

complex

**I wear a chain complex now. Chain
complexes are cool**

27.01.2017

27 January 2017

Kristin Krogh Arnesen

Oystein Skartsaeterhagen

Kamal Saleh

Kristin Krogh Arnesen

Email: kristink@math.ntnu.no

Homepage: <http://www.math.ntnu.no/~kristink>

Address: Trondheim

Oystein Skartsaeterhagen

Email: oysteini@math.ntnu.no

Homepage: <http://www.math.ntnu.no/~oysteini>

Address: Trondheim

Kamal Saleh

Email: kamal.saleh@uni-siegen.de

Homepage: <https://github.com/kamalsaleh/complex>

Address: Siegen

Contents

1	Complexes categories	3
1.1	Constructing chain and cochain categories	3
2	Complexes	5
2.1	Categories and filters	5
2.2	Creating chain and cochain complexes	7
2.3	Attributes	8
2.4	Operations	9
2.5	Truncations	11
2.6	Examples	13
3	Complexes morphisms	15
3.1	Categories and filters	15
3.2	Creating chain and cochain morphisms	17
3.3	Attributes	18
3.4	Properties	18
3.5	Operations	18
3.6	Examples	20
4	Functors	22
4.1	Basic functors for complex categories.	22
4.2	Examples	23
	Index	26

Chapter 1

Complexes categories

1.1 Constructing chain and cochain categories

1.1.1 IsChainOrCochainComplexCategory (for IsCapCategory)

▷ IsChainOrCochainComplexCategory(*arg*) (filter)
Returns: true or false
bla bla

1.1.2 IsChainComplexCategory (for IsChainOrCochainComplexCategory)

▷ IsChainComplexCategory(*arg*) (filter)
Returns: true or false
bla bla

1.1.3 IsCochainComplexCategory (for IsChainOrCochainComplexCategory)

▷ IsCochainComplexCategory(*arg*) (filter)
Returns: true or false
bla bla

1.1.4 ChainComplexCategory (for IsCapCategory)

▷ ChainComplexCategory(*A*) (attribute)
Returns: a CAP category
Creates the chain complex category $Ch(A)$ an Abelian category A .

1.1.5 CochainComplexCategory (for IsCapCategory)

▷ CochainComplexCategory(*A*) (attribute)
Returns: a CAP category
Creates the cochain complex category $Coch(A)$ an Abelian category A .

1.1.6 UnderlyingCategory (for IsChainOrCochainComplexCategory)

▷ UnderlyingCategory(B)

(attribute)

Returns: a CAP category

The input is a chain or cochain complex category $B=C(A)$ constructed by one of the previous commands. The output is A .

Let \mathbb{Q} be the field of rationals and let $\text{Vec}_{\mathbb{Q}}$ be the category of \mathbb{Q} -vector spaces. The cochain complex category of $\text{Vec}_{\mathbb{Q}}$ can be constructed as follows

Example

```
gap> LoadPackage( "LinearAlgebraForCap" );;
gap> LoadPackage( "complex" );;
gap> Q := HomalgFieldOfRationals( );;
gap> matrix_category := MatrixCategory( Q );
Category of matrices over Q
gap> cochain_cat := CochainComplexCategory( matrix_category );
Cochain complexes category over category of matrices over Q
```

Chapter 2

Complexes

2.1 Categories and filters

2.1.1 IsChainOrCochainComplex (for IsCapCategoryObject)

▷ IsChainOrCochainComplex(*arg*) (filter)
Returns: true or false
bla bla

2.1.2 IsChainComplex (for IsChainOrCochainComplex)

▷ IsChainComplex(*arg*) (filter)
Returns: true or false
bla bla

2.1.3 IsCochainComplex (for IsChainOrCochainComplex)

▷ IsCochainComplex(*arg*) (filter)
Returns: true or false
bla bla

2.1.4 IsBoundedBelowChainOrCochainComplex (for IsChainOrCochainComplex)

▷ IsBoundedBelowChainOrCochainComplex(*arg*) (filter)
Returns: true or false
bla bla

2.1.5 IsBoundedAboveChainOrCochainComplex (for IsChainOrCochainComplex)

▷ IsBoundedAboveChainOrCochainComplex(*arg*) (filter)
Returns: true or false
bla bla

2.1.6 **IsBoundedChainOrCochainComplex** (for **IsBoundedBelowChainOrCochainComplex** and **IsBoundedAboveChainOrCochainComplex**)

▷ **IsBoundedChainOrCochainComplex**(*arg*) (filter)
Returns: true or false
 bla bla

2.1.7 **IsBoundedBelowChainComplex** (for **IsBoundedBelowChainOrCochainComplex** and **IsChainComplex**)

▷ **IsBoundedBelowChainComplex**(*arg*) (filter)
Returns: true or false
 bla bla

2.1.8 **IsBoundedBelowCochainComplex** (for **IsBoundedBelowChainOrCochainComplex** and **IsCochainComplex**)

▷ **IsBoundedBelowCochainComplex**(*arg*) (filter)
Returns: true or false
 bla bla

2.1.9 **IsBoundedAboveChainComplex** (for **IsBoundedAboveChainOrCochainComplex** and **IsChainComplex**)

▷ **IsBoundedAboveChainComplex**(*arg*) (filter)
Returns: true or false
 bla bla

2.1.10 **IsBoundedAboveCochainComplex** (for **IsBoundedAboveChainOrCochainComplex** and **IsCochainComplex**)

▷ **IsBoundedAboveCochainComplex**(*arg*) (filter)
Returns: true or false
 bla bla

2.1.11 **IsBoundedChainComplex** (for **IsBoundedChainOrCochainComplex** and **IsChainComplex**)

▷ **IsBoundedChainComplex**(*arg*) (filter)
Returns: true or false
 bla bla

2.1.12 **IsBoundedCochainComplex** (for **IsBoundedChainOrCochainComplex** and **IsCochainComplex**)

▷ **IsBoundedCochainComplex**(*arg*) (filter)
Returns: true or false
 bla bla

2.2 Creating chain and cochain complexes

2.2.1 ChainComplex (for IsCapCategory, IsZList)

- ▷ ChainComplex(A, diffs) (operation)
- ▷ CochainComplex(A, diffs) (operation)

Returns: a chain complex

The input is category A and an infinite list diffs . The output is the chain (resp. cochain) complex $M_\bullet \in \text{Ch}(A)$ ($M^\bullet \in \text{CoCh}(A)$) where $d_i^M = \text{diffs}[i]$ ($d_M^i = \text{diffs}[i]$).

2.2.2 ChainComplex (for IsDenseList, IsInt)

- ▷ ChainComplex(diffs, n) (operation)
- ▷ CochainComplex(diffs, n) (operation)

Returns: a (co)chain complex

The input is a finite dense list diffs and an integer n . The output is the chain (resp. cochain) complex $M_\bullet \in \text{Ch}(A)$ ($M^\bullet \in \text{CoCh}(A)$) where $d_n^M := \text{diffs}[1]$ ($d_M^n := \text{diffs}[1]$), $d_{n+1}^M = \text{diffs}[2]$ ($d_M^{n+1} := \text{diffs}[2]$), etc.

2.2.3 ChainComplex (for IsDenseList)

- ▷ ChainComplex(diffs) (operation)
- ▷ CochainComplex(diffs) (operation)

Returns: a (co)chain complex

The same as the previous operations but with $n = 0$.

