UVG-MM2015:
Matemática Discreta
Instructor: Héctor Villafuerte

Proyecto de Grupos y Códigos
Entrega: 1 de Diciembre, 2009

---

1 Álgebra

La idea básica de un *grupo* es tomar un *conjunto* G y dotarlo de una *operación* $\odot$. La notación usual es $(G, \odot)$. Esta operación debe cumplir con ciertas propiedades:

- Asociatividad: $(a \odot b) \odot c = a \odot (b \odot c)$ para $a, b, c \in G$
- Elemento neutro: existe $e \in G$, tal que $a \odot e = e \odot a = a$ para todo elemento $a \in G$.
- Inversos: para cada $a \in G$, existe $a^{-1} \in G$ tal que $a^{-1} \odot a = a \odot a^{-1} = e$.

Si además se tiene conmutatividad ($a \odot b = b \odot a$) decimos que el grupo es *abeliano*. El *orden* del grupo es la cardinalidad de G, $|G|$. Algunos ejemplos de grupos son:

- Los enteros con la suma; $(\mathbb{Z}, +)$
- Los racionales excepto el cero con la multiplicación; $(\mathbb{Q} \setminus \{0\}, \times)$
- Cero y uno con la suma módulo 2; $(\{0, 1\}, +_{\text{mod } 2})$

Noten que la operación $\odot$ es parte fundamental del concepto de grupo. Los grupos pueden describirse de varias maneras: como conjuntos de matrices, como conjuntos de símbolos sujetos a ciertas relaciones, como diagramas, como permutaciones, etc. Es usual describir grupos empleando *permutaciones* (ver §6.6 de Farmer). Considere por ejemplo la permutación $\sigma = (1\ 3)(2\ 5\ 4)$, que en SAGE puede denotarse sintácticamente de dos formas:

- Como "string": `"(1,3) (2,5,4)"`
- Como lista de tuplas: `[(1,3), (2,5,4)]`

Considere un subconjunto H de G, $H \subset G$. Puede darse el caso que si dotamos a H con la misma operación $\odot$ obtengamos un grupo. Decimos entonces que $(H, \odot)$ es un *subgrupo* de $(G, \odot)$. Sea G un grupo y H un subgrupo de G; considere $g \in G$, definimos entonces los siguientes conjuntos:

- *Coset* izquierdo, $gH = \{g \odot h \mid h \in H\}$
- *Coset* derecho, $Hg = \{h \odot g \mid h \in H\}$

Dado un subgrupo H del grupo G, puede demostrarse que H induce una partición sobre G (donde los bloques de la partición son los cosets).

SAGE permite definir fácilmente varios grupos comunes:

- S_n, `SymmetricGroup(n)`: todas las $n!$ permutaciones en n símbolos. El orden del grupo es $n!$.
- D_n, `DihedralGroup(n)`: simetrías de un polígono con n lados (i.e. "n-gon"); rotaciones y "flips". El orden del grupo es $2n$.
- C_n, `CyclicPermutationGroup(n)`: rotaciones de un polígono con n lados (i.e. "n-gon") (sin "flips"). El orden del grupo es n.

- A_n, `AlternatingGroup(n)`: grupo alternante en n símbolos. El orden del grupo es $n!/2$.

IMPORTANTE En SAGE recuerden utilizar "tab-completion" para explorar estructuras (como grupos).

Veamos ahora algunos ejemplos:

```
sage: G = CyclicPermutationGroup(3)
sage: G.order()
3
sage: G.list()
[(), (1,2,3), (1,3,2)]
sage: G.is_abelian()
True
sage: G.cayley_table()
[x0 x1 x2]
[x1 x2 x0]
[x2 x0 x1]
sage: G.random_element()
()
sage: G.random_element()
(1,2,3)
sage: G = DihedralGroup(4)
sage: G.normal_subgroups()
[Permutation Group with generators [()],
 Permutation Group with generators [(1,3)(2,4)],
 Permutation Group with generators [(1,2)(3,4), (1,3)(2,4)],
 Permutation Group with generators [(2,4), (1,3)(2,4)],
 Permutation Group with generators [(1,2,3,4), (1,3)(2,4)],
 Permutation Group with generators [(1,2,3,4), (1,4)(2,3)]]
sage: G.order()
8
sage: [H.order() for H in G.normal_subgroups()]
[1, 2, 4, 4, 4, 8]
sage: G1 = DihedralGroup(5)
sage: G2 = CyclicPermutationGroup(5)
sage: G1.is_subgroup(G2)
False
sage: G2.is_subgroup(G1)
True
```

