An introduction to Singular

Daniel Andres

Sevilla
October 02 & 04, 2012

Overview

http://www.singular.uni-kl.de

A computer algebra system for polynomial computations, with special emphasis on commutative and non-commutative algebra, algebraic geometry, and singularity theory.

From the SINGULAR homepage

SINGULAR is ...

- freely available online at `http://www.singular.uni-kl.de`
- available for Linux/Unix, Windows, Mac OS X
- open-source under the GNU General Public Licence
- is developed under the direction of Wolfram Decker, Gert-Martin Greuel, Gerhard Pfister, and Hans Schönemann (Department of Mathematics of the University of Kaiserslautern)

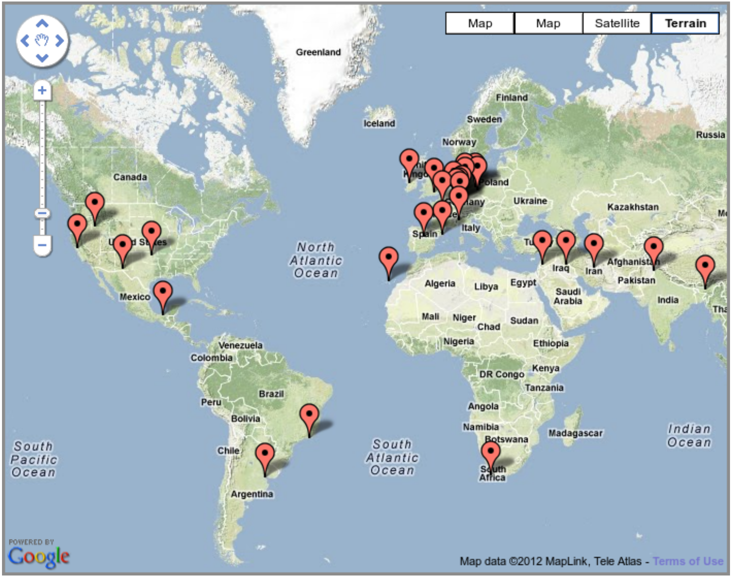
SINGULAR provides ...

- highly efficient core algorithms (Gröbner resp. standard bases, free resolutions, polynomial factorization, resultants, ...
- an intuitive, C-like programming language
- easy ways to make it user-extendible through libraries (as of today, more than 100 libraries are distributed together with SINGULAR)
- a comprehensive online manual and help function

From the SINGULAR homepage

The main objects SINGULAR can compute with are ideals, modules and matrices over

- polynomial rings over various ground fields and some rings (including the integers)
- localizations of the above
- a general class of non-commutative algebras (including the exterior algebra and the Weyl algebra)
- quotient rings of the above
- tensor products of the above


Map
Map
Satellite
Terrain
Greenland
Iceland
Finland
Sweden
Norway
Russia
Canada
Poland
Ukraine
Kazakhstan
Spain
Italy
North Atlantic Ocean
Mexico
Algeria
Libya
Egypt
Iraq
Iran
Afghanistan
Pakistan
Saudi Arabia
India
Mali
Niger
Chad
Sudan
Nigeria
Ethiopia
Venezuela
Colombia
Kenya
DR Congo
Tanzania
Brazil
Peru
Bolivia
Angola
Namibia
Madagascar
Indian Ocean
South Pacific Ocean
Chile
South Atlantic Ocean
Botswana
Argentina
POWERED BY
Google
Map data ©2012 MapLink, Tele Atlas - Terms of Use

Getting started


```
daniel@daniel-dot-a:~$ ▮
```

Getting started


```
daniel@daniel-dot-a:~$ Singular
```

Getting started


```
daniel@daniel-dot-a:~$ Singular
                     SINGULAR                                 /
 A Computer Algebra System for Polynomial Computations       /   version 3-1-5
                                                            0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann     \   Jul 2012
FB Mathematik der Universitaet, D-67653 Kaiserslautern        \
> ▮
```

```
daniel@daniel-dot-a:~$ Singular
                     SINGULAR                                 /
 A Computer Algebra System for Polynomial Computations       /   version 3-1-5
                                                           0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann     \   Jul 2012
FB Mathematik der Universitaet, D-67653 Kaiserslautern        \
> 1+1;
2
> █
```

Getting started

```
daniel@daniel-dot-a:~$ Singular
                     SINGULAR                                 /
 A Computer Algebra System for Polynomial Computations       /   version 3-1-5
                                                           0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann     \   Jul 2012
FB Mathematik der Universitaet, D-67653 Kaiserslautern        \
> 1+1;
2
> int k = 5;
> █
```

```
daniel@daniel-dot-a:~$ Singular
                     SINGULAR                                 /
 A Computer Algebra System for Polynomial Computations       /   version 3-1-5
                                                           0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann     \   Jul 2012
FB Mathematik der Universitaet, D-67653 Kaiserslautern        \
> 1+1;
2
> int k = 5;
> k;
5
> ▮
```

```
daniel@daniel-dot-a:~$ Singular
                     SINGULAR                                 /
 A Computer Algebra System for Polynomial Computations       /   version 3-1-5
                                                           0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann    \   Jul 2012
FB Mathematik der Universitaet, D-67653 Kaiserslautern       \
> 1+1;
2
> int k = 5;
> k;
5
> k div 2; k mod 2;
2
1
> ▮
```

Getting started


```
daniel@daniel-dot-a:~$ Singular
                     SINGULAR                                 /
 A Computer Algebra System for Polynomial Computations       /   version 3-1-5
                                                           0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann     \   Jul 2012
FB Mathematik der Universitaet, D-67653 Kaiserslautern        \
> 1+1;
2
> int k = 5;
> k;
5
> k div 2; k mod 2;
2
1
> intvec v = k,3,1;
> █
```

```
daniel@daniel-dot-a:~$ Singular
                     SINGULAR                                 /
 A Computer Algebra System for Polynomial Computations       /   version 3-1-5
                                                           0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann     \   Jul 2012
FB Mathematik der Universitaet, D-67653 Kaiserslautern        \
> 1+1;
2
> int k = 5;
> k;
5
> k div 2; k mod 2;
2
1
> intvec v = k,3,1;
> v;
5,3,1
> █
```

```
daniel@daniel-dot-a:~$ Singular
                     SINGULAR                                 /
 A Computer Algebra System for Polynomial Computations       /   version 3-1-5
                                                            0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann     \   Jul 2012
FB Mathematik der Universitaet, D-67653 Kaiserslautern        \
> 1+1;
2
> int k = 5;
> k;
5
> k div 2; k mod 2;
2
1
> intvec v = k,3,1;
> v;
5,3,1
> // This is a comment. Everything up to the end of the line is ignored.
. █
```

```
daniel@daniel-dot-a:~$ Singular
                     SINGULAR                              /
 A Computer Algebra System for Polynomial Computations    /   version 3-1-5
                                                        0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann   \   Jul 2012
FB Mathematik der Universitaet, D-67653 Kaiserslautern      \
> 1+1;
2
> int k = 5;
> k;
5
> k div 2; k mod 2;
2
1
> intvec v = k,3,1;
> v;
5,3,1
> // This is a comment. Everything up to the end of the line is ignored.
. v[2];   // indices start with 1 and not with 0
3
> ▮
```

