Formatted manual of singularcurve.lib

1 Singular libraries

1.1 singularcurve_lib

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

Library: singularcurve.lib

Purpose: targets calculation of Zeta-function of a curve with quotient singularity over a finite field

Authors: Sebastian Mitterle, sebastianmitterle@mathume.com

Procedures:

1.1.0.1 sumintvec

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

Usage: sumintvec(e); e intvec

Assume: none

Return: int: the sum of the entries of e

Note: none

Example:

```
LIB "singularcurve.lib";
intvec e = 1,2,3;
int six = sumintvec(e);
six;
↦ 6
```

1.1.0.2 list2ideal

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

Usage: list2ideal(polylist); list polylist

Assume: none

Return: ideal: the ideal I = polylist[1],...,polylist[n];

Note: this procedure can be used to sum ideals avoiding simplification

Example:

```
LIB "singularcurve.lib";
ring r = 0, x, dp;
ideal I1 = x;
ideal I2 = 1;
list IL;
for(int i = 1; i<=size(I1); i++){
IL = insert(IL, I1[i]);
}
for(int i = 1; i<=size(I2); i++){
IL = insert(IL, I2[i]);
}
↦ // ** redefining i **
```

```
ideal I3 = list2ideal(IL);
I1 + I2;
↦ _[1]=1
I3;
↦ I3[1]=1
↦ I3[2]=x
```

1.1.0.3 multinomial

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

Usage: multinomial(N, iv): int N, intvec iv

Assume: none

Return: bigint multinomial coefficient N!/(iv[1]!***iv[n]!)

Note: needs general.lib; doesn't check N = sumintvec(iv)

Example:

```
LIB "singularcurve.lib";
LIB "general.lib";
intvec iv = 1,2,3;
bigint mulCoef = multinomial(6, iv);
mulCoef;
↦ 60
```

1.1.0.4 ceilquot

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

Usage: ceilquot(n1, n2); int n1, int n2

Assume: none

Return: ceil(decimal(n1/n2))

Note: none

Example:

```
LIB "singularcurve.lib";
ceilquot(1,2);
↦ 1
ceilquot(2,2);
↦ 1
ceilquot(-1,2);
↦ 0
```

1.1.0.5 padicOrder

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

Usage: padicOrder(n, p): number n, bigint p

Assume: basering is of characteristic 0

Return: returns the padic order of n: if n = p^k * a/b then returns order(a) - order(b); returns 'Inf' if n == 0

Note: needs poly.lib

Example:

```
LIB "singularcurve.lib";
ring r = 0, x, dp;
LIB "poly.lib";
number n1 = number(1/2);
number n2 = number(98/3);
number n3 = number(0);
padicOrder(n1, 7);
↦ 0
padicOrder(n2, 7);
↦ 2
padicOrder(n3, 7);
↦ Inf
```

1.1.0.6 padicApprox

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

Usage: padicApprox(quot, p, N): number(quot), int p, int N

Assume: basering is of characteristic 0

Return: returns the p-adic approximation of a rational number up to certain precision

Note: a is approximation of b to precision N <=> a - b = 0 mod p^N, i.e. a_i = 0, i>=N

Example:

```
LIB "singularcurve.lib";
LIB "poly.lib";
ring r = 0, x, dp;
number n1 = number(0);
number n2 = number(1/2);
number n3 = 7*3 + 49*4;
padicApprox(n1, 7, 0);
↦ 0
padicApprox(n2, 7, 7);
↦ 411772
padicApprox(n2, 7, 4);
↦ 1201
padicApprox(n3, 7, 3);
↦ 217
```

1.1.0.7 reduceStep

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

Usage: reduceStep(nom, jf, s): poly nom, ideal jf, int s

Assume: nom = 0 mod jf, basering of characteristic 0

Return: the polynomial f s.t. f/F^(s-1) Omega = nom/F^(s) Omega

Note: none

Example:

```
LIB "singularcurve.lib";
ring r = 0, (x,y,z), dp;
poly f = x^2 + y^2 + z^2;
```

```
ideal jf = jacob(ideal(f));
"2s-2:0->2::-2,0,2";
↦ 2s-2:0->2::-2,0,2
"2s-2:3::4::s==3";
↦ 2s-2:3::4::s==3
poly nom = x^2*y^2;
def reduced = reduceStep(nom, jf, 3);
reduced;
↦ 1/4x2
```

