Prácticas con wxMaxima: preliminares

BENITO J. GONZÁLEZ RODRÍGUEZ (bjglez@ull.es)
DOMINGO HERNÁNDEZ ABREU (dhabreu@ull.es)
MATEO M. JIMÉNEZ PAIZ (mjimenez@ull.es)
M. ISABEL MARRERO RODRÍGUEZ (imarrero@ull.es)
ALEJANDRO SANABRIA GARCÍA (asgarcia@ull.es)

Departamento de Análisis Matemático
Universidad de La Laguna

Índice

PRÁCTICA 1: PRELIMINARES.

Duración estimada: 3 horas.

Objetivo: Iniciar y familiarizar al alumnado en algunas instrucciones básicas de wxMaxima.

1 Una breve introducción

Los comandos en wxMaxima se ejecutan tecleando SHIFT+ENTER.

```
--> (-3*4+5/2)/7;
```

```
--> 3/2-1/3;
```

Podemos ejecutar varias sentencias en grupo sin más que separarlas con un ";". Observar que cada instrucción de entrada tiene su correspondiente identificador de salida.

```
--> (-3*4+5/2)/7; 3/2-1/3; 5^2;
```

Debemos tener cuidado con los espacios, en especial en lo respecta multiplicaciones. En concreto, un espacio no se interpreta como una multiplicación.

```
--> -2 + 5;
```

```
--> 2 3;
```

Con el símbolo "$" al final de una sentencia logramos que no se muestre el resultado de la misma, aunque su resultado quede almacenado en la correspondiente variable de salida.

```
--> 5^2$
```

```
--> %;
```

Claramente debemos cuidar el uso de paréntesis en las operaciones.

```
--> 5^-1-2;
```

```
--> 5^(-1-2);
```

Por defecto wxMaxima considera un formato "xml" y las presentaciones en pantalla suelen abreviarse en caso de ser excesivamente largas.

```
--> 3^1000;
```

Podemos alternativamente pasar a formato "ascii" con la orden "set_display(ascii)".

```
--> set_display(ascii)$
    3^1000;
```

```
--> set_display(xml)$
```

Para introducir una raíz cuadrada usaremos la orden "sqrt", o bien una potencia fraccionaria.

```
--> sqrt(7); 7^(1/2);
```

```
--> sqrt(8); 8^(1/2);
```

Podemos también considerar raíces de cualquier índice, expresadas mediante potencias fraccionarias.

```
--> 5^(1/3);

--> 3^1/3*9^1/3;
```

Para asignar un valor a una variable o expresión usaremos ":".

```
--> a:1$ b:3$ c:a+b;

--> a; b; c;
```

Para reinicializar una (o varias variables) usaremos la orden "kill(a,b,...)".

Con la orden "kill(all)" reinicializamos todas las variables, incluidas las variables de salida %o.

```
--> kill(a,b)$ a; b; c;

--> kill(all);

--> a+b; c;
```

Con la instrucción "/* COMENTARIO */" se puede introducir un comentario que es ignorado a la hora de calcular.

```
--> v0:2 /*Velocidad inicial*/;
    e0:14 /*Espacio inicial*/;
    a:-0.5 /*Aceleración*/ ;
    t:3 /*Tiempo*/;
    e:e0+v0*t+1/2*a*t^2;

--> kill(all)$ v0; e0; a; t; e;
```

wxMaxima también puede trabajar en modo simbólico.

```
--> p:(x+y)^4;

--> q:expand(p);

--> p-q;

--> factor(q);

--> %-p;
```

La orden factor no sólo permite factorizar polinomios, sino también números enteros.

```
--> a:1235431450$ factor(a);

--> a-2*(5^2)*11*2246239;

--> b:23982323432432499843213123211$ factor(b);

--> b-3*733*15117701*72140687161773179;
```

2 Operaciones con números en coma flotante.

```
--> kill(all);
```

```
--> 1.2345678902+0.3333333333*3;
```

Maxima devuelve resultados exactos si los datos de entrada son exactos. En otro caso devuelve cantidades aproximadas. Observar además que las fracciones se simplican automáticamente.

```
--> a:36/128;
```

```
--> 36.0/128;
```

Con la orden "float" podemos obtener la expresión decimal con 16 dígitos correctos.

```
--> float(a);
```

```
--> float(sqrt(8));
```

Observar la diferencia en las siguientes operaciones al trabajar tanto con cantidades exactas en modo simbólico como con números en punto flotante.

```
--> b:sqrt(2)+sqrt(3);
```

```
--> bf:float(b);
```

```
--> c:1.0*sqrt(2)+sqrt(3);
```

```
--> cf:float(c);
```

```
--> c-b;
```

```
--> cf-bf;
```

```
--> c-bf;
```

```
--> float(c-bf);
```

```
--> kill(all);
```

Podemos cambiar la precisión en los cálculos con la sentencia "fpprec".

