homalg

An abstract MAPLE-package for homological algebra

Mohamed Barakat and Daniel Robertz

Lehrstuhl B für Mathematik
RWTH-Aachen University
Germany

September 21, 2006

Outline

the package homalg

- Q: what does “abstract” MAPLE-package mean?
  A: homalg depends on a “ring package” that implements the ring-specific arithmetics

the package homalg

- Q: what does "abstract" MAPLE-package mean?
  A: homalg depends on a "ring package" that implements the ring-specific arithmetics
- available ring packages: `PIR`, `Involutive`, `Janet`, `JanetOre`, `OreModules`...

the package homalg

- Q: what does “abstract” MAPLE-package mean?
  A: homalg depends on a “ring package” that implements the ring-specific arithmetics
- available ring packages: PIR, Involutive, Janet, JanetOre, OreModules...
- homological algebra in MAPLE:
  the full subcategory of *finitely presented modules* of the abelian category of modules over a ring R, in which one can *algorithmically* solve the *ideal membership problem* (e.g. *reduce* via a *basis*) and compute *syzygies*

the package homalg

- Q: what does “abstract” MAPLE-package mean?
  A: homalg depends on a “ring package” that implements the ring-specific arithmetics
- available ring packages: PIR, Involutive, Janet, JanetOre, OreModules...
- homological algebra in MAPLE:
  the full subcategory of *finitely presented modules* of the abelian category of modules over a ring R, in which one can *algorithmically* solve the *ideal membership problem* (e.g. *reduce* via a *basis*) and compute *syzygies*
- modules are given by a finite presentation:
  $\mathrm{coker}(R^{l_1} \xrightarrow{.A} R^{l_0})$, $A \in R^{l_1 \times l_0}$

presentation of a module

$$[[[1,0,0] = \begin{bmatrix} 0 & y & 0 \\ 0 & -y & 0 \end{bmatrix}, [0,1,0] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, [0,0,1] = \begin{bmatrix} 0 & 0 & -y \\ 0 & 0 & x \end{bmatrix}],$$

$$[[x-y,0,0],[y,xy,0],[0,0,z^3]],$$

"Presentation",

$$3 + 8\,s + 14\,s^2 + s^3 \left(\frac{14}{(1-s)} + \frac{6}{(1-s)^2}\right),$$

$$[14,6,0]]$$

generators

relations

HILBERT-series

CARTAN-characters

Outline

basic procedures in homalg

the ring package provides: BasisOfModule and Reduce

- RightDivide=PreImage, ReduceHomomorphism, Presentation
- Intersection, CompleteImSq
- ConnectingHom, Cokernel, TensorProduct
- OnMorphisms

basic procedures in homalg

the ring package provides: BasisOfModule and Reduce

- RightDivide=PreImage, ReduceHomomorphism, Presentation
- Intersection, CompleteImSq
- ConnectingHom, Cokernel, TensorProduct
- OnMorphisms

the ring package also provides: SyzygiesGenerators

- SubfactorModule, ResolutionOfModule, SyzygiesOp
- Kernel, DefectOfHoms, ParametrizeModule
- TorsionSubmodule, Pullback, Hom_R, Hom
- HomologyModules, CohomologyModules
- IsExactSeq, IsExactCoseq
- LongExactHomolSeq, LongExactCohomolSeq

advanced procedures in homalg

building upon the above

- `ResolveShortExactSeq`
- `LeftDerivedFunctor`, `LeftDerivedRightExactFunctor`, `RightDerivedFunctor`, `RightDerivedLeftExactCofunctor`
- `Tor`, `Ext_R`, `Ext`
- ...

the basic functors in homalg

using FunctorMap we have the following *functors*:

