Sage Logroño Álgebra

Enseñando álgebra de grado con Sage

Dependiendo de la asignatura, de la titulación, del tiempo disponible y de otros factores, nos puede interesar trabajar el álgebra con el ordenador de varias formas distintas, con mayor o menor detalle y con mayor o menor abstracción. Sage nos permite bastante flexibilidad, pero es importante conocer todas las opciones por si los alumnos usan otra forma de trabajar y obtienen resultados distintos.

Polinomios y matrices *out of the box*

Veamos primero qué podemos esperar si usamos polinomios y matrices en Sage de la manera más naive, sin preocuparnos por definir el anillo en que trabajamos.

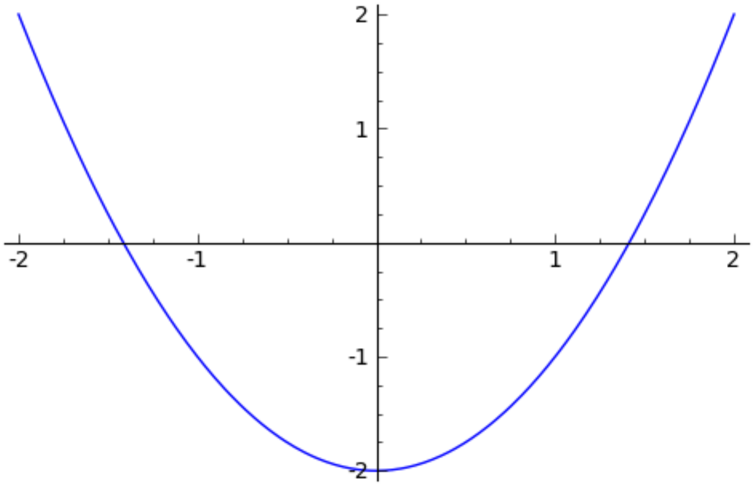
```
#Polinomios usando la variable 'x' por defecto
p1 = x^2 - 2
p2 = x^3 - 2*x + 1/2
p3 = x^5 - 2*x^2 + x + 1
```

