Formatted manual of compcox.lib

1 Singular libraries

1.1 compcox_lib

```
-------BEGIN OF PART WHICH IS INCLUDED IN MANUAL-----
```

Library: compcox.lib

Purpose: Modifying canonically embedded Mori Dream Spaces (CEMDS).

Authors: Ulrich Derenthal, Juergen Hausen, Armand Heim, Simon Keicher, Antonio Laface

Overview: This library provides a framework for modifying canonically embedded Mori Dream Spaces (CEMDS).

Note: Sometimes variables need to be exported. They carry names of the structure g*Result, where * may be any combination of prefixes and types defined below.

Procedures:

1.1.0.1 assignCEMDS

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: cemdsLeft = list(rvcvzP, scnSigma, spG) with rvcvzP: intmat, scnSigma: fan, spG: ideal.

Assume: A basering is defined.

Purpose: Overrides the standard singular assignment for more comfortable CEMDS assignments.

Return: The CEMDS fetched or composed from the passed parameter(s).

Note: cemdsLeft = cemdsRight; with cemdsRight: CEMDS may also work as soon as Singular allows overriding the standard assignment, too.

1.1.0.2 printCEMDS

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: printCEMDS(cemds); OR print(cemds); OR cemds; with cemds: CEMDS.

Assume: The CEMDS is associated with some ring.

Purpose: Prints the CEMDS's data to stdout.

Return: Nothing.

1.1.0.3 createCEMDS

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: createCEMDS(rvcvzP, scnSigma, spG); with rvcvzP: intmat, scnSigma: fan, spG: ideal.

Assume: A basering with spG local to it is defined. The columns of rvcvzP create the rays of scnSigma.

Purpose: Composes a CEMDS from a matrix of fan's rays, the fan itself and an ideal embedding the MDS in an ambient toric variety.

Return: A CEMDS that is CEMDS defined by the passed parameters.

Example:

```
LIB "compcox.lib";
//A CEMDS representing the projective plane P2
//Define input parameters first
ring R = 0,T(1..3),dp;
intmat rvcvzP[2][3] =
1,0,-1,
0,1,-1;
intmat cvrvz1[2][2] =
1,0,
0,1;
intmat cvrvz2[2][2] =
0,1,
-1,-1;
intmat cvrvz3[2][2] =
-1,-1,
1,0;
cone cn1 = coneViaPoints(cvrvz1);
cone cn2 = coneViaPoints(cvrvz2);
cone cn3 = coneViaPoints(cvrvz3);
fan scnSigma = fanViaCones(cn1, cn2, cn3);
ideal spG = 0;
//Then create a new CEMDS.
CEMDS cemds = createCEMDS(rvcvzP, scnSigma, spG);
print(cemds);
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      1     0    -1
↦      0     1    -1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Dimension 2:
↦ 1st maximal cone:
↦ -1, 0,
↦ -1, 1
↦
↦ 2nd maximal cone:
↦ 1, -1,
↦ 0, -1
↦
↦ 3rd maximal cone:
↦ 1, 0,
↦ 0, 1
↦
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
```

```
↦ 0
//Should yield the CEMDS arising from the parameters passed associated with the ring de-
fined above.
```

1.1.0.4 encodeCEMDS

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: encodeCEMDS(cemds); with cemds: CEMDS.

Assume: The CEMDS is associated with some ring.

Purpose: Prints SINGULAR code defining the passed CEMDS on execution.

Return: Nothing.

Example:

```
LIB "compcox.lib";
//A CEMDS representing the projective plane P2
//Define input parameters first
ring R = (0,a),T(1..3),dp;
intmat rvcvzP[2][3] =
1,0,-1,
0,1,-1;
intmat cvrvz1[2][2] =
1,0,
0,1;
intmat cvrvz2[2][2] =
0,1,
-1,-1;
intmat cvrvz3[2][2] =
-1,-1,
1,0;
cone cn1 = coneViaPoints(cvrvz1);
cone cn2 = coneViaPoints(cvrvz2);
cone cn3 = coneViaPoints(cvrvz3);
fan scnSigma = fanViaCones(cn1, cn2, cn3);
ideal spG = 0;
//Then create a new CEMDS.
CEMDS cemds = createCEMDS(rvcvzP, scnSigma, spG);
//Encode the CEMDS. This should yield SINGULAR code defining the CEMDS and thus similar to the code u
encodeCEMDS(cemds);
↦ //---------------START CEMDS ENCODING---------------
↦ ring R = (0,a),(T(1),T(2),T(3)),(dp(3),C);
↦ intmat rvcvzP = intmat(intvec(
↦ 1,0,-1,
↦ 0,1,-1
↦ ), 2, 3);
↦ intmat cvrvz;
↦ list scn = list();
↦ list scnList = list();
↦ cvrvz = intmat(intvec(
↦ -1, -1,
↦  0,  1
↦ ), 2, 2);
↦ scn = coneViaPoints(cvrvz);
↦ scnList = scnList + scn;
↦ cvrvz = intmat(intvec(
↦  1,  0,
```

```
↦ -1, -1
↦ ), 2, 2);
↦ scn = coneViaPoints(cvrvz);
↦ scnList = scnList + scn;
↦ cvrvz = intmat(intvec(
↦ 1, 0,
↦ 0, 1
↦ ), 2, 2);
↦ scn = coneViaPoints(cvrvz);
↦ scnList = scnList + scn;
↦ fan scnSigma = fanViaCones(scnList);
↦ ideal spG =
↦ 0
↦ ;
↦ CEMDS cemds = createCEMDS(rvcvzP, scnSigma, spG);
↦ print("");
↦ print("The CEMDS was successfully imported to the variable \"cemds\"!");
↦ //--------------- END CEMDS ENCODING ---------------
```

1.1.0.5 fetchCEMDS

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: fetchCEMDS(cemdsRight); with cemdsRight: CEMDS.

Assume: A basering is defined. The passed CEMDS is associated with a ring.

Purpose: Associates a CEMDS associated with an arbitrary ring with the current basering, especially fetching the ideal "spG" embedding the MDS in its ambient toric variety.

Return: A CEMDS that is the passed CEMDS associated with the current basering with the ideal "spG" embedding the MDS in its ambient toric variety fetched.

Example:

```
LIB "compcox.lib";
//Create a CEMDS associated with a certain basering
ring R = 0,T(1..5),dp;
intmat rvcvzP[4][5] =
1,0,-1,0,0,
0,1,-1,0,0,
0,0,-1,1,0,
0,0,-1,0,1;
intmat cvrvz1[4][4] =
-1,-1,-1,-1,
0,0,0,1,
0,0,1,0,
0,1,0,0;
intmat cvrvz2[4][4] =
-1,-1,-1,-1,
0,0,0,1,
0,0,1,0,
1,0,0,0;
intmat cvrvz3[4][4] =
-1,-1,-1,-1,
0,0,0,1,
0,1,0,0,
1,0,0,0;
intmat cvrvz4[4][4] =
```

```
-1,-1,-1,-1,
0,0,1,0,
0,1,0,0,
1,0,0,0;
intmat cvrvz5[4][4] =
0,0,0,1,
0,0,1,0,
0,1,0,0,
1,0,0,0;
cone cn1 = coneViaPoints(cvrvz1);
cone cn2 = coneViaPoints(cvrvz2);
cone cn3 = coneViaPoints(cvrvz3);
cone cn4 = coneViaPoints(cvrvz4);
cone cn5 = coneViaPoints(cvrvz5);
fan scnSigma = fanViaCones(cn1, cn2, cn3, cn4, cn5);
poly p1 = T(1)-T(2)+T(4);
poly p2 = T(2)-T(3)+T(5);
ideal spG = p1, p2;
CEMDS cemds = createCEMDS(rvcvzP, scnSigma, spG);
//Then switch to another ring and fetch the CEMDS.
ring S = 0,T(1..5),lp;
CEMDS cemdsFetched = fetchCEMDS(cemds);
print(cemdsFetched);
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   number of vars : 5
↦ //        block   1 : ordering lp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      1     0    -1     0     0
↦      0     1    -1     0     0
↦      0     0    -1     1     0
↦      0     0    -1     0     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Dimension 4:
↦ 1st maximal cone:
↦ -1, 0, 0, 0,
↦ -1, 1, 0, 0,
↦ -1, 0, 1, 0,
↦ -1, 0, 0, 1
↦
↦ 2nd maximal cone:
↦ 1, -1, 0, 0,
↦ 0, -1, 0, 0,
↦ 0, -1, 1, 0,
↦ 0, -1, 0, 1
↦
↦ 3rd maximal cone:
↦ 1, 0, -1, 0,
↦ 0, 1, -1, 0,
↦ 0, 0, -1, 0,
↦ 0, 0, -1, 1
↦
```

```
↦ 4th maximal cone:
↦ 1, 0, 0, -1,
↦ 0, 1, 0, -1,
↦ 0, 0, 1, -1,
↦ 0, 0, 0, -1
↦
↦ 5th maximal cone:
↦ 1, 0, 0, 0,
↦ 0, 1, 0, 0,
↦ 0, 0, 1, 0,
↦ 0, 0, 0, 1
↦
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ T(1)-T(2)+T(4),
↦ T(2)-T(3)+T(5)
//Should yield the original CEMDS defined first but associated with the new ring.
```

1.1.0.6 gale

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: gale(matz); with matz: bigintmat/intmat.

Assume: intmat limitations are not exceeded by bigintmats.

Purpose: Computes a matrix whose rows generate the integer kernel of the passed matrix.

Return: A bigintmat/intmat whose rows generate the integer kernel of the passed matrix.

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat rvcvzP1[2][4] =
1,0,-1,-1,
0,1,-1,0;
bigintmat rvcvzQ1 = gale(rvcvzP1);
print(rvcvzQ1);
↦ 1, 1, 1, 0,
↦ 1, 0, 0, 1
//Should yield
// (1 1 1 0)
// (1 0 0 1)
//or something in the linear span of these two rows.
//intmat example
intmat rvcvzP2[2][4] =
1,0,-1,-1,
0,1,-1,0;
intmat rvcvzQ2 = gale(rvcvzP2);
print(rvcvzQ2);
↦      1     1     1     0
↦      1     0     0     1
//Same as above, but as intmat.
```

1.1.0.7 galeExtension

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: galeExtension(matzQ, matzPOld); with matzQ: bigintmat/intmat, matzPOld: bigintmat/intmat.

Assume: intmat limitations are not exceeded by bigintmats.

Purpose: Computes a matrix whose rows generate the integer kernel of the first passed matrix, assuming this matrix is some gale dual of the second passed matrix extended by some columns.

Return: An intmat whose rows generate the integer kernel of the first passed matrix such that the second passed matrix is a submatrix of the resulting matrix located in the upper left corner.

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat rvcvzP1[2][6] =
1,0,-1,1,0,-1,
0,1,-1,1,-1,0;
bigintmat rvcvzQ1 = gale(rvcvzP1);
intmat rvcvzNewColCoefs[6][3] =
0,1,1,
1,0,0,
0,0,0,
1,1,0,
0,0,1,
0,0,0;
bigintmat rvcvzQ1Dash = intmatAppendCols(rvcvzQ1, rvcvzQ1 * rvcvzNewColCoefs);
bigintmat rvcvzP1Dash = galeExtension(rvcvzQ1Dash, rvcvzP1);
print(rvcvzP1Dash);
↦  1,  0, -1,  1,  0, -1, 0, 0, 0,
↦  0,  1, -1,  1, -1,  0, 0, 0, 0,
↦  0, -1,  0, -1,  0,  0, 1, 0, 0,
↦ -1,  0,  0, -1,  0,  0, 0, 1, 0,
↦ -1, -1,  1, -1,  0,  0, 0, 0, 1
//This matrix gale dual to rvcvzQ1Dash should have the original matrix as a submatrix in its top left
//intmat example
intmat rvcvzP2[2][6] =
1,0,-1,1,0,-1,
0,1,-1,1,-1,0;
intmat rvcvzQ2 = gale(rvcvzP2);
intmat rvcvzQ2Dash = intmatAppendCols(rvcvzQ2, rvcvzQ2 * rvcvzNewColCoefs);
intmat rvcvzP2Dash = galeExtension(rvcvzQ2Dash, rvcvzP2);
print(rvcvzP2Dash);
↦      1     0    -1     1     0    -1     0     0     0
↦      0     1    -1     1    -1     0     0     0     0
↦      0    -1     0    -1     0     0     1     0     0
↦     -1     0     0    -1     0     0     0     1     0
↦     -1    -1     1    -1     0     0     0     0     1
//This matrix gale dual to rvcvzQ2Dash should have the original matrix as a submatrix in its top left
intmat rvcvzP2DashDash = gale(rvcvzQ2Dash);
print(rvcvzP2DashDash);
↦      0    -1     1    -1     1     0     0     0     0
↦     -1     0     1    -1     0     1     0     0     0
↦      0    -1     0    -1     0     0     1     0     0
↦     -1     0     0    -1     0     0     0     1     0
↦     -1    -1     1    -1     0     0     0     0     1
```

```
//Cf. the "standard" matrix gale dual to rvcvzQ2Dash, which need not have the original ma-
trix as a submatrix.
```

1.1.0.8 intmatAppendCol

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intmatAppendCol(rvcvzInput, cvz); with rvcvzInput: bigintmat/intmat/intvec, cvz: intvec.

Assume: intmat limitations are not exceeded by bigintmats.

Purpose: Appends a column to a bigintmat/intmat/intvec.

Return: An bigintmat/intmat which is the passed bigintmat/intmat/intvec extended by the passed column.

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat rvcvz1[2][3] =
1,2,3,
4,5,6;
intvec vz = -1,9;
bigintmat rvcvzResult1 = intmatAppendCol(rvcvz1, vz);
print(rvcvzResult1);
↦ 1, 2, 3, -1,
↦ 4, 5, 6,  9
//Should yield the intmat
// (1 2 3 -1)
// (4 5 6  9)
//intmat example
intmat rvcvz2[2][3] =
1,2,3,
4,5,6;
intmat rvcvzResult2 = intmatAppendCol(rvcvz2, vz);
print(rvcvzResult2);
↦      1     2     3    -1
↦      4     5     6     9
//Should yield the intmat
// (1 2 3 -1)
// (4 5 6  9)
//intvec example
intvec cvz1 = 1,2,3;
intvec cvz2 = 7,8,9;
intmat rvcvzResult3 = intmatAppendCol(cvz1, cvz2);
print(rvcvzResult3);
↦      1     7
↦      2     8
↦      3     9
//Should yield
// (1 7)
// (2 8)
// (3 9)
```

1.1.0.9 intmatAppendCols

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intmatAppendCols(rvcvzA, rvcvzB); with rvcvzA: bigintmat/intmat, rvcvzB: bigintmat/intmat.

Assume: intmat limitations are not exceeded by bigintmats.

Purpose: Appends the second matrix to the first matrix by generating new columns.

Return: An bigintmat/intmat which is the passed first matrix extended by appending the second passed matrix as new columns.

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat rvcvzA1[2][3] =
1,2,3,
4,5,6;
intmat rvcvzB[2][2] =
7,8,
9,0;
bigintmat rvcvzResult1 = intmatAppendCols(rvcvzA1, rvcvzB);
print(rvcvzResult1);
↦ 1, 2, 3, 7, 8,
↦ 4, 5, 6, 9, 0
//Should yield
// (1 2 3 7 8)
// (4 5 6 9 0)
//intmat example
intmat rvcvzA2[2][3] =
1,2,3,
4,5,6;
intmat rvcvzResult2 = intmatAppendCols(rvcvzA2, rvcvzB);
print(rvcvzResult2);
↦      1     2     3     7     8
↦      4     5     6     9     0
//Should yield
// (1 2 3 7 8)
// (4 5 6 9 0)
```

1.1.0.10 intmatAppendRow

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intmatAppendRow(cvrvzInput, rvz); with cvrvzInput: bigintmat/intmat/intvec, rvz: intvec.

Assume: intmat limitations are not exceeded by bigintmats.

Purpose: Appends a row to a bigintmat/intmat/intvec.

Return: An bigintmat/intmat which is the passed bigintmat/intmat/intvec extended by the passed row.

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat cvrvz1[2][3] = (
1,2,3,
4,5,6);
intvec rvz = 7,8,9;
```

```
bigintmat cvrvzResult1 = intmatAppendRow(cvrvz1, rvz);
print(cvrvzResult1);
↦ 1, 2, 3,
↦ 4, 5, 6,
↦ 7, 8, 9
//Should yield
// (1 2 3)
// (4 5 6)
// (7 8 9)
//intmat example
intmat cvrvz2[2][3] = (
1,2,3,
4,5,6);
intmat cvrvzResult2 = intmatAppendRow(cvrvz2, rvz);
print(cvrvzResult2);
↦      1     2     3
↦      4     5     6
↦      7     8     9
//Should yield
// (1 2 3)
// (4 5 6)
// (7 8 9)
//intvec example
intvec rvz1 = 1,2,3;
intvec rvz2 = 7,8,9;
intmat cvrvzResult3 = intmatAppendRow(rvz1, rvz2);
print(cvrvzResult3);
↦      1     2     3
↦      7     8     9
//Should yield
// (1 2 3)
// (7 8 9)
```

1.1.0.11 intmatAppendRows

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intmatAppendRows(cvrvzA, cvrvzB); with cvrvzA: bigintmat/intmat, cvrvzB: bigintmat/intmat.

Assume: intmat limitations are not exceeded by bigintmats.

Purpose: Appends the second matrix to the first matrix by generating new rows.

Return: An bigintmat/intmat which is the passed first matrix extended by appending the second passed matrix as new rows.

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat cvrvzA1[3][2] =
1,2,
3,4,
5,6;
intmat cvrvzB[2][2] =
7,8,
9,0;
bigintmat cvrvzResult1 = intmatAppendRows(cvrvzA1, cvrvzB);
print(cvrvzResult1);
```

```
↦ 1, 2,
↦ 3, 4,
↦ 5, 6,
↦ 7, 8,
↦ 9, 0
//Should yield
// (1 2)
// (3 4)
// (5 6)
// (7 8)
// (9 0)
//intmat example
intmat cvrvzA2[3][2] =
1,2,
3,4,
5,6;
intmat cvrvzResult2 = intmatAppendRows(cvrvzA2, cvrvzB);
print(cvrvzResult2);
↦      1     2
↦      3     4
↦      5     6
↦      7     8
↦      9     0
//Should yield
// (1 2)
// (3 4)
// (5 6)
// (7 8)
// (9 0)
```

1.1.0.12 intmatContainsRow

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intmatContainsRow(cvrvz, rvz); with cvrvz: bigintmat/intmat, rvz: intvec.

Assume: intmat limitations are not exceeded by bigintmats.

Purpose: Checks whether a bigintmat/intmat contains a specific row.

Return: The int 1 if the passed bigintmat/intmat contains the passed row, 0 else.

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat cvrvz1[2][3] =
1,2,3,
4,5,6;
intvec rvz1 = 4,5,6;
int fResult1 = intmatContainsRow(cvrvz1, rvz1);
print(fResult1);
↦ 1
//Should be true
intvec rvz2 = 7,8,9;
int fResult2 = intmatContainsRow(cvrvz1, rvz2);
print(fResult2);
↦ 0
//Should be false
//intmat example
```

```
intmat cvrvz2[2][3] =
1,2,3,
4,5,6;
int fResult3 = intmatContainsRow(cvrvz2, rvz1);
print(fResult3);
↦ 1
//Should be true
int fResult4 = intmatContainsRow(cvrvz2, rvz2);
print(fResult4);
↦ 0
//Should be false
```

1.1.0.13 intmatDeleteRow

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intmatDeleteRow(cvrvz, irvzDel); with cvrvz: bigintmat/intmat, irvzDel: int.

Assume: intmat limitations are not exceeded by bigintmats.

Purpose: Deletes a specified row from a bigintmat/intmat.

Return: An bigintmat/intmat which is the passed bigintmat/intmat missing the passed index's row.

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat cvrvz1[3][3] =
1,2,3,
4,5,6,
7,8,9;
bigintmat cvrvzResult1 = intmatDeleteRow(cvrvz1, 2);
print(cvrvzResult1);
↦ 1, 2, 3,
↦ 7, 8, 9
//Should yield
// (1 2 3)
// (7 8 9)
//intmat example
intmat cvrvz2[3][3] =
1,2,3,
4,5,6,
7,8,9;
intmat cvrvzResult2 = intmatDeleteRow(cvrvz2, 2);
print(cvrvzResult2);
↦      1     2     3
↦      7     8     9
//Should yield
// (1 2 3)
// (7 8 9)
```

1.1.0.14 intmatNonNegativeCols

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intmatNonNegativeCols(rvcvz); with rvcvz: bigintmat/intmat.

Assume: intmat limitations are not exceeded by bigintmats.

Purpose: Adds an individual constant vector to each of the bigintmat's/intmat's columns so that no entry is negative and at least one entry equals zero.

Return: The bigintmat/intmat with no negative entries and at least one entry being zero in each column resulting from adding constant vectors to the columns of the passed matrix.

