

LMD

Práctica 2. Inducción y recurrencia

Como ya sabemos, Maxima numera las sucesivas líneas de entrada como (%i1), (%i2), etc. En estos guiones de prácticas, para indicar que el alumno ha de ejecutar un comando, escribimos (%ixx) seguido del comando en cuestión.

Si `expr` es una expresión aritmética, el comando `sum(expr,k,a,b)` calcula la suma de los valores que va tomando `expr` cuando la variable `k` varía (de forma ascendente) desde el valor `a` hasta el valor `b`. Puede ocurrir que `expr` dependa o no de la variable `k`. Ejecute el comando siguiente:

```
(%ixx) sum(3*k^2,k,1,7);
```

En este ejemplo `expr` es $3 \cdot k^2$ y hemos calculado la suma $3 \cdot 1^2 + 3 \cdot 2^2 + \dots + 3 \cdot 7^2$.

Observe lo que ocurre cuando ejecutamos `sum(expr,k,a,b)`, siendo $a > b$.

```
(%ixx) sum(3*k^2,k,7,1);
```

En este caso, por definición se obtiene siempre cero. Veamos otros ejemplos.

```
(%ixx) 1/1+1/4+1/9+1/16+1/25+1/36+1/49+1/64;
```

Las fracciones que estamos sumando tienen una forma especial, por lo que este comando lo podemos escribir también como:

```
(%ixx) sum(1/k^2,k,1,8);
```

¿Y si la expresión `expr` no depende de la variable `k`?

```
(%ixx) sum(3,k,1,10);
```

El resultado era de esperar. Hemos sumado diez veces el número 3.

El comando `sum` también funciona cuando `expr` es una expresión en la que intervienen matrices. En el siguiente ejemplo definimos una matriz de orden 2×2 , que llamamos `A`, y calculamos la suma de sus diez primeras potencias. (No confunda A^k con A^k .)

```
(%ixx) A:matrix([-1,2],[1/2,1]);
```

```
(%ixx) sum(A^k,k,1,10);
```

Cuando ejecutamos el comando `sum(expr,k,a,b)`, tanto `a` como `b` pueden ser expresiones a las cuales no se les ha asignado ningún valor. Hasta este momento, el símbolo `n`

no tiene asignado ningún valor.

```
(%ixx) n;
```

Vamos a intentar calcular una *fórmula* cerrada para la suma $1 + 2 + 3 + \dots + n$, es decir, el número de sumandos en dicha fórmula es constante y no depende de n .

```
(%ixx) sum(k,k,1,n);
```

En este caso Maxima devuelve una expresión sin simplificar. Podemos obligar a Maxima que intente simplificar la expresión, añadiendo al final la opción `simplsum`.

```
(%ixx) sum(k,k,1,n), simplsum;
```

Así obtenemos la fórmula cerrada que ya conocemos de un ejercicio de teoría.

Ejercicio. Escriba comandos en Maxima que calculen fórmulas cerradas para:

1. $1^2 + 2^2 + 3^2 + \dots + n^2$.
2. $1 + a^2 + \dots + a^n$, donde $a \in \mathbb{R} \setminus \{0, 1\}$.

□

El problema de simplificar expresiones es un problema difícil y no siempre se tiene éxito. Pruebe con el siguiente comando.

```
(%ixx) sum(1/((2*k-1)*(2*k+1)),k,1,n),simplsum;
```

Podemos utilizar el paquete `simplify_sum`, que cargamos de la forma siguiente.

```
(%ixx) load (simplify_sum);
```

Maxima lo carga y además devuelve un aviso al cual no le hacemos caso.

```
(%ixx) simplify_sum(sum(1/((2*k-1)*(2*k+1)),k,1,n));
```

Como vemos, Maxima ahora sí devuelve su forma “simplificada”.

