Capítulo 3

Aritmética entera y modular.

3.1. Aritmética entera.

De acuerdo con la teoría, dados dos números enteros, a, b, con $b \neq 0$, podemos realizar la división de a entre b, y eso nos da un cociente q y un resto r, satisfaciendo $a = b \cdot q + r$ y $0 \leq r < |b|$. Por ejemplo, si dividimos 27 entre 5, nos da de cociente 5 y resto 2. Maxima calcula el cociente y el resto mediante las instrucciones `quotient` y `mod`.

```
(%ixx) quotient(27,5);
(%oxx) 5
(%ixx) mod(27,5);
(%oxx) 2
(%ixx) quotient(605,31);
(%oxx) 19
(%ixx) mod(605,31);
(%oxx) 16
```

pues $605 = 31 \cdot 19 + 16$.

Sin embargo, tal y como Maxima trae implementadas las funciones `quotient` y `mod`, si `a` es negativo, y `b` es positivo, no es cierto que `a=b*quotient(a,b) + mod(a,b)`.

```
(%ixx) quotient(23,7); mod(23,7);
(%oxx) 3
(%oxx) 2
(%ixx) quotient(-23,7); mod(-23,7);
(%oxx) -3
(%oxx) 5
```

En la división de -23 entre 7 debería devolvernos de cociente -4, pues el múltiplo de 7 que más se acerca por la izquierda a -23 es -28. No obstante, calcula el resto no negativo, tal y como hemos establecido en teoría.

Además cuando el divisor es negativo, el resto nos lo devuelve negativo.

```
(%ixx) quotient(23,-7); mod(23,-7);
(%oxx) -3
(%oxx) -5
(%ixx) quotient(-23,-7); mod(-23,-7);
(%oxx) 3
(%oxx) -2
```

Para evitar ésto, definimos nosotros las funciones `cociente` y `modulo` de forma que el resto de una división siempre sea positivo o cero, y se satisfaga la igualdad $a = b \cdot \text{cociente}(a,b) + \text{modulo}(a,b)$. Para ello vamos a usar el operador condicional `if` y el operador lógico `or`:

```
(%ixx) modulo(x,y):=if (y>0 or mod(x,y)=0) then mod(x,y) else mod(x,y)-y$
```

Después de `if` viene una expresión lógica. Si se evalúa como verdadera, entonces se ejecuta lo que va después de `then`. Si se evalúa como falsa, se ejecuta lo que va después de `else`. En este caso, la expresión lógica es la disyunción de dos expresiones más simples. Por tanto, van separadas por `or`. Para que se evalúe como verdadera hace falta que al menos una de las dos expresiones `y>0` y `mod(x,y)=0` se evalúe como verdadera.

```
(%ixx) cociente(x,y):=quotient(x-modulo(x,y),y)$
```

```
(%ixx) cociente(23,-5); modulo(23,-5);
(%oxx) -4
(%oxx) 3
(%ixx) cociente(-23,-5); modulo(-23,-5);
(%oxx) 5
(%oxx) 2
```

Las funciones `gcd` (abreviatura de "*greatest common divisor*") y `lcm` (abreviatura de "*least common multiple*") calculan el máximo común divisor y el mínimo común múltiplo, respectivamente, de dos números enteros (y también de polinomios).

```
(%ixx) gcd(4,6); lcm(4,6);
(%oxx) 2
(%oxx) 12
```

```
(%ixx) gcd(45,123); lcm(45,123);
(%oxx) 3
(%oxx) 1845
```

La función `lcm` admite más de dos argumentos, no así la función `gcd`.

```
(%ixx) lcm(4,6,8);
(%oxx) 24
```

Si introducimos `gcd(4,6,8)`, nos da un mensaje de error.

Además la función `gcd` sólo devuelve valores no negativos, lo cual concuerda con las definiciones de teoría.

```
(%ixx) gcd(-4,10); gcd(4,-10); gcd(-4,-10); gcd(-4,0);
(%oxx) 2
(%oxx) 2
(%oxx) 2
(%oxx) 4
```

Sin embargo, para la función `lcm` el signo del resultado depende de los signos de los argumentos.

```
(%ixx) lcm(4,10); lcm(4,-10); lcm(-4,10); lcm(-4,-10);
(%oxx) 20
(%oxx) -20
(%oxx) -20
(%oxx) 20
```

Sabemos que si $d = mcd(a, b)$, entonces podemos encontrar dos enteros u, v, llamados unos coeficientes de Bezout para a y b, tales que $d = a \cdot u + b \cdot v$. Estos coeficientes, Maxima los calcula con la función `gcdex`, que significa "*gcd extended*".

```
(%ixx) gcdex(45,123);
(%oxx) [11,-4,3]
```

Lo que nos dice que el máximo común divisor de 45 y 123 vale 3 (el último de la lista) y los coeficientes u y v son respectivamente 11 y -4, es decir, $3 = 45 \cdot 11 + 123 \cdot (-4)$.

