Sage Quick Reference
Accompaniment to Tea Time Linear Algebra
Sage Version 9.4

Matrix Constructions

Caution: Row, column numbering begins at 0
Caution: `I` $= \sqrt{-1}$, do not overwrite with matrix name
`A = matrix(3,2,[1,2,3,4,5,6])` 3×2 over integers
  or `matrix([[1,2],[3,4],[5,6]])`
`B = matrix(2, [1,2,3,4,5,6])` 2 rows from list, so 2×3
`C = matrix([[5*I, 4*I], [I, 6]])` complex entries
`D = matrix(4, 4, 8)` diagonal entries 8, other entries 0
`Z = matrix(4, 4, 0)` zero matrix
`II = matrix(5, 5, 1)` 5×5 identity matrix
  or `identity_matrix(5)`
`A.augment(B)` `A` in first columns, matrix `B` to the right
`A.stack(B)` `A` in top rows, `B` below; `B` can be a vector
`L = matrix(20, 80, {(5,9):30, (15,77):-6})`
  20×80, two non-zero entries, sparse representation
`J = jordan_block(-2,3)`
  3×3 matrix, -2 on diagonal, 1's on super-diagonal
`E = block_matrix([[P,0],[1,R]])`, very flexible input
`A.block_sum(B)` Diagonal, `A` upper left, `B` lower right
Note: Matrices with symbolic entries are possible

Vector Constructions

Caution: First entry of a vector is numbered 0
`u = vector([1, 3/2, -1])` length 3 over rationals
`v = vector({2:4, 95:4, 210:0})`
  211 entries, nonzero in entry 2 and entry 95, sparse

Matrix Multiplication

`u = vector([1,2,3])`, `v = vector([1,2])`
`A = matrix([[1,2,3],[4,5,6]])`
`B = matrix([[1,2],[3,4]])`
`u*A`, `A*v`, `B*A`, `B^6`, `B^(-3)` all possible
`B.iterates(v, 6, rows = False)`
  produces $B^0v, B^1v, \ldots, B^5v$
  `rows` not specified defaults to $vB^0, vB^1, \ldots, vB^5$
`f(x)=x^2+5*x+3` then `f(B)` is possible
`B.exp()` matrix exponential, i.e. $\sum_{k=0}^{\infty} \frac{1}{k!}B^k$

Matrix Operations

`5*A+2*B` linear combination
`A.inverse()`, `A^(-1)`, `~A`
  when `A` is singular produces `ZeroDivisionError`
`A.transpose()`
`A.adjoint()` matrix of cofactors
`A.gram_schmidt()` converts the rows of matrix `A`
`A.conjugate()` entry-by-entry complex conjugates
`A.conjugate_transpose()`
`A.antitranspose()` transpose + reverse orderings
`A.restrict(V)` restriction to invariant subspace `V`

Eigenvalues and Eigenvectors

Note: Different behavior for exact rings (`QQ`) and inexact rings (`RDF`)
`A.eigenvalues()` unsorted list, with mutiplicities
`A.eigenvectors_left()` vectors on left, `_right` too
`A.charpoly('t')` no variable specified defaults to `x`
  or `A.characteristic_polynomial()`
`A.fcp('t')` factored characteristic polynomial
  Returns, per eigenvalue, a triple: `e`: eigenvalue;
  `V`: list of eigenspace basis vectors; `n`: multiplicity
`A.eigenmatrix_right()` vectors on right, `_left` too
  Returns pair: `D`: diagonal matrix with eigenvalues
  `P`: eigenvectors as columns (rows for left version)
    with zero columns if matrix not diagonalizable
`A.minimal_polynomial()` the minimal polynomial
  or `A.minpoly()`

Row Operations (change matrix in place)

Caution: first row is numbered 0
`A.rescale_row(i,a)` `a`*(row `i`)
`A.add_multiple_of_row(i,j,a)` `a`*(row `j`) + row `i`
`A.swap_rows(i,j)`
Each has a column variant, `row`$\to$`col`
For new matrix, use e.g. `B = A.with_rescaled_row(i,a)`

Echelon Form

`A = matrix([[4,2,1],[6,3,2]])`
  `A.echelon_form()` `A.rref()`

$$\begin{pmatrix} 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Note: `rref()` promotes matrix to fraction field
`A.pivots()` indices of columns spanning column space
`A.pivot_rows()` indices of rows spanning row space

Solutions to Systems

`A.solve_right(B)` `_left` too
  is solution to `A*X = B`, where `X` is a vector **or** matrix
`A = matrix([[1,2],[3,4]])`
`b = vector([3,4])`, then `A\b` is solution `(-2, 5/2)`

