Referencia Comandos Sage: Álgebra Lineal

Adaptado de Robert A. Beezer

Sage Version 8.0

`http://wiki.sagemath.org/quickref`

Construcción de matrices

Nota: La numeración de columna y fila comienza en 0

`A = matrix(ZZ, [[1,2],[3,4],[5,6]])`
3×2 sobre los enteros

`B = matrix(QQ, 2, [1,2,3,4,5,6])`
2 filas sobre lista, por lo que 2×3 sobre racionales

`C = matrix(CDF, 2, 2, [[5*I, 4*I], [I, 6]])`
números complejos, precision 53-bit

`L = matrix(ZZ, 20, 80, {(5,9):30, (15,77):-6})`
20×80, sólo dos dos entradas distintas de cero

`Z = matrix(QQ, 2, 2, 0)` Matriz cero 2×2

`D = matrix(QQ, 2, 2, 8)`
diagonales valor 8, resto elementos nulos

`II = identity_matrix(5)` matriz identidad 5×5

Nota: `I` $= \sqrt{-1}$, no sobreescribir la variable

`var('x y z'); K = matrix(SR, [[x,y+z],[0,x^2*z]])`
expresiones simbólicas sobre el anillo `SR`

Operaciones con Matrices

`5*A+2*B` Combinación lineal

`A.inverse()`, `A^(-1)`, `~A`, singular: `ZeroDivisionError`

`A.transpose()`, transpuesta

`A.adjoint()` matriz adjunta

Multiplicación de Matrices

```
u = vector(QQ, [1,2,3]),  v = vector(QQ, [1,2])
A = matrix(QQ, [[1,2,3],[4,5,6]])
B = matrix(QQ, [[1,2],[3,4]])
```

`u*A`, `A*v`, `B*A`, `B^6`, `B^(-3)` posibles operaciones

`B.iterates(v, 6)` retorna $vB^0, vB^1, \ldots, vB^5$
`rows = False` postmultiplica por `v`

Combinar Matrices

`A.augment(B)` `A` primeras columnas, matrix `B` a la derecha

`A.stack(B)` `A` en filas superiores, `B` filas inferiores; `B` puede ser un vector

Operaciones por filas en matrices

Nota: la primera fila se numera como 0.

`A.rescale_row(i,a)` `a*`(fila `i`)

`A.add_multiple_of_row(i,j,a)` `a*`(fila `j`) + fila `i`

`A.swap_rows(i,j)`

Similar para columnas, `row`→`col`

Para crear una nueva matriz, usar p.e..

`B = A.with_rescaled_row(i,a)`

Matrices por trozos

Nota: numeración fila, columna comienza en 0

`A.nrows()`, `A.ncols()`

`A[i,j]` entrada in fila `i` y columna `j`

`A[i]` fila `i` es una tupla inmutable. Por lo que,
Atención: OK: `A[2,3] = 8`, Error: `A[2][3] = 8`

`A.row(i)` retorna fila `i` como un vector Sage

`A.column(j)` retorna columna `j` como un vector Sage

`A.list()` retorna A como lista Python,por filas

`A.matrix_from_columns([8,2,8])`
nueva matriz a partir de las columnas de la lista

`A.matrix_from_rows([2,5,1])`
nueva matriz a partir de las filas de la lista no ordenada

`A.matrix_from_rows_and_columns([2,4,2],[3,1])`
elementos comunes de filas y columnas

`A.rows()` todas las filas como una lista de tuplas

`A.columns()` todas las columnas como una lista de tuplas

`A.submatrix(i,j,nr,nc)`
empieza en la entrada `(i,j)`, toma `nr` filas `nc` columnas

`A[2:4,1:7]`, `A[0:8:2,3::-1]` submatrices, similar a dividir listas en Python

Propiedades de Matrices

`.is_zero()`; `.is_symmetric()`; `.is_hermitian()`;
`.is_square()`; `.is_orthogonal()`; `.is_unitary()`;
`.is_scalar()`; `.is_singular()`; `.is_invertible()`;
`.is_one()`; `.is_nilpotent()`; `.is_diagonalizable()`

Espacios de Matrices

`M = MatrixSpace(QQ, 3, 4)` espacio de las matrices 3×4

`A = M([1,2,3,4,5,6,7,8,9,10,11,12])`
transforma lista a elemento de `M`, matriz 3×4 sobre `QQ`