Dada una ecuación diofántica de la forma $ax+by=c$, sabemos que tiene solución si, y sólo si, $mcd(a,b)|c$. En tal caso, para encontrar una solución particular podemos obtener previamente una solución particular para $ax+by=mcd(a,b)$ y luego multiplicarla por $\frac{c}{mcd(a,b)}$. Por ejemplo, vamos a buscar una solución particular de la ecuación $2475x+7548y=114$.

En primer lugar vemos si la ecuación tiene solución.

```
(%ixx) d:gcd(2475,7548); mod(114,d);
(%oxx) 3
(%oxx) 0
```

Como tiene solución, calculamos primero una solución particular de la ecuación $2475x+7548y=3$.

```
(%ixx) gcdex(2475,7548);
(%oxx) [-491,161,3]
```

Resulta $x_0=-491, y_0=161$.

Por tanto una solución particular de $2475x+7548y=114$ es

$$x=-491\cdot\tfrac{114}{3}=-18658,$$
$$y=161\cdot\tfrac{114}{3}=6118.$$

En Maxima

```
(%ixx) x0:-491*114/3; y0:161*114/3;
(%oxx) -18658
(%oxx) 6118
```

Podemos automatizar todo este proceso, con una función que vamos a llamar `diofantica`, y que tendrá tres argumentos.

```
(%ixx) diofantica(a,b,c):=if mod(c,gcd(a,b))=0 then rest(gcdex(a,b),-1)*c/gcd(a,b)
 else print("No tiene solucion");
```

Aquí hemos usado la función `rest` la cual tiene dos argumentos: una lista y un número. La función `rest(lista,n)` devuelve la lista que resulta de eliminar los n primeros elementos de `lista` si n es positivo, y la lista que resulta de eliminar los $-n$ últimos elementos de `lista` si n es negativo. En nuestro caso, al hacer `rest(gcdex(a,b),-1)` lo que hacemos es eliminar el último elemento de la lista `gcdex(a,b)`, que es precisamente el máximo común divisor de a y b, y así nos quedamos con los coeficientes u y v.

Lo primero que la función `diofantica` hace es comprobar si la ecuación $ax+by=c$ tiene solución. Para esto, comprueba si el resto de la división de c entre el máximo común divisor de a y b es cero. En caso afirmativo, da una solución, multiplicando los coeficientes u y v por $\frac{c}{mcd(a,b)}$. En caso negativo, nos dice que no tiene solución.

```
(%ixx) diofantica(2475,7548,114);
(%oxx) [-18658,6118]
(%ixx) diofantica(2465,7548,49);
No tiene solucion
(%oxx) No tiene solucion
```

Una vez encontrada una solución particular (x_0,y_0) de la ecuación $ax+by=c$, todas las soluciones pueden calcularse mediante la expresión

$$x=x_0+k\cdot\tfrac{b}{d}$$
$$y=y_0-k\cdot\tfrac{a}{d}$$

Si quisiéramos encontrar una solución de una ecuación de la forma $ax+by+cz=e$, el criterio para ver si tiene o no solución es el mismo. Como sabemos de teoría, la ecuación tiene solución si, y sólo si, $mcd(a,b,c)|e$. Una forma de resolverla sería llamar d al máximo común divisor de a y b, resolver $du+cz=e$, y una vez hecho esto, resolver $ax+by=du$ (en lugar de haber tomado a y b podríamos

haber tomado a y c, o b y c). Por ejemplo, vamos a encontrar una solución de $6x+10y+15z=7$. Puesto que $mcd(6,10)=2$, resolvemos $2u+15z=7$.

```
(%ixx) diofantica(2,15,7);
(%oxx) [-49,7]
```

Y ahora resolvemos $6x+10y=2\cdot(-49)$

```
(%ixx) diofantica(6,10,2*(-49));
(%oxx) [-98,49]
```

Luego una solución particular es $x_0=-98$, $y_0=49$, $z_0=7$.

