1.1 Vectores geométricos

1 - 1.1.7 Sintáxis Básica

Haremos algunos cálculos sencillos para ir calentando.

```
log(20);
```

Es probable que necesitemos el valor numérico de log(20).

```
log(20),numer;
```

```
420/16000;
```

```
420/16000,numer;
```

Podemos utilizar la función float para el mismo resultado

```
float(log(20)); float(420/16000);
```

El programa contiene una gran cantidad de funciones matemáticas básicas internas y entre ellas las trigonométricas:

```
sin(%pi/3);cos(%pi/3);tan(%pi/3);
```

Ahora recurriremos a una de las facilidades que nos ofrece el programa para agrupar objetos matemáticos: las listas. Las listas se escriben entre corchetes y los objetos de la lista separados con comas.

```
[sin(%pi/3),cos(%pi/3),tan(%pi/3)];
```

```
[sin(%pi/3),cos(%pi/3),tan(%pi/3)],numer;
```

Cuando necesitemos generar una lista por medio de alguna regla específica o formula usamos la función makelist. La sintaxis es la siguiente:

```
makelist(exp(t·x),t,1,10);
```

Podemos también aplicar una función a cada elemento de la lista, en este caso a cada elemento le aplicaremos la función ln(x). Para tal fin utilizaremos el comando map.

```
map(log,(makelist(exp(t·x),t,1,10)));
```

Si queremos, por ejemplo, sumar todos los elementos de la lista anterior utilizamos la función apply con la operación que queremos realizar. Aquí aprovecharemos para utilizar un atajo muy práctico que consiste en el uso del símbolo %, que toma la última salida del programa para ser usado en la instrucción siguiente, de esta manera nos evitamos volver a escribir toda la instrucción anterior. La sintaxis para todo esto es:

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

Podemos asignarle a una lista el nombre de una variable

```
L: [sin(%pi/3),log(3),sqrt(2),abs(x),exp(x^2)];
```

De manera que para aislar elementos de una lista escribimos:

```
L[1]; L[4];
```

Esto nos permite operar con sus elementos.

```
L[2]·(L[1]+ L[4])/L[5];
```

La primera utilidad que le podemos dar a las listas es que nos permite definir vectores. Si queremos que el programa interprete los vectores a = (a1,a2,a3), b = (b1,b2,b3), c = (c1,c2,c3), v0 = (0,0,0), los podemos escribir como listas:

```
a:[a1,a2,a3]; b:[b1,b2,b3];c:[c1,c2,c3]; v0:[0,0,0];
```

Como vimos, las operaciones básicas sobre los vectores cumplen un conjunto de propiedades:

```
a+b=b+a;
```

```
(a+b)+c=a+(b+c);
```

```
a+v0=a;
```

```
a-a=v0;
```

```
alpha·(a+b)=alpha·a+alpha·b,factor;
```

```
(alpha+beta)·a=alpha·a+beta·a,factor;
```

Con el siguiente comando limpiamos la memoria y borramos todos los cálculos anteriores

```
kill(all)$
```

FIN

2 - 1.2.8 Vectores en componentes

Con el programa de manipulación simbólica Maxima haremos algunos cálculos sencillos con vectores. Nuevamente recomendamos ver el apéndice 6.1 como introducción al programa.
Dados los vectores, en coordenadas cartesianas: a = i + 2j + 3k y b = 7i + 8j + 9k

```
a:[1,2,3];
```

```
b:[7,8,9];
```

La multiplicación por escalares y suma es simple, si queremos calcular α a + β b, escribimos:

```
α·a + β·b;
```

Para el producto escalar procedemos utilizando el operador punto, como se muestra a continuación.

```
a.b;
```

El cálculo de producto vectorial no es tan obvio, debemos cargar previamente la librería vect.

```
load(vect)$
```

```
express(a~b);
```

La norma de un vector, como ya vimos, es: √a · a

```
sqrt(a.a);
```

Si tenemos otro vector, digamos c = −4i + 5j − 6k, el producto triple: a · b × c se calcula así:

```
c:[-4,5,-6];
```

```
a.express(b~c);
```

El ángulo entre los vectores a y b.
En Maxima usamos la función acos(x) para el arcocoseno(x). Consultar el manual del programa para ver el resto de las funciones trigonométricas

```
acos((a.b)/(sqrt(a.a)·sqrt(b.b)));
```

Seguramente lo queremos es el valor numérico, esto se hace agregando la función float. Con la siguiente sintaxis logramos el objetivo:

```
acos((a.b)/(sqrt(a.a)·sqrt(b.b))),float;
```

Para finalizar, podemos considerar el problema de la independencia lineal de vectores. Tomemos el conjunto de vectores del ejemplo 1.2.7, es decir, los vectores: a = i+2j+3k, b = 4i+5j y c = 3i+2j+k

Disponemos de un comando que permite resolver sistemas de ecuaciones lineales, este comando se llama linsolve. Podemos escribir el sistema de ecuaciones como una lista y luego resolver:

```
ecus:[α+4·β+3·γ=0,2·α+5·β+2·γ=0,3·α+γ=0];
```

```
linsolve(ecus,[α,β,γ]);
```

```
kill(all)$
```

FIN

3 - 1.3.4 Aplicaciones del álgebra vectorial

Consideremos el plano determinado por los puntos P = (1, 1, 1), Q = (2, 4, 6) y R = (2, 2, 2). Para construir la ecuación del plano primero ingresamos los vectores posición de cada punto. También incorporaremos las librerías vect y draw que necesitaremos.

```
load(vect)$ load(draw)$
```

```
P:[1,1,1];
```

```
Q:[2,4,6];
```

```
R:[2,2,2];
```

Podemos verificar que los vectores estén en el mismo plano simplemente calculando el triple producto vectorial: P · (Q × R) entre ellos. Sabemos que si es nulo es porque los vectores son coplanares.

```
P.express(Q~R);
```

Necesitamos ahora calcular los vectores que van del punto P al punto Q y del punto P al punto R, es decir, los vectores: PQ = Q − P y PR = R − P

```
PQ:Q-P;
```

```
PR:R-P;
```

Un vector normal al plano será sencillamente el vector N = PQ × PR:

```
N:express(PQ~PR);
```

Podemos escoger cualquiera de los vectores originales, en este caso al vector P, para escribir la ecuación del plano: N · r = N · P.

```
r:[x,y,z];
```

```
plano:N.r=N.P;
```

Probemos a graficar el plano. Para saber mas sobre las diferentes opciones que incorpora Maxima para hacer gráficas, en dos o tres dimensiones, es recomendable consultar el manual de usuario. Utilizaremos el comando wxdraw3d para hacer que la figura aparezca embebida dentro de nuestra hoja de trabajo. Es recomendable consultar también las funciones draw y gr3d.

```
(%i)  wxdraw3d(implicit(plano,x,-1,1,y,-1,1,z,-1,1));
```