Magma: Working with matrices and all that

Charlotte Roelants

16 December 2025

Overview

Vector and vector space constructions

Matrix constructions

Elementary operations

Nullspaces and systems of equations

Determinants

Eigenvalues and eigenspaces

Construction of vector spaces

For `K` a field, `m`, `n` integers, `V`, `W` vector spaces over `K`:

- `VectorSpace(K, n);`
- `KMatrixSpace(K, m, n);`
- `Hom(V, W);`

```
Q := RationalField();
V := VectorSpace(Q, 3);
W := VectorSpace(Q, 4);
Hom(V, W);
> Full KMatrixSpace of 3 by 4 matrices over Rational
    Field
```

These vector spaces are created w.r.t. the standard bases.

Construction of vectors

- `V ! Q` with `Q` a sequence;
- `V ! 0` for the zero vector;
- `CharacteristicVector(V, S)`.

```
K := GF(7);
V := VectorSpace(K, 5);
v := V![1, -1, 0, 1, -1];
v;
> (1 6 0 1 6)

w := V![2*i: i in [1..5]];
w;
> (2 4 6 1 3)

u := CharacteristicVector(V, [1, 3]);
u;
> (1 0 1 0 0)
```

Bases

Choose your own basis with `VectorSpaceWithBasis(Q)` where `Q` is a sequence of vectors.

```
Q := RationalField();
Q3 := VectorSpace(Q, 3);
e1 := Q3![1, 2, 0]; e2 := Q3![1, 0, 1];
e3 := Q3![0, 1, 1];
V := VectorSpaceWithBasis([e1, e2, e3]);
BasisMatrix(V);
> [1 2 0]
  [1 0 1]
  [0 1 1]

Coordinates(Q3, e1);
> [ 1, 2, 0 ]

Coordinates(V, e1);
> [ 1, 0, 0 ]
```

Subspaces

```
Q := RationalField();
Q5 := VectorSpace(Q, 5);
S, f := sub<Q5 | [0, 0, 1, 0, 0], [1, 0, 0, 0, 1]>;
f;
> Mapping from: ModTupFld: S to ModTupFld: Q5

ExtendBasis(S, Q5);
> [
      (1 0 0 0 1),
      (0 0 1 0 0),
      (0 1 0 0 0),
      (0 0 0 1 0),
      (0 0 0 0 1)
  ]
```

Quotients

```
Q := RationalField(); Q5 := VectorSpace(Q, 5);
U, f := quo<Q5 | [0, 0, 1, 0, 0], [1, 0, 0, 0, 1]>;
U;
> Full Vector space of degree 3 over Rational Field

f;
> Mapping from: ModTupFld: Q5 to ModTupFld: U
```

Norms and inner products

```
Q := RationalField(); Q3 := VectorSpace(Q, 3);
Q33 := KMatrixSpace(Q, 3, 3);
A := Q33![2, 0, -1, 0, 1, 0, -1, 0, 1];
V := VectorSpace(Q, 3, A);

u1 := Q3![1, 0, -1]; u2 := V![1, 0, -1];
v1 := Q3![3, 2, 1]; v2 := V![3, 2, 1];

InnerProduct(u1, v1);
> 2

InnerProduct(u2, v2);
> 7

Norm(u1);
> 2

Norm(u2);
> 5
```

Construction of matrices over rings

`Matrix(R, m, n, Q)`

with `R` a ring, `m`, `n` integers and `Q` either

- a sequence of length `mn`;
- `m` sequences or vectors of length `n`;
- a sequence of tuples `<i, j, x>`.

```
A := Matrix(IntegerRing(), 3, 4,
    [1,9,7,4,8,3,0,3,7,2,5,6]);
A;
> [1 9 7 4]
  [8 3 0 3]
  [7 2 5 6]
```

Construction of matrices over rings

```
A := Matrix([[1,9,7,-4],[-8,3,0,3],[7,-2,5,6]]);
A;
> [ 1  9  7 -4]
  [-8  3  0  3]
  [ 7 -2  5  6]

BaseRing(A);
> Integer Ring
```

```
A := Matrix(GF(3), 4, 4, [<i,j,i+j>: i,j in [1..4]]);
A;
> [2 0 1 2]
  [0 1 2 0]
  [1 2 0 1]
  [2 0 1 2]
```

Construction of special types

Certain types of matrices can be constructed directly: zero matrices, identities, scalar, diagonal, triangular, symmetric, ...

```
A := LowerTriangularMatrix([1..6]);
A;
> [1 0 0]
  [2 3 0]
  [4 5 6]
```

```
A := PermutationMatrix(IntegerRing(), [1,5,3,2,4]);
A;
> [1 0 0 0 0]
  [0 0 0 0 1]
  [0 0 1 0 0]
  [0 1 0 0 0]
  [0 0 0 1 0]
```

