# Sesión 1: Introducción a Maxima. Cálculo simbólico y cálculo aproximado. Errores y estabilidad numérica.

Versión: 12 de marzo de 2011.

## 1 Introducción a Maxima

Más información sobre Maxima en

http://laguerre.ugr.es/maxima/

### 1.1 Cálculo simbólico y numérico

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

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

```
--> 3.0 + 1/2;
```

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

```
--> 3 + 1/2, numer;
```

```
--> %pi + %e, numer;
```

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

```
--> bfloat(%pi + %e);
```

```
--> fpprec : 50$
    bfloat(%pi^5 + %e^2);
```

```
--> fpprec : 25$
    bfloat(%pi^5 + %e^2);
```

```
--> fpprec : 50$
    bfloat(%pi^2 + %e^2);
```

En la subsección 2.1 se puede observar diferencias entre float y numer.

## 1.2 Asignación de variables

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

```
--> x : 23;
    y : 48;
    x*y;
```

```
--> x/y;
```

```
--> x/y, numer;
```

```
--> x-y;
```

## 1.3 Resolución de ecuaciones

Si queremos resolver la ecuación x^2+3x-2=0:

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

Dos formas de solucionar el problema:

1) conservando el valor previo de la incógnita,

```
--> x;
```

```
--> solve('x^2+3*'x-2=0, 'x);
```

```
--> x;
```

2) o perdiendo el valor previo de la incógnita.

```
--> kill(x) $
    solve(x^2+3*x-2=0, x);
```

```
--> x;
```

## 1.4 Ecuaciones "complicadas"

En algunos casos las ecuaciones no son fáciles de resolver. Veamos un ejemplo y la manera de obtener una "aproximación" de la solución.

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

```
--> float(solve(exp(-x)-x=0,x));
```

```
--> to_poly_solve(exp(-x)=x,x);
```

```
--> float(to_poly_solve(exp(-x)=x,x));
```

En principio no sabemos que método ha utilizado Maxima para resolver la ecuación. En la siguiente sesión haremos nuestra propias implementaciones de métodos para estos casos.

## 1.5 Algo más sobre cálculo simbólico

Maxima también trabaja con expresiones simbólicas.

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

```
--> expr : a+b/a+b^2+sqrt(b/a);
```

```
--> ratsimp(expr);
```

```
--> radcan(expr);
```

```
--> expand(expr);
```

Podemos evaluar las expresiones simbólicas de diversas maneras (aparentemente al menos).

```
--> expr,a=1,b=2;
```

```
--> a;
```

```
--> expr,a:1,b=2;
```

```
--> a;
```

```
--> a:1;
```

```
--> a;
```

## 1.6 Definición de funciones

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

```
--> f(x) := x^2 + sin(x) - exp(x);
```

```
--> f(3);
```

```
--> f(3.9);
```

## 1.7 Gráficas de funciones

Las siguientes formas de representar funciones dependen del sistema operativo que usemos (Windows, Linux, Mac) y de la versión de Maxima.

```
--> plot2d([f(x)], [x,-5,5],
     [plot_format, openmath])$
```

```
--> wxplot2d([f(x)], [x,-5,5])$
```

```
--> plot2d([f(x)], [x,-5,5],
     [plot_format, gnuplot])$
```

Si añades wx antes de wxplot2d entonces la gráfica sale siempre en línea.

```
--> wxplot2d([f(x)], [x,-5,5],
     [plot_format, openmath])$
```

```
--> wxplot2d([f(x)], [x,-5,5],
     [plot_format, gnuplot])$
```

## 1.8 Matrices

```
--> kill(all) $
```

Empezamos viendo como se define una matriz.

```
--> A : matrix([1, 2, 3], [4, 5, 6], [7, 8, 9]);
```

```
--> A[1,2];
```

```
--> B : genmatrix(b,3,3);
```

```
--> b[i,j]:=i-j;
```

```
--> B;
```

```
--> B : genmatrix(b,3,3);
```

¡Cuidado con el producto y la potencia de matrices!

```
--> B.A;
```

```
--> B*A;
```

```
--> A^2;
```

```
--> A^^2;
```

Algo más sobre matrices.

```
--> A;
```

```
--> submatrix(1,A,2);
```

```
--> row(A,2);
```

```
--> col(A,3);
```

```
--> addcol(A,[1,2,3]);
```

# 2 Error por cancelación de dígitos en una operación simple

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

```
--> a : 1.0 $
    b : 10.0^20 $
    c : -10.0^20 $
    [(a+b)+c, a+(b+c)];
```

```
--> a : 123456789012.13 $
    b : 123456789012.12 $
    [a^2-b^2, (a+b)*(a-b)];
```