Maxima también tiene implementados algunos aspectos relacionados con los números primos. Ya vimos en una práctica anterior cómo el comando `primep` nos decía si un número es o no primo.

Para factorizar un número como producto de potencias de primos distintos, tenemos la función `factor`.

```
(%ixx) factor(24);
(%oxx) 2^3*3
(%ixx) factor(60984);
(%oxx) 2^3*3^2*7*11^2
```

También podemos usar la función `ifactors`. Esta función, aplicada a un número nos devuelve una lista, en la que cada elemento de la lista vuelve a ser una lista con dos entradas. Un primo, divisor del número, y el exponente al que aparece elevado ese primo en la descomposición.

```
(%ixx) ifactors(256);
(%oxx) [[2,8]]
(%ixx) ifactors(30);
(%oxx) [[2,1],[3,1],[5,1]]
(%ixx) ifactors(60984);
(%oxx) [[2,3],[3,2],[7,1],[11,2]]
```

Lo que significa que el número 60984 se factoriza como $2^3\cdot 3^2\cdot 7\cdot 11^2$.

Para obtener el conjunto de los divisores positivos de un número tenemos la función `divisors`.

```
(%ixx) divisors(24);
(%oxx) {1,2,3,4,6,8,12,24}
(%ixx) divisors(256);
(%oxx) {1,2,4,8,16,32,64,128,256}
```

Si ahora quisiéramos obtener una lista con los divisores primos de un número, podríamos definir la función `div_primos`.

```
(%ixx) div_primos(n):=makelist(ifactors(n)[i][1],i,1,length(ifactors(n)))$
(%ixx) div_primos(60984);
(%oxx) [2,3,7,11]
(%ixx) div_primos(256);
(%oxx) [2]
```

Recuerde que `ifactors(n)` devuelve una lista formada por tantos elementos como indica la función `length(ifactors(n))`, cada uno de los cuales a su vez es una lista de longitud 2. Con `ifactors(n)[i][1]` estamos indicando la primera componente de la lista de longitud 2 que aparece en la posición `i`.

También podríamos obtener los divisores primos con la siguiente función.

```
(%ixx) div_primos2(n):=subset(divisors(n),primep)$
(%ixx) div_primos2(60984);
(%oxx) {2,3,7,11}
(%ixx) div_primos2(256);
```

```
(%oxx) [2]
```

Sin embargo esta última función tiene el inconveniente de que requiere construir el conjunto de todos los divisores positivos del número dado, lo cual puede ser altamente ineficiente.

Por último, mencionamos dos comandos que nos permiten obtener un número primo. Estos son `next_prime` y `prev_prime`, que nos dan el siguiente número primo, y el anterior número primo de un número dado.

```
(%ixx) next_prime(2); next_prime(4); next_prime(20);
       next_prime(12345); next_prime(-10);
(%oxx) 3
(%oxx) 5
(%oxx) 23
(%oxx) 12347
(%oxx) 2
(%ixx) prev_prime(10); prev_prime(20); prev_prime(12345);
(%oxx) 7
(%oxx) 19
(%oxx) 12343
```

Si le pregunta a Maxima por el primo anterior a 2, da error. Recuerde que el 1 no es primo.

3.2. Aritmética modular.

Vimos que la función `mod` nos calcula el resto de una división entera. O si queremos, podemos usar la función `modulo` que definimos previamente.

Entonces, para calcular sumas, restas y productos módulo un número m, es decir, el resto de la división entre m, no tenemos más que introducir la expresión como el primer argumento de la función `mod`. Por ejemplo, si queremos calcular el valor de la expresión $6 \cdot 9 - 4 \cdot (5 - 2)$ módulo 11, podemos escribir lo siguiente.

```
(%ixx) mod(6*9-4*(5-2),11);
(%oxx) 9
```

Pues el resultado de la operación $6 \cdot 9 - 4 \cdot (5 - 2)$ es 42, que al dividirlo por 11 da resto 9. Más adelante veremos otra forma de reducir expresiones módulo un entero positivo.

