MÁXIMA TEMA 2

Los conjuntos se definen en Maxima tal y como lo hemos hecho antes, delimintando sus elementos entre llaves.

```
(%i1) A: {2,3,5,5,3,-1,2,1};
```

$$\{-1,1,2,3,5\}$$

En este ejemplo podemos observar que Maxima elimina los elementos repetidos aunque nosotros los hayamos escrito. También observamos que ha escrito los elementos, numéricos en este caso, de forma ordenada

También podemos definir conjuntos con el operador `set()`, que será útil en combinación con otros operadores.

```
(%i2) set(2,2,3,1,5);
```

$$\{1,2,3,5\}$$

Si queremos definir un conjunto de cadenas de caracteres (strings), debemos delimitarlas por comillas, en caso contrario, Maxima lo interpretará como un parámetro. En el siguiente ejemplo vamos a utilizar el parametro xx y también la cadena de caracteres "x".

```
(%i3) B: set(x,"x");
```

$$\{x,x\}$$

Aparentemente, el conjunto está formado por un único elemento repetido. Pero si ahora le damos un valor al parámetro xx, vemos que el conjunto está formado por la cadena 'x' y el valor que tome el parámetro xx.

```
(%i4) ev(B,x=1);
```

$$\{1,x\}$$

Cuando trabajamos con conjuntos grandes, puede ser útil el operador `elementp` que analiza si un elemento pertenece o no a un conjunto.

```
(%i5) C: {0,1,-1,3,5,-7};
```

$$\{-7,-1,0,1,3,5\}$$

```
(%i6) elementp(3,C);
```

true

```
(%i7) elementp(2,C);
```

false

En Maxima, disponemos de los operadores `subsetp` y `setequalp` para analizar la relación de inclusión o igualdad de dos conjuntos,

```
(%i1) A: {-2,-1,0,2,4,6}$
      B: {-1,0,2}$
      C: {-1,0,1}$
      D: {0,-1,-2,2,4,6}$
(%i2) subsetp(B,A);
```

true

```
(%i3) subsetp(C,A);
```

false

```
(%i4) setequalp(A,D);
```

true

El conjunto vacío lo podemos introducir con un par de llaves o con set()set().

```
(%i5) setequalp({},set());
```

true

```
(%i6) subsetp({},A);
```

True

UNIÓN

En `Maxima`, la unión de dos conjuntos se determina con el operador `union`.

```
(%i1) A: {-1,1,2,3,5}$
   B: {2,3}$
(%i2) union(A,B)
```

$\{-1,1,2,3,5\}$

INTERSECCIÓN

El operador de `Maxima` para calcular la intersección de dos conjuntos es `intersection`.

```
(%i1) A: {-1,1,2,3,5}$
      B: {2,4,5}$
(%i2) intersection(A,B);
```

$\{2,5\}$

Y `setdifference` determina la diferencia de dos conjuntos

```
(%i3) setdifference({1, 2, 8, 9}-{2, 6, 7});
```

$\{1,8,9\}$

```
(%i4) setdifference({2, 6, 7}-{1, 2, 8, 9});
```

$\{6,7\}$

DETERMINAR CONJUNTOS Y PRIMOS

Podemos definir conjuntos por comprensión en `Maxima` de dos formas, con los operadores `makeset` y `subset`. El operador `makeset` tiene tres argumentos: el primero es una expresión que determina los elementos del conjunto a partir de unos parámetros; el segundo argumento es la lista de los parámetros que se usan en el argumento anterior y el tercer argumento es una lista con los distintos valores que toman los parámetros para construir los elementos del conjunto.

```
(%i1) makeset(j/k,
              [j,k],
              [[1,a],[1,b],[2,b],[3,c]]);
```

$$\{1a,1b,2b,3c\}$$

```
(%i2) makeset(x^2,[x],{[2],[-2],[3]});
```

$$\{4,9\}$$

El operador `subset` es más parecido a la definición matemática de la construcción por compresión. Ese operado tiene dos argumentos: el primero es un conjunto previamente definido y el segundo es una propiedad, es decir, un operador de `Maxima` que actúe sobre un argumento y cuya salida sea `true` o `false`. Por ejemplo, el operador `evenp` es uno de estos operadores; aplicado a un número, nos devuelve `true` o `false` según el número sea par o impar:

```
(%i3) evenp(4);
```

