Software Matemático

Aplicado a la Docencia

Colección manuales uex -75
(E.E.E.S.)

Amelia
Álvarez (coord.)

75

SOFTWARE MATEMÁTICO
APLICADO A LA DOCENCIA

MANUALES UEX

75

(E.E.E.S.)
Espacio
Europeo
Educación
Superior

AMELIA ÁLVAREZ SÁNCHEZ (coord.)
TERESA ARIAS MARCO
JOSÉ LUIS BRAVO TRINIDAD
ADRIÁN GORDILLO MERINO
EVA LÓPEZ SANJUÁN
PEDRO MARTÍN JIMÉNEZ
CONCEPCIÓN MARÍN PORGUERES
IGNACIO OJEDA MARTÍNEZ DE CASTILLA
MARÍA ISABEL PARRA ARÉVALO
DIEGO FRANCISCO YÁÑEZ MURILLO

SOFTWARE MATEMÁTICO
APLICADO A LA DOCENCIA

UNIVERSIDAD DE EXTREMADURA

2011

ÁLVAREZ SÁNCHEZ, Amelia (coordinadora)

SOFTWARE matemático aplicado a la docencia / Amelia Álvarez Sánchez (Coord.).
Cáceres : Universidad de Extremadura, Servicio de Publicaciones, 2011
pp. ; 17 x 24 cm. – (Manuales UEX, ISSN 1135-870-X ; 75)
ISBN de méritos: 978-84-694-5697-2

1. Matemáticas-Estudio y enseñanza. 2. Matemáticas-Informática. I.Álvarez Sánchez, Amelia. II.Arias Marco, Teresa. III.Bravo Trinidad, José Luis. IV.Gordillo Merino, Adrián. V.López Sanjuán, Eva. VI.Martín Jiménez, Pedro. VII.Marín Porgueres, Concepción. VIII.Ojeda Martínez de Castilla, Ignacio. IX.Parra Arévalo, Mª Isabel. X.Yáñez Murillo, Diego Fco. XI.Tít. XII.Serie. XIII. Universidad de Extremadura, Servicio de Publicaciones, ed.
51(075.8)
51:004

La publicación del presente manual forma parte de las actividades desarrolladas durante el curso 2009/10 por el grupo de innovación docente (GID) "Software Matemático Aplicado a la Docencia (SMAD)" financiado en convocatoria competitiva por el Vicerrectorado de Calidad y Formación Continua de la Universidad de Extremadura y coordinado por D. Ignacio Ojeda Martínez de Castilla

UNIVERSIDAD DE EXTREMADURA

UNIÓN EUROPEA
Fondo Social Europeo

JUNTA DE EXTREMADURA

Edita

Universidad de Extremadura. Servicio de Publicaciones
C./ Caldereros, 2 - Planta 2ª - 10071 Cáceres (España)
Telf. 927 257 041 - Fax 927 257 046
publicac@unex.es
www.unex.es/ publicaciones

ISSN 1135-870-X
ISBN 978-84-694-5697-2

Índice general

Introducción

El conocimiento de informática en el contexto de las nuevas titulaciones de grado se ha convertido en una competencia transversal, sobre todo en los grados de las ramas de Ciencias Experimentales e Ingeniería y Arquitectura. En particular, en las materias de Matemáticas y Estadística no se espera sólo que el alumno sepa resolver los problemas, sino que sepa hacerlo utilizando las herramientas informáticas que existen. Sin embargo, los profesores de esas materias no hemos recibido una formación especifica sobre el uso de software matemático en el aula, siendo nuestra formación, generalmente, autodidacta y a menudo incompleta. Es por ello que decidimos formar un Grupo de Innovación Didáctica en la Univesidad de Extremadura para mejorar nuestra formación docente adquiriendo, profundizando y actualizando nuestros conocimientos relativos al software matemático existente y su utilización actual en el aula.

En primer lugar decidimos cuáles eran los programas más apropiados para su uso en las aulas universitarias, por su facilidad de uso, por su versatilidad, por el uso profesional que eventualmente podrían hacer los estudiantes al finalizar sus estudios. Los programas elegidos fueron Octave/MATLAB (cálculo científico y visualización de datos), R (cálculo estadístico y generación de gráficos) y MAXIMA (cálculo simbólico y numérico).

A continuación diseñamos y elaboramos material para el aprendizaje y uso de estos programas en el aula como apoyo a la adquisición de competencias transversales y específicas de asignaturas concretas de los grados. Para ello se realizaron tutoriales de ayuda para cada programa elegido y se diseñaron una serie de prácticas-modelo, que se desarrollaron en los tres lenguajes de programación. Tanto los tutoriales como las prácticas se ajustan casi en su totalidad a un formato común, para hacer más fácil el paso de un sofware a otro si así se desea.

El objetivo de este manual es presentar estos tutoriales y prácticas, tanto en los ficheros pdf como en los ficheros originales TEX, y ponerlas a disposición de la comunidad universitaria, para que aquellos docentes universitarios que puedan necesitarlos tengan acceso a ellas y puedan adaptarlas a sus necesidades concretas.

Por último, quisiéramos agradecer a nuestros compañeros del Departamento de Matemáticas de UEX, especialmente a Carmen Calvo Jurado, que utilizaran los tutoriales y las prácticas, pues con sus comentarios ayudaron a mejorarlos.

Badajoz, junio de 2011.

Parte 1

Tutoriales

TUTORIAL 1

Breve introducción al MATLAB

1. Introducción

MATLAB y Octave, los programas, son entornos integrados para el cálculo científico y la visualización de datos. MATLAB está distribuido por The MathWorks (http://www.mathworks.com). El nombre proviene de MATrix LABoratory pues originariamente fue desarrollado para el cálculo matricial. Octave, también conocido como GNU Octave (http://www.gnu.org/software/octave/) es un software que se distribuye libremente sujeto a los términos de la licencia GPL (GNU Public License).

A lo largo de las prácticas haremos uso frecuente de la denominación "MATLAB"; en ese caso, MATLAB deberá ser entendido como el lenguaje que es el subconjunto común a ambos programas Octave y MATLAB.

En esta práctica el estudiante se familiarizará con el programa MATLAB, sus comandos y funciones básicas y su lenguaje de programación.

2. Uso como calculadora científica

Al ejecutar MATLAB se abre una ventana principal (consola de MATLAB) y aparece un indicador $\gg$ (u `octave:1>`). Esta es la línea de comandos, cualquier instrucción que escribamos aquí será ejecutada al pulsar la tecla `<intro>`. Podemos detener esa ejecución (por ejemplo, si tarda demasiado) pulsando `<control>+c`.

En la consola de MATLAB escribe el siguiente comando

```
>> 2+3
```

y a continuación pulsa la tecla `<intro>`.

También puedes escribir los siguientes comandos (pulsa la tecla `<intro>` después de cada línea):

```
>> 2-3
>> 2*3
>> 6/3
>> 2^3
```

Ahora ya sabes cómo sumar, restar, multiplicar, dividir y elevar un número a una potencia en MATLAB de la misma forma que puedes hacer con una calculadora o con otros paquetes de software.

Aunque en nuestros ejemplos hemos usado números enteros, MATLAB trabaja con números reales y complejos:

```
≫ 2.3131
≫ 8.6853e-12
≫ 2-3i
```

También MATLAB tiene predefinidas ciertas funciones o constantes. Escribe

```
≫ sqrt(5)
≫ pi
≫ i
```

En el primer caso has obtenido el valor de la raíz cuadrada de 5, en el segundo el número π y en el tercero la raíz cuadrada de -1.

```
≫ i^2
```

Tanto `i` como `pi` son algunos ejemplos de constantes con valores prefijados en MATLAB.

Observa que el resultado más reciente siempre se almacena en una variable llamada `ans` (de "answer"). Sólo el último resultado se almacena ahí.

Escribe

```
≫ ans
```

Como habrás comprobado el resultado anterior sigue ahí.

Escribe ahora

```
≫ 2*ans
```

Puedes hacer cálculos con el resultado que se almacena en `ans` de la misma manera que se puede hacer con cualquier número.

En la consola de MATLAB, pulsa la flecha hacia arriba `<up>` una vez, y luego pulsa `<intro>`. Pulsa ahora la flecha hacia arriba `<up>` varias veces y observa lo que sucede, repite el experimento pulsando también la flecha hacia abajo `<down>`. Puedes recuperar todas tus órdenes anteriores y volver a ejecutarlas si lo deseas.

Puedes utilizar las flechas izquierda `<left>` y derecha `<right>` del teclado para mover el cursor a derecha e izquierda y editar así una línea de comandos. De este modo puedes corregir los errores, recuperando un comando con la flecha hacia arriba y usando la edición para corregirlo. Puedes

escribir o cambiar tantos símbolos como quieras o utilizar la tecla de retroceso `<backspace>` para eliminarlos.

Utiliza la flecha arriba para recuperar el comando `sqrt(5)`. Edítalo usando las flechas izquierda y derecha y calcula el valor de $\sqrt{8+\pi}$.

Escribe

```
≫ format long
```

y a continuación

```
≫ pi
```

Prueba ahora a escribir

```
≫ format short
```

y a continuación

```
≫ pi
```

Escribe

```
≫ format short e
```

y pregunta por el valor de la variable `pi`.

A continuación escribe

```
≫ format rational
```

(si estás en Octave es `format rat`) y pregunta por el valor de la variable `pi` y luego calcula 1/7. ¿Qué ha ocurrido?

Finalmente, para terminar de divertirnos, escribe

```
≫ format hex
```

y pregunta por el valor de la variable `pi`.

A menos que desees ver el resto de las respuestas en base hexadecimal, cambia de nuevo a formato corto, escribe

```
≫ format short
```

Escribiendo sólo `format` volvemos al formato predefinido, que es el formato corto.

Ejercicio 1. Calcula con MATLAB el resultado de las siguientes expresiones:

$$\frac{1+\sqrt{2}}{1-\sqrt{5}}, \quad \cos^2(\pi), \quad e^{1+i}.$$

Ejercicio 2. Calcula

$$\frac{1}{10} + \frac{2}{10} - \frac{3}{10}.$$

Ejercicio 3. Calcula $1/10 + 2/10$ y escribe 0,3 ¿Coinciden los dos valores? ¿Y con `format long`?

3. Variables

No olvides pulsar `<intro>` después de cada orden.
Escribe

```
≫ x=3
≫ y=2
≫ 2*x+y
```

Ten en cuenta que a la variable `x` se le ha asignado el valor 3, y a la variable `y` se le ha asignado el valor 2, por lo que la expresión algebraica que se ha escrito al final se ha evaluado usando esos valores, y el resultado ha sido almacenado en la variable `ans`.

Escribe

```
≫ x, y
```

En general, para ver lo que se almacena en una variable, simplemente escribe su nombre.

Escribe

```
≫ z=5;
```

y luego escribe

```
≫ z
```

Observa el efecto del punto y coma `;`. Úsalo cuando no quieras ver el eco de la entrada. El equipo ya sabe lo que se almacena en la variable `z`.

Escribe

```
≫ 2+x-y*z;
≫ ans
```

Escribe ahora

```
≫ ans + 3;
≫ r = 2*ans
```

Para ver los nombres de las variables que has utilizado en la sesión actual, escribe

```
≫ who
```

Define ahora las siguientes variables

```
≫ a = 3;
≫ b = 5;
≫ c = 1;
≫ discr = b^2 - 4*a*c;
```

Escribe primero

```
≫ discr
```

y luego

```
≫ DISCR
```

¿Distingue MATLAB las mayúsculas de las minúsculas?

En unas ocasiones se desea utilizar la letra A para representar a un número real y en otras para representar a una matriz; MATLAB te permitirá hacerlo. Recuerda que si olvidas qué variables están actualmente en uso, puedes utilizar el comando `who` (o también el comando `whos`, que da más información).

Definamos ahora

```
≫ sqrt(discr)
≫ x = (-b + sqrt(discr)) / (2*a)
```

De este modo `x` es una de las raíces de la ecuación de segundo grado $ax^2 + bx + c = 0$ con `a` $= a$, `b` $= b$ y `c` $= c$.

La función `clear` elimina el valor asignado a una variable. Escribe

```
≫ clear a
≫ a
```

También se utiliza para eliminar todos los valores definidos en una sesión:

```
≫ clear
≫ who
```

Ejercicio 1. Asigna a una variable x el número uno. Sobre esa variable calcula $x = 2x$. Repetir esta operación hasta obtener el valor de 256.

