User's Guide for `4ti2` version 1.6.16

A software package
for algebraic, geometric and combinatorial problems on linear spaces

November 8, 2026

Contents

1 Beginner's guide **5**

1.1 Linear systems and their encodings 5

1.1.1 Linear systems and integer linear systems 5

1.1.2 Specifying a linear system in `4ti2` 6

1.1.3 What does an explicit solution to linear systems look like? . . 7

1.2 Brief tutorial 8

1.2.1 Solving linear systems over $\mathbb{Z}$ with `zsolve` 8

1.2.2 Solving linear systems over $\mathbb{Q}$ with `qsolve` 10

1.2.3 Computing extreme rays and Hilbert bases 10

1.2.4 Computing circuits and Graver bases 14

1.2.5 Integer programming and toric Gröbner bases 17

1.2.6 Markov Bases in Statistics 19

2 Advanced guide **23**

2.1 Affine systems and their encodings 23

3 Command-line reference **25**

3.1 circuits 25

3.2 genmodel 27

3.3 gensymm 29

3.4 graver 30

3.5 groebner 32
3.6 hilbert 34
3.7 markov 36
3.8 minimize 38
3.9 normalform 39
3.10 output 40
3.11 ppi 43
3.12 qsolve 44
3.13 rays 45
3.14 walk 46
3.15 zbasis 47
3.16 zsolve 48

4 4ti2 as a callable library 51
4.1 C API header file: 4ti2/4ti2.h 51
4.2 Example program: zsolve 54
4.3 Example program: qsolve 56

5 README: Instructions on configuring and building 4ti2 59

6 NEWS: Changes to 4ti2 since version 1.2 63

AUTHORS: The 4ti2 team 69

THANKS 70

Bibliography 72

Chapter 1

Beginner's guide

In this part, we use a few sample problems to introduce you to the basic functionality of `4ti2`. After working through this part, you should know about *linear systems* and their encodings in `4ti2`, and should be able to do computations using the following functions:

- `qsolve`, `rays`, `circuits`
- `zsolve`, `hilbert`, `graver`, `ppi`
- `minimize`, `groebner`, `normalform`
- `genmodel`, `markov`

1.1 Linear systems and their encodings

In this section you learn about the data structure `linear system` and how it is specified in `4ti2`.

1.1.1 Linear systems and integer linear systems

In `4ti2`, a *linear system* is defined by d constraints $Ax \sim b$ in n unknowns x, where each constraint is either $\leq$, $=$ or $\geq$, that is $\sim \in \{\leq, =, \geq\}^d$. Moreover, one may

specify sign constraints on the variables that need to be respected in an explicit continuous/integer representation of all solutions.

There is no particular difference in `4ti2` between a linear system and an integer linear system. Currently, the user chooses between one of the two by calling the appropriate functions on the linear system.

1.1.2 Specifying a linear system in `4ti2`

In order to use a linear system as input, we need to specify its parts to `4ti2`. As our running example, take

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix} x \begin{matrix} \leq \\ \leq \end{matrix} \begin{pmatrix} 6 \\ 10 \end{pmatrix}$$

with sign constraints $(1, 2, 2, 0)$, which we will explain below.

First, we have to give our problem a project name, say `PROJECT`.

- The matrix A has to be put into the file `PROJECT.mat`.

```
2 4
1 1 1 1
1 2 3 4
```

- The relations $\sim$ then have to be specified in `PROJECT.rel`.

```
1 2
< <
```

- The right-hand side vector goes into `PROJECT.rhs`.

```
1 2
6 10
```

- And finally, the sign constraints end up in `PROJECT.sign`.

```
1 4
1 2 2 0
```

Note.

- The input files all have the format of a matrix, preceded by the matrix dimensions. As the dimensions already specify how many symbols have to be read, the matrix could also be given in only one line or even in many lines of different lengths.
- In `4ti2` version 1.3.1 and later, all appearing numbers have to be integers.
- Consequently, this implies that, at the moment, `qsolve` only supports homogeneous linear systems, that is systems with $b = 0$, since minimal inhomogeneous solutions could have rational components.

1.1.3 What does an explicit solution to linear systems look like?

If the system is solved over $\mathbb{R}$ (using `qsolve`), `4ti2` returns two sets of integer vectors:

- a set H of support-minimal homogeneous solutions, and
- a set F defining the linear vector space the solution set lives in.