Probamos con otros ejemplos.

```
(%ixx) sum(k*5^k,k,1,n), simplsum;
```

```
(%ixx) simplify_sum(sum(k*5^k,k,1,n));
```

```
(%ixx) sum(k*2^k+1,k,1,n), simplsum;
```

```
(%ixx) simplify_sum(sum(k*2^k+1,k,1,n));
```

```
(%ixx) simplify_sum(sum(k*2^k+1,k,1,n)), simplsum;
```

Análogamente a lo que hemos visto para la suma, Maxima también incluye el comando `product(expr,k,a,b)` que calcula el producto de los valores devueltos por la expresión `expr` cuando la variable `k` varía (de forma ascendente) desde `a` hasta `b`. Maxima también acepta que escribamos `prod` en vez de `product`. Ejecute los siguientes comandos:

```
(%ixx) prod(k,k,1,5);
```

Con ésto hemos calculado el factorial de 5.

```
(%ixx) prod(2,k,1,5);
```

Hemos calculado $2^5 = 32$.

```
(%ixx) prod(7,k,5,3);
```

Vemos así que cuando $a > b$, el comando `product(expr,k,a,b)` devuelve 1.

```
(%ixx) prod(2^k,k,0,5);
```

En este último comando estamos multiplicando los números $2^0 = 1, 2^1, 2^2, 2^3, 2^4, 2^5$.

Pero sabemos que las potencias con la misma base se multiplican sumando los exponentes, por lo cual el resultado tiene que ser igual a

$$2^{0+1+2+3+4+5} = 2^{15} = 32768.$$

Puestos a rizar el rizo, este mismo resultado lo podríamos haber obtenido con el siguiente comando:

```
(%ixx) 2^sum(k,k,0,5);
```

Si el límite superior ó el límite inferior del producto no es un número concreto, se plantea el mismo problema que ya vimos con la suma. Pruebe con el siguiente comando:

```
(%ixx) prod(k,k,1,n);
```

La forma de obtener la expresión simplificada es:

```
(%ixx) prod(k,k,1,n), simpproduct;
```

La habilidad de Maxima es ahora más limitada. Por ejemplo, vea la igualdad siguiente:

$$\prod_{k=1}^n (2 \cdot k) = 2^n \cdot n!$$

Maxima es incapaz de obtener la expresión del miembro de la derecha a partir de la expresión del miembro de la izquierda:

```
(%ixx) prod(2*k,k,1,n), simpproduct;
```

Tampoco tiene éxito en los ejemplos siguientes:

```
(%ixx) product ((k+1)/(k+2), k, 1, n),simpproduct;
```

```
(%ixx) prod(2^k,k,1,n), simpproduct;
```

Existe un paquete llamado `simplify_products` para simplificar productos.

```
(%ixx) load (simplify_products);
```

Observe que a la hora de invocar el paquete, la palabra “product” se escribe en singular.

```
(%ixx) simplify_product(prod(2*k,k,1,n));
```

```
(%ixx) simplify_product(product ((k+1)/(k+2), k, 1, n));
```

```
(%ixx) simplify_product(prod(2^k,k,1,n));
```

Ejercicio: Obtenga a mano el resultado simplificado para este último ejemplo. \square

Maxima también nos puede servir de ayuda cuando queremos comprobar si una propiedad se cumple o no para los primeros valores para los cuales está definida.

Veamos un ejemplo. ¿Es verdad que $a_n = 7^{2n} + 16n - 1$ es múltiplo de 64 para todo entero $n \geq 0$?

Podemos generar una lista con los primeros valores de a_n , o mejor todavía, generar una lista con los restos de la división de los valores a_n entre 64. Esto se puede hacer de la forma siguiente:

```
(%ixx) makelist(mod(7^(2*n)+16*n-1,64),n,0,100);
```

En la ayuda de Maxima puede encontrar más información sobre los comandos `makelist` y `mod` que hemos usado.

El resultado que obtenemos nos hace pensar que muy probablemente a_n sea múltiplo de 64 para todo entero $n \geq 0$, pero no constituye ninguna demostración. Recuerde: Si lanzamos una moneda cinco veces y en todas ellas obtenemos cara, ¿podemos asegurar que en la sexta tirada saldrá también cara? Por tanto, ahora tendríamos que plantearnos hacer una demostración por inducción de la propiedad planteada.