Para visualizar el nuevo desarrollo usaremos la orden "bfloat".

```
--> fpprec:10$
```

```
--> a:float(sqrt(8));
```

```
--> b:bfloat(sqrt(8));
```

```
--> fpprec:50$ float(sqrt(8)); c:bfloat(sqrt(8));
```

```
--> fpprec:16;
```

Observemos que tras modificar el número de dígitos, las variables retienen el número de cifras respectivas computadas.

```
--> b-a; c-a; c-b;
```

Si deseamos una salida con potencias de 10, usaremos la sentencia "numer".

```
--> b-a, numer; c-a, numer; c-b, numer;
```

"numer" es realmente una variable lógica, cuyo valor por defecto es "false". Por ello, se trabaja de modo exacto con cantidades exactas.

```
--> numer;
```

```
--> 1/3+1/7;
```

Podemos, no obstante, cambiar el valor de "numer" en cualquier momento.

```
--> numer:true$ 1/3+1/7; numer:false$ 1/3+1/7;
```

Los números pi y e se obtienen como %pi y %e, respectivamente. Sin el símbolo "%" sólo se obtienen variables simbólicas.

```
--> pi; float(pi); %pi; float(%pi);
```

```
--> float(pi^100); float(%pi^100);
```

```
--> e; float(e); %e; float(%e);
```

```
--> float(sqrt(e+1)); float(sqrt(%e+1)); float(sqrt(exp(1)+1));
```

Algunos ejemplos adicionales.

```
--> float(2^-100);
```

```
--> kill(all);
```

```
--> a:sqrt(2)$ b:%pi/3$ c:a+b;
```

Podemos referirnos un cálculo realizado anteriormente con la sentencia "%o" indicando el número de salida correspondiente.

```
--> float(%o3^2);
```

```
--> bfloat(%o2^2000);
```

```
--> %,numer;
```

```
--> cos(1.23);
```

```
--> sin(3.24);
```

```
--> tan(pi/3); tan(%pi/3);
```

```
--> fpprec: 10$ bfloat(a^2+b^20+c^30);
```

```
--> %,numer;
```

```
--> fpprec: 20$ bfloat(a^2+b^20+c^30);
```

```
--> %,numer;
```

3 Introducción a la representación gráfica.

```
--> kill(all);
```

El comando plot2d permite la representación de funciones y curvas en el plano. Observamos que por defecto la representación se hace en formato "gnuplot".

```
--> plot2d([x^2],[x,-2,2]);

--> plot2d([x^2],[x,-2,2],[y,-1,3]);
```

De modo sencillo podemos representar varias curvas en una misma gráfica, crear etiquetas para los ejes, leyendas, título, rejilla, etc. Además, se dispone de una amplia gama de colores (blue, red, green, magenta, black, cyan, coral, salmon, etc.).

```
--> plot2d([x^2,2*x+1],[x,-3,3]);

--> plot2d([x^2,2*x+1],[x,-3,3],[xlabel,"Eje OX"],[ylabel,"Eje OY"],[gnuplot_preamble, "set title 'Gráfica de prueba'"],
    [color,green,magenta],[legend,"f(x)=x^2","g(x)=2x+1"]);

--> plot2d([x^2,2*x+1],[x,-3,3],[xlabel,"Eje OX"],[ylabel,"Eje OY"],[gnuplot_preamble, "set grid;set title 'Gráfica de prueba'"],
    [color,green,magenta],[legend,"f(x)=x^2","g(x)=2x+1"]);

--> plot2d([sin(x),cos(x)],[x,-2*%pi,2*%pi],[xlabel,"x"],[ylabel,"y"],[legend,"Función seno: f(x)=sen(x)","Función coseno: g(x)=cos(x)"],
    [gnuplot_preamble, "set grid; set title 'Funciones trigonométricas elementales'"]);

--> plot2d([sin,cos],[x,-2*%pi,2*%pi],[xlabel,"x"],[ylabel,"y"],[legend,"Función seno: f(x)=sen(x)","Función coseno: g(x)=cos(x)"],
    [gnuplot_preamble, "set grid; set title 'Funciones trigonométricas elementales'"]);
```