Getting started

```
daniel@daniel-dot-a:~$ Singular
                     SINGULAR                                 /
 A Computer Algebra System for Polynomial Computations       /   version 3-1-5
                                                           0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann     \   Jul 2012
FB Mathematik der Universitaet, D-67653 Kaiserslautern        \
> 1+1;
2
> int k = 5;
> k;
5
> k div 2; k mod 2;
2
1
> intvec v = k,3,1;
> v;
5,3,1
> // This is a comment. Everything up to the end of the line is ignored.
. v[2];   // indices start with 1 and not with 0
3
> intmat m[2][3] = 1,2,3,4,5,6; m;
1,2,3,
4,5,6
> █
```

```
daniel@daniel-dot-a:~$ Singular
                     SINGULAR                                 /
 A Computer Algebra System for Polynomial Computations       /   version 3-1-5
                                                            0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann     \   Jul 2012
FB Mathematik der Universitaet, D-67653 Kaiserslautern        \
> 1+1;
2
> int k = 5;
> k;
5
> k div 2; k mod 2;
2
1
> intvec v = k,3,1;
> v;
5,3,1
> // This is a comment. Everything up to the end of the line is ignored.
. v[2];   // indices start with 1 and not with 0
3
> intmat m[2][3] = 1,2,3,4,5,6; m;
1,2,3,
4,5,6
> string s = "Hola!"; s;
Hola!
> 
```

Getting started


```
> ▮
```

Getting started


```
> list l = s,m,v;
> ▮
```

Getting started


```
> list l = s,m,v;
> l;
[1]:
   Hola!
[2]:
   1,2,3,
   4,5,6
[3]:
   5,3,1
> 
```

Getting started


```
> list l = s,m,v;
> l;
[1]:
   Hola!
[2]:
   1,2,3,
   4,5,6
[3]:
   5,3,1
> intmat m2[3][2] = v,7,8,9;
> ▮
```

```
> list l = s,m,v;
> l;
[1]:
   Hola!
[2]:
   1,2,3,
   4,5,6
[3]:
   5,3,1
> intmat m2[3][2] = v,7,8,9;
> intmat m3 = m*m2; m3;
31,44,
73,101
> █
```

```
> list l = s,m,v;
> l;
[1]:
   Hola!
[2]:
   1,2,3,
   4,5,6
[3]:
   5,3,1
> intmat m2[3][2] = v,7,8,9;
> intmat m3 = m*m2; m3;
31,44,
73,101
> proc myTrace (intmat m)
. {
.   if (ncols(m) != nrows(m)) { ERROR("matrix is not a square matrix"); }
.   int i,tr;
.   for (i=1; i<=ncols(m); i++)
.   {
.     tr = tr + m[i,i];
.   }
.   return(tr);
. }
> ▮
```

Getting started


```
> list l = s,m,v;
> l;
[1]:
   Hola!
[2]:
   1,2,3,
   4,5,6
[3]:
   5,3,1
> intmat m2[3][2] = v,7,8,9;
> intmat m3 = m*m2; m3;
31,44,
73,101
> proc myTrace (intmat m)
. {
.   if (ncols(m) != nrows(m)) { ERROR("matrix is not a square matrix"); }
.   int i,tr;
.   for (i=1; i<=ncols(m); i++)
.   {
.     tr = tr + m[i,i];
.   }
.   return(tr);
. }
> myTrace(m3);
132
```

Getting help

- Use the online manual

Getting help

- Use the online manual
- Use the builtin help system (also available as `html`-version)

```
> ▮
```

Getting help

- Use the online manual
- Use the builtin help system (also available as `html`-version)

```
> LIB "general.lib";
[...]
> ▮
```

Getting help

- Use the online manual
- Use the builtin help system (also available as `html`-version)

```
> LIB "general.lib";
[...]
> help fibonacci;
// proc fibonacci from lib general.lib
proc fibonacci (int n)
USAGE:    fibonacci(n);  n,p integers
RETURN:   fibonacci(n): nth Fibonacci number, f(0)=f(1)=1, f(i+1)=f(i-1)+f(i)
@*        - computed in characteristic 0, of type bigint
SEE ALSO: prime
EXAMPLE: example fibonacci; shows an example

> ▮
```

Getting help

- Use the online manual
- Use the builtin help system (also available as `html`-version)

```
> LIB "general.lib";
[...]
> help fibonacci;
// proc fibonacci from lib general.lib
proc fibonacci (int n)
USAGE:    fibonacci(n);  n,p integers
RETURN:   fibonacci(n): nth Fibonacci number, f(0)=f(1)=1, f(i+1)=f(i-1)+f(i)
@*        - computed in characteristic 0, of type bigint
SEE ALSO: prime
EXAMPLE: example fibonacci; shows an example

> example fibonacci;
// proc fibonacci from lib general.lib
EXAMPLE:
   fibonacci(42);
267914296

> ▮
```

Getting help

- Use the online manual
- Use the builtin help system (also available as `html`-version)

```
> LIB "general.lib";
[...]
> help fibonacci;
// proc fibonacci from lib general.lib
proc fibonacci (int n)
USAGE:    fibonacci(n);  n,p integers
RETURN:   fibonacci(n): nth Fibonacci number, f(0)=f(1)=1, f(i+1)=f(i-1)+f(i)
@*        - computed in characteristic 0, of type bigint
SEE ALSO: prime
EXAMPLE: example fibonacci; shows an example

> example fibonacci;
// proc fibonacci from lib general.lib
EXAMPLE:
   fibonacci(42);
267914296

> ▮
```

- Feel free to ask me: `daniel.andres@math.rwth-aachen.de`

Saving and opening your work

- Copy & paste:
  Write your code in your favourite text editor and copy and paste it to SINGULAR.
- Run ESINGULAR:
  SINGULAR from within `Emacs` / `XEmacs`.