2.2.4 StalkChainComplex (for IsCapCategoryObject, IsInt)

- ▷ StalkChainComplex(diffs, n) (operation)
- ▷ StalkCochainComplex(diffs, n) (operation)

Returns: a (co)chain complex

The input is an object $M \in A$. The output is chain (resp. cochain) complex $M_\bullet \in \text{Ch}(A)$ ($M^\bullet \in \text{CoCh}(A)$) where $M_n = M$ ($M^n = M$) and $M_i = 0$ ($M^i = 0$) whenever $i \neq n$.

2.2.5 ChainComplexWithInductiveSides (for IsCapCategoryMorphism, IsFunction, IsFunction)

- ▷ ChainComplexWithInductiveSides(d, G, F) (operation)

Returns: a chain complex

The input is a morphism $d \in A$ and two functions F, G . The output is chain complex $M_\bullet \in \text{Ch}(A)$ where $d_0^M = d$ and $d_i^M = G^i(d)$ for all $i \leq -1$ and $d_i^M = F^i(d)$ for all $i \geq 1$.

2.2.6 CochainComplexWithInductiveSides (for IsCapCategoryMorphism, IsFunction, IsFunction)

- ▷ CochainComplexWithInductiveSides(d, G, F) (operation)

Returns: a cochain complex

The input is a morphism $d \in A$ and two functions F, G . The output is cochain complex $M^\bullet \in \text{CoCh}(A)$ where $d_M^0 = d$ and $d_M^i = G^i(d)$ for all $i \leq -1$ and $d_M^i = F^i(d)$ for all $i \geq 1$.

2.2.7 ChainComplexWithInductiveNegativeSide (for IsCapCategoryMorphism, Is-Function)

▷ ChainComplexWithInductiveNegativeSide(d , G) (operation)

Returns: a chain complex

The input is a morphism $d \in A$ and a functions G . The output is chain complex $M_\bullet \in \text{Ch}(A)$ where $d_0^M = d$ and $d_i^M = G^i(d)$ for all $i \leq -1$ and $d_i^M = 0$ for all $i \geq 1$.

2.2.8 ChainComplexWithInductivePositiveSide (for IsCapCategoryMorphism, Is-Function)

▷ ChainComplexWithInductivePositiveSide(d , F) (operation)

Returns: a chain complex

The input is a morphism $d \in A$ and a functions F . The output is chain complex $M_\bullet \in \text{Ch}(A)$ where $d_0^M = d$ and $d_i^M = F^i(d)$ for all $i \geq 1$ and $d_i^M = 0$ for all $i \leq -1$.

2.2.9 CochainComplexWithInductiveNegativeSide (for IsCapCategoryMorphism, Is-Function)

▷ CochainComplexWithInductiveNegativeSide(d , G) (operation)

Returns: a cochain complex

The input is a morphism $d \in A$ and a functions G . The output is cochain complex $M^\bullet \in \text{CoCh}(A)$ where $d_M^0 = d$ and $d_M^i = G^i(d)$ for all $i \leq -1$ and $d_M^i = 0$ for all $i \geq 1$.

2.2.10 CochainComplexWithInductivePositiveSide (for IsCapCategoryMorphism, Is-Function)

▷ CochainComplexWithInductivePositiveSide(d , F) (operation)

Returns: a cochain complex

The input is a morphism $d \in A$ and a functions F . The output is cochain complex $M^\bullet \in \text{CoCh}(A)$ where $d_M^0 = d$ and $d_M^i = F^i(d)$ for all $i \geq 1$ and $d_M^i = 0$ for all $i \leq -1$.

2.3 Attributes

2.3.1 Differentials (for IsChainOrCochainComplex)

▷ Differentials(C) (attribute)

Returns: an infinite list

The command returns the differentials of the chain or cochain complex as an infinite list.

2.3.2 Objects (for IsChainOrCochainComplex)

▷ Objects(C) (attribute)

Returns: an infinite list

The command returns the objects of the chain or cochain complex as an infinite list.

2.3.3 CatOfComplex (for IsChainOrCochainComplex)

▷ CatOfComplex(C) (attribute)

Returns: a Cap category

The command returns the category in which all objects and differentials of C live.

2.3.4 QuasiIsomorphismFromProjectiveResolution (for IsBoundedAboveCochainComplex)

▷ QuasiIsomorphismFromProjectiveResolution(C) (attribute)

Returns: a cochain morphism

The input is an above bounded cochain complex C^\bullet . The output is a quasi-isomorphism $q : P^\bullet \rightarrow C^\bullet$ such that P^\bullet is upper bounded and all its objects are projective in the underlying abelian category.

2.3.5 QuasiIsomorphismFromProjectiveResolution (for IsBoundedBelowChainComplex)

▷ QuasiIsomorphismFromProjectiveResolution(C) (attribute)

Returns: a chain morphism

The input is an below bounded chain complex C_\bullet . The output is a quasi-isomorphism $q : P_\bullet \rightarrow C_\bullet$ such that P_\bullet is lower bounded and all its objects are projective in the underlying abelian category.

2.4 Operations

2.4.1 \[\] (for IsChainOrCochainComplex, IsInt)

▷ \[\](C, i) (operation)

Returns: an object

The command returns the object of the chain or cochain complex in index i .

2.4.2 \^ (for IsChainOrCochainComplex, IsInt)

▷ \^(C, i) (operation)

Returns: a morphism

The command returns the differential of the chain or cochain complex in index i .

2.4.3 CertainCycle (for IsChainOrCochainComplex, IsInt)

▷ CertainCycle(C, n) (operation)

Returns: a morphism

The input is a chain or cochain complex C and an integer n . The output is the kernel embedding of the differential in index n .

2.4.4 CertainBoundary (for IsChainOrCochainComplex, IsInt)

▷ CertainBoundary(C, n) (operation)

Returns: a morphism

The input is a chain (resp. cochain) complex C and an integer n . The output is the image embeddin of $i + 1$ 'th (resp. $i - 1$ 'th) differential of C .

2.4.5 DefectOfExactness (for IsChainOrCochainComplex, IsInt)

▷ DefectOfExactness(C , n) (operation)

Returns: a object

The input is a chain (resp. cochain) complex C and an integer n . The outout is the homology (resp. cohomology) object of C in index n .

2.4.6 IsExactInIndex (for IsChainOrCochainComplex, IsInt)

▷ IsExactInIndex(C , n) (operation)

Returns: true or false

The input is a chain or cochain complex C and an integer n . The outout is *true* if C is exact in i . Otherwise the output is *false*.

2.4.7 SetUpperBound (for IsChainOrCochainComplex, IsInt)

▷ SetUpperBound(C , n) (operation)

Returns: Side effect

The command sets an upper bound n to the chain (resp. cochain) complex C . This means $C_{i \geq n} = 0$ ($C^{\geq n} = 0$). This upper bound will be called *active* upper bound of C . If C already has an active upper bound m , then m will be replaced by n only if n is better upper bound than m , i.e., $n \leq m$. If C has an active lower bound l and $n \leq l$ then the upper bound will set to equal l and as a consequence C will be set to zero.

2.4.8 SetLowerBound (for IsChainOrCochainComplex, IsInt)

▷ SetLowerBound(C , n) (operation)

Returns: Side effect

The command sets an lower bound n to the chain (resp. cochain) complex C . This means $C_{i \leq n} = 0$ ($C^{\leq n} = 0$). This lower bound will be called *active* lower bound of C . If C already has an active lower bound m , then m will be replaced by n only if n is better lower bound than m , i.e., $n \geq m$. If C has an active upper bound u and $n \geq u$ then the lower bound will set to equal u and as a consequence C will be set to zero.

