresión de una inversa $y=x^{1/2}$. Es tarea nuestra ahora conocer en qué intervalo esa función es inversa de `primera_funcion`.

6. Representaciones gráficas

MatLab es una potente herramienta para realizar representaciones gráficas. Aunque su capacidad gráfica se pone de manifiesto de forma más notoria en los gráficos tridimensionales, debido a las características de la asignatura en la que utilizamos este software, nos ceñiremos a representaciones en el plano.

Cuando se realiza un gráfico, **MatLab** abre una ventana (`figure`) (aparte de la ventana de comandos) que denomina `figure1`. Se pueden dibujar conjuntamente varias gráficas, distinguiendo cada una de ellas con un color y un trazado diferente, o bien se puede subdividir la ventana en subventanas en las que aparezcan las gráficas a tratar, para poder realizar, en ambos casos, un estudio comparativo de las mismas. Por otra parte, se puede influir en la apariencia de los gráficos variando el intervalo de presentación en los ejes cartesianos o incluyendo una cuadrícula, por ejemplo. Analizaremos estos aspectos, con más detalle, a continuación.

Una sencilla forma de obtener un gráfico es dibujar un conjunto de puntos en el plano que el programa unirá con segmentos de recta. Se forma así una línea quebrada (poligonal).

Para introducir esos conjuntos de puntos se utilizan vectores. De hecho, los gráficos en dos dimensiones de **MatLab** están orientados a la representación gráfica de vectores.

6.1. Vectores

Distinguiremos entre vectores fila y vectores columna.

Un vector fila se define escribiendo sus componentes entre corchetes y separadas unas de otras por espacios en blanco o por comas. Por ejemplo, la orden

```
>> v = [3 6 2 4 1 5]
```

genera el vector `v = (3, 6, 2, 4, 1, 5)` al igual que la orden

```
>> [3,6,2,4,1,5]
```

Para definir un vector columna cada componente debe separarse de la siguiente por un punto y coma (;) o bien mediante el retorno de carro (pulsando la tecla **Intro**). Por ejemplo el vector $w = (3,6,2,4,1,5)^t$ se define:

```
>> w = [3; 6; 2; 4; 1; 5]
```

Los vectores columna también se pueden obtener como traspuestos de los correspondientes vectores fila. El símbolo que identifica las trasposición de vectores es el apóstrofe (') que se encuentra en la tecla que comparte con el símbolo de cierre de interrogación (?).
Por ejemplo,

```
>> u = v'

 u =
     3
     6
     2
     4
     1
     5

>>
```

Una vez definido un vector se puede acceder a cada una de sus componentes. Por ejemplo, la respuesta a la orden:

```
>> v(1)
```

es

```
ans =

       3
```

```
>>
```

mientras que a la orden

```
>> v([1,2])
```

responde con el subvector formado por la primera y segunda componentes de v:

```
ans =
      3    6

>>
```

y la orden

```
>> v(1:2:5)
```

forma el subvector constituido por las componentes impares de v; es decir, desde la primera a la quinta, con un salto de dos posiciones en los índices de las componentes de v.

En general, **v = [v1: h: vn]** define el vector $v=(v_1, v_1+h, v_1+2h,\ldots,v_1+ph)$, donde p es el mayor entero tal que $p \leq (v_n-v_1)/h$.

En la orden anterior se puede prescindir de los corchetes.

Si el salto es de una unidad entonces no se escribe. Por ejemplo

```
>>  [1:4]
```

equivale a

```
>> [1:1:4]
```

Por otra parte, **linspace(a, b, n)** genera un vector de **n** componentes equiespaciadas, que empiezan en **a** y terminan en **b**. Así, por ejemplo, las dos órdenes siguientes son equivalentes:

```
>> v=2:1:10

>> v=linspace(2, 10, 9)
```

Ejercicio 8

Obtén el vector (3, 1'5, 0, -1'5, -3, -4'5) usando:

a) el operador : b) el comando **linspace**

6.2. Representaciones sencillas: comando `plot`

Consideremos el conjunto de puntos del plano:

