Matrix Multiplication
\[ u = \text{vector}(QQ, [1,2,3]), \ v = \text{vector}(QQ, [1,2]) \]
\[ A = \text{matrix}(QQ, [[1,2],[3,4]]) \]
\[ u*Av, B^*A, B^*(-3) \]
\[ \text{rows} = \text{False} \] moves \( v \) to the right of matrix powers
\[ f(x) = x^2 + 5x + 3 \text{ then } f(B) \] is possible
\[ \text{B.exp()} \] matrix exponential, i.e. \( \sum_{k=0}^{\infty} \frac{B^k}{k!} \)

Matrix Spaces
\[ M = \text{MatrixSpace}(QQ, 3, 4) \] is space of \( 3 \times 4 \) matrices
\[ A = M([1,2,3,4,5,6,7,8,9,10,11,12]) \]
\[ \text{coerce list to element of } M, \ a 3 \times 4 \] matrix over \( QQ \)
\[ M.basis() \]
\[ M.dimension() \]
\[ M.zero_matrix() \]

Matrix Operations
\[ 5*A+2*B \] linear combination
\[ A^{-1} \] singular is ZeroDivisionError
\[ A.transpose() \]
\[ A.conjugate() \] entry-by-entry complex conjugates
\[ A.antitranspose() \] transpose + reverse orderings
\[ A.adjoint() \] matrix of cofactors
\[ A.restrict(V) \] restriction to invariant subspace \( V \)

Row Operations
\[ \text{A.swap_rows(i,j), A.swap_cols(i,j)} \]
\[ \text{Each has a column variant, row} \rightarrow \text{col} \]
For a new matrix, use e.g. \( B = A.with_rescaled_row(i,a) \)

Echelon Form
\[ \text{A.rref()}, \text{A.echelon_form()}, \text{A.echolize()} \]
\[ \text{Note: rref()} \] promotes matrix to fraction field
\[ A = \text{matrix}(ZZ,[[4,2,1],[6,3,2]]) \]
\[ A.rref() \]
\[ A.echelon_form() \]
\[ \begin{bmatrix} 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

Matrix Properties
\[ A.pivots() \text{ indices of columns spanning column space} \]
\[ A.pivot_rows() \text{ indices of rows spanning row space} \]

Pieces of Matrices
\[ \text{Caution: row, column numbering begins at 0} \]
\[ A[0,0] \] entry in row 0 and column 0
\[ A[1] \] row 1 as immutable Python tuple. Thus,
\[ A.row(i) \] returns row \( i \) as Sage vector
\[ A.column(j) \] returns 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
\[ A.submatrix([i,j,nr,nc]) \]
start at entry \((i,j)\), use \( nr \) rows, \( nc \) cols
\[ A[2,4:1:7], A[0:8:2,3:1] \] Python-style list slicing

Combining Matrices
\[ A.augment(B) \] A in first columns, matrix B to the right
\[ A.block_lower(B) \] A in top rows, B below; B can be a vector
\[ A.block_upper(B) \] Diagonal, A upper left, B lower right
\[ A.tensor_product(B) \] Multiples of \( B \), arranged as in \( A \)

Scalar Functions on Matrices
\[ A.rank(), A.right_nullity() \]
\[ A.left_nullity() = A.nullity() \]
\[ A.determinant() = A.det() \]
\[ A.pivots(), A.trace() \]
\[ A.permutant() = A.norm(2) \] Euclidean norm
\[ A.norm() \] largest column sum
\[ A.norm(\infty) \] largest row sum
\[ A.norm(\text{frob}) \] Frobenius norm

Matrix Properties
\[ .is_zero(); .is_symmetric(); .is_hermitian(); .is_square(); .is_orthogonal(); .is_unitary(); .is_scalar(); .is_singular(); .is_invertible(); .is_one(); .is_nilpotent(); .is_diagonalizable() \]
Eigenvalues and Eigenvectors

Note: Contrast behavior depends on exact rings (QQ vs. RDF, CDF)

A.charpoly('t') no variable specified defaults to x
A.characteristic_polynomial() == A.charpoly()
A.fcp('t') factored characteristic polynomial
A.minpoly() the minimum polynomial
A.minimal_polynomial() == A.mminpoly()
A.eigenvectors_left() returns a pair of matrices with:
    A: a unitary matrix  
    D: a diagonal matrix with eigenvalues
P: nonsingular matrix
A.eigenvectors_right() vectors on right, .right too
    U: upper-triangular matrix, maybe 2×2 diagonal blocks
V: a unitary matrix
Q: a unitary matrix
A.eigenform() aka Frobenius form
A.symmetric_form()
A.hessenberg_form()
A.cholesky() (needs work)

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.jordan_form(transformation=Square) returns a pair of matrices with:
    A: 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.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.SVD() triple with:
    A == U*S*(V-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-transpose)
    Q: a unitary matrix
    T: upper-triangular matrix, maybe 2×2 diagonal blocks
A.rational_form(), aka Frobenius form
A.symmetric_form()
A.hessenberg_form()
A.cholesky() (needs work)

Solutions to Systems

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

Vector Spaces

VectorSpace(QQ, 4) dimension 4, rationals as field
VectorSpace(RR, 4) “field” is 53-bit precision reals
VectorSpace(RealField(200), 4)
    “field” has 200 bit precision
CC^4 4-dimensional, 53-bit precision complexes
Y = VectorSpace(GF(7), 4) finite
    Y.list() has 7^4 = 2401 vectors

Vector Space Properties

V.dimension()
V.basis()
V.echelonized_basis()
V.has_user_basis()
V.is_subspace(W) True if W is a subspace of V
V.is_full() rank equals degree (as module)?
Y = GF(7)^4, 4 finite
    Y.list() has 7^4 = 2401 vectors

Constructing Subspaces

span([v1, v2, v3], QQ) span of list of vectors over ring

For a matrix A, objects returned are
    vector spaces when base ring is a field
    modules when base ring is just a ring
A.left_kernel() == A.kernel() right too
A.row_space() == A.row_module()
A.column_space() == A.column_module()
A.eigenspaces_right() vectors on right, .right too
    Pairs: eigenvalues with their right eigenspaces
A.eigenspaces_right(form='galois')
    One eigenspace per irreducible factor of char poly

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

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

Rings

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

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

Vector Spaces versus Modules

Module “is” a vector space over a ring, rather than a field
Many commands above apply to modules
Some “vectors” are really module elements

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