Práctica 1: Introducción a Maxima
MNI, Curso 18/19

1 Acceso a Maxima

En la dirección

http://maxima.sourceforge.net/es/

puedes encontrar información sobre Maxima, incluido el modo en que es posible descargarlo de forma legal y gratuita (se trata de software libre) en un ordenador personal. La última versión disponible a día de hoy es la 5.42.2.

Para trabajar con wxMaxima desde un ordenador ubicado en un aula de la Universidad de Granada, debemos seguir los siguientes pasos:

(i) Conectar el equipo e introducir la opción "General".

(ii) Seleccionar Inicio/Programas/Maxima/wxMaxima.

```
;
```

2 Celdas de entrada y salida

Al cargar Maxima se abre un fichero en blanco. En su interior iremos escribiendo todas las operaciones algebraicas que deseemos realizar. Por ejemplo, podemos pedir a Maxima que calcule el resultado de la sencilla operación 1+1:

```
1+1;
```

que proporciona el valor 2 como resultado.

Para ejecutar una operación en Maxima pulsaremos la tecla Intro (teclado numérico) o simultáneamente las teclas Shift y Enter.

Toda la información de cada archivo se organiza según una serie de celdas. Cada una de ellas aparece con un corchete en su parte izquierda. Fundamentalmente, trabajaremos con dos tipos de celda:

1. Celdas de entrada (in), en las que se introducen datos (en el ejemplo anterior "1+1" es la celda de entrada);
2. Celdas de salida (out), en las que aparecen los resultados de los cálculos efectuados por el programa (en el ejemplo anterior "2" es la celda de salida).

Se puede hacer referencia e incluso operar con la salida que acaba de obtenerse en forma de %, o bien a una concreta mediante %on, donde n es el número de la salida.

Un primer comando de Maxima, que recoge en esencia la filosofía de la sintaxis de todos los de este programa, es print, cuyo funcionamiento ilustran estos ejemplos:

```
print(2·3);
```

```
print("hola");
```

En particular, todas las sentencias utilizan paréntesis para escribir sus argumentos y todas sus letras se escriben minúsculas.

3 Operaciones algebraicas básicas. Redondeo decimal

Los siguientes ejemplos ilustran las operaciones de suma, resta, producto, cociente y potencias de números reales, tanto de forma simbólica como redondeando en base decimal. Observa asímismo que el signo ";" aparece automáticamente al ejecutar cualquier comando u operación. La prioridad entre operaciones es la usual, y puede modificarse mediante el uso de paréntesis.

```
9+2;
```

```
3−5.88408;
```

```
8·(5−2);
```

```
6+2/2;
```

```
(8+2)/2;
```

```
1+5/6−9/72;
```

```
2.0+4/7−8/65;
```

```
2^3.01;
```

Podemos trabajar con la expresión decimal, bien como acabamos de hacer introduciendo alguno de los números en forma decimal, bien mediante la sentencia float, que genera redondeos decimales de números reales:

```
float(sqrt(2));
```

```
float(%pi);
```

```
float(%e);
```

La sentencia anterior admite una variante, bfloat, con la que podemos modificar el número de cifras significativas con las que deseamos trabajar (la letra "b" a la derecha de la salida denota "10^"):

```
fpprec : 20;
```

```
bfloat(%pi);
```

```
bfloat(%e);
```

```
;
```

4 Asignación de valores a variables

Para asignar el valor a una variable se utilizan los dos puntos:

```
x:3;
```

```
y:2;
```

```
x-y;
```

Para eliminar el valor asignado a una variable se usa la orden kill:

```
kill(x);
```

```
x;
```

El argumento all en esta sentencia elimina los valores que tengan asignados todas las variables que se hayan utilizado previamente.

A las variables se les puede asignar igualmente otro tipo de entes matemáticos que presentaremos un poco más adelante (vectores, matrices...).

5 Funciones matemáticas

5.1 Algunas funciones elementales

A continuación se presentan diversas funciones que usaremos a lo largo del curso. Los argumentos de las funciones trigonométricas se expresan en radianes.

```
sin(%pi);
```

```
cos(1.0);
```

```
tan(%pi/4);
```

```
asin(1);
```

```
acos(-1);
```

```
atan(sqrt(3));
```

```
sqrt(2.0);
```

```
log(%e);
```

```
exp(0.98);
```

```
abs(-3.4);
```

```
abs(3-%i);
```

5.2 Definición de funciones, derivadas, integrales y gráficas

Las funciones anteriores, junto con las operaciones elementales de funciones y la composición, permiten trabajar en Maxima con una amplia familia de funciones de sintaxis simple e intuitiva. Pero Maxima cuenta con la posibilidad de definir directamente funciones: basta escribir el nombre de la función, su variable (o variables) entre paréntesis y dos puntos antes del signo de igualdad. Por ejemplo

```
f(x):=-x^2+%e^x+3;
```

A partir de aquí se abre toda una gama de manipulaciones que empieza con unaa simple evaluación en un punto del dominio:

```
f(1);
```

También se puede derivar una función, tantas veces como se desee, en este caso dos:

```
diff(f(x),x,2);
```

Cuando el orden de derivación es uno, puede suprimirse como argumento:

```
diff(f(x),x);
```

Esta sentencia, al igual que otras muchas, está recogida en uno de los menús de la barra principal, en este caso en el de Análisis. Aunque no incidimos en ello en lo sucesivo, por lo sugerente de los nombre de los menús, puede recurrirse a ellos para recordar la sintaxis de gran parte de los comandos del curso.

