

Import SymPy

```
import sympy as sp
```

Matrix Creation

normal Matrix	<code>sp.Matrix([[1,2],[-3,4]])</code>
Matrix with all zeros	<code>sp.zeros(4,5)</code>
Matrix with all ones	<code>sp.ones(4,5)</code>
Square matrix with all zeros	<code>sp.zeros(5)</code>
Square matrix with all ones	<code>sp.ones(5)</code>
Identity matrix	<code>sp.eyes(5)</code>
Diagonal Matrix	<code>sp.diag(1,2,3,4)</code>
Generate element with func(i,j)	<code>sp.Matrix(2,3,func)</code>

Matrix Modification

Delete the i-th row	<code>M.row_del(i)</code>
Delete the j-th column	<code>M.col_del(j)</code>
Row join M1 and M2	<code>M1.row_join(M2)</code>
Column join M1 and M2	<code>M1.col_join(M2)</code>

Indexing(Slicing)

get the element in M at (i,j)	<code>M[i,j]</code>
get the i-th row in M	<code>M.row(i)</code>
get the i-th row in M	<code>M[i,:]</code>
get the j-th column in M	<code>M.col(j)</code>
get the j-th column in M	<code>M[:,j]</code>
get the i-th and the k-th rows	<code>M[[i,k],:]</code>
get the j-th and the k-th columns	<code>M[:,j,k]</code>
get rows from i to k	<code>M[i:k,:]</code>
get columns from j to k	<code>M[:,j:k]</code>
get sub-matrix (row i to k,col j to l)	<code>M[i:k,j:l]</code>

Note: All indices start from 0

Basic operations

Sum	<code>A+B</code>
Subtraction	<code>A-B</code>
Matrix Multiply	<code>A*B</code>

Basic operations (cont)

Scalar Multiply	<code>5*A</code>
Elementwise product	<code>sp.matrix_multiply_elementwise(A,B)</code>
Transpose	<code>A.T</code>
Determinant	<code>A.det()</code>
Inverse	<code>A.inv()</code>
Condition Number	<code>A.condition_number()</code>
Row count	<code>A.rows</code>
Column count	<code>A.cols</code>
Trace	<code>A.trace()</code>

Elementary Row Operations

Replacement	<code>m.row_op(i, lambda ele, col: ele+m.row(j)[col]*c)</code>
Interchange	<code>M.row_swap(i,j)</code>
Scaling	<code>m.row_op(i, lambda ele, col: ele*c)</code>

Linear Equations

Echelon Form	<code>M.echelon_form()</code>
Reduced Echelon Form	<code>M.rref()</code>
Solve AX=B (B can be a matrix)	<code>x, freevars=A.gauss_jordan_solve(B)</code>
least-square fit Ax=b	<code>A.solve_least_squares(b)</code>
solve Ax=b	<code>A.solve(b)</code>

Vector Space

Basis of column space	<code>M.columnspace()</code>
Basis of null space	<code>M.nullspace()</code>
Basis of row space	<code>M.rowspace()</code>
Rank	<code>M.rank()</code>

Eigenvalues and Eigenvectors

Find the eigenvalues	<code>M.eigenvals()</code>
Find the eigenvalues and the corresponding eigenspace	<code>M.eigenvecs()</code>

Eigenvalues and Eigenvectors (cont)

Diagonalize a matrix	<code>P, D = M.diagonalize()</code>
test if the matrix is diagonalizable	<code>M.is_diagonalizable</code>
Calculate Jordan Form	<code>P, J = M.jordan_form()</code>

Decomposition

LU Decomposition (PA=LU)	<code>P,L,U=A.LUdecomposition()</code>
QR Decomposition	<code>Q,R=A.QRdecomposition()</code>

Vector Operations

Create a column vector	<code>v=sp.Matrix([1,-2,3])</code>
dot product	<code>v1.dot(v2)</code>
cross product	<code>v1.cross(v2)</code>
length of the vector	<code>v.norm()</code>
normalize of vector	<code>v.normalize()</code>
the projection of v1 on v2	<code>v1.project(v2)</code>
Gram-Schmidt orthogonalize	<code>sp.GramSchmidt([v1,v2,v3])</code>
Gram-Schmidt orthogonalize with normalization	<code>sp.GramSchmidt([v1,v2,v3], True)</code>
Singular values	<code>M.singular_values()</code>

Block Matrix

Create a matrix by block	<code>M=sp.Matrix([[A,B],[C,D]])</code>
--------------------------	---

Useful Links

[SymPy Documentation](#)
[SymPy Tutorial](#)