PRÁCTICA 3. APLICACIONES LINEALES, PRODUCTO ESCALAR Y 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 APLICACIONES LINEALES.

2.1 Matriz asociada a una aplicación lineal.

Una vez que sabemos introducir matrices y resolver sistemas de ecuaciones lineales, podemos resolver la mayoría de ejercicios relacionados con aplicaciones lineales. Comenzamos introduciendo la matriz asociada a una aplicación lineal:

```
--> /*EJEMPLO 1: Introducir la matriz asociada a la a. lineal
    f:R^4->R^3 dada por f(x,y,z,t)=(x-z,0,y-2z+t)*/

    f(x,y,z,t):=([x-z,0,y-2*z+t]);
    B:apply(matrix,[f(1,0,0,0),f(0,1,0,0),f(0,0,1,0),f(0,0,0,1)]);
    A:transpose(B);
```

NOTA: Seguramente será más rápido introducir directamente la matriz vista la expresión de la aplicación lineal:

```
--> /*EJEMPLO 1 BIS: Introducir la matriz asociada a la a. lineal
    f:R^4->R^3 dada por f(x,y,z,t)=(x-z,0,y-2z+t)*/

    A:matrix([1,0,-1,0],[0,0,0,0],[0,1,-2,1]);
```

2.2 Imagen y Núcleo de una aplicación lineal.

Sabemos que la Im(f) está generada por las columnas de la matriz de f, por lo que si escogemos las que son linealmente independientes, obtendremos una base para el subespacio vectorial Im(f).
Sin embargo, en wxMaxima se puede obtener más fácilmente una base para Im(f) sin más que usar directamente el comando "columnspace(A)" junto con "args(%)":

```
--> /*EJEMPLO 2: Vamos a hallar el subespacio Im(f), siendo
    f la apl. lineal anterior*/

    columnspace(A);
    args(%);
```

De esta forma observamos que dim(Im(f))=2, por lo que f no puede ser suprayectiva (de paso, también sabemos que dim(ker(f))=2).

De igual forma podemos hallar ker(f) (y sin necesidad de resolver un sistema de ecuaciones A.X=0), usando el comando "nullspace(A)" junto con "args(%)":

```
--> /*EJEMPLO 3: Vamos a hallar el subespacio ker(f), siendo
    f la apl. lineal anterior*/

    nullspace(A);
    args(%);
```

2.3 Cambio de base en una aplicación lineal.

Aunque ésto ya se vió en el ejercicio 6.3 de la práctica anterior, volvemos a incluirlo por su gran utilidad para los ejercicios planteados en clase y en exámenes. Recordamos que para efectuar un cambio de base en una a. lineal, realizamos un esquema
V -- > V --- > W -- > W
B1 C C B2
y que la nueva matriz en las bases B1,B2 viene dada por
MB1,B2(f) = MC,B2(id) x MC,C(f) x MB1,C(id)

```
--> /*EJEMPLO 4: Vamos a hallar la nueva matriz asociada a f
    cuando se considera en R^4 la base canónica y en R^3 la
    nueva base dada por B={(1,1,0),(1,1,1),(1,0,0)}*/

    A:matrix([1,0,-1,0],[0,0,0,0],[0,1,-2,1]);
    MBC_id:matrix([1,1,1],[1,1,0],[0,1,0]);
    MBC_id.A;
```

En el ejemplo anterior la nueva matriz se ha calculado realizando el producto matricial MC,B(f) = MB,C(id) x A
(notemos que, en este ejemplo, en el conjunto inicial R^4 seguimos teniendo la base canónica, por lo que solo se cambia de base en el conjunto final R^3.

EJERCICIO 1: Dada la aplicación lineal f:R^3-->R^2 definida por f(x,y,z)=(x-y-z,x-y-z)
a) Escribir la matriz asociada a f en las bases canónicas.
b) Hallar bases para Im(f) y ker(f). Clasificar f.
c) Hallar la nueva matriz asociada a f cuando se consideran las nuevas bases B1 y B2 dadas por
B1={(1,1,0),(1,1,1),(1,0,0)} y B2={(2,-1),(3,1)}

3 PRODUCTO ESCALAR. ESPACIO VECTORIAL EUCLÍDEO.

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

```
--> /*EJEMPLO 5: 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 6: 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);
```

3.1 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 7: 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 8*/

    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 2: 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 3: 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.

4 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.

4.1 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 9: 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 10: 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 11: 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 12: 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 12 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 13: 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 14: 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 15: Podemos obtener una mejor presentación de la
    solución anterior*/

    %,rectform;
```

EJERCICIO 4:
a) Hallar las raíces de x^3-3x+1=0 mediante solve.

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

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

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

4.2 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 16: 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 17: 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 18: 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 19: 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 19 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 20: Intentamos resolver exp(x^2-1)-x = 0*/

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

EJERCICIO 5: 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)