$$\text{true}$$

```
(%i4) evenp(5);
```

$$\text{false}$$

Vamos a utilizar este operador para determinar el subconjunto de los números pares de un conjunto de números.

```
(%i5) A: {1,27,8,9,12}$
(%i6) subset(A,evenp);
```

$$\{8,12\}$$

En `Maxima` hay varios operadores que podemos usar como propiedades para definir subconjuntos, pero también podemos definir otras propiedades con el operador is()is() cuya sintaxis vemos en el siguiente ejemplo en el que definimos una propiedad que analiza si un número es o no múltiplo de 3.

```
(%i7) prop(x):= is(remainder(x,3)=0)$
(%i8) subset(A,prop);
```

$$\{9,12,27\}$$

Ya hemos dicho anteriormente que `Maxima` trabaja internamente con los conjuntos como si fueran listas, es decir, utilizando algún orden intrínseco de los elementos. De hecho, para definir y trabajar con nuestros propios conjuntos también es preferible utilizar la estructura de lista, es decir, con el operador `makelist` en lugar de `makeset` y con el operador `setify` que convierte listas en conjuntos.

```
(%i9) C: makelist(k^2,k,-5,5);
```

$$[25,16,9,4,1,0,1,4,9,16,25]$$

```
(%i10) setify(C);
```

$$\{0,1,4,9,16,25\}$$

Vemos otro ejemplo, en el que además hacemos uso de la propiedad `primep`, para determinar los números primos menores que 50.

```
(%i11) D: setify(makelist(i,i,1,50))$
(%i12) subset(D,primep);
```

$$\{2,3,5,7,11,13,17,19,23,29,31,37,41,43,47\}$$

El operador de `Maxima` para calcular la el conjunto de las partes o conjunto potencia es `powerset`.

```
(%i1) powerset({0,1,2,3});
```

$$\{\{\},\{0\},\{0,1\},\{0,1,2\},\{0,1,2,3\},\{0,1,3\},\{0,2\},\{0,2,3\},\{0,3\},\{1\},\{1,2\},\{1,2,3\},\{1,3\},\{2\},\{2,3\},\{3\}\}$$

PERMUTACIONES

En `Maxima`, disponemos de algunos operadores relacionados con los modelos de recuento que vamos a estudiar en este tema. Por ejemplo, el operador `permutations` permite obtener el conjunto de todas las permutaciones de los elementos de un conjunto.

```
(%i1) perm3: permutations({1,2,3});
```

$$\{[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]\}$$

Naturalmente, el resultado es un conjunto de listas. En este caso, estamos obteniendo permutaciones de todos los elementos del conjunto y por lo tanto, el número total coincide con el factorial del cardinal del conjunto.

```
(%i2) is(cardinality(perm3)=3!);
```

$$\mathrm{True}$$

Utilizando los operadores para trabajar con conjuntos que hemos aprendido, también podemos obtener variaciones. Por ejemplo, las 2-permutaciones de $A=\{1,2,3\}$A={1,2,3} son los elementos de $A\times A$A×A con las dos componentes distintas y por lo tanto, podemos obtener este conjunto como subconjunto del producto cartesiano.

```
(%i3) distintos(l):= is(l[1]#l[2])$
```

```
(%i4) distintos([1,1]);
```

$$False$$

```
(%i5) distintos([2,1]);
```

$$True$$

```
(%i6) perm23:
subset(cartesian_product({1,2,3},{1,2,3}),distintos)
;
```

$$\{[1,2],[1,3],[2,1],[2,3],[3,1],[3,2]\}$$

El cardinal de este conjunto es igual a $3\cdot 2=6$3·2=6:

```
(%i7) cardinality(perm23);
```

PERMUTACIONES GENERALIZADAS

En las permutaciones generalizas puede haber elementos repetidos, lo que significa que estamos permutando elementos de una lista. De hecho, el operador `permutations` de `Maxima` se puede aplicar a una lista y en ese caso, nos genera las permutaciones generalizadas.

```
(%i1) perm222: permutations([1,1,2,2,3,3]);
```

$$\{[1,1,2,2,3,3],[1,1,2,3,2,3],\ldots,[3,3,2,1,2,1],[3,3,2,2,1,1]\}$$

La expresión que determina el cardinal de este conjunto se denomina coeficiente multinomial y esa denominación es la que da nombre al operador en `Maxima`.