As only *homogeneous* linear systems are supported in this version of `4ti2`, no list of minimal inhomogeneous solutions is computed. Any solution z of the linear system can now be written as

$$z = \sum \alpha_j h_j + \sum \beta_k f_k \tag{1.1}$$

with $h_j \in H$, $f_k \in F$, and $\alpha_j \geq 0$.

If the system is solved over $\mathbb{Z}$ (using `zsolve`), `4ti2` returns three sets of integer vectors:

- a set H of minimal homogeneous *integer* solutions,
- a set I of minimal inhomogeneous *integer* solutions, and
- a set F defining the sublattice of $\mathbb{Z}^n$ the solution set lives in.

Any solution z of the linear system can now be written as

$$z = i + \sum \alpha_j h_j + \sum \beta_k f_k \tag{1.2}$$

for some $i \in I$ and with $h_j \in H$, $f_j \in F$, and $\alpha_j \in \mathbb{Z}_+$.

Sign file. Let us finally clarify what the sign file `PROJECT.sign` is good for. The sign file may declare a variable to be non-negative (1), to be non-positive (-1), or to consider both cases independently and unite the answers (2). If a nonzero sign has been assigned to a variable, the explicit representations (1.1) and (1.2) above of a solution z have to respect the sign on that variable. The default setting for each variable is 0 (when using `qsolve` and `zsolve`), that is, the sign need not be respected in the explicit representation. In our example above, the first variable is declared to be non-negative, the second and the third one expand to $2 \cdot 2 = 4$ orthant constraints, and the fourth variable is unconstrained. Note, however, that `4ti2` does *not* decompose the problem internally into the four problems with sign patterns $(1, 1, 1, 0)$, $(1, 1, -1, 0)$, $(1, -1, 1, 0)$, and $(1, -1, -1, 0)$, but deals with them more efficiently at the same time.

1.2 Brief tutorial

1.2.1 Solving linear systems over $\mathbb{Z}$ with `zsolve`

In this example you learn about the function `zsolve`.

Let us have a look at the linear system

$$\begin{array}{rcrcl} x & - & y & \leq & 2 \\ -3x & + & y & \leq & 1 \\ x & + & y & \geq & 1 \\ & & y & \geq & 0 \end{array}$$

over $\mathbb{Z}$. We have to create the files encoding the linear system. Let us call our project `system`. Then the input files look as follows:

system.mat	system.rel	system.rhs	system.sign
3 2	1 3	1 3	1 2
1 −1	< < >	2 1 1	0 1
−3 1			
1 1			

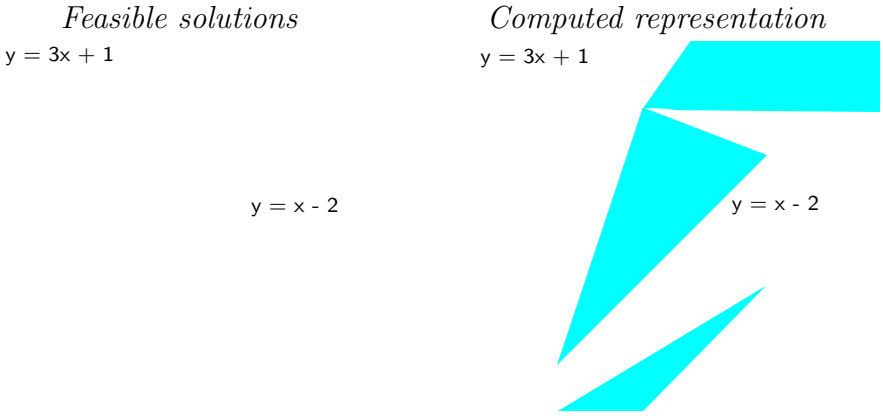
Then we call

```
./zsolve system
```

This call creates two files

system.zinhom	system.zhom
4 2	3 2
0 1	1 1
2 0	1 2
1 0	1 3
1 1	

which correspond to the explicit description of all integer solutions:



$$\left\{\begin{pmatrix}1\\0\end{pmatrix},\begin{pmatrix}2\\0\end{pmatrix},\begin{pmatrix}0\\1\end{pmatrix},\begin{pmatrix}1\\1\end{pmatrix}\right\}+\text{monoid}\left(\begin{pmatrix}1\\1\end{pmatrix},\begin{pmatrix}1\\2\end{pmatrix},\begin{pmatrix}1\\3\end{pmatrix}\right).$$