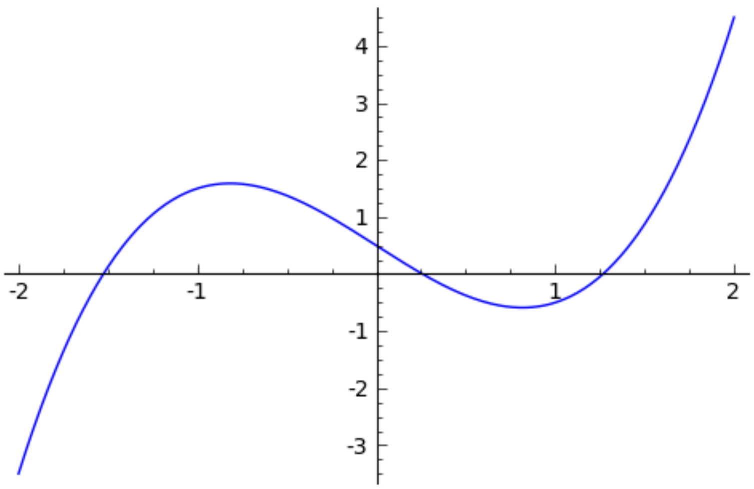
```
p1
```

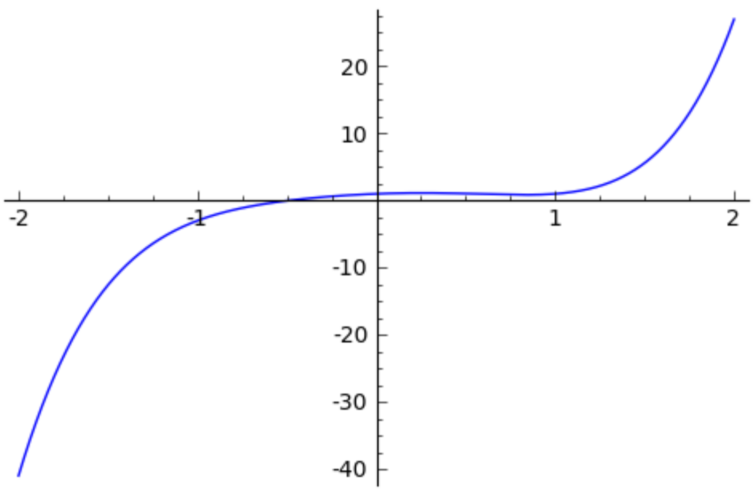
```
x^2 - 2
```

```
type(p1)
```

```
<type 'sage.symbolic.expression.Expression'>
```








```
#Las operacions habituales funcionan sin sorpresas
p1(x=1), p1+2*p2, p1/p3
```

```
(-1, 2*x^3 + x^2 - 4*x - 1, (x^2 - 2)/(x^5 - 2*x^2 + x + 1))
```

```
type(p1/p3)
```

```
<type 'sage.symbolic.expression.Expression'>
```

```
#El metodo roots devuelve las raices exactas, si puede
print p1.roots()
show(p2.roots())
show(p3.roots())
```

```
[(-sqrt(2), 1), (sqrt(2), 1)]
```

$$\left[\left(-\frac{1}{2}\left(i\sqrt{3}+1\right)\left(\frac{1}{36}i\sqrt{3}\sqrt{101}-\frac{1}{4}\right)^{\left(\frac{1}{3}\right)}+\frac{i\sqrt{3}-1}{3\left(\frac{1}{36}i\sqrt{3}\sqrt{101}-\frac{1}{4}\right)^{\left(\frac{1}{3}\right)}},1\right),\left(-\frac{1}{2}\left(-i\sqrt{3}+1\right)\left(\frac{1}{36}i\cdots\right.\right.\right.$$

```
Traceback (click to the left of this block for traceback)
...
RuntimeError: no explicit roots found
```

```
plot(p1,x,-2,2).show()
plot(p2,x,-2,2).show()
plot(p3,x,-2,2).show()
```

```
#Escribimos las raices de p2 en desarrollo decimal
[n(raiz) for raiz, multiplicidad in p2.roots()]
```

```
[0.258652022504153 - 2.22044604925031e-16*I, -1.52568712086552,
1.26703509836137 + 2.22044604925031e-16*I]
```

```
#Buscamos numericamente la raiz real de p3
p3.find_root(-1,1)
```

```
-0.4904777202073996
```

```
#Matriz como lista de filas
M1 = matrix([[0,1],[1,0]])
#Matriz con una sola lista para todos los coeficientes
#ahora es necesario indicar el tamanyo de la matriz
M2 = matrix(2,2,[0,1,-1,0])

#Una forma compacta de introducir una matriz dada por una formula
N = 3
M3 = matrix([[1/(k+j+1) for j in range(N)] for k in range(N)])
```

```
show(M1)
show(M2)
show(M3)
```