Ejercicio. ¿Es verdad que $b_n = n^4 - 12n^3 + 49n^2 - 78n + 40$ es múltiplo de 5 para todo entero $n \geq 1$? \square

Ejercicio. Encuentre una fórmula lo más simple posible para cada una de las expresiones siguientes:

- $\sum_{k=0}^n 3^{2k}$ para $n \geq 0$.
- $\sum_{k=0}^n k \cdot 3^{2k}$ para $n \geq 0$.
- $1 + 3 + 5 + \dots + (2n - 1)$ para $n \geq 1$.
- $n^3 + 4(n - 1)^3 + 4^2(n - 2)^3 + \dots + 4^{n-1}$ para $n \geq 1$.
- $\sum_{k=n}^{2n} (k \cdot 3^k + k^3)$ para $n \geq 0$.
- $\sum_{k=0}^n \binom{n}{k}$ para $n \geq 0$.
- $\sum_{k=0}^n k \cdot 3^k \binom{n}{k}$ para $n \geq 0$.
- $\sum_{k=0}^n \binom{2n}{2k}$ para $n \geq 0$.
- $\prod_{k=1}^n \frac{k}{k+10}$ para $n \geq 1$.
- $\prod_{k=1}^n \frac{k^2}{k+10}$ para $n \geq 1$.

$$\blacksquare \prod_{k=1}^n \frac{k+3}{4k} \text{ para } n \geq 1.$$

$$\blacksquare \prod_{k=1}^n \frac{4^k}{6^k} \text{ para } n \geq 1.$$

□

Ejercicio. ¿Es verdad que para todo entero $n \geq 0$ el valor de la expresión $f(n) = n^2 + n + 41$ es un número primo? Sugerencia: Puede usar la función `primep(x)` de Maxima, la cual devuelve `true` si x es primo, y `false` en caso contrario. □

Ejercicio. Encuentre el menor entero positivo k de modo que para todo entero $n \geq k$ se verifique que $n! \geq 11^n$. □

Ejercicio. Verifique para $1 \leq n \leq 1000$ que $(n+1)(n+2) \cdots (n+n)$ es múltiplo de 2^n , y a continuación demuéstrelo por inducción. □

Ejercicio. Para cada una de las matrices siguientes, A_k , proponga una fórmula para $(A_k)^n$, con $n \in \mathbb{N}$:

$$A_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}; \quad A_2 = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix}; \quad A_3 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix};$$

$$A_4 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}; \quad A_5 = \begin{pmatrix} 0 & 0 & a \\ 0 & b & 0 \\ c & 0 & 0 \end{pmatrix}.$$

□

En Maxima podemos definir funciones numéricas tales como $f(x) = x^2$ o bien $g(x, y) = 4x + y^2$ de la manera siguiente:

```
(%ixx) f(x):=x^2;
```

```
(%ixx) g(x,y):=4*x+y^2;
```

Seguidamente veremos cómo se definen en Maxima sucesiones de números reales dadas por recurrencia. Supongamos que $f(n)$ es el término general de una sucesión de números reales definida por:

$$f(n) = g(f(n-1), f(n-2), \dots, f(n-k), n),$$

donde $g : \mathbb{R}^{k+1} \rightarrow \mathbb{R}$ es una función dada. Entonces escribiremos una expresión tal como la siguiente `f[n] := g(f[n-1], f[n-2], ..., f[n-k], n)`.

Las condiciones iniciales de la recurrencia

$$f(0) = v_0, \quad f(1) = v_1, \dots, f(k-1) = v_{k-1}$$

se introducen como `f[0]:v0; f[1]:v1;...;f[k-1]:v_{k-1};`.

Observe que ahora no escribimos $f(n)$ sino $f[n]$. El uso de los corchetes indica a Maxima que tiene que memorizar los valores que se van calculando para ir obteniendo los siguientes.

Además vemos que para dar las condiciones iniciales se usa el símbolo de asignación de valores a variables “:=” en vez del símbolo “=” que se usa para definir la recurrencia.

Veamos un ejemplo concreto. La sucesión de los números de Fibonacci se define como

$$\begin{cases} f(0) = 0, & f(1) = 1; \\ \forall n \geq 2, & f(n) = f(n-1) + f(n-2). \end{cases}$$


Podemos definir en Maxima dicha sucesión de la manera siguiente:

```
(%ixx) f[0]:0;
(%ixx) f[1]:1;
(%ixx) f[n]:=f[n-1]+f[n-2];
```

Nótese que aquí $k = 2$, que es el orden de la recurrencia, y $g: \mathbb{R}^3 \rightarrow \mathbb{R}$ es la función definida por $g(x, y, z) = x + y$.