También podemos reducir potencias.

```
(%ixx) mod(12^31,47);
(%oxx) 34
```

Aunque Maxima dispone de un comando específico para el cálculo de potencias modulares. Este es `power_mod`.

```
(%ixx) power_mod(12,31,47);
(%oxx) 34
```

Y es conveniente calcularlas usando esta función. Sobre todo si trabajamos con números grandes. Por ejemplo, podemos ejecutar

```
(%ixx) power_mod(13579,123456789,987654321);
(%oxx) 691505902
```

Pero si quisiéramos ejecutar `mod(13579^123456789,987654321)`, Maxima probablemente se bloquearía, pues en primer lugar intenta calcular el número entero $13579^{123456789}$ y luego trata de reducirlo módulo 987654321. El número anterior es un número de más de 510 millones de cifras.

Escriba al azar tres números x, y, z de aproximadamente 100 cifras, y calcule `power_mod(x,y,z)`. Comprobará cómo el cálculo es inmediato.

El comando `power_mod` puede usarse también con exponentes negativos.

Tomando como exponente -1 nos calcula un inverso de un número módulo otro.

```
(%ixx) power_mod(5,-1,9);
(%oxx) 2
```

Ésto significa que 2 es un inverso de 5 módulo 9, es decir, $5 \cdot 2 \equiv 1 \mod 9$. Equivalentemente, $5^{-1} = 2$ en $\mathbb{Z}_9$.

```
(%ixx) power_mod(5,-2,9);
(%oxx) 4
```

Como $5^{-1} = 2$ en $\mathbb{Z}_9$, entonces $5^{-2} = (5^{-1})^2 = 2^2 = 4$ en $\mathbb{Z}_9$, como nos ha calculado antes.

Si quisiéramos calcular el inverso de a módulo m, y tal inverso no existe (pues $mcd(a, m) \neq 1$), Maxima nos devuelve `false`.

```
(%ixx) power_mod(6,-1,9);
(%oxx) false
```

El exponente debe ser un número entero (positivo o negativo). No admite, por ejemplo, como exponente $\frac{1}{2}$.

Para el cálculo de inversos modulares, Maxima dispone de una función específica. Ésta es `inv_mod`.

```
(%ixx) inv_mod(11,23);
(%oxx) 21
(%ixx) inv_mod(7,1000001);
(%oxx) 428572
```

Ésto significa que $11^{-1} = 21$ en $\mathbb{Z}_{23}$ y $7^{-1} = 428572$ en $\mathbb{Z}_{1000001}$.

Maxima trae implementada la aritmética modular. Para ello dispone de una variable global, `modulus`, cuyo valor por defecto es `false`, pero que podemos asignarle cualquier valor natural mayor que cero (también admite el valor cero, pero luego da error al realizar los cálculos). Si introducimos un valor para `modulus` que no sea primo, Maxima nos da un aviso, pero admite ese valor.

Sin embargo, para que nos muestre el resultado como queremos, tenemos que decirle que nos simplifique la expresión, mediante el comando `rat`.

```
(%ixx) modulus:11$
(%ixx) 6+7;
(%oxx) 13
(%ixx) rat(6+7);
(%oxx)/R/ 2
(%ixx) rat(7*8);
(%oxx)/R/ 1
```

La función `rat` lo que hace es simplificar expresiones racionales. Por ejemplo:

```
(%ixx) (x^2-1)/(x+1);
```

(%oxx) $\frac{x^2-1}{x+1}$

Si ahora introducimos la misma expresión, pero precedida del comando `rat`, vemos cómo la simplifica.

```
(%ixx) rat((x^2-1)/(x+1));
(%oxx) x-1
```

Cuando dentro de la función `rat` introducimos una expresión numérica (por ejemplo, `4*5`), lo que entiende Maxima por simplificar es reducirla módulo el valor que tenga en ese momento la variable `modulus`.

La representación que trae Maxima de los enteros módulo m, comprende los números desde $\frac{-m+1}{2}$ hasta $\frac{m-1}{2}$, si m es impar, y desde $\frac{-m}{2} + 1$ hasta $\frac{m}{2}$, si m es par.