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat rvcvz1[2][8] =
2,1,0,-1,1,0,-1,-2,
1,0,-1,-2,2,1,0,-1;
bigintmat rvcvzResult1 = intmatNonNegativeCols(rvcvz1);
print(rvcvzResult1);
↦ 1, 1, 1, 1, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 1, 1, 1, 1
//Should yield the bigintmat
// (1 1 1 1 0 0 0 0)
// (0 0 0 0 1 1 1 1)
//intmat example
intmat rvcvz2[2][8] =
2,1,0,-1,1,0,-1,-2,
1,0,-1,-2,2,1,0,-1;
intmat rvcvzResult2 = intmatNonNegativeCols(rvcvz2);
print(rvcvzResult2);
↦      1     1     1     1     0     0     0     0
↦      0     0     0     0     1     1     1     1
//Should yield the intmat
// (1 1 1 1 0 0 0 0)
// (0 0 0 0 1 1 1 1)
```

1.1.0.15 intmatReplaceCol

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intmatReplaceCol(rvcvz, icvz, cvzReplace); with rvcvz: bigintmat/intmat, icvz: int, cvzReplace: intvec.

Assume: intmat limitations are not exceeded by bigintmats.

Purpose: Replaces a bigintmat's/intmat's column by another one.

Return: The original bigintmat/intmat with the column corresponding to the passed index replaced by the passed intvec.

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat rvcvz1[2][3] =
1,2,0,
4,5,5;
intvec cvzReplace = 3,6;
bigintmat rvcvzResult1 = intmatReplaceCol(rvcvz1, 3, cvzReplace);
print(rvcvzResult1);
↦ 1, 2, 3,
↦ 4, 5, 6
//Should yield the bigintmat
```

```
// (1 2 3)
// (4 5 6)
//intmat example
intmat rvcvz2[2][3] =
1,2,0,
4,5,5;
intmat rvcvzResult2 = intmatReplaceCol(rvcvz2, 3, cvzReplace);
print(rvcvzResult2);
↦      1     2     3
↦      4     5     6
//Should yield the intmat
// (1 2 3)
// (4 5 6)
```

1.1.0.16 intmatTakeCol

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intmatTakeCol(rvcvz, icvzTake); with rvcvz: bigintmat/intmat, icvzTake: int.

Assume: intmat limitations are not exceeded by bigintmats.

Purpose: Extracts a specified column from a bigintmat/intmat.

Return: An intvec that is the icvzTake-th column of the passed bigintmat/intmat.

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat rvcvz1[2][3] =
1,2,3,
4,5,6;
intvec cvzResult1 = intmatTakeCol(rvcvz1, 2);
print(cvzResult1);
↦ 2,
↦ 5
//Should yield the vector (2,5)
//intmat example
intmat rvcvz2[2][3] =
1,2,3,
4,5,6;
intvec cvzResult2 = intmatTakeCol(rvcvz2, 2);
print(cvzResult2);
↦ 2,
↦ 5
//Should yield the vector (2,5)
```

1.1.0.17 intmatTakeCols

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intmatTakeCols(rvcvz, rvfTakeCol); with rvcvz: bigintmat/intmat, rvfTakeCol: intvec of flags.

Assume: intmat limitations are not exceeded by bigintmats.

Purpose: Generates a submatrix of a bigintmat/intmat by only taking the columns specified.

Return: An bigintmat/intmat that is the submatrix of the passed bigintmat/intmat consisting of the columns flagged true in the passed vector.

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat rvcvz1[2][4] =
1,2,3,4,
5,6,7,8;
intvec rvfTakeCol = 1,0,1,1;
bigintmat rvcvzResult1 = intmatTakeCols(rvcvz1, rvfTakeCol);
print(rvcvzResult1);
↦ 1, 3, 4,
↦ 5, 7, 8
//Should yield
// (1 3 4)
// (5 7 8)
//intmat example
intmat rvcvz2[2][4] =
1,2,3,4,
5,6,7,8;
intmat rvcvzResult2 = intmatTakeCols(rvcvz2, rvfTakeCol);
print(rvcvzResult2);
↦      1     3     4
↦      5     7     8
//Should yield
// (1 3 4)
// (5 7 8)
```

1.1.0.18 intmatTakeRow

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intmatTakeRow(cvrvz, irvzTake); with cvrvz: bigintmat/intmat, irvzTake: int.

Assume: intmat limitations are not exceeded by bigintmats.

Purpose: Extracts a specified row from a bigintmat/intmat.

Return: An intvec that is the irvzTake-th row of the passed bigintmat/intmat.

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat cvrvz1[3][3] =
1,2,3,
4,5,6,
7,8,9;
intvec rvzResult1 = intmatTakeRow(cvrvz1, 2);
print(rvzResult1);
↦ 4,
↦ 5,
↦ 6
//Should yield the vector (4,5,6)
//intmat example
intmat cvrvz2[3][3] =
1,2,3,
4,5,6,
7,8,9;
```

```
intvec rvzResult2 = intmatTakeRow(cvrvz2, 2);
print(rvzResult2);
↦ 4,
↦ 5,
↦ 6
//Should yield the vector (4,5,6)
```

1.1.0.19 intmatTakeRows

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intmatTakeRows(cvrvz, cvfTakeRow); with cvrvz: bigintmat/intmat, cvfTakeRow: intvec of flags.

Assume: intmat limitations are not exceeded by bigintmats.

Purpose: Generates a submatrix of a bigintmat/intmat by only taking the rows specified.

Return: An bigintmat/intmat which is the submatrix of the passed bigintmat/intmat consisting of the rows flagged true in the passed vector.

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat cvrvz1[4][2] =
1,5,
2,6,
3,7,
4,8;
intvec cvfTakeRow = 1,0,1,1;
bigintmat cvrvzResult1 = intmatTakeRows(cvrvz1, cvfTakeRow);
print(cvrvzResult1);
↦ 1, 5,
↦ 3, 7,
↦ 4, 8
//Should yield
// (1 5)
// (3 7)
// (4 8)
//intmat example
intmat cvrvz2[4][2] =
1,5,
2,6,
3,7,
4,8;
intmat cvrvzResult2 = intmatTakeRows(cvrvz2, cvfTakeRow);
print(cvrvzResult2);
↦      1     5
↦      3     7
↦      4     8
//Should yield
// (1 5)
// (3 7)
// (4 8)
```

1.1.0.20 intvecComponentAnd

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intvecComponentAnd(vfV, vfW); with vfV: intvec of flags, vfW: intvec of flags.

Purpose: Performs the logic AND operation (&&) on two vectors component by component.

Return: An intvec whose components are: result[i] = vfV[i] && vfW[i].

Example:

```
LIB "compcox.lib";
intvec rvfV = 1,1,0,0;
intvec rvfW = 1,0,1,0;
intvec rvfResult = intvecComponentAnd(rvfV, rvfW);
print(rvfResult);
↦ 1,
↦ 0,
↦ 0,
↦ 0
//Should yield the vector (1,0,0,0).
```

1.1.0.21 intvecComponentNot

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intvecComponentNot(vfV); with vfV: intvec of flags.

Purpose: Performs the logic NOT operation (!) on a vector component by component.

Return: An intvec whose components are: result[i] = !vfV[i].

Example:

```
LIB "compcox.lib";
intvec vf = 1,0;
intvec vfResult = intvecComponentNot(vf);
print(vfResult);
↦ 0,
↦ 1
//Should yield the vector (0,1).
```

1.1.0.22 intvecComponentOr

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intvecComponentOr(vfV, vfW); with vfV: intvec of flags, vfW: intvec of flags.

Purpose: Performs the logic OR operation (`||`) on two vectors component by component.

Return: An intvec whose components are: result[i] = vfV[i] `||` vfW[i].

Example:

```
LIB "compcox.lib";
intvec vfV = 1,1,0,0;
intvec vfW = 1,0,1,0;
intvec vfResult = intvecComponentOr(vfV, vfW);
print(vfResult);
↦ 1,
↦ 1,
↦ 1,
↦ 0
//Should yield the vector (1,1,1,0).
```

1.1.0.23 intvecMin

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intvecMin(vz); with vz: intvec.

Purpose: Gets the minimal entry of an intvec.

Return: An int that is the minimal entry of the passed intvec.

Example:

```
LIB "compcox.lib";
intvec vz = 1,8,-5,0;
int zMin = intvecMin(vz);
zMin;
↦ -5
//Should be -5.
```

1.1.0.24 intvecSumFlags

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intvecSumFlags(vfV); with vfV: intvec of flags.

Purpose: Determines the number of nonzero entries in an intvec.

Return: An int that is the count of all nonzero entries of the passed vector.

Example:

```
LIB "compcox.lib";
intvec vf = 0,1,2,3;
int nFlags = intvecSumFlags(vf);
print(nFlags);
↦ 3
//Should return 0 + 1 + 1 + 1 = 3.
```

1.1.0.25 isRowLinearlyDependent

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: isRowLinearlyDependent(cvrvz, intvec rvz); with cvrvz: bigintmat/intmat, rvz: intvec.

Assume: intmat limitations are not exceeded by bigintmats.

Purpose: Checks whether the collection consisting of the row vectors of a bigintmat/intmat and another intvec is linearly dependent.

Return: An int pointing out whether the collection consisting of the row vectors of the passed bigintmat/intmat and the passed intvec is linearly dependent (1) or not (0).

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat cvrvz1[2][3] =
1,0,0,
0,1,0;
intvec rvz1 = 0,0,1;
int f1 = isRowLinearlyDependent(cvrvz1, rvz1);
```

```
print(f1);
↦ 0
//Should be false
intvec rvz2 = 1,1,0;
int f2 = isRowLinearlyDependent(cvrvz1, rvz2);
print(f2);
↦ 1
//Should be true
//intmat example
intmat cvrvz2[2][3] =
1,0,0,
0,1,0;
int f3 = isRowLinearlyDependent(cvrvz2, rvz1);
print(f3);
↦ 0
//Should be false
int f4 = isRowLinearlyDependent(cvrvz2, rvz2);
print(f4);
↦ 1
//Should be true
```

1.1.0.26 lusolveInvertible

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: lusolveInvertible(P, L, U, cvnumB); with P: matrix of numbers, L: matrix of numbers, U: matrix of numbers, cvnumB: matrix of numbers.

Assume: P, L, U arise from the LU-decomposition of an invertible matrix A (with P*A = L*U).

Purpose: Solves the system of linear equations Ax = cvnumB <=> L*U*x = P*cvnumB for invertible A.

Return: A matrix of numbers containing exactly one column with the solution to the system of linear equations Ax = cvnumB.

Example:

```
LIB "compcox.lib";
ring R = 0,x,dp;
matrix matnumA[2][2] = 4,3,6,3;
matrix cvnumB[2][1] = 7,9;
list smatnumPLU = ludecomp(matnumA);
matrix cvnumX = lusolveInvertible(smatnumPLU[1], smatnumPLU[2], smatnumPLU[3], cvnumB);
print(cvnumX);
↦ 1,
↦ 1
//Should yield the 2x1-matrix [1,1].
```

1.1.0.27 lusolveUnderdetFullRank

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: lusolveUnderdetFullRank(P, L, U, cvnumB); with P: matrix of numbers, L: matrix of numbers, U: matrix of numbers, cvnumB: matrix of numbers.

Assume: P, L, U arise from the LU-decomposition of a matrix A of full rank (with P*A = L*U). Ax = cvnumB is an underdetermined system of linear equations.

Purpose: Solves the underdetermined system of linear equations Ax = cvnumB <=> L*U*x = P*cvnumB.

Return: A list containing
(i) a matrix of numbers containing exactly one column with a specific solution to the system of linear equations Ax = cvnumB. (ii) a matrix of numbers whose columns span the affine linear space of all solutions to the system of linear equations Ax = cvnumB.

Example:

```
LIB "compcox.lib";
ring R = 0,x,dp;
matrix matnumA[2][3] = 1,1,1,0,-1,-2;
matrix cvnumB[2][1] = 1,1;
list smatnumPLU = ludecomp(matnumA);
list smatnumResult = lusolveUnderdetFullRank(smatnumPLU[1], smatnumPLU[2], smatnumPLU[3], cvnumB);
print(smatnumResult);
↦ [1]:
↦    _[1,1]=2
↦    _[2,1]=-1
↦    _[3,1]=0
↦ [2]:
↦    _[1,1]=-1
↦    _[2,1]=2
↦    _[3,1]=-1
//Should yield a list of two entries:
// - A solution for the system of linear equations, e.g. the 3x1-matrix [2,-1,0];
// - Generators of the affine space of solutions, e.g. the 3x1-matrix [-1,2,-1].
kill matnumA;
kill cvnumB;
kill smatnumPLU;
kill smatnumResult;
matrix matnumA[2][3] = 0,1,0,1,1,1;
matrix cvnumB[2][1] = 0,0;
list smatnumPLU = ludecomp(matnumA);
list smatnumResult = lusolveUnderdetFullRank(smatnumPLU[1], smatnumPLU[2], smatnumPLU[3], cvnumB);
print(smatnumResult);
↦ [1]:
↦    _[1,1]=0
↦    _[2,1]=0
↦    _[3,1]=0
↦ [2]:
↦    _[1,1]=1
↦    _[2,1]=0
↦    _[3,1]=-1
//Should yield a list of two entries:
// - A solution for the system of linear equations, e.g. the 3x1-matrix [0,0,0];
// - Generators of the affine space of solutions, e.g. the 3x1-matrix [-1,0,1].
```

1.1.0.28 unitMatrix

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: unitMatrix(int n); with n: int.

Purpose: Builds an n x n unit matrix with integer entries.

Return: The n x n unit matrix with integer entries.

Example:

```
LIB "compcox.lib";
intmat result = unitMatrix(3);
print(result);
↦      1     0     0
↦      0     1     0
↦      0     0     1
//Should return
// (1 0 0)
// (0 1 0)
// (0 0 1)
```

1.1.0.29 vectorComponentZero

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: vectorComponentZero(vnum, nnum); with vnum: intvec/vector of numbers, nnum: int.

Assume: nnum is the number of entries of the passed vector, thus nnum = nrows(vnum) for intvecs.

Purpose: Marks all those entries of a vector positively which equal zero.

Return: An intvec of flags whose components are: result[i] == (vnum[i] != 0).

Example:

```
LIB "compcox.lib";
//intvec example
intvec vf = 1,0;
intvec vfResult = vectorComponentZero(vf, 2);
print(vfResult);
↦ 0,
↦ 1
//Should yield the intvec (0,1).
//vector example
ring R = (0,a),T,dp;
vector vnum = [1/2, 0, a];
intvec vfResult2 = vectorComponentZero(vnum, 4);
print(vfResult2);
↦ 0,
↦ 1,
↦ 0,
↦ 1
//Should yield the intvec (0,1,0,1).
```

1.1.0.30 vectorFromIntvec

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: vectorFromIntvec(vz); with vz: intvec.

Purpose: Converts an intvec to a vector.

Return: The vector resulting by converting the passed intvec's components from int to number.

Example:

```
LIB "compcox.lib";
ring R = (0,a),T,dp;
intvec vz = 1,2,0;
vector vnum = vectorFromIntvec(vz);
print(vnum);
↦ [1,2]
//Should yield the vector [1, 2].
```

1.1.0.31 vectorTake

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: vectorTake(vnum, vfTake); with vnum: vector, vfTake: intvec of flags.

Purpose: Generates a vector by taking specified entries of an original vector.

Return: A vector consisting of those entries of the first passed vector that were specified by the second passed vector.

Example:

```
LIB "compcox.lib";
ring R = (0,a),T,dp;
vector vnum = [1, 2, a, 4];
intvec vf = 1,0,1,0;
vector vnumResult = vectorTake(vnum, vf);
print(vnumResult);
↦ [1,(a)]
//Should yield the vector [1, a].
```

1.1.0.32 containedVariables

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: containedVariables(p); with p: poly.

Purpose: Collects the indices of all variables that are contained within the passed polynomial with a positive exponent.

Return: An intvec vf with entries vf[i] = 1 if the i-th variable is contained in the passed polynomial and vf[i] = 0 else.

Example:

```
LIB "compcox.lib";
ring R = 0,T(1..5),dp;
poly p1 = T(1)*T(3)^2 + T(4)^3;
intvec vfResult1 = containedVariables(p1);
print(vfResult1);
↦ 1,
↦ 0,
↦ 1,
↦ 1,
↦ 0
//Should return (1,0,1,1,0)
poly p2 = 0;
intvec vfResult2 = containedVariables(p2);
print(vfResult2);
↦ 0,
↦ 0,
↦ 0,
```

```
↦ 0,
↦ 0
//Should return (0,0,0,0,0)
```

1.1.0.33 evaluatePoly

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: evaluatePoly(p, vnum); with p: poly, vnum: intvec/vector of numbers only.

Assume: The size of vnum matches the number of ring variables of the currently defined basering.

Purpose: Calculates p(vnum).

Return: A number p(vnum).

Example:

```
LIB "compcox.lib";
//intvec example
ring R = (0,a),T(1..6),dp;
poly p = 3/2*T(1)*T(2)^2 + T(3)^2*T(4);
intvec vz = 1,2,3,0,4,0;
number numResult = evaluatePoly(p, vz);
print(numResult);
↦ 6
//Should return 3/2*1*2^2+3^2*0 = 6
//vector example
vector vnum = [2/3, a, 1/2, 0, 4];
numResult = evaluatePoly(p, vnum);
print(numResult);
↦ (a^2)
//Should return 3/2*2/3*a^2+(1/2)^2*0 = a^2
```

1.1.0.34 exponentMatrixFromPoly

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: exponentMatrixFromPoly(p); with p: poly.

Purpose: Lists the exponents of all polynomial's terms as rows of a matrix.

Return: An intmat containing the polynomial's terms' exponents as follows: Each row represents one of the polynomial's terms and each coulumn represents one of the ring's variables. The (i,j)-th matrix entry therefore is the exponent of the j-th variable in the i-th term.

Example:

```
LIB "compcox.lib";
ring R = 0,T(1..4),lp;
poly p1 = 2*T(1)*T(2)^3 - 1/2*T(3) + T(1)^2*T(3)*T(4)^3;
intmat cvrvzResult1 = exponentMatrixFromPoly(p1);
print(cvrvzResult1);
↦      2     0     1     3
↦      1     3     0     0
↦      0     0     1     0
//Should yield an intmat with the rows
// (2 0 1 3)
// (1 3 0 0)
```

```
// (0 0 1 0)
poly p2 = T(1) + 1;
intmat cvrvzResult2 = exponentMatrixFromPoly(p2);
print(cvrvzResult2);
↦      1     0     0     0
↦      0     0     0     0
//Should yield the intmat with the rows
// (1 0 0 0)
// (0 0 0 0)
poly p3 = 0;
intmat cvrvzResult3 = exponentMatrixFromPoly(p3);
print(cvrvzResult3);
↦      0     0     0     0
//Should yield the intmat with the rows
// (0 0 0 0)
```

1.1.0.35 getDegree

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: getDegree(rvcvzQ, hpQ); with rvcvzQ: intmat, hpQ: poly.

Assume: hpQ is homogeneous w.r.t. the basering's variables' degrees implied by rvcvzQ.

Purpose: Calculates the degree of the passed polynomial which is homogeneous w.r.t. to the grading given by the passed matrix.

Return: An intvec that is the degree of the passed polynomial w.r.t. the grading given by the passed matrix.

Example:

```
LIB "compcox.lib";
ring R = 0,T(1..3),dp;
poly hpQ = T(1)^10*T(2) - T(2)^3*T(3);
intmat rvcvzQ[2][3] =
1,3,4,
0,-3,6;
intvec cvzResult = getDegree(rvcvzQ, hpQ);
print(cvzResult);
↦ 13,
↦ -3
//Should yield the degree vector (13,-3);
```

1.1.0.36 intersectContainedVariables

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: intersectContainedVariables(sp); with sp: list of polys.

Purpose: Collects the indices of all variables that are contained within all passed polynomials with a positive exponent.

Return: An intvec vf with entries vf[i] = 1 if the i-th variable is contained in all passed polynomials and vf[i] = 0 else.

Example:

```
LIB "compcox.lib";
ring R = 0,T(1..5),dp;
poly p1 = T(1)*T(3)^2 + T(4)^3;
```

```
poly p2 = T(1)^2*T(2)^2 + T(4)^3 + T(5);
list sp1 = p1, p2;
intvec vzResult1 = intersectContainedVariables(sp1);
print(vzResult1);
↦ 1,
↦ 0,
↦ 0,
↦ 1,
↦ 0
//Should return the vector (1,0,0,1,0).
poly p3 = T(5);
poly p4 = 1;
list sp2 = p3, p4;
intvec vzResult2 = intersectContainedVariables(sp2);
print(vzResult2);
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0
//Should return the vector (0,0,0,0,0).
intvec vzResult3 = intersectContainedVariables(p3);
print(vzResult3);
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 1
//Should return the vector (0,0,0,0,1).
intvec vzResult4 = intersectContainedVariables(list());
print(vzResult4);
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0
//Should return the vector (0,0,0,0,0).
```

1.1.0.37 isOfTotalDegree

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: isOfTotalDegree(nDeg, p); with nDeg: int, p: poly.

Purpose: Checks whether a polynomial is homogeneous and of a specific degree w.r.t. standard grading.

Return: 1 if the passed polynomial is homogeneous and of the passed degree w.r.t. standard grading, 0 else.

Example:

```
LIB "compcox.lib";
ring R = 0,T(1..3),dp;
poly p1 = T(1)*T(2)^3;
int fResult1a = isOfTotalDegree(3, p1);
print(fResult1a);
↦ 0
//Should be false.
int fResult1b = isOfTotalDegree(4, p1);
```