Ejercicio 2. Calcula

$$\left(\frac{e^{1+i}-e^{-1-i}}{2i}\right)^4+\left(\frac{e^{1+i}+e^{-1-i}}{2}\right)^4+$$
$$+2\left(\frac{e^{1+i}-e^{-1-i}}{2i}\right)^2\left(\frac{e^{1+i}+e^{-1-i}}{2}\right)^2.$$

Para ello asigna a dos variables el interior de los paréntesis y opera con ellas.

4. Organización del trabajo

Si deseas guardar el trabajo que realizas en una sesión MATLAB, debes crear un archivo de diario. Este archivo contendrá una transcripción completa y literal de todo lo que se teclea en el ordenador y que, más adelante, puedes editar con cualquier procesador de textos.

Para iniciar un archivo de diario llamado, por ejemplo, `Lab1.txt` en C: debes escribir

```
≫ diary c:\Lab1.txt
```

y pulsar <intro>. Probablemente no percibas que haya ocurrido nada, pero el ordenador ha comenzado a registrar todo lo que ocurre en la consola de MATLAB. Puedes interrumpirlo en cualquier momento escribiendo

```
≫ diary off
```

Escribiendo `diary on` volverás a guardar tu trabajo en el archivo de diario que previamente habías creado, añadiendo la información nueva a continuación de la que ya tenías. Si hay un diario activo al salir de MATLAB, el archivo de diario será cerrado automáticamente.

Al salir de una sesión de MATLAB, los valores de todas las variables se pierden. El archivo de diario sólo mantiene un registro de lo que ha aparecido en la pantalla, no se almacenan los valores de las variables. Si lo deseas, antes de salir de MATLAB, puedes guardar los valores actuales de las variables en un archivo para su uso en futuras sesiones. El siguiente comando guarda los valores en un archivo llamado `mi_archivo`:

```
≫ save mi_archivo
```

Los valores de todas las variables se guardarán en forma binaria, ilegible para nosotros, en un archivo llamado `mi_archivo.mat` dentro de tu directorio de trabajo. Para cargar estos valores de las variables en una futura sesión de MATLAB, escribe

```
≫ load mi_archivo
```

Si quieres guardar sólo el valor de algunas variables (por ejemplo `v1` y `v2`), escribe

```
≫ save mi_archivo v1 v2
```

Para saber cuál es tu directorio de trabajo, escribe

```
≫ pwd
```

Para conocer los directorios y archivos que hay en tu directorio de trabajo, escribe

```
≫ dir
```

Si quieres acceder a otro directorio que está en tu directorio de trabajo, llamado `datos`, escribe

```
≫ cd datos
```

y para retroceder un nivel

```
≫ cd ..
```

Cualquier declaración en MATLAB que sea precedida por un signo de tanto por ciento % es tratada como un comentario, es decir, es ignorada por el intérprete de MATLAB. Escribe

```
≫% Ahora vamos a definir los coeficientes del polinomio:
≫ a = 2
≫ b = 5
≫ c = 6
```

Escribe

```
≫ n = 3% n es el número de raíces de la ecuación
```

Como ves, un comentario puede ser una línea entera o sólo la segunda parte de una línea.

En un fichero de diario de MATLAB conviene añadir muchos comentarios sobre el trabajo que estás realizando para que después, al leerlo, puedas comprender lo que estabas haciendo.

Por último, una órden muy útil cuando tenemos la pantalla llena de resultados de las operaciones es `clc`, que borra la pantalla sin modificar los contenidos de las variables.

Ejercicio 1.

1. Inicia un archivo de diario llamado `prueba.txt`.
2. Define las variables $a = 2$ y $b = 4$.
3. Guárdalas en un archivo llamado `variables`.
4. Borra las variables a y b. Escribe `who` para comprobar que ya no están.
5. Carga el archivo `variables`. Escribe `who` para ver qué variables hay ahora.
6. Cierra el diario.

5. Vectores, matrices y funciones con matrices

En muchas ocasiones usarás vectores y matrices en MATLAB. Un vector es simplemente una lista de números. Para definir un vector, escribe sus componentes entre corchetes y separadas por comas o espacios en blanco. Esto te dará un "vector fila" que se muestra horizontalmente. Puedes convertirlo en un "vector columna" poniendo una comilla simple ' después de escribir el vector.

Escribe

```
≫ x = [3, -1, 4, 2]
≫ y=x' % Trasponer el vector x
```

Definamos ahora

```
≫ v = [1 2 4 8 16] % Sin separar las entradas por comas
```

Para acceder al valor de una entrada concreta del vector, se debe escribir el nombre del vector y entre paréntesis la posición de la entrada. Por ejemplo, el producto de la segunda entrada del vector `x` y de la tercera entrada del vector `v` se calcula así:

```
≫ x(2)*v(3)
```

Para añadir una entrada a un vector previamente definido podemos escribir

```
≫ v(6) = 32
```

o también

```
≫ v = [v 64]
```

También se puede extraer un vector a partir de otro definido previamente:

```
≫ v([2,3]) % Entradas 2ª y 3ª de v
≫ x(2:4) % Todas las entradas desde la 2ª hasta la 4ª de x
≫ v(1:2:5) % Entradas 1ª, 3ª y 5ª de v
≫ w = [1 3 4];
≫ v(w) % Entradas 1ª, 3ª y 4ª de v
```

Observa que la operación `a:d:b` devuelve un vector cuyas entradas son los números que van desde a hasta b en incrementos de d unidades, es decir, $a, a+d, a+2d, \ldots, a+nd$ con $a+nd \leq b < a+(n+1)d$. La operación `a:1:b` coincide con la operación `a:b`.

Asignemos, por ejemplo, a la variable `w` el vector de coordenadas $(2, 4, 6, 8, 10, 12, 14)$ usando la operación anterior:

```
≫ w = [2:2:14]
```

Otra manera de hacer esto mismo es con la orden `linspace(a,b,n)`, que devuelve el vector de n componentes equiespaciadas entre a y b, es decir, $a, a+h, \ldots, a+(n-1)h = b$, donde $h = (b-a)/(n-1)$. Escribe

```
≫ w2 = linspace(2,14,7)
```

Puedes hacer operaciones aritméticas elementales en todas las coordenadas de un vector a la vez. Esto es especialmente importante cuando se desea trazar la gráfica de una función.

Escribe

```
≫ c = 2
≫ c*x % Multiplicar por c cada una de las entradas de x
≫ x-c % Restar c a cada una de las entradas de x
≫ x/c % Dividir entre c cada una de las entradas de x
```

Otras operaciones con vectores son las siguientes:

```
≫ sum(v) % Suma todas las entradas de v
≫ v+w % Suma de los vectores v y w
≫ w*(v') % Producto de w y v (considerados como matrices)
≫ sin(v) % Seno de cada entrada de v
```

Las operaciones anteriores son vectoriales, es decir, los argumentos son vectores. También se pueden definir operaciones elemento a elemento.

Concretamente, para poder elevar todas las coordenadas de un vector a una potencia o poder dividir o multiplicar un vector por otro coordenada a coordenada, debemos utilizar el operador punto (`.`).

Escribe

```
≫ x^3 % Dará error
≫ x.^3 % Ten en cuenta el uso del punto
≫ 3.^x
≫ v.*w % Producto entrada a entrada de los vectores v y w
≫ w./v % División entrada a entrada de w entre v
```

También podemos usar operadores de comparación con vectores. Por ejemplo, si se pone una ecuación o inecuación con un vector, MATLAB devolverá un vector con un 1 o un 0 según se cumpla o no la ecuación o inecuación en la entrada correspondiente.

Escribe

```
≫ v >5
≫ v == 0
```

Esto nos permite seleccionar de un vector los elementos que cumplan cierta condición. Por ejemplo, para quedarnos con los elementos de `v` mayores que 5, escribimos

```
≫ v5=v(v >5)
```

Si nos queremos quedar con los múltiplos de 3 del vector `w`, escribimos

```
≫ w(mod(w,3)==0)
```

La función `mod(a,b)` devuelve el resto de la dividión de `a` por `b`.

Para definir una matriz, se introducen sus elementos entre corchetes, separando las filas con ; (o con <intro>). Escribe

```
≫ A=[1 2 3;4 5 6;7 8 9]
```

Podemos seleccionar un elemento, una fila o una submatriz de una matriz dada. Escribe

```
≫ A(1,2) % Elemento en la 1ª fila, 2ª columna de A
≫ A(1,:) % Primera fila de A
≫ A(:,2) % Segunda columna de A
≫ A(1:2,1:2) % Submatriz de A
≫ A([1,3],[2,3]) % Submatriz de A
```

También están definidas algunas operaciones con matrices, como la trasposición, la suma, el producto, el determinante, etc.:

```
≫ B=A' % Traspuesta de A
≫ C=A+B % Suma de A y B
≫ D=A*C % Producto de A por C
≫ E=det(D) % Determinante de D
```

Ejercicio 1.

1. Crea un vector con los números naturales del 1 al 10.
2. Calcula la raíz cuadrada de cada uno de esos números.
3. Crea un vector con los números naturales del 1 al 1000.
4. Obtén un vector con el cociente de dividir cada número natural entre 1 y 1000 entre 7.
5. Obtén un vector de 1000 posiciones que en cada posición i tenga 1 si i es divisible entre 7 y 0 si no lo es.
6. Obtén un vector con los múltiplos de 7 entre 1 y 1000.

Ejercicio 2. Crea la matriz

$$A = \begin{pmatrix} 4 & 1 & 2 \\ 1 & 5 & 0 \\ 2 & 0 & 7 \end{pmatrix}$$

Llama B a la submatriz de 2×2 obtenida de A tomando sus filas $1, 3$ y columnas $1, 2$. Calcula el determinante de B.

6. Representación gráfica de funciones

Todas las funciones elementales, además de muchas otras funciones, están disponibles en MATLAB. Las funciones operan tanto sobre números como sobre vectores, de modo que puedes utilizar MATLAB como una calculadora científica.

Escribe

```
≫ y = cos(3.5)% En radianes
≫ x = [-2, 4, 1, -1]
≫ z = sin(x)
≫ w = log(1 + abs(x))./atan(x)
```

El método estándar de representar gráficamente funciones en MATLAB requiere hacer listas (en realidad vectores) de valores x y de valores y, y luego usar una orden para dibujar una curva (lineal a trozos) uniendo los pares de la forma (x, y). Puedes obtener una lista de valores de x que cubra el rango que necesitas mediante la operación de la forma `a:d:b`.

A continuación, dibujaremos la función $y = e^x$ en el intervalo $[-1, 2]$ usando la función `plot`:

```
≫ x = -1: 0.1: 2
≫ y = exp(x)
≫ plot(x, y)
```

Normalmente, no querrás ver en la pantalla el valor de las listas `x` e `y` por lo que deberás escribir el punto y coma al final de la línea para suprimir la salida.

También podemos representar dos curvas en la misma gráfica:

```
≫ x = 0: 0.1: 2*pi;
≫ plot(x, sin(x),x,cos(x))
```

Podemos controlar el color de la gráfica y el estilo de línea mediante la adición de un parámetro opcional al comando `plot`. Las opciones para los colores son `'c'` (cian), `'m'` (magenta), `'y'` (amarillo), `'r'` (rojo), `'g'` (verde), `'b'` (azul), `'w'` (blanco), y `'k'` (negro). Opciones de estilo de línea son `'-'` para la línea continua (opción predeterminada), `'- -'` para la línea de guiones, `':'` para la línea de puntos, y `'-.'` para la línea de raya-punto.

Vamos a hacer la gráfica de color rojo y de puntos:

```
≫ x = - 4: 0.5: 3;
≫ y = cos(x);
≫ plot(x, y, 'r:')
```

MATLAB tiene distintos marcadores para el tipo de punto `'+'`, `'o'`, `'*'`, `'x'`, `'s'` para el cuadrado, `'d'` para el diamante, `'^'`, `'v'`, `'<'`, `'>'` para distintos tipos de triángulos, `'p'` para el pentágono, y `'h'` para el hexágono. Si se especifica un marcador, pero no un estilo de línea, el gráfico consistirá en una serie de puntos no conectados.