The `write` and `read` commands

The `write` and `read` commands


```
> ▮
```

The write and read commands


```
> int a = 4; int b = 5;
> ▮
```

The `write` and `read` commands


```
> int a = 4; int b = 5;
> write("test.txt","Some examples for write and read");
> ▮
```

The write and read commands


```
> int a = 4; int b = 5;
> write("test.txt","Some examples for write and read");
> write("test.txt",a,b,a+b,a*b);
> 
```

The `write` and `read` commands


```
> int a = 4; int b = 5;
> write("test.txt","Some examples for write and read");
> write("test.txt",a,b,a+b,a*b);
> read("test.txt");
Some examples for write and read
4
5
9
20
> ▮
```

The write and read commands


```
> int a = 4; int b = 5;
> write("test.txt","Some examples for write and read");
> write("test.txt",a,b,a+b,a*b);
> read("test.txt");
Some examples for write and read
4
5
9
20
> write(":w test.txt",a*b);  // overwrite file
> ▮
```

The write and read commands

```
> int a = 4; int b = 5;
> write("test.txt","Some examples for write and read");
> write("test.txt",a,b,a+b,a*b);
> read("test.txt");
Some examples for write and read
4
5
9
20
> write(":w test.txt",a*b);  // overwrite file
> "int c = " + read("test.txt") + ";";
int c = 20;
> 
```

The write and read commands


```
> int a = 4; int b = 5;
> write("test.txt","Some examples for write and read");
> write("test.txt",a,b,a+b,a*b);
> read("test.txt");
Some examples for write and read
4
5
9
20
> write(":w test.txt",a*b);  // overwrite file
> "int c = " + read("test.txt") + ";";
int c = 20;
> c;
   ? `c` is undefined
   ? error occurred in or before STDIN line 7: `c;`
> ▮
```

The write and read commands

```
> int a = 4; int b = 5;
> write("test.txt","Some examples for write and read");
> write("test.txt",a,b,a+b,a*b);
> read("test.txt");
Some examples for write and read
4
5
9
20
> write(":w test.txt",a*b);  // overwrite file
> "int c = " + read("test.txt") + ";";
int c = 20;
> c;
   ? ‘c‘ is undefined
   ? error occurred in or before STDIN line 7: ‘c;‘
> execute("int c = " + read("test.txt") + ";"); c;
20
> ▮
```

```
> int a = 4; int b = 5;
> write("test.txt","Some examples for write and read");
> write("test.txt",a,b,a+b,a*b);
> read("test.txt");
Some examples for write and read
4
5
9
20
> write(":w test.txt",a*b);  // overwrite file
> "int c = " + read("test.txt") + ";";
int c = 20;
> c;
   ? `c` is undefined
   ? error occurred in or before STDIN line 7: `c;`
> execute("int c = " + read("test.txt") + ";"); c;
20
> write(":w test.txt", "int d = " + string(a+b) + ";");
> 
```

The write and read commands


```
> int a = 4; int b = 5;
> write("test.txt","Some examples for write and read");
> write("test.txt",a,b,a+b,a*b);
> read("test.txt");
Some examples for write and read
4
5
9
20
> write(":w test.txt",a*b);  // overwrite file
> "int c = " + read("test.txt") + ";";
int c = 20;
> c;
   ? `c` is undefined
   ? error occurred in or before STDIN line 7: `c;`
> execute("int c = " + read("test.txt") + ";"); c;
20
> write(":w test.txt", "int d = " + string(a+b) + ";");
> < "test.txt";  // short form of  execute(read("test.txt"));
> ▮
```

```
> int a = 4; int b = 5;
> write("test.txt","Some examples for write and read");
> write("test.txt",a,b,a+b,a*b);
> read("test.txt");
Some examples for write and read
4
5
9
20
> write(":w test.txt",a*b);  // overwrite file
> "int c = " + read("test.txt") + ";";
int c = 20;
> c;
   ? `c` is undefined
   ? error occurred in or before STDIN line 7: `c;`
> execute("int c = " + read("test.txt") + ";"); c;
20
> write(":w test.txt", "int d = " + string(a+b) + ";");
> < "test.txt";  // short form of  execute(read("test.txt"));
> d;
9
> ▮
```

Opening your work revisited

Suppose you have a plain text file `test.txt` with the following content.

```
int a,b = 4,5;
int c = a^2 + b^2 + 1;
c;
exit;
```

Opening your work revisited

Suppose you have a plain text file `test.txt` with the following content.

```
int a,b = 4,5;
int c = a^2 + b^2 + 1;
c;
exit;
```

The terminal command

```
daniel@daniel-dot-a:~$ Singular < test.txt > test.txt.res
```

Opening your work revisited

Suppose you have a plain text file `test.txt` with the following content.

```
int a,b = 4,5;
int c = a^2 + b^2 + 1;
c;
exit;
```

The terminal command

```
daniel@daniel-dot-a:~$ Singular < test.txt > test.txt.res
```

then produces a plain text file `test.txt.res` with the following content.

```
                     SINGULAR                                 /
 A Computer Algebra System for Polynomial Computations       /   version 3-1-5
                                                           0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann     \   Jul 2012
FB Mathematik der Universitaet, D-67653 Kaiserslautern        \
42
Auf Wiedersehen.
```

Opening your work revisited

Suppose you have a plain text file `test.txt` with the following content.

```
int a,b = 4,5;
int c = a^2 + b^2 + 1;
c;
exit;
```

The terminal command

```
daniel@daniel-dot-a:~$ Singular -q < test.txt > test.txt.res
```

Opening your work revisited

Suppose you have a plain text file `test.txt` with the following content.

```
int a,b = 4,5;
int c = a^2 + b^2 + 1;
c;
exit;
```

The terminal command

```
daniel@daniel-dot-a:~$ Singular -q < test.txt > test.txt.res
```

then produces a plain text file `test.txt.res` with the following content.

```
42
```

Opening your work revisited

Suppose you have a plain text file `test.txt` with the following content.

```
int a,b = 4,5;
int c = a^2 + b^2 + 1;
c;
exit;
```

The terminal command

```
daniel@daniel-dot-a:~$ Singular -q < test.txt > test.txt.res
```

then produces a plain text file `test.txt.res` with the following content.

```
42
```

$\rightsquigarrow$ Prepare multiple computations in individual files and use a batch script to compute them all!

Rings

So far we have seen the data types:
`int`, `intvec`, `intmat`, `string`, `list`(, `proc`), `bigint`.

So far we have seen the data types:
`int`, `intvec`, `intmat`, `string`, `list`(, `proc`), `bigint`.
But there are more:
`poly`, `vector`, `matrix`, `ideal`, `module`, `map`, `number`, `list`, ...

Rings

So far we have seen the data types (ring independent):
`int`, `intvec`, `intmat`, `string`, `list`(, `proc`), `bigint`.
But there are more (ring dependent):
`poly`, `vector`, `matrix`, `ideal`, `module`, `map`, `number`, `list`, ...

So far we have seen the data types (ring independent):
`int`, `intvec`, `intmat`, `string`, `list`(, `proc`), `bigint`.
But there are more (ring dependent):
`poly`, `vector`, `matrix`, `ideal`, `module`, `map`, `number`, `list`, ...