A continuación calculamos la lista formada por sus 21 primeros valores:

```
(%ixx) makelist(f[n],n,0,20);
```

Recordemos que al principio habíamos definido la función $f(x) = x^2$. Maxima distingue entre $f(n)$ y $f[n]$. 

```
(%ixx) f(10); f[10];
```

La sucesión de los números de Fibonacci viene predefinida en Maxima mediante el comando `fib(n)`. Nótese que aquí se usan paréntesis en vez de corchetes.

Con el comando siguiente calculamos la diferencia entre los primeros valores de ambas sucesiones:

```
(%ixx) makelist(f[n]-fib(n),n,0,20);
```

También se puede usar la construcción `if ... then ... else` en la definición de sucesiones por recurrencia. Vamos a definir de esta nueva forma la sucesión de los números de Fibonacci, aunque ahora usamos como nombre de función la letra `j` para que la nueva definición no entre en conflicto con la que ya teníamos:

```
(%ixx) j[n]:=if n=0 then 0 else
             if n=1 then 1 else
             j[n-1]+j[n-2];
```

Calculamos sus primeros valores:

```
(%ixx) makelist(j[n],n,0,20);
```

Y la comparamos con la sucesión `fib(n)` ya predefinida.

```
(%ixx) makelist(fib(n)-j[n],n,0,20);
```

La sucesión recurrente siguiente no es lineal, ya que $f(n)$ no es combinación lineal de los términos anteriores $f(n-1)$, $f(n-2)$, $f(n-3)$:

$$\begin{cases} f(0) = 1, f(1) = 4, f(2) = 5; \\ \forall n \geq 3, f(n) = f(n-1) \cdot f(n-2) + f(n-3) + 2n. \end{cases}$$

Aquí $k = 3$ y $g : \mathbb{R}^4 \rightarrow \mathbb{R}$ es $g(x_1, x_2, x_3, x_4) = x_1 \cdot x_2 + x_3 + 2x_4$. Antes de definir esta nueva sucesión, que queremos llamar de nuevo **f**, liberamos el símbolo **f** para que no entre en conflicto con la definición del ejemplo previo.

```
(%ixx) kill(f);
```

Para que vea lo que ocurre si no somos cautos con ésto, observe la siguiente secuencia de comandos.

```
(%ixx) f[n]:=2;
```

Hemos definido la sucesión constantemente igual a 2. Evaluamos **f** en **n=2014**.

```
(%ixx) f[2014];
```

Ahora usamos el símbolo **f** para definir la sucesión constantemente igual a 8.

```
(%ixx) f[n]:=8;
```

Evaluamos en algunos valores y, ¡sorpresa!

```
(%ixx) f[2013]; f[2014]; f[2015]; f[2016];
```

```
(%ixx) kill(f);
```

Definimos la sucesión propuesta como sigue:

```
(%ixx) f[n]:=if n=0 then 1 else
              if n=1 then 4 else
              if n=2 then 5 else
              f[n-1]*f[n-2]+f[n-3]+2*n;
```

Calculamos el valor de **f[20]**.

```
(%ixx) f[20];
```

Como el número es muy grande, sólo muestra en pantalla los dígitos iniciales y finales.

Por tanto, para definir en Maxima sucesiones por recurrencia, hemos de escribir los argumentos entre corchetes. Además, tenemos dos posibilidades. Una, definir las condiciones iniciales y el término general de la recurrencia en líneas separadas. Otra, usar la estructura if...then...else.

La función factorial ya está implementada en Maxima. Para calcular el factorial de n , simplemente escribimos **n!**. Por ejemplo, introduzca los siguientes comandos:

```
(%ixx) 0!; 1!; 2!; 3!; 10!;
```

Ejercicio. Defina para todo entero $n \geq 0$, una función recurrente `fac[n]` que calcule el factorial de n . (Puede consultar los apuntes del Tema 2 donde dimos la definición recurrente de la función factorial.) \square

Ahora definimos en Maxima la sucesión $f(0) = 1$, $f(n) = f(n-1) + 4\sqrt{f(n-1)} + 4$ para $n \geq 1$.