$$P(n;n_1,n_2,\ldots,n_r)=multinomial\backslash\_coeff(n_1,n_2,\ldots,n_r)=n!n_1!\cdot n_2!\cdots$$

```
(%i2) cardinality(perm222)=multinomial_coeff(2,2,2)

=6!/(2!*2!*2!);
```

$$90=90=90$$

COMNINACIONES

Ya habíamos visto que `binomial` es el operador de `Maxima` que determina los número combinatorios: $binomial(n,m)=(nm)$. Pero, ¿cómo determinamos los subconjuntos de un conjunto con un determinado número de elementos? Nuevamente, podemos recurrir a los operadores que hemos aprendido hasta ahora y, concretamente, a la forma de determinar subconjuntos definidos por una propiedad. Si queremos obtener los subconjuntos con 3 elementos, podemos usar la siguiente propiedad:

```
(%i1) card3(x):=is(cardinality(x)=3)$
```

De esta forma, los subconjuntos con 3 elementos del conjunto $\{1,2,3,4,5,6\}$$\{1,2,3,4,5,6\}$ se pueden determinar como sigue:

```
(%i2) subconj63:
subset(powerset({1,2,3,4,5,6}),card3);
```

$$\{\{1,2,3\},\{1,2,4\},\{1,2,5\},\{1,2,6\},\{1,3,4\},\{1,3,5\},\{1,3,6\},\{1,4,5\},\{1,4,6\},\{1,5,6\},\{2,3,4\},\{2,3,5\},\{2,3,6\},\{2,4,5\},\{2,4,6\},\{2,5,6\},\{3,4,5\},\{3,4,6\},\{3,5,6\},\{4,5,6\}\}$$

Naturalmente, el cardinal de este conjunto coincide con (63):

```
(%i3) cardinality(subconj63)=binomial(6,3);
```

$$20=20$$

COMBINACIONES CON REPETICIÓN

No disponemos de un operador en `Maxima` que genere las combinaciones con repetición, pero no es difícil hacerlo con los operadores que hemos aprendido hasta ahora. Concretamente, usaremos las soluciones de las ecuaciones lineales como ejemplo básico.

Por ejemplo, las combinaciones con repetición $CR(3,7)$ son las soluciones positivas de $x_1+x_2+x_3=7$ y por lo tanto, necesitamos generar listas de tres números positivos que sumen 7. Para que el proceso sea fácilmente generalizable, definiremos los operadores sin tener en cuenta la longitud de las misma. Empezamos por definir un operador que determina la suma de los elementos de una lista.

```
(%i1) suma_list(l):=lreduce("+",l)$
(%i2) suma_list([2,3,1,4]);
```

$$9$$

Para seleccionar las listas adecuadas, definimos una propiedad sobre listas, en este caso, que la suma sea igual a 7:

```
(%i3) suma7Q(l):=is(suma_list(l)=7)$
```

Ya podemos construir el subconjunto de las combinaciones con repetición.

```
(%i4) A: {0,1,2,3,4,5,6,7};
(%i5)
comb_rep37:=subset(cartesian_product(A,A,A),suma7Q);
```

{[0,0,7],[0,1,6],[0,2,5],[0,3,4],[0,4,3],[0,5,2],[0,6,1],[0,7,0],[1,0,6],[1,1,5],[1,2,4],[1,3,3],[1,4,2],[1,5,1],[1,6,0],[2,0,5],[2,1,4],[2,2,3],[2,3,2],[2,4,1],[2,5,0],[3,0,4],[3,1,3],[3,2,2],[3,3,1],[3,4,0],[4,0,3],[4,1,2],[4,2,1],[4,3,0],[5,0,2],[5,1,1],[5,2,0],[6,0,1],[6,1,0],[7,0,0]}

```
(%i6) cardinality(comb_rep37);
```