Aquí tenemos uno con diamantes azules:

```
≫ plot(x, y, 'bd')
```

Cuando queremos producir la gráfica de un conjunto de datos como puntos aislados (no conectados por líneas) el uso de marcadores es la opción apropiada en MATLAB. Rara vez lo que querremos es que esos puntos estén conectados por líneas:

```
≫ x = [1, 2, 4, 5, 7]
≫ y = [10, - 4, 3, 1, 6]
≫ plot(x, y, 'ro')
```

Otra manera de trazar más de una gráfica en una sola ventana es con `hold on` para fijar la gráfica más antigua y luego hacer otra gráfica de la

manera habitual. A continuación, añadimos la recta $y = -3x + 15$ al último gráfico:

```
≫ hold on
≫ x = 1: 0.2: 7;
≫ y = - 3*x + 15;
≫ plot(x, y, 'b')
≫ hold off
```

Como es natural la orden `hold off` cancela el efecto de `hold on`.

Hay otros comandos adicionales que nos permiten controlar la "ventana" de la gráfica, agregar líneas en forma de red, poner etiquetas en los ejes, añadir un título para la gráfica y especificar una leyenda. Vamos a añadir estas características a nuestra gráfica:

```
≫% Ventana: 1 <x <7, - 5 <y <1
≫ axis( [1, 7, - 5, 11] )
≫ grid on% grid off para eliminar la cuadrícula
≫ xlabel( 'valores de la x' )
≫ ylabel( 'eje OY' )
≫ title( 'Grafica de x frente a y' )
≫ legend( 'puntos de los datos', 'recta')
```

Otra forma, a veces más cómoda, de hacer una gráfica en MATLAB es utilizando la función `fplot`, cuyos argumentos son la función entre comillas simples y el rango de valores. Puedes utilizar opciones para el estilo de la línea de la curva como `'- +'`, `'- x'`, `'- o'`, y `'- *'`, pero no tienes elección de color (y generalmente tendrás menos control sobre el aspecto de la gráfica):

```
≫ clf% borra el gráfico de la ventana
≫ fplot('sin(x^2)', [- pi, pi])
≫ grid on
```

Experimenta con combinaciones de los comandos que se utilizan para la representación gráfica para asegurarte de que los entiendes bien.

Ejercicio 1. Dibuja la gráfica de la función $y = \sin(x)$ entre los puntos 0 y π. Añade la gráfica del polinomio $y = x - x^3/6 + x^5/120$. Cambia los colores para practicar.

7. Estructuras de programación

Para crear m-archivos y funciones en MATLAB/Octave, se dispone de una serie de operadores lógicos básicos que son fundamentales para la programación. Veamos algunos ejemplos.

Usemos el operador `if` para comprobar si dos variables tienen el mismo valor o no. En caso afirmativo la respuesta será `1` y en caso negativo la respuesta será `0`:

```
≫% If. c vale 1 si a=b y 0 en caso contrario
≫ a=3;
≫ b=2;
≫ if a==b
≫ c=1;
≫ else
≫ c=0;
≫ end
```

Observa que el símbolo de comparación "igual a" es `==`.

Usemos el operador `for` para repetir una instrucción un determinado número de veces:

```
≫% For. El índice i recorre los números del 1 al 5.
≫% Si el vector a tiene alguna entrada mayor o igual a 4
en
≫% alguna posición, finaliza la ejecución.
≫ a=[2 3 1 4 2];
≫ for i=1:5
≫ if a(i) >= 4
≫ break;
≫ end
≫ end
```

Veamos en qué etapa ha finalizado la ejecución:

```
≫ i
```

Observa que el signo de comparación "mayor o igual que" es `>=`.

Finalmente, usemos el operador `while` para repetir la ejecución de una sentencia un número indefinido de veces, siempre que se cumpla una determinada condición:

```
≫% While. Recorremos el vector a
≫% hasta que un elemento sea igual a 3.
≫% Si todos los elementos son distintos de 3, dará un
error.
≫ i=1;
≫ a=[3 3 3 2 3];
≫ while a(i)~=3
≫ i=i+1;
≫ end
```

Observa que el signo de comparación "distinto de" es `~=`.

Como se ha visto antes, la sentencia `break` detiene la ejecución de un bucle `for` o `while`.

Ejercicio 1. Crea una función que reciba un vector y devuelva la suma de sus componentes, usando un bucle para ello. Comprueba que el resultado es correcto usando la función `sum` de MATLAB.

Ejercicio 2. (Para valientes) Crea una función que reciba un vector de números naturales y devuelva uno formado por aquellos que sean primos.

8. m-archivos y funciones definidas por el usuario

Una buena forma de ejecutar "automáticamente" una secuencia de comandos de uso frecuente sin tener que tomarse la molestia de escribirlos en cada momento es utilizar un m-archivo (o *script*). Un m-archivo es un archivo de texto que se puede crear con cualquier editor de texto. En él, puedes poner el mismo tipo de comandos que escribes en la consola de MATLAB.

Escribiendo el nombre del m-archivo en la consola de MATLAB, el sistema ejecuta sucesivamente cada una de las órdenes en el archivo, como si las hubieses teclado individualmente en la consola de MATLAB.

Utiliza un editor de texto cualquiera para crear un archivo que contenga las líneas siguientes:

```
clear
d2r = pi/180
a = sin(30*d2r)
b = (1 + a)/4
```

Guarda el archivo en tu directorio de trabajo bajo el nombre de `prueba.m`. Ahora en la consola de MATLAB escribe

```
≫ prueba
```

Habrá ocasiones en las que tendrás que definir tus propias funciones porque no estén integradas en MATLAB. Para ello, tendrás que utilizar un tipo especial de m-archivo "función". Al igual que con las funciones estándar como $sin(x)$, un m-archivo función acepta unos argumentos de entrada y devuelve unos resultados.

Veamos a continuación un ejemplo de m-archivo función. La variable de entrada será x y la de salida $y = fun(x)$. Ten en cuenta que la primera línea debe contener la palabra clave `function` y describir las entradas y salidas. El valor de salida debe ser asignado dentro del archivo a la variable que aparece antes del signo de igualdad en la primera línea. El m-archivo función debe tener el mismo nombre que la función, a saber, `fun.m`.

Utiliza un editor de texto para crear un archivo que contenga las líneas siguientes:

```
function y = fun(x)
d2r = pi/180 ;
y = 3*cos(x*d2r) - 1 ;
```

Crea un fichero en tu directorio de trabajo bajo el nombre de `fun.m`.
En la consola de MATLAB, escribe

```
≫ fun(45)
```

Escribe ahora

```
≫ z = 30
≫ a = fun(z)+ 8
```

Las funciones definidas por nosotros se pueden dibujar con tanta facilidad como las incorporadas en MATLAB. Utilicemos la funciones de dibujo que hemos aprendido en la Sección 6 de este tutorial.

Escribe, por ejemplo,

```
≫ x = -10: 0.5: 20;
≫ y = fun(x);
≫ plot(x, y)
```

También puedes usar el comando `fplot`:

```
≫ fplot('fun(x)', [- 10, 20], '-o')
```

Puedes definir funciones cuyo argumento conste de más de una variable. Así mismo, la salida puede tener más de una variable. Es decir, tanto las variables de entrada como las de salida pueden ser vectores.

Por ejemplo, escribe en el editor

```
function z = polinomio(x,y)
z = x^4*x^2*y^2+y^2;
```

y guárdalo con el nombre `polinomio.m`. Ahora, en la consola de MATLAB escribe

```
≫ a = 0.13
≫ d = polinomio(sin(a),cos(a))
```

En este caso las variables `x` e `y` de la función son las variables de entrada, mientras que `z` es la variable donde se guardará el valor que devolverá la función.

Redefinamos ahora nuestra función de tal forma que la variable de entrada sea un vector. Por ejemplo, escribe en el editor

```
function z = polinomio2(v)
z = v(1)^4*v(1)^2*v(2)^2+v(2)^2;
```

guárdalo con el nombre `polinomio2.m` y en la consola de MATLAB escribe

```
≫ u = [sin(a), cos(a)]
≫ polinomio2(u)
```

Finalmente, veamos un ejemplo de una función cuya salida sea un vector:

```
function [a,b] = intercambia(x,y)
a = y;
b = x;
```

Escribe ahora en la consola de MATLAB

```
≫ x=1;y=2;
≫ [x,y]=intercambia(x,y)
```

Ejercicio 1. Crea una función que reciba dos valores a y b y que devuelva la solución de $ax + b = 0$.

Ejercicio 2. Crea una función que reciba tres valores a, b y c y que devuelva las soluciones de $ax^2 + bx + c = 0$.

9. Obteniendo ayuda

MATLAB posee una amplia colección de mensajes de ayuda. Si conoces el nombre de una función, se puede encontrar la sintaxis correcta para el comando y ejemplos de su uso escribiendo `help nombre_fun` , suponiendo que se conoce el nombre del comando y es `nombre_fun` .

Otra forma de obtener ayuda consiste en utilizar la tecla de tabulación `<tab>` en el indicador de la siguiente manera: escribe `at` en la consola de MATLAB y pulsa `<tab>` para obtener la lista funciones y comandos que comienzan por "at".

Las funciones de MATLAB son almacenadas en varias bibliotecas. Veamos una lista de las bibliotecas de MATLAB[1]:

```
≫ help
```

Supongamos que se desea saber si el arcotangente, la función tangente inversa, es una función elemental en MATLAB. Mira la lista de bibliotecas. Parece que `elfun` podría ser una posible biblioteca candidata a contener las funciones elementales. Mirando más abajo en la lista de bibliotecas, no parece haber un candidato más probable.

Para invocar la ayuda de una librería, no es necesario hacer referencia al directorio, basta escribir el nombre de la biblioteca:

```
≫ help elfun
```

Efectivamente, la lista tiene una función llamada `atan` en ella. Pidamos ayuda sobre esta función:

```
≫ help atan
```

Si no encontramos dónde está una función, podemos usar el comando `lookfor` que busca en la ayuda de todas las funciones el texto que digamos. Por ejemplo, si no nos acordamos cuál es la función inversa del coseno, podemos escribir

```
≫ lookfor cosine
```

y nos devolverá todas las funciones en cuya ayuda aparezca la palabra "cosine", junto con una breve descripción.

[1] Los siguientes comandos pueden funcionar de forma diferente en Octave, o simplemente no funcionar.

TUTORIAL 2

Breve introducción a R

1. Introducción

R, también conocido como "GNU S", es un entorno y un lenguaje de programación para el cálculo estadístico y la generación de gráficos. R implementa un dialecto del premiado lenguaje S, desarrollado en los Laboratorios Bell por John Chambers et al. Provee un acceso relativamente sencillo a una amplia variedad de técnicas estadísticas y gráficas y ofrece un lenguaje de programación completo, bien desarrollado, con el que añadir nuevas técnicas mediante la definición de funciones. Entre otras características, dispone de almacenamiento y manipulación efectiva de datos; operadores para cálculo sobre variables indexadas (arrays) y en particular matrices; una amplia, coherente e integrada colección de herramientas y posibilidades gráficas para análisis de datos.

R es un entorno en el que se han implementado muchas técnicas estadísticas, tanto clásicas como modernas. Algunas están incluidas en el entorno base de R y otras se acompañan en forma de bibliotecas (packages, disponibles en http://www.r-project.org). Actualmente, R y S son los dos lenguajes más utilizados en investigación en estadística.

Existe una diferencia fundamental en la filosofía que subyace en R (o S) y la de otros sistemas estadísticos (como, por ejemplo, SAS o SPSS). En R, un análisis estadístico se realiza en una serie de pasos, con unos resultados intermedios que se van almacenando en objetos, que pueden ser observados o analizados posteriormente, produciendo unas salidas mínimas. Sin embargo en SAS o SPSS se obtendrá de modo inmediato una salida copiosa para cualquier análisis, por ejemplo, una regresión o un análisis discriminante.

Los grandes atractivos de R/S son:

- La capacidad de combinar, sin fisuras, análisis "preempaquetados" (por ejemplo, una regresión logística) con análisis ad-hoc, específicos para una situación.
- La capacidad de manipular y modificar datos y funciones.