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

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{pmatrix}$$

```
#Una forma alternativa de definir M3
#Desgraciadamente, nos tenemos que adelantar, y declarar
#que los elementos son racionales
M3 = matrix(QQ,3,3)
N=3
for k in range(N):
    for j in range(N):
        M3[j,k]=1/(k+j+1)
print M3
```

```
[  1 1/2 1/3]
[1/2 1/3 1/4]
[1/3 1/4 1/5]
```

```
#Las operaciones habituales funcionan sin sorpresas
print M1*M2
print ~M1  #matriz inversa
print M1 + 2*M2
print M1*M3 #multiplicamos matrices de tamanyos incompatibles
```

```
[-1  0]
[ 0  1]
[0 1]
[1 0]
[ 0  3]
[-1  0]
Traceback (click to the left of this block for traceback)
...
TypeError: unsupported operand parent(s) for '*': 'Full MatrixSpace
of 2 by 2 dense matrices over Integer Ring' and 'Full MatrixSpace of
3 by 3 dense matrices over Rational Field'
```

```
#vectores, que son filas o columnas segun convenga
#(no es lo mismo que una matriz 1xn ni nx1)
M4 = matrix(2,3,[0,1,-1,0,2,2])
v1 = vector([1,1,1])
v2 = vector([1/2,1])
print M4
print M4*v1
print v2*M4
#el orden es importante
print v1*M4
```

```
[ 0  1 -1]
[ 0  2  2]
(0, 4)
(0, 5/2, 3/2)
Traceback (click to the left of this block for traceback)
...
TypeError: unsupported operand parent(s) for '*': 'Ambient free
module of rank 3 over the principal ideal domain Integer Ring' and
'Full MatrixSpace of 2 by 3 dense matrices over Integer Ring'
```

```
print M4\v2, M4.solve_right(v2) #son sinonimos
print M4.solve_right(v1)
```

```
(0, 1/2, 0) (0, 1/2, 0)
Traceback (click to the left of this block for traceback)
...
ValueError: number of rows of self must equal degree of B
```

```
#Podemos conseguir los autovalores de las matrices
print M1.eigenvalues()
print M2.eigenvalues()
print M3.eigenvalues()
```

```
[1, -1]
[-1*I, 1*I]
[0.002687340355773529?, 0.1223270658539O584?, 1.408318927123654?]
```

```
#Aunque el resultado es distinto si usamos la definicion
pc1 = det(M1-x)
print pc1.roots()

pc2 = det(M2-x)
print pc2.roots()

pc3 = det(M3-x)
show( pc3.roots())
```

```
[(-1, 1), (1, 1)]
[(-I, 1), (I, 1)]
```

$$\left[\left(-\frac{1}{2}\left(i\sqrt{3}+1\right)\left(\frac{1}{14400}i\sqrt{3}\sqrt{29933}+\frac{129287}{1458000}\right)^{\left(\frac{1}{3}\right)}-\frac{-6559i\sqrt{3}+6559}{64800\left(\frac{1}{14400}i\sqrt{3}\sqrt{29933}+\frac{129287}{1458000}\right)^{\left(\frac{1}{3}\right)}}+\right.\right.$$

```
#La forma de Jordan puede funcionar
M1.jordan_form()
```

```
[ 1| 0]
[--+--]
[ 0|-1]
```

```
#pero en general falla
M2.jordan_form()
```

```
Traceback (click to the left of this block for traceback)
...
RuntimeError: Some eigenvalue does not exist in Integer Ring.
```

```
M3.jordan_form()
```

```
Traceback (click to the left of this block for traceback)
...
RuntimeError: Some eigenvalue does not exist in Rational Field.
```

Trabajar módulo m

Para las clases de aritmética, necesitamos trabajar en $\mathbb{Z}_m$. Es posible trabajar usando números enteros normales de Sage, pero tomando restos módulo m.

Suma y producto módulo *m*

Para hacer sumas y productos sobre clases de equivalencia, podemos usar la suma y el producto habituales de números enteros, y tomar el resto de dividir por m:

```
m=31
a=12
b=23
s=(a+b)%m
p=(a*b)%m
```

Potencia módulo *m*

Aunque podemos calcular la clase de congruencia de $a^p(mod\ m)$ calculando el entero a^p y luego tomando el resto módulo m, debemos tener en cuenta que a^p puede ser un número muy grande, y el ordenador puede dedicar al cálculo demasiado tiempo y memoria. Para esta tarea, podemos usar la función power_mod:

```
power_mod(3,8,10) == (3^8)%10
```

```
True
```

```
timeit('power_mod(3,1000000,7)')
```

```
625 loops, best of 3: 895 µs per loop
```

```
timeit('(3^1000000)%7')
```

```
5 loops, best of 3: 93.6 ms per loop
```

El inverso de a módulo m, es decir, la clase b tal que $a \cdot b \equiv 1(mod\ m)$, no se puede reducir a una operación de aritmética usual. Una llamada a la función inverse_mod(a,m) devuelve el inverso de a módulo m (este número se calcula usando los coeficientes de una identidad de Bézout). La operación equivalente en `Integers(m)` es `1/a`.

```
a=7
m=11
(a*inverse_mod(a,m))%m
```

```
1
```

Ventajas de esta forma de trabajar

- Está tan a mano que los alumnos la van a encontrar aunque no se la cuentes.
- Minimizas el tiempo dedicado a Sage, luego más tiempo para las matemáticas.
- Se parece a la forma en que usamos otros programas similares.