Un *campo* ("field") es un conjunto F dotado con dos operaciones $(\mathsf{F}, \oplus, \odot)$ que cumplen con que:

- $(\mathsf{F}, \oplus)$ es un grupo abeliano.
- $(\mathsf{F} \setminus \{e\}, \odot)$ es un grupo abeliano; donde e es el elemento neutro de $(\mathsf{F}, \oplus)$.

Asegúrese de entender por qué tenemos campos en $(\mathbb{R}, +, \times)$, $(\mathbb{Q}, +, \times)$, $(\mathbb{C}, +, \times)$.

Un *anillo* ("ring") es un conjunto F dotado con dos operaciones $(\mathsf{F}, \oplus, \odot)$ que cumplen con que:

- $(\mathsf{F}, \oplus)$ es un grupo abeliano.

- $(\mathsf{F}, \odot)$ es un *monoide* o un *semi-grupo* (depende a quién le pregunten).

En otras palabras, un anillo es un campo con menos estructura (e.g. no hay "división"). Algunos anillos comunes, así como su sintaxis en SAGE son:

- $(\mathbb{Z}, +, \times)$, `ZZ`
- $(\mathbb{Q}, +, \times)$, `QQ`
- $(\mathbb{R}, +, \times)$, `RR`
- $(\mathbb{C}, +, \times)$, `CC`

A continuación encontrarán algo de funcionalidad en SAGE para trabajar con campos y anillos:

```
sage: poly.<x> = PolynomialRing(QQ)
sage: factor(x^2 - 2)    # no factorizable en QQ
x^2 - 2
sage: poly.<x> = PolynomialRing(RR)
sage: factor(x^2 - 2)
(1.00000000000000*x - 1.41421356237310) * (1.00000000000000*x + 1.41421356237310)
sage: sqrt(-1) in RR   # pertenencia de conjuntos
False
sage: sqrt(-1) in CC
True
sage: 1/2 in ZZ
False
sage: 1/2 in QQ
True
sage: QQ(1.2)    # cambio de representacion
6/5
```

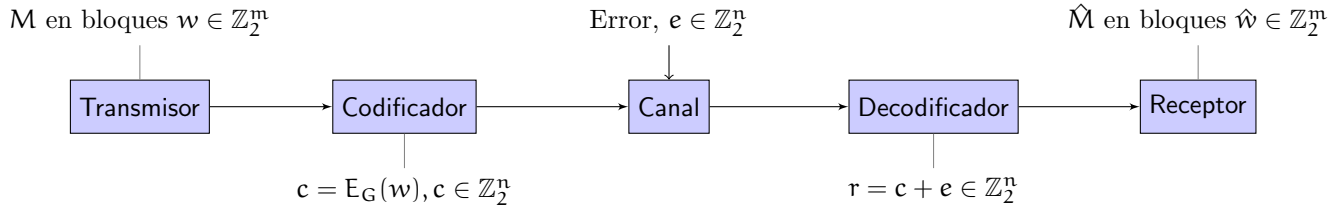
En particular nos interesará trabajar con matrices sobre ciertos anillos. Algunos ejemplos:

```
sage: random_matrix(QQ, 3)
[ 1  0  0]
[ 2  0  0]
[-1 -2 -1]
sage: random_matrix(QQ, 3)
[   2    0    1]
[   0   -2    0]
[   2 -1/2    0]
sage: random_matrix(Zmod(2), 3)
[0 1 1]
[0 1 1]
[0 0 0]
sage: random_matrix(Zmod(2), 3)
[1 1 1]
[0 0 1]
[1 1 0]
```

```
sage: M = Matrix([[1, 0.2], [3, -1]]); M
[ 1.0000000000  0.2000000000]
[ 3.0000000000 -1.0000000000]
sage: M = Matrix(QQ, [[1, 0.2], [3, -1]]); M
[  1 1/5]
[  3  -1]
```

