A Short Introduction to Singular

Thomas Keilen
Fachbereich Mathematik
Universität Kaiserslautern
67553 Kaiserslautern
keilen@mathematik.uni-kl.de

SINGULAR Version 2.0.4
Universität Kaiserslautern
Fachbereich Mathematik und Zentrum für Computeralgebra
Autoren: G.–M. Greuel, G. Pfister, H. Schönemann

SINGULAR Version 2.0.4
Universität Kaiserslautern
Fachbereich Mathematik und Zentrum für Computeralgebra
Autoren: G.–M. Greuel, G. Pfister, H. Schönemann

A SHORT INTRODUCTION TO SINGULAR

THOMAS KEILEN

CONTENTS

This short introduction to the computer algebra system SINGULAR does not claim to be complete. It introduces step by step basic structures and commands in SINGULAR. The introduction is not written in a strictly systematic manner. Therefore, for

a systematical and complete documentation of SINGULAR, we refer to the manual [DGPS10]. Anyone wishing to install SINGULAR on their personal computer can find the sources and the installation instructions on the SINGULAR home page:

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

1. First steps

1.1. **Notations.** The following notations will be used in this introduction:

- SINGULAR input and output as well as set words will be written in typewriter face, e.g. `exit;` oder `help`.
- The symbol $\mapsto$ starts SINGULAR output, e.g.:

```
int i=5;
i;
↦ 5
```

- Square brackets mark the parts of the syntax which are optional, that is, can be left out. E.g.

```
pmat(M[,n]);
```

  The above command, a procedure of the library `matrix.lib` is used to show a matrix M as a formatted matrix. The optional parameter `n` defines the width of the columns. If this is missing, a standard value will be used.
- Keys are also shown in typewriter face, such as:

  `n` (press the key `n`),
  `RETURN` (press the enter key),
  `CTRL-P` (press the control key and `P` simultaneously).

1.2. **Starting and terminating Singular.** Obviously, the first question is, how does one start the programme and how can it be terminated? SINGULAR is started by using the command

`Singular`

in the command line of the system.

After the start, SINGULAR shows an input prompt, a >, and is available to the user for interactive use. As soon as the user no longer wants to use this possibility, it is recommended to terminate the programme. There are three commands available for this: `exit;`, `quit;` or, for very lazy users, $.

Please note that the semicolons in the preceding paragraph are part of the SINGULAR commands.

In general, *every* command in Singular ends with a semicolon!

The semicolon tells the computer that the input is to be *interpreted* and, if this is successful, be *carried out*. The programme comes up with a result (possibly an error notification) followed by a new input prompt. Should the user forget the semicolon, SINGULAR shows this with an input prompt ., in words a dot, and enables further inputs, such as the missing semicolon. In this way it is possible to stretch longer commands over several lines.

1.3. **The online help `help`.** The next most important information after the start and terminate commands is how to find help. Here SINGULAR offers the command `help`, or in short `?`. Using the command `help` followed by a SINGULAR command, a SINGULAR function or procedure name or a SINGULAR library, information to the respective objects are shown. For the libraries one receives a list of the procedures contained therein, for commands, functions and procedures their purpose is explained as well as their syntax and one gets examples.

Examples:

```
help exit;
help standard.lib;
help printf;
```

By default an internet browser will be opened and the help will be displayed. Via self–explanatory buttons the entire handbook is available.

1.4. **Interrupt Singular.** Under Unix–like operating systems and under Windows, it is possible, via the key combination `CTRL-C`, to force an interruption in SINGULAR. SINGULAR reacts with an output of the currently performed command and awaits further instructions. The following options are available:

`a`	SINGULAR carries out the current command and returns then to top level,
`c`	SINGULAR carries on,
`q`	the programme SINGULAR is terminated.