Note that in the pictures above, we are only interested in the *lattice points* inside the colored regions! The full regions are colored only for the purpose of visualizing the covering of all feasible integer solutions by finitely many shifted copies of the monoid

$$\text{monoid}\left(\begin{pmatrix}1\\1\end{pmatrix},\begin{pmatrix}1\\2\end{pmatrix},\begin{pmatrix}1\\3\end{pmatrix}\right).$$

1.2.2 Solving linear systems over $\mathbb{Q}$ with `qsolve`

`qsolve` solves linear systems over $\mathbb{Q}$; however, note that it only supports homogeneous linear systems, that is, systems with $b = 0$.

```
./qsolve system
```

This call creates files

system.qhom	system.qfree

To solve an inhomogeneous system $Ax = b$, $x \geq 0$, you (still) need to do some work yourself:

1. Solve system $Ax - bu = 0$, $x \geq 0$, $u \geq 0$ using `qsolve`.
2. Keep those solutions with $u = 0$. (These generate the recession cone (of unbounded directions).
3. Normalize those solutions with $u > 0$ to have $u = 1$ (by dividing the vector by u). Be aware that this could create rational numbers.
4. Drop the u-component.

Any solution to $Ax = b$, $x \geq 0$ can then be obtained by adding one solution from 3. to a nonnegative linear combination of solutions from 2.

1.2.3 Computing extreme rays and Hilbert bases

In this example you learn about the functions `rays` and `hilbert`. (They are convenient versions of `qsolve` and `zsolve` for particular cases.)

Let us consider the set of magic 3×3 squares with non-negative real entries, that is, the set of all 3×3 arrays with non-negative real entries whose row sums, column sums, and main diagonal sums all add up to the same number, the magic constant of the square.

Clearly, addition of two magic squares gives another magic square, as well as does multiplication of a magic square by a non-negative number. Therefore, we may talk about the *cone* of magic 3×3 squares. In fact, this cone is a pointed rational polyhedral cone described by the linear system

$$\begin{aligned} x_{11}+x_{12}+x_{13} &= x_{21}+x_{22}+x_{23} \\ &= x_{31}+x_{32}+x_{33} \\ &= x_{11}+x_{21}+x_{31} \\ &= x_{12}+x_{22}+x_{32} \\ &= x_{13}+x_{23}+x_{33} \\ &= x_{11}+x_{22}+x_{33} \\ &= x_{31}+x_{22}+x_{13} \\ & x_{ij} \geq 0, \quad \text{for all } i,j=1,2,3. \end{aligned}$$

Bringing all x_{ij} to the left-hand side of these equations, the matrix $A_{3\times 3}$ defining this linear system is

$$A_{3\times 3} = \begin{pmatrix} 1 & 1 & 1 & -1 & -1 & -1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & -1 & -1 & -1 \\ 0 & 1 & 1 & -1 & 0 & 0 & -1 & 0 & 0 \\ 1 & 0 & 1 & 0 & -1 & 0 & 0 & -1 & 0 \\ 1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 & -1 & 0 & 0 & 0 & -1 \\ 1 & 1 & 0 & 0 & -1 & 0 & -1 & 0 & 0 \end{pmatrix}.$$

Below, we will deal with the more interesting case of *integer* magic squares. For the moment, however, we wish to compute the extreme rays of the magic square cone $\{z : A_{3\times 3}z = 0, z \geq 0\}$.

In order to call the function `rays`, we only have to create one file, say `magic3x3.mat`, in which we specify the problem matrix $A_{3\times 3}$. The remaining data is set by default to

"equations only", to "homogeneous system", and to "all variables are non-negative". Note that we are allowed to change these defaults (except homogeneity) by specifying data in `magic3x3.rel` and `magic3x3.sign`

magic3x3.mat

7	9							
1	1	1	−1	−1	−1	0	0	0
1	1	1	0	0	0	−1	−1	−1
0	1	1	−1	0	0	−1	0	0
1	0	1	0	−1	0	0	−1	0
1	1	0	0	0	−1	0	0	−1
0	1	1	0	−1	0	0	0	−1
1	1	0	0	−1	0	−1	0	0

Now we call

```
./rays magic3x3
```

which creates the single file

magic3x3.ray

4	9							
0	2	1	2	1	0	1	0	2
1	2	0	0	1	2	2	0	1
2	0	1	0	1	2	1	2	0
1	0	2	2	1	0	0	2	1

that corresponds to the four extremal rays of the 3×3 magic square cone:

0	2	1
2	1	0
1	0	2

1	2	0
0	1	2
2	0	1

2	0	1
0	1	2
1	2	0

1	0	2
2	1	0
0	2	1

Every magic 3×3 square is a non-negative linear combination of these four magic squares.