Con la orden "[nticks,n]" podemos mejorar la representación gráfica.

```
--> plot2d(sin(1/x),[x,-0.5,0.5],[y,-1.5,1.5]);

--> plot2d(sin(1/x),[x,-0.5,0.5],[y,-1.5,1.5],[nticks,500]);
```

Escribir el prefijo "wx" antes de la instrucción para representar la gráfica logramos una representación en línea.

```
--> wxplot2d(sin(x)*sin(20*x),[x,0,4*%pi]);

--> wxplot2d(sin(x)*sin(200*x),[x,0,2*%pi]);

--> wxplot2d(sin(x)*sin(200*x),[x,0,2*%pi],[nticks,500]);
```

Otras opciones alternativas para representar gráficas con plot2d pasan por considerar una representación "en línea" con la orden wxplot2d, o bien un formato "openmath".

```
--> wxplot2d(2*x-x^2,[x,-1,3]);

--> plot2d([2*x-x^2], [x,-1,3],
     [plot_format, openmath],
     [nticks,20]);

--> plot2d([2*x-x^2], [x,-1,3],
     [plot_format, gnuplot],
     [nticks,20]);
```

¿Puedes indicar por qué se produce error en la siguiente ejecución?

```
--> x:5$ wxplot2d([2*x-x^2], [x,-1,3]);

--> kill(x);
```

Puedes consultar en línea la ayuda de wxMaxima sobre un determinado comando tecleando "??" antes del comando respectivo.

```
--> ??plot2d;
```

Con la orden plot2d también se pueden representar gráficas de puntos discretos, o incluso curvas definidas a través de un parámetro.

```
--> puntos:[[-1,1],[3,2],[3,-2],[0,-3],[4,-2]] /* Aquí puntos es una lista de puntos en el plano */;

--> wxplot2d([discrete, puntos]);
```

Por defecto wxMaxima une los puntos mediante una poligonal. Podemos no obstante representar los puntos de modo discreto usando símbolos comunes (point, box, square, circle, diamond, etc.).

```
--> wxplot2d([discrete, puntos],[style,points], [point_type,diamond],[color, red]);
```

Veamos cómo tratar curvas definidas de modo parámetrico.

```
--> plot2d([parametric, t, abs(cosh(t/5)-2)+sin(t^2)/10,[t,-10,10],[nticks,1000]]);

--> r:(exp(cos(t))-2*cos(4*t)-sin(t/12)^5)$

    plot2d([parametric, r*sin(t), r*cos(t), [t, -8*%pi, 8*%pi], [nticks, 2000]]);
```

De modo similar, podemos representar curvas definidas implicitamente con la orden "implicit_plot". Para ello cargaremos previamente el paquete correspondiente.

```
--> load(implicit_plot);

--> implicit_plot(x^2+y^2-1,[x,-1.5,1.5],[y,-1.5,1.5]);

--> wximplicit_plot(x^2+y^2=1,[x,-1.5,1.5],[y,-1.5,1.5],[nticks,1000],[gnuplot_preamble, "set size ratio 1"]);

--> wximplicit_plot((x+2)^2+y^2=1,[x,-2,0],[y,-1.5,1.5],[nticks,1000],[gnuplot_preamble, "set size ratio 1.5; set zeroaxis"]);

--> wximplicit_plot(x^3-2*x*y + y^3,[x,-3,3],[y,-3,3],[nticks,1000],[gnuplot_preamble, "set size ratio 1"]);

--> wximplicit_plot(x^2-y^2=4,[x,-10,10],[y,-10,10],[color,green],[nticks,1000],[gnuplot_preamble, "set size ratio 1"]);

--> wximplicit_plot(abs(x)^(1/3)+abs(y)^(1/3)-2,[x,-10,10],[y,-10,10],[color,red],[nticks,1000],[gnuplot_preamble, "set size ratio 1"]);
```

Para gráficas en 3D podemos usar, de modo análogo, la orden plot3d.

```
--> wxplot3d(x^2+y^2,[x,-5,5],[y,-5,5]);

--> wxplot3d(x^2+y^2, [x,-5,5], [y,-5,5],
     [gnuplot_preamble, "set hidden3d"])$

--> plot3d(cos(x*y), [x,-5,5], [y,-5,5],[grid, 75,75]);

--> wxplot3d(cos(x*y), [x,-5,5], [y,-5,5], [plot_format,gnuplot],[gnuplot_preamble, "set pm3d at b"])$

--> plot3d(exp(-x^2-y^2), [x,-2,2], [y,-2,2],[grid, 50,50]);

--> wxplot3d(exp(-x^2-y^2), [x,-5,5], [y,-5,5],[plot_format,gnuplot],[gnuplot_preamble, "set pm3d at b"])$
```

En una próxima práctica veremos otros comandos adecuados para efectuar representaciones gráficas 2D y 3D algo más elaboradas.