```
print(fResult1b);
↦ 1
//Should be true.
poly p2 = T(1) + T(3);
int fResult2a = isOfTotalDegree(2, p2);
print(fResult2a);
↦ 0
//Should be false.
int fResult2b = isOfTotalDegree(1, p2);
print(fResult2b);
↦ 1
//Should be true.
poly p3 = 1;
int fResult3a = isOfTotalDegree(1, p3);
print(fResult3a);
↦ 0
//Should be false.
int fResult3b = isOfTotalDegree(0, p3);
print(fResult3b);
↦ 1
//Should be true.
poly p4 = T(1)*T(2) + T(3);
int fResult4 = isOfTotalDegree(1, p4);
print(fResult4);
↦ 0
//Should be false, as this is not a homogeneous polynomial.
```

1.1.0.38 isTotalDegreeHomogeneous

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: isTotalDegreeHomogeneous(p); with p: poly.

Purpose: Determines whether a polynomial is homogeneous w.r.t. standard grading.

Return: 1 if the passed polynomial is homogeneous w.r.t. standard grading, 0 else.

Example:

```
LIB "compcox.lib";
ring R = 0,T(1..3),dp;
poly p1 = T(1)*T(2)^3;
isTotalDegreeHomogeneous(p1);
↦ 1
//Should be true.
poly p2 = T(1)*T(2) + T(3)^2;
isTotalDegreeHomogeneous(p2);
↦ 1
//Should be true.
poly p3 = 0;
isTotalDegreeHomogeneous(p3);
↦ 1
//Should be true.
poly p4 = T(1)*T(2) + T(3);
isTotalDegreeHomogeneous(p4);
↦ 0
//Should be false.
```

1.1.0.39 veronese

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: veronese(nDeg, vnum); with nDeg: int, vnum: vector of numbers.

Purpose: Applies the veronese embedding of a certain degree on a vector.

Return: A vector of numbers that is the result of the veronese embedding of the passed degree on the passed vector.

Example:

```
LIB "compcox.lib";
ring R = 0,T(1..3),dp;
vector vnum = [1, 2, 3];
vector vnumResult = veronese(3, vnum);
print(vnumResult);
↦ [1,2,3,4,6,9,8,12,18,27]
//Should yield the vector
//   [1*1*1, 1*1*2, 1*1*3, 1*2*2, 1*2*3, 1*3*3, 2*2*2, 2*2*3, 2*3*3, 3*3*3]
// = [1, 2, 3, 4, 6, 9, 8, 1 2, 18, 27]
```

1.1.0.40 isPointInVariety

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: isPointInVariety(spI, vnum); with spI: ideal, vnum: intvec/vector numbers.

Purpose: Checks whether a point is contained in the variety V(K^n; spI) of the passed ideal spI.

Return: The int 1 if the passed point is contained in the variety of the passed ideal, 0 else.

Example:

```
LIB "compcox.lib";
//intvec example
ring R = 0,T(1..3),dp;
poly p1 = T(1) + T(2);
poly p2 = T(3) - T(2)^2;
ideal sp = p1, p2;
intvec vz1 = 1,-1,1;
int f1 = isPointInVariety(sp, vz1);
print(f1);
↦ 1
//Should be true.
intvec vz2 = 1,1,1;
int f2 = isPointInVariety(sp, vz2);
print(f2);
↦ 0
//Should be false.
//vector example
vector vnum1 = [1/2, -1/2, 1/4];
int f3 = isPointInVariety(sp, vnum1);
print(f3);
↦ 1
//Should be true.
vector vnum2 = [1, 1/2];
int f4 = isPointInVariety(sp, vnum2);
print(f4);
↦ 0
//Should be false.
```

1.1.0.41 hyperplanes

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: hyperplanes(svnumPts); with svnumPts: list of vectors of numbers.

Assume: The current basering is the polynomial ring in n >= 2 variables.

Purpose: Computes all hyperplanes through the passed points.

Return: A list of vectors containing the normal vectors on the resulting hyperplanes.

Example:

```
LIB "compcox.lib";
ring R = 0,T(1..3),dp;
vector vnumPt1 = [1,0,0];
vector vnumPt2 = [0,1,0];
vector vnumPt3 = [0,0,1];
vector vnumPt4 = [1,1,1];
list svnumPts = vnumPt1, vnumPt2, vnumPt3, vnumPt4;
list svnumNormal = hyperplanes(svnumPts);
print(svnumNormal);
↦ [1]:
↦    -gen(3)+gen(2)
↦ [2]:
↦    -gen(3)+gen(1)
↦ [3]:
↦    -gen(3)
↦ [4]:
↦    -gen(2)+gen(1)
↦ [5]:
↦    -gen(2)
↦ [6]:
↦    -gen(1)
//Should yield a list of normal vectors like
//[1,0,0], [0,1,0], [0,0,1], [0,1,-1], [1,0,-1] and [1,-1,0].
```

1.1.0.42 quadrics

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: quadrics(svnumPts); with svnumPts: list of vectors of numbers.

Assume: The current basering is the polynomial ring in n >= 2 variables.

Purpose: Computes all quadrics through the passed points.

Return: A list of vectors containing the coefficient vectors for the resulting quadrics (for processing with veronesePoly).

Example:

```
LIB "compcox.lib";
ring R = (0,a,b),T(1..3),dp;
vector vnumPt1 = [1,0,0];
vector vnumPt2 = [0,1,0];
vector vnumPt3 = [0,0,1];
vector vnumPt4 = [1,1,1];
vector vnumPt5 = [1,a,b];
list svnumPts = vnumPt1, vnumPt2, vnumPt3, vnumPt4, vnumPt5;
list svnumNormal = quadrics(svnumPts);
```

```
print(svnumNormal);
↦ [1]:
↦    -gen(5)+(-a*b+a)/(a-b)*gen(3)+(a*b-b)/(a-b)*gen(2)
//Should yield a list of cofficient vectors with one entry:
//[0, (a*b-a-b)/(a-b), (-a*b+2*a)/(a-b), 0, -1, 0].
```

1.1.0.43 containsRay

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: containsRay(scn, vzRay); with scn: fan, vzRay: intvec.

Assume: The passed ray's component count matches the fan's ambient dimension.

Purpose: Checks whether a ray is contained in a fan.

Return: The int 1 if the passed fan contains the passed ray, 0 else.

Example:

```
LIB "compcox.lib";
intmat cvrvz1[2][2] =
1,0,
0,1;
intmat cvrvz2[2][2] =
0,1,
-1,0;
cone cn1 = coneViaPoints(cvrvz1);
cone cn2 = coneViaPoints(cvrvz2);
fan scn = fanViaCones(cn1, cn2);
intvec vzRay1 = 0,1;
int fResult1 = containsRay(scn, vzRay1);
print(fResult1);
↦ 1
//Should be true.
intvec vzRay2 = 0,-1;
int fResult2 = containsRay(scn, vzRay2);
print(fResult2);
↦ 0
//Should be false.
intvec vzRay3 = 1,1;
int fResult3 = containsRay(scn, vzRay3);
print(fResult3);
↦ 0
//Should be false.
```

1.1.0.44 contractRay

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: contractRay(scn, vzRay); with scn: fan, vzRay: intvec.

Assume: The passed ray is indeed contained in the passed fan and contractible.

Purpose: Computes the fan obtained by contracting one of the rays of the original fan.

Return: The original fan with the passed ray contracted.

Example:

```
LIB "compcox.lib";
//The fan of the blowup of P2 in [1, 0, 0]. We contract the ray (-1, 0) added in the pro-
cess of blowing up P2, obtaining P2 itself.
intmat cvrvz1[2][2] =
1,0,
0,1;
intmat cvrvz2[2][2] =
0,1,
-1,0;
intmat cvrvz3[2][2] =
-1,0,
-1,-1;
intmat cvrvz4[2][2] =
-1,-1,
1,0;
cone cn1 = coneViaPoints(cvrvz1);
cone cn2 = coneViaPoints(cvrvz2);
cone cn3 = coneViaPoints(cvrvz3);
cone cn4 = coneViaPoints(cvrvz4);
fan scn = fanViaCones(cn1, cn2, cn3, cn4);
intvec vzRay = -1,0;
fan scnContracted = contractRay(scn, vzRay);
scnContracted;
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 2
↦
↦ DIM
↦ 2
↦
↦ LINEALITY_DIM
↦ 0
↦
↦ RAYS
↦ -1 -1 # 0
↦ 0 1 # 1
↦ 1 0 # 2
↦
↦ N_RAYS
↦ 3
↦
↦ LINEALITY_SPACE
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 # 0
↦ 0 -1 # 1
↦
↦ F_VECTOR
↦ 1 3 3
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
```

```
↦
↦ CONES
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 2}
↦
↦ MAXIMAL_CONES
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 2}
↦
↦ CONES_ORBITS
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 2}
↦
↦ MAXIMAL_CONES_ORBITS
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 2}
↦
//Should yield the fan with the maximal cones cn1, cn2 + cn3, cn4.
```

1.1.0.45 generateOrthantFace

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: generateOrthantFace(vfIndices); with vfIndices: intvec.

Purpose: Calculates the orthant face generated by the canonical basis vectors flagged true in the passed vector.

Return: A cone that is the orthant face generated by the canonical basis vectors flagged true in the passed vector.

Example:

```
LIB "compcox.lib";
intvec vfIndices = 1,0,1,1,0;
cone cnResult = generateOrthantFace(vfIndices);
rays(cnResult);
↦ 1, 0, 0, 0, 0,
↦ 0, 0, 1, 0, 0,
↦ 0, 0, 0, 1, 0
//Should yield the cone defined by the rays through (1,0,0,0,0), (0,0,1,0,0) and (0,0,0,1,0).
```

1.1.0.46 irrelevantIdealFromFan

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: irrelevantIdealFromFan(sof); with sof: fan consisting solely of faces of the positive orthant.

Purpose: Calculates the irrelevant ideal that is associated with a fan consisting of faces of the positive orthant.

Return: Returns the ring in which the monomials generating the irrelevant ideal were built in.

Side effects:
Exports an ideal gsmonResult containing the irrelevant ideal that is associated with the passed fan consisting of faces of the positive orthant.

Example:

```
LIB "compcox.lib";
intmat cvrvz1[2][4] =
1,0,0,0,
0,1,0,0;
intmat cvrvz2[2][4] =
0,1,0,0,
0,0,0,1;
intmat cvrvz3[2][4] =
0,0,0,1,
0,0,1,0;
intmat cvrvz4[2][4] =
0,0,1,0,
1,0,0,0;
cone of1 = coneViaPoints(cvrvz1);
cone of2 = coneViaPoints(cvrvz2);
cone of3 = coneViaPoints(cvrvz3);
cone of4 = coneViaPoints(cvrvz4);
fan sof = fanViaCones(of1, of2, of3, of4);
def R = irrelevantIdealFromFan(sof);
setring R;
ideal smonResult = gsmonResult;
print(smonResult);
↦ T(1)*T(2),
↦ T(1)*T(3),
↦ T(2)*T(4),
↦ T(3)*T(4)
//Should yield the ideal
// <T(1)T(2), T(1)T(3), T(2)T(4), T(3)T(4)>.
```

1.1.0.47 isRayContractible

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: isRayContractible(rvcvzP, icvzRay); with rvcvzP: bigintmat/intmat, icvzRay: int.

Purpose: Checks whether a ray from a list of rays is contractible, i.e. whether its corresponding ray in the gale dual version is extremal.

Return: The int 1 if the passed ray is contractible w.r.t. the passed other rays, 0 else.

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat rvcvzP1[2][4] =
1,0,-1,-1,
0,1,-1,0;
```

```
int fResult1 = isRayContractible(rvcvzP1, 3);
print(fResult1);
↦ 0
//Should be false.
//intmat example
intmat rvcvzP2[2][4] =
1,0,-1,-1,
0,1,-1,0;
int fResult2 = isRayContractible(rvcvzP2, 4);
print(fResult2);
↦ 1
//Should be true.
```

1.1.0.48 mapFan

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: mapFan(rvcvzP, scn); with rvcvzP: bigintmat/intmat, scn: fan.

Purpose: Maps a fan under a matrix by mapping each cone under that matrix.

Return: A fan that is the original fan mapped by rvcvzP.

Example:

```
LIB "compcox.lib";
intmat cvrvz1[2][3] =
1,0,0,
0,1,0;
intmat cvrvz2[2][3] =
0,1,0,
0,0,1;
intmat cvrvz3[2][3] =
0,0,1,
1,0,0;
cone cn1 = coneViaPoints(cvrvz1);
cone cn2 = coneViaPoints(cvrvz2);
cone cn3 = coneViaPoints(cvrvz3);
fan scn = fanViaCones(cn1, cn2, cn3);
//bigintmat example
bigintmat rvcvzP1[2][3] =
1,0,-1,
0,1,-1;
fan scnResult1 = mapFan(rvcvzP1, scn);
scnResult1;
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 2
↦
↦ DIM
↦ 2
↦
↦ LINEALITY_DIM
↦ 0
↦
↦ RAYS
↦ -1 -1 # 0
```

```
↦ 0 1 # 1
↦ 1 0 # 2
↦
↦ N_RAYS
↦ 3
↦
↦ LINEALITY_SPACE
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 # 0
↦ 0 -1 # 1
↦
↦ F_VECTOR
↦ 1 3 3
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 2}
↦
↦ MAXIMAL_CONES
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 2}
↦
↦ CONES_ORBITS
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 2}
↦
↦ MAXIMAL_CONES_ORBITS
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 2}
↦
//Should be the fan with the rays (1,0), (0,1), (-1,-1) and full support.
//intmat example
intmat rvcvzP2[2][3] =
1,0,-1,
0,1,-1;
fan scnResult2 = mapFan(rvcvzP2, scn);
scnResult2;
↦ _application PolyhedralFan
↦ _version 2.2
```

```
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 2
↦
↦ DIM
↦ 2
↦
↦ LINEALITY_DIM
↦ 0
↦
↦ RAYS
↦ -1 -1 # 0
↦ 0 1 # 1
↦ 1 0 # 2
↦
↦ N_RAYS
↦ 3
↦
↦ LINEALITY_SPACE
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 # 0
↦ 0 -1 # 1
↦
↦ F_VECTOR
↦ 1 3 3
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 2}
↦
↦ MAXIMAL_CONES
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 2}
↦
↦ CONES_ORBITS
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 2}
↦
↦ MAXIMAL_CONES_ORBITS
```

```
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 2}
↦
//Should be the fan with the rays (1,0), (0,1), (-1,-1) and full support.
```

1.1.0.49 movingConeFromWeights

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: movingConeFromWeights(rvcvzQ); with rvcvzWeights: bigintmat/intmat.

Purpose: Calculates the moving cone of the cone defined by the column vectors of the passed intmat.

Return: A cone that is the moving cone of the cone defined by the column vectors of the passed intmat.

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat rvcvzQ1[2][4] =
1,1,0,1,
0,1,1,0;
cone cnMov1 = movingConeFromWeights(rvcvzQ1);
rays(cnMov1);
↦ 1, 0,
↦ 1, 1
//intmat examples
//one column
intmat rvcvzQ2[2][1] =
1,
0;
cone cnMov2 = movingConeFromWeights(rvcvzQ2);
cnMov2;
↦ AMBIENT_DIM
↦ 2
↦ FACETS
↦
//Should yield the cone consisting only of the origin in ambient dimension 2.
//more columns
intmat rvcvzQ3[2][4] =
1,1,0,1,
0,1,1,0;
cone cnMov3 = movingConeFromWeights(rvcvzQ3);
rays(cnMov3);
↦ 1, 0,
↦ 1, 1
//Should yield the cone generated by the rays passing through (1,0) and (1,1).
```

1.1.0.50 orthantFanFromWeight

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: orthantFanFromWeight(rvcvzQ, cvzWeight); with rvcvzQ: bigintmat/intmat, cvzWeight: intvec.

Assume: The weight vector is an element of the relative interior of the moving cone defined by the grading matrix.

Purpose: Calculates the fan corresponding to a grading matrix and a weight vector.

Return: A fan that is the fan corresponding to the passed grading matrix and the passed weight vector.

Example:

```
LIB "compcox.lib";
//bigintmat example
bigintmat rvcvzQ1[2][4] =
1,1,1,0,
1,0,0,1;
cone cn1 = movingConeFromWeights(rvcvzQ1);
intvec cvzWeight = intvec(relativeInteriorPoint(cn1));
fan sof1 = orthantFanFromWeight(rvcvzQ1, cvzWeight);
sof1;
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 4
↦
↦ DIM
↦ 2
↦
↦ LINEALITY_DIM
↦ 0
↦
↦ RAYS
↦ 0 0 0 1 # 0
↦ 0 0 1 0 # 1
↦ 0 1 0 0 # 2
↦ 1 0 0 0 # 3
↦
↦ N_RAYS
↦ 4
↦
↦ LINEALITY_SPACE
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 0 0 # 0
↦ 0 -1 0 0 # 1
↦ 0 0 -1 0 # 2
↦ 0 0 0 -1 # 3
↦
↦ F_VECTOR
↦ 1 4 4
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
```

```
↦ {3}
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 3}
↦ {2 3}
↦
↦ MAXIMAL_CONES
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 3}
↦ {2 3}
↦
↦ CONES_ORBITS
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
↦ {3}
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 3}
↦ {2 3}
↦
↦ MAXIMAL_CONES_ORBITS
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 3}
↦ {2 3}
↦
//Should return four maximal cones in a fan:
// - cone generated by rays (1,0,0,0) and (0,1,0,0)
// - cone generated by rays (1,0,0,0) and (0,0,1,0)
// - cone generated by rays (0,1,0,0) and (0,0,0,1)
// - cone generated by rays (0,0,1,0) and (0,0,0,1).
//intmat example
intmat rvcvzQ2[2][4] =
1,1,1,0,
1,0,0,1;
fan sof2 = orthantFanFromWeight(rvcvzQ2, cvzWeight);
sof2;
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 4
↦
↦ DIM
↦ 2
↦
↦ LINEALITY_DIM
↦ 0
↦
↦ RAYS
↦ 0 0 0 1 # 0
↦ 0 0 1 0 # 1
↦ 0 1 0 0 # 2
↦ 1 0 0 0 # 3
```

```
↦
↦ N_RAYS
↦ 4
↦
↦ LINEALITY_SPACE
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 0 0 # 0
↦ 0 -1 0 0 # 1
↦ 0 0 -1 0 # 2
↦ 0 0 0 -1 # 3
↦
↦ F_VECTOR
↦ 1 4 4
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
↦ {3}
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 3}
↦ {2 3}
↦
↦ MAXIMAL_CONES
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 3}
↦ {2 3}
↦
↦ CONES_ORBITS
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
↦ {3}
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 3}
↦ {2 3}
↦
↦ MAXIMAL_CONES_ORBITS
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 3}
↦ {2 3}
↦
//Should return four maximal cones in a fan:
// - cone generated by rays (1,0,0,0) and (0,1,0,0)
// - cone generated by rays (1,0,0,0) and (0,0,1,0)
```

```
// - cone generated by rays (0,1,0,0) and (0,0,0,1)
// - cone generated by rays (0,0,1,0) and (0,0,0,1).
```

1.1.0.51 stellarSubdivision

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: stellarSubdivision(scn, vzNewRay); with scn: fan, vzNewRay: intvec.

Assume: The passed new ray is contained in the support of the passed fan.

Purpose: Calculates the stellar subdivision of a fan w.r.t. a new ray to be inserted.

Return: A fan that is the original fan except that stellar subdivisions were performed on cones containing the new ray to add.

Example:

```
LIB "compcox.lib";
//First example
intmat cvrvz1[2][2] =
1,0,
0,1;
intmat cvrvz2[2][2] =
0,1,
-1,-1;
intmat cvrvz3[2][2] =
-1,-1,
1,0;
cone cn1 = coneViaPoints(cvrvz1);
cone cn2 = coneViaPoints(cvrvz2);
cone cn3 = coneViaPoints(cvrvz3);
fan scn = fanViaCones(cn1, cn2, cn3);
intvec vzRay = -1,0;
fan scnSubdivision = stellarSubdivision(scn, vzRay);
scnSubdivision;
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 2
↦
↦ DIM
↦ 2
↦
↦ LINEALITY_DIM
↦ 0
↦
↦ RAYS
↦ -1 -1 # 0
↦ -1 0 # 1
↦ 0 1 # 2
↦ 1 0 # 3
↦
↦ N_RAYS
↦ 4
↦
↦ LINEALITY_SPACE
↦
```

```
↦ ORTH_LINEALITY_SPACE
↦ -1 0 # 0
↦ 0 -1 # 1
↦
↦ F_VECTOR
↦ 1 4 4
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
↦ {3}
↦ {0 1} # Dimension 2
↦ {1 2}
↦ {0 3}
↦ {2 3}
↦
↦ MAXIMAL_CONES
↦ {0 1} # Dimension 2
↦ {1 2}
↦ {0 3}
↦ {2 3}
↦
↦ CONES_ORBITS
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
↦ {3}
↦ {0 1} # Dimension 2
↦ {1 2}
↦ {0 3}
↦ {2 3}
↦
↦ MAXIMAL_CONES_ORBITS
↦ {0 1} # Dimension 2
↦ {1 2}
↦ {0 3}
↦ {2 3}
↦
//Should yield the fan with the maximal cones cn1, cn3 the two cones resulting of the di-
vision of cn2.
//Second example
intmat cvrvz4[4][3] =
1,0,0,
1,2,0,
1,2,2,
1,0,2;
intmat cvrvz5[3][3] =
1,0,0,
1,2,0,
```