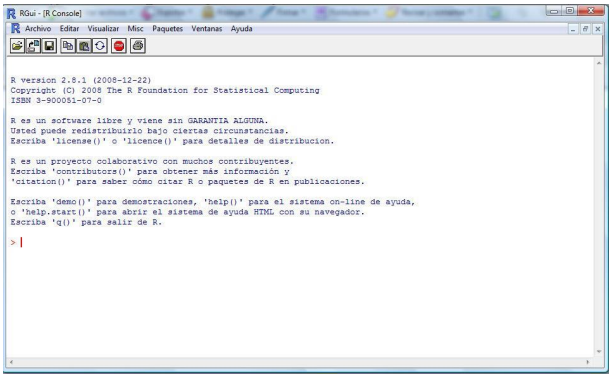
- Los gráficos de alta calidad: visualización de datos y producción de gráficos.
- La comunidad de R es muy dinámica, con gran crecimiento del número de paquetes, e integrada por estadísticos de gran renombre.
- Hay extensiones específicas a nuevas áreas como bioinformática, geoestadística y modelos gráficos.
- Es un lenguaje orientado a objetos.
- Se parece a Matlab y a Octave, y su sintaxis recuerda a C/C++.
- Es gratuito y su descarga e instalación son sencillas.

Entre otros, vamos a utilizar en estas prácticas el paquete R Commander (Rcmdr), que proporciona un entorno gráfico que facilita el manejo de R y sirve como generador de instrucciones. Para tareas sencillas, es posible que no se necesite otro nivel de uso que el que proporciona Rcmdr. Esta herramienta es bastante reciente y está creciendo de una manera bastante rápida. Para situaciones más complejas de trabajo e investigación, una vez superado el respeto inicial a la herramienta, lo habitual es decantarse por trabajar directamente con la consola de R, creando y editando instrucciones con una evidente economía de recursos y, lo que es más importante, con un control total sobre los procedimientos que en cada momento se van a aplicar.

Instalación e inicio de R y RCMDR.

1. Descargar el archivo de instalación o el código que corresponda al sistema operativo del

   Tras abrir el programa R, comienza su ejecución, apareciendo una ventana similar a la siguiente:



2. Para instalar un paquete, como puede ser Rcmdr, escribiremos

```
> install.packages("Rcmdr", dependencies = TRUE)
```

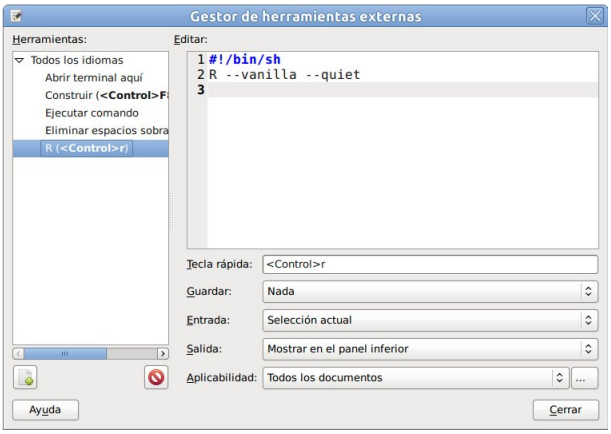
En Windows, la ventana de R incluye un gestor de paquetes. En ese caso también podemos:

a) Entrar en el menú Paquetes -> Instalar paquete(s)...
b) En la ventana que aparece, elegir un repositorio (o espejo) desde el que descargar el paquete. Cuanto más cercano esté, mejor.
c) Elegir el nombre del paquete que se quiere instalar, en este caso Rcmdr. En el caso de este paquete se van a instalar muchos otros subpaquetes (dependencias), que son necesarios para que puedan funcionar todas las opciones de los menús del entorno gráfico.

Además de utilizar R introduciendo directamente el código en la ventana de comando, hay una forma cómoda de ir evaluando (y guardando, si se desea) el código de R que tengamos escrito.

- En Windows, basta dividir la ventana principal de R en dos partes: en una estará el código que vamos escribiendo, desde la que será enviado a la otra para que se evalúe. Para ello hay que hacer que la ventana de comandos ocupe una mitad, y la ventana de editar código la otra. La subventana para editar código aparece al entrar en el menú: *Archivo -> Nuevo script*, si el código no está guardado en ningún archivo, o *Archivo -> Abrir script...* si ya lo está.

  Ahora, para evaluar una parte del código, basta seleccionarla y pulsar <*ctrl*> +*r*, o, menos cómodo, entrar en el menú *Editar -> Correr línea o selección*. Para evaluar todo el código del archivo, se puede entrar en el menú *Editar -> Ejecutar todo* o seleccionarlo todo y utilizar la forma anterior.
- En Linux, podemos configurar el editor de textos *gedit* como indicamos a continuación: En primer lugar abrimos gedit (si no lo tenemos instalado debemos escribir `sudo apt-get install gedit` en un terminal). A continuación abrimos la ventana para crear, eliminar y modificar herramientas externas, *Herramientas -> Gestionar herramientas externas ...*. Pulsamos el icono *nuevo* (un folio con un más en verde) y creamos una herramienta nueva como en la imagen:



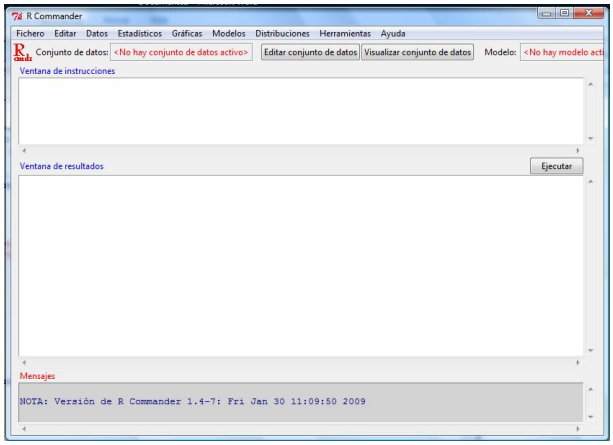
Ahora, para evaluar una parte del código, basta seleccionarla y pulsar `<ctrl>+r`.

Inicio y manejo de R Commander.

Una vez que R se está ejecutando, simplemente cargando el paquete Rcmdr mediante la instrucción

```
library(Rcmdr)
```

en la consola de R se inicia la interfaz gráfica de usuario de R Commander:



Las ventanas de R Commander y R Console flotan libremente en el escritorio. Siempre que sea posible, es más sencillo usar los menús y cuadros de diálogo de R Commander.

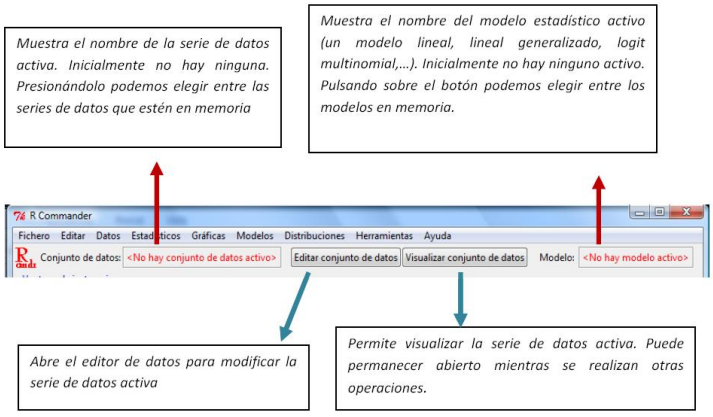
La interfaz R Commander incluye los siguientes elementos:

Menú: aparece en la parte superior de la ventana, con las siguientes opciones:

Fichero	Opciones de menú para cargar y guardar archivos de instrucciones, de resultados y/o el área de trabajo de R; cambiar directorio de trabajo y salir.
Editar	Opciones para editar los contenidos de las ventanas (Copiar, Cortar, Pegar, Borrar, Buscar, Seleccionar, Deshacer y Rehacer) y limpiar la ventana en la cual esté situado el cursor.
Datos	Opciones para leer y manipular datos.
Estadísticos	Submenús que contienen opciones de menú para una variedad de análisis estadísticos básicos.
Gráficos	Opciones de menú para crear gráficos estadísticos simples.
Modelos	Opciones de menú para obtener resúmenes, intervalos de confianza, test de hipótesis, diagnósticos y gráficas para un modelo estadístico, así como para añadir cantidades diagnósticas, como residuos, a la serie de datos.
Distribuciones	Probabilidades, cuantiles y gráficos, para distribuciones estadísticas estándares.
Herramientas	Opciones para cargar paquetes R, y para establecer algunas opciones.
Ayuda	Opciones de menú para obtener información sobre R Commander.

Cuando una opción de menú aparece en gris, significa que está inactiva y no puede aplicarse en el contexto actual.

Bajo los menús aparece una **barra de herramientas** con una fila de botones:



Ventana de instrucciones: en ella aparecen las instrucciones de R generadas por R Commander. También es posible escribir directamente o modificar dichas instrucciones, como lo haríamos en R, tras el símbolo de sistema >. Sin embargo, el propósito de utilizar R Commander es evitar esto. Presionando el botón *Ejecutar*, que está a la derecha bajo la ventana, la línea que contiene el cursor o bien el bloque de instrucciones seleccionadas se ejecuta para su resolución.

Ventana de resultados: en ella aparecen los resultados que se van generando al ejecutar las instrucciones. Las instrucciones escritas en esta ventana aparecen en rojo y los resultados en azul igual que ocurre en *R Console*.

Mensajes: aparece sombreada en la parte inferior y muestra mensajes de error, advertencias y otro tipo de información de utilidad.

Gráficos: cuando creamos gráficos, éstos aparecen en una ventana separada.

2. Uso como calculadora científica

R puede ser utilizado como una calculadora de modo interactivo.

Operadores		*Significado*
Aritméticos	%%	Resto de una división
	%/%	Cociente entero de la división
	*	Multiplicación
	+	Suma
	−	Resta
	^	Elevar a una potencia
	/	División
Lógicos	<	Menor que
	<=	Menor o igual que
	>	Mayor que
	>=	Mayor o igual que
	==	Igual
	!=	Distinto

Ejercicio 1. Realizar algunos cálculos aritméticos y con operadores lógicos en R.

Resumimos a continuación los comandos en R de algunas funciones elementales, así como una breve descripción de los mismos.

Función	Significado
`exp(x)`	Exponencial de x (e^x)
`log(x)`	Logaritmo neperiano de x
`log(x, n)`	Logaritmo en base n de x
`log10(x)`	Logaritmo en base 10 de x
`sqrt(x)`	Raíz cuadrada de x
`factorial(x)`	Factorial de x (x!)
`floor(x)`	Mayor valor entero < x
`ceiling(x)`	Menor valor entero > x
`trunc(x)`	Valor entero más próximo a x, entre x y 0. Coincide con floor(x) para valores positivos y con ceiling (x) para valores negativos
`round(x, digits=0)`	Redondea el valor de x, siendo digits el número de decimales que se permiten.
`signif(x, digits=6)`	Devuelve el valor de x con 6 dígitos en notación científica
`cos(x)`	Coseno de x (en radianes)
`sin(x)`	Seno de x (en radianes)
`tan(x)`	Tangente de x (en radianes)
`acos(x), asin(x), atan(x)`	Funciones trigonométricas inversas
`abs(x)`	Valor absoluto de x

3. Variables, vectores, matrices, arrays y data.frames

En R, a diferencia de otros lenguajes, no es necesario declarar a priori el nombre y el tipo de las variables. Para asignarle un valor, se usa el signo <- . Los nombres de las variables empiezan por una letra, que puede ir seguida de más letras, dígitos o los caracteres punto (.) o línea de subrayado (_). Los distintos tipos de valores que R asigna a las variables son:

Modo	Tipo
logic	valores lógicos: `TRUE (T)` o `FALSE (F)`
numeric	números reales
integer	valores enteros
double	valores reales con doble precisión
complex	números complejos
character	caracteres (letras y/o números)
NA	dato no disponible (*not available*)
NaN	no es un número (*not a number*)

El tipo básico de objetos de R son los vectores. Se trata de estructuras atómicas, puesto que todos sus elementos del vector deben ser del mismo tipo (numérico, complejo, lógico o de carácter). Ejemplos de creación de vectores son:

```
> c(1,2,3,4,5)
> c(T,F,T,T)
> x <- c("Badajoz",Çáceres"); x
> c(1,2,3,çuatro") ->x; x
> c(1,2,3,F)
```

La función `c( )` también permite concatenar vectores. Por ejemplo:

```
> x <- c(1,3,5)
> y <- c(2,4,6)
> c(x,y)
```

Podemos extraer elementos de un vector de este modo:

```
> x <- c(2,4,8,16,32,64,128,256,512,1024)
> x[3]
> x[-3]
> x[c(1,5,7)]
```

O evaluar expresiones para el vector

```
> x >256
```

Para crear patrones hay varias formas. Veamos cómo funcionan:

```
> 1:5
> seq(1,6)
> seq(1,6,by=0.5)
> seq(1,6,length=10)
> rep(1,5)
> rep(c(1,2),5)
> rep(1:4,2)
> rep(1:3,c(1,4,5))
```

Ejercicio 1. Crear sendos vectores con los 50 primeros números pares e impares, a los que llamaréis vp y vi, respectivamente. Concatenar ambos vectores y seleccionar aquellos valores que ocupan posiciones que sean múltiplos de 3.