Nos proponemos encontrar una expresión no recurrente para $f(n)$. Nótese que esta recurrencia no es lineal, pues $f(n)$ depende de la raíz cuadrada de $f(n-1)$, por lo que no podemos usar la técnica de resolución de recurrencias estudiada en el Tema 2.

Como seguimos empeñados en usar el símbolo f , comenzamos liberando la variable f .

```
(%ixx) kill(f);
(%ixx) f[0]:1;
(%ixx) f[n]:=f[n-1]+4*sqrt(f[n-1])+4;
```

A continuación, obtemos la lista de los primeros diez valores de f y analizamos el resultado obtenido.

```
(%ixx) makelist(f[k],k,0,9);
```

Quizá resulte más claro tras ejecutar el comando siguiente.

```
(%ixx) factor(%);
```

Parece ser $f(n) = (2n+1)^2$, para $n \geq 0$. Por facilidad a la hora de referirnos a esta fórmula, definimos

```
(%ixx) g(n):=(2*n+1)^2;
```

Podemos construir la tabla de las diferencias entre $f(n)$ y la expresión propuesta para los primeros valores de n .

```
(%ixx) makelist(f[n]-g(n),n,0,30);
```

Y efectivamente, coinciden totalmente. Por tanto ahora ya tendríamos una buena base para iniciar una demostración por inducción de la propiedad $f(n) = (2n+1)^2$, para $n \geq 0$. Dejamos los detalles para el alumno.

A continuación veremos cómo se pueden resolver las recurrencias lineales en Maxima. Comenzamos liberando las definiciones hechas.

```
(%ixx) kill(all);
```

Maxima incluye un paquete para resolver recurrencias lineales. Para cargarlo en memoria, ejecutamos:

```
(%ixx) load(solve_rec);
```


Vamos a resolver la recurrencia lineal homogénea siguiente:

$$\begin{cases} f(0) = 0, f(1) = 1; \\ \forall n \geq 2, f(n) = 11f(n-1) - 28f(n-2). \end{cases}$$

Para ello escribimos

```
(%ixx) solve_rec(f[n]=11*f[n-1]-28*f[n-2], f[n], f[0]=0, f[1]=1);
```

Es importante mencionar que el comando `solve_rec` no define ninguna sucesión. Sólo resuelve la recurrencia y muestra el resultado en pantalla.

```
(%ixx) f[n];
```

Como puede apreciar, Maxima no sabe nada acerca de `f`.

Por otra parte, con el segundo argumento en el comando `solve_rec` le indicamos a Maxima qué variable queremos que utilice para representar el término general no recurrente. Escriba lo siguiente y entenderá mejor lo que trato de decirle:

```
(%ixx) solve_rec(f[n]=11*f[n-1]-28*f[n-2], f[x], f[0]=0, f[1]=1);
```

Otra posibilidad al invocar el comando `solve_rec` es no especificar condiciones iniciales, en cuyo caso se obtendrá la solución general de la recurrencia. En dicha solución aparecerán unos parámetros que se designan como `%k1`, `%k2`, etc. Pruebe con el siguiente comando.

```
(%ixx) solve_rec(f[n]=11*f[n-1]-28*f[n-2], f[n]);
```

Ahora resolvemos la recurrencia de los números de Fibonacci:

```
(%ixx) solve_rec(f[n]=f[n-1]+f[n-2], f[n], f[0]=0, f[1]=1);
```

Escrita un poco más decentemente, la fórmula obtenida sería

$$f(n) = \frac{1}{\sqrt{5}} \cdot \left(\left(\frac{1 + \sqrt{5}}{2} \right)^n - \left(\frac{1 - \sqrt{5}}{2} \right)^n \right), \quad \text{para todo } n \geq 0,$$

la cual se denomina *la fórmula de Binet*.