```
1,1,-1;
cone cn4 = coneViaPoints(cvrvz4);
cone cn5 = coneViaPoints(cvrvz5);
fan scn2 = fanViaCones(cn4, cn5);
intvec vzRay2 = 1,1,0;
fan scnSubdivision2 = stellarSubdivision(scn2, vzRay2);
scnSubdivision2;
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 3
↦
↦ DIM
↦ 3
↦
↦ LINEALITY_DIM
↦ 0
↦
↦ RAYS
↦ 1 0 0 # 0
↦ 1 0 2 # 1
↦ 1 1 -1 # 2
↦ 1 1 0 # 3
↦ 1 2 0 # 4
↦ 1 2 2 # 5
↦
↦ N_RAYS
↦ 6
↦
↦ LINEALITY_SPACE
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 0 # 0
↦ 0 -1 0 # 1
↦ 0 0 -1 # 2
↦
↦ F_VECTOR
↦ 1 6 10 5
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
↦ {3}
↦ {4}
↦ {5}
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {0 3}
```

```
↦ {1 3}
↦ {2 3}
↦ {1 5}
↦ {2 4}
↦ {3 4}
↦ {3 5}
↦ {4 5}
↦ {0 1 3} # Dimension 3
↦ {0 2 3}
↦ {1 3 5}
↦ {2 3 4}
↦ {3 4 5}
↦
↦ MAXIMAL_CONES
↦ {0 1 3} # Dimension 3
↦ {0 2 3}
↦ {1 3 5}
↦ {2 3 4}
↦ {3 4 5}
↦
↦ CONES_ORBITS
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
↦ {3}
↦ {4}
↦ {5}
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {0 3}
↦ {1 3}
↦ {2 3}
↦ {1 5}
↦ {2 4}
↦ {3 4}
↦ {3 5}
↦ {4 5}
↦ {0 1 3} # Dimension 3
↦ {0 2 3}
↦ {1 3 5}
↦ {2 3 4}
↦ {3 4 5}
↦
↦ MAXIMAL_CONES_ORBITS
↦ {0 1 3} # Dimension 3
↦ {0 2 3}
↦ {1 3 5}
↦ {2 3 4}
↦ {3 4 5}
↦
//Should yield the fan where both cones are divided, the simplicial one in two, the other one in thre
```

1.1.0.52 binomialIdeal

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: binomialIdeal(rvcvzP); with rvcvzP: intmat.

Assume: The current basering has as many variables as the passed matrix has columns.

Purpose: Calculates the lattice ideal generated by the rows of a matrix.

Return: The ideal that is the lattice ideal associated with the passed matrix.

Example:

```
LIB "compcox.lib";
ring R = 0,X(1..3),dp;
intmat rvcvzP[2][3] =
1,0,-1,
0,2,-1;
ideal sbinResult = binomialIdeal(rvcvzP);
print(sbinResult);
↦ X(1)-X(3),
↦ X(2)^2-X(3)
//Should yield the ideal <X(1)-X(3), X(2)^2-X(3)>.
```

1.1.0.53 minimalizeIdeal

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: minimalizeIdeal(sp); with sp: ideal.

Purpose: Computes a minimal representation of an ideal.

Return: The passed ideal minimally represented.

Note: This is only a wrapper for SINGULAR's minbase function.

Example:

```
LIB "compcox.lib";
//Parameterless example
ring R = 0,X(1..3),dp;
poly p1 = X(1)*X(2) + X(3);
poly p2 = 2*X(1)*X(2)^2 + X(3);
poly p3 = 5*X(2);
ideal sp = p1, p2, p3;
ideal spMin = minimalizeIdeal(sp);
print(spMin);
↦ X(1)*X(2)+X(3),
↦ X(2)
//Should yield the input ideal, but with a minimal representation, such as <X(1)*X(2) + X(3), X(2)>
//Example with parameter
ring S = (0,a),X(1..3),dp;
poly p1 = X(1)*X(2) + a*X(3);
poly p2 = 2*X(1)*X(2)^2 + a*X(3);
poly p3 = 5*X(2);
ideal sp = p1, p2, p3;
ideal spMin = minimalizeIdeal(sp);
print(spMin);
↦ X(1)*X(2)+(a)*X(3),
↦ X(2)
//Should yield the input ideal, but with a minimal representation, such as <X(1)*X(2) + a*X(3), X(2)>
```

1.1.0.54 toricMorphismPullback

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: toricMorphismPullback(matz, sp); with matz: intmat, sp: ideal.

Purpose: Pulls back an ideal under a morphism of standard tori determined by a martix.

Return: The ring (standard polynomial ring) in which the pulled back ideal is defined.

Side effects:
Exports the ideal gspResult which is the pullback of the passed ideal unter the morphism of standard tori determined by matz. As the returned ring is not a Laurent polynomial ring, care is taken that the generators of gspResult will have positive exponents only.

Example:

```
LIB "compcox.lib";
//Parameterless example
ring R = 0,X(1..2),dp;
intmat matz[2][4] =
0,-1,1,0,
-1,-1,3,-3;
poly p1 = X(1)*X(2) + X(1) + 3;
poly p2 = X(1) + X(2);
ideal sp = p1, p2;
def RPullback = toricMorphismPullback(matz, sp);
setring RPullback;
ideal spResult = gspResult;
print(spResult);
↦ 3*T(1)*T(2)^2*T(4)^3+T(1)*T(2)*T(3)*T(4)^3+T(3)^4,
↦ T(1)*T(4)^3+T(3)^2
//Should yield the ideal
//<3*X(1)*X(2)^2*X(4)^3 + X(1)*X(2)*X(3)*X(4)^3 + X(3)^4, X(1)*X(3)*X(4)^3 + X(3)^3>
//in Q[X(1)+-, ..., X(4)+-].
//Example with parameter
ring S = (0,a),X(1..2),dp;
poly p1 = X(1)*X(2) + X(1) + a;
poly p2 = X(1) + X(2);
ideal sp = p1, p2;
def SPullback = toricMorphismPullback(matz, sp);
setring SPullback;
ideal spResult = gspResult;
print(spResult);
↦ (a)*T(1)*T(2)^2*T(4)^3+T(1)*T(2)*T(3)*T(4)^3+T(3)^4,
↦ T(1)*T(4)^3+T(3)^2
//Should yield the ideal
//<a*X(1)*X(2)^2*X(4)^3 + X(1)*X(2)*X(3)*X(4)^3 + X(3)^4, X(1)*X(3)*X(4)^3 + X(3)^3>
//in Q[X(1)+-, ..., X(4)+-].
```

1.1.0.55 varproductOrbitClosureIdeal

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: varproductOrbitClosureIdeal(rvcvzP, rvnumPointInOrbit); with rvcvzP: intmat, rvnumPointInOrbit: intvec/vector of numbers.

Assume: The current basering has as many variables as the passed matrix has columns.

Purpose: Calculates the ideal of a point's orbit closure under a torus action implied by the rays of the considered ambient toric variety's fan.

Return: The ideal of the passed point's orbit closure under the torus action defined by the passed matrix.

Example:

```
LIB "compcox.lib";
//First example:
ring R1 = 0,X(1..3),dp;
intmat rvcvzP1[2][3] =
1,0,2,
0,3,1;
intvec vz1 = 1,1,1;
ideal spResult1 = varproductOrbitClosureIdeal(rvcvzP1, vz1);
print(spResult1);
↦ 1
//Should yield the ideal <X(2)^6-X(1), -X(2)^3+X(1)*X(3), X(2)^3*X(3)-1>.
//Second example:
ring R2 = 0,X(1..6),dp;
intmat rvcvzP2[2][6] =
1,0,-1,1,0,-1,
0,1,-1,1,-1,0;
intvec vz2 = 2,1,1,1,2,1;
ideal spResult2 = varproductOrbitClosureIdeal(rvcvzP2, vz2);
print(spResult2);
↦ X(1)*X(5)-4*X(2)*X(6),
↦ 2*X(2)*X(4)-X(3)*X(5),
↦ X(1)*X(4)-2*X(3)*X(6)
//Should yield the ideal <1/2*X(1)*X(4) - X(3)*X(6), X(2)*X(4) - 1/2*X(3)*X(5), 1/4*X(1)*X(5) -
X(2)*X(6)>.
//Third example:
ring R3 = (0,a),X(1..6),dp;
intmat rvcvzP3[2][6] =
1,0,-1,1,0,-1,
0,1,-1,1,-1,0;
vector vnum3 = [1, 1, a, 1, 1];
ideal spResult3 = varproductOrbitClosureIdeal(rvcvzP3, vnum3);
print(spResult3);
↦ (a)*X(2)*X(4)-X(3)*X(5),
↦ X(6)
//Should yield the ideal <a*X(2)*X(4) - X(3)*X(5), X(6)>.
//Fourth example:
ring R4 = 0,X(1..3),dp;
intmat rvcvzP4[2][3] =
1,0,-1,
0,1,-1;
intvec vz4 = 0,0,0;
ideal spResult4 = varproductOrbitClosureIdeal(rvcvzP4, vz4);
print(spResult4);
↦ X(1),
↦ X(2),
↦ X(3)
//Should yield the ideal <X(1), X(2), X(3)>.
```

1.1.0.56 varproductIdealSaturation

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: varproductIdealSaturation(spI); with spI: ideal.

Purpose: Calculates the saturation of an ideal w.r.t. the monomial var(1)*...*var(nvars).

Return: The ideal that is the saturation of the passed ideal w.r.t. the monomial var(1)*...*var(nvars).

Example:

```
LIB "compcox.lib";
//Parameterless example
ring R = 0,X(1..10),dp;
ideal spI = X(7)*X(10) - X(2)*X(4) + X(3)*X(5),
X(8)*X(10) - X(1)*X(4) + X(3)*X(6),
X(9)*X(10) - X(1)*X(5) + X(2)*X(6);
ideal spISat = varproductIdealSaturation(spI);
print(spISat);
↦ X(1)*X(5)-X(2)*X(6)-X(9)*X(10),
↦ X(2)*X(4)-X(3)*X(5)-X(7)*X(10),
↦ X(1)*X(4)-X(3)*X(6)-X(8)*X(10),
↦ X(6)*X(7)-X(5)*X(8)+X(4)*X(9),
↦ X(1)*X(7)-X(2)*X(8)+X(3)*X(9)
//Should yield the ideal generated by:
// X(7)*X(10) - X(2)*X(4) + X(3)*X(5),
// X(8)*X(10) - X(1)*X(4) + X(3)*X(6),
// X(9)*X(10) - X(1)*X(5) + X(2)*X(6),
// X(6)*X(7) - X(5)*X(8) + X(4)*X(9),
// X(1)*X(7) - X(2)*X(8) + X(3)*X(9).
//Example with parameter
ring S = (0,a),X(1..10),dp;
ideal spI = a*X(7)*X(10) - X(2)*X(4) + X(3)*X(5),
X(8)*X(10) - X(1)*X(4) + X(3)*X(6),
X(9)*X(10) - X(1)*X(5) + X(2)*X(6);
ideal spISat = varproductIdealSaturation(spI);
print(spISat);
↦ X(1)*X(5)-X(2)*X(6)-X(9)*X(10),
↦ X(2)*X(4)-X(3)*X(5)+(-a)*X(7)*X(10),
↦ X(1)*X(4)-X(3)*X(6)-X(8)*X(10),
↦ (a)*X(6)*X(7)-X(5)*X(8)+X(4)*X(9),
↦ (a)*X(1)*X(7)-X(2)*X(8)+X(3)*X(9)
//Should yield the ideal generated by:
// a*X(7)*X(10) - X(2)*X(4) + X(3)*X(5),
// X(8)*X(10) - X(1)*X(4) + X(3)*X(6),
// X(9)*X(10) - X(1)*X(5) + X(2)*X(6),
// a*X(6)*X(7) - X(5)*X(8) + X(4)*X(9),
// a*X(1)*X(7) - X(2)*X(8) + X(3)*X(9).
```

1.1.0.57 xIdealSaturation

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: xIdealSaturation(spI, simonLexVars, vfSatVars); with spI: ideal, simonLexVars: list of integers (the indices of those variables that shall be ordered lexicographically), vfSatVars: intvec (indices of the variables to consider in saturation).

Purpose: Calculates the saturation of an ideal w.r.t. the monomial defined by vfSatVars. In contrast to working with the standard elimination order, some variables can be ordered lexicographically.

Return: The ideal that is the saturation of the passed ideal w.r.t. the monomial defined by vfSatVars.

Example:

```
LIB "compcox.lib";
//Parameterless example
ring R = 0,X(1..10),dp;
```

```
ideal spI = X(7)*X(10) - X(2)*X(4) + X(3)*X(5),
X(8)*X(10) - X(1)*X(4) + X(3)*X(6),
X(9)*X(10) - X(1)*X(5) + X(2)*X(6);
list simonLexVars = 9,8,7;
intvec vfSatVars = 0,0,0,0,0,0,0,0,0,1;
ideal spISat = xIdealSaturation(spI, simonLexVars, vfSatVars);
print(spISat);
↦ -X(2)*X(4)+X(3)*X(5)+X(7)*X(10),
↦ -X(1)*X(4)+X(3)*X(6)+X(8)*X(10),
↦ -X(1)*X(5)+X(2)*X(6)+X(9)*X(10),
↦ -X(1)*X(4)*X(7)+X(3)*X(6)*X(7)+X(2)*X(4)*X(8)-X(3)*X(5)*X(8),
↦ X(6)*X(7)-X(5)*X(8)+X(4)*X(9),
↦ X(1)*X(7)-X(2)*X(8)+X(3)*X(9)
//Should yield the ideal generated by:
// X(7)*X(10) - X(2)*X(4) + X(3)*X(5),
// X(8)*X(10) - X(1)*X(4) + X(3)*X(6),
// X(9)*X(10) - X(1)*X(5) + X(2)*X(6),
// X(6)*X(7) - X(5)*X(8) + X(4)*X(9),
// X(1)*X(7) - X(2)*X(8) + X(3)*X(9).
//Example with parameter
ring S = (0,a),X(1..10),dp;
ideal spI = a*X(7)*X(10) - X(2)*X(4) + X(3)*X(5),
X(8)*X(10) - X(1)*X(4) + X(3)*X(6),
X(9)*X(10) - X(1)*X(5) + X(2)*X(6);
ideal spISat = xIdealSaturation(spI, simonLexVars, vfSatVars);
print(spISat);
↦ -X(2)*X(4)+X(3)*X(5)+(a)*X(7)*X(10),
↦ -X(1)*X(4)+X(3)*X(6)+X(8)*X(10),
↦ -X(1)*X(5)+X(2)*X(6)+X(9)*X(10),
↦ (-a)*X(1)*X(4)*X(7)+(a)*X(3)*X(6)*X(7)+X(2)*X(4)*X(8)-X(3)*X(5)*X(8),
↦ (a)*X(6)*X(7)-X(5)*X(8)+X(4)*X(9),
↦ (a)*X(1)*X(7)-X(2)*X(8)+X(3)*X(9)
//Should yield the ideal generated by:
// a*X(7)*X(10) - X(2)*X(4) + X(3)*X(5),
// X(8)*X(10) - X(1)*X(4) + X(3)*X(6),
// X(9)*X(10) - X(1)*X(5) + X(2)*X(6),
// a*X(6)*X(7) - X(5)*X(8) + X(4)*X(9),
// a*X(1)*X(7) - X(2)*X(8) + X(3)*X(9).
```

1.1.0.58 veronesePoly

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: veronesePoly(nDeg, vnum); with nDeg: int, vnum: vector of numbers.

Purpose: Expands a vector of coefficients for the monomials of a certain degree to the sum of those monomials multiplied by their corresponding coefficient.

Return: A poly that is the sum of all monomials of the passed degree, each of them multiplied by its corresponding passed coefficient.

Example:

```
LIB "compcox.lib";
ring R = 0,T(1..3),dp;
vector vnum = veronese(3, [1, 2, 3]);
print(vnum);
↦ [1,2,3,4,6,9,8,12,18,27]
//Should yield the vector
//[1, 2, 3, 4, 6, 9, 8, 12, 18, 27]
```

```
poly pResult = veronesePoly(3, vnum);
print(pResult);
↦ T(1)^3+2*T(1)^2*T(2)+4*T(1)*T(2)^2+8*T(2)^3+3*T(1)^2*T(3)+6*T(1)*T(2)*T(3\
   )+12*T(2)^2*T(3)+9*T(1)*T(3)^2+18*T(2)*T(3)^2+27*T(3)^3
//Should yield the polynomial
//T(1)^3 + 2*T(1)^2*T(2) + 3*T(1)^2*T(3) + 4*T(1)*T(2)^2 + 6*T(1)*T(2)*T(3) + 9*T(1)*T(3)^2 + 8*T(2)^
```

1.1.0.59 blowupCEMDS

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: blowupCEMDS(cemds, spC, vnD); with cemds: CEMDS, spC: list of polynomials, vnD: intvec consisting of positive integers; blowupCEMDS(cemds, spC, vnD, fComputeFan); with cemds: CEMDS, spC: list of polynomials, vnD: intvec consisting of positive integers; fComputeFan: int. blowupCEMDS(cemds, spC, vnD, fComputeFan, vzWeight); with cemds: CEMDS, C: ideal, fVerify: int, fComputeFan: int, vzWeight: intvec.

Assume: C is an ideal in the Cox ring of the given CEMDS and the generators of C are all K-prime; we assume that the subvariety defined by C is contained in the smooth locus.

Purpose: Blows up the given CEMDS in the subvariety defined by the ideal C.

Return: The modified CEMDS. Whether the result is indeed a CEMDS is checked iff fVerify != 0. The CEMDS's fan is computed iff fComputeFan != 0.

Note: By default: svnumPts = list(); fVerify = 0; fComputeFan = 0; vzWeight is an element of the relative interior of the moving cone generated by the columns of the grading matrix rvcvzQ = gale(cemds.rvcvzP) extended by the passed polynomials' degrees.

Example:

```
LIB "compcox.lib";
// //First example: No fan computation.
//  ring R = (0,a),T(1..3),dp;
//  //Define the input data
//  intmat rvcvzP[2][3] =
//  1,0,-1,
//  0,1,-1;
//  ideal spG = 0;
//  vector vnum1 = [1, 0, 0];
//  vector vnum2 = [0, 1, 0];
//  list svnum = vnum1, vnum2;
//  intmat cvrvz1[2][2] =
//  1,0,
//  0,1;
//  intmat cvrvz2[2][2] =
//  0,1,
//  -1,-1;
//  intmat cvrvz3[2][2] =
//  -1,-1,
//  1,0;
//  cone cn1 = coneViaPoints(cvrvz1);
//  cone cn2 = coneViaPoints(cvrvz2);
//  cone cn3 = coneViaPoints(cvrvz3);
//  fan scnSigma = fanViaCones(cn1, cn2, cn3);
```

```
//  //Compose the CEMDS from the data provided until now.
//  ideal spGFetched = fetch(R, spG);
//  CEMDS cemds = createCEMDS(rvcvzP, scnSigma, spGFetched);
//  //Perform the blowup
//  CEMDS cemdsBlowup = blowupCEMDS(cemds, svnum);
//  //Obtains the CEMDS's ring
//  def RBlowup = cemdsBlowup.R;
//  setring RBlowup;
//  print(cemdsBlowup);
//  //Should yield
//  // - A matrix with the following rays as columns
//  //   (1 0 -1 -1  0)
//  //   (0 1 -1  0 -1)
//  //   or there must exist a linear isomorphism mapping the output rays to those above.█
//  // - An empty fan.
//  // - The zero ideal.
// //Second example: With fan computation.
//  setring R;
//  //Perform the blowup
//  CEMDS cemdsBlowupWithFan = blowupCEMDS(cemds, svnum, 1, 1);
//  //Obtains the CEMDS's ring
//  def RBlowupWithFan = cemdsBlowupWithFan.R;
//  setring RBlowupWithFan;
//  print(cemdsBlowupWithFan);
//  //Should yield
//  // - A matrix with the following rays as columns
//  //   (1 0 -1 -1  0)
//  //   (0 1 -1  0 -1)
//  //   or there must exist a linear isomorphism mapping the output rays to those above.█
//  // - The fan consisting of the maximal cones generated by the following rays:
//  //   * (1,0), (0,1);
//  //   * (0,1), (-1,0);
//  //   * (-1,0), (-1,-1);
//  //   * (-1,-1), (0,-1);
//  //   * (0,-1), (1,0);
//  // - The zero ideal.
printlevel=1;
//Third example:
// should be unsuccessful
intmat Q1[5][8] =
1,0,0,0,2,0,3,-1,
0,1,0,0,1,0,2,-1,
0,0,1,0,-3,0,-2,2,
0,0,0,1,-2,0,-1,1,
0,0,0,0,0,1,1,-1;
intmat P1[3][8] = gale(Q1);
fan Sigma1 = emptyFan(2);
ring R1 = 0,T(1..8),dp;
ideal G1 = T(3)^3*T(4)^2*T(5) - T(1)^2*T(2) - T(7)*T(8);
CEMDS X1 = createCEMDS(P1, Sigma1, G1);
list spC = T(1),T(3),T(7);
intvec vnD = 1,1,1;
CEMDS X2 = blowupCEMDS(X1, spC, vnD);
↦
↦ ----- Starting elimination, please be patient... -----
↦ Time elapsed while eliminating (in ms):
↦ 0
↦ ----- Elimination done! -----
```

