A new programmer's interface for vectors and matrices

Max Neunhöffer

University of St Andrews

11.9.2007

What is a vector? What is a matrix?

What is a vector? What is a matrix?

Up to now in GAP, they are just lists:

```
gap> v := [1,2,3];
[ 1, 2, 3 ]
gap> m := [[0,1],[1,0]];
[ [ 0, 1 ], [ 1, 0 ] ]
```

What is a vector? What is a matrix?

Up to now in GAP, they are just lists:

```
gap> v := [1,2,3];
[ 1, 2, 3 ]
gap> m := [[0,1],[1,0]];
[ [ 0, 1 ], [ 1, 0 ] ]
```

However, there are different representations:

```
gap> m := m*Z(2);;
gap> for r in m do ConvertToVectorRep(r,2);od;
gap> m;
[ <a GF2 vector of length 2>,
  <a GF2 vector of length 2> ]
gap> ConvertToMatrixRep(m,2);;
gap> m;
<a 2x2 matrix over GF2>
```

What is a vector? What is a matrix?

Up to now in GAP, they are just lists:

```
gap> v := [1,2,3];
[ 1, 2, 3 ]
gap> m := [[0,1],[1,0]];
[ [ 0, 1 ], [ 1, 0 ] ]
```

However, there are different representations:

```
gap> m := m*Z(2);;
gap> for r in m do ConvertToVectorRep(r,2);od;
gap> m;
[ <a GF2 vector of length 2>,
  <a GF2 vector of length 2> ]
gap> ConvertToMatrixRep(m,2);;
gap> m;
<a 2x2 matrix over GF2>
```

We can use the method selection only for the last matrix!

Method selection problems

```
gap> h:=[1..100];;
gap> m:=List([1..100000],i->Z(2)*[1..1000]);;
gap> TypeObj(m);; time;
1908
```

Method selection problems

```
gap> h:=[1..100];;
gap> m:=List([1..100000],i->Z(2)*[1..1000]);;
gap> TypeObj(m);; time;
1908
gap> TypeObj(m);; time;
16
```

Method selection problems

```
gap> h:=[1..100];;
gap> m:=List([1..100000],i->Z(2)*[1..1000]);;
gap> TypeObj(m);; time;
1908
gap> TypeObj(m);; time;
16
gap> for i in h do Reversed(m); od; time;
24
gap> for i in h do ReversedOp(m); od; time;
2888
```

Method selection problems

```
gap> h:=[1..100];;
gap> m:=List([1..100000],i->Z(2)*[1..1000]);;
gap> TypeObj(m);; time;
1908
gap> TypeObj(m);; time;
16
gap> for i in h do Reversed(m); od; time;
24
gap> for i in h do ReversedOp(m); od; time;
2888
gap> ConvertToMatrixRep(m,2);;
gap> TypeObj(m);; time;
0
gap> for i in h do TypeObj(m); od; time;
0
```

Method selection problems

```
gap> h:=[1..100];;
gap> m:=List([1..100000],i->Z(2)*[1..1000]);;
gap> TypeObj(m);; time;
1908
gap> TypeObj(m);; time;
16
gap> for i in h do Reversed(m); od; time;
24
gap> for i in h do ReversedOp(m); od; time;
2888
gap> ConvertToMatrixRep(m,2);;
gap> TypeObj(m);; time;
0
gap> for i in h do TypeObj(m); od; time;
0
```

Type computation and method selection for
mutable plain lists
can take a significant amount of time!

New filters

Solution: Wrap 'em up.

New filters

Solution: Wrap 'em up. Define an interface to them.

New filters

Solution: Wrap 'em up. Define an interface to them.

```
DeclareCategory("IsRowVectorObj",
      IsVector and IsCopyable);

DeclareCategory("IsMatrixObj",
      IsVector and IsScalar and IsCopyable);
```

New filters

Solution: Wrap 'em up. Define an interface to them.

```
DeclareCategory("IsRowVectorObj",
      IsVector and IsCopyable);

DeclareCategory("IsMatrixObj",
      IsVector and IsScalar and IsCopyable);
```

Vectors and matrices are no longer necessarily lists.