{(1,1), (2,-1), (3,7), (-4,4), (5,-2)}

Para dibujar la poligonal que los une, basta teclear la siguiente orden:

```
>> x = [1 2 3 -4 5]; y = [1 -1 7 4 -2]; plot(x, y)
```

Consideremos ahora el conjunto {(1,2),(2,4),(3,6),(4,8),(5,10), (10,20)} y la orden

```
>> x = [1  2  3  4  5  10];  y = [ 2  4  6  8  10 20];  plot(x,y)
```

dado que los puntos están alineados, lo que hemos hecho es dibujar la recta de ecuación y = 2x.

Como se puede observar el comando **`plot`** dibuja puntos del plano generados al utilizar las componentes del primer vector introducido frente a las componentes del segundo vector. Es decir, en el ejemplo anterior, **MatLab** ha dibujado la poligonal que une los puntos (x_i, y_i), con i=1,…,6, donde x_i (respectivamente y_i) es la i-ésima componente del vector `x` (respectivamente `y`).

Para que ello sea posible, obviamente, ambos vectores deben ser de la misma dimensión.

Si en lugar de un par de vectores aplicamos el comando **`plot`** sobre un vector, `v`, pongamos de **`n`** componentes `v` = $(v_1, v_2, \ldots, v_n)$, entonces **MatLab** dibuja la poligonal que une el conjunto de **`n`** puntos (i, v_i), con `i = 1,…,n`. Es decir, cada punto del plano tiene por ordenada una componente v_i del vector introducido `v`, mientras que la abscisa que acompaña a v_i es la posición que v_i ocupa en el vector `v`.

Por ejemplo, la orden

```
>> plot([3,1,1/2,7])
```

dibuja la poligonal que une los puntos del plano: (1,3), (2,1), (3, 1/2) y (4,7).

Al realizar una representación gráfica, se puede añadir un título a la gráfica --con **`title`**-- o incluir etiquetas a los ejes cartesianos --con **`xlabel`** y/o **`ylabel`**--.

También se puede introducir texto con **`textbox`** en un zona de interés. Además se pueden añadir etiquetas (con **`legend`**) que identifiquen los trazos, en caso

de que hayamos dibujado conjuntamente varias curvas. Para todo ello basta acceder al menú de la ventana del gráfico y en **Insert** escoger la opción deseada.

Ejercicio 9

Añade un título al gráfico de la poligonal que une los puntos:
(-1,1), ($\frac{1}{2}$, 0), (-π,$\sqrt{2}$), (3,0) y (2,1).
Etiqueta cada uno de los ejes cartesianos.

Otra forma de añadir etiquetas en un gráfico es escribir las órdenes correspondientes en la ventana de comandos, cuando está activa una ventana de gráfico (en caso contrario, la orden genera una ventana de gráfico con la etiqueta, pero sin gráfico alguno). La sintaxis de cada una de las órdenes es:

```
>> title('Este es el título')

>> xlabel('esta es la etiqueta del eje de abscisas')
```

(con `xlabel off` se borra).

```
>> ylabel('esta es la etiqueta del eje de ordenadas')
```

(con `ylabel off` se borra la etiqueta).

```
>> text(a,b,'este es el punto (a,b)')

>> legend('este texto identifica una línea')
```

La siguiente orden sirve para introducir texto con el ratón

```
>> gtext('este texto va aquí')
```

la orden agrega el texto colocado entre apóstrofes al gráfico actual. Para ello se ejecuta el comando y con el ratón se selecciona la coordenada deseada. Se presiona el botón derecho del ratón, quedando el texto fijado en pantalla en esa posición.

Ejercicio 10

Recupera con las flechas la orden que dibuja los puntos del ejercicio 9 y, a continuación, en la nueva línea de comandos teclea:

```
title('esta es mi primera representación gráfica'); xlabel('este
es el eje de abscisas');
ylabel('este es el eje de ordenadas');
legend ('trazado rectilíneo')
```

Como es una orden muy larga utiliza los tres puntos para continuar en la línea siguiente.