Block matrices

`BlockMatrix(m,n,Q)`

with `m,n` the dimensions and `Q` a sequence of blocks or rows of blocks.
Blocks can also be joined using `HorizontalJoin`, `VerticalJoin` and `DiagonalJoin` or multiplied with `KroneckerProduct`.

```
A := Matrix([[1, 2], [1, 0]]);
B := Matrix([[3, 1], [0, 2]]);
C := BlockMatrix(2, 2, [[A, B], [B, A]]);
C;
> [1 2 3 1]
  [1 0 0 2]
  [3 1 1 2]
  [0 2 1 0]
```

Accessing invariants and predicates

For a vector space `V`, vector `v` and matrix `A`:

- `Nrows(A)`, `Ncols(A)` or `Ncols(v)`;
- `BaseRing(A)` or `BaseField(V)`;
- `Eltseq(A)` or `Eltseq(v)`;
- `Degree(V)`;
- `Dimension(V)`;
- `Generators(V)`.

`IsZero(A)` or `IsZero(v)` to test whether `A` or `v` is the zero matrix or vector.
Similarly for matrices: `IsOne`, `IsScalar`, `IsDiagonal`, `IsSymmetric`, `IsUpperTriangular`, `IsUnit`, ...

Arithmetic

For `A`, `B` matrices, `u`, `v` vectors and `x` a scalar:

- `A + B`, `u + v`;
- `A - B`, `u - v`;
- `x * A`, `A * x`;
- `x * v`, `v / x`;
- `Transpose(A)`.

- `A * B`;
- `A * v`, `v * A`;
- `A ^ -1`;
- `A ^ n`;

```
K := GF(7);
V := VectorSpace(K, 4);
v := V![1, 2, 3, 4];
A := Matrix(K, 4, 2,
    [<i, j, 2*i+j>: i in [1..4], j in [1, 2]]);
v*A;
> (0 3)
```

Accessing and modifying entries

To access/modify the `i`-th row of `A`:

```
A[i];                    A[i] := u;
```

To access/modify the element in position `(i,j)` of `A` or position `i` of `v`:

```
A[i,j];  v[i];           A[i,j] := x;  v[i] := x;
```

To access multiple rows at once:

```
A[i..j];                 A[Q];
```

with `Q` a sequence of row numbers.

Accessing and modifying entries

```
A := Matrix([[1, 7, -3], [4,-2, 0], [-1, 3, 4], [-5,
    2, 0]]);
A[1,3];
> -3

A[[1,3]];
> [
       ( 1  7 -3),
       (-1  3  4)
  ]

A[2] := Vector([1,2,3]);
A;
> [ 1  7 -3]
  [ 1  2  3]
  [-1  3  4]
  [-5  2  0]
```

Extracting blocks

- `Submatrix(A,i,j,p,q)`: extracts the `p x q`-submatrix of `A` starting from position `(i,j)`;
- `Submatrix(A,I,J)`: extracts the submatrix of elements in position `(i,j)` with `i` in `I` and `j` in `J`;
- `SubmatrixRange(A,i,j,r,s)`: extracts the submatrix from `(i,j)` to `(r,s)`.

```
A := Matrix(4, 4, [1..16]);
Submatrix(A, 2, 2, 3, 3);
> [ 6  7  8]
  [10 11 12]
  [14 15 16]

SubmatrixRange(A, 2, 2, 3, 3);
> [ 6  7]
  [10 11]
```

Inserting blocks

In general: adding ~ in front of the matrix argument changes the matrix, without ~ the function creates a new matrix!
`InsertBlock("~"A,B,i,j)` inserts the block `B` in `A` at position `(i,j)`.

```
A := Matrix(3,3,[0..8]);
A;
> [0 1 2]
  [3 4 5]
  [6 7 8]

B := Matrix(2,2,[1..4]);
InsertBlock(~A,B,2,2);
A;
> [0 1 2]
  [3 1 2]
  [6 3 4]
```

Row and column operations

- `SwapRows("~"A,i,j);`
- `ReverseRows("~"A);`
- `AddRow("~"A,c,i,j)`: adds `c*A[i]` to `A[j]`;
- `Multiply("~"A,c,i)`: multiplies `A[i]` with `c`;
- `RemoveRow("~"A,i);`

Equivalent functions exist for columns!

```
A := DiagonalMatrix([1..5]);
SwapColumns(~A, 1, 5);
A;
> [0 0 0 0 1]
  [0 2 0 0 0]
  [0 0 3 0 0]
  [0 0 0 4 0]
  [5 0 0 0 0]
```

Row and column operations

```
AddRow(~A,3,2,4);
A;
> [0  0  0  0  1]
  [0  2  0  0  0]
  [0  0  3  0  0]
  [0  6  0  4  0]
  [5  0  0  0  0]

RemoveRowColumn(~A,3,3);
A;
> [0  0  0  1]
  [0  2  0  0]
  [0  6  4  0]
  [5  0  0  0]
```