Grupos y Anillos

Sage tiene definidos un buen número de anillos, grupos y otras estructuras algebraicas.

Podemos operar con elementos que representan elementos de un anillo o un espacio vectorial, por ejemplo, además de con objetos que representan anillos, grupos, subgrupos, subespacios vectoriales y otras estructuras de nivel más alto.

Muchos anillos comunes están definidos en Sage:

- `ZZ`: $\mathbb{Z}$

- `Integers(m)`: $\mathbb{Z}_m$ (ó $\mathbb{Z}/m\mathbb{Z}$)
- `QQ`: $\mathbb{Q}$
- `QQbar`: $\bar{\mathbb{Q}}$ (clausura akgebraica de Q)
- `RR`: $\mathbb{R}$, representados por números reales de doble precisión
- `RealField(bits)`:, números reales con una cantidad arbitraria de bits de precisión
- `RDF` (o `RealDoubleField()`), números reales de 64 bits.
- `CDF` (o `ComplexDoubleField()`), números complejos de 64 bits.
- `SR`, o expresiones simbólicas. Cualquier expresión algebraica que contenga símbolos como pi, I, sqrt(2) pertenece a este anillo.

Además, podemos usar otros constructores para definir anillos derivados, como el constructor de anillos de polinomios `PolynomialRing` que veremos en detalle más abajo.

```
R1 = Integers(7)
R2 = Integers(21)
```

```
a = ZZ(3)
b = R1(3)
c = R2(3)
print a, a.parent()
print b, b.parent()
print c, c.parent()
print c, (b*c).parent()
```

```
3 Integer Ring
3 Ring of integers modulo 7
3 Ring of integers modulo 21
3 Ring of integers modulo 7
```

```
#Al calcular el inverso, Sage puede devolver el inverso
#en el cuerpo de fracciones del anillo(en QQ en vez de ZZ)
print a, 1/a, (1/a).parent()

#Si el anillo es un cuerpo, obtenemos un elemento del mismo anillo
print b, 1/b, (1/b).parent()

#Si el anillo no es dominio de integridad, algunos elementos
#no tienen inversos (en ningún cuerpo que contenga al anillo)
print c, 1/c, (1/c).parent()
```

```
3 1/3 Rational Field
3 5 Ring of integers modulo 7
3
Traceback (click to the left of this block for traceback)
...
ZeroDivisionError: Inverse does not exist.
```

Anillos de Polinomios

La sintaxis para definir un anillo de polinomios con coeficientes en otro anillo **R** es:

```
Pols.<t> = PolynomialRing(R, 't')
```

donde hemos definido el anillo de polinomios `Pols` con coeficientes en el anillo `R` y la variable independiente `t`.

```
#Definimos varios anillos de polinomios con coeficientes en distintos anillos
PR1.<t> = PolynomialRing(ZZ)
PR2.<s> = PolynomialRing(QQ)
```

```
PR3.<y> = PolynomialRing(RR)
PR4.<z> = PolynomialRing(CC)
```

```
t.parent()
```

```
Univariate Polynomial Ring in t over Integer Ring
```

Una vez hemos definido el anillo, podemos usar la variable independiente para definir polinomios, operando con ella como una variable más.

```
p=t^3-2*t+1
print p.base_ring()
#Las raices tienen que ser elementos del anillo de coefs
print p.roots()
```

```
Integer Ring
[(1, 1)]
```

```
p=s^3-2*s+1
print p.base_ring()
print p.roots()
```

```
Rational Field
[(1, 1)]
```

```
p=y^3-2*y+1
print p.base_ring()
print p.roots()
```

```
Real Field with 53 bits of precision
[(-1.61803398874989, 1), (0.618033988749895, 1), (1.00000000000000,
1)]
```