Veamos como afecta la precisión a los cálculos.

```
--> fpprec;
```

Aumentemos la precisión en el primer ejemplo.

```
--> fpprec : 20$
    a : 1.0 $
    b : 10.0^20 $
    c : -10.0^20 $
    [(a+b)+c, a+(b+c)];
```

Parece que no basta con aumentar la precisión.

```
--> fpprec : 20$
    a : bfloat(1.0) $
    b : bfloat(10.0^20) $
    c : bfloat(-10.0^20) $
    [(a+b)+c, a+(b+c)];
```

Operemos en el segundo ejemplo.

```
--> fpprec : 30$
    a : bfloat(123456789012.13) $
    b : bfloat(123456789012.12) $
    [a^2-b^2, (a+b)*(a-b)];
```

Otro ejemplo más.

```
--> fpprec : 16$
```

```
--> for i : 1 thru 18 step 1 do (
        x : 33.12*10^i,
        r1 : sqrt(x)/(sqrt(x+1)+sqrt(x)),
        r2 : sqrt(x)*(sqrt(x+1)-sqrt(x)),
        print(i, ", ", r1, ", ", r2, ", ", r1-r2)
    )$
```

Aprovechemos este ejemplo para mejorar la presentación de la salida.

```
--> for i : 1 thru 18 step 1 do (
        x : 33.12*10^i,
        r1 : sqrt(x)/(sqrt(x+1)+sqrt(x)),
        r2 : sqrt(x)*(sqrt(x+1)-sqrt(x)),
        printf(true, "~2d ~t ~19,16f ~t ~19,16f ~t ~19,16f ~%", i, r1, r2, r1-r2)
    )$
```

Para entender mejor como funciona printf vamos a variar los parámetros.

```
--> for i : 1 thru 18 step 1 do (
        x : 33.12*10^i,
        r1 : sqrt(x)/(sqrt(x+1)+sqrt(x)),
        r2 : sqrt(x)*(sqrt(x+1)-sqrt(x)),
        printf(true, "~5d ~t ~25,10f ~t ~25,12f ~t ~25,14f ~%", i, r1, r2, r1-r2)
    )$
```

Atención: la orden kill(all) no afecta a fpprec.

```
--> fpprec:18;
```

```
--> fpprec;
```

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

```
--> fpprec;
```

## 2.1 Más sobre float y numer

Las siguientes instrucciones te serviran para comprender la diferencia entre float y numer.

```
--> fpprec : 16$
    a : 12345678901213/100 $
    b : 12345678901212/100 $
    [a^2-b^2, (a+b)*(a-b)];
```

```
--> [float(a^2-b^2), float((a+b)*(a-b))];
```

```
--> [a^2-b^2, (a+b)*(a-b)],numer;
```

# *3 Error propagado en un proceso iterativo inestable*

Computación simbólica:

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

```
(%i) x : 1 $
     y : 1/3 $
     for i : 1 thru 20 step 1 do (
         z : 10*y/3-x,
         t : 1/3^(i+1),
         printf(true, "~2d ~t ~15,a ~t ~15,a ~t ~2,a ~%", i, z, t, z-t),
         x : y,
         y : z
     ) $
```

En este caso es fácil "entender" los números que aparecen en la tabla. Una opción para ver la representación en punto flotante de los resultados finales es la siguiente.
(¡Ojo! Los cálculos intermedios han sido simbólicos).

```
(%i) x : 1 $
     y : 1/3 $
     for i : 1 thru 20 step 1 do (
         z : 10*y/3-x,
         t : 1/3^(i+1),
         printf(true, "~2d ~t ~19,16f ~t ~19,16f ~t ~19,16f ~%", i, z, t, z-t),
         x : y,
         y : z
     ) $
```

Computación numérica:

```
(%i) kill(all);
```

```
--> x : 1.0 $
    y : 1.0/3.0 $
    for i : 1 thru 40 step 1 do (
        z : 10*y/3-x,
        t : 1.0/3^(i+1),
        printf(true, "~2d ~t ~22,16f ~t ~19,16f ~t ~22,16f ~%",i , z, t, z-t),
        x : y,
        y : z
    ) $
```

# 4 Criterio de parada por repetición de dígitos

```
--> x : 1.0 $
    for i : 1 thru 1000 step 1 do x : cos(x) $
    print(float(x)) $
```

```
--> y : 0.1 $
    x : cos(y) $
    for i : 1 while abs(x-y)>10^-8 step 1 do (
    y : x,
    x : cos(y),
    k : i
    )$
    print(k, "," , float(x)) $
```

Por cierto, cuidado con ciertas comparaciones:

```
--> is(cos(0)=1.0);
```

```
--> is(cos(0)=1);
```

```
--> is(cos(0.0)=1.0);
```

```
--> is(cos(0.0)=1);
```

# 5 Límite incorrecto por cancelación

Calculemos la derivada de la función seno en x=%pi a partir de la definición de derivada (es decir, como un límite).