`M.basis()`

`M.dimension()`

`M.zero_matrix()`

Cuerpos

Nota: Muchos algoritmos depende del cuerpo sobre el que se implementa

`<object>.base_ring(R)` para vectores, matrices,...
para determinar el cuerpo de un objeto

`<object>.change_ring(R)` para vectores, matrices,...
para cambiar el cuerpo de un objeto, `R`

Algunos cuerpos que implementa Sage

- `ZZ` enteros, anillo
- `QQ` racionales, cuerpo
- `RDF` cuerpo números reales , inexacto
- `CDF` cuerpo números complejos, inexacto
- `RR` números reales representados con 53-bit
- `RealField(400)` reales 400 bits, inexacto
- `CC`, `ComplexField(400)` complejos

Funciones Escalares en Matrices

`A.rank()`

`A.determinant() == A.det()`

`A.norm() == A.norm(2)` Norma Euclídea

`A.norm(1)` mayor suma por columnas

`A.norm(Infinity)` mayor suma por fila

`A.norm('frob')` norma de Frobenius

Echelon Form

`A.rref()`, `A.echelon_form()`, `A.echelonize()`

Nota: `rref()` cambia el cuerpo base a los racionales

```
A = matrix(ZZ,[[4,2,1],[6,3,2]])
```

`A.rref()`
$$\begin{pmatrix} 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

`A.echelon_form()`
$$\begin{pmatrix} 1 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Decomposiciones

`A.LU()` tripleta con: `P*A == L*U`
`P`: una matriz de permutación
`L`: triangular inferior, `U`: triangular superior

`A.QR()` doblete con: `A == Q*R`
`Q`: una matriz formada por columnas ortormales
`R`: triangular superior inversible

Resolución de sistemas

`A.solve_right(B)` `_left` también
es solucion a `A*X = B`, donde `X` es un vector **o** matriz
`A = matrix(QQ, [[1,2],[3,4]])`
`b = vector(QQ, [3,4])`, entonces `A\b` es solución `(-2, 5/2)`

Constructor de vectores

Nota: Las posiciones de los vectores empiezan en 0
`u = vector(QQ, [1, 3/2, -1])` vector de dimensión 3 con coeficientes racionales
`v = vector(QQ, {2:4, 95:4, 210:0})`
vector de dimension 211 donde solo la entrada 2 y la 95 son diferentes de cero

Operaciones con vectores

`u = vector(QQ, [1, 3/2, -1])`
`v = vector(ZZ, [1, 8, -2])`
`2*u - 3*v` combinación lineal
`u.dot_product(v)`
`u.norm() == u.norm(2)` norma euclídea
`A.gram_schmidt()` ortogonalización de Gram Schmidt

Espacios Vectoriales

`VectorSpace(QQ, 4)` Espacio vectorial de dimensión 4 sobre los racionales
`VectorSpace(RR, 4)` El "cuerpo" son los números representables con 53 bits en coma flotante
`VectorSpace(RealField(200), 4)`
el cuerpo ahora tiene 200 bits de precisión
`CC^4` el cuerpo contiene números complejos, representables en coma flotante con 53 bits.
`Y = VectorSpace(GF(7), 4)` espacio vectorial finito de dimensión 4 con
`Y.list()` $7^4 = 2401$ vectores

Valores propios y vectores propios

Nota: La implementación varia entre utilizar anillos exactos como (`QQ`) y `RDF`, `CDF`
`A.charpoly('t')` en caso de no especificar el nombre de la variable se utilizará `x`
`A.characteristic_polynomial() == A.charpoly()`
`A.fcp('t')` factorización del polinomio característico
`A.minpoly()` polinomio mínimmo
`A.minimal_polynomial() == A.minpoly()`
`A.eigenvalues()` valores propios, con sus multiplicidades

Operaciones en espacios vectoriales

`V.dimension()`
`V.basis()`
`V.is_subspace(W)` `True` si `W` es un subespacio de `V`

Fuentes adicionales de ayuda

"tab-completion" para completar comandos
"tab-completion" en `<object.>` para los métodos
`<command>?` para un resumen de la ayuda y ejemplos
`<command>??` para el código fuente en python completo