```
↦
↦ CEMDS verification failed on blowing up.
↦
print(X2);   // should be unsuccessful
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   number of vars : 9
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) T(6) T(7) T(8) T(\
   9)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      0     1     1     1     1    -1     1     0     0
↦     -2    -1    -2    -3    -2    -1     0     1     0
↦     -1    -1    -2    -3    -3     1     0     0     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Empty fan of ambient dimension 3
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ -T(2)^2*T(3)*T(7)^3*T(9)^2+T(1)*T(6)^2*T(9)+T(5)*T(8)
kill Q1,G1, X1, spC, X2, Sigma1, P1, R1;
printlevel=1;
// Fourth example:
// should be successful
intmat Q1[5][8] =
1,0,0,0,2,0,3,-1,
0,1,0,0,1,0,2,-1,
0,0,1,0,-3,0,-2,2,
0,0,0,1,-2,0,-1,1,
0,0,0,0,0,1,1,-1;
intmat P1[3][8] = gale(Q1);
fan Sigma1 = emptyFan(2);
ring R1 = 0,T(1..8),dp;
ideal G1 = T(3)^3*T(4)^2*T(5) - T(1)^2*T(2) - T(7)*T(8);
CEMDS X1 = createCEMDS(P1, Sigma1, G1);
list spC = T(1),T(3),T(7);
intvec spD = 1,1,2;
CEMDS X2 = blowupCEMDS(X1, spC, spD);
↦
↦ ----- Starting elimination, please be patient... -----
↦ Time elapsed while eliminating (in ms):
↦ 0
↦ ----- Elimination done! -----
↦
↦ CEMDS verification successful.
↦
print(X2);   // should be successful
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   number of vars : 9
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) T(6) T(7) T(8) T(\
   9)
```

```
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      0     1     1     1     1    -1     1     0     0
↦     -1     0     0     0     1    -2     0     1     0
↦     -1    -1    -2    -3    -3     1     0     0     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
  x of its rays:
↦ Empty fan of ambient dimension 3
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ -T(2)^2*T(3)*T(7)^3*T(9)+T(1)*T(6)^2+T(5)*T(8)
```

See also: ⟨undefined⟩ [blowupCEMDSpoints treats the special case of blowing up points on a CEMDS], page ⟨undefined⟩.

1.1.0.60 blowupCEMDSpoints

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: blowupCEMDSpoints(cemds, svnumPts); with cemds: CEMDS, svnumPts: list of vectors of numbers/a vector of numbers; blowupCEMDSpoints(cemds, svnumPts, fVerify); with cemds: CEMDS, svnumPts: list of vectors of numbers, fVerify: int; blowupCEMDSpoints(cemds, svnumPts, fVerify, fComputeFan); with cemds: CEMDS, svnumPts: list of vectors of numbers, fVerify: int, fComputeFan: int. blowupCEMDSpoints(cemds, svnumPts, fVerify, fComputeFan, vzWeight); with cemds: CEMDS, svnumPts: list of vectors of numbers, fVerify: int, fComputeFan: int, vzWeight: intvec.

Assume: All of the passed points to blow up are contained in the passed CEMDS.

Purpose: Blows up the ambient toric variety of a CEMDS in some points contained therein.

Return: The modified CEMDS in the blown up toric ambient variety. Whether the result is indeed a CEMDS is checked iff fVerify != 0. The CEMDS's fan is computed iff fComputeFan != 0.

Note: By default: svnumPts = list(); fVerify = 0; fComputeFan = 0; vzWeight is an element of the relative interior of the moving cone generated by the columns of the grading matrix rvcvzQ = gale(cemds.rvcvzP) extended by the passed polynomials' degrees.

Example:

```
LIB "compcox.lib";
//First example: No fan computation.
ring R = (0,a),T(1..3),dp;
//Define the input data
intmat rvcvzP[2][3] =
1,0,-1,
0,1,-1;
ideal spG = 0;
vector vnum1 = [1, 0, 0];
vector vnum2 = [0, 1, 0];
list svnum = vnum1, vnum2;
```

```
intmat cvrvz1[2][2] =
1,0,
0,1;
intmat cvrvz2[2][2] =
0,1,
-1,-1;
intmat cvrvz3[2][2] =
-1,-1,
1,0;
cone cn1 = coneViaPoints(cvrvz1);
cone cn2 = coneViaPoints(cvrvz2);
cone cn3 = coneViaPoints(cvrvz3);
fan scnSigma = fanViaCones(cn1, cn2, cn3);
//Compose the CEMDS from the data provided until now.
ideal spGFetched = fetch(R, spG);
CEMDS cemds = createCEMDS(rvcvzP, scnSigma, spGFetched);
//Perform the blowup
CEMDS cemdsBlowup = blowupCEMDSpoints(cemds, svnum);
↦
↦ NOTE: You have chosen to not perform a verification step; the result may \
   not be a CEMDS.
//Obtains the CEMDS's ring
def RBlowup = cemdsBlowup.R;
setring RBlowup;
print(cemdsBlowup);
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : 0
↦ //   number of vars : 5
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      1     0    -1    -1     0
↦      0    -1     1     0     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Empty fan of ambient dimension 2
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ 0
//Should yield
// - A matrix with the following rays as columns
//   (1 0 -1 -1  0)
//   (0 1 -1  0 -1)
//   or there must exist a linear isomorphism mapping the output rays to those above.
// - An empty fan.
// - The zero ideal.
//Second example: With fan computation.
setring R;
//Perform the blowup
CEMDS cemdsBlowupWithFan = blowupCEMDSpoints(cemds, svnum, 1, 1);
↦
↦ The resulting embedded space was successfully verified to be a CEMDS.
```

```
//Obtains the CEMDS's ring
def RBlowupWithFan = cemdsBlowupWithFan.R;
setring RBlowupWithFan;
print(cemdsBlowupWithFan);
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : 0
↦ //   number of vars : 5
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      1     0    -1    -1     0
↦      0    -1     1     0     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Dimension 2:
↦ 1st maximal cone:
↦ -1, -1,
↦  0,  1
↦
↦ 2nd maximal cone:
↦ -1,  0,
↦  0, -1
↦
↦ 3rd maximal cone:
↦ -1, 0,
↦  1, 1
↦
↦ 4th maximal cone:
↦ 1,  0,
↦ 0, -1
↦
↦ 5th maximal cone:
↦ 1, 0,
↦ 0, 1
↦
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ 0
//Should yield
// - A matrix with the following rays as columns
//   (1 0 -1 -1  0)
//   (0 1 -1  0 -1)
//   or there must exist a linear isomorphism mapping the output rays to those above.
// - The fan consisting of the maximal cones generated by the following rays:
//   * (1,0), (0,1);
//   * (0,1), (-1,0);
//   * (-1,0), (-1,-1);
//   * (-1,-1), (0,-1);
//   * (0,-1), (1,0);
// - The zero ideal.
```

1.1.0.61 compressCEMDS

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: compressCEMDS(cemds); with cemds: CEMDS; compressCEMDS(cemds, fVerify); with cemds: CEMDS, fVerify: int; compressCEMDS(cemds, fVerify, svnumPts); with cemds: CEMDS, fVerify: int, svnumPts: list of vectors of numbers; compressCEMDS(cemds, fVerify, svnumPts, fComputeFan); with cemds: CEMDS, fVerify: int, svnumPts: list of vectors of numbers, fComputeFan: int; compressCEMDS(cemds, fVerify, svnumPts, fComputeFan, vzWeight); with cemds: CEMDS, fVerify: int, svnumPts: list of vectors of numbers, fComputeFan: int, vzWeight: intvec.

Assume: (i) All points to compress along with the CEMDS, if there were any passed, are contained in the CEMDS to compress; (ii) The intvec "vzWeight", if passed, is element of the ample cone defined by the grading matrix rvcvzQ extended by the passed polynomials' degrees and some true saturated connected collection of facets of the positive orthant.

Purpose: Embeds a CEMDS in a new ambient toric variety such that abundant relations in the ideal defining the CEMDS (i.e. of the type var(i) - p with p not containing var(i)) are erased. Optionally checks whether the result really is a CEMDS and transfers points from the uncompressed CEMDS to the compressed one.

Return: If no optional output was requested, the compressed CEMDS only. If only verification was requested, but no points or an empty list of points to compress along with the CEMDS were passed, a list containing the following (in order of appearance): (i) the compressed CEMDS with or without a fan (depending on fComputeFan and vzWeight) (always present), (ii) the verification result (present as fVerify was passed and fVerify != 0). If a non-empty list of points to compress along with the CEMDS was passed, the compressed CEMDS's basering.

Side effects:
Iff a non-empty list of points to compress along with the CEMDS was passed, exports a list gsResult containing the following (if present and in order of appearance): (i) the compressed CEMDS with or without a fan (depending on fComputeFan and vzWeight) (always present), (ii) the verification result (present iff fVerify != 0), (iii) the list of compressed points (present as there was a list passed).

Note: By default: fVerify = 0; svnumPts = list(); fComputeFan = 0; vzWeight is an element of the relative interior of the moving cone generated by the columns of the grading matrix rvcvzQ = gale(cemds.rvcvzP) extended by the passed polynomials' degrees.

Example:

```
LIB "compcox.lib";
//First example: No optional arguments
ring R = (0,a),T(1..5),dp;
//Blowup of P2 (stretched in order to blow up [1,1,1])
intmat rvcvzP[4][5] =
```

```
1,0,-1,0,0,
0,1,-1,0,0,
0,0,-1,1,0,
0,0,-1,0,1;
intmat cvrvz1[4][4] =
-1,-1,-1,-1,
0,0,0,1,
0,0,1,0,
0,1,0,0;
intmat cvrvz2[4][4] =
-1,-1,-1,-1,
0,0,0,1,
0,0,1,0,
1,0,0,0;
intmat cvrvz3[4][4] =
-1,-1,-1,-1,
0,0,0,1,
0,1,0,0,
1,0,0,0;
intmat cvrvz4[4][4] =
-1,-1,-1,-1,
0,0,1,0,
0,1,0,0,
1,0,0,0;
intmat cvrvz5[4][4] =
0,0,0,1,
0,0,1,0,
0,1,0,0,
1,0,0,0;
cone cn1 = coneViaPoints(cvrvz1);
cone cn2 = coneViaPoints(cvrvz2);
cone cn3 = coneViaPoints(cvrvz3);
cone cn4 = coneViaPoints(cvrvz4);
cone cn5 = coneViaPoints(cvrvz5);
fan scnSigma = fanViaCones(cn1, cn2, cn3, cn4, cn5);
poly p1 = T(1)-T(2)+T(4);
poly p2 = T(2)-T(3)+T(5);
ideal spG = p1, p2;
CEMDS cemds = createCEMDS(rvcvzP, scnSigma, spG);
CEMDS cemdsResult = compressCEMDS(cemds);
//Obtains the CEMDS's ring
def RCompressed = cemdsResult.R;
setring RCompressed;
print(cemdsResult);
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : 0
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦     -1     1     0
↦     -1     0     1
↦
```

```
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Empty fan of ambient dimension 2
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ 0
//Should yield the CEMDS consisting of
// - A rays' matrix with the columns
//   (1 0 -1)
//   (0 1 -1)
//   or a matrix with its rows linearly dependent on that
// - The empty fan of ambient dimension 2
// - The zero ideal
//Second example: Verify that the result indeed is a CEMDS.
setring R;
list sCompressResult2 = compressCEMDS(cemds, 1);
print(sCompressResult2);
↦ [1]:
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : 0
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦     -1     1     0
↦     -1     0     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Empty fan of ambient dimension 2
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ 0
↦
↦ [2]:
↦    1
//Should yield a list with two entries:
// - The same CEMDS as in the first example
// - The verification result (1)
//Third example: Points are transferred to the compressed CEMDS.
setring R;
vector vnumPt1 = [1, 2, 3, 1, 1];
vector vnumPt2 = [1, 2, a, 1, 1];
list svnumPts = vnumPt1, vnumPt2;
def RCompressed3 = compressCEMDS(cemds, 0, svnumPts);
setring RCompressed3;
list sCompressResult3 = gsResult;
print(sCompressResult3);
↦ [1]:
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   1 parameter    : a
```

```
↦ //   minpoly        : 0
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦     -1     1     0
↦     -1     0     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Empty fan of ambient dimension 2
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ 0
↦
↦ [2]:
↦    [1]:
↦       gen(3)+gen(2)+3*gen(1)
↦    [2]:
↦       gen(3)+gen(2)+(a)*gen(1)
//Should yield a list with two entries:
// - The same CEMDS as in the first example
// - A list containing the transferred points ([3, 1, 1] and [a, 1, 1], if the first vari-
ables are compressed first).
//Fourth example: The CEMDS's fan shall also be computed.
setring R;
CEMDS cemdsResult4 = compressCEMDS(cemds, 0, list(), 1);
print(cemdsResult4);
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : 0
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦     -1     1     0
↦     -1     0     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Dimension 2:
↦ 1st maximal cone:
↦ -1, 0,
↦ -1, 1
↦
↦ 2nd maximal cone:
↦ 1, -1,
↦ 0, -1
↦
↦ 3rd maximal cone:
↦ 1, 0,
↦ 0, 1
```

```
↦
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ 0
//Should yield the CEMDS consisting of
// - The rays' matrix
//   (1 0 -1)
//   (0 1 -1)
//   or a matrix with its rows linearly dependent on that
// - The fan consisting of the maximal cones generated by the following rays:
//   * (1,0), (0,1);
//   * (0,1), (-1,-1);
//   * (-1,-1), (1,0);
// - The zero ideal
//Fifth example: Verify that the result indeed is a CEMDS, transfer a point to the com-
pressed CEMDS and compute the CEMDS's fan with a user defined start vector.
setring R;
intvec vzWeight = 1;
def RCompressed5 = compressCEMDS(cemds, 1, svnumPts, 1, vzWeight);
setring RCompressed5;
list sCompressResult5 = gsResult;
print(sCompressResult5);
↦ [1]:
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : 0
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦     -1     1     0
↦     -1     0     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Dimension 2:
↦ 1st maximal cone:
↦ -1, 0,
↦ -1, 1
↦
↦ 2nd maximal cone:
↦ 1, -1,
↦ 0, -1
↦
↦ 3rd maximal cone:
↦ 1, 0,
↦ 0, 1
↦
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ 0
↦
↦ [2]:
↦    1
```

```
↦ [3]:
↦    [1]:
↦       gen(3)+gen(2)+3*gen(1)
↦    [2]:
↦       gen(3)+gen(2)+(a)*gen(1)
//Should yield a list with three entries:
// - The same CEMDS as in the fourth example
// - The verification result (1)
// - A list containing the transferred points ([3, 1, 1] and [a, 1, 1], if the first vari-
ables are compressed first).
```

1.1.0.62 contractCEMDS

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: contractCEMDS(cemds, vfKeepRay); with cemds: CEMDS, vfKeepRay: intvec; contractCEMDS(cemds, vfKeepRay, fComputeFan); with cemds: CEMDS, vfKeepRay: intvec, fComputeFan: int;

Assume: The set of rays to contract is contractible.

Purpose: Blows down the ambient toric variety of a CEMDS; all rays not to be kept are deleted.

Return: The original CEMDS with the specified rays contracted including the contracted fan, if not specified otherwise.

Note: By default: fComputeFan = 1.

Example:

```
LIB "compcox.lib";
//First example: Standard mode
ring R = (0,a),T(1..6),dp;
//Blowup of P2 in [1,1,1] (already stretched)
intmat rvcvzP[4][6] =
1,0,-1,0,0,0,
0,1,-1,0,0,0,
0,0,-1,1,0,1,
0,0,-1,0,1,1;
intmat cvrvz1[4][4] =
-1,-1,-1,-1,
0,0,0,1,
0,0,1,1,
0,1,0,0;
intmat cvrvz2[4][4] =
-1,-1,-1,-1,
0,0,1,0,
0,0,1,1,
0,1,0,0;
intmat cvrvz3[4][4] =
-1,-1,-1,-1,
0,0,0,1,
0,0,1,1,
1,0,0,0;
intmat cvrvz4[4][4] =
-1,-1,-1,-1,
0,0,1,0,
0,0,1,1,
1,0,0,0;
```

```
intmat cvrvz5[4][4] =
-1,-1,-1,-1,
0,0,0,1,
0,1,0,0,
1,0,0,0;
intmat cvrvz6[4][4] =
-1,-1,-1,-1,
0,0,1,0,
0,1,0,0,
1,0,0,0;
intmat cvrvz7[4][4] =
0,0,0,1,
0,0,1,1,
0,1,0,0,
1,0,0,0;
intmat cvrvz8[4][4] =
0,0,1,0,
0,0,1,1,
0,1,0,0,
1,0,0,0;
cone cn1 = coneViaPoints(cvrvz1);
cone cn2 = coneViaPoints(cvrvz2);
cone cn3 = coneViaPoints(cvrvz3);
cone cn4 = coneViaPoints(cvrvz4);
cone cn5 = coneViaPoints(cvrvz5);
cone cn6 = coneViaPoints(cvrvz6);
cone cn7 = coneViaPoints(cvrvz7);
cone cn8 = coneViaPoints(cvrvz8);
fan scnSigma = fanViaCones(cn1, cn2, cn3, cn4, cn5, cn6, cn7, cn8);
poly p1 = a*T(1)-T(2)+T(4)*T(6);
poly p2 = T(2)-T(3)+T(5)*T(6);
ideal spG = p1, p2;
intvec vfKeepRay = 1,1,1,1,1,0;
CEMDS cemds = createCEMDS(rvcvzP, scnSigma, spG);
CEMDS cemdsContracted = contractCEMDS(cemds, vfKeepRay);
//Obtains the CEMDS's ring
def RContracted = cemdsContracted.R;
setring RContracted;
print(cemdsContracted);
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : 0
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦     -1     1     0
↦     -1     0     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Dimension 2:
↦ 1st maximal cone:
↦ -1, 0,
```

```
↦ -1, 1
↦
↦ 2nd maximal cone:
↦ 1, -1,
↦ 0, -1
↦
↦ 3rd maximal cone:
↦ 1, 0,
↦ 0, 1
↦
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ 0
//Should yield
// - A matrix with the following rays as columns
// (1 0 -1)
// (0 1 -1)
//or there must exist a linear isomorphism mapping the output rays to those above.
// - The fan having the three maximal cones generated by the rays
//   [(1,0), (0,1)],
//   [(0,1), (-1,-1)] and
//   [(-1,-1), (1,0)]
// - The zero ideal
//Second example: Do not compute a fan.
setring R;
CEMDS cemdsContracted2 = contractCEMDS(cemds, vfKeepRay, 0);
//Obtains the CEMDS's ring
def RContracted2 = cemdsContracted2.R;
setring RContracted2;
print(cemdsContracted2);
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : 0
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦     -1     1     0
↦     -1     0     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Empty fan of ambient dimension 2
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ 0
//Should yield
// - A matrix with the following rays as columns
// (1 0 -1)
// (0 1 -1)
//or there must exist a linear isomorphism mapping the output rays to those above.
// - The empty fan of dimension two.
// - The zero ideal
```

1.1.0.63 linearBlowup

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: linearBlowup(svnumPts); with svnumPts: list of vectors of numbers; linearBlowup(svnumPts, fVerify); with svnumPts: list of vectors of numbers, fVerify: int; linearBlowup(svnumPts, fVerify, fComputeFan); with svnumPts: list of vectors of numbers, fVerify: int, fComputeFan: int; linearBlowup(svnumPts, fVerify, fComputeFan, vzWeight); with svnumPts: list of vectors of numbers, fVerify: int, fComputeFan: int, vzWeight: intvec.

Assume: The current basering contains at least n >= 2 variables. There are at least n points to blow up containing a subset of n points in general position.

Purpose: Blows up the projective space of dimension (n-1) in some of its points using linear relations for the stretching part only.

Return: If starting with a basering with n variables, the projective space of dimension (n-1) blown up in the points specified. Whether the result is indeed a CEMDS is checked iff fVerify != 0. The CEMDS's fan is computed iff fComputeFan != 0.

Note: By default: fVerify = 0; fComputeFan = 0; vzWeight is an element of the relative interior of the moving cone generated by the columns of the grading matrix rvcvzQ = gale(cemds.rvcvzP) extended by the passed polynomials' degrees.

Example:

```
LIB "compcox.lib";
//First example: No verification or fan computation
ring R = 0,T(1..3),dp;
vector vnumPt1 = [1,0,0];
vector vnumPt2 = [0,1,0];
vector vnumPt3 = [0,0,1];
vector vnumPt4 = [1,1,1];
list svnumPts = vnumPt1, vnumPt2, vnumPt3, vnumPt4;
CEMDS cemdsBlowup = linearBlowup(svnumPts);
↦
↦ The resulting embedded space could not be verified to be a CEMDS.
def RBlowup = cemdsBlowup.R;
setring RBlowup;
print(cemdsBlowup);
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   number of vars : 10
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) T(6) T(7) T(8) T(\
   9) T(10)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      1     0    -1    -1     0     1     0     0     0     0
↦      0    -1     1     1    -1     0     0     0     0     0
↦      1    -1     0     0     0     0    -1     1     0     0
↦      1     0    -1     0     0     0    -1     0     1     0
↦      0    -1     0     1     0     0    -1     0     0     1
↦
```