2.4.9 HasActiveUpperBound (for IsChainOrCochainComplex)

▷ HasActiveUpperBound(C) (operation)

Returns: true or false

The input is chain or cochain complex. The output is *true* if an upper bound has been set to C and *false* otherwise.

2.4.10 HasActiveLowerBound (for IsChainOrCochainComplex)

▷ HasActiveLowerBound(C) (operation)

Returns: true or false

The input is chain or cochain complex. The output is *true* if a lower bound has been set to C and *false* otherwise.

2.4.11 ActiveUpperBound (for IsChainOrCochainComplex)

▷ ActiveUpperBound(C) (operation)

Returns: an integer

The input is chain or cochain complex. The output is its active upper bound if such has been set to C . Otherwise we get error.

2.4.12 ActiveLowerBound (for IsChainOrCochainComplex)

▷ ActiveLowerBound(C) (operation)

Returns: an integer

The input is chain or cochain complex. The output is its active lower bound if such has been set to C . Otherwise we get error.

2.4.13 Display (for IsChainOrCochainComplex, IsInt, IsInt)

▷ Display(C, m, n) (operation)

Returns: nothing

The input is chain or cochain complex C and two integers m and n . The command displays all components of C between the indices m, n .

2.5 Truncations

2.5.1 GoodTruncationBelow (for IsChainComplex, IsInt)

▷ GoodTruncationBelow(C, n) (operation)

Returns: chain complex

Let C_\bullet be chain complex. A good truncation of C_\bullet below n is the chain complex $\tau_{\geq n}C_\bullet$ whose differentials are defined by

$$d_i^{\tau_{\geq n}C_\bullet} = \begin{cases} 0 : 0 \leftarrow 0 & \text{if } i < n, \\ 0 : 0 \leftarrow Z_n & \text{if } i = n, \\ \text{KernelLift}(d_n^C, d_{n+1}^C) : Z_n \leftarrow C_{n+1} & \text{if } i = n+1, \\ d_i^C : C_{i-1} \leftarrow C_i & \text{if } i > n+1. \end{cases}$$

where Z_n is the cycle in index n . It can be shown that $H_i(\tau_{\geq n}C_\bullet) = 0$ for $i < n$ and $H_i(\tau_{\geq n}C_\bullet) = H_i(C_\bullet)$ for $i \geq n$.

$$\begin{array}{ccccccc} C_\bullet & \cdots & \longleftarrow & C_{n-1} & \longleftarrow & C_n & \longleftarrow & C_{n+1} & \longleftarrow & C_{n+2} & \longleftarrow & \cdots \\ \tau_{\geq n}C_\bullet & \cdots & \longleftarrow & 0 & \longleftarrow & Z_n & \longleftarrow & C_{n+1} & \longleftarrow & C_{n+2} & \longleftarrow & \cdots \end{array}$$

(Note: In the original image, blue arrows indicate the mapping of differentials: from C_n to Z_n , from Z_n to C_{n+1} , and from C_{n+1} to C_{n+2} in the bottom row.)

2.5.2 GoodTruncationAbove (for IsChainComplex, IsInt)

▷ GoodTruncationAbove(C, n) (operation)

Returns: chain complex

Let C_\bullet be chain complex. A good truncation of C_\bullet above n is the quotient chain complex $\tau_{< n}C_\bullet = C_\bullet / \tau_{\geq n}C_\bullet$. It can be shown that $H_i(\tau_{< n}C_\bullet) = 0$ for $i \geq n$ and $H_i(\tau_{< n}C_\bullet) = H_i(C_\bullet)$ for $i < n$.

2.5.3 GoodTruncationAbove (for IsCochainComplex, IsInt)

▷ GoodTruncationAbove(C, n)

(operation)

Returns:

Let C^\bullet be cochain complex. A good truncation of C^\bullet above n is the cochain complex $\tau^{\leq n}C^\bullet$ whose differentials are defined by

$$d_{\tau^{\leq n}C^\bullet}^i = \begin{cases} 0 : 0 \rightarrow 0 & \text{if } i > n, \\ 0 : Z_n \rightarrow 0 & \text{if } i = n, \\ \text{KernelLift}(d_C^n, d_C^{n-1}) : C_{n-1} \rightarrow Z_n & \text{if } i = n-1, \\ d_C^i : C_i \rightarrow C_{i+1} & \text{if } i < n-1. \end{cases}$$

where Z_n is the cycle in index n . It can be shown that $H^i(\tau^{\leq n}C^\bullet) = 0$ for $i > n$ and $H^i(\tau^{\leq n}C^\bullet) = H_i(C^\bullet)$ for $i \leq n$.

$$\begin{array}{ccccccc} \cdots & \longrightarrow & C_{n-2} & \longrightarrow & C_{n-1} & \longrightarrow & C_n & \longrightarrow & C_{n+1} & \longrightarrow & \cdots & C_\bullet \\ & & & & & & \uparrow & & & & & \\ & & & & & & Z_n & \longrightarrow & 0 & \longrightarrow & \cdots & \tau^{\leq n}C^\bullet \end{array}$$

2.5.4 GoodTruncationBelow (for IsCochainComplex, IsInt)

▷ GoodTruncationBelow(C, n)

(operation)

Returns: cochain complex

Let C^\bullet be cochain complex. A good truncation of C^\bullet below n is the quotient cochain complex $\tau^{> n}C^\bullet = C^\bullet / \tau^{\leq n}C^\bullet$. It can be shown that $H^i(\tau^{> n}C^\bullet) = 0$ for $i \leq n$ and $H^i(\tau^{> n}C^\bullet) = H_i(C^\bullet)$ for $i > n$.

2.5.5 BrutalTruncationBelow (for IsChainComplex, IsInt)

▷ BrutalTruncationBelow(C, n)

(operation)

Returns: chain complex

Let C_\bullet be chain complex. A brutal truncation of C_\bullet below n is the chain complex $\sigma_{\geq n}C_\bullet$ where $(\sigma_{\geq n}C_\bullet)_i = C_i$ when $i \geq n$ and $(\sigma_{\geq n}C_\bullet)_i = 0$ otherwise.

2.5.6 BrutalTruncationAbove (for IsChainComplex, IsInt)

▷ BrutalTruncationAbove(C, n)

(operation)

Returns: chain complex

Let C_\bullet be chain complex. A brutal truncation of C_\bullet above n is the chain quotient chain complex $\sigma_{< n}C_\bullet := C_\bullet / \sigma_{\geq n}C_\bullet$. Hence $(\sigma_{< n}C_\bullet)_i = C_i$ when $i < n$ and $(\sigma_{< n}C_\bullet)_i = 0$ otherwise.

2.5.7 BrutalTruncationAbove (for IsCochainComplex, IsInt)

▷ BrutalTruncationAbove(C, n)

(operation)

Returns: chain complex

Let C^\bullet be cochain complex. A brutal truncation of C_\bullet above n is the cochain complex $\sigma^{\leq n}C^\bullet$ where $(\sigma^{\leq n}C^\bullet)_i = C_i$ when $i \leq n$ and $(\sigma^{\leq n}C^\bullet)_i = 0$ otherwise.

▷ BrutalTruncationBelow(C, n) (operation)

Let C^\bullet be cochain complex. A brutal truncation of C^\bullet below n is the quotient cochain complex $\sigma^{>n}C^\bullet := C^\bullet / \sigma^{\leq n}C_\bullet$. Hence $(\sigma^{>n}C^\bullet)_i = C_i$ when $i > n$ and $(\sigma^{>n}C^\bullet)_i = 0$ otherwise.