Recuerde que estamos trabajando módulo 11.

```
(%ixx) rat(8+9);
(%oxx)/R/ -5
(%ixx) makelist(rat(i),i,0,20);
(%oxx)/R/ [0,1,2,3,4,5,-5,-4,-3,-2,-1,0,1,2,3,4,5,-5,-4,-3,-2]
(%ixx) modulus:10$
warning: assigning 10, a non-prime, to 'modulus'
(%ixx) makelist(rat(i),i,0,20);
(%oxx)/R/ [0,1,2,3,4,5,-4,-3,-2,-1,0,1,2,3,4,5,-4,-3,-2,-1,0]
```

Cuando la variable `modulus` toma un valor distinto de `false`, Maxima realiza sus cálculos módulo ese valor. De esta forma, puede ser que obtengamos resultados no esperados al llamar a ciertas funciones.

```
(%ixx) gcd(6,27);
(%oxx) 1
```

Si le damos a `modulus` su valor por defecto, entonces realiza el cálculo correctamente.

```
(%ixx) modulus:false$ gcd(6,27);
(%oxx) 3
```

Dado un número natural n, definimos el conjunto de las unidades de $\mathbb{Z}_n$, y lo denotamos por $U(\mathbb{Z}_n)$, como el conjunto de todos los números enteros positivos menores que n que tienen inverso módulo n.

Por ejemplo, para $n = 28$, este conjunto es $\{1, 3, 5, 9, 11, 13, 15, 17, 19, 23, 25, 27\}$. Vamos a ver cómo podemos obtenerlo con Maxima.

En primer lugar tomamos el conjunto de todos los enteros no negativos menores que 28.

```
(%ixx) Z28:setify(makelist(i,i,0,27));
(%oxx) {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27}
```

Y ahora nos quedamos con el subconjunto de los que son primos relativos con 28. Para introducir la condición de que el máximo común divisor del elemento y 28 valga 1, vamos a emplear una función anónima ó función lambda. (Consulte la ayuda de Maxima para más información.)

```
(%ixx) UZ28:subset(Z28, lambda([x],is(gcd(x,28)=1)));
(%oxx) {1,3,5,9,11,13,17,19,23,25,27}
```

`lambda([x],is(gcd(x,28)))` es la función que para cada elemento x pregunta si $mcd(x, 28)$ vale 1 y devuelve `true` en tal caso, y `false` en caso contrario.

Con lo visto, podemos definir una función que para cada número natural $n > 1$ calcule el conjunto $U(\mathbb{Z}_n)$.

```
(%ixx) UZ(n):=subset(setify(makelist(i,i,1,n-1)),lambda([x],is(gcd(x,n)=1)))$
(%ixx) UZ(28);
(%oxx) {1,3,5,9,11,13,17,19,23,25,27}
(%ixx) UZ(30);
(%oxx) {1,7,11,13,17,19,23,29}
```

La función φ de Euler, aplicada a un número natural $n > 1$, calcula el número de elementos menores que n que son primos relativos con n. Maxima emplea para ello el comando `totient`.

```
(%ixx) totient(28); totient(30);
(%oxx) 12
(%oxx) 8
```

que son justamente los cardinales de los conjuntos $U(\mathbb{Z}_{28})$ y $U(\mathbb{Z}_{30})$, respectivamente.

```
(%ixx) is(totient(28)=cardinality(UZ(28)));
(%oxx) true
(%ixx) is(totient(30)=cardinality(UZ(30)));
(%oxx) true
```

```
(%ixx) is(totient(500)=cardinality(UZ(500)));
(%oxx) true
```

Sabemos, por el teorema de Euler-Fermat, que si $mcd(a,m)=1$ entonces $a^{\varphi(m)} \equiv 1$ (mód m). Vamos a hacer algunas comprobaciones con Maxima. Recordemos que para $m=28$, se verifica que $\varphi(m)=12$ y $U(\mathbb{Z}_m)=\{1,3,5,9,11,13,17,19,23,25,27\}$.

```
(%ixx) uz28:listify(UZ(28))$
(%ixx) makelist(power_mod(uz28[i],12,28),i,1,length(uz28));
(%oxx) [1,1,1,1,1,1,1,1,1,1,1,1]
```

Es decir, hemos elevado cada elemento de $U(\mathbb{Z}_{28})$ a 12, y en todos los casos nos ha salido 1.

Ahora vamos a calcular la décimosegunda potencia de todos los elementos de $\mathbb{Z}_{28}$ (no sólo de las unidades).

```
(%ixx) makelist(power_mod(i,12,28),i,0,27);
(%oxx) [0,1,8,1,8,1,8,21,8,1,8,1,8,1,0,1,8,1,8,1,8,21,8,1,8,1,8,1]
```

Y vemos que sale 1 únicamente en los lugares que se corresponden con las unidades.

