PRÁCTICA 6. ESPACIO VECTORIAL EUCLÍDEO. RESOLUCIÓN DE ECUACIONES Y SISTEMAS NO LINEALES. MÉTODOS NUMÉRICOS DE RESOLUCIÓN DE ECUACIONES

1 INTRODUCCIÓN.

El alumno debe ir realizando todos los EJEMPLOS que aparecen en esta práctica e ir anotando los resultados en la hoja que se le ha entregado. Además de estos ejemplos, hay planteados unos EJERCICIOS, que, a diferencia de lo que ocurre con los EJEMPLOS, no vienen desarrollados en el lenguaje Maxima, sino que tendrá que ser el alumno el que tenga que introducirlos para obtener el resultado.

2 PRODUCTO ESCALAR. ESPACIO VECTORIAL EUCLÍDEO.

Podemos calcular el producto escalar de dos vectores (o de dos listas) usando el comando "v.w"

```
--> /*EJEMPLO 1: Calculamos el producto escalar de los vectores
    (1,2,3) y (-1,5,0)*/

    [1,2,3].[-1,5,0];
```

Como sabemos, el producto escalar nos permite comprobar si dos vectores v y w son ortogonales (si v.w=0) o calcular la norma (o módulo) de un vector, puesto que modulo(v)=(v.v)^(1/2):

```
--> /*EJEMPLO 2: Calculamos el módulo de los vectores
    v=(1,2,3) y w=(-1,5,0)*/

    v:[1,2,3]$
    w:[-1,5,0]$
    modulo_v:(v.v)^(1/2);
    modulo_w:(w.w)^(1/2);
```

2.1 El método de Gram-Schmidt.

Con Maxima también podemos hacer otras muchas operaciones típicas del álgebra lineal, como, por ejemplo, es el método de ortogonalización de Gram-Schmidt. Para poder aplicar este método previamente hemos de cargar un paquete, lo que haremos en el siguiente ejemplo con "load(eigen)". El paquete eigen contiene funciones para el cálculo simbólico de valores y vectores propios. Maxima carga el paquete automáticamente si se hace una llamada a cualquiera de las dos funciones eigenvalues o eigenvectors. El paquete se puede cargar de forma explícita mediante load(eigen).

NOTA:Hay muchas aplicaciones, muchas ellas del álgebra lineal, que no podemos realizar directamente con el menú que lleva incluido wxMaxima, sino que tendremos que buscar comandos específicos para ellas.
Un buen manual donde encontrar todos estos comandos nos viene en la siguiente dirección:
http://maxima.sourceforge.net/docs/manual/es/maxima.html

En el ejemplo siguiente podemos observar como aplicar este método de G-S para ortogonalizar vectores:

```
--> /*EJEMPLO 3: Usar el método de G-S para ortogonalizar los
    vectores {(1,2,3),(9,18,30),(12,48,60)}*/

    load(eigen)$
    x:matrix([1,2,3],[9,18,30],[12,48,60]);
    y:gramschmidt(x);
```

NOTA: Por defecto, el método anterior se calcula usando el producto escalar usual. Pero es posible definir un producto escalar específico y aplicar el método anterior usando dicho producto. Veámoslo con un típico ejemplo desarrollado en clase:

Sea V=C([a,b],R) el conjunto de las funciones reales que son continuas en un intervalo de la forma [a,b]. Sabemos que V es un espacio vectorial. Podemos entonces definir en este espacio vectorial el producto escalar dado por
<f,g>=integral, en [a,b], del producto f(x).g(x)

Vamos a usar entonces este producto escalar para aplicar el método de G-S a los vectores (que son funciones) {1, sin(x), cos(x)} y calcular un conjunto de vectores (funciones) ortonormales a ellas; lo haremos en intervalo [a,b]=[-Pi/2, Pi/2]:

```
--> /*EJEMPLO 4*/

    load(eigen)$
    ip(f,g):=integrate(f*g,u,a,b);
    y:gramschmidt([1,sin(u),cos(u)],ip),a=-%pi/2,b=%pi;
```

EJERCICIO 1: Comprobar que, con el producto escalar definido por ip(f,g) del ejemplo anterior, las funciones que se obtienen en el mismo efectivamente son ortogonales a las funciones {1,sin(u),cos(u)}.