If we turn now to *integer* magic squares, we are looking for a Hilbert basis of the 3×3 magic square cone. As the default settings for `hilbert` are the same as for `rays`, we can use the same input file

magic3x3.mat								
7	9							
1	1	1	−1	−1	−1	0	0	0
1	1	1	0	0	0	−1	−1	−1
0	1	1	−1	0	0	−1	0	0
1	0	1	0	−1	0	0	−1	0
1	1	0	0	0	−1	0	0	−1
0	1	1	0	−1	0	0	0	−1
1	1	0	0	−1	0	−1	0	0

for this computation. However, to compute the Hilbert basis, we call

```
./hilbert magic3x3
```

which creates the single output file

magic3x3.hil								
5	9							
0	2	1	2	1	0	1	0	2
1	2	0	0	1	2	2	0	1
2	0	1	0	1	2	1	2	0
1	0	2	2	1	0	0	2	1
1	1	1	1	1	1	1	1	1

that corresponds to the five elements in the minimal Hilbert basis of the 3×3 magic square cone:

0	2	1
2	1	0
1	0	2

1	2	0
0	1	2
2	0	1

2	0	1
0	1	2
1	2	0

1	0	2
2	1	0
0	2	1

1	1	1
1	1	1
1	1	1

Every integer magic 3×3 square is a non-negative *integer* linear combination of these five integer magic squares. Note that the all-1 square is in the interior of the magic square cone.

See [2, section 3.8] or [6] for details on the algorithm implemented.

1.2.4 Computing circuits and Graver bases

In this example you learn about the functions `graver`, `ppi`, and `circuits`.

As an example of a Graver basis computation, let us compute the primitive partition identities of order $n = 4$. Before we do the simple computation, let us explain what a primitive partition identity is.

A **partition identity** is any identity of the form

$$a_1 + \ldots + a_k = b_1 + \ldots + b_l$$

with (generally not distinct) integer numbers $0 < a_i, b_j \leq n$. A partition identity is called **primitive** if no proper subidentity exists.

For example,

$$1 + 2 + 3 = 2 + 2 + 2$$

is a partition identity which is not primitive, since it contains the subidentity

$$1 + 3 = 2 + 2$$

which is in fact primitive.

The description of the primitive partition identities for fixed n, however, is exactly the description of the Graver basis of the matrix

$$A_n = \begin{pmatrix} 1 & 2 & 3 & \ldots & n \end{pmatrix}.$$

Let us finally do the computation for $n = 3$. We create an input file `ppi3` for `4ti2` which looks as follows:

ppi3.mat
1 3
1 2 3

and call

```
./graver ppi3
```

This call will create an output file `ppi3.gra` that looks like:

ppi3.gra		
5	3	
3	0	−1
2	−1	0
0	3	−2
1	1	−1
1	−2	1

Thus, there are 5 primitive partition identities of order $n = 3$:

$$\begin{aligned} 1+1+1 &= 3 \\ 1+1 &= 2 \\ 2+2+2 &= 3+3 \\ 1+2 &= 3 \\ 1+3 &= 2+2 \end{aligned}$$

You may try and compute the primitive partition identities for bigger n, say $n = 17$, 20, or 23. Be aware, especially the latter two problems take a long, long time. What is the biggest n for which you can compute the primitive partition identities of order n on your machine within one hour?

Due to the very special structure of the matrix, there are algorithmic speed-ups [4, 10, 13]. The currently fastest algorithm to compute primitive partition identities is implemented in the function `ppi` of `4ti2`. Try running

```
./ppi 17
```

which creates two files `ppi17.mat` (so we do not really have to create this file ourselves) and the file `ppi17.gra` containing the desired identities. Compare this running time with the time taken by

```
./graver ppi17
```

Do you notice the speed-up?