```
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Empty fan of ambient dimension 5
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ T(3)*T(8)-T(2)*T(9)+T(6)*T(10),
↦ T(6)*T(7)-T(5)*T(8)+T(4)*T(9),
↦ T(3)*T(7)-T(1)*T(9)+T(5)*T(10),
↦ T(2)*T(7)-T(1)*T(8)+T(4)*T(10),
↦ T(3)*T(4)-T(2)*T(5)+T(1)*T(6)
//Should yield
// - A matrix with the following rays as columns
//   (1  0 -1 -1  0 1  0 0 0 0)
//   (0 -1  1  1 -1 0  0 0 0 0)
//   (1 -1  0  0  0 0 -1 1 0 0)
//   (1  0 -1  0  0 0 -1 0 1 0)
//   (0 -1  0  1  0 0 -1 0 0 1)
//   or there must exist a linear isomorphism mapping the output rays to those above.
// - The empty fan of dimension 5;
// - The ideal generated by
//   T(3)*T(8)-T(2)*T(9)+T(6)*T(10),
//   T(6)*T(7)-T(5)*T(8)+T(4)*T(9),
//   T(3)*T(7)-T(1)*T(9)+T(5)*T(10),
//   T(2)*T(7)-T(1)*T(8)+T(4)*T(10),
//   T(3)*T(4)-T(2)*T(5)+T(1)*T(6).
setring R;
//Second example: Verification and fan computation
CEMDS cemdsBlowup2 = linearBlowup(svnumPts, 1, 1);
↦
↦ The resulting embedded space was successfully verified to be a CEMDS.
def RBlowup2 = cemdsBlowup2.R;
setring RBlowup2;
print(cemdsBlowup2);
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   number of vars : 10
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) T(6) T(7) T(8) T(\
   9) T(10)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      1     0    -1    -1     0     1     0     0     0     0
↦      0    -1     1     1    -1     0     0     0     0     0
↦      1    -1     0     0     0     0    -1     1     0     0
↦      1     0    -1     0     0     0    -1     0     1     0
↦      0    -1     0     1     0     0    -1     0     0     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Dimension 4:
↦ 1st maximal cone:
↦  0,  0, -1, -1,
↦ -1, -1,  1,  1,
↦  0, -1,  0,  0,
↦  0,  0, -1,  0,
↦  0, -1,  0,  1
```

```
↦
↦ 2nd maximal cone:
↦ 0,  0,  0, -1,
↦ 0, -1,  0,  1,
↦ 0, -1, -1,  0,
↦ 0,  0, -1,  0,
↦ 1, -1, -1,  1
↦
↦ 3rd maximal cone:
↦  0,  0, -1, 0,
↦ -1,  0,  1, 0,
↦  0, -1,  0, 0,
↦  0, -1, -1, 0,
↦  0, -1,  0, 1
↦
↦ 4th maximal cone:
↦ -1, 0, 0,  0,
↦  1, 0, 0, -1,
↦  0, 1, 0, -1,
↦ -1, 0, 1,  0,
↦  0, 0, 0, -1
↦
↦ 5th maximal cone:
↦  0, 0, 0, -1,
↦ -1, 0, 0,  1,
↦  0, 1, 0,  0,
↦  0, 0, 1,  0,
↦  0, 0, 0,  1
↦
↦ 6th maximal cone:
↦ 1,  0, 0, -1,
↦ 0,  0, 0,  1,
↦ 0, -1, 0,  0,
↦ 0, -1, 1,  0,
↦ 0, -1, 0,  1
↦
↦ 7th maximal cone:
↦ 0, 1, -1,  0,
↦ 0, 0,  1,  0,
↦ 0, 1,  0, -1,
↦ 1, 1, -1, -1,
↦ 0, 0,  0, -1
↦
↦ 8th maximal cone:
↦ 1, 0, -1, 0,
↦ 0, 0,  1, 0,
↦ 0, 1,  0, 0,
↦ 0, 0, -1, 0,
↦ 0, 0,  0, 1
↦
↦ 9th maximal cone:
↦ 0, 0, 1, -1,
↦ 0, 0, 0,  1,
↦ 0, 1, 1,  0,
↦ 0, 0, 1,  0,
↦ 1, 0, 0,  1
↦
↦ 10th maximal cone:
```

```
↦ 1, -1, 1, -1,
↦ 0,  1, 0,  1,
↦ 0,  0, 1,  0,
↦ 0, -1, 1,  0,
↦ 0,  0, 0,  1
↦
↦ 11th maximal cone:
↦ 1,  0, 0, 0,
↦ 0, -1, 0, 0,
↦ 0, -1, 0, 0,
↦ 0,  0, 1, 0,
↦ 0, -1, 0, 1
↦
↦ 12th maximal cone:
↦ 1,  0, 0,  0,
↦ 0, -1, 0,  0,
↦ 0,  0, 1, -1,
↦ 0,  0, 0, -1,
↦ 0,  0, 0, -1
↦
↦ 13th maximal cone:
↦  0, 0, 1,  0,
↦ -1, 0, 0,  0,
↦ -1, 1, 1, -1,
↦  0, 0, 1, -1,
↦ -1, 0, 0, -1
↦
↦ 14th maximal cone:
↦  0, 0, 1, 0,
↦ -1, 0, 0, 0,
↦  0, 0, 1, 0,
↦  0, 1, 1, 0,
↦  0, 0, 0, 1
↦
↦ 15th maximal cone:
↦ 1,  0, 1,  0,
↦ 0, -1, 0, -1,
↦ 0,  0, 1, -1,
↦ 0,  0, 1,  0,
↦ 0,  0, 0, -1
↦
↦ Dimension 5:
↦ 1st maximal cone:
↦  0, -1, -1, 0,  0,
↦ -1,  1,  1, 0,  0,
↦ -1,  0,  0, 0, -1,
↦  0, -1,  0, 1, -1,
↦ -1,  0,  1, 0, -1
↦
↦ 2nd maximal cone:
↦  0, -1, 0, -1, 0,
↦ -1,  1, 0,  1, 0,
↦  0,  0, 1,  0, 0,
↦  0,  0, 0, -1, 0,
↦  0,  1, 0,  0, 1
↦
↦ 3rd maximal cone:
↦ -1,  0, 0,  0,  0,
```

```
↦  1, -1, 0, -1,  0,
↦  0,  0, 1, -1, -1,
↦ -1,  0, 0,  0, -1,
↦  0,  0, 0, -1, -1
↦
↦ 4th maximal cone:
↦  0, -1,  0, 0, 0,
↦ -1,  1, -1, 0, 0,
↦  0,  0, -1, 0, 0,
↦  0,  0,  0, 1, 0,
↦  0,  1, -1, 0, 1
↦
↦ 5th maximal cone:
↦ 1, 0,  0, -1, -1,
↦ 0, 0,  0,  1,  1,
↦ 0, 0, -1,  0,  0,
↦ 0, 0, -1, -1,  0,
↦ 0, 1, -1,  0,  1
↦
↦ 6th maximal cone:
↦ 0, 0, -1, 1, -1,
↦ 0, 0,  1, 0,  1,
↦ 1, 0,  0, 1,  0,
↦ 0, 1, -1, 1,  0,
↦ 0, 0,  0, 0,  1
↦
↦ 7th maximal cone:
↦ 1,  0, 0,  0,  0,
↦ 0, -1, 0, -1,  0,
↦ 0,  0, 0, -1, -1,
↦ 0,  0, 0,  0, -1,
↦ 0,  0, 1, -1, -1
↦
↦ 8th maximal cone:
↦ 0,  0, 0, 1,  0,
↦ 0, -1, 0, 0, -1,
↦ 0,  0, 1, 1, -1,
↦ 1,  0, 0, 1,  0,
↦ 0,  0, 0, 0, -1
↦
↦ 9th maximal cone:
↦ 1, 0, -1, 1,  0,
↦ 0, 0,  1, 0,  0,
↦ 0, 1,  0, 1, -1,
↦ 0, 0, -1, 1, -1,
↦ 0, 0,  0, 0, -1
↦
↦ 10th maximal cone:
↦ 1, 0, 0, 1, -1,
↦ 0, 0, 0, 0,  1,
↦ 0, 0, 0, 1,  0,
↦ 0, 0, 1, 1,  0,
↦ 0, 1, 0, 0,  1
↦
↦ 11th maximal cone:
↦ 1,  0, 1, 0,  0,
↦ 0, -1, 0, 0,  0,
↦ 0, -1, 1, 0, -1,
```

```
↦ 0,  0, 1, 1, -1,
↦ 0, -1, 0, 0, -1
↦
↦ 12th maximal cone:
↦ 1,  0, 0, 1, 0,
↦ 0, -1, 0, 0, 0,
↦ 0,  0, 1, 1, 0,
↦ 0,  0, 0, 1, 0,
↦ 0,  0, 0, 0, 1
↦
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ T(3)*T(8)-T(2)*T(9)+T(6)*T(10),
↦ T(6)*T(7)-T(5)*T(8)+T(4)*T(9),
↦ T(3)*T(7)-T(1)*T(9)+T(5)*T(10),
↦ T(2)*T(7)-T(1)*T(8)+T(4)*T(10),
↦ T(3)*T(4)-T(2)*T(5)+T(1)*T(6)
//Should yield the same as above, but with an appropriate fan.
```

1.1.0.64 linearQuadricBlowupP2

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: linearQuadricBlowupP2(svnumPts); with svnumPts: list of vectors of numbers; linearQuadricBlowupP2(svnumPts, fVerify); with svnumPts: list of vectors of numbers, fVerify: int; linearQuadricBlowupP2(svnumPts, fVerify, fComputeFan); with svnumPts: list of vectors of numbers, fVerify: int, fComputeFan: int; linearQuadricBlowupP2(svnumPts, fVerify, fComputeFan, vzWeight); with svnumPts: list of vectors of numbers, fVerify: int, fComputeFan: int, vzWeight: intvec.

Assume: The current basering contains at exactly three variables (thus representing the Cox ring of P2). There are at least 3 points to blow up containing a subset of 3 points in general position.

Purpose: Blows up the projective space of dimension 2 in some of its points using relations of degree 1 and 2 for the stretching part only.

Return: The projective space of dimension 2 blown up in the points specified. Whether the result is indeed a CEMDS is checked iff fVerify != 0. The CEMDS's fan is computed iff fComputeFan != 0.

Note: By default: fVerify = 0; fComputeFan = 0; vzWeight is an element of the relative interior of the moving cone generated by the columns of the grading matrix rvcvzQ = gale(cemds.rvcvzP) extended by the passed polynomials' degrees.

Example:

```
LIB "compcox.lib";
ring R = (0,a,b),T(1..3),dp;
vector vnumPt1 = [1,0,0];
vector vnumPt2 = [0,1,0];
vector vnumPt3 = [0,0,1];
vector vnumPt4 = [1,1,1];
vector vnumPt5 = [1,1,0];
list svnumPts = vnumPt1, vnumPt2, vnumPt3, vnumPt4, vnumPt5;
CEMDS cemdsBlowup = linearQuadricBlowupP2(svnumPts);
```

```
↦
↦ The resulting embedded space could not be verified to be a CEMDS.
def RBlowup = cemdsBlowup.R;
setring RBlowup;
print(cemdsBlowup);
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   2 parameter    : a b
↦ //   minpoly        : 0
↦ //   number of vars : 11
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) T(6) T(7) T(8) T(\
   9) T(10) T(11)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦     -1     1     0     0     1    -1     0     0     0     0     0
↦      1    -1     0     0     0     0    -1     1     0     0     0
↦      1     0    -1     1     0    -1    -2     0     2     0     0
↦      1    -1    -1     1     0     0    -2     0     1     1     0
↦     -1     0     1     0     0     0     1     0    -1     0     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Empty fan of ambient dimension 5
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ -T(3)*T(8)*T(11)+T(2)*T(9)-T(6)*T(10),
↦ -T(3)*T(7)*T(11)+T(1)*T(9)-T(5)*T(10),
↦ T(4)*T(9)*T(11)+T(6)*T(7)-T(5)*T(8),
↦ T(4)*T(10)*T(11)+T(2)*T(7)-T(1)*T(8),
↦ -T(3)*T(4)*T(11)^2+T(2)*T(5)-T(1)*T(6)
//Should yield
// - A matrix with the following rays as columns
//   (-1  1  0 0 1 -1  0 0  0 0 0)
//   ( 1 -1  0 0 0  0 -1 1  0 0 0)
//   ( 1  0 -1 1 0 -1 -2 0  2 0 0)
//   ( 1  1 -1 1 0  0 -2 0  1 1 0)
//   (-1  0  1 0 0  0  1 0 -1 0 1)
//   or there must exist a linear isomorphism mapping the output rays to those above.
// - The empty fan of dimension 5;
// - The ideal generated by
//   -T(3)*T(8)*T(11)+T(2)*T(9)-T(6)*T(10),
//   -T(3)*T(7)*T(11)+T(1)*T(9)-T(5)*T(10),
//   T(4)*T(9)*T(11)+T(6)*T(7)-T(5)*T(8),
//   T(4)*T(10)*T(11)+T(2)*T(7)-T(1)*T(8),
//   -T(3)*T(4)*T(11)^2+T(2)*T(5)-T(1)*T(6).
setring R;
//Second example: Verification and fan computation
CEMDS cemdsBlowup2 = linearQuadricBlowupP2(svnumPts, 1, 1);
↦
↦ The resulting embedded space was successfully verified to be a CEMDS.
def RBlowup2 = cemdsBlowup2.R;
setring RBlowup2;
print(cemdsBlowup2);
↦
↦ The CEMDS's ring:
```

```
↦ //   characteristic : 0
↦ //   2 parameter    : a b
↦ //   minpoly        : 0
↦ //   number of vars : 11
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) T(6) T(7) T(8) T(\
   9) T(10) T(11)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦     -1     1     0     0     1    -1     0     0     0     0     0
↦      1    -1     0     0     0     0    -1     1     0     0     0
↦      1     0    -1     1     0    -1    -2     0     2     0     0
↦      1    -1    -1     1     0     0    -2     0     1     1     0
↦     -1     0     1     0     0     0     1     0    -1     0     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Dimension 4:
↦ 1st maximal cone:
↦ 0, -1, 0,  0,
↦ 1,  0, 0,  0,
↦ 0, -1, 0, -1,
↦ 0,  0, 1, -1,
↦ 0,  0, 0,  1
↦
↦ 2nd maximal cone:
↦ 0,  0, -1, 0,
↦ 0,  0,  1, 0,
↦ 1,  2,  1, 0,
↦ 1,  1,  1, 0,
↦ 0, -1, -1, 1
↦
↦ 3rd maximal cone:
↦ 0, 0, -1, 0,
↦ 1, 0,  1, 0,
↦ 0, 0,  1, 0,
↦ 0, 1,  1, 0,
↦ 0, 0, -1, 1
↦
↦ 4th maximal cone:
↦ 0,  0,  1,  0,
↦ 1,  0, -1,  0,
↦ 0,  2,  0, -1,
↦ 0,  1, -1, -1,
↦ 0, -1,  0,  1
↦
↦ 5th maximal cone:
↦ 1, 0, 0, 0,
↦ 0, 0, 0, 0,
↦ 0, 0, 1, 0,
↦ 0, 1, 1, 0,
↦ 0, 0, 0, 1
↦
↦ 6th maximal cone:
↦ 1, 0,  0, 0,
↦ 0, 1,  0, 0,
↦ 0, 0,  2, 0,
```

```
↦ 0, 0,  1, 0,
↦ 0, 0, -1, 1
↦
↦ Dimension 5:
↦ 1st maximal cone:
↦  0, -1, -1, 0,  0,
↦ -1,  1,  0, 0,  0,
↦ -2,  1, -1, 0, -1,
↦ -2,  1,  0, 0, -1,
↦  1, -1,  0, 1,  1
↦
↦ 2nd maximal cone:
↦ -1, 0, -1,  0,  0,
↦  1, 1,  0,  0, -1,
↦  1, 0, -1, -1, -2,
↦  1, 0,  0, -1, -2,
↦ -1, 0,  0,  1,  1
↦
↦ 3rd maximal cone:
↦ 0, -1, -1, 0, 0,
↦ 0,  0,  1, 0, 0,
↦ 1, -1,  1, 0, 0,
↦ 1,  0,  1, 1, 0,
↦ 0,  0, -1, 0, 1
↦
↦ 4th maximal cone:
↦ -1, 0,  0, 0, -1,
↦  0, 0,  0, 0,  1,
↦ -1, 1,  2, 0,  1,
↦  0, 1,  1, 1,  1,
↦  0, 0, -1, 0, -1
↦
↦ 5th maximal cone:
↦ 0, -1,  0, 0,  0,
↦ 0,  0, -1, 0,  0,
↦ 0, -1, -2, 0, -1,
↦ 0,  0, -2, 1, -1,
↦ 1,  0,  1, 0,  1
↦
↦ 6th maximal cone:
↦ 0, -1,  0, 0, 0,
↦ 0,  0, -1, 0, 0,
↦ 1, -1, -2, 0, 0,
↦ 1,  0, -2, 1, 0,
↦ 0,  0,  1, 0, 1
↦
↦ 7th maximal cone:
↦  0,  0, -1,  1, -1,
↦  0, -1,  0, -1,  1,
↦  2, -2, -1,  0,  1,
↦  1, -2,  0, -1,  1,
↦ -1,  1,  0,  0, -1
↦
↦ 8th maximal cone:
↦ -1,  0, 0, 0,  0,
↦  0,  0, 0, 0, -1,
↦ -1,  2, 1, 0, -2,
↦  0,  1, 1, 1, -2,
```

```
↦  0, -1, 0, 0,  1
↦
↦ 9th maximal cone:
↦  0, -1,  0, 0,  0,
↦ -1,  1,  0, 0,  0,
↦ -2,  1,  2, 0, -1,
↦ -2,  1,  1, 0, -1,
↦  1, -1, -1, 1,  1
↦
↦ 10th maximal cone:
↦ 1, 0, -1, -1,  0,
↦ 0, 1,  0,  1, -1,
↦ 0, 0, -1,  1, -2,
↦ 0, 0,  0,  1, -2,
↦ 0, 0,  0, -1,  1
↦
↦ 11th maximal cone:
↦ 1, 0, -1, 0, -1,
↦ 0, 1,  0, 0,  1,
↦ 0, 0, -1, 0,  1,
↦ 0, 0,  0, 1,  1,
↦ 0, 0,  0, 0, -1
↦
↦ 12th maximal cone:
↦ -1, 0,  0, 0,  0,
↦  1, 1,  0, 0,  0,
↦  1, 0, -1, 0,  2,
↦  1, 0, -1, 0,  1,
↦ -1, 0,  1, 1, -1
↦
↦ 13th maximal cone:
↦  0, 0, -1,  1,  0,
↦  0, 0,  0, -1, -1,
↦  2, 0, -1,  0, -2,
↦  1, 1,  0, -1, -2,
↦ -1, 0,  0,  0,  1
↦
↦ 14th maximal cone:
↦ 1, -1,  1, -1,  0,
↦ 0,  0, -1,  1, -1,
↦ 0, -1,  0,  1, -2,
↦ 0,  0, -1,  1, -2,
↦ 0,  0,  0, -1,  1
↦
↦ 15th maximal cone:
↦ 1, 0,  0, 0, -1,
↦ 0, 0,  0, 0,  1,
↦ 0, 0,  2, 1,  1,
↦ 0, 1,  1, 1,  1,
↦ 0, 0, -1, 0, -1
↦
↦ 16th maximal cone:
↦ 1,  1, -1, 0,  0,
↦ 0, -1,  0, 0, -1,
↦ 0,  0, -1, 0, -2,
↦ 0, -1,  0, 1, -2,
↦ 0,  0,  0, 0,  1
↦
```

```
↦ 17th maximal cone:
↦  0,  0, 0,  1,  0,
↦ -1,  0, 0, -1,  0,
↦ -2,  2, 0,  0, -1,
↦ -2,  1, 0, -1, -1,
↦  1, -1, 1,  0,  1
↦
↦ 18th maximal cone:
↦  0,  0, 0,  1, 0,
↦ -1,  0, 0, -1, 0,
↦ -2,  2, 1,  0, 0,
↦ -2,  1, 1, -1, 0,
↦  1, -1, 0,  0, 1
↦
↦ 19th maximal cone:
↦  0, 0, 0,  1,  0,
↦  0, 0, 0, -1, -1,
↦  2, 0, 1,  0, -2,
↦  1, 1, 1, -1, -2,
↦ -1, 0, 0,  0,  1
↦
↦ 20th maximal cone:
↦ 1,  0, 0, 0,  0,
↦ 0, -1, 0, 0,  0,
↦ 0, -2, 0, 0, -1,
↦ 0, -2, 0, 1, -1,
↦ 0,  1, 1, 0,  1
↦
↦ 21st maximal cone:
↦ 1, 0, -1,  1,  0,
↦ 0, 1,  1, -1, -1,
↦ 0, 0,  1,  0, -2,
↦ 0, 0,  1, -1, -2,
↦ 0, 0, -1,  0,  1
↦
↦ 22nd maximal cone:
↦ 1, 0,  0, 0, 0,
↦ 0, 1,  0, 0, 0,
↦ 0, 0, -1, 0, 0,
↦ 0, 0, -1, 1, 0,
↦ 0, 0,  1, 0, 1
↦
↦ 23rd maximal cone:
↦ 1, 0, -1,  1,  0,
↦ 0, 1,  1, -1,  0,
↦ 0, 0,  1,  0,  2,
↦ 0, 0,  1, -1,  1,
↦ 0, 0, -1,  0, -1
↦
↦ 24th maximal cone:
↦ 1,  0, 0,  1,  0,
↦ 0, -1, 0, -1,  0,
↦ 0, -2, 0,  0, -1,
↦ 0, -2, 0, -1, -1,
↦ 0,  1, 1,  0,  1
↦
↦ 25th maximal cone:
↦ 1, 0,  0,  1,  0,
```