Ejercicio 2. Crear un vector con los números de vuestro D.N.I., donde cada uno de los números que lo componen debe aparecer repetido un número de veces igual a él mismo.

Una **matriz** en R es un conjunto de objetos indizados por filas y columnas. Un **array** en R es lo mismo, salvo que puede tener más de dos dimensiones. La sintaxis general de la orden para crear una matriz es: `matrix(data, nrow, ncol, byrow=T)`, donde `data` son las entrada de la matriz, `nrow` y `ncol` el número filas y columnas respectivamente y la opción `byrow=T` indica que los datos se irán leyendo por filas (la opción por defecto es `byrow=F`).

Podemos crear matrices del siguiente modo:

```
> matrix(1:12)
> m1 <- matrix(1:12, nrow=4)
> m2 <- matrix(1:12, ncol=4)
> matrix(c(1,-1,3,4,0,6), ncol=3,nrow=2)
> matrix(c(1,-1,3,4,0,6), ncol=3,nrow=2,byrow=T)
```

Los datos que contiene una matriz, igual que ocurría para los vectores, deben ser todos del mismo tipo (numérico, carácter o lógico).

Funciones con matrices	
`dim()`	Devuelve las dimensiones de una matriz
`dimnames()`	Devuelve el nombre de las dimensiones de una matriz
`colnames()`	Devuelve el nombre de las columnas de una matriz
`rownames()`	Devuelve el nombre de las filas de una matriz
`mode()`	Devuelve el tipo de datos de los elementos de una matriz
`length()`	Devuelve el número total de elementos de una matriz
`is.matrix()`	Devuelve T si el objeto es una matriz, F si no lo es
`[ , ]`	Accede a elementos dentro de la matriz
`apply()`	Aplica una función sobre las filas o columnas de una matriz
`cbind()`	Añade una columna a una matriz dada o crea una matriz leyendo los elementos por columnas
`rbind()`	Añade una fila a una matriz dada o crea una matriz leyendo los elementos por filas
`t()`	Traspone la matriz dada

Un ejemplo del uso de estos comandos es el siguiente:

```
> x <- cbind(x1=3, x2=c(4:1,2:5)); x
> y <- cbind(x, c(1:8)); y
> apply(x, 2, sort)
> dimnames(x)[[1]] <- letters[1:8]
> x
> col.sums <- apply(x, 2, sum)
> row.sums <- apply(x, 1, sum)
> rbind(cbind(x, Rtot = row.sums), Ctot = c(col.sums,
sum(col.sums)))
```

El producto matricial se realiza mediante el operador $\%*\%$, mientras que $*$ realiza el producto componente a componente.

```
> m1 %*% m2
```

Ejercicio 3. Crear una matriz con cuatro alumnos (de este curso) y su edad, año de nacimiento y número de teléfono. Deberá aparecer el nombre de la columna ("edad", "año de nacimiento", "teléfono") y el nombre de la fila, que será el nombre del alumno al que corresponden los datos.

Los comandos para manejar "arrays" son similares a los que manejan matrices. Su definición general es de la forma

```
> array(data, dim=length(data))
```

donde data son los datos que forman el *array* y *dim* es un vector con las dimensiones del array.

Un ejemplo de $3-$array de formato $3 \times 2 \times 4$ es

```
> array(c(1:24),c(3,2,4))
```

Los "data.frames" u hojas de datos son listas que generalizan a las matrices, en el sentido de que las columnas (variables a menudo) pueden ser de diferente tipo entre sí (no todas numéricas, por ejemplo). Sin embargo, todos los elementos de una misma columna deben ser del mismo tipo.

Al igual que las filas y columnas de una matriz, todos los elementos de un data.frame deben tener la misma longitud. De este modo, pueden usarse funciones tales como `dimnames, dim, nrow` sobre un data.frame como si se tratara de una matriz y se pueden imprimir en forma matricial.

Para crear un objeto de este tipo, usaremos la función `data.frame( )` La manera más sencilla de construirlo es utilizar la función `read.table( )` para leer la hoja de datos desde un archivo, por ejemplo de texto o excel. Detallaremos el uso de este comando más adelante.

4. Organización del trabajo

Los datos suelen leerse desde archivos externos y no teclearse de modo interactivo. Las capacidades de lectura de archivos de R son sencillas y sus requisitos son bastante estrictos, pues se presupone que el usuario es capaz de modificar los archivos de datos con otras herramientas, como editores de texto, para ajustarlos a las necesidades de R. Generalmente esta tarea es bastante simple.

Si queremos almacenar los datos en hojas de datos, podemos leerlos directamente con la función scan( ). La orden `scan("datos")` lee los datos de un fichero de nombre "datos". También podemos utilizar esta orden para introducir datos desde la propia ventana de comandos. Vamos introduciendo datos, hasta que queramos finalizar, momento en que pulsamos la tecla <intro>. Por ejemplo, si queremos introducir los datos 2, 4 y 6, haremos:

```
> scan()
1: 2
2: 4
3: 6
4:
```

La función `read.table` lee datos desde un archivo, donde aparecen en forma de tabla, y crea un data.frame a partir de ellos. Su sintaxis es `read.table("file", head=T)`, donde file es el nombre del archivo que contiene los datos. Por defecto el separador entre campos es el espaciado, el carácter que denota los decimales es el punto "." y `head=T` indica que la primera línea de los datos contiene los nombres de las variables.

Por ejemplo, si escribiesemos `read.table("datos.dat",sep='',')` estaríamos leyendo los datos del archivo datos.dat, teniendo en cuenta que están separados por comas y que no hay cabecera.

En la distribución de R se incluyen algunos objetos con datos, tanto en la biblioteca base como en el resto. El siguiente comando

```
> library(car)
```

carga el paquete o librería car.

Para obtener una lista de todos los datos disponibles de las bibliotecas conectadas en un determinado momento, basta con teclear

```
> data()
```

y para cargar unos datos concretos `data(nombre)`, por ejemplo

```
> data(AMSsurvey)
```

Ahora, podemos comprobar que el conjunto de datos AMSsurvey se trata efectivamente de una varible de tipo data.frame, ver su descripción y ver su contenido:

```
> is.data.frame(AMSsurvey)
> help(AMSsurvey)
> AMSsurvey
```

La orden `ls()` proporciona una lista de todos los objetos creados en R durante la sesión de trabajo.

```
> ls()
```

Para escribir los datos "x" en el archivo "datos", podemos usar la función `write`.

```
> write(x, file="datos", sep=, dec=".")
```

La orden `write.table` escribe el contenido de la variable `x` en el archivo `mas_datos` tras haberlo convertido en un `data.frame` si no lo era.

```
> write.table(y, file="mas_datos", sep=, dec=".")
```

5. Representación gráfica de funciones

Para representar gráficas de funciones usamos la orden `plot()`. Por ejemplo,

```
> plot(sin, -pi, 2*pi)
```

Otro uso de esta función es producir un diagrama de puntos de y frente a x. Por ejemplo,

```
> x <- seq(-3,3,0.01)
> plot(x, dnorm(x), type="p", col=2, main="Desindad
N(0,1)")
```

Ahora no cierres la ventana gráfica.

La orden `type` sirve para cambiar la representación de los puntos, que por defecto se unen mediante líneas (`p` corresponde a puntos, `l` a líneas y `b` a puntos unidos por líneas). El color de la gráfica se puede definir mediante `col` y la orden `main` permite añadir un título a la gráfica. Además, las funciones `lines()` y `points()` permiten superponer líneas a un gráfico ya existente. La primera pinta líneas y la segunda puntos.

```
> lines(x,dnorm(x,1),col=3)
> points(x,dnorm(x,-1),col=4)
```

6. Estructuras de programación

R permite escribir comandos en la misma línea, utilizando el carácter ";". Los comandos pueden agruparse entre llaves: `{comando1; comando2; comando3; ...}`

Para la ejecución condicional de sentencias, la orden que usaremos es `if` con la siguiente sintaxis: `if (expr1) expr2 else expr3`, donde `expr1` debe producir un valor lógico, de modo, que, si éste es verdadero (`T`), se ejecuta `expr2` y, si es falso (`F`), se ejecuta `expr3`. Por ejemplo

```
> a <- exp(7)
> if (a >1000) {.ª es mayor que mil"} else {.ª es menor o
igual que mil"}
```

Para crear un bucle repetitivo *for*, la sintaxis es la siguiente: `for (nombre in expr1) expr2`, donde `nombre` es la variable de control de la iteración, `expr1` es un vector (usualmente de la forma `m:n`) y `expr2` es la expresión a ejecutar, que se evalúa repetidamente. Por ejemplo,

```
> for(i in 1:10) {print(i)}
```

Veamos un ejemplo combinado `for` e `if`. Vamos a crear dos listas; una para guardar los números pares de 1 a 10, y otra para los impares:

```
> n <- 10
> pares <- c()
> impares <- c()
> for(i in 1:n) {if (i%%2==0) pares <- c(pares,i) else
impares <- c(impares,i) }
> pares; impares
```

En cuanto a *while*, la sintaxis es como sigue: `while (condicion) expr`, que ejecuta `expr` mientras que se cumpla condición. Por ejemplo,

```
> n <- 1
> while (n <= 10) {print(n); n=n+1}
```

Ejercicio 1. Encontrar cuál es el mayor número entero cuyo cuadrado no excede de 1000, utilizando la orden `while`.

7. Funciones definidas por el usuario

R permite crear nuevas funciones, que se pueden utilizar a su vez en expresiones posteriores. La forma general de hacerlo es la siguiente: `nombre <- function (arg1 , arg2, .....) expr` donde `nombre` es el nombre que le damos a la función, `arg1, arg2,...` contienen los parámetros de la misma, y `expr` es una expresión que utiliza los argumentos de la función para calcular un valor.

Las variables definidas dentro del cuerpo de una función son locales, y desaparecen al terminar la ejecución de la función.

Si queremos fijar alguno de los argumentos, simplemente le asignamos el valor usando el signo "=" es decir, `nombre <- function(arg1, arg2=val, ...) expr`

Por ejemplo, supongamos que queremos definir una función que calcule el valor medio de dos números:

```
> media2 <- function(x,y) return((x+y)/2)
```

El comando `return` le dice a la función creada qué valor debe devolver. No es necesario que toda función contenga necesariamente la instrucción `return`; hay muchas funciones que efectúan distintas tareas sin necesidad de devolver un resultado.

Ejercicio 1. Escribir una función que intercambie los valores de dos variables, denominadas x e y.

Ejercicio 2. Escribir una función que genere los n primeros términos de la serie de Fibonacci: $1, 1, 2, 3, 5, 8, 13, \ldots$

Ejercicio 3. Escribir una función que escriba *Hola* si el valor de la variable es menor o igual que 0 y escriba *Adios* si es mayor. Evaluar la función para los valores 5 y -5.

8. Obteniendo ayuda

R dispone de un sistema de ayuda que se puede invocar en cualquier momento desde la ventana de comandos.