El método `roots` devuelve sólo las raíces enteras, y `real_roots` devuelve todas las raíces reales, y además lo hace de forma numérica. Otra opción es extender el polinomio a un polinomio con coeficientes racionales, y entonces el método `roots` devuelve las raíces racionales.

```
s1 = 2*t-1
print s1.roots()
print s1.real_roots()
qs1 = s1.base_extend(QQ)
print qs1.roots()
```

```
[]
[0.500000000000000]
[(1/2, 1)]
```

El anillo de coeficientes puede ser $\mathbb{Z}_m$.

```
R1 = Integers(7)
PR5.<u> = PolynomialRing(R1)
r = u^2+3
r.roots()
```

```
[(5, 1), (2, 1)]
```

```
#evaluamos r en todos los elementos de Z_7
print [(j, r(u=j)) for j in srange(7)]
```

```
[(0, 3), (1, 4), (2, 0), (3, 5), (4, 5), (5, 0), (6, 4)]
```

Factorización

Podemos factorizar polinomios, o miembros de cualquier anillo

- **`factor(q)`** ó **`q.factor()`**: la factorización del elemento del anillo
- **`list(q.factor())`** : lista de tuplas (factor, multiplicidad)
- **`q.is_irreducible()`**: True si y sólo si q es irreducible.

```
q=2*(t^3 - 3*t^2 + 4)
print q.factor()
print list(q.factor())
print q, q.is_irreducible()
print q+1, (q+1).is_irreducible()
```

```
2 * (t + 1) * (t - 2)^2
[(2, 1), (t + 1, 1), (t - 2, 2)]
2*t^3 - 6*t^2 + 8 False
2*t^3 - 6*t^2 + 9 True
```

```
q=60
print q.factor()
print list(q.factor())
print q, q.is_irreducible()
print q+1, (q+1).is_irreducible()
```

```
2^2 * 3 * 5
[(2, 2), (3, 1), (5, 1)]
60 False
61 True
```

```
#Un elemento de Z_21 no se puede factorizar
R2(1).factor()
```

```
Traceback (click to the left of this block for traceback)
...
AttributeError: 'sage.rings.finite_rings.integer_mod.IntegerMod_int'
object has no attribute 'factor'
```

Matrices

```
#Matrices
M1 = matrix(QQ,[[0,1],[1,0]])
M2 = matrix(QQbar,2,2,[0,1,-1,0])

N = 3
M3 = matrix(RealField(14),[[1/(k+j+1) for j in range(N)] for k in range(N)])
M4 = matrix(RDF,[[1/(k+j+1) for j in range(N)] for k in range(N)])

M5 = matrix(Integers(7), [[3,2],[1,0]])
```

```
show(M1)
show(M2)
show(M3)
show(M4)
show(M5)
```

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

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1.00 & 0.500 & 0.333 \\ 0.500 & 0.333 & 0.250 \\ 0.333 & 0.250 & 0.200 \end{pmatrix}$$

$$\begin{pmatrix} 1.0 & 0.5 & 0.333333333333 \\ 0.5 & 0.333333333333 & 0.25 \\ 0.333333333333 & 0.25 & 0.2 \end{pmatrix}$$