```
↦ 0, 1, -1, -1,  0,
↦ 0, 0, -2,  0, -1,
↦ 0, 0, -2, -1, -1,
↦ 0, 0,  1,  0,  1
↦
↦ 26th maximal cone:
↦ 1, 0,  1,  0, 0,
↦ 0, 0, -1,  0, 0,
↦ 0, 1,  0,  2, 0,
↦ 0, 1, -1,  1, 0,
↦ 0, 0,  0, -1, 1
↦
↦ 27th maximal cone:
↦ 1, 0,  1, 0,  0,
↦ 0, 0, -1, 0,  0,
↦ 0, 0,  0, 1,  2,
↦ 0, 1, -1, 1,  1,
↦ 0, 0,  0, 0, -1
↦
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ -T(3)*T(8)*T(11)+T(2)*T(9)-T(6)*T(10),
↦ -T(3)*T(7)*T(11)+T(1)*T(9)-T(5)*T(10),
↦ T(4)*T(9)*T(11)+T(6)*T(7)-T(5)*T(8),
↦ T(4)*T(10)*T(11)+T(2)*T(7)-T(1)*T(8),
↦ -T(3)*T(4)*T(11)^2+T(2)*T(5)-T(1)*T(6)
//Should yield the same as above, but with an appropriate fan.
```

1.1.0.65 modifyCEMDS

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: modifyCEMDS(cemds, sRaysFan); with cemds: CEMDS, sRaysFan: list of an intmat and a fan; modifyCEMDS(cemds, sRaysFan, fVerify); with cemds: CEMDS, sRaysFan: list of an intmat and a fan, fVerify: int; modifyCEMDS(cemds, sRaysFan, fVerify, rvfFakeVars); with cemds: CEMDS, sRaysFan: list of an intmat and a fan, fVerify: int, rvfFakeVars: intvec; modifyCEMDS(cemds, sRaysFan, fVerify, rvfFakeVars, svnumPts); with cemds: CEMDS, sRaysFan: list of an intmat and a fan, fVerify: int, rvfFakeVars: intvec, svnumPts: list of vectors of numbers.

Assume: (i) The matrix passed in "sRaysFan" is the matrix "cemds.rvcvzP" extended by further columns that are integer sums of columns of this matrix; (ii) The fan passed in "sRaysFan" has the columns of the matrix passed in "sRaysFan" as its rays; (iii) rvfFakeVars marks only indices of fake variables as true (i.e. variables for which there exists a generator of the type var(i) - p with p not containing var(i) in the CEMDS's ideal); (iv) All points to modify along with the CEMDS, if there were any passed, are contained in the CEMDS to modify.

Purpose: Blows up the ambient toric variety of a CEMDS by a refining of its fan. Optionally checks whether the result really is a CEMDS and transfers points from the unmodified CEMDS to the modified one.

Return: If no optional output was requested, the modified CEMDS only. If only verification was requested, but no points or an empty list of points to modify along with the CEMDS were passed, a list containing the following (in order of appearance): (i) the modified CEMDS that is obtained by blowing up the passed CEMDS's ambient toric variety by adding some rays to its fan and performing stellar subdivision (with or without a fan, depending on fComputeFan) (always present); (ii) the verification result (present as fVerify was passed and fVerify != 0). If a non-empty list of points to modify along with the CEMDS was passed, the modified CEMDS's basering.

Side effects:
Iff a non-empty list of points to modify along with the CEMDS was passed, exports a list gsResult containing the following (if present and in order of appearance): (i) the modified CEMDS that is obtained by blowing up the passed CEMDS's ambient toric variety by adding some rays to its fan and performing stellar subdivision (with or without a fan, depending on fComputeFan) (always present); (ii) the verification result (present iff fVerify was passed and fVerify != 0), (iii) the list of modified points (present as there was a list passed).

Note: By default: fVerify = 0; rvfFakeVars = 0:nvars; svnumPts = list().

Example:

```
LIB "compcox.lib";
//First example: No optional arguments.
ring R = 0,T(1..3),dp;
//The CEMDS's rays' matrix
intmat rvcvzP[2][3] =
1,0,-1,
0,1,-1;
//The CEMDS's fan
intmat cvrvz1[2][2] =
1,0,
0,1;
intmat cvrvz2[2][2] =
0,1,
-1,-1;
intmat cvrvz3[2][2] =
-1,-1,
1,0;
cone cn1 = coneViaPoints(cvrvz1);
cone cn2 = coneViaPoints(cvrvz2);
cone cn3 = coneViaPoints(cvrvz3);
fan scnSigma = fanViaCones(cn1, cn2, cn3);
//The CEMDS's embedding ideal
ideal spG = 0;
CEMDS cemds = createCEMDS(rvcvzP, scnSigma, spG);
//The modified CEMDS's rays' matrix and fan:
intmat rvcvzPDash[2][6] = (
1,0,-1,1,0,-1,
0,1,-1,1,-1,0);
//The CEMDS's fan
intmat cvrvz1Dash[2][2] =
1,0,
1,1;
intmat cvrvz2Dash[2][2] =
1,1,
0,1;
intmat cvrvz3Dash[2][2] =
```

```
0,1,
-1,0;
intmat cvrvz4Dash[2][2] =
-1,0,
-1,-1;
intmat cvrvz5Dash[2][2] =
-1,-1,
0,-1;
intmat cvrvz6Dash[2][2] =
0,-1,
1,0;
cone cn1Dash = coneViaPoints(cvrvz1Dash);
cone cn2Dash = coneViaPoints(cvrvz2Dash);
cone cn3Dash = coneViaPoints(cvrvz3Dash);
cone cn4Dash = coneViaPoints(cvrvz4Dash);
cone cn5Dash = coneViaPoints(cvrvz5Dash);
cone cn6Dash = coneViaPoints(cvrvz6Dash);
fan scnSigmaDash = fanViaCones(cn1Dash, cn2Dash, cn3Dash, cn4Dash, cn5Dash, cn6Dash);
list sRaysFan = rvcvzPDash, scnSigmaDash;
CEMDS cemdsResult = modifyCEMDS(cemds, sRaysFan);
//Obtains the CEMDS's ring
def RModified = cemdsResult.R;
setring RModified;
print(cemdsResult);
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) T(6)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      1     0    -1     1     0    -1
↦      0     1    -1     1    -1     0
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Dimension 2:
↦ 1st maximal cone:
↦ -1, -1,
↦  0, -1
↦
↦ 2nd maximal cone:
↦ -1,  0,
↦ -1, -1
↦
↦ 3rd maximal cone:
↦ -1, 0,
↦  0, 1
↦
↦ 4th maximal cone:
↦ 1,  0,
↦ 0, -1
↦
↦ 5th maximal cone:
↦ 0, 1,
↦ 1, 1
```

```
↦
↦ 6th maximal cone:
↦ 1, 1,
↦ 0, 1
↦
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ 0
//Should yield the CEMDS consisting of
// - The matrix rvcvzPDash;
// - The fan scnSigmaDash;
// - The zero ideal.
```

1.1.0.66 modifyCEMDS2

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: modifyCEMDS2(cemds, rvnVanishingDegrees); with cemds: CEMDS, rvnVanishingDegrees: intvec; modifyCEMDS2(cemds, rvnVanishingDegrees, fVerify); with cemds: CEMDS, rvnVanishingDegrees: intvec, fVerify: int; modifyCEMDS2(cemds, rvnVanishingDegrees, fVerify, rvfFakeVars); with cemds: CEMDS, rvnVanishingDegrees: intvec, fVerify: int, rvfFakeVars: intvec; modifyCEMDS2(cemds, rvnVanishingDegrees, fVerify, rvfFakeVars, svnumPts); with cemds: CEMDS, rvnVanishingDegrees: intvec, fVerify: int, rvfFakeVars: intvec, svnumPts: list of vectors of numbers; modifyCEMDS2(cemds, rvnVanishingDegrees, fVerify, rvfFakeVars, svnumPts, fComputeFan); with cemds: CEMDS, rvnVanishingDegrees: intvec, fVerify: int, rvfFakeVars: intvec, svnumPts: list of vectors of numbers, fComputeFan: int.

Assume: (i) rvfFakeVars marks only indices of fake variables as true (i.e. variables for which there exists a generator of the type var(i) - p with p not containing var(i) in the CEMDS's ideal) (ii) All points to modify along with the CEMDS, if there were any passed, are contained in the CEMDS to modify.

Purpose: Blows up the ambient toric variety of a CEMDS by adding some rays to its fan and performing stellar subdivision. Optionally checks whether the result really is a CEMDS and transfers points from the unmodified CEMDS to the modified one.

Return: If no optional output was requested, the modified CEMDS only. If only verification was requested, but no points or an empty list of points to modify along with the CEMDS were passed, a list containing the following (in order of appearance): (i) the modified CEMDS that is obtained by blowing up the passed CEMDS's ambient toric variety by adding some rays to its fan and performing stellar subdivision (with or without a fan, depending on fComputeFan) (always present); (ii) the verification result (present as fVerify was passed and fVerify != 0). If a non-empty list of points to modify along with the CEMDS was passed, the modified CEMDS's basering.

Side effects:
Iff a non-empty list of points to modify along with the CEMDS was passed, exports a list gsResult containing the following (if present and in order of ap-

pearance): (i) the modified CEMDS that is obtained by blowing up the passed CEMDS's ambient toric variety by adding some rays to its fan and performing stellar subdivision (with or without a fan, depending on fComputeFan) (always present); (ii) the verification result (present iff fVerify was passed and fVerify != 0), (iii) the list of modified points (present as there was a list passed).

Note: By default: fVerify = 0; rvfFakeVars = 0:nvars; svnumPts = list(); fComputeFan = 0.

Example:

```
LIB "compcox.lib";
//First example: No optional arguments.
ring R = 0,T(1..3),dp;
//The CEMDS's rays' matrix
intmat rvcvzP[2][3] =
1,0,-1,
0,1,-1;
//The CEMDS's fan
intmat cvrvz1[2][2] =
1,0,
0,1;
intmat cvrvz2[2][2] =
0,1,
-1,-1;
intmat cvrvz3[2][2] =
-1,-1,
1,0;
cone cn1 = coneViaPoints(cvrvz1);
cone cn2 = coneViaPoints(cvrvz2);
cone cn3 = coneViaPoints(cvrvz3);
fan scnSigma = fanViaCones(cn1, cn2, cn3);
//The CEMDS's embedding ideal
ideal spG = 0;
CEMDS cemds = createCEMDS(rvcvzP, scnSigma, spG);
intvec rvnVanishingDegrees = 0,1,1;
CEMDS cemdsResult = modifyCEMDS2(cemds, rvnVanishingDegrees);
//Obtains the CEMDS's ring
def RModified = cemdsResult.R;
setring RModified;
print(cemdsResult);
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      1     0    -1    -1
↦      0     1    -1     0
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Empty fan of ambient dimension 2
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ 0
```

```
//Should yield the CEMDS consisting of
// - A matrix with the entries
//   (1 0 -1 -1)
//   (0 1 -1  0)
// - The empty fan of ambient dimension 2.
// - The zero ideal.
//Second example: Verify that the result indeed is a CEMDS
setring R;
list sModifyResult = modifyCEMDS2(cemds, rvnVanishingDegrees, 1);
↦ WARNING: The verification tests do not include the test whether the CEMDS\
   's ring is normal. Please check that manually.
print(sModifyResult);
↦ [1]:
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      1     0    -1    -1
↦      0     1    -1     0
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Empty fan of ambient dimension 2
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ 0
↦
↦ [2]:
↦    1
//Should yield a list with two entries:
// - the same CEMDS as in the first example
// - the verification result (1).
kill cemds;
kill rvcvzP;
kill cvrvz1, cvrvz2, cvrvz3;
kill cn1, cn2, cn3;
kill scnSigma;
kill spG;
kill rvnVanishingDegrees;
kill cemdsResult;
kill sModifyResult;
kill RModified;
kill R;
//Third example: Fake variables are marked.
ring R = (0,a),T(1..9),dp;
//The CEMDS's rays' matrix
intmat rvcvzP[5][9] =
1,  0, -1,  1,  0, -1, 0, 0, 0,
0,  1, -1,  1, -1,  0, 0, 0, 0,
0, -1,  0, -1,  0,  0, 1, 0, 0,
-1,  0,  0, -1,  0,  0, 0, 1, 0,
-1,  0,  0,  0, -1,  0, 0, 0, 1;
//The CEMDS's fan
```

```
intmat cvrvz;
list scn = list();
list scnList = list();
cvrvz = intmat(intvec(
0, -1,  0,  0,  0,
-1, -1,  0,  0,  1,
0,  1, -1,  0, -1,
1,  1, -1, -1, -1
), 4, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
-1,  0,  0,  0,  0,
1,  1, -1, -1, -1,
-1, -1,  0,  0,  1,
1,  0,  0, -1, -1
), 4, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
-1,  0,  0,  0,  0,
0, -1,  0,  0,  0,
1,  0,  0, -1, -1,
0,  1, -1,  0, -1
), 4, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
-1,  0, 0, 0, 0,
0, -1, 0, 0, 0,
0,  0, 1, 0, 0,
0,  0, 0, 1, 0,
-1, -1, 0, 0, 1
), 5, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
-1,  0,  0, 0,  0,
0, -1,  0, 0,  0,
0,  1, -1, 0, -1,
0,  0,  0, 1,  0,
-1, -1,  0, 0,  1
), 5, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
-1,  0, 0, 0, 0,
-1, -1, 0, 0, 1,
0,  0, 1, 0, 0,
0,  0, 0, 1, 0,
0,  0, 0, 0, 1
), 5, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
-1,  0,  0, 0,  0,
-1, -1,  0, 0,  1,
0,  1, -1, 0, -1,
```

```
0,  0,  0, 1,  0,
0,  0,  0, 0,  1
), 5, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
-1,  0, 0,  0,  0,
0, -1, 0,  0,  0,
0,  0, 1,  0,  0,
1,  0, 0, -1, -1,
-1, -1, 0,  0,  1
), 5, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
-1, -1, 0, 0, 1,
0, -1, 0, 0, 0,
0,  0, 1, 0, 0,
0,  0, 0, 1, 0,
0,  0, 0, 0, 1
), 5, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
-1,  0,  0,  0,  0,
-1, -1,  0,  0,  1,
0,  1, -1,  0, -1,
1,  1, -1, -1, -1,
0,  0,  0,  0,  1
), 5, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
-1, 0,  0, 0,  0,
0, 1, -1, 0, -1,
0, 0,  1, 0,  0,
0, 0,  0, 1,  0,
0, 0,  0, 0,  1
), 5, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
-1, -1, 0,  0,  1,
0, -1, 0,  0,  0,
0,  0, 1,  0,  0,
1,  0, 0, -1, -1,
0,  0, 0,  0,  1
), 5, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
-1, 0,  0,  0,  0,
0, 0,  0,  0,  1,
0, 0,  1,  0,  0,
1, 1, -1, -1, -1,
0, 1, -1,  0, -1
), 5, 5);
scn = coneViaPoints(cvrvz);
```

```
scnList = scnList + scn;
cvrvz = intmat(intvec(
-1, -1,  0,  0,  1,
0, -1,  0,  0,  0,
1,  1, -1, -1, -1,
1,  0,  0, -1, -1,
0,  0,  0,  0,  1
), 5, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
1,  0, 0, -1, -1,
0, -1, 0,  0,  0,
0,  0, 1,  0,  0,
0,  0, 0,  1,  0,
0,  0, 0,  0,  1
), 5, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
-1, 0,  0,  0,  0,
1, 0,  0, -1, -1,
0, 0,  1,  0,  0,
0, 1, -1,  0, -1,
1, 1, -1, -1, -1
), 5, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
0, 1, -1,  0, -1,
0, 0,  0,  0,  1,
0, 0,  1,  0,  0,
0, 0,  0,  1,  0,
1, 1, -1, -1, -1
), 5, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
0,  0,  0,  0,  1,
0, -1,  0,  0,  0,
1,  1, -1, -1, -1,
0,  0,  0,  1,  0,
1,  0,  0, -1, -1
), 5, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
0,  1, -1,  0, -1,
0, -1,  0,  0,  0,
1,  0,  0, -1, -1,
0,  0,  0,  1,  0,
1,  1, -1, -1, -1
), 5, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
1, 0,  0, -1, -1,
1, 1, -1, -1, -1,
```

```
0, 0,  1,  0,  0,
0, 0,  0,  1,  0,
0, 0,  0,  0,  1
), 5, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
cvrvz = intmat(intvec(
0, 1, -1,  0, -1,
1, 0,  0, -1, -1,
0, 0,  1,  0,  0,
0, 0,  0,  1,  0,
1, 1, -1, -1, -1
), 5, 5);
scn = coneViaPoints(cvrvz);
scnList = scnList + scn;
fan scnSigma = fanViaCones(scnList);
//The CEMDS's embedding ideal
poly p1 = T(7) - T(2)*T(4) + a*T(3)*T(5);
poly p2 = T(8) - T(1)*T(4) + T(3)*T(6);
poly p3 = T(9) - a*T(1)*T(5) + T(2)*T(6);
ideal spG = p1, p2, p3;
CEMDS cemds = createCEMDS(rvcvzP, scnSigma, spG);
intvec rvnVanishingDegrees = 0,0,0,0,0,0,1,1,1;
intvec rvfFakeVars = 0,0,0,0,0,0,1,1,1;
CEMDS cemdsResult = modifyCEMDS2(cemds, rvnVanishingDegrees, 0, rvfFakeVars);
print(cemdsResult);
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : 0
↦ //   number of vars : 10
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) T(6) T(7) T(8) T(\
   9) T(10)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      1     0    -1     1     0    -1     0     0     0     0
↦      0     1    -1     1    -1     0     0     0     0     0
↦      0    -1     0    -1     0     0     1     0     0     1
↦     -1     0     0    -1     0     0     0     1     0     1
↦     -1     0     0     0    -1     0     0     0     1     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Empty fan of ambient dimension 5
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ -T(2)*T(4)+(a)*T(3)*T(5)+T(7)*T(10),
↦ -T(1)*T(4)+T(3)*T(6)+T(8)*T(10),
↦ (-a)*T(1)*T(5)+T(2)*T(6)+T(9)*T(10),
↦ -T(1)*T(4)*T(7)+T(3)*T(6)*T(7)+T(2)*T(4)*T(8)+(-a)*T(3)*T(5)*T(8),
↦ T(6)*T(7)+(-a)*T(5)*T(8)+T(4)*T(9),
↦ T(1)*T(7)-T(2)*T(8)+T(3)*T(9)
//Should yield the CEMDS consisting of:
// - A matrix with the entries
//   ( 1  0 -1  1  0 -1 0 0 0 0)
```

```
//   ( 0  1 -1  1 -1  0 0 0 0 0)
//   ( 0 -1  0 -1  0  0 1 0 0 1)
//   (-1  0  0 -1  0  0 0 1 0 1)
//   (-1  0  0  0 -1  0 0 0 1 1)
// - The empty fan of ambient dimension 5
// - The ideal generated by
//   T(7)*T(10) - T(2)*T(4) + a*T(3)*T(5),
//   T(8)*T(10) - T(1)*T(4) + T(3)*T(6),
//   T(9)*T(10) - a*T(1)*T(5) + T(2)*T(6),
//   T(6)*T(7) - a*T(5)*T(8) + T(4)*T(9),
//   T(1)*T(7) - T(2)*T(8) + T(3)*T(9).
//Fourth example: Fake variables are marked and points are transferred to the modified CEMDS.
vector vnumPt = [1, 2, 1, 1, 1, 2, 1, -1, -3];
list svnumPts = vnumPt;
def RModified = modifyCEMDS2(cemds, rvnVanishingDegrees, 0, rvfFakeVars, svnumPts);
setring RModified;
list sModifyResult = gsResult;
print(sModifyResult);
↦ [1]:
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : 0
↦ //   number of vars : 10
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) T(6) T(7) T(8) T(\
   9) T(10)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      1     0    -1     1     0    -1     0     0     0     0
↦      0     1    -1     1    -1     0     0     0     0     0
↦      0    -1     0    -1     0     0     1     0     0     1
↦     -1     0     0    -1     0     0     0     1     0     1
↦     -1     0     0     0    -1     0     0     0     1     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Empty fan of ambient dimension 5
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ -T(2)*T(4)+(a)*T(3)*T(5)+T(7)*T(10),
↦ -T(1)*T(4)+T(3)*T(6)+T(8)*T(10),
↦ (-a)*T(1)*T(5)+T(2)*T(6)+T(9)*T(10),
↦ -T(1)*T(4)*T(7)+T(3)*T(6)*T(7)+T(2)*T(4)*T(8)+(-a)*T(3)*T(5)*T(8),
↦ T(6)*T(7)+(-a)*T(5)*T(8)+T(4)*T(9),
↦ T(1)*T(7)-T(2)*T(8)+T(3)*T(9)
↦
↦ [2]:
↦    [1]:
↦       gen(10)-3*gen(9)-gen(8)+gen(7)+2*gen(6)+gen(5)+gen(4)+gen(3)+2*gen(\
   2)+gen(1)
//Should yield a list with two entries:
// - the same CEMDS as in the third example
// - a list containing the transferred point [1, 2, 1, 1, 1, 2, 1, -1, -3, 1]
kill cemds;
kill rvcvzP;
```