1.5. **Editting inputs.** If a command has been misspelled, or if an earlier command is needed again, it is not absolutely necessary to renew the input. Existing SINGULAR text can be edited. For this, SINGULAR records a history of all commands of a SINGULAR session. Below is a selection of the available key combinations for text editing:

`TAB`	automatic completion of function and file names
←	
`CTRL-B`	moves the cursor to the left
→	
`CTRL-F`	moves the cursor to the right
`CTRL-A`	moves the cursor to the beginning of the line
`CTRL-E`	moves the cursor to the end of the line
`CTRL-D`	deletes the letter under the cursor — never use in an empty line!
`BACKSPACE`	
`DEL`	
`CTRL-H`	deletes the letter in front of the cursor
`CTRL-K`	deletes all from the cursor to the end of the line
`CTRL-U`	deletes all from the cursor to the beginning of the line
↓	
`CTRL-N`	supplies the next line from the history
↑	
`CTRL-P`	supplies the preceding line from the history
`RETURN`	sends the current line to the SINGULAR parser

1.6. **Procedures.** The user can create new commands in SINGULAR. These are called procedures. The syntax of a procedure is fairly simple:

```
proc PROCEDURENAME [PARAMETERLIST]
{
  PROCEDUREBODY
}
```

For PROCEDURENAME, any not otherwise resevered sequence of letters can be used. The types and names of the arguments which are passed on to the procedure are laid down in the PARAMETERLIST. The PARAMETERLIST should be encased in round brackets. The PROCEDUREBODY contains a sequence of SINGULAR code. If the procedure is to return a result, the result should be stored in a variable result and the procedure should terminate with the command return(result);.

An example is more useful than thousands of words:

```
proc permcol (matrix A, int c1, int c2)
{
  matrix B=A;
  B[1..nrows(B),c1]=A[1..nrows(A),c2];
  B[1..nrows(B),c2]=A[1..nrows(A),c1];
  return(B);
}
```

The procedure permcol should exchange two columns of a matrix. For this three arguments are necessary. The first argument of name A is of type matrix, the two following arguments c1 and c2 are of type int. SINGULAR instructions follow and the result is stored in the variable B of type matrix, which is then returned with return(B);. This means, in particular, that the result of the procedure is of type matrix.

A procedure can be invoked by entering the procedure name, followed by the arguments in round brackets. E.g.

```
LIB "matrix.lib"; LIB "inout.lib";
ring r=0,(x),lp;
matrix A[3][3]=1,2,3,4,5,6,7,8,9;
pmat(A,2);
↦ 1  2  3
   4  5  6
   7  8  9

matrix B=permcol(A,2,3);
pmat(B,2);
↦ 1  3  2
   4  6  5
   7  9  8
```

Variables, which are defined within a procedure, are only known there and may, therefore, have the same name as objects which are defined outside the procedure.

1.7. **Libraries.** To make procedures available for more than one SINGULAR session, it makes sense to store them in files, which can be loaded as SINGULAR libraries. The library names always have the ending .lib. Libraries are read into SINGULAR through the command LIB followed by library name enclosed in ", such as

```
LIB "123456.lib";
```

(Library names should, if possible, only consist of *eight* letters, to guarantee compatibility with systems such as Dos!) If they are not builtin SINGULAR libraries, then they should be in the subdirectory from which SINGULAR is started.

Of course, a library must conform to certain syntax rules, and procedures, which are stored in libraries, should be extended by two explanatory additions. We show this in an example:

```
////////////////////////////////////////////////////////////////
version="1.0";

info="
     LIBRARY:     linalgeb.lib FIRST STEPS IN LINEAR ALGEBRA

     AUTHOR:      Thomas Markwig, email:  keilen@mathematik.uni-kl.de

     PROCEDURES:
       permcol(matrix,int,int)  vertauscht Spalten der Matrix
       permrow(matrix,int,int)  vertauscht Zeilen der Matrix
";
////////////////////////////////////////////////////////////////
LIB "inout.lib";
////////////////////////////////////////////////////////////////
proc permcol (matrix A, int c1, int c2)
 "USAGE:   permcol(A,c1,c2); A matrix, c1,c2 positive integers
 RETURN:   matrix, A being modified by permuting column c1 and c2
 NOTE:     space for important remark
           can be stretched over several lines
 EXAMPLE:  example permcol; shows an example"
{
  matrix B=A;
  B[1..nrows(B),c1]=A[1..nrows(A),c2];
  B[1..nrows(B),c2]=A[1..nrows(A),c1];
  return(B);
}
example
{
"EXAMPLE:";
echo = 2;
ring r=0,(x),lp;
matrix A[3][3]=1,2,3,4,5,6,7,8,9;
pmat(A);
pmat(permcol(A,2,3));
}

⋮
```