$$\begin{pmatrix} 3 & 2 \\ 1 & 0 \end{pmatrix}$$

```
print M1.jordan_form()
print M2.jordan_form()
```

```
[ 1| 0]
[--+--]
[ 0|-1]
[-1*I|   0]
[----+----]
[   0| 1*I]
```

```
#La forma de Jordan no tiene sentido si hay errores numericos de por medio
M3.jordan_form()
```

```
Traceback (click to the left of this block for traceback)
...
ValueError: Jordan normal form not implemented over inexact rings.
```

```
#Trabajar con coeficientes en RealField(bits) permite observar los errores
#de redondeo mas de cerca
M3*(~M3+identity_matrix(3))-M3
```

```
[    1.00  0.00391 -0.00522]
[   0.000     1.00 -0.00391]
[0.000153    0.000    0.999]
```

```
#Las matrices en RDF tienen un metodo SVD (singular value decomposition)
M4.SVD()
```

```
(
[-0.827044926972  0.547448430721  0.127659329747]
[-0.459863904366 -0.528290235067 -0.713746885803]
[-0.323298435244 -0.649006658852  0.688671531671],

[   1.40831892712             0.0             0.0]
[             0.0   0.122327065854            0.0]
[             0.0             0.0 0.00268734035577],

[-0.827044926972  0.547448430721  0.127659329747]
[-0.459863904366 -0.528290235067 -0.713746885803]
[-0.323298435244 -0.649006658852  0.688671531671]
)
```

```
#Calculo en M_{2x2}[Z_m]
~M5
```

```
[0 1]
[4 2]
```

```
#Formas de mostrar un numero con menos digitos
print '%.4e'%(1/3)
print (1/3).n(digits=6)
```

```
3.3333e-01
0.333333
```

```
#Una forma de mostrar una matriz con menos digitos
#apply_map aplica una funcion a cada elemento de la matriz
#devuelve otra matriz, no modifica la matriz original
print M4.apply_map(lambda x:x.n(16))
```

```
[ 1.000 0.5000 0.3333]
[0.5000 0.3333 0.2500]
[0.3333 0.2500 0.2000]
```

Vectores y espacios vectoriales

Aunque es posible usar las matrices para hacer álgebra lineal con el ordenador, puede ser interesante usar los objetos que representan espacios y subespacios vectoriales. Algunos problemas se plantean de forma más conceptual, y se evitan errores usuales como escribir los coeficientes en el orden equivocado y terminar con la traspuesta de la matriz de paso.

```
V1 = VectorSpace(QQ,3)
V2 = VectorSpace(RDF,3)    #Numeros reales de precision doble
print V1
print V2
```

```
Vector space of dimension 3 over Rational Field
Vector space of dimension 3 over Real Double Field
```

```
v1 = V1([1,1,1])
v2 = V2([1,1,0])
#La suma de V1 y V2 tiene sentido en V2
v3 = 2*v1+v2
print v1 ,v1.parent()
print v2 ,v2.parent()
print v3 ,v3.parent()
```

```
(1, 1, 1) Vector space of dimension 3 over Rational Field
(1.0, 1.0, 0.0) Vector space of dimension 3 over Real Double Field
(3.0, 3.0, 2.0) Vector space of dimension 3 over Real Double Field
```

```
L1 = V1.subspace([v1,v2,v1+v2])
#Comprobacion de igualdad
print L1 == V1
print L1 == V1.subspace([v1,v1+v2])
```

```
False
True
```

```
#Comprobacion de inclusion
print L1 <= V1
print L1 >= V1
print L1 >= V1.subspace([v1])
```

```
True
False
True
```

```
#Si queremos podemos dotar al subespacio de una base
L3 = V1.subspace_with_basis([v1,v2])
print L1
print L3
#A pesar de tener distintas bases, ambos subespacios se declaran iguales
print L1 == L3
#Si no marcamos la base, la construye Sage (toma una matriz escalonada)
print L1.basis_matrix() == L3.basis_matrix()
```

```
Vector space of degree 3 and dimension 2 over Rational Field
Basis matrix:
[1 1 0]
[0 0 1]
Vector space of degree 3 and dimension 2 over Rational Field
User basis matrix:
[1 1 1]
[1 1 0]
True
False
```

```
#Coordenadas de
print L1.coordinates(v3)
print L3.coordinates(v3)
```

```
[3, 2]
[2, 1]
```

```
#Los subespacios nucleo e imagen, utiles para
#construir un subespacio dado por coordenadas
M1 = matrix(QQ,[[1,2,3,4],[4,2,3,1]])
show(M1)
print M1.kernel()   #lo mismo que M1.left_kernel()
print
print M1.image()
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 2 & 3 & 1 \end{pmatrix}$$