36

```
(%i7) binomial(3+7-1,7);
```

36

NUMEROS STIRLING

En `Maxima` disponemos tanto del operador que calcula los números de Stirling de segunda especie, `stirling2`, como del operador que determina las particiones de un conjunto con un determinado número de partes, `set_partitions`.

```
(%i1) partes74: set_partitions({1,2,3,4,5,6,7},4)$
(%i2) cardinality(partes74)=stirling2(7,4);
```

350=350

FUNCIONES GENERADORAS

Naturalmente, si utilizamos `Maxima`, no necesitaremos realizar las transformaciones del ejemplo anterior. Por ejemplo, vamos a determinar el número de soluciones de la ecuación $x_1+x_2+x_3=30$x1+x2+x3=30 teniendo en cuenta que $1\geq x_1\geq 10$1≥x1≥10, $2\geq x_2\geq 7$2≥x2≥7, $1\geq x_3$1≥x3. Vamos a hacerlo de dos formas. En la primera vamos a utilizar que, dado que la suma de los tres sumandos es 30, ninguno de ellos puede ser mayor que 30, es decir, $x_3\geq 30$x3≥30. De esta forma, la función generadora de este problema es el siguiente polinomio:

```
(%i1) gen:
sum(z^n,n,1,10)*sum(z^n,n,2,7)*sum(z^n,n,1,30),expand;
```

Z^47+3z^46+6z^45+10z^44+15z^43+21z^42+27z^41+33z^40 +39z^39+45z^38+50z^37+54z^36+57z^35+59z^34++60z^33 +60z^32+60z^31+60z^30+60z^29+60z^28+60z^27++60z^26 +60z^25+60z^24+60z^23+60z^22+60z^1+60z^20++60z^19+

$$60z^{18}+59z^{17}+57z^{16}+54z^{15}+50z^{14}+45z^{13}++39z^{12}+33z^{11}+27z^{10}+21z^{9}+15z^{8}+10z^{7}+6z^{6}+3z^{5}+z^{4}$$

Aunque podemos ver fácilmente cual es el coeficiente de z^{30}z30, también podemos determinarlo con el operador `coeff`. Hay que tener en cuenta que para que el resultado sea correcto, debemos aplicar este operador a una expresión polinómica en su forma expandida, tal y como estamos haciendo en este ejemplo.

```
(%i2) coeff(gen,z,30);
```

$$60$$

La función generadora anterior solo nos sirve para encontrar el numero de soluciones de $x_1+x_2+x_3=m$x1+x2+x3=m si $m\geq 30$m≥30, ya que hemos añadido la restricción $x_3\geq 30$x3≥30. Si queremos hallar la función generadora que podamos utilizar para cualquier valor de mm con las restricciones que habíamos indicado al principio, $1\geq x_1\geq 10$1≥x1≥10, $2\geq x_2\geq 7$2≥x2≥7, $1\geq x_3$1≥x3., tendremos que usar series:

```
(%i3) gen1:
sum(z^n,n,1,10)*sum(z^n,n,2,7)*sum(z^n,n,1,inf)$
```

En este caso, no podemos recurrir directamente el operador `coeff`, ya que la expresión no está expandida:

```
(%i4) coeff(gen1(z),z,30);
```

$$0$$

La alternativa a la opción `expand` cuando trabajamos con series es el operador `taylor`, que determina el polinomio de Taylor de la función generadora hasta el grado que deseemos y que en este caso coincide con la forma expandida truncada en ese grado:

```
(%i5) taylor(gen1,z,0,35);
```

$$z^{4}+3z^{5}+6z^{6}+10z^{7}+15z^{8}+21z^{9}+27z^{10}+33z^{11}+39z^{12}+45z^{13}+50z^{14}++54z^{15}+57z^{16}+59z^{17}+60z^{18}+60z^{19}+60z^{20}+60z^{21}+60z^{22}+60z^{23}++60z^{24}+60z^{25}+60z^{26}+60z^{27}+60z^{28}+60z^{29}+60z^{30}+60z^{31}+60z^{32}++60z^{33}+60z^{34}+60z^{35}+\ldots$$

```
(%i6) coeff(%,z,30);
```

$$60$$

ECUACIONES EN RECURRENCIA

En este primer ejemplo con Maxima vamos a utilizar el paquete solve_rec, que resuelve directamente ecuaciones en recurrencia lineales (homogéneas o no).

```
(%i1) load(solve_rec)$
```