- Cokernel, CokernelMap
- Kernel, KernelMap
- DefectOfHoms, DefectOfHomsMap
- DirectSum, DirectSumMap
- Pullback, PullbackMap
- TorsionSubmodule, TorsionSubmoduleMap
- TensorProduct, TensorProductMap, TensorProduct2Map
- Hom_R, HomMap_R
- Hom, HomMap, Hom2Map

the use of FunctorMap in homalg

```
‘homalg/HomMap‘ := proc(M,A,N,L,var::list,RingPackage) ### Main: "Func
  local RP, nar, optional, cofunctor;

  RP := ‘homalg/tablename‘(procname,args[-1],nar);

  if type(RP,procedure) then RETURN(RP(args[1..-1-nar])) fi;

  optional := args[6..-1-nar];

  #=====# begin of the core procedure #=====#

  cofunctor := proc(M) ‘homalg/Hom‘(M,L,args[1+1..-1]) end;

  ‘homalg/FunctorMap‘(cofunctor,
    [M], A, [N]
  ,var,optional,RP)
end:
```

the use of FunctorMap in homalg

```
'homalg/Hom2Map' := proc(L,M,A,N,var::list,RingPackage) ### Main: "Fun
  local RP, nar, optional, functor;

  RP := 'homalg/tablename'(procname,args[-1],nar);

  if type(RP,procedure) then RETURN(RP(args[1..-1-nar])) fi;

  optional := args[6..-1-nar];

  #=====# begin of the core procedure #=====#

  functor := proc(M) 'homalg/Hom'(L,M,args[1+1..-1]) end;

  'homalg/FunctorMap'(functor,
    [M], A, [N]
  ,var,optional,RP)
end:
```

The bi-functor Functor_Hom in homalg

```
‘homalg/Functor_Hom‘ := proc(B) ### Main: "Functors"

  [ proc(M) ‘homalg/Hom‘(M,B,args[1+1..-1]) end,
    proc(M,phi,N) ‘homalg/HomMap‘(M,phi,N,B,args[3+1..-1]) end,
    ## take care of multi-functoriality:
    B,
    proc(P)
      proc(M,phi,N) ‘homalg/Hom2Map‘(P,M,phi,N,args[3+1..-1]) end
    end ]
end:
```

The bi-functor Functor_Hom2 in homalg

```
‘homalg/Functor_Hom2‘ := proc(A) ### Main: "Functors"

  [ proc(M) ‘homalg/Hom‘(A,M,args[1+1..-1]) end,
    proc(M,phi,N) ‘homalg/Hom2Map‘(A,M,phi,N,args[3+1..-1]) end,
    ## take care of multi-functoriality:
    A,
    proc(P)
      proc(M,phi,N) ‘homalg/HomMap‘(M,phi,N,P,args[3+1..-1]) end
    end ]
end:
```

the use of CofunctorOnSeqs in homalg

```
`homalg/HomOnSeqs` := proc(L,alpha::{table,list},_var,RingPackage) ###
  local RP, nar, cofunctor;

  RP := `homalg/tablename`(procname,args[-1],nar);

  if type(RP,procedure) then RETURN(RP(args[1..-1-nar])) fi;

  #=====# begin of the core procedure #=====#

  cofunctor := `homalg/Functor_Hom`(L);

  `homalg/CofunctorOnSeqs`(cofunctor,args[1+1..-1])
end:
```

the use of FunctorOnSeqs in homalg

```
`homalg/Hom2OnSeqs` := proc(L,alpha::{table,list},_var,RingPackage) ##
  local RP, nar, functor;

  RP := `homalg/tablename`(procname,args[-1],nar);

  if type(RP,procedure) then RETURN(RP(args[1..-1-nar])) fi;

  #=====# begin of the core procedure #=====#

  functor := `homalg/Functor_Hom2`(L);

  `homalg/FunctorOnSeqs`(functor,args[1+1..-1])
end:
```

composed functors in homalg

using ComposeFunctors we have the following *(multi)-functors*:

- HomHom_R, HomHomMap_R
- HomHom, HomHomMap, HomHom2Map, HomHom3Map

the use of ComposeFunctors in homalg

```
'homalg/HomHom' := proc(M,A,B,var::list,RingPackage) ### Main: "Functors"
  local RP, nar, optional, cofunctor1, cofunctor2;

  RP := 'homalg/tablename'(procname,args[-1],nar);

  if type(RP,procedure) then RETURN(RP(args[1..-1-nar])) fi;

  optional := args[5..-1-nar];

  #======# begin of the core procedure #======#

  cofunctor1 := 'homalg/Functor_Hom'(B);

  cofunctor2 := 'homalg/Functor_Hom'(A);

  eval(
    'homalg/ComposeFunctors'(cofunctor1,cofunctor2,var,optional,RP)(M,var,RP)
  )
end:
```