Cada vez que **MatLab** genera un gráfico, utiliza ciertos criterios --opciones por defecto-- sobre la apariencia del mismo, como los sucesivos colores o trazados que emplea en las líneas que dibuja, por ejemplo. Algunas de esas opciones por defecto se pueden modificar. Por ejemplo, al dibujar una circunferencia, la apariencia es la de una elipse, por lo que se debe modificar ese control interno del programa para que su apariencia sea la que esperamos. Esto se consigue con la orden **`axis equal`**. Practica esto introduciendo los dos grupos de órdenes siguientes:

```
>> t = [0:pi/180:2*pi];  x=cos(t); y=sin(t); plot(x, y)
```

```
>> t = [0:pi/180:2*pi];  x=cos(t);  y=sin(t); plot(x, y)
>> axis equal
```

Ambos grupos de órdenes deben generar la gráfica de la curva dada por $x^2+y^2 = 1$; esto es, la circunferencia de centro el origen de coordenadas y radio la unidad; sin embargo, la apariencia de la primera gráfica no es la de una circunferencia.

Por otra parte, se puede modificar el rectángulo de dibujo de la gráfica variando los valores que, por defecto, genera **MatLab** de forma automática. Así, si deseamos representar una curva en el rectángulo [a,b]x[c,d] fijamos ese rango en cada uno de los ejes con la orden:

```
>> axis([a b c d])
```

Con la orden

```
>> axis auto
```

se restaura el rango que el programa emplea por defecto. Para más información sobre la orden `axis` basta teclear:

```
>> helpwin axis
```

Entre otras muchas opciones que se pueden manejar al crear un gráfico está la introducción de una cuadrícula, si ello ayuda a interpretar mejor los datos. Esto se puede hacer con el comando **`grid`** (**`grid off`** desactiva la cuadrícula).

Por otra parte, cada vez que se teclea una orden que genera un gráfico, ésta activa la ventana **`figure1`** y el gráfico substituye a cualquier otro que se hubiese creado con anterioridad. Esto no significa que no se puedan dibujar conjuntamente varias gráficas. De hecho, existen varias formas de hacerlo. La primera que se explica aquí consiste en incluir en el comando **`plot`** tantos pa-

res de vectores como gráficos queremos representar conjuntamente. Por ejemplo, las órdenes

```
>> x=0:pi/100:3*pi; y=cos(x); z=cos(3*x); plot(x,y,x,z);
```

dibujan las funciones y = cos(x) e y = cos(3x) conjuntamente.

Veamos, con un ejemplo distinto, otra forma de dibujar conjuntamente dos gráficas:

```
>> x=0:pi/100:pi/4; y=tan(x);  plot(x,y)
>> hold on
>> z=sin(2*x); plot(x,z)
```

la primera orden ha generado la gráfica de la función tangente en el intervalo [0, π/4]. Después, **`hold on`** mantiene activa la ventana en la que se ha dibujado ese gráfico y con la tercera orden se representa la función y = sen(2x) conjuntamente con la tangente.

Por otra parte, se pueden superponer dos gráficas visualizando las escalas de cada una de ellas, usando el comando **`plotyy`**. Estrictamente, este comando dibuja dos funciones con dos escalas distintas para las ordenadas. Las escalas aparecen una a la derecha y la otra a la izquierda de la figura representada.
La orden

```
>> plotyy(x,y,x,z)
```

actúa sobre los vectores **x**, **y**, **z** definidos previamente. Las componentes del vector **x** son las abscisas de los puntos que dibuja en ambas gráficas, mientras que las ordenadas del primer gráfico (respectivamente el segundo) vienen dadas por las componentes del vector **y** (respectivamente **z**). Veamos un ejemplo concreto:

```
>> x = -1:0.01:12; y=300*exp(-0.01*x); z=0.2*exp(-0.1*x);

>> plotyy(x,y,x,z)
```