2 Códigos

A finales de la década de 1940 Richard Hamming reconoció la importancia de la detección y corrección de errores en el uso de las computadoras. En ese entonces se utilizaba el chequeo de paridad, que solamente detectaba errores (era incapaz de corregirlos). Los códigos de Hamming permiten corregir errores de un bit y aún son ampliamente utilizados en telecomunicaciones, compresión de datos, entre otros. El siguiente diagrama ayudará a ilustrar las ideas que presentaremos en esta sección:



La *teoría de códigos* estudia la eficiente *detección* y *corrección* de errores en una señal. Utilizaremos algunos de los conceptos presentados en la sección anterior. Sea Σ un conjunto finito de símbolos, llamado el *alfabeto*. En particular, utilizaremos $\Sigma = \{0, 1\}$. Sea Σ^* el conjunto infinito de posibles mensajes empleando el alfabeto Σ. Sea $M \in \Sigma^*$ el mensaje enviado y $\hat{M}$ el mensaje recibido. Idealmente queremos que $M = \hat{M}$.

Considere el conjunto $\mathbb{Z}_2 = \{0, 1\}$ dotado con las operaciones suma módulo 2 y multiplicación módulo 2, en breve: $(\mathbb{Z}_2, \oplus, \otimes)$. Sea G una matriz $m \times n$ sobre $\mathbb{Z}_2$. Consideraremos matrices G de la forma $[I|A]$ donde I es $m \times m$ y A es $m \times (n - m)$. Definimos entonces una *función de codificación* $E_G(w) : \mathbb{Z}_2^m \to \mathbb{Z}_2^n$ como:

$$E_G(w) = wG, \qquad w \in \mathbb{Z}_2^m$$

Llamaremos a G la *matriz generadora* del código C, donde C es un subconjunto del espacio vectorial $\mathbb{Z}_2^n$. Si C es además un sub-espacio de $\mathbb{Z}_2^n$ decimos que C es un *código lineal*.

En SAGE representamos a $\mathbb{Z}_2$ con las operaciones usuales con el comando `FiniteField(2)` (que es equivalente al comando más corto `GF(2)`).

```
sage: FiniteField(2)
Finite Field of size 2
sage: GF(2)
Finite Field of size 2
sage: FiniteField(2) == GF(2)
True
```

Por ejemplo, considere la matriz G y su construcción en SAGE :

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

```
sage: I = identity_matrix(GF(2), 3); I
[1 0 0]
[0 1 0]
[0 0 1]
sage: A = Matrix(GF(2), [[1,1,0], [0,1,1], [1,0,1]]); A
[1 1 0]
[0 1 1]
[1 0 1]
sage: G = I.augment(A); G
[1 0 0 1 1 0]
[0 1 0 0 1 1]
[0 0 1 1 0 1]
```

Si tomamos ahora vectores $w \in \mathbb{Z}_2^3$, podemos empezar a experimentar con las palabras codificadas por $E_G(w) = wG$. Por supuesto, SAGE también maneja *espacios vectoriales* ($\mathbb{Z}_2^3$ en SAGE es `VectorSpace(GF(2), 3)`):

```
sage: Vm = VectorSpace(GF(2), 3)
sage: Vm.list()
[(0, 0, 0),
 (1, 0, 0),
 (0, 1, 0),
 (1, 1, 0),
 (0, 0, 1),
 (1, 0, 1),
 (0, 1, 1),
 (1, 1, 1)]
sage: for w in Vm:
....:     print w, '  ---->  ', w*G
....:
(0, 0, 0)   ---->   (0, 0, 0, 0, 0, 0)
(1, 0, 0)   ---->   (1, 0, 0, 1, 1, 0)
(0, 1, 0)   ---->   (0, 1, 0, 0, 1, 1)
(1, 1, 0)   ---->   (1, 1, 0, 1, 0, 1)
(0, 0, 1)   ---->   (0, 0, 1, 1, 0, 1)
(1, 0, 1)   ---->   (1, 0, 1, 0, 1, 1)
(0, 1, 1)   ---->   (0, 1, 1, 1, 1, 0)
(1, 1, 1)   ---->   (1, 1, 1, 0, 0, 0)
```