New filters

Solution: Wrap 'em up. Define an interface to them.

```
DeclareCategory("IsRowVectorObj",
       IsVector and IsCopyable);

DeclareCategory("IsMatrixObj",
       IsVector and IsScalar and IsCopyable);
```

Vectors and matrices are no longer necessarily lists.

```
DeclareCategory("IsRowListMatrix",
                IsMatrixObj);
DeclareCategory("IsFlatMatrix",IsMatrixObj);
```

These two types of matrices are not only different representations, they also behave differently.

"Row list" vs. "flat" matrices

A row list matrix

- behaves like a list of row objects and
- has individual GAP objects as rows,

"Row list" vs. "flat" matrices

A row list matrix

- behaves like a list of row objects and
- has individual GAP objects as rows,
- is like a list that insists on being dense and containing only row objects of the right type and size.

"Row list" vs. "flat" matrices

A row list matrix

- behaves like a list of row objects and
- has individual GAP objects as rows,
- is like a list that insists on being dense and containing only row objects of the right type and size.

A flat matrix

- consists of a single GAP object,

"Row list" vs. "flat" matrices

A row list matrix

- behaves like a list of row objects and
- has individual GAP objects as rows,
- is like a list that insists on being dense and containing only row objects of the right type and size.

A flat matrix

- consists of a single GAP object,
- the rows are part of this object, not individual objects,

"Row list" vs. "flat" matrices

A row list matrix

- behaves like a list of row objects and
- has individual GAP objects as rows,
- is like a list that insists on being dense and containing only row objects of the right type and size.

A flat matrix

- consists of a single GAP object,
- the rows are part of this object, not individual objects,
- has to copy rows to exchange or permute them.

"Row list" vs. "flat" matrices

A row list matrix

- behaves like a list of row objects and
- has individual GAP objects as rows,
- is like a list that insists on being dense and containing only row objects of the right type and size.

A flat matrix

- consists of a single GAP object,
- the rows are part of this object, not individual objects,
- has to copy rows to exchange or permute them.

All matrices

- know their base domain,

"Row list" vs. "flat" matrices

A row list matrix

- behaves like a list of row objects and
- has individual GAP objects as rows,
- is like a list that insists on being dense and containing only row objects of the right type and size.

A flat matrix

- consists of a single GAP object,
- the rows are part of this object, not individual objects,
- has to copy rows to exchange or permute them.

All matrices

- know their base domain,
- know their dimensions, and

"Row list" vs. "flat" matrices

A row list matrix

- behaves like a list of row objects and
- has individual GAP objects as rows,
- is like a list that insists on being dense and containing only row objects of the right type and size.

A flat matrix

- consists of a single GAP object,
- the rows are part of this object, not individual objects,
- has to copy rows to exchange or permute them.

All matrices

- know their base domain,
- know their dimensions, and
- can have 0 rows or 0 columns.

Operations

Operations

Attributes for vectors:

`BaseDomain`, `Length`.

Operations

Attributes for vectors:

`BaseDomain`, `Length`.

Attributes for matrices:

`BaseDomain`, `Length`, `RowLength`, `DimensionsMat`.

Operations

Attributes for vectors:

`BaseDomain`, `Length`.

Attributes for matrices:

`BaseDomain`, `Length`, `RowLength`, `DimensionsMat`.

Lots of operations are defined (see below).

Operations

Attributes for vectors:

`BaseDomain`, `Length`.

Attributes for matrices:

`BaseDomain`, `Length`, `RowLength`, `DimensionsMat`.

Lots of operations are defined (see below).

Important:

Objects and derived objects keep their representation!

Generic code does not have to worry about this!

Operations

Attributes for vectors:

`BaseDomain`, `Length`.

Attributes for matrices:

`BaseDomain`, `Length`, `RowLength`, `DimensionsMat`.

Lots of operations are defined (see below).

Important:

Objects and derived objects keep their representation!

Generic code does not have to worry about this!

```
gap> Display(m);
 1 . 1
 . 1 .
gap> ExtractSubMatrix(m,[2,1],[1,3]);
<a 2x2 matrix over GF2>
gap> Display(last);
 . .
 1 1
```

Constructing new vectors and matrices

```
gap> v := NewRowVector(IsPlistVectorRep,
                       Rationals,[1,2,3]);
<plist vector over Rationals of length 3>
gap> m := NewMatrix(IsPlistMatrixRep,
                    Rationals,3,[[4,5,6]]);
<1x3-matrix over Rationals>
gap> Add(m,v);
```

Constructing new vectors and matrices

```
gap> v := NewRowVector(IsPlistVectorRep,
                                Rationals,[1,2,3]);
<plist vector over Rationals of length 3>
gap> m := NewMatrix(IsPlistMatrixRep,
                             Rationals,3,[[4,5,6]]);
<1x3-matrix over Rationals>
gap> Add(m,v);
```

This uses GAP's constructors.