Como se ha comentado previamente, se puede elegir el color y trazo de la curva a dibujar. Si no especificamos una preferencia, el programa por defecto utiliza trazo continuo y una secuencia de colores preestablecida. Si deseamos modificar eso, debemos incluir nuestro diseño como una opción en el comando `plot` (debe ir entre apóstrofes). Los posibles colores y algunos de los posibles trazados se relacionan a continuación:

```
b: azul (blue)                  -: trazado lineal continuo
c: verde claro  (cyan)         --: trazado lineal discontinuo
g: verde oscuro (green)       -.: trazado lineal discontinuo intercalando punto y línea
k: negro (black)               : : trazado con línea de puntos
m: rojo oscuro (magenta)        *: marca *
r: rojo (red)                   +: marca +
w: blanco (white)               o: círculo
y: amarillo (yellow)            x: marca con un aspa
                               .: un punto
```

Veamos un ejemplo de esto:

```
>> x = 0:pi/100:3*pi;  y = cos(x);  z = cos(3*x);

>>  plot(x, y, 'r--',x, z, 'g:')
```

Debemos señalar que si dibujamos varias gráficas en una misma ventana con varias órdenes `plot` (usando `hold on`), las gráficas se superponen en un mismo color, salvo que incluyamos en el comando nuestras preferencias de colores. En cambio, si se dibujan varias curvas en un mismo comando **`plot`**, las gráficas se van dibujando en colores distintos, elegidos por el programa.

Para profundizar en las diferentes opciones que permiten modificar nuestros gráficos, basta recurrir, por ejemplo, a la ayuda *on-line* del comando **`plot`**.

Ejercicio 11
Se desea medir la altura que alcanza un cultivo, sabiendo que dicha altura es función del tiempo. Las mediciones se hacen una vez al día y los resultados obtenidos se resumen en la siguiente tabla:

Tiempo (días)	*1*	*2*	*3*	*4*	*5*
Altura (cm)	*5'2*	*6'6*	*7'3*	*8'6*	*10'7*

Obtén la gráfica que explique ese crecimiento. Dicha gráfica debe incluir un título apropiado y etiquetas en los ejes que indiquen cuál es la variable correspondiente a cada uno de ellos.
Hazlo de varias formas distintas. Para ello, varía el trazo y el color.

En el ámbito de esta asignatura, nos interesará comparar varias gráficas para analizar cuestiones relativas a propiedades del cálculo diferencial o integral. Por ejemplo, visualizaremos la gráfica de una función junto con las de sus derivadas primera y segunda --de forma superpuesta en una misma ventana-- para aprender a extraer conclusiones sobre una función a partir de las características de su derivada (utilizando, por ejemplo, las propiedades de las derivadas para determinar los intervalos de crecimiento o decrecimiento). En otros casos será preferible dibujarlas por separado para su estudio; como en el caso de la interpretación geométrica del teorema fundamental de integración, por ejemplo. Para esto último emplearemos el comando **`subplot`**.

6.3. El comando `subplot`

El comando **`subplot`** permite ver varias gráficas en una misma ventana subdividiendo ésta en sub-ventanas. La sintaxis genérica del comando **`subplot`** es:

```
>> subplot(m,n,i)
```

que indica que la ventana de gráficos activa se divide en **m** partes horizontales y **n** verticales. El tercer argumento de la orden anterior (**i**) indica la subdivisión que está activa –numerada de izquierda a derecha y de arriba abajo--. Veamos esto con un ejemplo:

```
>> x = -2*pi:pi/25:2*pi;  y = sin(x);  z = abs(x);  t=log(x+10);
w = exp(x-1);
>> subplot(2, 2, 1); plot(x, y, 'r-')
>> subplot(2, 2, 2); plot(x, z, 'g-.')
>> subplot(2, 2, 3); plot(x, t, 'b o')
>> subplot(2, 2, 4); plot(x, w, 'k*')
```

Una tercera forma de disponer de varias gráficas es activar ventanas sin eliminar la que hayamos creado previamente (llamada **figure1**). Para ello, desde el menú de la ventana del gráfico, podemos escoger en **File** la opción **New...** y, a continuación, seleccionar **figure**. En ese caso se genera la ventana denominada **figure2**, sin perder **figure1**.