Hacemos lo mismo con otro número, por ejemplo el 31, el cual es primo. Recuerde que si p es primo, entonces $\varphi(p)=p-1$. Ahora elevemos cada elemento a 30.

```
(%ixx) makelist(power_mod(i,30,31),i,0,30);
(%oxx) [0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
```

Vemos por tanto que todos los elementos, salvo el cero, son unidades en $\mathbb{Z}_{31}$. Este resultado concuerda con la teoría, según la cual, cuando p es primo, el anillo $\mathbb{Z}_p$ es cuerpo.

3.3. Sistemas de congruencias.

Vamos a ver aquí cómo resolver sistemas de ecuaciones en congruencia lineales con Maxima. Nos planteamos, en primer lugar, cómo resolver una congruencia de la forma

$$ax \equiv b (\text{mód } m)$$

Sabemos que esta congruencia tiene solución si, y sólo si, $mcd(a,m)$ es un divisor de b. En caso afirmativo, si d es el máximo común divisor de a y m, la congruencia anterior es equivalente a

$$a'x \equiv b' (\text{mód } m') \quad \text{donde } a'=\frac{a}{d},\ b'=\frac{b}{d} \ \text{ y } m'=\frac{m}{d}.$$

Y ahora, como $mcd(a',m')=1$ podemos calcular un inverso de a' módulo m'. Si llamamos a este inverso u, la congruencia es equivalente a

$$x \equiv u \cdot b' (\text{mód } m')$$

cuyas soluciones son todas de la forma $x=ub'+k\cdot m'$, con $k \in \mathbb{Z}$. Note que éste es otro método alternativo al que hemos visto en teoría.

Éstas ideas se plasman en la siguiente función, la cual, ó bien nos dice que la congruencia propuesta $ax \equiv b \mod m$ no tiene solución, ó bien nos devuelve una lista $[x_0, m']$, lo cual significa que todas las soluciones son de la forma $x=x_0+m'\cdot k$ con $k \in \mathbb{Z}$:

```
(%ixx) cong(a,b,m):=block(d:gcd(a,m),
                      if mod(b,d)=0
                         then [mod(b/d*inv_mod(a/d,m/d),m/d), m/d]
                         else "No tiene solución"
                       )$
```

Ahora podemos decirle que nos resuelva la ecuación en congruencia $963x \equiv 291 (\text{mód } 1578)$.

```
(%ixx) cong(963,291,1578);
(%oxx) [505,526]
```

Por tanto todas las soluciones son de la forma $x = 505 + 526 \cdot k$ con $k \in \mathbb{Z}$.

Si cambiamos el coeficiente 291 por 290, ya no hay solución:

```
(%ixx) cong(963,290,1578);
(%oxx) No tiene solución
```

Ahora lo que vamos a hacer es resolver un sistema de congruencias. Por ejemplo, tomamos el sistema

$$\begin{cases} 675x & \equiv & 485 & \text{mód } 1085 \\ 1211x & \equiv & 2156 & \text{mód } 2247 \end{cases}$$

Para resolverlo, seguimos los siguientes pasos:

1. Resolvemos la primera congruencia.

```
(%ixx) cong(675,485,1085);
(%oxx) [192,217]
```

Es decir, la solución es $x = 192 + 217 \cdot k$.

2. Sustituimos en la segunda congruencia, y tenemos $1211(192 + 217k) \equiv 2156(\text{mód } 2247)$, lo que nos da una congruencia en la que la incógnita es k.

3. Agrupamos los términos:

```
(%ixx) 1211*217; 2156-1211*192;
(%oxx) 262787
(%oxx) -230356
```

4. Luego tenemos que resolver la congruencia $262787k \equiv -230356(\text{mód } 2247)$.

```
(%ixx) cong(262787,-230356,2247);
(%oxx) [211,321]
```

Por tanto, la solución es $k = 211 + 321 \cdot k'$.

5. Sustituimos en la solución de la primera congruencia $x = 192 + 217 \cdot (211 + 321 \cdot k')$.

```
(%ixx) [192+217*211,217*321];
(%oxx) [45979,69657]
```

Y la solución del sistema es $x = 45979 + 69657 \cdot k'$.