So what's a `ring`?
A `ring` is a triple consisting of a coefficient ring, names of variables and a monomial ordering.

Possible coefficient rings:

- $\mathbb{Q}$,
- $\mathbb{Z}/p$ for a prime $p \leq 2147483647$,
- transcendental or simple algebraic extensions of the above,
- $\mathrm{GF}(p^n)$ for a prime p with $p^n \leq 2^{16}$,
- $\mathbb{R}$ or $\mathbb{C}$ (floating point numbers of user defined precision),
- $\mathbb{Z}$,
- $\mathbb{Z}/n$ for $n \in \mathbb{Z}$.

Possible coefficient rings:

- $\mathbb{Q}$,
- $\mathbb{Z}/p$ for a prime $p \leq 2147483647$,
- transcendental or simple algebraic extensions of the above,
- $\mathrm{GF}(p^n)$ for a prime p with $p^n \leq 2^{16}$,
- $\mathbb{R}$ or $\mathbb{C}$ (floating point numbers of user defined precision),
- $\mathbb{Z}$,
- $\mathbb{Z}/n$ for $n \in \mathbb{Z}$.

In case of coefficient rings, which are not fields, only the following functions are guaranteed to work:

- basic polynomial arithmetic, i.e. addition, multiplication, division, exponentation
- `std`, i.e. computing standard bases
- `interred`
- `reduce` a.k.a. `NF`

Possible coefficient rings:

- $\mathbb{Q}$,
- $\mathbb{Z}/p$ for a prime $p \leq 2147483647$,
- transcendental or simple algebraic extensions of the above,
- $\mathrm{GF}(p^n)$ for a prime p with $p^n \leq 2^{16}$,
- $\mathbb{R}$ or $\mathbb{C}$ (floating point numbers of user defined precision),
- $\mathbb{Z}$,
- $\mathbb{Z}/n$ for $n \in \mathbb{Z}$.

In case of coefficient rings, which are not fields, only the following functions are guaranteed to work:

- basic polynomial arithmetic, i.e. addition, multiplication, division, exponentation
- `std`, i.e. computing standard bases
- `interred`
- `reduce` a.k.a. `NF`

Let's see some examples of ring declarations.

`lp` (left) lexicographical ordering

`dp` degree reverse lexicographical ordering

`Dp` degree lexicographical ordering

`wp( <intvec> )` weighted reverse lexicographical ordering; the weight vector must consist of positive integers only.

`Wp( <intvec> )` weighted lexicographical ordering; the weight vector must consist of positive integers only.

Rings: Monomial orderings

`lp` (left) lexicographical ordering

`dp` degree reverse lexicographical ordering

`Dp` degree lexicographical ordering

`wp( <intvec> )` weighted reverse lexicographical ordering; the weight vector must consist of positive integers only.

`Wp( <intvec> )` weighted lexicographical ordering; the weight vector must consist of positive integers only.

`ls` negative lexicographical ordering

`ds` negative degree reverse lexicographical ordering

`Ds` negative degree lexicographical ordering

`ws( <intvec> )` (general) weighted reverse lexicographical ordering; the first element of the weight vector has to be non-zero.

`Ws( <intvec> )` (general) weighted lexicographical ordering; the first element of the weight vector has to be non-zero.

Rings: Monomial orderings

Let $\prec_1$ be a monomial ordering on $\mathrm{Mon}(x_1, \ldots, x_n)$ and let $\prec_2$ be a monomial ordering on $\mathrm{Mon}(y_1, \ldots, y_m)$.
The *product* (or *block*) *ordering* $\prec = (\prec_1, \prec_2)$ on $\mathrm{Mon}(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is defined by

$$x^\alpha y^\beta \prec x^{\alpha'} y^{\beta'} \quad :\Leftrightarrow \quad x^\alpha \prec_1 x^{\alpha'} \text{ or } (x^\alpha = x^{\alpha'} \text{ and } y^\beta \prec_2 y^{\beta'}).$$

```
> ▮
```

Let $\prec_1$ be a monomial ordering on $\mathrm{Mon}(x_1,\ldots,x_n)$ and let $\prec_2$ be a monomial ordering on $\mathrm{Mon}(y_1,\ldots,y_m)$.
The *product* (or *block*) *ordering* $\prec = (\prec_1, \prec_2)$ on $\mathrm{Mon}(x_1,\ldots,x_n,y_1,\ldots,y_m)$ is defined by

$$x^\alpha y^\beta \prec x^{\alpha'} y^{\beta'} \quad :\Leftrightarrow \quad x^\alpha \prec_1 x^{\alpha'} \text{ or } (x^\alpha = x^{\alpha'} \text{ and } y^\beta \prec_2 y^{\beta'}).$$

```
> int n,m = 2,3;
> ▮
```

Rings: Monomial orderings

Let $\prec_1$ be a monomial ordering on $\mathrm{Mon}(x_1, \ldots, x_n)$ and let $\prec_2$ be a monomial ordering on $\mathrm{Mon}(y_1, \ldots, y_m)$.
The *product* (or *block*) *ordering* $\prec = (\prec_1, \prec_2)$ on $\mathrm{Mon}(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is defined by

$$x^\alpha y^\beta \prec x^{\alpha'} y^{\beta'} \quad :\Leftrightarrow \quad x^\alpha \prec_1 x^{\alpha'} \text{ or } (x^\alpha = x^{\alpha'} \text{ and } y^\beta \prec_2 y^{\beta'}).$$

```
> int n,m = 2,3;
> ring r = 0,(x(1..n),y(1..m)),(lp(n),dp(m));
> ▮
```

Rings: Monomial orderings

Let $\prec_1$ be a monomial ordering on $\mathrm{Mon}(x_1, \ldots, x_n)$ and let $\prec_2$ be a monomial ordering on $\mathrm{Mon}(y_1, \ldots, y_m)$.
The *product* (or *block*) *ordering* $\prec = (\prec_1, \prec_2)$ on $\mathrm{Mon}(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is defined by

$$x^\alpha y^\beta \prec x^{\alpha'} y^{\beta'} \quad :\Leftrightarrow \quad x^\alpha \prec_1 x^{\alpha'} \text{ or } (x^\alpha = x^{\alpha'} \text{ and } y^\beta \prec_2 y^{\beta'}).$$

```
> int n,m = 2,3;
> ring r = 0,(x(1..n),y(1..m)),(lp(n),dp(m));
> r;
//   characteristic : 0
//   number of vars : 5
//        block   1 : ordering lp
//                  : names    x(1) x(2)
//        block   2 : ordering dp
//                  : names    y(1) y(2) y(3)
//        block   3 : ordering C
> ▮
```