Esto mismo podríamos haberlo hecho a través de la ventana de comandos con la orden

```
>> figure
```

o bien

```
>> figure(2)
```

que genera una ventana de gráficos con el número 2.

Ejercicio 12

Dibuja en el intervalo [-2, 2] las curvas $y = x^3$ *e* $y = x^2+1$ *conjuntamente, etiquetándolas con un título que indique cuál es cada una de ellas. Hazlo de tres formas distintas:*

a) *superponiendo las gráficas utilizando los mismos ejes para ambas gráficas.*
b) *en dos subventanas, cada gráfica con sus propios ejes coordenados*
c) *generando dos ventanas de gráficos distintas.*

Ya hemos visto que el comando **plot** dibuja gráficas uniendo pares de puntos; es decir, dibuja pares de vectores, definidos previamente. Pero, para obtener la representación gráfica de una función real de variable real usando **plot**, sería entonces necesario introducir primero un número elevado de puntos para que, de este modo, la poligonal que los une tenga una apariencia parecida a la gráfica de la función deseada. Por ejemplo, para dibujar la función *seno* en el intervalo cerrado [-3π/2, 5π/2] se podría utilizar:

```
>> x = linspace(-3*pi/2, 5*pi/2, 100);
```

Y, a continuación, debería aplicarse la función *seno* a cada componente del vector resultante de la orden anterior; pues cada una de esas componentes se corresponde con la abscisa de un punto sobre la curva:

```
>> y = sin(x);   plot(x, y)
```

Esto significa que la función *seno* admite como argumento un número real o un vector y, en este último caso, actúa sobre cada elemento del vector.

También la función *coseno* admite como argumento un vector. Sin embargo, no todas las funciones actúan sobre vectores, aplicándose a cada componente y generando un nuevo vector de resultados. Por eso, es necesario disponer de operaciones que se realicen elemento a elemento entre vectores o, en su defecto, disponer de algún comando que nos exima de construir los vectores de puntos por los que deba pasar la función.

Además existe otro inconveniente para representar gráficamente una función si usamos `plot`: sería necesario conocer *a priori* las zonas en las que la gráfica varía de forma más rápida para elegir un mayor número de puntos en esa zona. Eso implica un conocimiento específico de la forma de la gráfica que, en la mayor parte de los casos, no tenemos. Para evitar estas dificultades, MatLab dispone (en el toolbox de cálculo simbólico) de los comandos **`fplo`**t y **`ezplot`**.

6.4. El comando `fplot`

Las gráficas de funciones reales de variable real se pueden obtener de forma sencilla utilizando un comando que evita las operaciones sobre cada componente de un vector: el comando **`fplot`**.

La ventaja de este comando es que **`fplot`** es adaptativo en el sentido de que utiliza más puntos en aquellas zonas en donde la función sufre más variación.

La sintaxis genérica es:

```
>> fplot(f, [a b])
```

que dibuja `f` en el intervalo `[a,b]`

```
>> fplot(f, [a b c d])
```

dibuja `f` en el rectángulo `[a,b]x[c,d]`. Por ejemplo, la orden

```
>> fplot('sin(x)', [0 4*pi], 'g-.')
```

dibuja la función *seno* en el intervalo [0, 4π] en color verde y trazado discontinuo alternando segmentos de líneas y puntos. Mientras que la orden

```
>> fplot('sin(x)', [0 4*pi 0 1.5], 'r')
```

dibuja la gráfica de la función *seno* en el rectángulo [0, 4π]×[0, 1'5] en color rojo y trazado continuo. En consecuencia, al ejecutar esa orden sólo vemos la mitad superior de la gráfica del seno en el intervalo [0, 4π].

Ejercicio 13

Dibuja la función seno en el intervalo [-3π, π]. *Dibuja la función coseno en ese mismo intervalo, conjuntamente con la función seno, usando el comando* **fplot**.