1.1.0.8 reduceAll

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

Usage: reduceAll(nom, jf, s): poly nom, ideal jf, int s

Assume: nom == 0 mod jf

Return: poly expression that expresses nom in terms of a base of R/jf

Note: s is the degree of the denominator F_k^s

Example:

```
LIB "singularcurve.lib";
ring r = 0, (x,y,z), dp;
poly f = x2z3 + y2z3 + x5 + y5 + 7z5;
ideal jf = jacob(f);
poly nom = x^2 + y^2;
nom = nom*(f^10);
int s = (deg(nom)+3)/5;
reduceAll(nom, jf, s);
↦ x2+y2
```

1.1.0.9 extractIdeal

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

Usage: extractIdeal(base, degrees): ideal base, intvec degrees

Assume: none

Return: the ideal containing only those generators of the input ideal of certain degrees

Note: none

Example:

```
LIB "singularcurve.lib";
LIB "general.lib";
ring r = 0, (x,y,z), dp;
poly f = x2z3 + y2z3 + x5 + y5 + 7z5;
ideal jf = jacob(f);
ideal base = kbase(std(jf));
size(base);
↦ 64
base = extractIdeal(base, intvec(2,7));
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
```

```
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
↦ // ** redefining j **
```

```
base;
↦ base[1]=z2
↦ base[2]=yz
↦ base[3]=xz
↦ base[4]=y2
↦ base[5]=xy
↦ base[6]=x2
↦ base[7]=z7
↦ base[8]=yz6
↦ base[9]=xz6
↦ base[10]=xyz5
↦ base[11]=xy3z3
↦ base[12]=x3y3z
```

1.1.0.10 getMatrix

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

Usage: getMatrix(base, image): ideal base, list image

Assume: the generators of base are base vectors of a module; image contains the image of each generator represented by a linear combination of the generators of base

Return: the matrix representing the map base[i] -> image[i]

Example:

```
LIB "singularcurve.lib";
LIB "general.lib";
ring r = 0, (x,y,z), dp;
ideal bf = x2,y2,z2, xy,xz,yz;
bf = sort(bf)[1];
matrix m1[6][6] = 1,2,3,4,5,6,7,8,9,10,11,12,1,2,3,4,5,6,7,8,9,10,11,12,1,2,3,4,5,6,7,8,9,10,11,12;█
list image = list();
for(int col = 6; col>=1; col--){
poly tmpoly = 0;
for(int row = 1; row<=6; row++){
tmpoly = tmpoly + m1[row, col]*bf[row];
}
image = insert(image, tmpoly);
}
↦ // ** redefining tmpoly **
↦ // ** redefining row **
↦ // ** redefining tmpoly **
↦ // ** redefining row **
↦ // ** redefining tmpoly **
↦ // ** redefining row **
↦ // ** redefining tmpoly **
↦ // ** redefining row **
↦ // ** redefining tmpoly **
↦ // ** redefining row **
def m2 = getMatrix(bf, image);
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining bindex **
↦ // ** redefining row **
↦ // ** redefining row **
```

```
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining bindex **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining bindex **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining bindex **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining bindex **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
↦ // ** redefining row **
m2;
↦ m2[1,1]=1
↦ m2[1,2]=2
↦ m2[1,3]=3
↦ m2[1,4]=4
↦ m2[1,5]=5
↦ m2[1,6]=6
↦ m2[2,1]=7
↦ m2[2,2]=8
↦ m2[2,3]=9
↦ m2[2,4]=10
↦ m2[2,5]=11
↦ m2[2,6]=12
↦ m2[3,1]=1
↦ m2[3,2]=2
↦ m2[3,3]=3
↦ m2[3,4]=4
↦ m2[3,5]=5
↦ m2[3,6]=6
↦ m2[4,1]=7
↦ m2[4,2]=8
↦ m2[4,3]=9
↦ m2[4,4]=10
↦ m2[4,5]=11
↦ m2[4,6]=12
↦ m2[5,1]=1
```

```
↦ m2[5,2]=2
↦ m2[5,3]=3
↦ m2[5,4]=4
↦ m2[5,5]=5
↦ m2[5,6]=6
↦ m2[6,1]=7
↦ m2[6,2]=8
↦ m2[6,3]=9
↦ m2[6,4]=10
↦ m2[6,5]=11
↦ m2[6,6]=12
```

2 Index