Constructing new vectors and matrices

```
gap> v := NewRowVector(IsPlistVectorRep,
                       Rationals,[1,2,3]);
<plist vector over Rationals of length 3>
gap> m := NewMatrix(IsPlistMatrixRep,
                    Rationals,3,[[4,5,6]]);
<1x3-matrix over Rationals>
gap> Add(m,v);
```

This uses GAP's constructors.

A constructor is an operation, for which the method selection works differently in the first argument:
The argument is a filter, and a method must be installed for a subfilter to be taken.

Constructing new vectors and matrices

```
gap> v := NewRowVector(IsPlistVectorRep,
                       Rationals,[1,2,3]);
<plist vector over Rationals of length 3>
gap> m := NewMatrix(IsPlistMatrixRep,
                    Rationals,3,[[4,5,6]]);
<1x3-matrix over Rationals>
gap> Add(m,v);
```

This uses GAP's constructors.

A constructor is an operation, for which the method selection works differently in the first argument: The argument is a filter, and a method must be installed for a subfilter to be taken.

Packages can have constructor methods for new types.

GAP's constructors explained

```
DeclareCategory("IsA",IsComponentObjectRep);
DeclareConstructor("MakeA",[IsA,IsInt]);
tA := NewType(CyclotomicsFamily,IsA);;
InstallMethod(MakeA,[IsA,IsInt],
   function(f,x)
     return Objectify(tA,rec(x := x));
   end);
```

GAP's constructors explained

```
DeclareCategory("IsA",IsComponentObjectRep);
DeclareConstructor("MakeA",[IsA,IsInt]);
tA := NewType(CyclotomicsFamily,IsA);;
InstallMethod(MakeA,[IsA,IsInt],
   function(f,x)
     return Objectify(tA,rec(x := x));
   end);
DeclareCategory("IsAB",IsA);
tAB := NewType(CyclotomicsFamily,IsAB);;
InstallMethod(MakeA,[IsAB,IsInt],
   function(f,x)
     return Objectify(tAB,rec(x := x));
   end);
```

GAP's constructors explained

```
DeclareCategory("IsA",IsComponentObjectRep);
DeclareConstructor("MakeA",[IsA,IsInt]);
tA := NewType(CyclotomicsFamily,IsA);;
InstallMethod(MakeA,[IsA,IsInt],
   function(f,x)
     return Objectify(tA,rec(x := x));
   end);
```

```
DeclareCategory("IsAB",IsA);
tAB := NewType(CyclotomicsFamily,IsAB);;
InstallMethod(MakeA,[IsAB,IsInt],
   function(f,x)
     return Objectify(tAB,rec(x := x));
   end);
```

```
gap> a := MakeA(IsA,17);;
gap> [ IsA(a), IsAB(a) ];
[ true, false ]
gap> b := MakeA(IsAB,17);;
gap> [ IsA(b), IsAB(b) ];
[ true, true ]
```

Preserving the representation

Preserving the representation

```
gap> ConstructingFilter(m);
<Operation "IsPlistMatrixRep">
```

Preserving the representation

```
gap> ConstructingFilter(m);
<Operation "IsPlistMatrixRep">
```

Derived objects:

`ZeroMutable`, `ShallowCopy`, `OneImmutable`, `MutableCopyMat`, ...

Preserving the representation

```
gap> ConstructingFilter(m);
<Operation "IsPlistMatrixRep">
```

Derived objects:

`ZeroMutable`, `ShallowCopy`, `OneImmutable`, `MutableCopyMat`, ...