6.5. La orden `ezplot`

La orden **`ezplot`** es otra alternativa útil para generar de forma sencilla la gráfica de una función. La sintaxis genérica es:

```
>> ezplot (f)
```

donde `f` es el nombre de la función a representar. La orden

```
>> ezplot (f, [a, b])
```

representa `f` en el intervalo `[a, b]`.

Observa la equivalencia entre la orden

```
>>  y=sym('x^2-1'); ezplot(y)
```

y la orden

```
>> ezplot('x^2-1')
```

Observa además que, en ambos casos, aparece un título por defecto en la gráfica que genera el programa.

Cuando la función viene dada implícitamente; esto es, mediante una ecuación que relaciona la variable independiente `x` con la variable dependiente `y`, `F(x,y)=0`, para representar esta expresión en un rectángulo `[a, b]x[c,d]`, basta crear una expresión simbólica, llamémosle `F`, y, a continuación, teclear:

```
>> ezplot (F, [a, b, c, d])
```

si no incluimos los límites de representación para los ejes, la gráfica se representa, por defecto, en el cuadrado $\left[-2\pi,2\pi\right]\times\left[-2\pi,2\pi\right]$.

Veamos un par de ejemplos de utilización del comando **`ezplot`**:

Ejemplo 1 (función definida de forma explícita):

Representación de la gráfica de $y=x^2+sen(2x)-1/2$ en el intervalo [-10, 10].

```
>> syms x; y=x^2+sin(3*x)-1/2; ezplot(y, [-10,10])
```

Ejemplo 2 (función definida implícitamente):

Representación de la semicircunferencia inferior de centro el origen de coordenadas y radio 4:

```
>> syms x y;  F = x^2+y^2-16;  axis equal; ezplot(F,[-4,4,-4,0])
```

Ejercicio 14

Representa las funciones siguientes en los intervalos o rectángulos indicados:

a) $f(x) = tg(x)$ *en* $[-\pi/4, \pi/4] \times [-10, 10]$

b) $f(x) = ln(x)$ *en* $[0, e]$

c) $f(x) = e^{3*x}cos(6x)$ *en* $[0, 4\pi] \times [-1, 2]$

d) $ln(xy) = 25$ *en* $[1, 10]$

7. Manejo de polinomios

Aunque en un apartado anterior hemos tratado las funciones en general, por su especificidad, dedicamos este apartado a los polinomios.

En **MatLab** se puede definir un polinomio a partir de sus coeficientes. Para ello deben introducirse en un vector, empezando por el coeficiente correspondiente al término de mayor grado del polinomio. Por ejemplo:

```
>> polinomio = [4 0 1 -1]
```

representa al polinomio $4x^3+x-1$. Nótese que es imprescindible incluir la segunda componente nula, que corresponde al término de grado dos del polinomio que no aparece.

Se pueden calcular las raíces de un polinomio (es decir, los valores que lo anulan) así por ejemplo, las soluciones de la ecuación $x^2-6x+9 = 0$ se calculan con la orden:

```
>> roots([1,-6,9])

ans =
```

```
   3.0000 + 0.0000i
   3.0000 - 0.0000i
```

y, recíprocamente, también se puede obtener la expresión de un polinomio si conocemos sus raíces. Por ejemplo, a la orden

```
>> poly([2 0 -1])
```

el programa responde con:

```
ans =

      1 -1 -2 0
```

lo que quiere decir que el polinomio cuyos ceros son 2, 0 y -1 no es otro que x^3-x^2-2x.

Si deseamos conocer el valor de ese polinomio en un punto, por ejemplo en x = 7, basta teclear ahora

```
>> polyval(ans, 7)
```

y la respuesta es:

```
ans =

      280
>>
```

Veamos otro ejemplo:

```
>> polinomio = [1 1 1]; polyval(polinomio, [-1,0,1,3,1/2])
```

proporciona el valor del polinomio x^2+x+1 en x=-1, x=0, x=1, x=3 y x=1/2 y devuelve el valor en un vector.