EJERCICIO 2: Calcular (usando la definición de producto escalar anterior) el módulo de cada uno de los vectores obtenidos en el ejemplo 8, y construir la correspondiente base ortonormal.

3 RESOLUCIÓN DE ECUACIONES Y SISTEMAS (no lineales).

En Maxima una ecuación consiste en dos expresiones vinculadas por el signo "=". Una ecuación por sí mismo no es evaluada; es decir, si tecleamos
x^4-5*x^3+2*x^2-5*x=x
el programa nos devuelve la misma expresión.

No debemos confundir "x : y" con "x = y" : El primero es una declaración imperativa (es decir, definimos la expresión de x dándole el valor de y) mientras que el segundo no es sino una ecuación.

3.1 Soluciones de ecuaciones algebraicas.

Para resolver cualquier tipo de ecuación o sistema algebraico con Maxima, podemos usar el comando "solve(ecu,x)", que nos resolverá (aunque no siempre, como veremos) la ecuación en términos de la variable x.

"solve" siempre trata de dar fórmulas explícitas para las soluciones de ecuaciones. Sin embargo, para ecuaciones suficientemente complicadas, no pueden darse soluciones algebraicas explícitas. Si se tiene una sola ecuación algebraica en una variable de grado menor o igual que 4, entonces Maxima puede dar la solución de la ecuación. Sin embargo, si el grado de la ecuación es 5 o mayor, no siempre es posible dar fórmulas algebraicas para las soluciones.

```
--> /*EJEMPLO 5: Resolvemos la ecuación x^4-5x^2-3=0*/

    kill(all)$ /*Con este comando eliminamos todos los valores
    que hayamos asignado a variables o funciones anteriormente*/
    solve(x^4-5*x^2-3=0,x);
```

```
--> /*EJEMPLO 6: Podemos resolver algunas ecuaciones de grado
    mayor que 4, como por ejemplo x^6-1=0*/

    solve(x^6-1=0,x);
```

aunque, como hemos comentado, no siempre es posible resolver ecuaciones de grado mayor que 4, como podemos ver si intentamos resolver la del siguiente ejemplo:

```
--> /*EJEMPLO 7: Intentamos resolver x^5-4x+2=0*/

    solve(x^5-4*x+2=0,x);
```

En este caso solve no nos da solución alguna. Entonces puede ser aconsejable que acudamos a métodos numéricos de resolución (y por tanto son métodos aproximados) de estas ecuaciones. Para ello podemos usar el comando "allroots(ecu)", que nos da aproximaciones numéricas de las raíces reales y complejas de ecu (válido para ecuaciones polinómicas con una variable): Si lo aplicamos a este último caso, tendremos:

```
--> /*EJEMPLO 8: Resolver numéricamente x^5-4x+2=0*/

    allroots(x^5-4*x+2=0);
```

Si solo estamos interesados en las raíces reales, podemos usar "realroots":

```
--> /*EJEMPLO 8 BIS: Calcular todas las raíces de x^5-6x^2+8x+3=0
    y determinar las que son reales*/

    allroots(x^5-6*x^2+8*x+3=0);
    realroots(x^5-6*x^2+8*x+3=0);
```

También puede usarse solve para resolver sistemas de ecuaciones, para lo que lo emplearemos en la forma "solve([ecu1,...,ecun],[x1,...,xn])". Este comando sirve para resolver un sistema de ecuaciones polinómicas simultáneas (ya sean lineales o no).

```
--> /*EJEMPLO 9: Resolver el sistema lineal
    ax+y=0; 2x+(1-a)y=1*/

    solve([a*x+y=0,2*x+(1-a)*y=1],[x,y]);
```

```
--> /*EJEMPLO 10: Resolver el sistema no lineal
    x^2+xy+y^2=4; x+xy+y=2*/

    solve([x^2+x*y+y^2=4,x+x*y+y=2],[x,y]);
```

Podemos usar "rectform" para obtener una mejor presentación. Por ejemplo, si lo aplicamos a la solución anterior, tendremos

```
--> /*EJEMPLO 11: Podemos obtener una mejor presentación de la
    solución anterior*/

    %,rectform;
```

EJERCICIO 3:
a) Hallar las raíces de x^3-3*x+1=0 mediante solve.

b) Hallar todas las raíces del polinomio x^5-2*x^3+x^2-1.

c) Resolver el sistema lineal -3*x+2*y=3; x-4*y=-1.

d) Resolver el sistema no lineal 3*x^2+2*y=0; x+2*y^2-y=-1.