If a double slash `//` in a line appears, the rest of the line is interpreted as a comment and ignored.

The first section is the head of the library. The first line contains the reserved name `version`, through which the version number of the library is fixed. General information to the library follows the reserved name `info`.

It should be noted that under the item `PROCEDURES:` all procedure names are listed with a one-line description.

SINGULAR shows this part when the help command is called on the library, that is

```
help linalgeb.lib;
```

It should also be noted that strings are allocated to `version` and `info` by means of the sign of equality, `=`, so that the `"` are just as necessary as the semicolon at the end of the line!

Section two serves the loading of further libraries, whose procedures one wants to use. As an example, the library `inout.lib`, whose procedure `pmat` in the `example` part of the procedure `permcol` is used.

In the third section the procedures follow one by one. (It should be noted that the command `proc` is always shown at the beginning of a new line!)

It is recommended that the Syntax in section 1.6 is extended by two sections for procedures. A commentary block can be inserted between the procedure head and body, enclosed in `"`, which contains certain key words followed by the relative information. Under `USAGE:` should be shown how the command is invoked and of which type the arguments are. `RETURN:` should contain information on the type of the return value and, if necessary, any further information. `NOTE:` is used to show important comments to the procedure, its use, etc. `EXAMPLE:` shows how an example of the use of the procedure can be displayed in SINGULAR. The commentary block contains the information which is shown when the help command is called for the procedure under, e.g. through

```
help permcol;
```

The second additional section at the end of the procedure is initiated through the reserved name `example`, followed by a section in curly brackets which contains the SINGULAR code. The aim is to show an example for the operation of the procedure which explains its use to the user. The user obtains the example by entering `example PROCEDURENAME;`.

1.8. **Write to files / read from files.** The command `write` offers the possibility to store the values of variables or any string in a file. For this, the variable values are converted to strings. The following lines store variable values, resp. a string, in the file `hallo.txt`:

```
int a=5;
int b=4;
write("hallo.txt",a,b);
write("hallo.txt","This is Singular.");
```

Several variables or strings can be stored at a time, separated by commas. The value of each variable is written in a separate line.

Data contained in a file can be read in by the command `read`. They are, however, interpreted as strings, e.g.

```
read("hallo.txt");
↦ 5
  4
  This is Singular.
```

Should SINGULAR code, which is read in from a file, be recognised as such, then the `read` command must be passed on to the command `execute`. If the file `hallo.txt` contains the following lines,

```
4*5-3;
6/3;
```

then the command

```
execute(read("hallo.txt"));
```

leads to the following SINGULAR output:

```
↦  17
   2
```

A short form for `execute(read(...))` is <, e.g.

```
< "hallo.txt";
```

Anyone wanting to document a SINGULAR session for security in a file, e.g. `hallo.txt`, can do this with the command `monitor`, e.g.

```
monitor("hallo.txt","io");
```

The option `"io"` causes input as well as output to be stored. The omission of one of the letters leads to only the input or only the output being stored. The option `monitor` is very helpful when working on an operating system on which SINGULAR is instable.