Un caso famoso es el *código de chequeo de paridad*, $E(w) = wp_w$ donde $p_w = 1$ si w tiene un número impar de unos y $p_w = 0$ si w tiene un número par de unos. Noten que la siguiente matriz G es una válida matriz generadora para dicho código:

```
sage: I = identity_matrix(GF(2), 3); I
[1 0 0]
[0 1 0]
[0 0 1]
sage: A = Matrix(GF(2), [[1], [1], [1]]); A
[1]
[1]
[1]
sage: G = I.augment(A); G
[1 0 0 1]
[0 1 0 1]
[0 0 1 1]
sage: for w in Vm:
....:     print w, '  --->  ', w*G
....:
(0, 0, 0)   --->   (0, 0, 0, 0)
(1, 0, 0)   --->   (1, 0, 0, 1)
(0, 1, 0)   --->   (0, 1, 0, 1)
(1, 1, 0)   --->   (1, 1, 0, 0)
(0, 0, 1)   --->   (0, 0, 1, 1)
(1, 0, 1)   --->   (1, 0, 1, 0)
(0, 1, 1)   --->   (0, 1, 1, 0)
(1, 1, 1)   --->   (1, 1, 1, 1)
```

Noten además que un código lineal obedece que para todo $c_1, c_2 \in C$ se cumple que $c_1 + c_2 \in C$. Adicionalmente, las siguientes condiciones son equivalentes:

- C es un código lineal.
- C es generado por una matriz generadora G.
- $E_G(w_1 + w_2) = E_G(w_1) + E_G(w_2)$

Para una matriz generadora $G = [I|A]$, definimos la *matriz de chequeo de paridad* $H = [A^T|I]$. Noten que H es una matriz $(n - m) \times m$ sobre $\mathbb{Z}_2$. Por ejemplo:

```
sage: I = identity_matrix(GF(2), 3)
sage: A = Matrix(GF(2), [[1,1,0], [0,1,1], [1,0,1]])
sage: G = I.augment(A); G
[1 0 0 1 1 0]
[0 1 0 0 1 1]
[0 0 1 1 0 1]
sage: H = A.transpose().augment(I); H
[1 0 1 1 0 0]
[1 1 0 0 1 0]
[0 1 1 0 0 1]
```

Sea $c = wG$ donde $c \in C$. Observe lo que sucede cuando tomamos Hc^T:

```
sage: Vm = VectorSpace(GF(2), 3)
sage: for w in Vm:
....:     print w, '  --->  ', w*G, '  --->  ', (H*(w*G).transpose()).transpose()
....:
(0, 0, 0)   --->   (0, 0, 0, 0, 0, 0)   --->   [0 0 0]
(1, 0, 0)   --->   (1, 0, 0, 1, 1, 0)   --->   [0 0 0]
(0, 1, 0)   --->   (0, 1, 0, 0, 1, 1)   --->   [0 0 0]
(1, 1, 0)   --->   (1, 1, 0, 1, 0, 1)   --->   [0 0 0]
(0, 0, 1)   --->   (0, 0, 1, 1, 0, 1)   --->   [0 0 0]
(1, 0, 1)   --->   (1, 0, 1, 0, 1, 1)   --->   [0 0 0]
(0, 1, 1)   --->   (0, 1, 1, 1, 1, 0)   --->   [0 0 0]
(1, 1, 1)   --->   (1, 1, 1, 0, 0, 0)   --->   [0 0 0]
```