Igualmente podemos integrar la función f en un intervalo compacto mediante la orden integrate

```
integrate(f(x),x,0,2);
```

Obsérvese el efecto sobre integrate del siguiente cambio de argumentos:

```
integrate(f(x),x);
```

También podemos dibujar su gráfica gracias a la sentencia wxplot2d:

```
wxplot2d([f(x)],[x,0,2]);
```

Esta gráfica se ha generado en línea. Es posible generar directamente un archivo que contenga solo esta gráfica en una ventana distinta:

```
plot2d([f(x)],[x,0,2],[plot_format,gnuplot]);
```

Maxima permite igualmente definir funciones a trozos a través de un comando condicional: if then else. Así, para la función

```
       | x^2+2x, si x<0
g(x)= |
       | sen(x+1), en otro caso
```

hacemos

```
g(x):=if x<0 then x^2+2·x else sin(x+1);
```

Aunque la derivación y la integración (donde proceda) hay que realizarlas a trozos, se puede evaluar en un punto

```
g(−1);
```

y dibujar su gráfica (repara en el segmento vertical a la altura de x=0)

```
wxplot2d([g(x)], [x,−%pi,%pi])$
```

Para una función con más de trozos, basta con anidar convenientemente más condicionales. Por ejemplo, para

```
       | 2x, si x<1
h(x)= | x^3-5, si 1≤x≤2,
       | log(x), si x>2
```

introducimos

```
h(x):=if x<1 then 2·x else
                if x<=2 then x^3−5 else log(x) ;
```

Todo lo anterior es extendible a funciones reales de varias variables, salvo la representación de la gráfica de una tal función, que obviamente solo tiene sentido para dos variables.

Al igual que mencionamos con la asignación de valores a variables, el comando kill elimina la definición de las funciones definidas con anterioridad.

```
;
```

6 Vectores

En los siguientes ejemplos se ilustra cómo introducir un vector, hallar su longitud y obtener un elemento mediante su posición

```
a:[3,7,−5/9];
```

```
length(a);
```

```
a[2];
```

En ocasiones manejaremos vectores cuyas coordenadas dependen de la posición que ocupan. En tal caso, podemos construir fácilmente el vector mediante la sentencia makelist:

```
b:makelist(i^2/3,i,1,10);
```

El producto escalar usual (euclídeo) de dos vectores de la misma longitud se calcula con un simple punto:

```
[1,2,1].[3,2,1];
```

Además, podemos calcular la suma de los elementos de un vector, su producto, su máximo... gracias al comando apply:

```
apply("+",b);
```

```
apply("·",b);
```

```
apply(max,b);
```

7 Matrices

Las matrices se introducen fácilmente y admiten manipulaciones análogas a las realizadas con vectores

```
c:matrix([1,2,3],[4,5,6]);
```

Comprueba que también puede introducirse una matriz a partir del menú Álgebra. Manipula la ventana que se despliega, cambiando las dimensiones y el tipo de matriz. Repara en que, para introducir matrices diagonales de orden elevado, resulta rentable. Sin embargo, la matriz identidad se puede obtener directamente:

```
ident(6);
```

Observa además:

```
matrix_size(c);
```

```
c[1,2];
```

```
c[1];
```

Al igual que con makelist, las matrices con coeficientes expresables en función de su posición se generan a partir de una sentencia, en este caso genmatrix, que además vuelve a aparecer en el menú Álgebra

```
d: genmatrix(lambda([i,j], i−j/4), 3, 4);
```

7.1 Operaciones con matrices

Las operaciones usuales con matrices se efectúan con Maxima de forma inmediata

```
m: matrix([2,−1],[2.345,−8]);
```

```
n: matrix([1/2,1/3],[1/4,1/5]);
```

```
o: matrix([1,−11.2,1.1],[1,4,0]);
```

```
m+n;
```

```
3·n;
```

```
n.o;
```

```
transpose(o);
```

```
transpose((4·m−6·n).o);
```

```
m^^2;
```

Podemos también multiplicar una matriz por un vector:

```
c.a;
```

Es posible asimismo calcular rangos, determinantes e inversas de matrices regulares

```
rank(n);
```

```
determinant(n);
```

```
invert(n);
```

```
n^^(−1);
```

Señalemos que el comando kill también permite limpiar valores de variables a los que se han asignado matrices o vectores.