Below we define the complex

$$\begin{array}{ccccccccccc} \dots & 2 & 3 & 4 & 5 & 6 & 7 & \dots \\ & & & & \begin{pmatrix} 0 \\ 0 \end{pmatrix} & & & \\ \dots & \longrightarrow & 0 & \longrightarrow & \mathbb{Q}^{1 \times 1} & \xrightarrow{\begin{pmatrix} 1 & 3 \end{pmatrix}} & \mathbb{Q}^{1 \times 2} & \xrightarrow{\begin{pmatrix} 0 \\ 0 \end{pmatrix}} & \mathbb{Q}^{1 \times 1} & \xrightarrow{\begin{pmatrix} 2 & 6 \end{pmatrix}} & \mathbb{Q}^{1 \times 2} & \longrightarrow & 0 & \longrightarrow & \dots \end{array}$$

```

gap> A := VectorSpaceObject( 1, Q );
<A vector space object over Q of dimension 1>
gap> B := VectorSpaceObject( 2, Q );
<A vector space object over Q of dimension 2>
gap> f := VectorSpaceMorphism( A, HomalgMatrix( [ [ 1, 3 ] ], 1, 2, Q ), B );
<A morphism in Category of matrices over Q>
gap> g := VectorSpaceMorphism( B, HomalgMatrix( [ [ 0 ], [ 0 ] ], 2, 1, Q ), A );
<A morphism in Category of matrices over Q>
gap> C := CochainComplex( [ f, g, 2*f ], 3 );
<A bounded object in cochain complexes category over category of matrices over Q
with active lower bound 2 and active upper bound 7.>
gap> ActiveUpperBound( C );
7
gap> ActiveLowerBound( C );
2
gap> C[ 1 ];
<A vector space object over Q of dimension 0>
gap> C[ 3 ];
<A vector space object over Q of dimension 1>
gap> C^3;
<A morphism in Category of matrices over Q>
gap> C^3 = f;
true
gap> Display( CertainCycle( C, 4 ) );
[ [ 1, 0 ],
  [ 0, 1 ] ]

A split monomorphism in Category of matrices over Q
gap> diffs := Differentials( C );
<An infinite list>
gap> diffs[ 1 ];
<A zero, isomorphism in Category of matrices over Q>
gap> diffs[ 10000 ];

```

```

<A zero, isomorphism in Category of matrices over Q>
gap> objs := Objects( C );
<An infinite list>
gap> DefectOfExactness( C, 4 );
<A vector space object over Q of dimension 1>
gap> DefectOfExactness( C, 3 );
<A vector space object over Q of dimension 0>
gap> IsExactInIndex( C, 4 );
false
gap> IsExactInIndex( C, 3 );
true
gap> C;
<A not cyclic, bounded object in cochain complexes category over category of
matrices over Q with active lower bound 2 and active upper bound 7.>
gap> P := CochainComplex( matrix_category, diffs );
<An object in Cochain complexes category over category of matrices over Q>
gap> SetUpperBound( P, 15 );
gap> P;
<A bounded from above object in cochain complexes category over category of
matrices over Q with active upper bound 15.>
gap> SetUpperBound( P, 20 );
gap> P;
<A bounded from above object in cochain complexes category over category of
matrices over Q with active upper bound 15.>
gap> ActiveUpperBound( P );
15
gap> SetUpperBound( P, 7 );
gap> P;
<A bounded from above object in cochain complexes category over category of
matrices over Q with active upper bound 7.>
gap> ActiveUpperBound( P );
7

```

Chapter 3

Complexes morphisms

3.1 Categories and filters

3.1.1 IsChainOrCochainMorphism (for IsCapCategoryMorphism)

▷ IsChainOrCochainMorphism(ϕ) (filter)
Returns: true or false
bla bla

3.1.2 IsBoundedBelowChainOrCochainMorphism (for IsChainOrCochainMorphism)

▷ IsBoundedBelowChainOrCochainMorphism(ϕ) (filter)
Returns: true or false
bla bla

3.1.3 IsBoundedAboveChainOrCochainMorphism (for IsChainOrCochainMorphism)

▷ IsBoundedAboveChainOrCochainMorphism(ϕ) (filter)
Returns: true or false
bla bla

3.1.4 IsBoundedChainOrCochainMorphism (for IsBoundedBelowChainOrCochainMorphism and IsBoundedAboveChainOrCochainMorphism)

▷ IsBoundedChainOrCochainMorphism(ϕ) (filter)
Returns: true or false
bla bla

3.1.5 IsChainMorphism (for IsChainOrCochainMorphism)

▷ IsChainMorphism(ϕ) (filter)
Returns: true or false
bla bla

3.1.6 **IsBoundedBelowChainMorphism** (for **IsBoundedBelowChainOrCochainMorphism** and **IsChainMorphism**)

▷ **IsBoundedBelowChainMorphism**(*phi*) (filter)
Returns: true or false
 bla bla

3.1.7 **IsBoundedAboveChainMorphism** (for **IsBoundedAboveChainOrCochainMorphism** and **IsChainMorphism**)

▷ **IsBoundedAboveChainMorphism**(*phi*) (filter)
Returns: true or false
 bla bla

3.1.8 **IsBoundedChainMorphism** (for **IsBoundedChainOrCochainMorphism** and **IsChainMorphism**)

▷ **IsBoundedChainMorphism**(*phi*) (filter)
Returns: true or false
 bla bla

3.1.9 **IsCochainMorphism** (for **IsChainOrCochainMorphism**)

▷ **IsCochainMorphism**(*phi*) (filter)
Returns: true or false
 bla bla

3.1.10 **IsBoundedBelowCochainMorphism** (for **IsBoundedBelowChainOrCochainMorphism** and **IsCochainMorphism**)

▷ **IsBoundedBelowCochainMorphism**(*phi*) (filter)
Returns: true or false
 bla bla

3.1.11 **IsBoundedAboveCochainMorphism** (for **IsBoundedAboveChainOrCochainMorphism** and **IsCochainMorphism**)

▷ **IsBoundedAboveCochainMorphism**(*phi*) (filter)
Returns: true or false
 bla bla

3.1.12 **IsBoundedCochainMorphism** (for **IsBoundedChainOrCochainMorphism** and **IsCochainMorphism**)

▷ **IsBoundedCochainMorphism**(*phi*) (filter)
Returns: true or false
 bla bla

3.2 Creating chain and cochain morphisms

3.2.1 ChainMorphism (for IsChainComplex, IsChainComplex, IsZList)

▷ ChainMorphism(C, D, l) (operation)

Returns: a chain morphism

The input is two chain complexes C, D and an infinite list l . The output is the chain morphism $\phi : C \rightarrow D$ defined by $\phi_i := l[i]$.

3.2.2 ChainMorphism (for IsChainComplex, IsChainComplex, IsDenseList, IsInt)

▷ ChainMorphism(C, D, l, k) (operation)

Returns: a chain morphism

The input is two chain complexes C, D , dense list l and an integer k . The output is the chain morphism $\phi : C \rightarrow D$ such that $\phi_k = l[1]$, $\phi_{k+1} = l[2]$, etc.

3.2.3 ChainMorphism (for IsDenseList, IsInt, IsDenseList, IsInt, IsDenseList, IsInt)

▷ ChainMorphism(c, m, d, n, l, k) (operation)

Returns: a chain morphism

The output is the chain morphism $\phi : C \rightarrow D$, where $C_m = c[1]$, $C_{m+1} = c[2]$, etc. $D_n = d[1]$, $D_{n+1} = d[2]$, etc. and $\phi_k = l[1]$, $\phi_{k+1} = l[2]$, etc.