HomHomMap using FunctorMap

```
‘homalg/HomHomMap‘ := proc(M,phi,N,A,B,var::list,RingPackage) ### Main
  local RP, nar, optional, functor;

  RP := ‘homalg/tablename‘(procname,args[-1],nar);

  if type(RP,procedure) then RETURN(RP(args[1..-1-nar])) fi;

  optional := args[7..-1-nar];

  #=====# begin of the core procedure #=====#

  functor := proc(M) ‘homalg/HomHom‘(M,A,B,args[1+1..-1]) end;

  ‘homalg/FunctorMap‘(functor,
    [M], phi, [N]
  ,var,optional,RP)
end:
```

the tri-functor Functor_HomHom in homalg

```
#=======================================================================
# The functor Hom(Hom(-,A),B)
#_______________________________________________________________________
`homalg/Functor_HomHom` := proc(A,B) ### Main: "Functors"

  [ proc(M) `homalg/HomHom`(M,A,B,args[1+1..-1]) end,
    proc(M,phi,N) `homalg/HomHomMap`(M,phi,N,A,B,args[3+1..-1]) end,
    ## take care of multi-functoriality:
    A,
    proc(P)
      proc(M,phi,N) `homalg/HomHom2Map`(P,M,phi,N,B,args[3+1..-1]) end
    end,
    B,
    proc(P)
      proc(M,phi,N) `homalg/HomHom3Map`(P,A,M,phi,N,args[3+1..-1]) end
    end ]
end:
```

right derived cofunctors in homalg

using `RightDerivedLeftExactCofunctor` we have the following *multi-functor*:

- `Ext`, `ExtMap`, `Ext2Map`

right derived cofunctors in homalg

using `RightDerivedLeftExactCofunctor` we have the following *multi-functor*:

- `Ext`, `ExtMap`, `Ext2Map`

using `RightDerivedCofunctor` we have the following *multi-functors*:

- `Ext_R`, `ExtMap_R`
- `Extq`, `ExtqMap`, `Extq2Map`
- `RHomHom`, `RHomHomMap`, `RHomHom2Map`, `RHomHom3Map`

definition of the $\mathrm{Ext}^i(-,D)$-modules

take a free resolution of M:

$$\ldots \xrightarrow{d_3} D^r \xrightarrow{d_2} D^q \xrightarrow{d_1} D^p \longrightarrow M \longrightarrow 0$$

definition of the $\mathrm{Ext}^i(-,D)$-modules

take a free resolution of M:

$$\dots \xrightarrow{d_3} D^r \xrightarrow{d_2} D^q \xrightarrow{d_1} D^p \longrightarrow M \longrightarrow 0$$

apply the functor $\mathrm{Hom}_D(-,D)$:

$$\dots \xleftarrow{d_3^*} \mathrm{Hom}_D(D^r,D) \xleftarrow{d_2^*} \mathrm{Hom}_D(D^q,D) \xleftarrow{d_1^*} \mathrm{Hom}_D(D^p,D) \leftarrow 0$$

definition of the $\mathrm{Ext}^i(-,D)$-modules

take a free resolution of M:

$$\ldots \xrightarrow{d_3} D^r \xrightarrow{d_2} D^q \xrightarrow{d_1} D^p \longrightarrow M \longrightarrow 0$$

apply the functor $\mathrm{Hom}_D(-,D)$:

$$\ldots \xleftarrow{d_3^*} \mathrm{Hom}_D(D^r,D) \xleftarrow{d_2^*} \mathrm{Hom}_D(D^q,D) \xleftarrow{d_1^*} \mathrm{Hom}_D(D^p,D) \leftarrow 0$$

define

$$\begin{aligned}
\mathrm{Ext}^0_D(M,D) &:= \ker(d_1^*) = \mathrm{Hom}_D(M,D), \\
\mathrm{Ext}^i_D(M,D) &:= \ker(d_{i+1}^*)/\operatorname{im}(d_i^*), \qquad i \geq 1.
\end{aligned}$$

definition of the $\mathrm{Ext}^i(-, D)$-modules

take a free resolution of M:

$$\ldots \xrightarrow{d_3} D^r \xrightarrow{d_2} D^q \xrightarrow{d_1} D^p \longrightarrow M \longrightarrow 0$$