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

```
--> f(x) := sin(x) $
    x : %pi,numer $
    for i : 1 thru 20 step 1 do (
        h : 10^-i,
        print([i, (f(x+h)-f(x))/h])
    ) $
```

Algunas veces obtenemos el resultado correcto (casi) por casualidad.

```
--> f(x) := sin(x) $
    x : 0.0 $
    for i : 1 thru 20 step 1 do (
        h : 10^-i,
        print([i, (f(x+h)-f(x))/h])
    ) $
```

Para practicar, mejora la salida en los dos ejemplos anteriores.

# 6 Sistema lineal mal condicionado

## 6.1 Ejemplo 1

Antes de nada, un programilla de cosecha propia para resolver sistemas a partir de la matriz de coeficientes y el vector de términos independientes, es decir, sin tener que escribir las ecuaciones completas.

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

```
--> /*Parece que Maxima no tiene algo así de serie*/

    linearsolve(A,b) := linsolve (
        makelist(
            (A.makelist(var[k],k,1,length(transpose(A)))-b)[k][1]=0, k, 1, length(A)
        ),
        makelist(var[k], k, 1, length(transpose(A)))
    ) $
```

Aplicamos el programa a un sistema concreto.

```
--> A : matrix(
        [-73, 78, 24],
        [-80, 37, 10],
        [92, 66, 25]
    ) $
    b : [29, 183, -33] $
    linearsolve(A,b);
```

Veamos que ocurre si variamos un poco una de las entradas de la matriz de coeficientes.

```
--> A[1,1]:-73+1/100;
```

```
--> A[1,1], numer;
```

```
--> linearsolve(A,b);
```

Introducimos float para "ver" mejor el resultado.

```
--> float(linearsolve(A,b));
```

## 6.2 Ejemplo 2

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

```
--> linearsolve(A,b) := linsolve (
        makelist(
            (A.makelist(var[k],k,1,length(transpose(A)))-b)[k][1]=0, k, 1, length(A)
        ),
        makelist(var[k], k, 1, length(transpose(A)))
    ) $
```

```
--> x : [1234, 4567] $
    A : matrix(
        [7891234, 4571833],
        [7891234*2-1, 4571833*2-1]
    ) $
    b : A.x $
    print(A.matrix([xx],[yy]),"=", b) $
```

```
--> linearsolve(A,b);
```

Veamos que pasa si variamos aleatoriamente las entradas de la matriz de coeficientes.

```
--> dA : genmatrix(lambda([i,j],(random(0.1)-0.05)),2,2);
```

```
--> float(linearsolve(A+dA,b)) ;
```

Parece que todo va bien. Pero si aumentamos un poco la perturbación:

```
--> ddA : genmatrix(lambda([i,j],(random(1.0)-0.5)),2,2);
```

```
--> float(linearsolve(A+ddA,b)) ;
```

Veamos que esta situación no está causada por el programa que hemos diseñado para resolver sistemas a partir de las matrices asociadas a un sistema.

```
--> linsolve(
        [7891234*xx+4571833*yy = 30617344067, 15782467*xx+9143665*yy = 61234682333],
        [xx, yy]
    );
```

Perturbando poco:

```
--> float( linsolve(
        [ (7891234+random(0.1)-0.05)*xx + (4571833+random(0.1)-0.05)*yy = 30617344067,
        (15782467+random(0.1)-0.05)*xx + (9143665+random(0.1)-0.05)*yy = 61234682333 ],
        [xx, yy]
        )
    );
```

Perturbando algo más:

```
--> float( linsolve(
        [ (7891234+random(1)-0.5)*xx + (4571833+random(1)-0.5)*yy = 30617344067,
        (15782467+random(1)-0.5)*xx + (9143665+random(1)-0.5)*yy = 61234682333 ],
       [xx, yy]
        )
    );
```

En los cálculos anteriores Maxima ha operado de acuerdo con las siguientes variables.

```
--> keepfloat;
```

```
--> ratprint;
```

```
--> describe(keepfloat) $
```

```
--> describe(ratprint) $
```

Repite el ejemplo 2 cambiando los valores de estas variables.