En Windows, en la barra del menú que aparece al iniciar el programa se halla la opción *help* que se estructura de la siguiente manera:

Console	Ayuda sobre el uso de las teclas y sus combinaciones en R
R language (standard)	Proporciona ayuda sobre funciones concretas
R language (html)	Arranca un entorno de ayuda completo en formato html
Manuals	Da acceso al manual de referncia de R en formato pdf
Apropos	Da información sobre las funciones relacionadas con una dada
About	Informa de la versión actual de R

Algunos ejemplos en línea mediante comandos[1]:

```
> help()
```

muestra una ventana de ayuda general sobre R.

```
> help.start()
```

arranca un manual de ayuda completo en formato html, utilizando el navegador de internet predeterminado.

```
> help(log)
```

muestra la ayuda sobre la función logaritmo.

```
> apropos("plot")
```

[1] En algunos casos, la ayuda se muestra en la misma ventana de R, para salir de ayuda basta pulsa la tecla q

muestra las funciones u objetos que incluyen la cadena plot en su nombre.

```
> help.search("plot")
```

muestra la ayuda sobre las funciones que incluyen la cadena plot.

TUTORIAL 3

Breve introducción a wxMAXIMA

1. Introducción

MAXIMA es un sistema para la manipulación de expresiones simbólicas y numéricas, incluyendo diferenciación, integración, expansión en series de Taylor, transformadas de Laplace, ecuaciones diferenciales ordinarias, sistemas de ecuaciones lineales, y vectores, matrices y tensores. MAXIMA produce resultados con alta precisión usando fracciones exactas y representaciones con aritmética de coma flotante arbitraria. Adicionalmente puede representar gráficamente funciones y datos en dos y tres dimensiones.

El código fuente de MAXIMA puede ser compilado sobre varios sistemas incluyendo Windows, Linux y MacOS X. El código fuente para todos los sistemas y los binarios precompilados para Windows y Linux están disponibles en el Administrador de archivos de SourceForge (http://sourceforge.net/projects/maxima/files/).

MAXIMA es un descendiente de Macsyma, un sistema de álgebra computacional desarrollado a finales de 1960 en el instituto tecnológico de Massachusetts (MIT). Está basado en el esfuerzo voluntario de una comunidad de usuarios activa, gracias a la naturaleza del *open source*. Macsyma fue revolucionario es sus días y muchos sistemas posteriores, tales como Maple y Mathematica, estuvieron inspirados en él.

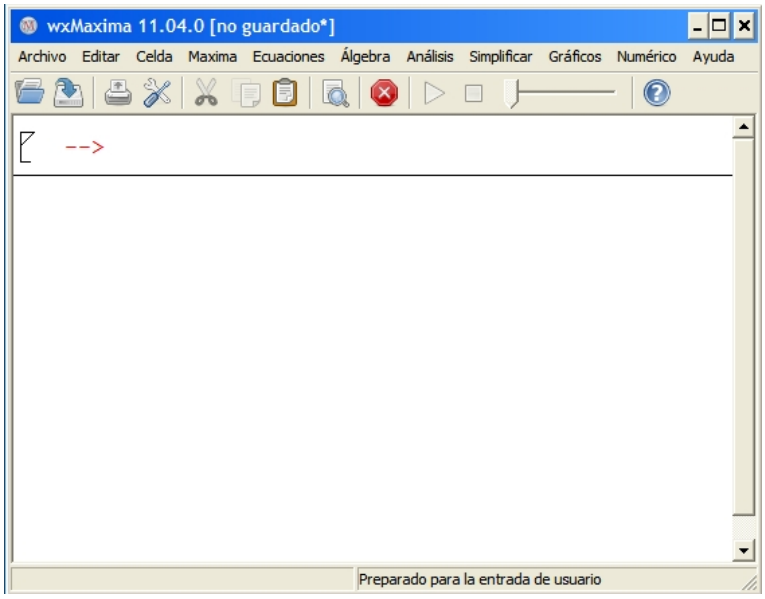
La rama MAXIMA de Macsyma fue mantenida por William Schelter desde 1982 hasta su muerte en 2001. En 1998 él obtuvo permiso para liberar el código fuente bajo la licencia pública general (GPL) de GNU.

MAXIMA está en constante actualización, corrigiendo y mejorando el código y la documentación. Existe una lista de correo de MAXIMA donde se pueden hacer sugerencias y consultar dudas (http://maxima.sourceforge.net/maximalist.html).

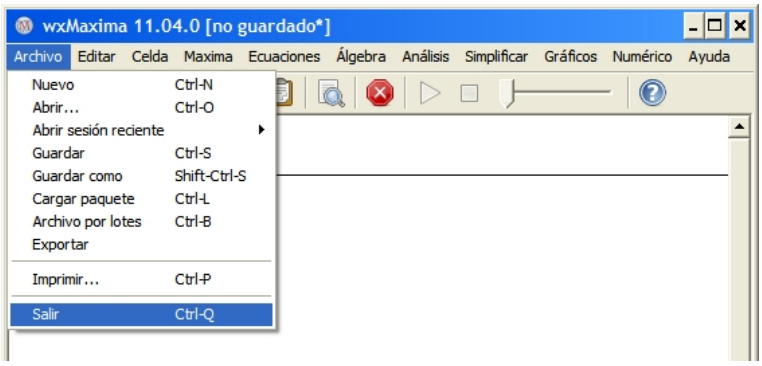
2. Uso como calculadora científica

Se inicia MAXIMA haciendo clic sobre el icono de acceso directo una vez instalado el programa. Al abrirse, MAXIMA despliega (no necesariamente en

todas las versiones) alguna información importante acerca de la versión que se está usando y un prompt.



Para finalizar una sesión en MAXIMA se emplea el comando "`quit();`" o también en la barra de herramientas superior pulsando en *Archivo ->Salir*, tal y como se indica en la figura:



Si se teclea la operación a realizar y se pulsa la tecla *shift* y *enter* simultáneamente, aparecerá en pantalla el resultado:

```
(%i1) 2+3;
```

Las entradas van numeradas por orden de actuación (`%i`n) (`input`) y la salida correspondiente lleva la misma numeración (`%o`n) (`output`). Esta

numeración facilita hacer referencia a resultados anteriores cuando se han realizado ya varias operaciones.

En el ejemplo anterior ($2+3=5$) se utiliza MAXIMA como una simple calculadora. Los símbolos de las operaciones básicas son los habituales para el producto, la potencia y el cociente:

```
(%i2) 2*3;
(%i3) 2^3;
(%i4) 2/3;
```

Para obtener la aproximación decimal de un resultado se selecciona en la barra de herramientas superior *Numérico ->A Real*, o bien se usa la orden `float(expr)`:

```
(%i5) float(2/3);
```

Siempre que sea necesario se introducen paréntesis para que las operaciones se realicen en el orden correcto. Por ejemplo:

```
(%i6) 2+3^2;
(%i7) (2+3)^2;
```

Para hacer referencia a un resultado anterior se utiliza el símbolo `%i`n ó `%o`n. Por ejemplo, en nuestro caso, `%i6` es 11, de modo que:

```
(%i8) %i6-4;
```

Para hacer referencia al resultado inmediatamente anterior basta escribir %, sin ningún número; en nuestro caso el último input ha sido (`%i8`), de modo que:

```
(%i9) %^2;
```

También es posible operar con números complejos, teniendo en cuenta que la unidad imaginaria i habrá que introducirla como `%i` (no confundir con la numeración de las entradas):

```
(%i10) (2*%i)*%i;
```

3. Variables y funciones

Se puede asignar un valor a una variable. La sintaxis es `nombre:expresión matemática`. Por ejemplo, asignamos a x el valor $2\cdot 3$, a y el valor 2 y posteriormente calculamos $x\cdot y$:

```
(%i11) x:3*2; y:2;
(%i12) x; y; x*y;
```

Si desarrollamos $(x+3y)^5$, también se puede asignar este resultado a una variable (darle un nombre)

```
(%i13) h:expand((x+3y)^5);
```

Podemos referirnos a esa expresión con el nombre dado, de modo que, por ejemplo:

```
(%i14) h/(45+z);
```

Para borrar el valor asignado a la variable seleccionamos en la barra de herramientas pulsamos *Maxima ->Borrar variable*. Si ahora escribimos el nombre de la variable:

```
(%i15) h;
```

veremos que la variable `h` no tiene asignado ningún valor.

Hay que notar que al seleccionar borrar variable aparece por defecto "`all`". Es necesario escribir el nombre de la variable que se desea borrar si no se quiere borrar todas.

MAXIMA puede manejar funciones de una o varias variables. Estas son algunas de las funciones que están definidas en el programa:

`sqrt(x)`	raíz cuadrada		$n!$	factorial
`exp(x)`	exponencial(e^x)		abs(x)	valor absoluto
`sin(x)`	seno		`asin(x)`	arco seno
`cos(x)`	coseno		`acos(x)`	arco coseno
`tan(x)`	tangente		`atan(x)`	arco tangente
`log (x)`	logaritmo neperiano			

Es importante notar:

- que el argumento de las funciones siempre se escribe entre paréntesis;
- por defecto las funciones trigonométricas operan en radianes

Para definir una función propia f de variable x se utiliza "`:=`". Por ejemplo:

```
(%i16) f(x) := x^2;
```

de modo que ahora podemos aplicar f a cualquier número:

```
(%i17) f(3);
```

Los nombres asignados a funciones se pueden borrar con la opción "borrar función" de forma análoga a como se borran variables.

MAXIMA manipula constantes numéricas, por ejemplo, el número e se escribe `%e` y el número π se escribe `%pi`.

```
(%i18) log(%e);
(%i19) sin(%pi/2);
(%i20) atan(1);
```

Ejercicio 1. Obtén el resultado exacto y decimal aproximado:

$$\frac{1+\sqrt{2}}{1-\sqrt{5}}, \quad \cos^2(\pi), \quad e^{1+i}.$$

Ejercicio 2. Obtén el resultado exacto y decimal aproximado:

$$\frac{2}{10} - \frac{3}{10}.$$

Ejercicio 3. Asigna a una variable x el número uno. Sobre esa variable calcula $x = 2x$. Repetir esta operación hasta obtener el valor de 256.

Ejercicio 4. Calcula

$$\left(\frac{e^{1+i}-e^{-1-i}}{2i}\right)^4 + \left(\frac{e^{1+i}+e^{-1-i}}{2}\right)^4 + \\ +2\left(\frac{e^{1+i}-e^{-1-i}}{2i}\right)^2\left(\frac{e^{1+i}+e^{-1-i}}{2}\right)^2.$$

Para ello asigna a dos variables el interior de los paréntesis y opera con ellas.

4. Cálculo simbólico

Una de las mayores utilidades de MAXIMA es que permite realizar no sólo cálculo numérico ($3+62-1=64$), sino también cálculo simbólico ($3x - x + 2 = 2 + 2x$). Veamos algunos comandos para operar con expresiones algebraicas:

`expand(expresión)`	Desarrolla los productos y potencias que figuran en "expresión"
`factor(expresión)`	Factoriza "expresión"
`ratsimp(expresión)`	Simplificación racional de "expresión"

Veamos algunos ejemplos:

```
(%i21) expand((a+b)^3);
(%i22) factor(x^2-4*x+4);
(%i23) ratsimp(1/(x+2)+1/(x-2));
```

Estas operaciones también pueden hacerse a partir de la barra de herramientas pulsando *Simplificar ->Expandir expresión ...*

MAXIMA permite calcular límites de funciones utilizando el comando `limit(expr, x, val, dir)` que calcula el límite de "expr" cuando la variable "x" tiende al valor "val" desde la dirección "dir" (`plus`: por la derecha, `minus`: por la izquierda). Por ejemplo:

```
(%i24) limit(1/x,x,0);
(%i25) limit(1/x,x,0,minus);
(%i26) limit(1/x,x,0,plus);
```

Si el límite no existe la respuesta es "`und`", es decir, indeterminado. Por ejemplo:

```
(%i27) f(x) := x/abs(x);
(%i28) limit(f(x),x,0);
(%i29) limit(f(x),x,0,plus);
(%i30) limit(f(x),x,0,minus);
```

También se puede calcular el límite utilizando la barra de herramientas: *Análisis ->Calcular límite ...*

Ejercicio 1. Calcula el límite de la función $\frac{e^{1/x}}{x}$ cuando x tiende a 0.