3.2.4 CochainMorphism (for IsCochainComplex, IsCochainComplex, IsZList)

▷ CochainMorphism(C, D, l) (operation)

Returns: a cochain morphism

The input is two cochain complexes C, D and an infinite list l . The output is the cochain morphism $\phi : C \rightarrow D$ defined by $\phi_i := l[i]$.

3.2.5 CochainMorphism (for IsCochainComplex, IsCochainComplex, IsDenseList, IsInt)

▷ CochainMorphism(C, D, l, k) (operation)

Returns: a chain morphism

The input is two cochain complexes C, D , dense list l and an integer k . The output is the cochain morphism $\phi : C \rightarrow D$ such that $\phi^k = l[1]$, $\phi^{k+1} = l[2]$, etc.

3.2.6 CochainMorphism (for IsDenseList, IsInt, IsDenseList, IsInt, IsDenseList, IsInt)

▷ CochainMorphism(c, m, d, n, l, k) (operation)

Returns: a cochain morphism

The output is the cochain morphism $\phi : C \rightarrow D$, where $C^m = c[1]$, $C^{m+1} = c[2]$, etc. $D^n = d[1]$, $D^{n+1} = d[2]$, etc. and $\phi^k = l[1]$, $\phi^{k+1} = l[2]$, etc.

3.3 Attributes

3.3.1 Morphisms (for IsChainOrCochainMorphism)

- ▷ `Morphisms(phi)` (attribute)
Returns: infinite list
 The output is morphisms of the chain or cochain morphism as an infinite list.

3.3.2 MappingCone (for IsChainOrCochainMorphism)

- ▷ `MappingCone(phi)` (attribute)
Returns: complex
 The input a chain (resp. cochain) morphism $\phi : C \rightarrow D$. The output is its mapping cone chain (resp. cochain) complex $\text{Cone}(\phi)$.

3.3.3 NaturalInjectionInMappingCone (for IsChainOrCochainMorphism)

- ▷ `NaturalInjectionInMappingCone(phi)` (attribute)
Returns: chain (resp. cochain) morphism
 The input a chain (resp. cochain) morphism $\phi : C \rightarrow D$. The output is the natural injection $i : D \rightarrow \text{Cone}(\phi)$.

3.3.4 NaturalProjectionFromMappingCone (for IsChainOrCochainMorphism)

- ▷ `NaturalProjectionFromMappingCone(phi)` (attribute)
Returns: chain (resp. cochain) morphism
 The input a chain (resp. cochain) morphism $\phi : C \rightarrow D$. The output is the natural projection $\pi : \text{Cone}(\phi) \rightarrow C[u]$ where $u = -1$ if ϕ is chain morphism and $u = 1$ if ϕ is cochain morphism.

3.4 Properties

3.4.1 IsQuasiIsomorphism_ (for IsChainOrCochainMorphism)

- ▷ `IsQuasiIsomorphism_(phi)` (property)
Returns: true or false
 The input a chain (resp. cochain) morphism $\phi : C \rightarrow D$. The output is *true* if ϕ is quasi-isomorphism and *false* otherwise. If ϕ is not bounded an error is raised.

3.5 Operations

3.5.1 SetUpperBound (for IsChainOrCochainMorphism, IsInt)

- ▷ `SetUpperBound(phi, n)` (operation)
Returns: a side effect
 The command sets an upper bound to the morphism ϕ . An upper bound of ϕ is an integer u with $\phi_{i \geq u} = 0$. The integer u will be called *active* upper bound of ϕ . If ϕ already has an active upper bound, say u' , then u' will be replaced by u only if $u \leq u'$.

3.5.2 SetLowerBound (for IsChainOrCochainMorphism, IsInt)

▷ SetLowerBound(ϕ , n) (operation)

Returns: a side effect

The command sets an lower bound to the morphism ϕ . A lower bound of ϕ is an integer l with $\phi_{i \leq l} = 0$. The integer l will be called *active* lower bound of ϕ . If ϕ already has an active lower bound, say l' , then l' will be replaced by l only if $l \geq l'$.

3.5.3 HasActiveUpperBound (for IsChainOrCochainMorphism)

▷ HasActiveUpperBound(ϕ) (operation)

Returns: true or false

The input is chain or cochain morphism ϕ . The output is *true* if an upper bound has been set to ϕ and *false* otherwise.

3.5.4 HasActiveLowerBound (for IsChainOrCochainMorphism)

▷ HasActiveLowerBound(ϕ) (operation)

Returns: true or false

The input is chain or cochain morphism ϕ . The output is *true* if a lower bound has been set to ϕ and *false* otherwise.

3.5.5 ActiveUpperBound (for IsChainOrCochainMorphism)

▷ ActiveUpperBound(ϕ) (operation)

Returns: an integer

The input is chain or cochain morphism. The output is its active upper bound if such has been set to ϕ . Otherwise we get error.

3.5.6 ActiveLowerBound (for IsChainOrCochainMorphism)

▷ ActiveLowerBound(ϕ) (operation)

Returns: an integer

The input is chain or cochain morphism. The output is its active lower bound if such has been set to ϕ . Otherwise we get error.

3.5.7 \[\] (for IsChainOrCochainMorphism, IsInt)

▷ \[\](ϕ , n) (operation)

Returns: an integer

The input is chain (resp. cochain) morphism and an integer n . The output is the component of ϕ in index n , i.e., ϕ_n (resp. ϕ'').

3.5.8 Display (for IsChainOrCochainMorphism, IsInt, IsInt)

▷ Display(ϕ , m , n) (operation)

Returns:

The command displays the components of the morphism between m and n .

3.6 Examples

Let us define a morphism

$$\begin{array}{ccccccccccc}
 \dots & & 2 & & 3 & & 4 & & 5 & & 6 & & 7 & & \dots \\
 & & & & & & & & & & & & & & \\
 \dots & \longrightarrow & 0 & \longrightarrow & \mathbb{Q}^{1 \times 1} & \xrightarrow{\begin{pmatrix} 1 & 3 \end{pmatrix}} & \mathbb{Q}^{1 \times 2} & \xrightarrow{\begin{pmatrix} 0 \\ 0 \end{pmatrix}} & \mathbb{Q}^{1 \times 1} & \xrightarrow{\begin{pmatrix} 2 & 6 \end{pmatrix}} & \mathbb{Q}^{1 \times 2} & \longrightarrow & 0 & \longrightarrow & \dots \\
 & & & & & & \downarrow \begin{pmatrix} 0 \\ 0 \end{pmatrix} & & \downarrow \begin{pmatrix} 10 \end{pmatrix} & & & & & & \\
 \dots & \longrightarrow & 0 & \longrightarrow & 0 & \longrightarrow & \mathbb{Q}^{1 \times 1} & \xrightarrow{\begin{pmatrix} 5 \end{pmatrix}} & \mathbb{Q}^{1 \times 1} & \longrightarrow & 0 & \longrightarrow & 0 & \longrightarrow & \dots
 \end{array}$$

Example

```

gap> h := VectorSpaceMorphism( A, HomalgMatrix( [ [ 5 ] ], 1, 1, Q ), A );
<A morphism in Category of matrices over Q>
gap> phi4 := g;
<A morphism in Category of matrices over Q>
gap> phi5 := 2*h;
<A morphism in Category of matrices over Q>
gap> D := CochainComplex( [ h ], 4 );
<A bounded object in cochain complexes category over category of matrices
over Q with active lower bound 3 and active upper bound 6.>
gap> phi := CochainMorphism( C, D, [ phi4, phi5 ], 4 );
<A bounded morphism in cochain complexes category over category of matrices
over Q with active lower bound 3 and active upper bound 6.>
gap> Display( phi[ 5 ] );
[ [ 10 ] ]

```

A morphism in Category of matrices over Q

```

gap> ActiveLowerBound( phi );
3
gap> IsZeroForMorphisms( phi );
false
gap> IsExact( D );
true
gap> IsExact( C );
false

```

Now lets define the previous morphism using the command `CochainMorphism(c, m, d, n, l, k)`.