Please note that `monitor` opens a file, but does not terminate it. This can be done by the following input:

```
monitor("");
```

2. Types of data in Singular and rings

In SINGULAR different data types are available, and when introducing a variable first one has to specify the data type of the variable. Most data types in SINGULAR depend on a meta structure, the base ring, over which they exist. Exceptions are `string`, `int`, `intvec` und `intmat`. To perform a computation in SINGULAR it is first absolutely necessary to define the ring over which one is working.

`ring r=0,x(0..2),lp;`	The ring of polynomials in the variables $x(0), x(1), x(2)$ with coefficients in the rational numbers $\mathbb{Q}$ and global lexicographical ordering.
`ring r=(0,a,b),(x,y,z),dp;`	The ring of polynomials in the variables x, y, z, where the coefficients are rational functions in the variables a and b. The global degree reverse lexicographical ordering is used.
`ring r=(real,15),x,ls;`	The localised ring of polynomials in the variables x with coefficients in real numbers $\mathbb{R}$ — for computations with 15 places after the decimal point. The local lexicographical ordering is used.
`ring r=5,x,ds;`	The localised ring of polynomials in the variables x with coefficients in $\mathbb{Z}/5\mathbb{Z}$ and local degree reverse lexicographical ordering.

A list of the available data types in SINGULAR is given below.

`int i=1;`	The data type `int` represents the machine integers (between -2^{31} und $2^{31}-1$). In addition, `boolean` values are represented as `integers`, 0 = FALSE, `1 = TRUE`.
`string s="Hallo";`	`strings` are chains of letters enclosed by ".
`intvec iv=1,2,3,4;`	A vector of `integers`.
`intmat im[2][3]=1,2,3,4,5,6;`	A matrix with two lines and three columns with `integer` entries.
`ring R=(0,a),(x,y),lp;`	$\mathbb{Q}(a)[x,y]$ with lexicographical order.
`number n=4/6;`	`numbers` are the elements of the field based on the ring. By `ring r=0,x,lp;` the rational numbers, by `ring r=(0,a),x,lp;` also fractions of polynomials in a with complete number coefficients, i.e. $\frac{a^2+1}{a-1}$.
`list l=n,iv,s;`	A list can contain objects of different types. `l[2]` refers to the second entry of `l`.
`matrix m[2][3]=1,2,3,4,5,6;`	A matrix with two lines and three columns, the entries being of type `poly`,
`vector v=[1,2,3];`	A vector in the module $\mathtt{R}^3$.
`proc`	The data type `procedure` is discussed at length in 1.6.
`poly f=x2+2x+1;`	A polynomial in the indeterminates of the ring with `numbers` as coefficients, here $f = x^2+2x+1$. Note that numbers in front of the monomials are interpreted as coefficients, whereas Singular interprets integers after single variables as exponents.
`ideal i=f,x3;`	The ideal generated by f and x^3.
`qring Q=i;`	The quotient ring `R/i`.
`map g=R,x;`	A map from the ring `R` to the current ring sending the first variable of `R` to x.
`module mo=v,[x,x2,x+1];`	The module generated `v` and $(x, x^2, x+1)$.
`def j;`	In case one does not want to specify the data type yet, one can use the type `def`. The first time a value is assigned to j this value determines the data type of j.
`link`	For the data type `link`, we refer to the handbook [DGPS10].
`resolution`	For the data type `resolution`, we refer to the handbook [DGPS10].

At first glance it might seem as though the matrices `im` and `m` are identical. In the case of Singular that is not the case as they are of different types!

3. Some elements of the programming language Singular

3.1. **Allocations.** In Singular the operator `=` is used to assign a value to a variable. It is possible to assign a value at the time of the definition of the variable,

```
int i=1;
```

or later,

```
int i;
⋮
i=2;
```

3.2. **Loops.** There are two types of loops, the `for` and the `while` loop.