Rings: Monomial orderings

Let $\prec_1$ be a monomial ordering on $\mathrm{Mon}(x_1, \ldots, x_n)$ and let $\prec_2$ be a monomial ordering on $\mathrm{Mon}(y_1, \ldots, y_m)$.
The *product* (or *block*) *ordering* $\prec = (\prec_1, \prec_2)$ on $\mathrm{Mon}(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is defined by

$$x^\alpha y^\beta \prec x^{\alpha'} y^{\beta'} \quad :\Leftrightarrow \quad x^\alpha \prec_1 x^{\alpha'} \text{ or } (x^\alpha = x^{\alpha'} \text{ and } y^\beta \prec_2 y^{\beta'}).$$

```
> int n,m = 2,3;
> ring r = 0,(x(1..n),y(1..m)),(lp(n),dp(m));
> r;
//   characteristic : 0
//   number of vars : 5
//        block   1 : ordering lp
//                  : names    x(1) x(2)
//        block   2 : ordering dp
//                  : names    y(1) y(2) y(3)
//        block   3 : ordering C
> poly f = x(2)^2*x(1) + 1 + y(1)^10 + y(2)^5*x(1) + y(3);
> ▮
```

Rings: Monomial orderings

Let $\prec_1$ be a monomial ordering on $\mathrm{Mon}(x_1, \ldots, x_n)$ and let $\prec_2$ be a monomial ordering on $\mathrm{Mon}(y_1, \ldots, y_m)$.
The *product* (or *block*) *ordering* $\prec = (\prec_1, \prec_2)$ on $\mathrm{Mon}(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is defined by

$$x^\alpha y^\beta \prec x^{\alpha'} y^{\beta'} \quad :\Leftrightarrow \quad x^\alpha \prec_1 x^{\alpha'} \text{ or } (x^\alpha = x^{\alpha'} \text{ and } y^\beta \prec_2 y^{\beta'}).$$

```
> int n,m = 2,3;
> ring r = 0,(x(1..n),y(1..m)),(lp(n),dp(m));
> r;
//   characteristic : 0
//   number of vars : 5
//        block   1 : ordering lp
//                  : names    x(1) x(2)
//        block   2 : ordering dp
//                  : names    y(1) y(2) y(3)
//        block   3 : ordering C
> poly f = x(2)^2*x(1) + 1 + y(1)^10 + y(2)^5*x(1) + y(3);
> f;
x(1)*x(2)^2+x(1)*y(2)^5+y(1)^10+y(3)+1
> ▮
```

For simplicity, let $\mathbb{K}$ be a field and let $\prec$ be a monomial ordering on $\mathbb{K}[x_1, \ldots, x_n]$.
Denote by $\mathrm{Loc}_{\prec}(\mathbb{K}[x_1, \ldots, x_n])$ the localization of $\mathbb{K}[x_1, \ldots, x_n]$ with respect to the multiplicatively closed set

$$\{1 + g \mid g = 0 \text{ or } g \in \mathbb{K}[x_1, \ldots, x_n] \setminus \{0\} \text{ and } \mathrm{lm}(g) < 1\}.$$

For simplicity, let $\mathbb{K}$ be a field and let $\prec$ be a monomial ordering on $\mathbb{K}[x_1, \ldots, x_n]$.
Denote by $\mathrm{Loc}_{\prec}(\mathbb{K}[x_1, \ldots, x_n])$ the localization of $\mathbb{K}[x_1, \ldots, x_n]$ with respect to the multiplicatively closed set

$$\{1 + g \mid g = 0 \text{ or } g \in \mathbb{K}[x_1, \ldots, x_n] \setminus \{0\} \text{ and } \operatorname{lm}(g) < 1\}.$$

Example

- For any global ordering $\prec$ (i.e. $1 \prec m$ for any monomial $m \neq 1$), $\mathrm{Loc}_{\prec}(\mathbb{K}[x_1, \ldots, x_n]) = \mathbb{K}[x_1, \ldots, x_n]$.

Rings associated to orderings

For simplicity, let $\mathbb{K}$ be a field and let $\prec$ be a monomial ordering on $\mathbb{K}[x_1, \ldots, x_n]$.
Denote by $\mathrm{Loc}_\prec(\mathbb{K}[x_1, \ldots, x_n])$ the localization of $\mathbb{K}[x_1, \ldots, x_n]$ with respect to the multiplicatively closed set

$$\{1 + g \mid g = 0 \text{ or } g \in \mathbb{K}[x_1, \ldots, x_n] \setminus \{0\} \text{ and } \operatorname{lm}(g) < 1\}.$$

Example

- For any global ordering $\prec$ (i.e. $1 \prec m$ for any monomial $m \neq 1$), $\mathrm{Loc}_\prec(\mathbb{K}[x_1, \ldots, x_n]) = \mathbb{K}[x_1, \ldots, x_n]$.
- For any local ordering $\prec$ (i.e. $m \prec 1$ for any monomial $m \neq 1$), $\mathrm{Loc}_\prec(\mathbb{K}[x_1, \ldots, x_n]) = \mathbb{K}[x_1, \ldots, x_n]_{\langle x_1, \ldots, x_n \rangle}$.

For simplicity, let $\mathbb{K}$ be a field and let $\prec$ be a monomial ordering on $\mathbb{K}[x_1, \ldots, x_n]$.
Denote by $\mathrm{Loc}_{\prec}(\mathbb{K}[x_1, \ldots, x_n])$ the localization of $\mathbb{K}[x_1, \ldots, x_n]$ with respect to the multiplicatively closed set

$$\{1 + g \mid g = 0 \text{ or } g \in \mathbb{K}[x_1, \ldots, x_n] \setminus \{0\} \text{ and } \mathrm{lm}(g) < 1\}.$$

Example

- For any global ordering $\prec$ (i.e. $1 \prec m$ for any monomial $m \neq 1$), $\mathrm{Loc}_{\prec}(\mathbb{K}[x_1, \ldots, x_n]) = \mathbb{K}[x_1, \ldots, x_n]$.
- For any local ordering $\prec$ (i.e. $m \prec 1$ for any monomial $m \neq 1$), $\mathrm{Loc}_{\prec}(\mathbb{K}[x_1, \ldots, x_n]) = \mathbb{K}[x_1, \ldots, x_n]_{\langle x_1, \ldots, x_n \rangle}$.
- For any global ordering $\prec_1$ on $\mathrm{Mon}(x_1, \ldots, x_n)$ and any local ordering $\prec_2$ on $\mathrm{Mon}(y_1, \ldots, y_m)$, $\mathrm{Loc}_{(\prec_1, \prec_2)}(\mathbb{K}[x_1, \ldots, x_n, y_1, \ldots, y_m]) = (\mathbb{K}[y_1, \ldots, y_m]_{\langle y_1, \ldots, y_m \rangle})[x_1, \ldots, x_n]$.