Pieces of Matrices

Caution: Row, column numbering begins at 0
`A.nrows()`, `A.ncols()`
`A[i,j]` entry in row `i` and column `j`
`A[i,:]` row `i` as a Sage matrix
`A.delete_rows([1,5])`, `A.delete_columns([2,3,7])`
`A[2:4,1:7]` Python-style list slicing
  or `A.submatrix(2,1,2,6)`
`A.submatrix(i,j,nr,nc)`
  start at entry `(i,j)`, use `nr` rows, `nc` cols
`A.row(i)` row `i` as Sage vector
`A.column(j)` column `j` as Sage vector
`A.list()` returns single Python list, row-major order
`A.matrix_from_columns([8,2,8])`
  new matrix from columns in list, repeats OK
`A.matrix_from_rows([2,5,1])`
  new matrix from rows in list, out-of-order OK
`A.matrix_from_rows_and_columns([2,4,2],[3,1])`
  common to the rows and the columns
`A.rows()` all rows as a list of tuples
`A.columns()` all columns as a list of tuples

Scalar Functions on Matrices

`A.rank()`, `A.nullity()`, `A.trace()`, `A.norm()`
`A.determinant()` or `A.det()`
`A.permanent()`
`A.norm(1)` largest column sum
`A.norm(Infinity)` largest row sum
`A.norm('frob')` Frobenius norm

Vector Operations

`2*u - 3*v` linear combination
`u * v` dot product
  or `u.dot_product(v)`
`u.norm()`
`u.cross_product(v)` order: `u`$\times$`v`
`u.inner_product(v)` inner product matrix from parent
`u.pairwise_product(v)` vector as a result
`u.norm(1)` sum of entries

u.norm(Infinity) maximum entry

Decompositions

Note: availability depends on base ring of matrix,
try RDF or CDF for numerical work, QQ for exact
"unitary" is "orthogonal" in real case

A.LU() triple with: P*A == L*U
P: a permutation matrix
L: lower triangular matrix, U: upper triangular matrix

A.QR() pair with: A == Q*R
Q: a unitary matrix, R: upper triangular matrix

A.jordan_form(transformation=True)
returns a pair of matrices with: A == P^(-1)*J*P
J: matrix of Jordan blocks for eigenvalues
P: nonsingular matrix

A.smith_form() triple with: D == U*A*V
D: elementary divisors on diagonal
U, V: with unit determinant

A.SVD() triple with: A == U*S*(V-conj-transpose)
U: a unitary matrix
S: zero off the diagonal, dimensions same as A
V: a unitary matrix

A.schur() pair with: A == Q*T*(Q-conj-transpose)
Q: a unitary matrix
T: upper-triangular matrix, maybe 2×2 diagonal blocks

A.rational_form(), aka Frobenius form

Matrix Properties

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

Subspaces

span([v1,v2,v3], RDF) span of list of vectors over reals

For a matrix A, objects returned are
vector spaces when base ring is a field
modules when base ring is just a ring

A.kernel(), A.row_space(), A.column_space()
A.eigenspaces_right() vectors on right, _left too
Pairs: eigenvalues with their right eigenspaces

If V and W are subspaces
V.subspace([v1,v2,v3]) specify basis vectors in a list
V.intersection(W) intersection of V and W
V.quotient(W) quotient of V by subspace W
V.direct_sum(W) direct sum of V and W

Vector Space Properties

V.dimension(), V.basis(), V.echelonized_basis()
V.is_subspace(W) True if W is a subspace of V

Rings

R.is_ring(), R.is_field(), R.is_exact()

Some common Sage rings and fields
- ZZ integers, ring
- QQ rationals, field
- RDF real double field, inexact
- CDF complex double field, inexact
- RealField(400) 400-bit reals, inexact
- CC, ComplexField(400) complexes, too
- SR ring of symbolic expressions
- GF(2) mod 2, field, specialized implementations
- GF(p) == FiniteField(p) p prime, field
- Integers(6) integers mod 6, ring only
- AA, QQbar algebraic number fields, exact
- RR 53-bit reals, inexact, not same as RDF
- RIF real interval field
- CyclotomicField(7) rationals with 7^{th} root of unity
- QuadraticField(-5, 'x') rationals with x=$\sqrt{-5}$

Note: Many algorithms depend on the base ring
<object>.base_ring(R) for vectors, matrices,...
to determine the ring in use
<object>.change_ring(R) for vectors, matrices,...
to change to the ring (or field), R

Dense versus Sparse

Note: Algorithms may depend on representation
Vectors and matrices have two representations
Dense: lists, and lists of lists
Sparse: Python dictionaries
.is_dense(), .is_sparse() to check
A.sparse_matrix() returns sparse version of A
A.dense_rows() returns dense row vectors of A
Some commands have boolean sparse keyword

More Help

"tab-completion" on partial commands
"tab-completion" on <object.> for all relevant methods
<command>? for summary and examples
<command>?? for complete source code

Based on Robert A. Beezer's Quick Reference
http://wiki.sagemath.org/quickref