Example

```

gap> psi := CochainMorphism( [ f, g, 2*f ], 3, [ h ], 4, [ phi4, phi5 ], 4 );
<A bounded morphism in cochain complexes category over category of matrices
over Q with active lower bound 3 and active upper bound 6.>

```

In some cases the morphism can change its lower bound when we apply the function `IsZeroForMorphisms`.

Example

```
gap> IsZeroForMorphisms( psi );
false
gap> psi;
<A bounded morphism in cochain complexes category over category of matrices
over Q with active lower bound 4 and active upper bound 6.>
```

In the following we compute the mapping cone of ψ and its natural injection and projection.

$$\begin{array}{ccccccccccc}
 & & & & 2 & & 3 & & 4 & & 5 & & 6 & & \dots \\
 & & & & & & & & & & & & & & \\
 C: & \longrightarrow & 0 & \longrightarrow & \mathbb{Q}^{1 \times 1} & \xrightarrow{\begin{pmatrix} 1 & 3 \end{pmatrix}} & \mathbb{Q}^{1 \times 2} & \xrightarrow{\begin{pmatrix} 0 \\ 0 \end{pmatrix}} & \mathbb{Q}^{1 \times 1} & \xrightarrow{\begin{pmatrix} 2 & 6 \end{pmatrix}} & \mathbb{Q}^{1 \times 2} & \longrightarrow & \dots \\
 \psi \downarrow & & & & & & \downarrow \begin{pmatrix} 0 \\ 0 \end{pmatrix} & & \downarrow \begin{pmatrix} 0 \\ 0 \end{pmatrix} & & \downarrow \begin{pmatrix} 10 \end{pmatrix} & & & & \\
 D: & \longrightarrow & 0 & \longrightarrow & 0 & \longrightarrow & \mathbb{Q}^{1 \times 1} & \xrightarrow{\begin{pmatrix} 5 \end{pmatrix}} & \mathbb{Q}^{1 \times 1} & \longrightarrow & 0 & \longrightarrow & \dots \\
 i \downarrow & & & & & & \downarrow \begin{pmatrix} 0 & 1 \end{pmatrix} & & \downarrow \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} & & & & & \\
 Cone(\psi): & \longrightarrow & \mathbb{Q}^{1 \times 1} & \xrightarrow{\begin{pmatrix} -1 & -3 \end{pmatrix}} & \mathbb{Q}^{1 \times 2} & \xrightarrow{\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}} & \mathbb{Q}^{1 \times 2} & \xrightarrow{\begin{pmatrix} -2 & -6 & -10 \\ 0 & 0 & 5 \end{pmatrix}} & \mathbb{Q}^{1 \times 3} & \longrightarrow & 0 & \longrightarrow & \dots \\
 p \downarrow & & \downarrow \begin{pmatrix} 1 \end{pmatrix} & & \downarrow \begin{pmatrix} 1,0 \\ 0,1 \end{pmatrix} & & \downarrow \begin{pmatrix} 0 \\ 0 \end{pmatrix} & & \downarrow \begin{pmatrix} 1 \\ 0 \end{pmatrix} & & \downarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} & & & \\
 C[1]: & \longrightarrow & \mathbb{Q}^{1 \times 1} & \xrightarrow{\begin{pmatrix} -1 & -3 \end{pmatrix}} & \mathbb{Q}^{1 \times 2} & \xrightarrow{\begin{pmatrix} 0 \\ 0 \end{pmatrix}} & \mathbb{Q}^{1 \times 1} & \xrightarrow{\begin{pmatrix} -2 & -6 \end{pmatrix}} & \mathbb{Q}^{1 \times 2} & \longrightarrow & 0 & \longrightarrow & \dots
 \end{array}$$

Example

```
gap> cone := MappingCone( psi );
<A bounded object in cochain complexes category over category of matrices over
Q with active lower bound 1 and active upper bound 6.>
gap> cone^4;
<A morphism in Category of matrices over Q>
gap> Display( cone^4 );
[ [ -2, -6, -10 ],
  [ 0, 0, 5 ] ]

A morphism in Category of matrices over Q
gap> i := NaturalInjectionInMappingCone( psi );
<A bounded morphism in cochain complexes category over category of matrices over
Q with active lower bound 3 and active upper bound 6.>
gap> p := NaturalProjectionFromMappingCone( psi );
<A bounded morphism in cochain complexes category over category of matrices over
Q with active lower bound 1 and active upper bound 6.>
```

Chapter 4

Functors

4.1 Basic functors for complex categories.

4.1.1 HomologyFunctor (for IsChainComplexCategory, IsCapCategory, IsInt)

- ▷ `HomologyFunctor(Ch(A), A, n)` (operation)
- ▷ `CohomologyFunctor(Coch(A), A, n)` (operation)

Returns: a functor

The first argument in the input must be the chain (resp. cochain) complex category of an abelian category A , the second argument is A and the third argument is an integer n . The output is the n 'th homology (resp. cohomology) functor : $\text{Ch}(A) \rightarrow A$.

4.1.2 ShiftFunctor (for IsChainOrCochainComplexCategory, IsInt)

- ▷ `ShiftFunctor(Comp(A), n)` (operation)

Returns: a functor

The inputs are complex category $\text{Comp}(A)$ and an integer. The output is a the endofunctor $T[n]$ that sends any complex C to $C[n]$ and any complex morphism $\phi : C \rightarrow D$ to $\phi[n] : C[n] \rightarrow D[n]$. The shift chain complex $C[n]$ of a chain complex C is defined by $C[n]_i = C_{n+i}$, $d_i^{C[n]} = (-1)^n d_{n+i}^C$ and the same for chain complex morphisms, i.e., $\phi[n]_i = \phi_{n+i}$. The same holds for cochain complexes and morphisms.

4.1.3 UnsignedShiftFunctor (for IsChainOrCochainComplexCategory, IsInt)

- ▷ `UnsignedShiftFunctor(Comp(A), n)` (operation)

Returns: a functor

The inputs are complex category $\text{Comp}(A)$ and an integer. The output is a the endofunctor $T[n]$ that sends any complex C to $C[n]$ and any complex morphism $\phi : C \rightarrow D$ to $\phi[n] : C[n] \rightarrow D[n]$. The shift chain complex $C[n]$ of a chain complex C is defined by $C[n]_i = C_{n+i}$, $d_i^{C[n]} = d_{n+i}^C$ and the same for chain complex morphisms, i.e., $\phi[n]_i = \phi_{n+i}$. The same holds for cochain complexes and morphisms.

4.1.4 ChainToCochainComplexFunctor (for IsCapCategory)

- ▷ `ChainToCochainComplexFunctor(A)` (operation)

Returns: a functor

The input is a category A . The output is the functor $F : \text{Ch}(A) \rightarrow \text{Coch}(A)$ defined by $C_\bullet \mapsto C^\bullet$ for any for any chain complex $C_\bullet \in \text{Ch}(A)$ and by $\phi_\bullet \mapsto \phi^\bullet$ for any map ϕ where $C^i = C_{-i}$ and $\phi^i = \phi_{-i}$.