Let $\mathbb{K}$ be a field. The $\mathbb{K}$-algebra

$$A := \mathbb{K}\langle x_1, \ldots, x_n \mid x_j x_i = c_{ij} \cdot x_i x_j + d_{ij},\ 1 \leq i < j \leq n \rangle,$$

where $c_{ij} \in \mathbb{K}^*$ and d_{ij} is a polynomial involving only standard monomials, is called a *G-algebra* if the following two conditions hold.

Let $\mathbb{K}$ be a field. The $\mathbb{K}$-algebra

$$A := \mathbb{K}\langle x_1, \ldots, x_n \mid x_j x_i = c_{ij} \cdot x_i x_j + d_{ij},\ 1 \leq i < j \leq n \rangle,$$

where $c_{ij} \in \mathbb{K}^*$ and d_{ij} is a polynomial involving only standard monomials, is called a *G-algebra* if the following two conditions hold.

- *Ordering condition.* There exists a **global** ordering $\prec$ on A such that $\mathrm{lm}(d_{ij}) \prec x_i x_j$ for $i < j$.
- *Non-degeneracy condition.* For $f, g \in A$ and $a, b \in \mathbb{K}^*$, define $[f,g]_b^a := afg - bgf$ and put $l_{ij} := \mathrm{lc}(x_i x_j)$ such that $c_{ij} = l_{ji}/l_{ij}$. For $1 \leq i < j < k \leq n$ the expression

$$\Big[[x_i, x_j]_{l_{ij}}^{l_{ji}}, x_k \Big]_{l_{ik} l_{jk}}^{l_{ki} l_{kj}} + \Big[[x_j, x_k]_{l_{jk}}^{l_{kj}}, x_i \Big]_{l_{ji} l_{ki}}^{l_{ij} l_{ik}} + \Big[[x_k, x_i]_{l_{ki}}^{l_{ik}}, x_j \Big]_{l_{kj} l_{ij}}^{l_{jk} l_{ji}}$$

  equals zero.

E.g., $U(\mathfrak{sl}_2) = \mathbb{K}\langle e,\ f,\ h \mid fe = ef - h, he = eh + 2e, hf = fh - 2f \rangle$.

```
> ▮
```

E.g., $U(\mathfrak{sl}_2) = \mathbb{K}\langle e,\ f,\ h \mid fe = ef - h, he = eh + 2e, hf = fh - 2f\rangle$.

```
> ring r = 0,(e,f,h),lp;
> ▮
```

Non-commutative rings

E.g., $U(\mathfrak{sl}_2) = \mathbb{K}\langle e,\ f,\ h \mid fe = ef - h, he = eh + 2e, hf = fh - 2f\rangle$.

```
> ring r = 0,(e,f,h),lp;
> matrix C[3][3];
> ▮
```

Non-commutative rings

E.g., $U(\mathfrak{sl}_2) = \mathbb{K}\langle e,\ f,\ h \mid fe = ef - h, he = eh + 2e, hf = fh - 2f\rangle$.

```
> ring r = 0,(e,f,h),lp;
> matrix C[3][3];
> C[1,2] = 1;  C[1,3] = 1;  C[2,3] = 1;
> ▮
```

E.g., $U(\mathfrak{sl}_2) = \mathbb{K}\langle e,\ f,\ h \mid fe = ef - h, he = eh + 2e, hf = fh - 2f\rangle$.

```
> ring r = 0,(e,f,h),lp;
> matrix C[3][3];
> C[1,2] = 1;  C[1,3] = 1;  C[2,3] = 1;
> matrix D[3][3];
> ▮
```

Non-commutative rings

E.g., $U(\mathfrak{sl}_2) = \mathbb{K}\langle e,\ f,\ h \mid fe = ef - h, he = eh + 2e, hf = fh - 2f\rangle$.

```
> ring r = 0,(e,f,h),lp;
> matrix C[3][3];
> C[1,2] = 1;  C[1,3] = 1;  C[2,3] = 1;
> matrix D[3][3];
> D[1,2] = -h; D[1,3] = 2e; D[2,3] = -2f;
> ▮
```

E.g., $U(\mathfrak{sl}_2) = \mathbb{K}\langle e,\ f,\ h \mid fe = ef - h, he = eh + 2e, hf = fh - 2f\rangle$.

```
> ring r = 0,(e,f,h),lp;
> matrix C[3][3];
> C[1,2] = 1;  C[1,3] = 1;  C[2,3] = 1;
> matrix D[3][3];
> D[1,2] = -h; D[1,3] = 2e; D[2,3] = -2f;
> def Usl2 = nc_algebra(C,D);
> ▮
```

E.g., $U(\mathfrak{sl}_2) = \mathbb{K}\langle e,\ f,\ h \mid fe = ef - h, he = eh + 2e, hf = fh - 2f\rangle$.

```
> ring r = 0,(e,f,h),lp;
> matrix C[3][3];
> C[1,2] = 1;  C[1,3] = 1;  C[2,3] = 1;
> matrix D[3][3];
> D[1,2] = -h; D[1,3] = 2e; D[2,3] = -2f;
> def Usl2 = nc_algebra(C,D);
> setring Usl2;
> ▮
```

E.g., $U(\mathfrak{sl}_2) = \mathbb{K}\langle e,\ f,\ h \mid fe = ef - h, he = eh + 2e, hf = fh - 2f\rangle$.

```
> ring r = 0,(e,f,h),lp;
> matrix C[3][3];
> C[1,2] = 1;  C[1,3] = 1;  C[2,3] = 1;
> matrix D[3][3];
> D[1,2] = -h; D[1,3] = 2e; D[2,3] = -2f;
> def Usl2 = nc_algebra(C,D);
> setring Usl2;
> Usl2;
//   characteristic : 0
//   number of vars : 3
//        block   1 : ordering lp
//                  : names    e f h
//        block   2 : ordering C
//   noncommutative relations:
//    fe=ef-h
//    he=eh+2e
//    hf=fh-2f
> ▮
```

E.g., $U(\mathfrak{sl}_2) = \mathbb{K}\langle e,\ f,\ h \mid fe = ef - h, he = eh + 2e, hf = fh - 2f\rangle$.

```
> ring r = 0,(e,f,h),lp;
> matrix C[3][3];
> C[1,2] = 1;  C[1,3] = 1;  C[2,3] = 1;
> matrix D[3][3];
> D[1,2] = -h; D[1,3] = 2e; D[2,3] = -2f;
> def Usl2 = nc_algebra(C,D);
> setring Usl2;
> Usl2;
//   characteristic : 0
//   number of vars : 3
//        block   1 : ordering lp
//                  : names    e f h
//        block   2 : ordering C
//   noncommutative relations:
//    fe=ef-h
//    he=eh+2e
//    hf=fh-2f
> h*f*e;
efh-h2
> ▮
```