También se puede representar la gráfica de un polinomio:

```
>> x = -3:0.2:3; p = [2 -1 0 1 -1/2]; y=polyval(p,x);

>> plot(x, y, 'm')
```

Se pueden derivar e integrar los polinomios con órdenes específicas para ellos. Las órdenes que derivan, respectivamente integran, un polinomio llamado `p` son:

```
>> polyder(p)

>> polyint(p)
```

Al igual que estos dos comandos específicos para trabajar con polinomios, existen otros muchos que se pueden consultar en la ayuda de MatLab: **conv**, **deconv**, **residue**, **polyder**(p,q), etc.

Ejercicio 15

Dado el polinomio x^4+5x^2-2,

a) *llámale p y calcula sus raíces.*
b) *Calcula el valor de p en* x = 0.
c) *Deriva p.*
d) *Calcula una primitiva de p.*

8. Resolución de ecuaciones

8.1 El comando `solve`

Esta orden se emplea para resolver ecuaciones. Actúa sobre expresiones simbólicas o sobre una cadena de caracteres.

Será un comando útil para hallar cortes de la gráfica de una función con los ejes coordenados, obtener puntos críticos (posibles extremos), puntos de inflexión, determinar recintos de integración, etc. Su sintaxis genérica es:

```
>> solve('ec', 'x')
```

que trata de resolver la ecuación `ec`, hallando el valor de la variable `x`.

Nótese que una ecuación debe ir escrita entre apóstrofes para convertirla en una cadena de caracteres.

```
>> solve('ec')
```

es la forma abreviada de la orden anterior, cuando la incógnita es única.

```
>> solve(expr)
```

trata de resolver la ecuación `expr = 0`, cuando la expresión es simbólica y la incógnita es única.

```
>> solve(expr, x)
```

esta orden se utiliza para hallar el valor de `x` en la ecuación `exp = 0` si `expr` es una expresión simbólica.
Veamos algunos ejemplos:

```
>> solve('a*x+b=2', 'a')

ans =
```

```
-(b-2)/x

>> solve('a*x+b=2', 'b')

ans =

2-a*x

>> solve('a*x^2+b*x+c', 'x')

ans =

 1/2/a*(-b+(b^2-4*a*c)^(1/2))
 1/2/a*(-b-(b^2-4*a*c)^(1/2))

>>
```

Una orden equivalente a la anterior es:

```
>> syms a b c x; solve(a*x^2+b*x+c, x)
```

Otro ejemplo:

```
>> sym x
>> solve(x^4+1)

ans =

  1/2*2^(1/2)+1/2*i*2^(1/2)
 -1/2*2^(1/2)+1/2*i*2^(1/2)
  1/2*2^(1/2)-1/2*i*2^(1/2)
 -1/2*2^(1/2)-1/2*i*2^(1/2)

>>
```

Si no puede obtener soluciones simbólicas **MatLab** proporciona aproximaciones numéricas. Por ejemplo:

```
>>  sym x

>> solve(exp(x^6)-sin(x))

ans =

-.818667178532169461253401996899l3-
.837227884093015530365404189022l33*sqrt(-1)

>>
```

El comando **solve** también se usa para resolver sistemas de ecuaciones. En este caso la sintaxis es:

```
>> [y1 … yn]=solve('ec1', …, 'ecn', 'var1', …, 'varn')
```