The `for` loop is used typically, if a command sequence is to be performed several times and the number of times is known beforehand. E.g.

```
int s=0;
int i;
for (i=1; i<=10; i=i+1)
{
  s=s+i;
}
```

The command sequence in curly brackets are the commands executed when passing the loop. The commands in round brackets determine how often the loop is to be passed. The first entry fixes the control variable and is here of type `int`; the second entry shows the termination condition, i.e. the loop is passed through as long as this condition is fulfilled; the third entry fixes how the control variable should change in each passage. The example computes the sum of the first ten natural numbers.

`while` loops are used, when the number of passages is not a priori clear. E.g.

```
int s=10000;
int i=1;
while (s > 50)
{
  i=i*i;
  s=s-i;
}
```

Again the command sequence is shown in curly brackets, whilst the termination conditions are shown in round brackets. As long as these show the value `TRUE`, the loop is performed.

The termination condition is checked before the first entry into the loop.

3.3. **Branchings.** SINGULAR offers as a branching the `if-else` command, where, however, the `else` part could be missing. E.g.

```
int i=10;
int s=7;
if (i<5 or s<10)
{
  s=5;
}
else
{
  s=0;
}
```

Again the command sequences are shown as a block in curly brackets, where as the branching conditions are in round brackets.

3.4. **Comparison operators.** In SINGULAR we have the comparison operators `==` and `=!`, with which objects of the same type (e.g. `int`, `string`, `matrix`, etc.) can be compared to one another. `==` tests for equality and supplies the value `1` if the objects are the same and otherwise `0`. `!=` checks for inequality. `<>` has the same effect.

For the data types `int`, `number`, `poly` and `vector`, the operators `<`, `>`, `<=` and `>=` are available. Its significance for `integers` and `monomials` is clear. We refer to the handbook for further data types [DGPS10].

3.5. **Some further operators in Singular.** As we have already seen, the operators may depend on the data types.

`boolean`: For `boolean` variables, the connecting operators `and` and `or` as well as the negating operator `not` are defined.

```
not ((1==0) or (1!=0));
↦ 0
```

`int`: For `integers` the operations `+`, `-` and `*` are entirely clear. `^` means raising to some power

```
int i=4;
i^3;
↦ 64
```

The commands `div` and `mod` are more difficult, whereby the first is synonymous to `/`. If, for two integers, a division with rest is performed, then `mod` supplies the rest, and `div` the result without rest. E.g. $7 = 2 * 3 + 1$, also

```
7 div 3;
↦ 2
7 mod 3;
↦ 1
```

`list`: The following operators are given for the data type `list`.

`+`	Combines the elements of two lists.
`delete`	Deletes an element from a list, `delete(L,3)` deletes the third element of the list `L`.
`insert`	Inserts an element into a list. `insert(L,4)` inserts the element `4` at the start of the list `L`, `insert(L,4,2)` inserts four into into the second position.

`matrix`: The operators `+`, `-` and `*` are available with their obvious meaning. We show, by examples, how single entries of a matrix, resp. whole lines or columns of a matrix, can by accessed:

```
matrix m[2][3]=1,2,3,4,5,6;
print(m);
↦  1,2,3,
   4,5,6
m[1,2];
↦ 2;
m[1,1..3];
↦ 1 2 3
m[1..2,3];
↦ 3 6
```

4. Some selected functions in Singular

SINGULAR has a quite notable arsenal of functions available, which are, in part, integrated in the SINGULAR core, in part made available via libraries. We only

wish to show a small selection of function names, which are useful for computations in linear algebra. Information on their syntax can be found via `help` or in the handbook.

4.1. **Functions which are connected to the data type `matrix`.** `ncols, nrows, print, size, transpose, det`, as functions in the core of SINGULAR. Further the functions of the library `matrix.lib`, in particular `permrow, permcol, multrow, multcol, addrow, addcol, concat, unitmat, gauss_row, gauss_col, rowred, colred`. Also the function `pmat` from the library `inout.lib` is interesting.

4.2. **Functions which are connected to the data type `int`.** `random, gcd, prime` as functions in the core of `Singular`.