Non-commutative rings

E.g., $U(\mathfrak{sl}_2) = \mathbb{K}\langle e,\ f,\ h \mid fe = ef - h, he = eh + 2e, hf = fh - 2f\rangle$.

```
> ring r = 0,(e,f,h),lp;
> matrix C[3][3];
> C[1,2] = 1;  C[1,3] = 1;  C[2,3] = 1;
> matrix D[3][3];
> D[1,2] = -h; D[1,3] = 2e; D[2,3] = -2f;
> def Usl2 = nc_algebra(C,D);
> setring Usl2;
> Usl2;
//   characteristic : 0
//   number of vars : 3
//        block   1 : ordering lp
//                  : names    e f h
//        block   2 : ordering C
//   noncommutative relations:
//    fe=ef-h
//    he=eh+2e
//    hf=fh-2f
> h*f*e;
efh-h2
> █
```

Have a look at `ncalg.lib` for some predefined G-algebras!

A map $f : A \to B$ is always defined from within B!

```
> ▮
```

A map $f : A \to B$ is always defined from within B!

```
> ring A = 0,(a,b,c),dp;
> ▮
```

A map $f : A \to B$ is always defined from within B!

```
> ring A = 0,(a,b,c),dp;
> poly p = a + b + a*b + c^3;
> █
```

A map $f : A \to B$ is always defined from within B!

```
> ring A = 0,(a,b,c),dp;
> poly p = a + b + a*b + c^3;
> p;
c3+ab+a+b
> ▮
```

Maps

A map $f : A \to B$ is always defined from within B!

```
> ring A = 0,(a,b,c),dp;
> poly p = a + b + a*b + c^3;
> p;
c3+ab+a+b
> ring B = 0,(x,y,z),dp;
> ▮
```

Maps

A map $f : A \to B$ is always defined from within B!

```
> ring A = 0,(a,b,c),dp;
> poly p = a + b + a*b + c^3;
> p;
c3+ab+a+b
> ring B = 0,(x,y,z),dp;
> p;
   ? `p` is undefined
> ▮
```

Maps

A map $f : A \to B$ is always defined from within B!

```
> ring A = 0,(a,b,c),dp;
> poly p = a + b + a*b + c^3;
> p;
c3+ab+a+b
> ring B = 0,(x,y,z),dp;
> p;
   ? `p` is undefined
> map f = A, x+y,x-y,z; // homomorphism f: A -> B with f(a)=x+y, f(b)=x-y, f(c)=z
> ▮
```

Maps

A map $f : A \to B$ is always defined from within B!

```
> ring A = 0,(a,b,c),dp;
> poly p = a + b + a*b + c^3;
> p;
c3+ab+a+b
> ring B = 0,(x,y,z),dp;
> p;
   ? `p` is undefined
> map f = A, x+y,x-y,z; // homomorphism f: A -> B with f(a)=x+y, f(b)=x-y, f(c)=z
> poly q = f(p); q;
z3+x2-y2+2x
> 
```

A map $f : A \to B$ is always defined from within B!

```
> ring A = 0,(a,b,c),dp;
> poly p = a + b + a*b + c^3;
> p;
c3+ab+a+b
> ring B = 0,(x,y,z),dp;
> p;
   ? `p` is undefined
> map f = A, x+y,x-y,z; // homomorphism f: A -> B with f(a)=x+y, f(b)=x-y, f(c)=z
> poly q = f(p); q;
z3+x2-y2+2x
> // canonically realized maps:
. ring C = 0,(x,y,c,b,a,z),dp;
> ▮
```

Maps

A map $f : A \to B$ is always defined from within B!

```
> ring A = 0,(a,b,c),dp;
> poly p = a + b + a*b + c^3;
> p;
c3+ab+a+b
> ring B = 0,(x,y,z),dp;
> p;
   ? `p` is undefined
> map f = A, x+y,x-y,z; // homomorphism f: A -> B with f(a)=x+y, f(b)=x-y, f(c)=z
> poly q = f(p); q;
z3+x2-y2+2x
> // canonically realized maps:
. ring C = 0,(x,y,c,b,a,z),dp;
> imap(A,p);             // preserves names of variables
c3+ba+b+a
> ▮
```

Maps

A map $f : A \to B$ is always defined from within B!

```
> ring A = 0,(a,b,c),dp;
> poly p = a + b + a*b + c^3;
> p;
c3+ab+a+b
> ring B = 0,(x,y,z),dp;
> p;
   ? `p` is undefined
> map f = A, x+y,x-y,z; // homomorphism f: A -> B with f(a)=x+y, f(b)=x-y, f(c)=z
> poly q = f(p); q;
z3+x2-y2+2x
> // canonically realized maps:
. ring C = 0,(x,y,c,b,a,z),dp;
> imap(A,p);              // preserves names of variables
c3+ba+b+a
> fetch(A,p);             // preserves position of variables
c3+xy+x+y
> ▮
```

Maps

A map $f : A \to B$ is always defined from within B!

```
> ring A = 0,(a,b,c),dp;
> poly p = a + b + a*b + c^3;
> p;
c3+ab+a+b
> ring B = 0,(x,y,z),dp;
> p;
   ? `p` is undefined
> map f = A, x+y,x-y,z; // homomorphism f: A -> B with f(a)=x+y, f(b)=x-y, f(c)=z
> poly q = f(p); q;
z3+x2-y2+2x
> // canonically realized maps:
. ring C = 0,(x,y,c,b,a,z),dp;
> imap(A,p);             // preserves names of variables
c3+ba+b+a
> fetch(A,p);            // preserves position of variables
c3+xy+x+y
> setring B;             // switch between rings
> ▮
```

Maps

A map $f : A \to B$ is always defined from within B!

```
> ring A = 0,(a,b,c),dp;
> poly p = a + b + a*b + c^3;
> p;
c3+ab+a+b
> ring B = 0,(x,y,z),dp;
> p;
   ? `p` is undefined
> map f = A, x+y,x-y,z; // homomorphism f: A -> B with f(a)=x+y, f(b)=x-y, f(c)=z
> poly q = f(p); q;
z3+x2-y2+2x
> // canonically realized maps:
. ring C = 0,(x,y,c,b,a,z),dp;
> imap(A,p);            // preserves names of variables
c3+ba+b+a
> fetch(A,p);           // preserves position of variables
c3+xy+x+y
> setring B;            // switch between rings
> poly p = x3+y3+(x-y)*x2y2+z2;
> █
```