apply the functor $\mathrm{Hom}_D(-, D)$:

$$\ldots \xleftarrow{d_3^*} \mathrm{Hom}_D(D^r, D) \xleftarrow{d_2^*} \mathrm{Hom}_D(D^q, D) \xleftarrow{d_1^*} \mathrm{Hom}_D(D^p, D) \leftarrow 0$$

define

$$\begin{aligned} \mathrm{Ext}_D^0(M, D) &:= \ker(d_1^*) = \mathrm{Hom}_D(M, D), \\ \mathrm{Ext}_D^i(M, D) &:= \ker(d_{i+1}^*)/\mathrm{im}(d_i^*), \qquad i \geq 1. \end{aligned}$$

Theorem

$\mathrm{Ext}_D^i(M, D)$ *is independent of the choice of the projective resolution of* M.

the use of RightDerivedLeftExactCofunctor

```
`homalg/Ext` := proc(q::nonnegint,A,B,var::list,RingPackage) ### Main:
  local RP, nar, optional, cofunctor;

  RP := `homalg/tablename`(procname,args[-1],nar);

  if type(RP,procedure) then RETURN(RP(args[1..-1-nar])) fi;

  optional := args[5..-1-nar];

  #=====# begin of the core procedure #=====#

  cofunctor := `homalg/Functor_Hom`(B);

  `homalg/RightDerivedLeftExactCofunctor`(
    q,cofunctor,A
  ,var,optional,RP)
end:
```

the use of RightDerivedCofunctor in homalg

```
‘homalg/Extq‘ := proc(q::nonnegint,A,B,var::list,RingPackage) ### Main
  local RP, nar, optional, cofunctor;

  RP := ‘homalg/tablename‘(procname,args[-1],nar);

  if type(RP,procedure) then RETURN(RP(args[1..-1-nar])) fi;

  optional := args[5..-1-nar];

  #=====# begin of the core procedure #=====#

  cofunctor := ‘homalg/Functor_Hom‘(B);

  ‘homalg/RightDerivedCofunctor‘(
    q,cofunctor,A
  ,var,optional,RP)
end:
```

linear control theory

characterizing system/module properties

system	module	homological algebra
autonomous elements	$t(M) \neq 0$	$\mathrm{Ext}^1_D(M^\top, D) \neq 0$
controllability, parametrizability	$t(M) = 0$	$\mathrm{Ext}^1_D(M^\top, D) = 0$
parametrizability of the parametrization	reflexive	$\mathrm{Ext}^i_D(M^\top, D) = 0$, $i = 1, 2$
...	...	...
int. stabilizability, BÉZOUT-itdentity, chain of n parametrizations	projective	$\mathrm{Ext}^i_D(M^\top, D) = 0$, $1 \leq i \leq n$
flatness	free	in general no criteria but for PIR: torsion free = free

left derived functors in homalg

using `LeftDerivedRightExactFunctor` we have the following *multi-functor*:

- `Tor`, `TorMap`, `Tor2Map`

left derived functors in homalg

using `LeftDerivedRightExactFunctor` we have the following *multi-functor*:

- `Tor`, `TorMap`, `Tor2Map`

using `LeftDerivedFunctor` we have the following *multi-functors*:

- `Torq`, `TorqMap`, `Torq2Map`
- `LHom`, `LHomMap`, `LHom2Map`
- `LHomHom_R`, `LHomHomMap_R`
- `LHomHom`, `LHomHomMap`, `LHomHom2Map`, `LHomHom3Map`

the use of LeftDerivedFunctor in homalg

```
‘homalg/LHomHom‘ := proc(q::nonnegint,M,A,B,var::list,RingPackage) ###
  local RP, nar, optional, functor;

  RP := ‘homalg/tablename‘(procname,args[-1],nar);

  if type(RP,procedure) then RETURN(RP(args[1..-1-nar])) fi;

  optional := args[6..-1-nar];

  #=====# begin of the core procedure #=====#

  functor := ‘homalg/Functor_HomHom‘(A,B);

  ‘homalg/LeftDerivedFunctor‘(
    q,functor,M
  ,var,optional,RP)
end:
```