3.2 Soluciones de ecuaciones trascendentes.

El paquete "to_poly_solver" será de gran utilidad para resolver con Maxima ecuaciones más complicadas. Para ello haremos en primer lugar

```
--> load(to_poly_solver)$
```

De esta forma llamamos a este paquete para, posteriormente, usar el comando "to_poly_solve(ec,inc,[options])", que intentará resolver la ecuación "ec" con incógnitas "inc".

```
--> /*EJEMPLO 12: Podemos resolver la ecuación algebraica
    2x^2-3x+5=0 usando este comando*/

    to_poly_solve(2*x^2-3*x+5=0,x);
```

```
--> /*EJEMPLO 13: Podemos resolver el sistema de ec. lineales
    2x-y=5; x+3y=1 usando este comando*/

    to_poly_solve([2*x-y=5,x+3*y=1],[x,y]);
```

```
--> /*EJEMPLO 14: Podemos resolver un sistema no lineal, como
    (x^2+1)(y^2+1)=10; (x+y)(xy-1)=3*/

    to_poly_solve([(x^2+1)*(y^2+1)=10,(x+y)*(x*y-1)=3],[x,y]);
```

A veces, es posible que no obtengamos solución para un sistema, como vemos en el siguiente ejemplo:

```
--> /*EJEMPLO 15: Intentamos resolver el sistema
    x^2+y^2=4; (x-1)^2+(y-1)=4*/

    to_poly_solve([x^2+y^2=4,(x-1)^2+(y-1)^2=4],[x,y]);
```

Entonces es posible resolverlo añadiendo la opción "use_grobner=true" (el porqué de esta opción está explicado en el manual que puede encontrarse en la web que aparece al principio de esta práctica):

```
--> /*EJEMPLO 15 BIS: Intentamos resolver el sistema
    x^2+y^2=4; (x-1)^2+(y-1)=4*/

    to_poly_solve([x^2+y^2=4,(x-1)^2+(y-1)^2=4],[x,y],
    use_grobner=true);
```

Vemos una ecuación que no puede resolverse mediante este comando:

```
--> /*EJEMPLO 16: Intentamos resolver exp(x^2-1)-x = 0*/

    to_poly_solve(exp(x^2-1)-x = 0,x);
```

EJERCICIO 4: Resolver las siguientes ecuaciones:

a) (x+x^(1/2))^(1/2)-(x-x^(1/2))^(1/2) = 3/2 (x/(x+x^(1/2)))^(1/2)

b) log(1+(x+1)^(1/2)) = log(x-40)

c) (cos(x))/(sin(x)+1) + (sin(x)+1)/(cos(x)) = 4

d) sin(x + Pi/6) = cos(x - Pi/4)

e) tan(x) = cotan(x)

f) sin(x^2+1) = cos(x)

4 MÉTODOS NUMÉRICOS DE RESOLUCIÓN DE ECUACIONES: MÉTODO DE NEWTON

Aunque los comandos que hemos visto en la última sección nos permiten resolver ecuaciones usando métodos numéricos, es aconsejable que realicemos una aproximación al método más usual de resolución numérica de ecuaciones, y que es el método de Newton (además, este método no da tiempo a explicarlo en las clases de teoría).

Este método se usa para resolver ecuaciones en una variable, dadas por una expresión f(x)=0, siendo f una función derivable. Si "a" es la solución que buscamos, la idea del método es construir una sucesión de valores a partir de un valor inicial x0, de la forma:
x1=g(x0); x2=g(x1);... xn+1=g(xn)
y tal que esta sucesión tenga por límite el punto "a", donde g(x) es una determinada función construida a partir de la f(x) inicial.

Evidentemente, habremos de imponer determinadas condiciones tanto para g(x) como para el punto inicial "a", que nos aseguren que la sucesión recurrente que construimos sea convergente (tenga límite finito).