Para derivar funciones se utiliza el comando `diff(expr, x, n)` que deriva "expr" respecto a la variable x, hasta orden n.

```
(%i31) diff(sin(x),x,1);
(%i32) diff(atan(x),x,1);
(%i33) diff(atan(x),x,2);
```

O directamente a partir de la barra de herramientas: *Análisis ->Derivar...*

Para integrar se utiliza el comando `integrate(expr,x)` y si se trata de una integral definida se especifican los límites de integración de la siguiente forma: `integrate(expr,x,a,b)`. Por ejemplo:

```
(%i34) integrate(sin(x)*cos(x), x);
(%i35) integrate(sin(x)*cos(x), x, 0,%pi/2);
```

O directamente a partir de la barra de herramientas: *Análisis ->Integrar...*

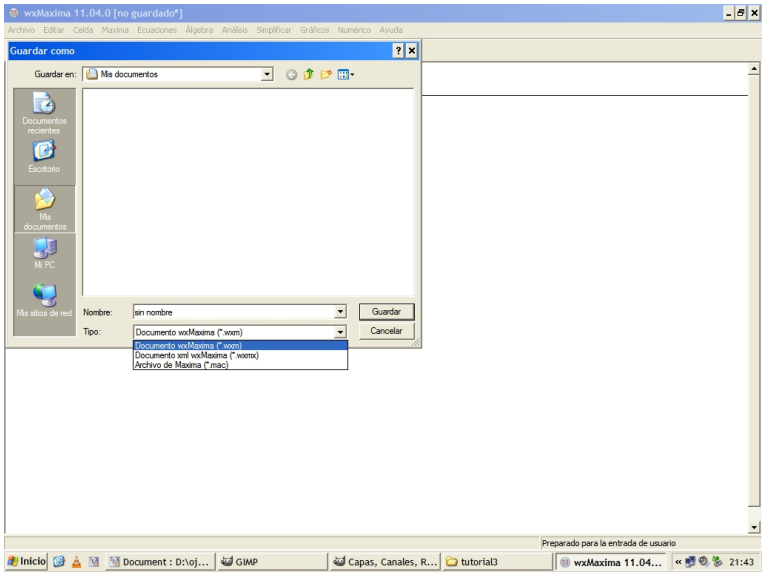
Ejercicio 2. Calcula la función derivada y la integral de $e^{-x}\cos(x)$.

5. Organización del trabajo

Si deseas guardar el trabajo que realizas en una sesión MAXIMA puedes hacerlo a través de la barra de herramientas.

Es posible guardarlo con la extensión .wxm o .mac. En cualquier caso, los archivos contendrán una transcripción completa y literal de todos las órdenes que teclean en el ordenador y que no son borradas durante la sesión. En el caso de los archivos .vxm y .max, se pueden editar con cualquier editor de texto. También podrás ejecutarlos de nuevo con MAXIMA abriendo el archivo correspondiente con la barra de herramientas.

Al salir de una sesión de MAXIMA o reiniciar el programa, los valores de todas las variables se pierden.



Ejercicio 1.

1. Abre una sesión en MAXIMA y ejecuta alguna operación. Guarda el archivo con el nombre `prueba`.
2. Cierra la sesión y vuelve a cargar el archivo `prueba`. Ejecuta de nuevo las operaciones.

6. Vectores, matrices y funciones con matrices

Para manejar un vector, se introducen entre corchetes sus coordenadas separadas por comas.

Las operaciones básicas (suma de vectores, resta de vectores y multiplicación de un vector por un escalar) se realizan elemento a elemento. El producto escalar se designa con un punto. Algunos ejemplos son:

```
(%i36) [2,3]+[1,0];
(%i37) [3,2,1].[-1,0,5];
(%i38) u:[a,b];
(%i39) u-v;
(%i40) u.v;
```

La operación "*elevar a*" se lleva a cabo también elemento a elemento si los operandos son un escalar y un vector o un vector y un escalar.

```
(%i41) v:[2,3]; v^3; 3^v;
```

Las matrices se introducen con el comando `matrix()`, cuyo argumento son los elementos de cada fila entre corchetes y separados por comas. Por ejemplo:

```
(%i42) matrix([1,2,3],[4,5,6]);
```

También se pueden introducir a partir de la barra de herramientas: *Álgebra ->Introducir matriz...*

Para generar matrices diagonales podemos usar el comando `ident(n)` devuelve la matriz identidad de orden n o bien el `diagmatrix(n,x)` que devuelve la matriz diagonal de orden n y elemento x, es decir, x por la matriz identidad de orden n. También podemos utilizar el comando `diag(d1, ..., dn)` para definir una matriz diagonal con los elementos `d1, ..., dn` en su diagonal principal.

```
(%i43) ident(4);
(%i44) diagmatrix(3,25);
(%i45) diagmatrix(4,k);
(%i46) diagmatrix(1,2,3);
```

Las operaciones básicas (suma de matrices, resta de matrices y multiplicación de una matriz por un escalar) se llevan a cabo elemento a elemento. Si $\lambda \in \mathbb{R}$ y A es una matrix, las operaciones λ^A y A^λ se calculan elemento a elemento. El producto matricial de dos matrices A y B se obtiene como $A.B$. Dada la matriz A, el producto $A.A$ se obtiene como $A\text{^^}2$ y la inversa, si existe, como $A\text{^^}(-1)$.

```
(%i47) x:matrix([17,3],[-8,11]);
(%i48) y:matrix([%pi,%e],[a,b]);
```

Operaciones básicas:

```
(%i49) x+y;
(%i50) x-y;
(%i51) x*y;
```

Multiplicación de matrices:

```
(%i52) x.y;
(%i53) y.x;
```

Una matriz elevada al exponente -1 con el operador de potencia no conmutativa equivale a la matriz inversa, si existe:

```
(%i54) x^^(-1);
(%i55) x.x^^(-1);
```

El comando `row(A,i)` devuelve la fila i de la matriz A :

```
(%i56) row(A,3);
```

El comando `col(A,i)` devuelve la columna i de la matriz A :

```
(%i57) col(A,5);
```

El comando `submatrix`(i_1, ..., i_m, A, j_1, ..., j_n) devuelve una matriz en la que se han eliminado las filas i_1, ...,i_m de la matriz A y columnas j_1, ..., j_n.

```
(%i58) submatrix(1,4,A,1,2,3);
```

Otros comandos importantes son:

`mat_trace(M)` calcula la traza de la matriz M.
`minor(M, i, j)` calcula el menor ij de la matriz M.
`rank(M)` calcula el rango de la matriz M.
`transpose(M)` calcula la matriz transpuesta de la matriz M.
`triangularize(M)` devuelve la forma triangular superior de la matriz M, obtenida por eliminación gaussiana.

```
(%i59) A:matrix([1,3,2,-1,0], [2,7,-5,6,4], [2,6,4,-2,0],
[3,10,-3,5,4]);
(%i60) transpose(A);
(%i61) rank(A);
(%i62) triangularize(A);
```

El comando `eigenvalues(M)` devuelve una lista con dos sublistas. La primera sublista la forman los valores propios de la matriz M y la segunda sus multiplicidades correspondientes.

El comando `eigenvectors(M)` devuelve una lista de listas, la primera de las cuales es la salida de `eigenvalues` y las siguientes son los vectores propios de la matriz asociados a los valores propios correspondientes.

```
(%i63) A:matrix([1,0,0],[0,1,0],[0,0,4]);
(%i64) eigenvalues(A);
(%i65) eigenvectors(A);
```

Para generar una tabla de valores se utiliza el comando `makelist`. Concretamente el comando `makelist(expr, i,` i_0`,` i_1`)` devuelve una lista con los valores que va tomando "expr" para cada valor de i comprendido entre i_0 e i_1, a saltos de tamaño 1 y el comando `makelist([expr1,expr2], i, i0, i1)` devuelve una lista con parejas de los valores que toma "expr1" y "expr2".

Por ejemplo, dada una función $f(x)$

```
(%i66) f(x) := 2*x/(x^2+4);
```

se puede calcular el valor $f(x)$ en $-2, -1, 0, 1$ y 2 simultáneamente

```
(%i67) makelist(f(x),x,-2,2);
```

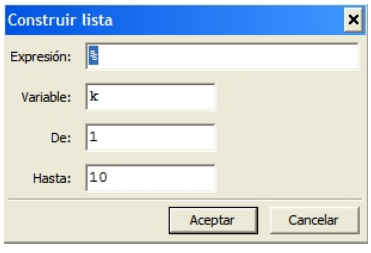
También se puede escribir:

```
(%i68) makelist([x,f(x)],x,-2,2);
```

para obtener los puntos de absisas $-2, 1, 0, 1$ y 2 y ordenadas $f(-2)$, $f(-1)$, $f(0)$, $f(1)$ y $f(2)$, respectivamente.

Es posible introducir una lista a partir de la barra de herramientas: *Álgebra ->Construir lista ...*

Aparece una ventana donde se introduce la expresión en función de una variable. Por ejemplo:



y el resultado es

```
(%i69) makelist(k^2+1,k,1,10);
```

Si no se desea que la variable tome valores de 1 en 1 hay que escribir explícitamente la lista de valores que toma, como un argumento del comando `makelist`

```
(%i70) makelist(k^2+1,k,[2,4,6]);
```

o introducir previamente la lista de valores que tomará y hacer referencia a ella con el nombre que se le haya dado. Por ejemplo

```
(%i71) salto:makelist(k/2,k,3,6);
(%i72) makelist(k^2+1,k,salto);
```

Ejercicio 1.

1. Crear un vector con los números naturales del 1 al 10.
2. Calcular la raíz cuadrada de cada uno de esos números.
3. Crear un vector con los números naturales del 1 al 1000.
4. Obtener un vector con el cociente de dividir cada número natural entre 1 y 1000 entre 7.

Ejercicio 2. Crea la matriz

$$A = \begin{pmatrix} 4 & 1 & 2 \\ 1 & 5 & 0 \\ 2 & 0 & 7 \end{pmatrix}$$

Llama B a la submatriz de 2×2 obtenida de A tomando sus filas $1, 3$ y columnas $1, 2$. Calcula el determinante de B.

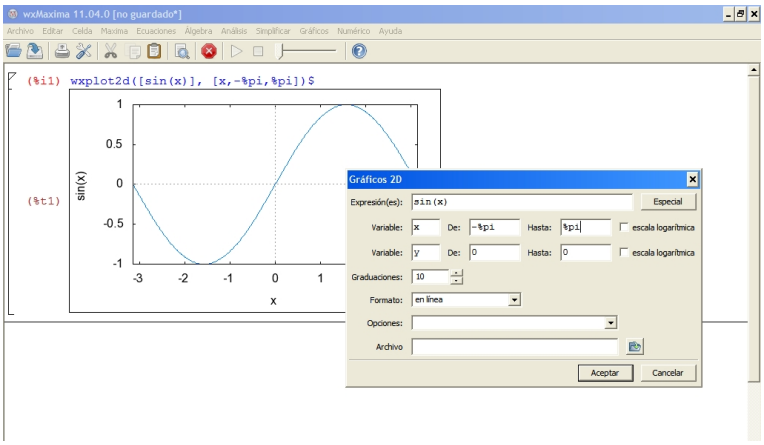
7. Representación gráfica de funciones

Para representar gráficamente una función se utiliza el comando `wxplot2d` que responde a la sintaxis `wxplot2d(expr,[x,xmin,xmax])`, donde "expr" es la función que se va a representar y $[x, min, max]$ es el intervalo de x en el que se dibuja la gráfica de la función. Por ejemplo:

```
(%i73) wxplot2d([sin(x)], [x-%pi,%pi])$
```

Si se escribe `plot2d(expr,[x,xmin,xmax])` la gráfica se muestra en una nueva ventana usando el programa gnuplot.

También se puede hacer gráficas a partir de la barra de herramientas en la parte inferior de la pantalla. En la opción de formato se elige "en línea" si se desea que la gráfica aparezca en la misma ventana de trabajo.



Si se ha definido una función previamente se puede utilizar el nombre que se le dio a la función en el comando `wxplot2d`.