5. ESINGULAR - OR THE EDITOR EMACS

There are many editors in which SINGULAR procedures and libraries can be written. On Unix or similar systems the editor emacs (oder Xemacs) should be considered, as it simplifies the entered code through using coloured underlaying of the key words, and they offer many options which simplify editing and error correction.

There is another reason for the recommendation to use Emacs. SINGULAR can be started in a special Emacs mode, as `ESingular`. This means that first the editor Emacs is started and then inside Emacs the programme SINGULAR. The advantage is that apart from the full functionality of the editor Emacs for editing files, a bunch of further options can be made available, which simplify the use — in particular for the inexperienced user, for whom pulldown menu buttons are available. By calling

```
ESingular --emacs=xemacs
```

it is possible to fix the version of Emacs which is to be used, in this case Xemacs. Alternatively, the standard can be changed by means of the environment variable `EMACS`.

6. EXERCISES

Exercise 6.1
Write a procedure `binomi`, which reads in two natural numbers `n` and `k` and returns the binomial coefficient $\binom{n}{k}$. (Convention, if $k < 0$ or $k > n$, then $\binom{n}{k} = 0$.)

Exercise 6.2
Write a procedure `squaresum`, which reads in the natural number `n` and returns the sum of the square numbers $1^2, 2^2, 3^2, \ldots, n^2$.

Exercise 6.3
Write a procedure `minimum`, which reads in a vector of natural numbers and returns the minimum of the numbers.

Exercise 6.4
Write a procedure `rowsumnorm`, `maximumnorm` and `q_eukl_norm`, which read in a $(m \times n)$ matrix `A` of real numbers and calculate

(1) the row-sum-norm of `A` (i.e. $\max_{i=1,\ldots,m} \left(\Sigma_{j=1}^{n} |A_{ij}|\right)$),

(2) the maximum norm of `A` (i.e. $\max\big(|A_{ij}| \;\big|\; i = 1,\dots,m,\ j = 1,\dots,n\big)$), respectively
(3) the square of the euclidian norm of `A` (i.e. $\Sigma_{i,j}|A_{ij}|^2$).

Use the function `abs` from the library `linalg.lib` for the absolute value.

7. Solutions

Solution to Exercise 7.1

```
proc binomi (int n, int k)
"USAGE: binomi(n,k); int n, int k
RETURN: int, binomial coefficient n over k
EXAMPLE: example binomi; shows an example"
{
  if ((k < 0) or (k > n))
  {
   return(0);
  }
  else
  {
   int i;
   int denominator,nominator1,nominator2 = 1,1,1;
   for (i=1;i<=n;i++)
   {
    denominator = denominator * i;
   }
   for (i=1;i<=k;i++)
   {
    nominator1 = nominator1 * i;
   }
   for (i=1;i<=n-k;i++)
   {
    nominator2 = nominator2 *i;
   }
   return (denominator / (nominator1 * nominator2));
  }
}
example
{
  "Example:";
  echo = 2;
  binomi(5,2);
  binomi(7,5);
}
```

Solution to Exercise 7.2

```
proc squaresum (int n)
"USAGE: squaresum(n); int n
RETURN: int, Sum of the first n square numbers
EXAMPLE: example squaresum; shows an example"
{
  if (n < 0)
   {
    return (0);
   }
   else
   {
    int i;
    int result = 0;
    for (i=1;i<=n;i++)
    {
     result = result + i*i;
    }
    return (result);
   }
}
example
{
  "Example:";
  echo = 2;
  squaresum(3);
  squaresum(5);
}
```

Solution to Exercise 7.3

```
proc minimum (intvec iv)
"USAGE: minimum(iv); iv intvector
RETURN: int, the minimum of the entries in iv
EXAMPLE: example minium; shows an example"
{
  int i;
  int k=size(iv);
  int result=iv[1];
  for (i=2;i<=k;i++)
   {
    if (iv[i] < result)
    {
     result=iv[i];
    }
```

```
    }
  return(result);
}
example
{
  "EXAMPLE:";
  echo=2;
  intvec iv=3,2,5,2,1;
  print(iv);
  minimum(iv);
  iv =-3,4,5,3,-6,7;
  print(iv);
  minimum(iv);
}
```