```
Vector space of degree 2 and dimension 0 over Rational Field
Basis matrix:
[]

Vector space of degree 4 and dimension 2 over Rational Field
Basis matrix:
[  1   0   0  -1]
[  0   1 3/2 5/2]
```

Ejercicio resuelto: Encuentra una base del subespacio de $\mathbb{C}^4$ dado por $x_1 + 2x_2 - x_4$ y una base de su intersección con el subespacio engendrado por $[1, 1, 1, 1]$ y $[1, 1, 0, 0]$.

```
V3 = VectorSpace(CDF,4)
v1 = vector([1,1,1,1])
v2 = vector([1,1,0,0])
L1 = V3.subspace([v1,v2])
```

```
#Subespacio dado por x1 + 2*x2 -x4 en V(CDF,4)
M = matrix(CDF,4,1,[1,2,0,-1])
print M
L2 = M.left_kernel()
print L2
print L2.intersection(L1)
```

```
[ 1.0]
[ 2.0]
[   0]
[-1.0]
Vector space of degree 4 and dimension 3 over Complex Double Field
Basis matrix:
[1.0   0   0 1.0]
[  0 1.0   0 2.0]
[  0   0 1.0   0]
Vector space of degree 4 and dimension 1 over Complex Double Field
Basis matrix:
[1.0 1.0 3.0 3.0]
```

Ejercicio resuelto: Expresa el vector v=(1,0,0,0) como suma de un vector del subespacio de $\mathbb{C}^4$ dado por $x_1 + 2x_2 - x_4 = 0, 2x_1 + x_4 = 0$y otro del subespacio engendrado por $[1, 1, 1, 1]$ y $[1, 1, 0, 0]$.

```
v  = vector([1,0,0,0])
v1 = vector([1,1,1,1])
v2 = vector([1,1,0,0])
L1 = V3.subspace([v1,v2])
M = matrix(CDF,4,2,[1,2,0,-1,  2,0,0,1])
L2 = M.left_kernel()
L1.intersection(L2), L1+L2
```

```
(Vector space of degree 4 and dimension 0 over Complex Double Field
Basis matrix:
[], Vector space of degree 4 and dimension 4 over Complex Double
Field
Basis matrix:
[1.0   0   0   0]
[  0 1.0   0   0]
[  0   0 1.0   0]
[  0   0   0 1.0])
```

```
d1, d2 = L1.dimension(), L2.dimension()
base   = L1.basis() + L2.basis()
V  = L1.ambient_vector_space()
#V y V2 son el mismo espacio, pero con distinta base
V2 = V.subspace_with_basis(base)
coords = V2.coordinates(v)
v  = sum(coords[j]*base[j] for j in range(d1+d2) )
v1 = sum(coords[j]*base[j] for j in range(d1) )
v2 = sum(coords[j]*base[j] for j in range(d1,d1+d2) )
print v
print v1
print v2
```

```
(1.0, 4.4408920985e-16, 0, 0)
(3.0, 3.0, -1.0, -1.0)
(-2.0, -3.0, 1.0, 1.0)
```

Método de Gauss a mano

A veces nos interesa que hagan las cuentas, pero sin echar la vida en ello. El ordenador debe ser poco más que una calculadora. A modo de ejemplo, ponemos una matriz en forma escalonada poco a poco (kudos para Juan Ramón Esteban).

Ejercicio resuelto: Encuentra una base de los siguientes subespacios de $\mathbb{C}^4$:

- el subespacio engendrado por $[1,2,0,-1]$ y $[2,0,0,1]$
- el subespacio engendrado por $[0,1,-1,-2]$ y $[2,2,2,2]$.
- el subespacio suma de los dos anteriores