Se pueden sobreentender `var1`, `var2`, …, `varn`, si son las únicas variables simbólicas, como en el siguiente ejemplo:

```
>> syms x1 x2;

>> [X1 X2]=solve(5*x1+6*x2-23, 3*x1-x2)

X1= 1
X2= 3

>>
```

que son las soluciones del sistema de ecuaciones

$$\{5x_1+6x_2 = 23,\ 3x_1-x_2 = 0\}$$

Ejercicio 16

a) Resuelve $2x^3+11x^2+12x-9 = 0$

b) Resuelve el sistema de ecuaciones $\{x+y+z = 1,\ 3x+y=3,\ x-2y-z=0\}$

9. Los ficheros .m

En el apartado 1 de estos apuntes ya se ha comentado la posibilidad de grabar nuestro trabajo para poder incluir comentarios y ver su contenido en sesiones posteriores de trabajo. Grabar el trabajo de una sesión es importante si queremos guardar un conjunto de comandos o definir funciones que poder incorporar en un momento determinado a una sesión de trabajo posterior. Esto puede hacerse mediante ficheros **`m`** (**`m-files`**). Los ficheros de extensión **`m`** son ficheros de texto ASCII que se pueden crear desde cualquier editor de texto, aunque **MatLab** posee su propio editor que es el que utilizaremos nosotros. Al invocar el nombre del archivo en la ventana de comandos se ejecutan las órdenes que contengan una tras otra[4]. Veamos un ejemplo: Comencemos abriendo el editor de textos. Para ello basta acceder en el menú del botón **Start** a la opción **Desktop Tools** y en el submenú elegir **Editor**. También se puede abrir el editor tecleando tras el `prompt` del programa:

```
>> edit
```

En la pantalla que se abre, tecleamos por ejemplo:

```
% Este fichero contiene un gráfico
x = [1, 2, 3, 4];
y = [4, 3, 2, 1]
```

[4] Se pueden incluir comentarios a varias líneas conjuntamente con la opción **comment** (con **uncomment** se deshace la acción)

plot(x, y)

Una vez tecleadas las órdenes anteriores lo grabamos con el nombre `primer_dibujo.m`
(Observa la ventana `workspace`)

Volvemos a la ventana de comandos y tecleamos

```
>> primer_dibujo.m
```

Observa que en la ventana de comandos no aparece la línea del vector `x` pero sí se ejecuta ya que el gráfico sí se carga.

Finalmente, recopilamos algunos comandos útiles al manejar ficheros:

- **`what`**: proporciona una lista de los `m`-ficheros (también muestra ficheros con extensión `.mat`) del directorio actual (`current directory`).
- **`delete`** *nombre_del_fichero.ext*: borra el fichero con el nombre especificado en la orden.
- **`type`**: permite visualizar el contenido de ficheros en la ventana de comandos así como el código de las funciones
- **`which`** *nombre_de_archivo.ext:* informa del directorio en el que está el archivo especificado.
  También sirve para saber si una variable es una función predefinida en **MatLab**. Por ejemplo, a la orden

```
>> which abs
```

el programa responde:

```
built-in (C:\Program
Files\MATLAB\R2008a\toolbox\matlab\elfun\@logical\abs)%logical method
```

indicando la ubicación de la función valor absoluto.

En los comandos **`type`**, **`delete`**, **`which`** si no se especifica la extensión el programa asigna la extensión **`.m`** al nombre de los ficheros.

10. Referencias

Aunque son innumerables los textos que se pueden hallar dedicados a **MatLab**, e innumerables también los enlaces en la *web*, destacamos aquí algunos en concreto, por su idoneidad en relación con la asignatura **Cálculo**. Debemos mencionar que, por lo novedoso de la versión de **MatLab** con la que trabajamos (de 5 de marzo de 2010), las referencias incluidas aquí manejan versiones anteriores del programa pero, en lo esencial, las referencias son válidas.

10.1. Bibliografía

B. D. Hahn, D. T. Valentine. *Essential MATLAB for Engineers and Scientists* (3ª ed.) BH, 2007.
J. H. Mathews, K. D. Fink. *Métodos Numéricos con MatLab* (3ª ed). Prentice Hall, 1999.
H. Moore. *MATLAB para ingenieros*. Pearson Prentice Hall, 2007
A. Quarteroni, F. Saleri. *Cálculo científico con MatLab y Octave*. Springer, 2006.

10.2. Enlaces web

http://steve.hollasch.net/cgindex/coding/ieeefloat.html

http://www.mathworks.com/

http://mat21.etsii.upm.es/ayudainf/aprendainf/