Podemos conjeturar entonces que $\mathsf{Hc}^\mathsf{T} = \mathsf{0}$ para todo $\mathsf{c} \in \mathsf{C}$. Con un poco de álgebra obtenemos que $\mathsf{Hc}^\mathsf{T} = \mathsf{H}(\mathsf{wG})^\mathsf{T} = \mathsf{HG}^\mathsf{T}\mathsf{w}^\mathsf{T} = \mathsf{0}$. En breve, se puede demostrar que el *kernel* de H cumple con que $\ker(\mathsf{H}) = \mathsf{C}$. En otras palabras, $\mathsf{c} \in \mathsf{C}$ si y sólo si $\mathsf{Hc}^\mathsf{T} = \mathsf{0}$. En base a lo anterior: suponga que se envía $\mathsf{c} \in \mathsf{C}$ y se recibe $\mathsf{r} = \mathsf{c} + \mathsf{e}$ donde e es el *error en la transmisión*; tenemos entonces que,

$$\begin{aligned} \mathsf{r} &= \mathsf{c} + \mathsf{e} \\ \mathsf{r}^\mathsf{T} &= (\mathsf{c} + \mathsf{e})^\mathsf{T} = \mathsf{c}^\mathsf{T} + \mathsf{e}^\mathsf{T} \\ \mathsf{Hr}^\mathsf{T} &= \mathsf{H}(\mathsf{c}^\mathsf{T} + \mathsf{e}^\mathsf{T}) = \mathsf{Hc}^\mathsf{T} + \mathsf{He}^\mathsf{T} \\ &= \mathsf{0} + \mathsf{He}^\mathsf{T} \\ \mathsf{Hr}^\mathsf{T} &= \mathsf{He}^\mathsf{T} \end{aligned}$$

Para un mensaje recibido $\mathsf{r} \in \mathbb{Z}_2^\mathsf{n}$, denominamos a $\mathsf{Hr}^\mathsf{T} \in \mathbb{Z}_2^\mathsf{m}$ el *síndrome* de r. Puede demostrarse que el conjunto de todos los mensajes recibidos r con el mismo síndrome es un *coset* del código C. Ahora bien, dada una matriz generadora G podemos definir un código lineal C en SAGE empleando el comando `LinearCode(G)` (nuevamente noten que $\mathsf{C} \in \mathbb{Z}_2^6$):

```
sage: G
[1 0 0 1 1 0]
[0 1 0 0 1 1]
[0 0 1 1 0 1]
sage: C = LinearCode(G); C
Linear code of length 6, dimension 3 over Finite Field of size 2
sage: C.list()
[(0, 0, 0, 0, 0, 0),
 (1, 0, 0, 1, 1, 0),
 (0, 1, 0, 0, 1, 1),
 (1, 1, 0, 1, 0, 1),
 (0, 0, 1, 1, 0, 1),
 (1, 0, 1, 0, 1, 1),
 (0, 1, 1, 1, 1, 0),
 (1, 1, 1, 0, 0, 0)]
sage: C.gen_mat() == G
True
```

```
sage: C.check_mat() == H
False
sage: C.check_mat() == H.echelon_form()
True
```

Similarmente podemos trabajar en SAGE con códigos de Hamming (que son un caso especial de códigos lineales). Por ejemplo, Hamming(7,4) codifica 4 bits en 7 bits. Noten que dado el código C, podemos encontrar las matrices generadora G y de chequeo de paridad H. Vale la pena mencionar la función `A.solve_left(b)` que resuelve la ecuación matricial $xA = b$.

```
sage: C = HammingCode(3, GF(2)); C
Linear code of length 7, dimension 4 over Finite Field of size 2
sage: G = C.gen_mat(); G
[1 0 0 1 0 1 0]
[0 1 0 1 0 1 1]
[0 0 1 1 0 0 1]
[0 0 0 0 1 1 1]
sage: H = C.gen_mat(); H
[1 0 0 1 0 1 0]
[0 1 0 1 0 1 1]
[0 0 1 1 0 0 1]
[0 0 0 0 1 1 1]
sage: w = vector([1,0,0,1])  # vector de prueba
sage: c = w*G; c
(1, 0, 0, 1, 1, 0, 1)
sage: G.solve_left(c)    # == w
[1 0 0 1]
```