New objects in same representation:

```
gap> v := NewRowVector(IsPlistVectorRep,
             Rationals,[1,2,3]);;
gap> m := NewMatrix(IsPlistMatrixRep,
             Rationals,3,[[4,5,6]]);;
gap> ZeroVector(10,v);
<plist vector over Rationals of length 10>
gap> Vector([6,7,8,9],m);
<plist vector over Rationals of length 4>
gap> IdentityMatrix(12,m);
<12x12-matrix over Rationals>
gap> n := Matrix([],3,m);
<0x3-matrix over Rationals>
```

Flat vs. row list matrices

Objects in the filter `IsRowListMatrix`

- have most list operations: `Add`, `Remove`, `IsBound`, `Unbind`, `[]`, `[]:=`, `{}`, `{}:=`, `Append`, `ShallowCopy`, `List`,

Flat vs. row list matrices

Objects in the filter `IsRowListMatrix`

- have most list operations: `Add`, `Remove`, `IsBound`, `Unbind`, `[]`, `[]:=`, `{}`, `{}:=`, `Append`, `ShallowCopy`, `List`,
- they simply insist on being dense and on containing only vectors of the right length and type.

Flat vs. row list matrices

Objects in the filter `IsRowListMatrix`

- have most list operations: `Add`, `Remove`, `IsBound`, `Unbind`, `[]`, `[]:=`, `{}`, `{}:=`, `Append`, `ShallowCopy`, `List`,
- they simply insist on being dense and on containing only vectors of the right length and type.

Objects in the filter `IsFlatMatrix`

- have `[]`, which creates a reference,

Flat vs. row list matrices

Objects in the filter `IsRowListMatrix`

- have most list operations: `Add`, `Remove`, `IsBound`, `Unbind`, `[]`, `[]:=`, `{}`, `{}:=`, `Append`, `ShallowCopy`, `List`,
- they simply insist on being dense and on containing only vectors of the right length and type.

Objects in the filter `IsFlatMatrix`

- have `[]`, which creates a reference,
- `[]:=`, `{}`, `{}:=`, which copy data, and

Flat vs. row list matrices

Objects in the filter `IsRowListMatrix`

- have most list operations: `Add`, `Remove`, `IsBound`, `Unbind`, `[]`, `[]:=`, `{}`, `{}:=`, `Append`, `ShallowCopy`, `List`,
- they simply insist on being dense and on containing only vectors of the right length and type.

Objects in the filter `IsFlatMatrix`

- have `[]`, which creates a reference,
- `[]:=`, `{}`, `{}:=`, which copy data, and
- do not support `Add`, `Remove`, `IsBound`, `Unbind`, `Append`.

Flat vs. row list matrices

Objects in the filter `IsRowListMatrix`

- have most list operations: `Add`, `Remove`, `IsBound`, `Unbind`, `[]`, `[]:=`, `{}`, `{}:=`, `Append`, `ShallowCopy`, `List`,
- they simply insist on being dense and on containing only vectors of the right length and type.

Objects in the filter `IsFlatMatrix`

- have `[]`, which creates a reference,
- `[]:=`, `{}`, `{}:=`, which copy data, and
- do not support `Add`, `Remove`, `IsBound`, `Unbind`, `Append`.
- `ShallowCopy` is a full copy.

Creating a companion matrix

```
cm := function(p,mat)
  local bd,one,l,n,ll,i;
  bd := BaseDomain(mat);  one := One(bd);
  l := CoefficientsOfUnivariatePolynomial(p);
  n := Length(l)-1;
  l := Vector(-l{[1..n]},mat);
  ll := ListWithIdenticalEntries(n,0);
  ll[n] := l;
  for i in [1..n-1] do
    ll[i] := ZeroMutable(l);
    ll[i][i+1] := one;
  od;
  return Matrix(ll,n,mat);
end;
```

Creating a companion matrix

```
cm := function(p,mat)
  local bd,one,l,n,ll,i;
  bd := BaseDomain(mat);  one := One(bd);
  l := CoefficientsOfUnivariatePolynomial(p);
  n := Length(l)-1;
  l := Vector(-l{[1..n]},mat);
  ll := ListWithIdenticalEntries(n,0);
  ll[n] := l;
  for i in [1..n-1] do
    ll[i] := ZeroMutable(l);
    ll[i][i+1] := one;
  od;
  return Matrix(ll,n,mat);
end;
gap> x:=X(Rationals);;
gap> Display(cm(x^3-2*x^2-5,m));
<3x3-matrix over Rationals:
[[ 0, 1, 0 ]
 [ 0, 0, 1 ]
 [ 5, 0, 2 ]]>
```