SpasmInterface

A package to communicate with the software Spasm

2020.01.20

20 January 2020

Martin Bies

Martin Bies
Email: martin.bies@alumni.uni-heidelberg.de
Homepage: https://martinbies.github.io/
Address: Mathematical Institute
University of Oxford
Andrew Wiles Building
Radcliffe Observatory Quarter
Woodstock Road
Oxford OX2 6GG
United Kingdom

Contents

Chapter 1

Introduction

1.1 What is the goal of the SpasmInterface package?

SpasmInterface provides an interface, to communicate with the software Spasm via gap. Thereby, for example, kernel of sparse matrices can be computed directly via gap.

Chapter 2

Interface to Spasm

2.1 Finding the SpasmDirectory

2.1.1 FindSpasmDirectory

▷ `FindSpasmDirectory(`*`none`*`)` (operation)

Returns: the corresponding directory

This operation identifies the location of the spasm programs.

2.1.2 SetSpasmDirectory (for IsString)

▷ `SetSpasmDirectory(`*`none`*`)` (operation)

Returns: the corresponding directory

This operation sets and stores the location of the spasm programs. In future operations, the method FindSpasmDirectory will use this path. If you wish to alter this path later, call this function again with the new path. If for example Spasm is installed at /home/person/spasm then the input for this function should be "/home/person/spasm/bench".

2.2 Executing Spasm

2.2.1 ExecuteSpasm (for IsDirectory, IsString, IsList, IsList, IsList)

▷ `ExecuteSpasm(`*`A, Directory, a, string, and, two, lists`*`)` (operation)

Returns: the corresponding quantity as computed by Spasm as a string

This operation executes Spasm with four pieces of input information. The first is the directory of Spasm, the second the name of the binary that is to be executed, the third is the input required by this binary. The fourth argument is a list of options supported by Spasm and the fifth the list of values for these arguments.

Chapter 3

Functionality of Spasm

3.1 Elementary manipulations of SMSSparseMatrices

3.1.1 TurnIntoSMSString (for IsSMSSparseMatrix)

▷ `TurnIntoSMSString(`*`SMSSparseMatrix, M`*`)` (operation)

Returns: String

Turn an SMSSparseMatrix M into a string used as input for Spasm

3.1.2 UnionOfRowsOp (for IsSMSSparseMatrix, IsSMSSparseMatrix)

▷ `UnionOfRowsOp(`*`SMSSparseMatrices, M1, M2`*`)` (operation)

Returns: SMSSparseMatrix

This functions forms the union of the rows of two SMSSparseMatrices M1, M2.

3.1.3 UnionOfColumnsOp (for IsSMSSparseMatrix, IsSMSSparseMatrix)

▷ `UnionOfColumnsOp(`*`SMSSparseMatrices, M1, M2`*`)` (operation)

Returns: SMSSparseMatrix

This functions forms the union of the columns of two SMSSparseMatrices M1, M2.

3.1.4 Involution (for IsSMSSparseMatrix)

▷ `Involution(`*`SparseMatrix, M`*`)` (operation)

Returns: SMSSparseMatrix

This function forms the involution or transposition of a SMSSparseMatrix M. Note that this operation is always performed modulo 42013 in spasm.

3.1.5 CertainRows (for IsSMSSparseMatrix, IsList)

▷ `CertainRows(`*`SparseMatrix, M, List, of, indices, L`*`)` (operation)

Returns: SMSSparseMatrix

This function forms a new SMSSparseMatrix from all rows of an SMSSparseMatrix M whose index is contained in the list L

3.1.6 CertainColumns (for IsSMSSparseMatrix, IsList)

▷ `CertainColumns(`*`SparseMatrix, M, List, of, indices, L`*`)` (operation)

Returns: SMSSparseMatrix

This function forms a new SMSSparseMatrix from all columns of an SMSSparseMatrix M whose index is contained in the list L

3.1.7 SumOfColumns (for IsSMSSparseMatrix)

▷ `SumOfColumns(`*`SparseMatrix, M`*`)` (operation)

Returns: List

This function adds the columns of a given SMSSparseMatrix

3.1.8 SumOfRows (for IsSMSSparseMatrix)

▷ `SumOfRows(`*`SparseMatrix, M`*`)` (operation)

Returns: List

This function adds the rows of a given SMSSparseMatrix

3.1.9 SumEntriesOfSomeColumns (for IsSMSSparseMatrix, IsInt)

▷ `SumEntriesOfSomeColumns(`*`SparseMatrix, M, integer, N`*`)` (operation)

Returns: List

This function picks N columns a given SMSSparseMatrix by random and sums the absolute values of their entries.