Lo que vamos a hacer ahora es definir una función `Cong`. Esta función va a tener 4 argumentos: tres números y una lista de longitud 2. Supongamos que introducimos `Cong(a,b,m,li)`, donde `li` es la lista `[li1,li2]`. Entonces, nos debe devolver las soluciones de la congruencia $ax \equiv b(\text{mód } m)$, en las que x sea de la forma $x = li1 + li2 \cdot k$.

Así, por ejemplo, si introducimos `Cong(1211,2156,2247,[192,217])` nos devolvería `[45979,69657]`. Esta misma respuesta debe dar si introducimos `Cong(1211,2156,2247,cong(675,485,1085))`.

De esta forma, vemos cómo podemos resolver sistemas de dos o más congruencias. Caso de que el sistema no tenga solución, nos devolverá un mensaje de error.

```
(%ixx) Cong(a,b,m,l):=block(sol:cong(a*l[2],b-a*l[1],m),[l[1]+l[2]*sol[1],l[2]*sol[2]])$
```

Por ejemplo, si quisiéramos resolver el sistema

$$\begin{cases} x & \equiv & 30 & \text{mód } 100 \\ x & \equiv & 25 & \text{mód } 99 \\ x & \equiv & 13 & \text{mód } 97 \end{cases}$$

podemos escribir

```
(%ixx) Cong(1,30,100,Cong(1,25,99,cong(1,13,97)));
```

y lo que nos debe devolver es

```
(%oxx) [316330,960300]
```

lo que significa que la solución del sistema es $x = 316330 + 960300 \cdot k$.

Caso de que el sistema no tenga solución, nos dará un mensaje de error

```
(%ixx) Cong(1,2,6,cong(1,6,9));
MARRAY-TYPE-UNKNOWN: array type "No tiene solución" not recognized.
#0: Cong(a=1,b=2,m=6,l=[6,9])
 -- an error. To debug this try: debugmode(true);
```

Para evitar esto, podemos decir en la función `cong` que cuando no tenga solución nos devuelva `[0,0]`, es decir,

```
(%ixx) cong(a,b,m):=block(d:gcd(a,m),
                     if mod(b,d)=0
                        then [mod(b/d*inv_mod(a/d,m/d),m/d), m/d]
                        else [0,0]
                      )$
```

Y ahora, al ejecutar la instrucción anterior

```
(%ixx) Cong(1,2,6,cong(1,6,9));
```

nos devuelve `[6,0]`. El hecho de que nos haya dado 0 en la segunda parte significa que no tiene solución.

Ejercicio propuesto 1. Utilice algunos comandos ó funciones de las estudiadas en esta práctica, ó en prácticas anteriores, para resolver los ejercicios siguientes de la Relación de ejercicios del Tema 3: 4, 5, 6, 9, 14, 15, 20, 21, 22, 29, 30, 31, 32, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 46, 48.

Ejercicio propuesto 2. Resuelva la ecuación diofántica $31028063741 \cdot x + 85226124409 \cdot y = 1$.

Ejercicio propuesto 3. Obtenga una solución particular de la ecuación diofántica

$$7560398597 \cdot x + 51605476123 \cdot y + 98548325591 \cdot z = 2014.$$

Ejercicio propuesto 4. Consideramos el número 59. Mediante comandos apropiados de Maxima, compruebe que 59 es primo y calcule el producto (reducido módulo 59) de todos los elementos no nulos del anillo $\mathbb{Z}_{59}$. A continuación elija otro número primo, y para éste repita el mismo proceso. ¿Qué se observa? El resultado general de éstos cálculos se denomina Teorema de Wilson.

Ejercicio propuesto 5. Mediante comandos apropiados de Maxima, defina en primer lugar un conjunto $UZ56$ formado por las unidades del anillo $\mathbb{Z}_{56}$. A continuación calcule el siguiente subconjunto de $\mathbb{Z}_{56}$:

$$P = \Big\{ a \cdot b \mid a, b \in UZ56 \Big\}.$$

¿Qué se observa tras estos cálculos? ¿Sabría justificar por qué ocurre eso?

Ejercicio propuesto 6: Resuelva el siguiente sistema de ecuaciones en congruencia:

$$\begin{cases} 5679523045x & \equiv & 238165236 & \text{mód } 291342571 \\ 57892345987x & \equiv & 231293175 & \text{mód } 714897237 \\ 74202349823x & \equiv & 2813407650 & \text{mód } 57492734832 \\ 758298734023x & \equiv & 47366104086 & \text{mód } 65929875234 \end{cases}$$