Changing the coefficient ring

For `A` a matrix over a ring `S`, use either `Matrix(R,A)` or `ChangeRing(A,R)` to change the coefficient ring to `R`.

```
A := Matrix([[2, 1], [0, -2]]);
IsUnit(A);
> false

BaseRing(A);
> Integer Ring

B := ChangeRing(A, RationalField());
B^-1;
> [ 1/2  1/4]
  [   0 -1/2]
```

Changing the coefficient field

For `V` a vector space over a field `K`, `L` a field extension and `F` a subfield of `K`:

- `ExtendField(V, L);`
- `RestrictField(V, F);`
- `VectorSpace(V, F).`

```
K := GF(9);
F := GF(3);
V := VectorSpace(K, 4);
U := RestrictField(V, F);
U;
> Full Vector space of degree 4 over GF(3)

W := VectorSpace(V, F);
W;
> Full Vector space of degree 8 over GF(3)
```

Nullspaces

- `Nullspace(A)`: returns the R-space of all vectors `v` such that `v * A = 0`;
- `NullspaceOfTranspose(A)`: returns the R-space of all vectors `v` such that `A * v = 0`;
- `NullspaceMatrix(A)`: returns a basis matrix of the nullspace of `A`.

```
A := Matrix([[1, -3, 2], [0, -1, 1], [1, -2, 1]]);
NullspaceMatrix(A);
> [ 1 -1 -1]

NullspaceOfTranspose(A);
> RSpace of degree 3, dimension 1 over Integer Ring
  Echelonized basis:
  (1 1 1)
```

Systems of solutions

`Solution(A,W)`: for `W` a vector, matrix or a sequence of vectors, it returns a solution `V` such that `V * A = W` and the nullspace of the solution.

```
A := Matrix([[1,-2,1], [3,-6,-1], [-2,4,2]]);
W := Vector([-3,6,5]);
V, N := Solution(A,W);
V;
> (0 1 3)

N;
> RSpace of degree 3, dimension 1 over Integer Ring
  Echelonized basis:
  ( 1 -1 -1)
```

Determinant calculations

- `Determinant(A);`
- `Trace(A);`
- `Rank(A);`
- `Minor(A, I, J)`: returns the minor of `(i,j)` for all `i` in `I` and `j` in `J`.
  `Minors(A, r)`: returns a sequence of all the `r x r` minors of `A`;
- `Cofactor(A, I, J)`, `Cofactors(A, r)`.

Determinant calculations

```
A := Matrix(3, 3, [-4..4]);
A;
> [-4 -3 -2]
  [-1  0  1]
  [ 2  3  4]

Cofactors(A,2);
> [ -3, 6, -3, 6, -12, 6, -3, 6, -3 ]

Rank(A);
> 2

Determinant(A);
> 0
```

Determinant calculations with parameters

```
Q := RationalField();
P<x,y> := PolynomialAlgebra(Q, 2);
A := Matrix(P, [[1, x, -1], [1, y, 1], [0, 0, x*y]]);
Tr := Trace(A);
Tr;
> x*y + y + 1

det := Determinant(A);
det;
> -x^2*y + x*y^2

Evaluate(Tr, [2, -2]);
> -5

Evaluate(det, [2, -2]);
> 16
```

Determinant calculations with parameters

```
P5<x,y> := PolynomialAlgebra(GF(5), 2);
B := ChangeRing(A, P5);
B;
> [   1    x    4]
  [   1    y    1]
  [   0    0 x*y]

detB := Determinant(B);
Evaluate(detB, [2, -2]);
> 1
```

Minimal and characteristic polynomials

```
A := DiagonalMatrix(GF(7), [1, 2, 1]);
P<x> := PolynomialRing(GF(7));
MinimalPolynomial(A);
> x^2 + 4*x + 2

FactoredCharacteristicPolynomial(A);
> [
      <x + 5, 1>,
      <x + 6, 2>
  ]
```

Similarly, there is `CharacteristicPolynomial` and `FactoredMinimalPolynomial`.

Eigenvalues and eigenspaces

```
M := Matrix(RationalField(), [[1/2,-1/4,1/4,1/4],
    [1,-1/2,-1/2,1/2], [2,-1,0,1], [0,0,0,0]]);
Eigenvalues(M);
> { <1, 1>, <0, 2>, <-1, 1> }

Eigenspace(M, 1);
> Vector space of degree 4, dimension 1 over Rational
    Field
  Echelonized basis:
  (   1 -1/2  1/2  1/2)

Eigenspace(M, 0);
> Vector space of degree 4, dimension 2 over Rational
    Field
  Echelonized basis:
  (   1  1/2 -1/2    0)
  (   0    0    0    1)
```

Eigenvalues and eigenspaces

```
A := Matrix([[-1, -1], [1, -1]]);
Eigenvalues(A);
> {}

C<i> := ComplexField(3);
B := ChangeRing(A, C);
Eigenvalues(B);
> [ <-1.00 - 1.00*i, 1>, <-1.00 + 0.999*i, 1> ]

D := ChangeRing(A, GF(13));
Eigenvalues(D);
> { <4, 1>, <7, 1> }
```