Primero introducimos la ecuación, en donde la sucesión a determinar se escribe como una lista. Si escribimos solo una expresión involucrando elementos de la lista, Maxima entiende que la expresión está igualada a 0. Por ejemplo, la ecuación en recurrencia de orden 3 dada por $18a_n - 21a_{n-1} + 8a_{n-2} - a_{n-3} = 0$ se introduce como sigue:

```
(%i2) ecrec: 18*a[n]-21*a[n-1]+8*a[n-2]-a[n-3]$
```

El operador que resuelve la ecuación es solve_rec y toma como argumentos la ecuación y la incógnita.

```
(%i3) solve_rec(ecrec,a[n]);
```

$$a[n] = \frac{\%k_3 * n + \%k_2}{3^n} + \frac{\%k_1}{2^n}$$

En este caso, nos devuelve la solución general en función de tres parámetros, $\%k_1$%k1, $\%k_2$%k2 y $\%k_3$%k3, ya que la ecuación es de orden 3.

También podemos añadir, como argumentos, las condiciones iniciales de la ecuación y, en tal caso, nos devolverá la solución particular correspondiente.

```
(%i1)
solve_rec(ecrec,a[n],a[0]=1,a[1]=5/6,a[2]=17/36);
```

$$a[n] = \frac{n}{3^n} + \frac{1}{2^n}$$

CON EC.CARACTERÍSTICA

En este ejemplo, vamos resolver una ecuación en recurrencia usando la ecuación característica. Concretamente, vamos a resolver la ecuación que determina la sucesión de Fibonacci:

$$f[n] - f[n-1] - f[n-2] = 0, \quad f[0] = 0, \quad f[1] = 1.$$

La ecuación característica es $r^2 - r - 1 = 0$:

```
(%i1) eccar: r^2-r-1$
(%i2) solve(eccar,r);
```

$$r = -\frac{\sqrt{5}-1}{2}, \quad r = \frac{\sqrt{5}+1}{2}$$

Por lo tanto, el esquema de la solución de la ecuación en recurrencia es:

```
(%i3) sol: A*(-(sqrt(5)-1)/2)^n+B*((sqrt(5)+1)/2)^n$
```

Para hallar los valores de A y B evaluamos esta expresión con $n = 0$ y $n = 1$ e igualamos los resultados a los correspondientes valores iniciales:

```
(%i4) ec1: ev(sol,n=0)=0$
(%i5) ec2: ev(sol,n=1)=1$
```

El operador `solve` nos da los valores de A y B:

```
(%i6) solve([ec1,ec2],[A,B]);
```

$$A = \frac{-1}{\sqrt{5}}, \quad B = \frac{1}{\sqrt{5}}$$

Por lo tanto, la solución de la ecuación en recurrencia que nos da la forma explícita de la sucesión de Fibonacci:

$$a[n] = \frac{-1}{\sqrt{5}} \cdot \left(\frac{-\sqrt{5}+1}{2}\right)^n + \frac{1}{\sqrt{5}} \cdot \left(\frac{\sqrt{5}+1}{2}\right)^n$$

El resultado obtenido con el operador `solve_rec` es naturalmente el mismo.

```
(%i7) load(solve_rec)$
(%i8) solve_rec(f[n]-f[n-1]-f[n-2]=0, f[n],f[0]=0,
f[1]=1.);
```

$$f[n] = \frac{(\sqrt{5}+1)^n}{\sqrt{5} \cdot 2^n} - \frac{(\sqrt{5}-1)^n(-1)^n}{\sqrt{5} \cdot 2^n}$$

EJEMPLO

$a_0 = 5$, $a_1 = 6$, $a_n = 3a_{n-1} - 2a_{n-2} - 2^{n-1}$

Dado que $2^n = 1 \cdot 2^n$, la ecuación característica de esta recurrencia es

$$(r^2 - 3r + 2)(r-2)^{0+1} = (r^2 - 3r + 2)(r-2) = 0$$

```
(%i1) load(solve_rec)$
(%i2) solve_rec(a[n] = 3*a[n-1] - 2*a[n-2] - 2^(n-
1),a[n], a[0]=5,a[1]=6);
```

$$a[n] = -n \cdot 2^n + 3 \cdot 2^n + 2$$