El comando `plot` admite algunas opciones para especificar, por ejemplo, el rango de valores tanto del eje x como del eje y, el punto que se elige como origen de coordenadas, la escala de los ejes, etc. En el siguiente ejemplo se muestra la gráfica de la función exponencial con el eje y en escala lineal en la primera y en escala logarítmica en la segunda,

```
(%i74) wxplot2d([exp(x)],[x,-2,2])$
(%i75) wxplot2d([exp(x)],[x,-2,2],[logy]);
```

Se pueden también dibujar varias funciones en el mismo gráfico. Por ejemplo:

```
(%i76) wxplot2d([sin(x),cos(x)],[x,-%pi,%pi]);
```

Para representar la gráfica de una función dada en coordenadas paramétricas se introduce la clave `parametric`, seguida de dos expresiones, x_{expr}, y_{expr}, que dependen de una única variable cuyo rango se especifica de la forma $[var, min, max]$. El gráfico mostrará los pares $[x_{expr}, y_{expr}]$ según var varía de min a max. Por ejemplo:

```
(%i77) wxplot2d([parametric,sin(t),cos(t)],[t,0,2*%pi]);
```

Para representar una lista de puntos

```
(%i78) puntos:makelist([x,x^2],x,-3,3)$
(%i79) wxplot2d([discrete,puntos],[style,points]);
```

Ejercicio 1. Dibuja la gráfica de la función $y = \sin(x)$ entre los puntos 0 y π. Añade la gráfica del polinomio $y = x - x^3/6 + x^5/120$.

8. Estructuras de programación

MAXIMA dispone de una serie de operadores lógicos básicos que son fundamentales para la programación. Veamos algunos ejemplos:

Operación	Símbolo	Tipo
menor que	<	operador relacional infijo
menor o igual que	<=	operador relacional infijo
igualdad (sintáctica)	=	operador relacional infijo
negación de =	#	operador relacional infijo
igualdad (por valor)	equal	operador relacional infijo
negación de equal	notequal	operador relacional infijo
mayor o igual que	>=	operador relacional infijo
mayor que	>	operador relacional infijo
y	and	operador lógico infijo
o	or	operador lógico infijo
no	not	operador lógico prefijo

Usemos el operador `if` para comprobar si dos variables tienen el mismo valor o no:

```
(%i80) a:3;
(%i81) b:2;
(%i82) if a=b then c:1 else c:0;
```

Con la sentencia anterior el resultado será 1 si son iguales y 0 si son distintas.

Usemos el operador `for` para repetir una instrucción un determinado número de veces:

1. `for` i`:`i_0 `step` k `thru` i_1 `do` *cuerpo*. Con este comando, el índice i recorre los valores desde i_0 hasta i_1 a pasos de tamaño k. En cada paso se ejecuta la sentencia que aparezca en *cuerpo*. Por ejemplo:

```
(%i83) a:[2,3,1,4,2];
(%i84) for i:2 thru 5 do display(a[i]-a[i-1]);
```

2. `for` i`:`i_0 `step` k `while` *condición* `do` *cuerpo* . Con este comando, mientras que se verifique la *condición*, el índice i recorre los valores desde i_0 a pasos de tamaño k y en cada paso se ejecuta la la sentencia que aparezca en *cuerpo*. Por ejemplo:

```
(%i85) a:[2,3,1,4,2,8,9];
(%i86) for i:1 thru 5 while (a[i]<=3 and i<=5) do
display(a[i]);
```

3. `for` i:i_0 `step` k `unless` *condición* i_1 `do` *cuerpo* Con este comando, hasta que se verifique la *condición*, el índice i recorre los valores desde i_0 a pasos de tamaño k y en cada paso se ejecuta la sentencia que aparezca en *cuerpo*. Por ejemplo:

```
(%i87) a:[2,3,1,4,2,8,9];
(%i88) for i:1 unless (a[i] >= 7 or i>4) do
display(a[i]);
```

Ejercicio 1. Crea una función que reciba un vector y devuelva la suma de sus componentes, usando un bucle para ello. Comprueba el resultado.

Ejercicio 2. (Para valientes) Crea una función que reciba un vector de números naturales y devuelva uno formado por aquellos que sean primos.

Parte 2

Prácticas

Sistemas de ecuaciones lineales

En esta práctica analizaremos los tipos de sistemas de ecuaciones lineales en función de sus soluciones. El objetivo es aprender a discutir y resolver sistemas de ecuaciones lineales. Discutir un sistema consiste en averiguar si tiene o no solución, y si la tiene, saber si es única o no. Resolver un sistema es calcular todas las soluciones que tenga.

Prerrequisitos: Ninguno.

PINCHA en el icono del programa para descargar el material de la práctica (se abrirá un navegador).



Matrices, inversas y determinantes

En esta práctica se espera que el alumno trabaje con la aritmética de las matrices, que explore las propiedades de las matrices inversas utilizando como principal herramienta la expresión de las operaciones elementales por filas como productos por la izquierda de matrices elementales, y por último que experimente con las propiedades de los determinantes de las matrices. De modo opcional, se presentan las matrices de Hilbert para comprobar su "singularidad".

Prerrequisitos: Conocimientos básicos sobre matrices y sus operaciones, la capacidad de transformar una matriz en su forma escalonada por filas mediante operaciones elementales.

PINCHA en el icono del programa para descargar el material de la práctica (se abrirá un navegador).



Diagonalización

En esta práctica se espera que el alumno aprenda a calcular y a utilizar los autovalores y autovectores de una matriz.

Prerrequisitos: Saber calcular el polinomio característico, los autovalores y autovectores de una matriz de orden 3*, familiaridad con los criterios para determinar si una matriz es diagonalizable.*

PINCHA en el icono del programa para descargar el material de la práctica (se abrirá un navegador).



Polinomios: raíces y factorización

En esta práctica introduciremos la notación y terminología de los polinomios en una variable, y exploraremos el concepto de raíz de un polinomio y su relación con la factorización.

Prerrequisitos: Ninguno.

PINCHA en el icono del programa para descargar el material de la práctica (se abrirá un navegador).



Funciones polinómicas

En esta práctica exploraremos las propiedades de la funciones polinómicas, dedicando especial interés a las funciones constantes, lineales y cuadráticas.

Prerrequisitos: Haber realizado las prácticas sobre polinomios y sobre sistemas de ecuaciones lineales.

PINCHA en el icono del programa para descargar el material de la práctica (se abrirá un navegador).



Sucesiones y límites de funciones

En cualquier asignatura de cálculo el estudio de sucesiones y sus límites, tanto de números como de funciones, es fundamental. El objetivo de esta práctica es aprender a definir sucesiones de números reales y calcular su límite con MATLAB, así como con sucesiones de funciones. Además, en esta práctica se utilizará la representación gráfica de funciones para facilitar que el alumno visualice el concepto de límite.

Prerrequisitos: ninguno.

PINCHA en el icono del programa para descargar el material de la práctica (se abrirá un navegador).



Representación gráfica de funciones

En esta práctica analizaremos cualitativamente las gráficas de las funciones de una variable (real).

Prerrequisitos: Límites, continuidad y diferenciabilidad. Propiedades de las funciones elementales (racionales, trigonométricas, exponencial y logaritmo).

PINCHA en el icono del programa para descargar el material de la práctica (se abrirá un navegador).



Desarrollos de Taylor

Se pretende que el alumno aprenda a calcular polinomios de Taylor de una función en un punto y los sepa utilizar para calcular valores aproximados de la función.

Prerrequisitos: antes del desarrollo de esta práctica se ha de comprender el concepto de derivada de una función en un punto y los procedimientos para su cálculo. Es necesario también tener unos conocimientos generales sobre el manejo de algún programa informático (Octave, Maxima, R, etc.) que se pueden adquirir con el tutorial correspondiente.

PINCHA en el icono del programa para descargar el material de la práctica (se abrirá un navegador).



Ceros de funciones. Máximos y mínimos

Sea f una función. La resolución de la ecuación $f(x) = 0$ es un problema matemático de gran utilidad pero también de una gran complejidad. Obviamente, esta complejidad depende de la función. En determinadas ocasiones la gráfica puede ayudarnos a resolver este problema, aunque la verdadera clave del éxito radica en un buen conocimiento de las propiedades elementales de la función.

En esta práctica utilizaremos el ordenador para hallar soluciones de las ecuaciones $f(x) = 0$ y $f'(x) = 0$, y visualizar su significado geométrico.

Prerrequisitos: Haber realizado la práctica de Representación gráfica.

PINCHA en el icono del programa para descargar el material de la práctica (se abrirá un navegador).



Ecuaciones diferenciales ordinarias

En esta práctica se hará una pequeña introducción a las ecuaciones diferenciales y los sistemas de ecuaciones diferenciales. Comenzaremos viendo las ecuaciones autónomas con una única variable (de primer orden), estudiando cómo son las soluciones y cómo se comportan. Después veremos los sistemas de ecuaciones lineales en dos variables, realizando un estudio cualitativo.

Prerrequisitos: Funciones de una variable. Autovalores y autovectores.

PINCHA en el icono del programa para descargar el material de la práctica (se abrirá un navegador).



Estadística descriptiva

La Estadística Descriptiva proporciona medidas, gráficos y procedimientos que actúan a modo de guías para extraer y resumir información de un conjunto de datos de cualquier tamaño. Hoy día, los estudios estadísticos son habituales en todos los campos precisamente por este extraordinario poder informativo. En esta práctica mostramos sus herramientas principales: elaboración de tablas y gráficos, y obtención de las medidas resumen de los aspectos más importantes de un conjunto de datos.

Prerrequisitos: Conocimiento de los conceptos elementales de Estadística Descriptiva.

PINCHA en el icono del programa para descargar el material de la práctica (se abrirá un navegador).



Distribuciones de probabilidad

Se pretende que el alumno profundice en el conocimiento de algunos de los principales modelos de distribuciones de probabilidad de variables aleatorias, tanto discretas como continuas, así como en la comprensión del teorema central del límite.

Prerrequisitos: antes del desarrollo de esta práctica se han de manejar los conceptos y procedimientos de cálculo de estadística descriptiva; comprender qué es la distribución de probabilidad de una variable aleatoria y qué significa generar un número aleatorio según una distribución de probabilidad; conocer las distribuciones de probabilidad más importantes, tanto discretas como continuas (Uniforme discreta y continua, Normal, Chi-Cuadrado, t-Student, F-Snedecor); conocer el concepto de muestra y distinguirlo del concepto de población; conocer y saber calcular los principales estimadores (media muestral, cuasivarianza muestral) de los parámetros media y varianza de una población.

PINCHA en el icono del programa para descargar el material de la práctica (se abrirá un navegador).



Inferencia

La Estadística Descriptiva se dedica al análisis y tratamiento de datos. A partir de ellos, resume, ordena y extrae los aspectos más relevantes de la información que contienen. Sin embargo, los objetivos de la Estadística son más ambiciosos. No nos conformamos con describir unos datos contenidos en una muestra sino que pretendemos extraer conclusiones para la población de la que fueron extraídos. A esta última tarea la llamamos Inferencia Estadística. Obtendremos las muestras de forma aleatoria y por tanto necesitaremos la Teoría de la Probabilidad era elaborar nuestros argumentos.

En esta práctica utilizaremos el ordenador para realizar contrastes de hipótesis y construir intervalos de confianza para extraer conclusiones sobre la población a partir de los datos de una muestra.

Prerrequisitos: Haber realizado la práctica de Estadística Descriptiva.

PINCHA en el icono del programa para descargar el material de la práctica (se abrirá un navegador).



Bibliografía

[1] V.J. Bolós, J. Cayetano y B. Requejo. *Álgebra Lineal y Geometría*. Universidad de Extremadura, Manuales UEx 50, 2007.

[2] M.A. Mulero e I. Ojeda. *Matemáticas para primero de Ciencias.* Universidad de Extremadura, Manuales UEx 54, 2008.

[3] J.A. Navarro. *Álgebra Conmutativa Básica.* Universidad de Extremadura, Manuales Unex 19, 1996.

[4] I. Ojeda Martínez de Castilla y J. Gago Vargas. *Métodos Matemáticos para Estadística*. Universidad de Extremadura, Manuales UEx 58, 2008.

[5] A. Quarteroni y F. Saleri. *Cálculo científico con MATLAB y Octave*. Springer-Verlag, Italia, 2006.

[6] The Connected Curriculum Project: Interactive Learning Materials for Mathematics and Its Applications. *http://www.math.duke.edu/education/ccp/*. Department of Mathematics, Duke University, Durham, USA.

ISBN 978-84-694-5697-2

UNIVERSIDAD DE EXTREMADURA

UNIÓN EUROPEA
Fondo Social Europeo

JUNTA DE EXTREMADURA

75