Las siguientes funciones nos permitirán codificar y decodificar e introducir ruido en el canal.

```
from sage.all import *

def str2vec(M, m):
    Vm = VectorSpace(GF(2), m)      # Espacio vectorial Vm.
    L = [c for c in M]              #
    L = map(ord, L)                 # Representa
    L = map(Integer, L)             # el mensaje M (ASCII)
    L = [k.str(base=2) for k in L] # como una
    L = [k.zfill(7) for k in L]     # cadena
    L = reduce(lambda a,b: a+b, L) # de bits.
    r = len(L) % m                  #
    L = L + (m - r)*"0"             #
    L = [L[ m*k : m*(k+1) ] \
            for k in range(len(L) / m)]  # Representa la cadena
    L = map(list,L)                      # de bits como una lista
    L = map(Vm,L)                        # de vectores en Vm.
    return L

def vec2str(M):
    L = [[str(k) for k in c] for c in M]         # Dada una lista
    L = [reduce(lambda a,b: a+b, c) for c in L] # de vectores en Vm
    L = reduce(lambda a,b: a+b, L)               # encuentra su
    r = len(L) % 7                               # representacion
    if r <> 0:                                   # como cadena ASCII
        L = L[:-r]
    L = [L[ 7*k : 7*(k+1) ] \
            for k in range(len(L) / 7)]
    L = [Integer(c,2) for c in L]
    L = [chr(c) for c in L]
    L = reduce(lambda a,b: a+b, L)
    return L

def error_canal(n, s):
    Vn = VectorSpace(GF(2), n)   #
    e = Vn.zero_vector()         # Introduce s-bits
    for j in range(s):           # de error aleatorio
        k = randint(0, n - 1)    # en el canal
        e[k] = e[k] + 1          #
    return e
```

Los siguientes son algunos escenarios posibles en el marco de la transmisión del mensaje M:

```
########################################
# Caso 0:
# - Sin bloques Codificador/Decodificador
# - Sin ruido (error) en el canal
M = 'Hola mundo'
Mv = str2vec(M, 4)
Mhat = vec2str(Mv)
print M, '  ---> ', Mhat
print 'M == Mhat:', M == Mhat

########################################
# Caso 1:
# - Sin bloques Codificador/Decodificador
# - Con ruido (1-bit error) en el canal
M = 'Hola mundo'
Mv = str2vec(M, 4)
Mv = [c + error_canal(4,1) for c in Mv]
Mhat = vec2str(Mv)
print M, '  ---> ', Mhat
print 'M == Mhat:', M == Mhat

########################################
# Caso 2:
# - Con codificacion lineal
# - Con ruido (1-bit error) en el canal
C = HammingCode(3, GF(2))
G = C.gen_mat()
H = C.check_mat()

M = 'Hola mundo'
Mv = str2vec(M, 4)
Mv = [c*G for c in Mv]
Mv = [c + error_canal(7,1) for c in Mv]
Mv = [C.decode(c) for c in Mv]
Mv = [G.solve_left(c).list() for c in Mv]
Mhat = vec2str(Mv)
print M, '  ---> ', Mhat
print 'M == Mhat:', M == Mhat
```

3 Preguntas

1. **Grupos**.
   (a) Aleatoriamente seleccione un tipo de grupo finito: S_n, D_n, C_n, A_n. Puede usar dados, monedas, el minuto actual módulo su número de la suerte, SAGE , etc.
   (b) Aleatoriamente seleccione un entero n que determinará el orden del grupo del inciso anterior. Denotemos a este grupo por G.
   (c) Utilice `G.normal_subgroups()` para obtener un listado de subgrupos H de G.
   (d) ¿Cuál es el orden de G?
   (e) ¿Cuál es el orden de cada uno de los subgrupos H de G?
   (f) ¿Observa algún patrón? Conjeture.
   (g) Repita los incisos anteriores algunas veces para ganar confianza en su conjetura.
2. **Códigos**.
   (a) Elija un mensaje M breve, e.g. "Eterna primavera".
   (b) Ayudándose del código fuente de las secciones anteriores, codifíquelo empleando Hamming(7,4) sobre un canal que introduce 1-bit de error. ¿Puede recuperar el mensaje M?
   (c) ¿Qué sucede si el canal introduce 2-bits de error?