LHomHomMap using FunctorMap

```
`homalg/LHomHomMap` := proc(q,M,phi,N,A,B,var::list,RingPackage) ### M
  local RP, nar, optional, functor;

  RP := `homalg/tablename`(procname,args[-1],nar);

  if type(RP,procedure) then RETURN(RP(args[1..-1-nar])) fi;

  optional := args[8..-1-nar];

  #=====# begin of the core procedure #=====#

  functor := proc(M) `homalg/LHomHom`(q,M,A,B,args[1+1..-1]) end;

  `homalg/FunctorMap`(functor,
    [M], phi, [N]
  ,var,optional,RP)
end:
```

the tri-functor LHomHom in homalg

```
#==========================================================================
# The functor LHomHom_q(-,A,B)
#__________________________________________________________________________
`homalg/Functor_LHomHom` := proc(q,A,B) ### Main: "Functors"

  [ proc(M) `homalg/LHomHom`(q,M,A,B,args[1+1..-1]) end,
    proc(M,phi,N) `homalg/LHomHomMap`(q,M,phi,N,A,B,args[3+1..-1]) end,
    ## take care of multi-functoriality:
    A,
    proc(P)
      proc(M,phi,N) `homalg/LHomHom2Map`(q,P,M,phi,N,B,args[3+1..-1]) end
    end,
    B,
    proc(P)
      proc(M,phi,N) `homalg/LHomHom3Map`(q,P,A,M,phi,N,args[3+1..-1]) end
    end ]
end:
```

natural transformations in homalg

we also have the following *natural transformations*:

- NatTrTargetToCokernel=CokernelEpi
- NatTrKernelToSource=KernelEmb
- NatTrTorsionSubmoduleToId=TorsionSubmoduleEmb
- NatTrPullbackToTwoSources=PullbackPairOfMaps
- NatTrIdToHomHom_R
- NatIsoKerOfSqToImOfSq=Lambek

Outline

features of the ring package PIR

- R is one of the fields $\{\mathbb{F}_p, \mathbb{Q}\}$, or one of the Euclidean domains (and hence principal ideal domains) $\{\mathbb{Z}, \mathbb{Z}[I], \mathbb{Q}[x], \mathbb{F}_p[x]\}$ or one of its quotients, which are in general only principle ideal rings

features of the ring package PIR

- R is one of the fields $\{\mathbb{F}_p, \mathbb{Q}\}$, or one of the Euclidean domains (and hence principal ideal domains) $\{\mathbb{Z}, \mathbb{Z}[I], \mathbb{Q}[x], \mathbb{F}_p[x]\}$ or one of its quotients, which are in general only principle ideal rings
- the global dimension of the non-trivial principal ideal domains is 1, that of their semi-simple quotients is 0 and that of their non-semi-simple quotients is ∞

features of the ring package PIR

- R is one of the fields $\{\mathbb{F}_p, \mathbb{Q}\}$, or one of the Euclidean domains (and hence principal ideal domains) $\{\mathbb{Z}, \mathbb{Z}[I], \mathbb{Q}[x], \mathbb{F}_p[x]\}$ or one of its quotients, which are in general only principle ideal rings
- the global dimension of the non-trivial principal ideal domains is 1, that of their semi-simple quotients is 0 and that of their non-semi-simple quotients is ∞
- essentially a sample ring package:
  homalg provides extras to easily use packages for principal ideal rings

features of the ring package PIR

- the triangular basis is given by the HERMITE normal form

features of the ring package PIR

- the triangular basis is given by the HERMITE normal form
- the SMITH normal form (elementary divisors) provides a normal form for a presentation of a module

features of the ring package PIR

- the triangular basis is given by the HERMITE normal form
- the SMITH normal form (elementary divisors) provides a normal form for a presentation of a module
- applications to algebraic topology and CONLEY-index-theory

