Sage Quick Reference: Linear Algebra

Robert A. Beezer
Sage Version 4.8
`http://wiki.sagemath.org/quickref`

Based on work by Peter Jipsen, William Stein

Vector Constructions

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

Vector Operations

`u = vector(QQ, [1, 3/2, -1])`
`v = vector(ZZ, [1, 8, -2])`
`2*u - 3*v` linear combination
`u.dot_product(v)`
`u.cross_product(v)` order: `u`×`v`
`u.inner_product(v)` inner product matrix from parent
`u.pairwise_product(v)` vector as a result
`u.norm() == u.norm(2)` Euclidean norm
`u.norm(1)` sum of entries
`u.norm(Infinity)` maximum entry
`A.gram_schmidt()` converts the rows of matrix `A`

Matrix Constructions

Caution: Row, column numbering begins at 0
`A = matrix(ZZ, [[1,2],[3,4],[5,6]])`
  3×2 over the integers
`B = matrix(QQ, 2, [1,2,3,4,5,6])`
  2 rows from a list, so 2×3 over rationals
`C = matrix(CDF, 2, 2, [[5*I, 4*I], [I, 6]])`
  complex entries, 53-bit precision
`Z = matrix(QQ, 2, 2, 0)` zero matrix
`D = matrix(QQ, 2, 2, 8)`
  diagonal entries all 8, other entries zero
`E = block_matrix([[P,0],[1,R]])`, very flexible input
`II = identity_matrix(5)` 5×5 identity matrix
  $\mathrm{I} = \sqrt{-1}$, do not overwrite with matrix name
`J = jordan_block(-2,3)`
  3×3 matrix, -2 on diagonal, 1's on super-diagonal
`var('x y z'); K = matrix(SR, [[x,y+z],[0,x^2*z]])`
  symbolic expressions live in the ring `SR`
`L = matrix(ZZ, 20, 80, {(5,9):30, (15,77):-6})`
  20×80, two non-zero entries, sparse representation

Matrix Multiplication

`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)` all possible
`B.iterates(v, 6)` produces $vB^0, vB^1, \ldots, vB^5$
  `rows = False` moves `v` to the right of matrix powers
`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 Spaces

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

Matrix Operations

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

Row Operations

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`→`col`
For a new matrix, use e.g. `B = A.with_rescaled_row(i,a)`

Echelon Form

`A.rref()`, `A.echelon_form()`, `A.echelonize()`
Note: `rref()` promotes matrix to fraction field
`A = matrix(ZZ,[[4,2,1],[6,3,2]])`

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

`A.pivots()` indices of columns spanning column space
`A.pivot_rows()` indices of rows spanning row space

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 immutable Python tuple. Thus,
  Caution: OK: `A[2,3] = 8`, Error: `A[2][3] = 8`
`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.stack(B)` `A` in top rows, `B` below; `B` can be a vector
`A.block_sum(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.permanent()`, `A.trace()`
`A.norm() == A.norm(2)` Euclidean norm
`A.norm(1)` largest column sum
`A.norm(Infinity)` largest row sum
`A.norm('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 for 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.minpoly()`
`A.eigenvalues()` unsorted list, with mutiplicities
`A.eigenvectors_left()` vectors on left, `_right` too
  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
Eigenspaces: see "Constructing Subspaces"

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=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.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-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
`A.symplectic_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()` with non-canonical 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`, `T = Y.subspaces(2)`
  `T` is a generator object for 2-D subspaces of `Y`
  `[U for U in T]` is list of 2850 2-D subspaces of `Y`,
  or use `T.next()` to step through subspaces

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, `_left` too
  Pairs: eigenvalues with their right eigenspaces
`A.eigenspaces_right(format='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 7^{th} 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