3.1.10 SumEntriesOfSomeRows (for IsSMSSparseMatrix, IsInt)

▷ `SumEntriesOfSomeRows(`*`SparseMatrix, M, integer, N`*`)` (operation)

Returns: List

This function picks N rows a given SMSSparseMatrix by random and sums the absolute values of their entries.

3.2 Attributes of SMSSparseMatrices

3.2.1 NonZeroRows (for IsSMSSparseMatrix)

▷ `NonZeroRows(`*`SparseMatrix, M`*`)` (attribute)

Returns: List

This function accepts a SMSSparseMatrix M and finds all non-zero rows of this matrix.

3.2.2 NonZeroColumns (for IsSMSSparseMatrix)

▷ `NonZeroColumns(`*`SparseMatrix, M`*`)` (attribute)

Returns: List

This function accepts a SMSSparseMatrix M and finds all non-zero columns of this matrix.

3.3 Computation of syzygies

3.3.1 SyzygiesOfRowsBySpasm (for IsSMSSparseMatrix)

▷ SyzygiesOfRowsBySpasm(*SMSSparseMatrix, M*) (operation)

Returns: SMSSparseMatrix

Compute left kernel of SMSSparseMatrix M by Spasm. By default we compute it over the finite field of order 42013. As second argument, an integer can be provided to overwrite this default.

3.3.2 SyzygiesOfRowsBySpasm (for IsSMSSparseMatrix, IsInt)

▷ SyzygiesOfRowsBySpasm(*arg1, arg2*) (operation)

3.3.3 SyzygiesOfColumnsBySpasm (for IsSMSSparseMatrix)

▷ SyzygiesOfColumnsBySpasm(*SMSSparseMatrix, M*) (operation)

Returns: SMSSparseMatrix

Compute right kernel of SMSSparseMatrix M by Spasm. By default we compute it over the finite field of order 42013. As second argument, an integer can be provided to overwrite this default.

3.3.4 SyzygiesOfColumnsBySpasm (for IsSMSSparseMatrix, IsInt)

▷ SyzygiesOfColumnsBySpasm(*arg1, arg2*) (operation)

3.3.5 RowSyzygiesGenerators (for IsSMSSparseMatrix, IsSMSSparseMatrix)

▷ RowSyzygiesGenerators(*SparseMatrices, M1, M2*) (operation)

Returns: SMSSparseMatrix

This function computes the RowSyzygyGenerators of two SMSSparseMatrices M1, M2. By default we compute it over the finite field of order 42013. As second argument, an integer can be provided to overwrite this default.

3.3.6 RowSyzygiesGenerators (for IsSMSSparseMatrix, IsSMSSparseMatrix, IsInt)

▷ RowSyzygiesGenerators(*arg1, arg2, arg3*) (operation)

3.3.7 ColumnSyzygiesGenerators (for IsSMSSparseMatrix, IsSMSSparseMatrix)

▷ ColumnSyzygiesGenerators(*SparseMatrices, M1, M2*) (operation)

Returns: SMSSparseMatrix

This function computes the ColumnSyzygyGenerators of two SMSSparseMatrices M1, M2. By default we compute it over the finite field of order 42013. As second argument, an integer can be provided to overwrite this default.

3.3.8 ColumnSyzygiesGenerators (for IsSMSSparseMatrix, IsSMSSparseMatrix, IsInt)

▷ ColumnSyzygiesGenerators(*arg1, arg2, arg3*) (operation)

3.4 Computation of rank

3.4.1 RankGPLUBySpasm (for IsSMSSparseMatrix)

▷ RankGPLUBySpasm(*SMSSparseMatrix, M*) (operation)

Returns: Integer

Compute the rank of an SMSSparseMatrix M by Spasm by GPLU. By default we compute it over the finite field of order 42013. As second argument, an integer can be provided to overwrite this default.

3.4.2 RankGPLUBySpasm (for IsSMSSparseMatrix, IsInt)

▷ RankGPLUBySpasm(*arg1, arg2*) (operation)

3.4.3 RankDenseBySpasm (for IsSMSSparseMatrix)

▷ RankDenseBySpasm(*SMSSparseMatrix, M*) (operation)

Returns: Integer

Compute the rank of an SMSSparseMatrix M by Spasm. This uses an algorithm designed for handeling dense matrices, but is here applied to a sparse matrix nonetheless. By default we compute it over the finite field of order 42013. As second argument, an integer can be provided to overwrite this default.

3.4.4 RankDenseBySpasm (for IsSMSSparseMatrix, IsInt)

▷ RankDenseBySpasm(*arg1, arg2*) (operation)

3.4.5 RankHybridBySpasm (for IsSMSSparseMatrix)

▷ RankHybridBySpasm(*SMSSparseMatrix, M*) (operation)

Returns: Integer

Compute the rank of an SMSSparseMatrix M by Spasm. This uses the hybrid strategy described in [PASCO'17]. By default we compute it over the finite field of order 42013. As second argument, an integer can be provided to overwrite this default.