4.1.5 CochainToChainComplexFunctor (for IsCapCategory)

▷ CochainToChainComplexFunctor(A) (operation)

Returns: a functor

The input is a category A . The output is the functor $F : \text{Coch}(A) \rightarrow \text{Ch}(A)$ defined by $C^\bullet \mapsto C_\bullet$ for any cochain complex $C^\bullet \in \text{Coch}(A)$ and by $\phi^\bullet \mapsto \phi_\bullet$ for any map ϕ where $C_i = C^{-i}$ and $\phi_i = \phi^{-i}$.

4.1.6 ExtendFunctorToChainComplexCategoryFunctor (for IsCapFunctor)

▷ ExtendFunctorToChainComplexCategoryFunctor(F) (operation)

Returns: a functor

The input is a functor $F : A \rightarrow B$. The output is its extention functor $F : \text{Ch}(A) \rightarrow \text{Ch}(B)$.

4.1.7 ExtendFunctorToCochainComplexCategoryFunctor (for IsCapFunctor)

▷ ExtendFunctorToCochainComplexCategoryFunctor(F) (operation)

Returns: a functor

The input is a functor $F : A \rightarrow B$. The output is its extention functor $F : \text{Coch}(A) \rightarrow \text{Coch}(B)$.

4.2 Examples

The theory tells us that the composition $i\psi$ is null-homotopic. That implies that the morphisms induced on cohomologies are all zero.

Example

```
gap> i_o_psi := PreCompose( psi, i );
<A bounded morphism in cochain complexes category over category of matrices
over Q with active lower bound 4 and active upper bound 6.>
gap> H5 := CohomologyFunctor( cochain_cat, matrix_category, 5 );
5-th cohomology functor in category of matrices over Q
gap> IsZeroForMorphisms( ApplyFunctor( H5, i_o_psi ) );
true
```

Next we define a functor $\mathbf{F} : \text{Vec}_{\mathbb{Q}} \rightarrow \text{Vec}_{\mathbb{Q}}$ that maps every \mathbb{Q} -vector space A to $A \oplus A$ and every morphism $f : A \rightarrow B$ to $f \oplus f$. Then we extend it to the functor $\mathbf{Coch}_{\mathbf{F}} : \text{Coch}(\text{Vec}_{\mathbb{Q}}) \rightarrow \text{Coch}(\text{Vec}_{\mathbb{Q}})$ that maps each cochain complex C to the cochain complex we get after applying the functor \mathbf{F} on every object and differential in C and maps any morphism $\phi : C \rightarrow D$ to the morphism we get after applying the functor \mathbf{F} on every object, differential or morphism in C, D and ϕ .

Example

```
gap> F := CapFunctor( "double functor", matrix_category, matrix_category );
double functor
gap> u := function( obj ) return DirectSum( [ obj, obj ] ); end;;
gap> AddObjectFunction( F, u );
gap> v := function( s, mor, r ) return DirectSumFunctorial( [ mor, mor ] ); end;;
gap> AddMorphismFunction( F, v );
gap> Display( f );
[[ 1, 3 ]]
```

```

A morphism in Category of matrices over Q
gap> Display( ApplyFunctor( F, f ) );
[ [ 1, 3, 0, 0 ],
  [ 0, 0, 1, 3 ] ]

A morphism in Category of matrices over Q
gap> Coch_F := ExtendFunctorToCochainComplexCategoryFunctor( F );
Extended version of double functor from cochain complexes category over category
of matrices over Q to cochain complexes category over category of matrices over Q
gap> psi;
<A bounded morphism in cochain complexes category over category of matrices
over Q with active lower bound 4 and active upper bound 6.>
gap> Coch_F_psi := ApplyFunctor( Coch_F, psi );
<A bounded morphism in cochain complexes category over category of matrices
over Q with active lower bound 4 and active upper bound 6.>
gap> Display( psi[ 5 ] );
[ [ 10 ] ]

A morphism in Category of matrices over Q
gap> Display( Coch_F_psi[ 5 ] );
[ [ 10, 0 ],
  [ 0, 10 ] ]

A morphism in Category of matrices over Q

```

Next we will compute the shift $C[3]$. As we know the standard shift functor may change the sign of the differentials since $d_{C[n]}^i = (-1)^n d_C^{i+n}$. Hence if we don't want the signs to be changed we may use the unsigned shift functor.

Example

```

gap> T := ShiftFunctor( cochain_cat, 3 );
Shift (3 times to the left) functor in cochain complexes category over category
of matrices over Q
gap> C;
<A not cyclic, bounded object in cochain complexes category over category of
matrices over Q with active lower bound 2 and active upper bound 7.>
gap> C_3 := ApplyFunctor( T, C );
<A not cyclic, bounded object in cochain complexes category over category of
matrices over Q with active lower bound -1 and active upper bound 4.>
gap> Display( C^3 );
[ [ 1, 3 ] ]

A morphism in Category of matrices over Q
gap> Display( C_3^0 );
[ [ -1, -3 ] ]

A morphism in Category of matrices over Q
gap> S := UnsignedShiftFunctor( cochain_cat, 3 );
Unsigned shift (3 times to the left) functor in cochain complexes category over
category of matrices over Q
gap> C_3_unsigned := ApplyFunctor( S, C );
<A bounded object in cochain complexes category over category of matrices over
Q with active lower bound -1 and active upper bound 4.>

```



```
gap> Display( C_3_unsigned^0 );  
[ [ 1, 3 ] ]
```

A morphism in Category of matrices over \mathbb{Q}

Index

- $\backslash[\backslash]$
 - for `IsChainOrCochainComplex`, `IsInt`, [9](#)
 - for `IsChainOrCochainMorphism`, `IsInt`, [19](#)
- $\backslash\sim$
 - for `IsChainOrCochainComplex`, `IsInt`, [9](#)
- `ActiveLowerBound`
 - for `IsChainOrCochainComplex`, [11](#)
 - for `IsChainOrCochainMorphism`, [19](#)
- `ActiveUpperBound`
 - for `IsChainOrCochainComplex`, [11](#)
 - for `IsChainOrCochainMorphism`, [19](#)
- `BrutalTruncationAbove`
 - for `IsChainComplex`, `IsInt`, [12](#)
 - for `IsCochainComplex`, `IsInt`, [12](#)
- `BrutalTruncationBelow`
 - for `IsChainComplex`, `IsInt`, [12](#)
 - for `IsCochainComplex`, `IsInt`, [13](#)
- `CatOfComplex`
 - for `IsChainOrCochainComplex`, [9](#)
- `CertainBoundary`
 - for `IsChainOrCochainComplex`, `IsInt`, [9](#)
- `CertainCycle`
 - for `IsChainOrCochainComplex`, `IsInt`, [9](#)
- `ChainComplex`
 - for `IsCapCategory`, `IsZList`, [7](#)
 - for `IsDenseList`, [7](#)
 - for `IsDenseList`, `IsInt`, [7](#)
- `ChainComplexCategory`
 - for `IsCapCategory`, [3](#)
- `ChainComplexWithInductiveNegativeSide`
 - for `IsCapCategoryMorphism`, `IsFunction`, [8](#)
- `ChainComplexWithInductivePositiveSide`
 - for `IsCapCategoryMorphism`, `IsFunction`, [8](#)
- `ChainComplexWithInductiveSides`
 - for `IsCapCategoryMorphism`, `IsFunction`, `IsFunction`, [7](#)
- `ChainMorphism`
 - for `IsChainComplex`, `IsChainComplex`, `IsDenseList`, `IsInt`, [17](#)
 - for `IsChainComplex`, `IsChainComplex`, `IsZList`, [17](#)
 - for `IsDenseList`, `IsInt`, `IsDenseList`, `IsInt`, `IsDenseList`, `IsInt`, [17](#)
- `ChainToCochainComplexFunctor`
 - for `IsCapCategory`, [22](#)
- `CochainComplex`
 - for `IsCapCategory`, `IsZList`, [7](#)
 - for `IsDenseList`, [7](#)
 - for `IsDenseList`, `IsInt`, [7](#)
- `CochainComplexCategory`
 - for `IsCapCategory`, [3](#)
- `CochainComplexWithInductiveNegativeSide`
 - for `IsCapCategoryMorphism`, `IsFunction`, [8](#)
- `CochainComplexWithInductivePositiveSide`
 - for `IsCapCategoryMorphism`, `IsFunction`, [8](#)
- `CochainComplexWithInductiveSides`
 - for `IsCapCategoryMorphism`, `IsFunction`, `IsFunction`, [7](#)
- `CochainMorphism`
 - for `IsCochainComplex`, `IsCochainComplex`, `IsDenseList`, `IsInt`, [17](#)
 - for `IsCochainComplex`, `IsCochainComplex`, `IsZList`, [17](#)
 - for `IsDenseList`, `IsInt`, `IsDenseList`, `IsInt`, `IsDenseList`, `IsInt`, [17](#)
- `CochainToChainComplexFunctor`
 - for `IsCapCategory`, [23](#)
- `CohomologyFunctor`
 - for `IsCochainComplexCategory`, `IsCapCategory`, `IsInt`, [22](#)
- `DefectOfExactness`
 - for `IsChainOrCochainComplex`, `IsInt`, [10](#)
- `Differentials`