```
#Metodo de Gauss
#suma de ...
M1 = matrix(CDF,2,4,[1,2,0,-1,  2,0,0,1])
show(M1)
M1.echelonize()
show(M1)
```

$$\begin{pmatrix} 1.0 & 2.0 & 0 & -1.0 \\ 2.0 & 0 & 0 & 1.0 \end{pmatrix}$$

$$\begin{pmatrix} 1.0 & 0 & 0 & 0.5 \\ 0 & 1.0 & 0 & -0.75 \end{pmatrix}$$

```
M2 = matrix(CDF,2,4,[0,1,-1,-2,  2,2,2,2])
show(M2)
#hacemos lo mismo con M2, pero a mano
M2.swap_rows(0,1)
```

```
M2
```

$$\begin{pmatrix} 0 & 1.0 & -1.0 & -2.0 \\ 2.0 & 2.0 & 2.0 & 2.0 \end{pmatrix}$$

```
[ 2.0  2.0  2.0  2.0]
[   0  1.0 -1.0 -2.0]
```

```
M1.rows() + M2.rows()
```

```
[(1.0, 0, 0, 0.5), (0, 1.0, 0, -0.75), (2.0, 2.0, 2.0, 2.0), (0,
1.0, -1.0, -2.0)]
```

```
#Ahora buscamos una base de el subespacio suma, muy despacio
M=matrix(M1.rows()+M2.rows())
M
```

```
[  1.0     0     0   0.5]
[    0   1.0     0 -0.75]
[  2.0   2.0   2.0   2.0]
[    0   1.0  -1.0  -2.0]
```

```
M.add_multiple_of_row(2,0,-2)
M
```

```
[  1.0     0     0   0.5]
[    0   1.0     0 -0.75]
[    0   2.0   2.0   1.0]
[    0   1.0  -1.0  -2.0]
```

```
M.add_multiple_of_row(2,1,-2)
M.add_multiple_of_row(3,1,-1)
M
```

```
[  1.0     0     0   0.5]
[    0   1.0     0 -0.75]
[    0     0   2.0   2.5]
[    0     0  -1.0 -1.25]
```

```
M.add_multiple_of_row(3,2,0.5)
M
```

```
[  1.0     0     0   0.5]
[    0   1.0     0 -0.75]
[    0     0   2.0   2.5]
[    0     0     0     0]
```

```
M.set_row_to_multiple_of_row(2,2,1/2)
M
```

```
[  1.0     0     0   0.5]
[    0   1.0     0 -0.75]
[    0     0   1.0  1.25]
[    0     0     0     0]
```

```
#Matrices con parametros
#Definimos una variable simbolica para el parametro
var('a')
#Los elementos de la matriz estan en el anillo SR,
#de expresiones simbolicas
M = matrix(3,[1,1,-1, 1,2,-1,  a,2,2])
M
```

```
[ 1  1 -1]
[ 1  2 -1]
[ a  2  2]
```

```
#Para calcular el rango de M dependiendo de a
#no sirve de nada pedir la forma escalonada de M en SR
#porque a es invertible en ese anillo
M.echelon_form()
```

```
[1 0 0]
[0 1 0]
[0 0 1]
```

```
#Pero se puede hacer "semi-a-mano"
M.add_multiple_of_row(1,0,-1)
M.add_multiple_of_row(2,0,-a)
M
```

```
[     1      1     -1]
[     0      1      0]
[     0 -a + 2  a + 2]
```

```
M.add_multiple_of_row(2,1,a-2)
M
```

```
[    1     1    -1]
[    0     1     0]
[    0     0 a + 2]
```

Créditos

Mis agradecimientos a Juan Ramón Esteban y a mis compañeros de laboratorio: Patricio Cifuentes, Daniel Ortega, Rafael Hernández, Bernardo López por sus comentarios en los pasillos.

Para profundizar

- Nuestros apuntes de laboratorio de Álgebra. http://www.uam.es/personal_pdi/ciencias/pangulo/doc/laboratorio/bloqueIII.html
- Pregunta a la comunidad!: http://ask.sagemath.org/
- Ayuda con Sage: http://sagemath.org/help.html