Para garantizarnos la convergencia de este método, se actuará como sigue:

- Tomamos como punto inicial x0 un valor que sea próximo al de la raíz que queremos hallar. Para ello, usaremos el teorema de Bolzano, es decir, teniendo en cuenta que es fácil representar gráficamente la función f(x) usando wxMaxima, localizaremos un intervalo pequeño, que contenga a la raíz buscada, en el que f(x) tome valores de signo opuesto en sus extremos.
- Tomaremos un punto inicial x0 en dicho intervalo.
- Formaremos la sucesión recurrente dada por
x1=x0-f(x0)/f'(x0)
(es decir, x1 es el punto de corte con el eje OX de la recta tangente a la curva f(x) en el punto x0)
x2=x1-f(x1)/f'(x1)
x3=x2-f(x2)/f'(x2)
....

El porqué de las elecciones anteriores puede verse en cualquier libro de teoría (o simplemente tecleando "método de newton" en google y viendo cualquiera de las páginas que nos aconsejan visitar).

Vemos el desarrollo de este método en el ejemplo siguiente:

```
--> /*EJEMPLO 17: Intentamos resolver x^3+3*x+6= 0*/

    /*en primer lugar hacemos una gráfica de f(x)=x^3+3*x+6
    para hallar un intervalo pequeño donde se encuentre situada
    la raíz*/

    plot2d(x^3+3*x+6,[x,-3,3]);
```

```
--> /*EJEMPLO 17 (cont.): Se observa que la raíz está en [-2,-1].
    Intentamos hallar un intervalo más pequeño ampliando la gráfica*/

    plot2d(x^3+3*x+6,[x,-2,-1]);
```

```
--> /*EJEMPLO 17 (cont.): Se observa que la raíz está en [-1.4,-1.2]*/
    /*Entonces aplicaremos el método tomando como valor inicial x0
    cualquiera de los extremos de este intevalo, por ejemplo x0=-1.4*/
    /*En primer lugar introducimos las funciones f(x):=x^3+3*x+6
    g(x)=x-f(x)/f'(x)*/

    g(x):=x-(x^3+3*x+6)/(3*x^2+3);
```

En realidad solo hemos introducido g(x), puesto que hemos calculado "a mano" la expresión de f'(x). Entonces aplicamos el método iterativo comenzando por x0=-1.4:

```
--> x0=-1.4;
    x1=g(-1.4);
```

```
--> x2=g(%);
```

```
--> x3=g(%);
```

```
--> x4=g(%);
```

```
--> /*Observamos que a partir de x3 se mantienen fijos 9 decimales,
    por lo que podemos tomar como valor aproximado de la raíz buscada
    a=-1.287909750*/
```

No obstante, no es preciso realizar el anterior método iterativo para resolver una ecuación por el método de Newton, ya que wxMaxima tiene implementado dicho método, sin más que teclear lo que aparece desarrollado en el ejemplo siguiente:

```
--> /*EJEMPLO 18: Veamos como se resuelve la misma ecuación anterior
    usando el comando que wxMaxima dispone para este método*/

    /*en primer lugar hemos de cargar el paquete externo "newton1"*/

    load(newton1)$

    /*A continuación introducimos el comando del método, que es de la
    forma: "newton(f(x),x,x0,e)", siendo f(x)=0 la ecuación a resolver,
    x0 el valor inicial, y "e" una cota del error máximo permitido para
    aproximar la solución de la ecuación*/

    /*En nuestro caso, pondremos, si queremos aproximar la raíz con un
    error menor que una milésima*/

    newton(x^3+3*x+6,x,-1.4,0.001);
```

```
--> /*EJEMPLO 19: Resolver, utilizando el método de Newton, la ecuación
    1+cos(x)=e^x*/

    /*Hacemos una representación gráfica de f(x)=1+cos(x)-%e^x, y
    observamos que la raíz es próxima a 0.5*/
    plot2d(1+cos(x)-%e^x,[x,-1,2]);
```

```
--> /*Aplicamos entonces Newton a partir del valor inicial 0.5, y
    tomando como tolerancia para el error 0.001*/

    newton(1+cos(x)-%e^x,x,0.5,0.001);
```

EJERCICIO 5: Utilizando el método de Newton, hallar las raíces de la ecuación x^3+5*x^2-3*x-6=0, partiendo de los puntos
a) x0=-6
b) x0=-1
c) x0=1
Realizar previamente una representación gráfica de la ecuación para ver el porqué de esta elección de puntos iniciales.

EJERCICIO 6: Justificar gráficamente que la ecuación log(x)+x=0 tiene solución única y hallarla por el método de Newton.

EJERCICIO 7: Hallar la mayor de las soluciones de la ecuación sin(x)=e^x.