```
kill scnSigma;
kill rvnVanishingDegrees;
kill cemdsResult;
kill sModifyResult;
kill RModified;
kill R;
//Fifth example: Verification of the resulting CEMDS, fake variables are marked, points are trans-
ferred to the modified CEMDS and its fan is computed.
ring R = 0,T(1..3),dp;
//The CEMDS's rays' matrix
intmat rvcvzP[2][3] =
1,0,-1,
0,1,-1;
//The CEMDS's fan
intmat cvrvz1[2][2] =
1,0,
0,1;
intmat cvrvz2[2][2] =
0,1,
-1,-1;
intmat cvrvz3[2][2] =
-1,-1,
1,0;
cone cn1 = coneViaPoints(cvrvz1);
cone cn2 = coneViaPoints(cvrvz2);
cone cn3 = coneViaPoints(cvrvz3);
fan scnSigma = fanViaCones(cn1, cn2, cn3);
//The CEMDS's embedding ideal
ideal spG = 0;
CEMDS cemds = createCEMDS(rvcvzP, scnSigma, spG);
intvec rvnVanishingDegrees = 0,1,1;
list sModifyResult = modifyCEMDS2(cemds, rvnVanishingDegrees, 1, 0:3, list(), 1);
↦ WARNING: The verification tests do not include the test whether the CEMDS\
   's ring is normal. Please check that manually.
print(sModifyResult);
↦ [1]:
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      1     0    -1    -1
↦      0     1    -1     0
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Dimension 2:
↦ 1st maximal cone:
↦ -1, -1,
↦  0, -1
↦
↦ 2nd maximal cone:
↦ -1, 0,
↦  0, 1
```

```
↦
↦ 3rd maximal cone:
↦ 1, -1,
↦ 0, -1
↦
↦ 4th maximal cone:
↦ 1, 0,
↦ 0, 1
↦
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ 0
↦
↦ [2]:
↦    1
//Should yield a list with three entries:
// - The CEMDS containing:
//   * The matrix with the entries
//     (1 0 -1 -1)
//     (0 1 -1  0)
//   * The fan consisting of the maximal cones generated by the following rays:
//     . (1,0), (0,1);
//     . (0,1), (-1,0);
//     . (-1,0, (-1,-1);
//     . (-1,-1), (1,0);
//   * The zero ideal
// - the verification result (1)
```

1.1.0.67 stretchCEMDS

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: stretchCEMDS(cemds, shpQPrimes); with cemds: CEMDS, shpQPrimes: list of poly; stretchCEMDS(cemds, shpQPrimes, svnumPts); with cemds: CEMDS, shpQPrimes: list of poly, svnumPts: list of vectors of numbers; stretchCEMDS(cemds, shpQPrimes, svnumPts, fComputeFan); with cemds: CEMDS, shpQPrimes: list of poly, svnumPts: list of vectors of numbers, fComputeFan: int; stretchCEMDS(cemds, shpQPrimes, svnumPts, fComputeFan, vzWeight); with cemds: CEMDS, shpQPrimes: list of poly, svnumPts: list of vectors of numbers, fComputeFan: int, vzWeight: intvec.

Assume: (i) The passed list of polynomials shpQPrimes is homogeneous w.r.t. the grading given by rvcvzQ = gale(cemds.rvcvzP); (ii) all points to stretch along with the CEMDS, if there were any passed, are contained in the CEMDS to stretch; (iii) the intvec `"vzWeight"`, if passed, is element of the ample cone defined by the grading matrix rvcvzQ extended by the passed polynomials' degrees and some true saturated connected collection of facets of the positive orthant.

Purpose: Embeds a CEMDS in a new ambient toric variety such that each passed polynomial is set in relation to a new variable. Optionally transfers points from the unstretched CEMDS to the stretched one.

Return: The stretched CEMDS only or, iff a non-empty list of points to stretch along with the CEMDS was passed, the basering the stretched CEMDS is defined in.

Side effects:
Iff a non-empty list of points to stretch along with the CEMDS were passed, a list gsResult containing the following (in order of appearance): (i) the stretched CEMDS with or without a fan (depending on fComputeFan and vzWeight) (always present), (ii) the list of stretched points (present as there was a non-empty list list passed).

Note: By default: fComputeFan = 0; vzWeight is an element of the relative interior of the moving cone generatwd by the columns of the grading matrix rvcvzQ = gale(cemds.rvcvzP) extended by the passed polynomials' degrees.

Example:

```
LIB "compcox.lib";
//First example: Parameterized basering with no optional arguments
ring R = (0,a),T(1..6),dp;
//The CEMDS's rays' matrix
intmat rvcvzP[2][6] = (
1,0,-1,1,0,-1,
0,1,-1,1,-1,0);
//The CEMDS's fan
intmat cvrvz1[2][2] =
1,0,
1,1;
intmat cvrvz2[2][2] =
1,1,
0,1;
intmat cvrvz3[2][2] =
0,1,
-1,0;
intmat cvrvz4[2][2] =
-1,0,
-1,-1;
intmat cvrvz5[2][2] =
-1,-1,
0,-1;
intmat cvrvz6[2][2] =
0,-1,
1,0;
cone cn1 = coneViaPoints(cvrvz1);
cone cn2 = coneViaPoints(cvrvz2);
cone cn3 = coneViaPoints(cvrvz3);
cone cn4 = coneViaPoints(cvrvz4);
cone cn5 = coneViaPoints(cvrvz5);
cone cn6 = coneViaPoints(cvrvz6);
fan scnSigma = fanViaCones(cn1, cn2, cn3, cn4, cn5, cn6);
//The CEMDS's embedding ideal
ideal spG = 0;
CEMDS cemds = createCEMDS(rvcvzP, scnSigma, spG);
//The orbit ideal generators of (1,a,1,1,1,1);
poly p1 = 1/a*T(2)*T(4) - T(3)*T(5);
poly p2 = T(1)*T(4) - T(3)*T(6);
poly p3 = T(1)*T(5) - 1/a*T(2)*T(6);
list shpQPrimes = p1,p2,p3;
CEMDS cemdsResult = stretchCEMDS(cemds, shpQPrimes);
//Obtains the CEMDS's ring
def RResult = cemdsResult.R;
setring RResult;
```

```
print(cemdsResult);
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : 0
↦ //   number of vars : 9
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) T(6) T(7) T(8) T(\
   9)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      1     0    -1     1     0    -1     0     0     0
↦      0     1    -1     1    -1     0     0     0     0
↦      0    -1     0    -1     0     0     1     0     0
↦     -1     0     0    -1     0     0     0     1     0
↦     -1    -1     1    -1     0     0     0     0     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Empty fan of ambient dimension 5
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ -T(2)*T(4)+(a)*T(3)*T(5)+T(7),
↦ -T(1)*T(4)+T(3)*T(6)+T(8),
↦ (-a)*T(1)*T(5)+T(2)*T(6)+T(9)
//Should yield
// - The rays' matrix
//   ( 1  0 -1  1  0 -1 0 0 0)
//   ( 0  1 -1  1 -1  0 0 0 0)
//   ( 0 -1  0 -1  0  0 1 0 0)
//   (-1  0  0 -1  0  0 0 1 0)
//   (-1  0  0  0 -1  0 0 0 1)
//   or some other matrix whose first two rows do not differ and whose further rows each lin-
early depend on the rows of this matrix
// - An empty fan of ambient dimension 5.
// - The ideal generated by
//   T(7) - T(2)*T(4) + a*T(3)*T(5),
//   T(8) - T(1)*T(4) + T(3)*T(6),
//   T(9) - a*T(1)*T(5) + T(2)*T(6).
//Second example with one optional argument: Points whose coordinates shall be stretched along with th
setring R;
vector vnumPt1 = [1, 1, 1, 1, 1, 1];
vector vnumPt2 = [0, 1, 2, 3, 4, 5];
list svnumPts = vnumPt1, vnumPt2;
def RResult2 = stretchCEMDS(cemds, shpQPrimes, svnumPts);
setring RResult2;
list sStretchResult2 = gsResult;
print(sStretchResult2);
↦ [1]:
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : 0
↦ //   number of vars : 9
↦ //        block   1 : ordering dp
```

```
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) T(6) T(7) T(8) T(\
  9)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      1     0    -1     1     0    -1     0     0     0
↦      0     1    -1     1    -1     0     0     0     0
↦      0    -1     0    -1     0     0     1     0     0
↦     -1     0     0    -1     0     0     0     1     0
↦     -1    -1     1    -1     0     0     0     0     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
  x of its rays:
↦ Empty fan of ambient dimension 5
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ -T(2)*T(4)+(a)*T(3)*T(5)+T(7),
↦ -T(1)*T(4)+T(3)*T(6)+T(8),
↦ (-a)*T(1)*T(5)+T(2)*T(6)+T(9)
↦
↦ [2]:
↦    [1]:
↦       (a-1)*gen(9)+(-a+1)*gen(7)+gen(6)+gen(5)+gen(4)+gen(3)+gen(2)+gen(1\
  )
↦    [2]:
↦       -5*gen(9)-10*gen(8)+(-8*a+3)*gen(7)+5*gen(6)+4*gen(5)+3*gen(4)+2*ge\
  n(3)+gen(2)
//Should yield a list with two entries:
// - The same CEMDS as in the first example.
// - A list of two vectors: [1,1,1,1,1,1,-a+1,0,a-1], [0,1,2,3,4,5,-8a+3,-10,-5].
//Third example with two optional arguments: Fan computation enabled.
setring R;
CEMDS cemdsResult3 = stretchCEMDS(cemds, list(), list(), 1);
print(cemdsResult3);
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : 0
↦ //   number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) T(6)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      1     0    -1     1     0    -1
↦      0     1    -1     1    -1     0
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
  x of its rays:
↦ Dimension 2:
↦ 1st maximal cone:
↦ -1, -1,
↦  0, -1
↦
↦ 2nd maximal cone:
↦ -1,  0,
↦ -1, -1
```

```
↦
↦ 3rd maximal cone:
↦ -1, 0,
↦  0, 1
↦
↦ 4th maximal cone:
↦ 1,  0,
↦ 0, -1
↦
↦ 5th maximal cone:
↦ 0, 1,
↦ 1, 1
↦
↦ 6th maximal cone:
↦ 1, 1,
↦ 0, 1
↦
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ 0
//Should yield the original CEMDS.
//Fourth example with three optional arguments: Points whose coordinates shall be stretched along with
putation enabled and starting with a user defined weight.
setring R;
intvec vzWeight = 3,-1,2,2;
def RResult4 = stretchCEMDS(cemds, shpQPrimes, svnumPts, 1, vzWeight);
setring RResult4;
list sStretchResult4 = gsResult;
print(sStretchResult4);
↦ [1]:
↦
↦ The CEMDS's ring:
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : 0
↦ //   number of vars : 9
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) T(6) T(7) T(8) T(\
   9)
↦ //        block   2 : ordering C
↦
↦ The column matrix P of the CEMDS's fan's rays:
↦      1     0    -1     1     0    -1     0     0     0
↦      0     1    -1     1    -1     0     0     0     0
↦      0    -1     0    -1     0     0     1     0     0
↦     -1     0     0    -1     0     0     0     1     0
↦     -1    -1     1    -1     0     0     0     0     1
↦
↦ The CEMDS's fan via its maximal cones, each one denoted by a column matri\
   x of its rays:
↦ Dimension 4:
↦ 1st maximal cone:
↦  0, -1,  0,  1,
↦ -1, -1,  1,  1,
↦  0,  0, -1, -1,
↦  0,  0,  0, -1,
↦  0,  1, -1, -1
↦
```

```
↦ 2nd maximal cone:
↦ -1,  1, -1,  1,
↦  0,  1, -1,  0,
↦  0, -1,  0,  0,
↦  0, -1,  0, -1,
↦  0, -1,  1, -1
↦
↦ 3rd maximal cone:
↦ -1,  0,  1,  0,
↦  0, -1,  0,  1,
↦  0,  0,  0, -1,
↦  0,  0, -1,  0,
↦  0,  0, -1, -1
↦
↦ Dimension 5:
↦ 1st maximal cone:
↦ -1,  0, 0, 0, -1,
↦  0, -1, 0, 0, -1,
↦  0,  0, 1, 0,  0,
↦  0,  0, 0, 1,  0,
↦  0,  0, 0, 0,  1
↦
↦ 2nd maximal cone:
↦ -1,  0,  0, 0, -1,
↦  0, -1,  1, 0, -1,
↦  0,  0, -1, 0,  0,
↦  0,  0,  0, 1,  0,
↦  0,  0, -1, 0,  1
↦
↦ 3rd maximal cone:
↦ -1, -1, 0, 0, 0,
↦  0, -1, 0, 0, 0,
↦  0,  0, 1, 0, 0,
↦  0,  0, 0, 1, 0,
↦  0,  1, 0, 0, 1
↦
↦ 4th maximal cone:
↦ -1, -1,  0, 0, 0,
↦  0, -1,  1, 0, 0,
↦  0,  0, -1, 0, 0,
↦  0,  0,  0, 1, 0,
↦  0,  1, -1, 0, 1
↦
↦ 5th maximal cone:
↦ -1,  0, 0,  1, -1,
↦  0, -1, 0,  0, -1,
↦  0,  0, 1,  0,  0,
↦  0,  0, 0, -1,  0,
↦  0,  0, 0, -1,  1
↦
↦ 6th maximal cone:
↦ -1,  0, 0, 0, 0,
↦ -1, -1, 0, 0, 0,
↦  0,  0, 1, 0, 0,
↦  0,  0, 0, 1, 0,
↦  1,  0, 0, 0, 1
↦
↦ 7th maximal cone:
```

```
↦ -1, -1,  0,  1, 0,
↦  0, -1,  1,  1, 0,
↦  0,  0, -1, -1, 0,
↦  0,  0,  0, -1, 0,
↦  0,  1, -1, -1, 1
↦
↦ 8th maximal cone:
↦ -1,  0, 0, 0, 0,
↦  0,  1, 0, 0, 0,
↦  0, -1, 1, 0, 0,
↦  0,  0, 0, 1, 0,
↦  0, -1, 0, 0, 1
↦
↦ 9th maximal cone:
↦ -1,  0, 0,  1, 0,
↦ -1, -1, 0,  0, 0,
↦  0,  0, 1,  0, 0,
↦  0,  0, 0, -1, 0,
↦  1,  0, 0, -1, 1
↦
↦ 10th maximal cone:
↦ -1, 0, 0,  1,  0,
↦  0, 0, 0,  1,  1,
↦  0, 0, 1, -1, -1,
↦  0, 0, 0, -1,  0,
↦  0, 1, 0, -1, -1
↦
↦ 11th maximal cone:
↦ -1,  0,  1,  1, 0,
↦ -1, -1,  1,  0, 0,
↦  0,  0, -1,  0, 0,
↦  0,  0, -1, -1, 0,
↦  1,  0, -1, -1, 1
↦
↦ 12th maximal cone:
↦  1,  0, 0, 0, 0,
↦  0, -1, 0, 0, 0,
↦  0,  0, 1, 0, 0,
↦ -1,  0, 0, 1, 0,
↦ -1,  0, 0, 0, 1
↦
↦ 13th maximal cone:
↦ -1,  1, 0,  0,  1,
↦  0,  0, 0,  1,  1,
↦  0,  0, 1, -1, -1,
↦  0, -1, 0,  0, -1,
↦  0, -1, 0, -1, -1
↦
↦ 14th maximal cone:
↦  0, 0, 0, 0,  1,
↦  1, 0, 0, 0,  1,
↦ -1, 0, 1, 0, -1,
↦  0, 0, 0, 1, -1,
↦ -1, 1, 0, 0, -1
↦
↦ 15th maximal cone:
↦ 0,  0,  1, 0,  1,
↦ 0, -1,  1, 0,  0,
```

```
↦ 0,  0, -1, 0,  0,
↦ 0,  0, -1, 1, -1,
↦ 1,  0, -1, 0, -1
↦
↦ 16th maximal cone:
↦  0,  0,  1, 0,  1,
↦  1, -1,  0, 0,  1,
↦ -1,  0,  0, 0, -1,
↦  0,  0, -1, 1, -1,
↦ -1,  0, -1, 0, -1
↦
↦ 17th maximal cone:
↦  1,  1, 0, 0, 0,
↦  0,  1, 0, 0, 0,
↦  0, -1, 1, 0, 0,
↦ -1, -1, 0, 1, 0,
↦ -1, -1, 0, 0, 1
↦
↦ 18th maximal cone:
↦  0,  1, 0, 0,  1,
↦  1,  0, 0, 0,  1,
↦ -1,  0, 1, 0, -1,
↦  0, -1, 0, 1, -1,
↦ -1, -1, 0, 0, -1
↦
↦
↦ The equations' ideal G embedding the MDS into its ambient toric variety:
↦ -T(2)*T(4)+(a)*T(3)*T(5)+T(7),
↦ -T(1)*T(4)+T(3)*T(6)+T(8),
↦ (-a)*T(1)*T(5)+T(2)*T(6)+T(9)
↦
↦ [2]:
↦    [1]:
↦       (a-1)*gen(9)+(-a+1)*gen(7)+gen(6)+gen(5)+gen(4)+gen(3)+gen(2)+gen(1\
   )
↦    [2]:
↦       -5*gen(9)-10*gen(8)+(-8*a+3)*gen(7)+5*gen(6)+4*gen(5)+3*gen(4)+2*ge\
   n(3)+gen(2)
//Should yield a list with two entries:
// - The same CEMDS as in the first example, but with a fan computed
// - A list of two vectors: [1,1,1,1,1,1,-a+1,0,a-1], [0,1,2,3,4,5,-8a+3,-10,-5].
```

1.1.0.68 verifyAlmostFree

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: verifyAlmostFree(rvcvzP); with rvcvzP: intmat.

Purpose: Checks whether the grading defined by a gale dual matrix to the passed matrix is an almost free grading.

Return: The int 1 if the grading defined by a gale dual matrix to the passed matrix is an almost free grading, the int 0 else.

Example:

```
LIB "compcox.lib";
intmat rvcvzP = intmat(intvec(
-1,0,1,1,0,-1,0,0,0,0,
1,-1,0,-1,1,0,0,0,0,0,
```

```
1,-1,0,0,0,0,1,-1,0,0,
-1,0,1,0,0,0,-1,0,1,0,
1,-1,-1,-1,0,0,1,0,0,1
), 5, 10);
int fVerified = verifyAlmostFree(rvcvzP);
print(fVerified);
↦ 1
//Should be true.
```

1.1.0.69 verifyDimension

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: verifyDimension(spG); with spG: ideal.

Purpose: Checks whether dim(spG) - dim(spG + <var(i)> + <var(j)>) >= 2 for all i != j.

Return: The int 1 if dim(spG) - dim(spG + <var(i)> + <var(j)>) >= 2 for all i != j, the int 0 else.

Example:

```
LIB "compcox.lib";
ring R = 0,T(1..10),dp;
ideal spG =
T(4)*T(7)-T(5)*T(8)+T(6)*T(9),
T(1)*T(7)-T(2)*T(8)+T(3)*T(9),
T(3)*T(5)-T(2)*T(6)-T(7)*T(10),
T(3)*T(4)-T(1)*T(6)-T(8)*T(10),
T(2)*T(4)-T(1)*T(5)-T(9)*T(10);
int fVerified = verifyDimension(spG);
print(fVerified);
↦ 1
//Should be true.
```

1.1.0.70 verifyVarPrimality

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

Usage: verifyVarPrimality(spG); with spG: ideal.

Purpose: Checks whether each ring variable's class in the basering modulo spG defines a prime ideal.

Return: The int 1 if each ring variable's class in the basering modulo spG defines a prime ideal, the int 0 else.

Example:

```
LIB "compcox.lib";
ring R = 0,T(1..10),dp;
ideal spG =
T(4)*T(7)-T(5)*T(8)+T(6)*T(9),
T(1)*T(7)-T(2)*T(8)+T(3)*T(9),
T(3)*T(5)-T(2)*T(6)-T(7)*T(10),
T(3)*T(4)-T(1)*T(6)-T(8)*T(10),
T(2)*T(4)-T(1)*T(5)-T(9)*T(10);
int fVerified = verifyVarPrimality(spG);
print(fVerified);
↦ 1
//Should be true.
```

1.1.0.71 vanishingDegrees

Procedure from library `compcox.lib` (see Section 1.1 [compcox_lib], page 1).

2 Index