7.2 Valores y vectores propios

La orden eigenvalues calcula los valores propios de una matriz cuadrada:

```
e:matrix(
 [1/2,2/3,3/4],
 [0,1,2],
 [0,1,1]
 );
```

```
eigenvalues(e);
```

La matriz e posee tres valores propios, y cada uno de ellos tiene multiplicidad algebraica 1. Para hallar una base de vectores propios recurrimos a la sentencia eigenvectors

```
eigenvectors(e);
```

La salida generada proporciona nuevamente los valores propios, con sus respectivas multiplicidades algebraicas, junto con una base de vectores propios.

8 Rudimentos de programación

Nos ocupamos finalmente de analizar algunas de las herramientas de programación que incluye Maxima y que nos serán de uitildad a lo largo de la asignatura. En la definición de una función a trozos ya hemos introducido una de ellas, if. Se trata de una sentencia condicional. Veamos otro ejemplo de su funcionamiento

```
f: matrix(
 [1,%e/4],
 [−2.3,55.7896543]
 );
```

```
if determinant(f)=0 then print("singular") else print("regular");
```

Otra estructura esencial en programación es el bucle. En Maxima podemos hacer uso del comando for, cuya sintaxis responde a la estructura comienzo-incremento-fin-cuerpo. La parte comienzo-incremento-fin determina el rango de variación de la variable de control: su comienzo o inicialización, su incremento o paso, y su finalización . La última, en cambio, determina la acción a ejecutar en función de la variable de control. Veamos un ejemplo:

```
for i:1 step 1 thru 6 do print("el cubo de",i,"es ",i^3);
```

Cuando el paso es 1, que es el caso más usual (de hecho, todos pueden reducirse a él, modificando convenientemente el cuerpo del bucle), es omitible. Además, el cuerpo puede contener más de una acción:

```
for i:1 thru 3 do (print((i^3)/4.0),print(i));
```

En el caso de paso 1, además podemos expresar la finalización con una condición (observa la orden display, que se usa con variables):

```
for i:2 while i<16 do display(i^2);
```

Veamos otro ejemplo. Calculemos la suma de los primeros 100 números naturales:

```
s:0;
```

```
for i:1 thru 100 do s:s+i;
```

```
s;
```

Comprobemos el resultado con apply:

```
y:makelist(i,i,1,100);
```

```
apply("+",y);
```

```
s;
```

Observemos que Maxima permite calcular sumas y productos sin necesidad de construir previamente un vector:

```
sum(i,i,1,100)
```

```
product(i^2,i,1,8)
```

Un último ejemplo. Podemos hacer un pequeño programa que nos devuelva la norma del máximo de una matriz cuadrada cualquiera. El dato será a (la matriz).

```
a: 1/10.0·genmatrix(lambda([i,j], (−1)^(i+j)·i/j), 5, 5);
```

```
n:matrix_size(a)[1];
```

```
v:makelist(apply("+",abs(a[i])),i,1,n);
```

```
apply(max,v);
```

Se puede hacer incluso de forma compacta, mediante un módulo:

```
normmax(b):=block(apply(max,makelist(apply("+",abs(b[i])),i,1,
        matrix_size(b)[1])));
```

Testémoslo con los datos anteriores:

```
normmax(a);
```

9 EJERCICIOS

1.- Define la función real g en el intervalo [0,1] como

$$g(x)=\begin{cases} 2x-\log(x), & \text{si } 0.3\le x\le 0.5 \\ 2/x+|x-0.6|, & \text{si } 0.5\le x\le 0.8 \\ 0, & \text{en otro caso.} \end{cases}$$

Calcula su integral y dibuja su gráfica.

2.- Halla el radio espectral de la matriz de orden 4x4 cuyo coeficiente (i,j) es |2i-4j|.

3.- Calcula la suma de los cubos de los 23 primeros números naturales mediante un adecuado bucle. Comprueba que el resultado obtenido es correcto mediante el comando apply.

4.- Halla el producto de los inversos de los números naturales comprendidos entre 6 y 19 haciendo uso de un bucle, y compruba la validez de tu respuesta con la orden apply.

5.- Determina el término 43º de la sucesión de Fibonacci de forma recurrente y a partir de la expresión explícita de dicho término.

6.- Calcula, aplicando el comando float, el valor de $\sqrt{(5+10^{-n})}-\sqrt{5}$, para n=1,...,20. Repite los cálculos, reescribiendo la expresión $\sqrt{(5+10^{-n})}-\sqrt{5}$ como $10^{(-n)}/(\sqrt{(5+10^{-n})}+\sqrt{5})$. Interpreta los resultados.

7.- Programa (no necesariamente módulo) el cálculo de la norma 1 de una matriz cuadrada cualquiera.

8.- Diseña un programa que, a partir de una matriz cuadrada, genere como salida su condicionamiento (norma infinito) si es regular, o el mensaje "la matriz no es regular" en caso contrario.

9.- Calcula la norma euclídea de la matriz 2x4 cuyo coeficiente (i,j) sea i/(i+j+1).