4 Funciones y expresiones.

--> kill(all);

Una función en wxMaxima es un "mecanismo de sustitución" y se define con la sintaxis "nombre(variables):=expresión".

--> log10(x):=log(x)/log(10);

--> log10(10.0);

--> %,numer;

--> log10(1.23),numer;

--> wxplot2d(log10,[x,1,10]);

Observación: Maxima sólo trae definida por defecto la función logaritmo natural (o neperiano).

--> f(x):=x^2-4*x+3;

--> f*2; f(x)*2;

--> f(s+1); expand(%);

Observa los siguientes modos de representar una función de una variable real.

--> wxplot2d(f(x),[x,0,4]);

--> wxplot2d(f,[s,0,4]);

--> wxplot2d(f,[x,0,4]);

Es claro que la siguiente instrucción es errónea.

--> wxplot2d(f(x),[s,0,4]);

Veamos el siguiente ejemplo en el que dibujamos funciones trasladadas.

--> g(x):=f(x-2)$ h(x):=f(x+5)$ wxplot2d([f,g,h],[x,-6,6],[y,-1.5,4],[color,red,blue,green]);

Si cambiamos la definición de f(x), las funciones g(x) y h(x) se verán afectadas.

--> f(x):=x/(x^2+1)$ wxplot2d([f,g,h],[x,-8,8],[y,-1,1],[color,red,blue,green]);

--> kill(f);

--> f(x); g(x); h(x);

--> kill(g,h);

También podemos trabajar con expresiones, en lugar de funciones.

--> expr:x^2-4*x+3;

--> wxplot2d(expr,[x,0,4]);

Sin embargo, al evaluar una expresión debemos tener cuidado y usar las órdenes "subst" o "ev", tal como se indica a continuación.

```
--> expr(1);

--> subst(x=1,expr);

--> subst(1,x,expr); ev(expr,x=1);

--> expand(subst(s+1,x,expr)); expand(ev(expr,x=s+1));

--> u:x^2+sin(2*x);

--> subst(x=%pi/4,u); ev(u,x=%pi/4);
```

Podemos además definir una función a partir de una expresión, sin más que evaluar la expresión en un argumento genérico.

```
--> f(x):=ev(expr,x=x); f(1); f(1.5);

--> wxplot2d(f(x+1),[x,-2,3]);
```

5 Resolución de ecuaciones y sistemas de ecuaciones.

```
--> kill(all);
```

Podemos resolver ecuaciones o sistemas de ecuaciones algebraicas (!) con la orden "solve".

```
--> solve(x^2=25,x); solve(x^2-25,x);

--> solve(a*x+b=c,x);

--> solve(a*x+b=c,a) /* Se resuelve la ecuación respecto de "a", siendo "b", "c" y "x" parámetros. */;

--> solve(x^2=a+1,x); solve(x^2=a+1,a);

--> solve([y=x+1,x+y=5]);

--> solve([y=x+a,x+y=5],[x,y]) /* En este caso interpretamos "a" como parámetro */;

--> solve([y=x+a,x+y=5],[y,a]) /* En este caso interpretamos "x" como parámetro */;

--> solve(a*x^2+b*x+c=0,x);
```

Resolución por radicales de la ecuación cúbica.

```
--> solve(a*x^3+b*x^2+c*x+d=0,x);
```

Resolución por radicales de la ecuación de cuarto grado.

```
--> solve(a*x^4+b*x^3+c*x^2+d*x+e=0,x);

--> set_display(ascii)$ solve(a*x^4+b*x^3+c*x^2+d*x+e=0,x);

--> set_display(xml)$
```

Imposibilidad de resolución de la quíntica por radicales.

```
--> solve(a*x^5+b*x^4+c*x^3+d*x^2+e*x+f=0,x);
```

Dado que las soluciones vienen dadas en forma de lista, podemos realizar una asignación con los valores obtenidos. Podemos proceder de diversas maneras, con las órdenes "rhs" y "map".

```
--> eq:6*x^2+x-2;

--> sols:solve(eq=0,x);

--> sols[1]; sols[2];

--> rhs(part(sols,1)); rhs(part(sols,2));

--> solu:map(rhs,sols);

--> solu[1]; solu[2];

--> subst(x=solu[1],eq); subst(x=solu[2],eq);
```