- for `IsChainOrCochainComplex`, 8
- `Display`
 - for `IsChainOrCochainComplex`, `IsInt`, `IsInt`, 11
 - for `IsChainOrCochainMorphism`, `IsInt`, `IsInt`, 19
- `ExtendFunctorToChainComplexCategory-Functor`
 - for `IsCapFunctor`, 23
- `ExtendFunctorToCochainComplexCategory-Functor`
 - for `IsCapFunctor`, 23
- `GoodTruncationAbove`
 - for `IsChainComplex`, `IsInt`, 11
 - for `IsCochainComplex`, `IsInt`, 12
- `GoodTruncationBelow`
 - for `IsChainComplex`, `IsInt`, 11
 - for `IsCochainComplex`, `IsInt`, 12
- `HasActiveLowerBound`
 - for `IsChainOrCochainComplex`, 10
 - for `IsChainOrCochainMorphism`, 19
- `HasActiveUpperBound`
 - for `IsChainOrCochainComplex`, 10
 - for `IsChainOrCochainMorphism`, 19
- `HomologyFunctor`
 - for `IsChainComplexCategory`, `IsCapCategory`, `IsInt`, 22
- `IsBoundedAboveChainComplex`
 - for `IsBoundedAboveChainOrCochainComplex` and `IsChainComplex`, 6
- `IsBoundedAboveChainMorphism`
 - for `IsBoundedAboveChainOrCochainMorphism` and `IsChainMorphism`, 16
- `IsBoundedAboveChainOrCochainComplex`
 - for `IsChainOrCochainComplex`, 5
- `IsBoundedAboveChainOrCochainMorphism`
 - for `IsChainOrCochainMorphism`, 15
- `IsBoundedAboveCochainComplex`
 - for `IsBoundedAboveChainOrCochainComplex` and `IsCochainComplex`, 6
- `IsBoundedAboveCochainMorphism`
 - for `IsBoundedAboveChainOrCochainMorphism` and `IsCochainMorphism`, 16
- `IsBoundedBelowChainComplex`
 - for `IsBoundedBelowChainOrCochainComplex` and `IsChainComplex`, 6
- `IsBoundedBelowChainMorphism`
 - for `IsBoundedBelowChainOrCochainMorphism` and `IsChainMorphism`, 16
- `IsBoundedBelowChainOrCochainComplex`
 - for `IsChainOrCochainComplex`, 5
- `IsBoundedBelowChainOrCochainMorphism`
 - for `IsChainOrCochainMorphism`, 15
- `IsBoundedBelowCochainComplex`
 - for `IsBoundedBelowChainOrCochainComplex` and `IsCochainComplex`, 6
- `IsBoundedBelowCochainMorphism`
 - for `IsBoundedBelowChainOrCochainMorphism` and `IsCochainMorphism`, 16
- `IsBoundedChainComplex`
 - for `IsBoundedChainOrCochainComplex` and `IsChainComplex`, 6
- `IsBoundedChainMorphism`
 - for `IsBoundedChainOrCochainMorphism` and `IsChainMorphism`, 16
- `IsBoundedChainOrCochainComplex`
 - for `IsBoundedBelowChainOrCochainComplex` and `IsBoundedAboveChainOrCochainComplex`, 6
- `IsBoundedChainOrCochainMorphism`
 - for `IsBoundedBelowChainOrCochainMorphism` and `IsBoundedAboveChainOrCochainMorphism`, 15
- `IsBoundedCochainComplex`
 - for `IsBoundedChainOrCochainComplex` and `IsCochainComplex`, 6
- `IsBoundedCochainMorphism`
 - for `IsBoundedChainOrCochainMorphism` and `IsCochainMorphism`, 16
- `IsChainComplex`
 - for `IsChainOrCochainComplex`, 5
- `IsChainComplexCategory`
 - for `IsChainOrCochainComplexCategory`, 3
- `IsChainMorphism`
 - for `IsChainOrCochainMorphism`, 15
- `IsChainOrCochainComplex`
 - for `IsCapCategoryObject`, 5
- `IsChainOrCochainComplexCategory`
 - for `IsCapCategory`, 3

IsChainOrCochainMorphism
 for IsCapCategoryMorphism, 15
 IsCochainComplex
 for IsChainOrCochainComplex, 5
 IsCochainComplexCategory
 for IsChainOrCochainComplexCategory, 3
 IsCochainMorphism
 for IsChainOrCochainMorphism, 16
 IsExactInIndex
 for IsChainOrCochainComplex, IsInt, 10
 IsQuasiIsomorphism_
 for IsChainOrCochainMorphism, 18

 MappingCone
 for IsChainOrCochainMorphism, 18
 Morphisms
 for IsChainOrCochainMorphism, 18

 NaturalInjectionInMappingCone
 for IsChainOrCochainMorphism, 18
 NaturalProjectionFromMappingCone
 for IsChainOrCochainMorphism, 18

 Objects
 for IsChainOrCochainComplex, 8

 QuasiIsomorphismFromProjective-
 Resolution
 for IsBoundedAboveCochainComplex, 9
 for IsBoundedBelowChainComplex, 9

 SetLowerBound
 for IsChainOrCochainComplex, IsInt, 10
 for IsChainOrCochainMorphism, IsInt, 19
 SetUpperBound
 for IsChainOrCochainComplex, IsInt, 10
 for IsChainOrCochainMorphism, IsInt, 18
 ShiftFunctor
 for IsChainOrCochainComplexCategory,
 IsInt, 22
 StalkChainComplex
 for IsCapCategoryObject, IsInt, 7
 StalkCochainComplex
 for IsCapCategoryObject, IsInt, 7

 UnderlyingCategory
 for IsChainOrCochainComplexCategory, 4
 UnsignedShiftFunctor