De manera similar podemos invocar al procedimiento `solve_rec` para resolver recurrencias lineales no homogéneas. Pruebe con los siguientes ejemplos:

```
(%ixx) solve_rec(f[n] = 3*f[n-1]-2*f[n-2]+1, f[n], f[0]=0, f[1]=1);
```

```
(%ixx) solve_rec(f[n] = 3*f[n-1]-2*f[n-2]+5^n*(3*n-10),  
f[n], f[0]=0, f[1]=1);
```

```
(%ixx) solve_rec(f[n] = 3*f[n-1]-2*f[n-2]+5^n*n^2+2^n*n+1,  
f[n], f[0]=0, f[1]=1);
```

Cuando tenemos una sucesión de números reales definida mediante una recurrencia lineal y las raíces características son números complejos no reales, hemos visto en clase que podemos obtener una fórmula alternativa que evita los números complejos. A cambio,

hemos de usar además las funciones trigonométricas seno y coseno. Pruebe el siguiente comando:

```
(%ixx) solve_rec(f[n]=2*f[n-1]-2*f[n-2], f[n], f[0]=1, f[1]=2);
```

Como ve, Maxima no es perfecto. Hemos visto en clase que la solución de esta recurrencia se puede expresar como

$$f(n) = (\sqrt{2})^n \cdot \left(\cos\left(\frac{n\pi}{4}\right) + \sin\left(\frac{n\pi}{4}\right) \right).$$

Vamos a comprobar que efectivamente se da la igualdad para los 1000 primeros valores de n .

```
(%ixx) f[n]:=2*f[n-1]-2*f[n-2];
```

```
(%ixx) f[0]:1; f[1]:2;
```

```
(%ixx) g(n):=sqrt(2)^n*(cos(n*pi/4)+sin(n*pi/4));
```

```
(%ixx) setify(makelist(f[n]-g(n),n,0,999));
```

Con este último comando, primero hemos generado la lista de las diferencias desde $n = 0$ hasta 999, y seguidamente, dicha lista la hemos convertido en un conjunto. Recuerde que al transformar listas en conjuntos, los elementos repetidos se escriben sólo una vez. Como el conjunto que obtenemos es $\{0\}$, ello significa que $f(n) = g(n)$ para todo $n = 0, 1, \dots, 999$.

Maxima incluso es capaz de resolver algunas recurrencias, también denominadas lineales, en las que los coeficientes de la parte derecha de la igualdad no son constantes, es decir, dependen de la variable n . Se aprecia que las soluciones no siempre serán tan elegantes como en los casos estudiados anteriormente.

```
(%ixx) kill(f);
```

```
(%ixx) solve_rec(f[n] = n*f[n-1], f[n], f[0]=1);
```

```
(%ixx) solve_rec(f[n] = 3*n*f[n-1]+2, f[n], f[0]=1);
```

```
(%ixx) solve_rec(f[n] = a*n*f[n-1]+b^n, f[n], f[0]=1);
```

Ejercicio. Una misma tarea, que depende de un número natural n , puede ser resuelta mediante dos métodos distintos que denominamos A y B. Se sabe que el número de pasos empleado por el método A viene dado por la recurrencia

$$f(n) = 2f(n-1) - f(n-2) \text{ para } n \geq 2; \quad f(0) = 4, f(1) = 1205,$$

mientras que el número de pasos empleado por B viene dado por

$$g(n) = 3g(n-1) - 2g(n-2) \text{ para } n \geq 2; \quad g(0) = 1, g(1) = 3.$$

Determine en función de n cuál de los dos métodos hay que aplicar. □

Cuando tenemos una sucesión $f(n)$, a veces nos interesa calcular los valores de n menores o iguales que cierto número dado n_0 tales que $f(n)$ verifique una condición dada.

Por ejemplo, vamos a calcular los números naturales $0 \leq n \leq 1000$ tales que $f(n) = 2n^2 + 1$ es un número primo.

```
(%ixx) A:setify(makelist(k,k,0,1000))$
```

Con ésto hemos definido un conjunto **A** formado por todos los números naturales menores o iguales que 1000. Nótese que al final del comando escribimos el símbolo **\$** en vez de punto y coma, pues no nos interesa que Maxima presente en pantalla tales números.