3.4.6 RankHybridBySpasm (for IsSMSSparseMatrix, IsInt)

▷ RankHybridBySpasm(*arg1, arg2*) (operation)

3.5 Examples

We can create sparse matrices and compute their syzygies of rows, i.e. their left kernels as follows:

Example

```
gap> entries1 := [ [ 1, 1, 1 ], [ 2, 1, -1 ], [ 3, 2, 1 ] ];;
gap> m1 := SMSSparseMatrix( 3, 2, entries1 );;
gap> NumberOfRows( m1 );
3
gap> NumberOfColumns( m1 );
2
gap> Entries( m1 );
[ [ 1, 1, 1 ], [ 2, 1, -1 ], [ 3, 2, 1 ] ]
gap> s1 := TurnIntoSMSString( m1 );;
gap> k1 := SyzygiesOfRowsBySpasm( m1 );;
gap> SyzygiesOfColumnsBySpasm( m1 );;
gap> NumberOfRows( k1 );
1
gap> NumberOfColumns( k1 );
3
```

Here is another example.

Example

```
gap> entries2 := [ [ 1, 2, 1 ], [ 2, 1, 1 ], [ 2, 2, 1 ], [ 3, 1, -1 ], [ 3, 2, 1 ] ];;
gap> m2 := SMSSparseMatrix( 3, 2, entries2 );;
gap> NumberOfRows( m2 );
3
gap> NumberOfColumns( m2 );
2
gap> s2 := TurnIntoSMSString( m2 );;
gap> k2 := SyzygiesOfRowsBySpasm( m2 );;
gap> SyzygiesOfColumnsBySpasm( m2 );;
gap> NumberOfRows( k2 );
1
gap> NumberOfColumns( k2 );
3
```

Given two sparse matrices, we can stack them in that we take the collection formed from their union of rows and interpret the result as a new sparse matrix. This is also used to compute the relative syzygies of a matrix $m1$ with respect to a second matrix $m2$. We demonstrate this with the above two matrices:

Example

```
gap> m3 := UnionOfRowsOp( m1, m2 );;
gap> NumberOfRows( m3 );
6
gap> NumberOfColumns( m3 );
2
gap> m4 := RowSyzygiesGenerators( m1, m2 );
<A 4x3 sparse matrix in SMS-format>
gap> ColumnSyzygiesGenerators( m1, m2 );
<A 2x1 sparse matrix in SMS-format>
gap> NumberOfRows( m4 );
4
gap> NumberOfColumns( m4 );
3
```

We can also pick certain rows and columns and compute the transposed matrix.

Example

```
gap> m5 := CertainRows( m3, [ 1 ] );
<A 1x2 sparse matrix in SMS-format>
gap> m6 := CertainColumns( m3, [ 2 ] );
<A 6x1 sparse matrix in SMS-format>
gap> m7 := Involution( m6 );
<A 1x6 sparse matrix in SMS-format>
```

Let us now create a matrix with trivial columns and rows. We can identify and strip these. Also, after minor modifications, they are taken into account by Spasm when computing kernels.

Example

```
gap> entries3 := [ [ 1, 1, 1 ], [ 1, 2, 1 ], [ 1, 3, -1 ], [ 3, 2, 1 ] ];;
gap> m8 := SMSSparseMatrix( 3, 4, entries3 );
<A 3x4 sparse matrix in SMS-format>
gap> m9 := NonZeroRows( m8 );
[ 1,3 ]
gap> m10 := NonZeroColumns( m8 );
[ 1,2,3 ]
gap> m11 := SyzygiesOfRowsBySpasm( m8 );
<A 1x3 sparse matrix in SMS-format>
```

Also transposition of sparse matrices is supported by spasms. As of this writing, this will always be performed modulo 42013.

Example

```
gap> m12 := Involution( m1 );
<A 2x3 sparse matrix in SMS-format>
```

We can also form sums of rows and columns. Here are some examples

Example

```
gap> SumOfRows( m2 );
[ 0, 3 ]
gap> SumOfColumns( m2 );
[ 1, 2, 0 ]
gap> SumEntriesOfSomeRows( m2, 2 );;
gap> SumEntriesOfSomeColumns( m2, 2 );;
```

Spasm also supports various algorithms for computing ranks of sparse matrices. Here are some examples

Example

```
gap> RankGPLUBySpasm( m3 );
2
gap> RankDenseBySpasm( m3 );
2
gap> RankHybridBySpasm( m3 );
2
```

Index