También podemos simplificar ecuaciones con la orden "factor". Este comando realiza por defecto factorizaciones con coeficientes racionales.

```
--> factor(eq);

--> 6*x^2+x-2=0; factor(%); solve(%);

--> factor(2*x^2-x-4); solve(2*x^2-x-4);

--> factor(2*x^7-x^4-4*x^2); solve(2*x^7-x^4-4*x^2);
```

Para el cálculo aproximado de raíces (tanto reales como complejas) de polinomios podemos usar la orden "allroots".

```
--> allroots(2*x^2-x-4);

--> allroots(2*x^5-x^2-4),numer;

--> allroots(2*x^7-x^4-4*x^2),numer;

--> subst(x=1.224178132898109,2*x^7-x^4-4*x^2);

--> expand(subst(x=0.74756650341073*%i-0.90601917769566,2*x^7-x^4-4*x^2));
```

La orden "solve" sólo resuelve ecuaciones algebraicas. Para resolver numéricamente ecuaciones trascendentes podemos usar la orden "find_root" indicando un intervalo que contenga a la raíz buscada.

```
--> ecu:2-x^2=exp(x);

--> solve(ecu); float(%);

--> find_root(ecu,x,0,2);

--> subst(x=0.53727444917386,2-x^2-exp(x));

--> wxplot2d([2-x^2,exp(x)],[x,0,1]);
```

En general, la resolución de ecuaciones no lineales se hará a través de algoritmos numéricos con la orden "find_root". Podemos también usar el célebre método de Newton cargando previamente el paquete "mnewton" correspondiente y usando la instrucción "mnewton(expr,x,x_0)".

```
--> load(mnewton);

--> mnewton(2-x^2-exp(x),x,1);

--> find_root(2-x^2-exp(x),x,0,1);

--> f:2-cos(x^2)$ g:x*exp(x^2)$ wxplot2d([f,g],[x,0,1]);

--> s:solve(f=g,x) /* El comando "solve" no es capaz de resolver numéricamente la ecuación */;

--> float(s);

--> find_root(f=g,x,0,1);

--> mnewton(f=g,x,1); mnewton(f=g,x,0);
```

La sentencia "mnewton" se puede aplicar también a sistemas de ecuaciones no lineales.

```
--> mnewton([x^2+y-3,x+y^3+1],[x,y],[0,0]);

--> subst([x=-1.487589175670373,y=0.78707844442834],[x^2+y-3,x+y^3+1]);

--> mnewton([2*3^x-x/y-5, x+2^y-4], [x,y], [1,1]);

--> mnewton([x^2+exp(y)+z^2-2,exp(x)+y^2+z^2-2,x^2+y^2*z-2],[x,y,z],[1,1,1]);

--> mnewton([x^2+exp(y)+tan(z)-2,exp(x^2)+y^2+z^2-2,x^2+y^2*z-2],[x,y,z],[1,1,1]);
```

Otros comandos útiles a efectos de simplificación de expresiones son "ratsimp" (para expresiones racionales), "radcan" (irracionales) y "trigsimp" (trigonométricas).

```
--> ratsimp(2*x*(x^2-3)+x^2-x^3);

--> ratsimp(1/x+4/(x-3));

--> radcan(sqrt(6)/(sqrt(6)-sqrt(3)));

--> radcan((exp(2*x)-1)/(exp(x)+1));

--> trigsimp(3*sin(x)^2+cos(x)^2+exp(x)*exp(2*x));

--> trigsimp(3*tan(x)^2+cot(x)^2);
```

Finalmente, comentamos cómo resolver algunas ecuaciones trigonométricas elementales con una variante de la orden "solve". En los siguientes casos, en las situaciones más favorables, "solve" sólo devuelve una solución entre las infinitas posibles.

```
--> solve(sin(x)=0,x);

--> solve(cos(x)=0,x);

--> solve(sin(1/x)=0,x);

--> solve(sin(x)-cos(x)=0,x);
```

En algunos casos, podemos solventar este problema cargando el paquete "to_poly_solve" y usando la orden "%solve".

```
--> load(to_poly_solve)$
```

```
--> solve(sin(x)=0,x);
```

```
--> %solve(sin(x)=0,x); nicedummies(%);
```

```
--> nicedummies(%solve(cos(x),x));
```

```
--> nicedummies(ratsimp(%solve(sin(1/x)=0,x)));
```

```
--> nicedummies(ratsimp(%solve(sin(x)-cos(x)=0,x)));
```

```
--> nicedummies(ratsimp(%solve(sin(x)=1/2,x)));
```

```
--> nicedummies(ratsimp(%solve(sin(x)=2,x)));
```

Created with wxMaxima.