the homalg translation table of PIR

```
`PIR/homalg`:=table(
  [ ## Must only then be provided by the RingPackage in case the default
    ## "service" function does not match the Ring
    GlobalDim=`PIR/PGlobalDim`,
    PresentationInfo=
      proc(M,var) `homalg/DiagonalElementsAndRank`(M,var,'PIR') end,

    ## Can optionally be provided by the RingPackage
    ## (homalg functions check if these functions are defined or not)
    ## (`homalg/tablename` gives no default value)
    BestBasis=`PIR/PBestBasis`,
    RingElementNormalForm=`PIR/PRingElementNormalForm`,

    ## Must be defined if other functions are not defined
    TriangularBasis=`PIR/PTriangularBasis`, ## needed by `homalg/BasisOfModule`
    QuotientWithRemainder=`PIR/PQuo` ## needed by `homalg/Reduce`

    ## Must only then be provided by the RingPackage in case the default
    ## value provided by `homalg/tablename` does not match the Ring
  ]):
```

features of the ring package Involutive

- $R = K[x_1, \ldots, x_n]$, where $K \geq \mathbb{Q}$ or $K = \mathbb{Z}/p\mathbb{Z}$

features of the ring package Involutive

- $R = K[x_1, \ldots, x_n]$, where $K \geq \mathbb{Q}$ or $K = \mathbb{Z}/p\mathbb{Z}$
- the basis is given by the involutive basis

features of the ring package Involutive

- $R = K[x_1, \ldots, x_n]$, where $K \geq \mathbb{Q}$ or $K = \mathbb{Z}/p\mathbb{Z}$
- the basis is given by the involutive basis
- application to commutative algebra and algebraic geometry

the homalg conversion table of Involutive

```
'Involutive/homalg':=table(
  [
    ## Must only then be provided by the RingPackage in case the default
    ## "service" function does not match the Ring
    BasisOfModule='homalg/Involutive/IBasis',
    PresentationInfo='homalg/Involutive/PolHilbertCartan',
    Reduce='homalg/Involutive/PolInvoReduce',
    SimplifyBasis='Involutive/jetsdepcheck',
    SyzygiesGenerators='homalg/Involutive/PolSyzygies'
  ]):
```

features of the ring package Janet

- $R = K[\partial_1, \ldots, \partial_n]$, where K is a differential field
- the basis is given by the JANET basis

features of the ring package Janet

- $R = K[\partial_1, \ldots, \partial_n]$, where K is a differential field
- the basis is given by the JANET basis
- in the univariate case (ODEs) the JACOBSON normal form (with just one elementary divisor!) provides a normal form for a presentation of a module: R is then a left principal ideal ring

features of the ring package Janet

- $R = K[\partial_1, \ldots, \partial_n]$, where K is a differential field
- the basis is given by the JANET basis
- in the univariate case (ODEs) the JACOBSON normal form (with just one elementary divisor!) provides a normal form for a presentation of a module: R is then a left principal ideal ring
- application to linear control theory

the homalg conversion table of Janet

```
`Janet/homalg`:=table(
  [
    ## Must only then be provided by the RingPackage in case the default
    ## "service" function does not match the Ring
    AddMat=`Janet/jaddmat`,
    BasisOfModule=`homalg/Janet/JBasis`,
    Compose=`Janet/CmpOp`,
    Involution=`homalg/Janet/Involution`,
    MulMat=`Janet/jmulmat`,
    PresentationInfo=`homalg/Janet/JanHilbertCartan`,
    Reduce=`homalg/Janet/InvoReduce`,
    SimplifyBasis=`homalg/Janet/Jandepcheck`,
    SyzygiesGenerators=`homalg/Janet/Syzygies`,
    SubMat=`Janet/jsubmat`,

    ## Can optionally be provided by the RingPackage
    ## (homalg functions check if these functions are defined or not)
    ## (`homalg/tablename` gives no default value)
    IsRingElement=proc(a) `Janet/jchkdop`(a,"") end,
```

the homalg conversion table of Janet

```
  ## Must only then be provided by the RingPackage in case the default
  ## value provided by ‘homalg/tablename‘ does not match the Ring
  DivideByUnit=‘Janet/jdividebyunit‘,
  matrix=‘Janet/jkmat‘,
  Minus=‘Janet/jsubcon‘,
  One=[[1, []]]
]):
```