Maps

A map $f : A \to B$ is always defined from within B!

```
> ring A = 0,(a,b,c),dp;
> poly p = a + b + a*b + c^3;
> p;
c3+ab+a+b
> ring B = 0,(x,y,z),dp;
> p;
   ? `p` is undefined
> map f = A, x+y,x-y,z; // homomorphism f: A -> B with f(a)=x+y, f(b)=x-y, f(c)=z
> poly q = f(p); q;
z3+x2-y2+2x
> // canonically realized maps:
. ring C = 0,(x,y,c,b,a,z),dp;
> imap(A,p);            // preserves names of variables
c3+ba+b+a
> fetch(A,p);           // preserves position of variables
c3+xy+x+y
> setring B;            // switch between rings
> poly p = x3+y3+(x-y)*x2y2+z2;
> ideal I = p, p^2*(2x-y);
> ▮
```

Maps

A map $f : A \to B$ is always defined from within B!

```
> ring A = 0,(a,b,c),dp;
> poly p = a + b + a*b + c^3;
> p;
c3+ab+a+b
> ring B = 0,(x,y,z),dp;
> p;
   ? `p` is undefined
> map f = A, x+y,x-y,z; // homomorphism f: A -> B with f(a)=x+y, f(b)=x-y, f(c)=z
> poly q = f(p); q;
z3+x2-y2+2x
> // canonically realized maps:
. ring C = 0,(x,y,c,b,a,z),dp;
> imap(A,p);             // preserves names of variables
c3+ba+b+a
> fetch(A,p);            // preserves position of variables
c3+xy+x+y
> setring B;             // switch between rings
> poly p = x3+y3+(x-y)*x2y2+z2;
> ideal I = p, p^2*(2x-y);
> // is (1,2,0) in V(I)?
. subst(I,x,1,y,2,z,0); // substitution of variables
_[1]=5
_[2]=0
```

```
> ▮
```

Standard bases and normal forms


```
> // is (1,2,0) in V(I)?
. ideal G = groebner(I);
> ▮
```

Standard bases and normal forms


```
> // is (1,2,0) in V(I)?
. ideal G = groebner(I);
> NF((x-1)*(y-2)*z,G);
xyz-2xz-yz+2z
```

There are many procedures for computing standard bases:

- `std`
- `slimgb`
- `groebner`
- `stdhilb`
- `fglm`
- `modStd`

Standard bases and normal forms


```
> // is (1,2,0) in V(I)?
. ideal G = groebner(I);
> NF((x-1)*(y-2)*z,G);
xyz-2xz-yz+2z
```

There are many procedures for computing standard bases:

- `std`
- `slimgb`
- `groebner`
- `stdhilb`
- `fglm`
- `modStd`

From our experience:
If you compute in a commutative ring, use `std` or `groebner`.

```
> // is (1,2,0) in V(I)?
. ideal G = groebner(I);
> NF((x-1)*(y-2)*z,G);
xyz-2xz-yz+2z
```

There are many procedures for computing standard bases:

- `std`
- `slimgb`
- `groebner`
- `stdhilb`
- `fglm`
- `modStd`

From our experience:
If you compute in a commutative ring, use `std` or `groebner`.
If you compute in a Weyl algebra, you **want** to use `slimgb`.

```
> // is (1,2,0) in V(I)?
. ideal G = groebner(I);
> NF((x-1)*(y-2)*z,G);
xyz-2xz-yz+2z
```

There are many procedures for computing standard bases:

- `std`
- `slimgb`
- `groebner`
- `stdhilb`
- `fglm`
- `modStd`

From our experience:
If you compute in a commutative ring, use `std` or `groebner`.
If you compute in a Weyl algebra, you **want** to use `slimgb`.
Note: SINGULAR does not compute reduced standard bases by default.
To force reduced SBs, use: `option(redSB); option(redTail);`

Let M $= \bigoplus_{i\in\mathbb{Z}} M_i$ be a graded module over $\mathbb{K}[x_1,\ldots,x_n]$.
The *Hilbert function* of M is given by $H_M : \mathbb{Z} \to \mathbb{Z}, k \mapsto \dim_{\mathbb{K}}(M_k)$.
The *Hilbert-Poincaré series* of M is the power series

$$\mathrm{HP}_M(t) := \sum_{i=-\infty}^{\infty} H_M(i)t^i = \frac{Q(t)}{(1-t)^n} = \frac{P(t)}{(1-t)^{\dim(M)}}$$

for polynomials $Q(t), P(t) \in \mathbb{Z}[t]$, which are called the *first* and *second Hilbert series*, respectively.
If $P(t) = \sum_{k=0}^{N} a_k t^k$ and $d = \dim(M)$, then

$$H_M(s) = \sum_{k=0}^{N} a_k \binom{d+s-k-1}{d-1} \text{ for } s \geq N.$$

The latter expression is called the *Hilbert polynomial* of M.

Further Reading

The SINGULAR Team
SINGULAR *Online Manual*
http://www.singular.uni-kl.de

Stefan Steidel
SINGULAR–*Tutorial*. Summer School in Trieste, 2008
http://www.mathematik.uni-kl.de/~steidel/Talks/singular_tutorial_Trieste.pdf

Gert-Martin Greuel and Gerhard Pfister
A SINGULAR *Introduction to Commutative Algebra*
Springer, 2nd edition, 2008
With contributions by O. Bachmann, C. Lossen and H. Schönemann

```
>
```

Further Reading

The SINGULAR Team
SINGULAR *Online Manual*
http://www.singular.uni-kl.de

Stefan Steidel
SINGULAR–*Tutorial*. Summer School in Trieste, 2008
http://www.mathematik.uni-kl.de/~steidel/Talks/singular_tutorial_Trieste.pdf

Gert-Martin Greuel and Gerhard Pfister
A SINGULAR *Introduction to Commutative Algebra*
Springer, 2nd edition, 2008
With contributions by O. Bachmann, C. Lossen and H. Schönemann

```
> exit;
Auf Wiedersehen.
```

¡Muchas gracias!

Overview

Further Reading

The SINGULAR Team
SINGULAR *Online Manual*
http://www.singular.uni-kl.de

Daniel Andres, Michael Brickenstein, Viktor Levandovskyy, Jorge Martín-Morales and Hans Schönemann
Constructive D-Module Theory with SINGULAR
Mathematics in Computer Science, 4(2):359–383, 2010.


```
> ▮
```

Further Reading

The Singular Team
Singular *Online Manual*
http://www.singular.uni-kl.de

Daniel Andres, Michael Brickenstein, Viktor Levandovskyy, Jorge Martín-Morales and Hans Schönemann
Constructive D-Module Theory with Singular
Mathematics in Computer Science, 4(2):359–383, 2010.

```
> exit;
Auf Wiedersehen.
```

¡Muchas gracias!