Ahora escribimos una función auxiliar que se aplica a un número natural n y devuelve **true** si y sólo si $f(n)$ es primo.

```
(%ixx) test(n):=primep(2*n^2+1);
```

Finalmente, calculamos el subconjunto de valores de **A** para los cuales la función **test** vale **true**.

```
(%ixx) subset(A,test);
```

Ejercicio. Sea la recurrencia

$$f(n) = 3f(n-1) - 3f(n-2) + f(n-3) \quad \text{para } n \geq 0; \quad f(0) = 1, f(1) = 5, f(2) = 17.$$

1. Calcule todos los números naturales n menores o iguales que 1500 tales que $f(n)$ es un número primo.
2. Calcule todos los números naturales n menores o iguales que 1500 tales que $f(n)$ es un múltiplo de 13. (Sugerencia: Aplique el procedimiento anterior pero ahora construya la función **test** usando, con los argumentos apropiados, el comando **is(x=y)** que vale **true** si y sólo si $x=y$.)

□

Ya hemos visto al principio de esta práctica cómo podemos utilizar Maxima para obtener expresiones cerradas para sumatorias cuyo número de sumandos depende de un natural n , como por ejemplo $f(n) = \sum_{k=1}^n k^8$. Ejecutamos el comando:

```
(%ixx) sum(k^8,k,1,n), simpsum;
```

Una forma alternativa de obtener ésto mismo, consiste en plantear una recurrencia para $f(n)$ y resolverla. Tenemos

$$f(1) = 1; f(n) = f(n-1) + n^8 \quad \text{para } n \geq 2.$$

Ahora la resolvemos.

```
(%ixx) solve_rec(f[n]=f[n-1]+n^8, f[n], f[1]=1);
```

```
(%ixx) rat(%);
```

Ejercicio. Para cada uno de los apartados siguientes, determine una recurrencia lineal homogénea cuya solución es la expresión dada en dicho apartado definida para $n \geq 0$:

1. $2n^3 + 7$
2. $n^2 + (-1)^n \cdot n + 6$
3. $n \cdot (2^{2n} + 1)$
4. $3\sin(n\pi/2) + 2\cos(n\pi/2)$
5. $(3\sin(n\pi/2) + 2\cos(n\pi/2)) \cdot (n + 1)$
6. $2^n \cdot (3\sin(n\pi/2) + 2\cos(n\pi/2)) \cdot (n + 1)$

A continuación resuelva cada una de las recurrencias obtenidas en los apartados 1, 2 y 3 para comprobar que su solución no recurrente es la expresión dada al principio. \square

Ejercicio. Se sabe que

$$1, 1, 1, 3, 17, 83, 345, 1291, 4513, 15075.$$

son los primeros términos de una sucesión definida para $n \geq 0$, la cual verifica una recurrencia lineal homogénea de orden menor o igual que 5. Determine una expresión no recurrente para dicha sucesión. \square

Ejercicio. Repita el mismo ejercicio anterior ahora para la secuencia

$$0, 0, 6, 10, 20, 28, 42, 54, 72, 88.$$

Ejercicio. Sea la sucesión siguiente definida para todo $n \geq 2$:

$$a_n = 1 \cdot 2 - 2 \cdot 3 + 3 \cdot 4 - \cdots + (-1)^n (n - 1)n.$$

Encuentre una expresión lo más reducida posible para a_n , en primer lugar usando el comando `simplify_sum`. A continuación, planteando una recurrencia lineal y resolviéndola. \square

Ejercicio. Encuentre una expresión no recurrente para cada una de las sucesiones de término general a_n y b_n , definidas ambas para $n \geq 0$, verificando que $a_0 = 2$, $b_0 = 1$, y

$$\begin{cases} a_n = 2a_{n-1} + b_{n-1}, \\ b_n = 3a_{n-1} + 4b_{n-1}, \end{cases}$$

para todo $n \geq 1$. A continuación resuelva cada una de las recurrencias. \square

Ejercicio. Para $n \geq 1$ se define la siguiente sucesión de números reales:

$$x_1 = a, \quad x_2 = b, \quad x_n = \frac{x_{n-1}x_{n-2}}{2x_{n-2} - x_{n-1}} \quad \text{para } n \geq 3.$$

Encuentre una expresión no recurrente para x_n . ¿Qué condiciones hemos de exigir sobre a y b para que la sucesión dada esté bien definida? \square