Solution to Exercise 7.4

We write first a short procedure to compute the absolute value. `proc abs_val (poly r)`

```
"USAGE: abs_val(r); poly r - a rational/real number
RETURN: poly, the absolute value of r
EXAMPLE: example abs_value; shows an example"
{
  if (r < 0)
  {
   return(-r);
  }
  else
  {
   return(r);
  }
}
example
{
  "Example:";
  echo = 2;
  ring r=real,x,lp;
  abs_val(-5.45);
  ring s=0,x,lp;
  abs_val(-4/5);
}
```

```
proc rowsumnorm (matrix A)
"USAGE: rowsumnorm(A); matrix A with rational/real entries
RETURN: poly, the row-sum-norm of A
EXAMPLE: example rowsumnorm; shows an example"
```

```
{
  int i,j;
  int n,m = ncols(A),nrows(A);
  poly r,s = 0,0;
  for (i=1;i<=m;i++)
  {
   for (j=1;j<=n;j++)
   {
    r = r + abs(A[i,j]);
   }
   if (r > s)
   {
    s = r;
   }
   r = 0;
  }
  return (s);
}
example
{
  "Example:";
  echo = 2;
  ring r=real,x,lp;
  matrix A[3][2]=-3,-2,-1,3,-4,2;
  print(A);
  rowsumnorm(A);
  ring r=0,x,lp;
  matrix B[3][2]=-7,0,0,3,-4,2;
  print(B);
  rowsumnorm(B);
}
proc maximumnorm (matrix A)
"USAGE: maximumnorm(A); matrix A with rational/real entries
RETURN: poly, the maximum norm of A
EXAMPLE: example maximumnorm; shows an example"
{
  int i,j;
  int n,m = ncols(A),nrows(A);
  poly r = 0;
  for (i=1;i<=m;i++)
  {
   for (j=1;j<=n;j++)
   {
```

```
     if (abs(A[i,j]) > r)
     {
      r = abs(A[i,j]);
     }
    }
   }
   return(r);
}
example
{
   "Example:";
   echo = 2;
   ring r=real,x,lp;
   matrix A[3][2]=-3,-2,-1,3,-4,2;
   print(A);
   maximumnorm(A);
   ring r=0,x,lp;
   matrix B[3][2]=-7,0,0,3,-4,2;
   print(B);
   maximumnorm(B);
}
proc q_eukl_norm (matrix A)
"USAGE: q_eukl_norm(A); matrix A with rational/real entries
RETURN: poly, the square of the euclidean norm of A
EXAMPLE: example q_eukl_norm; shows an example"
{
   int i,j;
   int n,m = ncols(A),nrows(A);
   poly r = 0;
   for (i=1;i<=m;i++)
   {
    for (j=1;j<=n;j++)
    {
     r = r + abs(A[i,j]) * abs(A[i,j]);
    }
   }
   return (r);
}
example
{
   "Example:";
   echo = 2;
   ring r=real,x,lp;
```

```
  matrix A[3][2]=-3,-2,-1,3,-4,2;
  print(A);
  q_eukl_norm(A);
  ring r=0,x,lp;
  matrix B[3][2]=-7,0,0,3,-4,2;
  print(B);
  q_eukl_norm(B);
}
```

References

[DGPS10] Wolfram Decker, Gert-Martin Greuel, Gerhard Pfister, and Hans Schönemann, Singular *3-1-1 — A computer algebra system for polynomial computations*, Tech. report, Centre for Computer Algebra, University of Kaiserslautern, 2010, http://www.singular.uni-kl.de.

Thomas Keilen, Fachbereich Mathematik, Universität Kaiserslautern, 67553 Kaiserslautern

E-mail address: keilen@mathematik.uni-kl.de