the (pseudo) package Janet1

```
‘Janet1/homalg‘:=table(
  [
    ## Must only then be provided by the RingPackage in case the default
    ## "service" function does not match the Ring
    AddMat=‘Janet/jaddmat‘,
    BasisOfModule=‘homalg/Janet/JBasis‘,
    Compose=‘Janet/CmpOp‘,
    Involution=‘homalg/Janet/Involution‘,
    MulMat=‘Janet/jmulmat‘,
    PresentationInfo=‘homalg/Janet/JanHilbertCartan‘,
    Reduce=‘homalg/Janet/InvoReduce‘,
    SimplifyBasis=‘homalg/Janet/Jandepcheck‘,
    SyzygiesGenerators=‘homalg/Janet/Syzygies‘,
    SubMat=‘Janet/jsubmat‘,

    ## Can optionally be provided by the RingPackage
    ## (homalg functions check if these functions are defined or not)
    ## (‘homalg/tablename‘ gives no default value)
    BestBasis=‘homalg/Janet/Jacobson‘,
    IsRingElement=proc(a) ‘Janet/jchkdop‘(a,"") end,
```

the (pseudo) ring package Janet1

```
  ## Must only then be provided by the RingPackage in case the default
  ## value provided by ‘homalg/tablename‘ does not match the Ring
  DivideByUnit=‘Janet/jdividebyunit‘,
  matrix=‘Janet/jmkmat‘,
  Minus=‘Janet/jsubcon‘,
  One=[[1, []]]
]):
```

features of the ring package JanetOre

- R is an ORE-domain

features of the ring package JanetOre

- R is an ORE-domain
- the basis is given by the involutive basis

features of the ring package JanetOre

- R is an ORE-domain
- the basis is given by the involutive basis
- application to linear control theory

features of the ring package JanetOre

- R is an ORE-domain
- the basis is given by the involutive basis
- application to linear control theory
- using the BGG correspondence to compute the cohomology of coherent sheaves over projective spaces

the homalg conversion table of JanetOre

```
`JanetOre/homalg`:=table(
  [
    ## Must only then be provided by the RingPackage in case the default
    ## "service" function does not match the Ring
    BasisOfModule=`homalg/JanetOre/Basis`,
    Compose=`homalg/JanetOre/Mult`,
    Involution=`homalg/JanetOre/Involution`,
    PresentationInfo=`homalg/JanetOre/HilbertCartan`,
    Reduce=`homalg/JanetOre/Reduce`,
    SimplifyBasis=`JanetOre/jets_depcheck`,
    SyzygiesGenerators=`homalg/JanetOre/Syzygies`,
    IsUnit=`homalg/JanetOre/IsUnit`
  ]):
```

features of the ring package OreModules

- R is an ORE-domain

features of the ring package OreModules

- R is an ORE-domain
- the basis is given by the GRÖBNER basis

features of the ring package `OreModules`

- R is an ORE-domain
- the basis is given by the GRÖBNER basis
- application to linear control theory

features of the ring package OreModules

- R is an Ore-domain
- the basis is given by the Gröbner basis
- application to linear control theory

the homalg conversion table of OreModules

```
`OreModules/homalg`:=table(
  [
    ## Must only then be provided by the RingPackage in case the default
    ## "service" function does not match the Ring
    BasisOfModule=`homalg/OreModules/Basis`,
    Compose=`OreModules/Mult`,
    Involution=`homalg/OreModules/Involution`,
    PresentationInfo=`homalg/OreModules/HilbertSeries`,
    Reduce=`homalg/OreModules/Reduce`,
    SimplifyBasis=`homalg/OreModules/jetsdepcheck`,
    SyzygiesGenerators=`homalg/OreModules/SyzygiesGen`
  ]):
```

Outline

todo

- applications: topology, cohomology of groups, number theory, control theory ...

todo

- applications: topology, cohomology of groups, number theory, control theory ...
- spectral sequences (simply more applications)

todo

- applications: topology, cohomology of groups, number theory, control theory ...
- spectral sequences (simply more applications)
- create and link more ring packages to homalg

todo

- applications: topology, cohomology of groups, number theory, control theory ...
- spectral sequences (simply more applications)
- create and link more ring packages to homalg
- handle all functors on an equal footing, regardless of the (source and) target abelian category

todo

- applications: topology, cohomology of groups, number theory, control theory ...
- spectral sequences (simply more applications)
- create and link more ring packages to homalg
- handle all functors on an equal footing, regardless of the (source and) target abelian category
- use `homalg` with other CASs: Singular, Kash, GAP, ...