Let us now turn to the question of determining the support-minimal partition identities. This, in fact, is the question of computing the circuits of the matrix

$$A_n = \begin{pmatrix} 1 & 2 & 3 & \dots & n \end{pmatrix}.$$

We use the same input file

ppi3.mat
1 3
1 2 3

as above and call

```
./circuits ppi3
```

This call will create an output file `ppi3.cir` that looks like:

ppi3.cir
3 3
3 0 −1
2 −1 0
0 3 −2

Thus, there are 3 support-minimal partition identities of order $n = 3$:

$$\begin{aligned} 1+1+1 &= 3 \\ 1+1 &= 2 \\ 2+2+2 &= 3+3 \end{aligned}$$

Note that support-minimal partition identities are primitive, since the circuits of a matrix are contained in the Graver basis of this matrix.

See the book [2, section 3.8], or Hemmecke [7] for details on the algorithm implemented.

1.2.5 Integer programming and toric Gröbner bases

In this example you learn about the functions `minimize`, `groebner`, and `normalform`.

The following neat example is based on the example presented in [12]. Let us assume that we want to give change worth 99 cents using only pennies (1ct), nickels (5ct), dimes (10ct), and quarters (25ct). Clearly,

$$4 \cdot 1 + 4 \cdot 5 + 0 \cdot 10 + 3 \cdot 25 = 99$$

would be one way to do it. Is this there another choice of 11 coins that sums up to 99ct but uses fewer nickels and quarters (in total)? In other words, we would like to solve

$$\min\{x_2 + x_4 : x_1 + x_2 + x_3 + x_4 = 11, x_1 + 5x_2 + 10x_3 + 25x_4 = 99, x_1, x_2, x_3, x_4 \in \mathbb{Z}_+\}$$

Let us set up the problem in `4ti2`.

4coins.mat	4coins.zsol	4coins.sign	4coins.cost
2 4	1 4	1 4	1 4
1 1 1 1	4 4 0 3	1 1 1 1	0 1 0 1
1 5 10 25			

Note that we do not have to specify a relations file `4coins.rel`, since already by default all relations are assumed to be equations. Now we simply call

```
./minimize 4coins
```

which creates the single output file

4coins.min
1 4
4 1 4 2

From this, we conclude that

$$4 \cdot 1 + 1 \cdot 5 + 4 \cdot 10 + 2 \cdot 25 = 99$$

is an optimal choice, using only 3 instead of 7 nickels and quarters.

Remark. Earlier versions of 4ti2 allowed to specify the right-hand side vector in a file called `4coins.rhs`, instead of giving a solution in `4coins.zsol`. This is no longer supported. □

Since we already know a feasible solution, there is another way we might attack this problem, namely via toric Gröbner bases. (See [2, Chapter 11] for an introduction to toric ideals and their Gröbner bases, and also their generalizations, lattice ideals.) For this, we first need to specify the matrix A and the cost vector c in the two files `4coins.mat` and `4coins.cost`:

4coins.mat	4coins.cost
2 4	1 4
1 1 1 1	0 1 0 1
1 5 10 25	

Then we compute the Gröbner basis of the toric ideal

$$I_A = \langle x^u - x^v : Au = Av, u, v \in \mathbb{Z}_+^4 \rangle$$

with respect to a term ordering $\prec$ compatible with c, that is, $c^\intercal v < c^\intercal u$ implies $x^v \prec x^u$. This toric Gröbner basis is computed by

```
./groebner 4coins
```

and gives the output file

4coins.gro

Remark. Many algorithm options are available and can be selected by command-line options of `groebner`, see section 3.5. As reference to the algorithms we recommend the book [2, section 11.4] or Hemmecke and Malkin [8], as well as Bigatti, LaScala, and Robbiano [1], Gebauer and Möller [3], and Hoşten and Sturmfels [9]. Since runnning times of the various algorithms are hard to predict, it may for some hard problems make sense to start several computations in parallel, each with different algorithms. □

Then we specify our feasible solution in

4coins.feas
1 4
4 4 0 3

and call

```
./normalform 4coins
```

to produce the file

4coins.nf
1 4
4 1 4 2

that also contains the desired optimal solution.

Remark. We could also specify a list of feasible solutions in `4coins.feas`. Then the call

```
./normalform 4coins
```

creates a file `4coins.nf` containing the minima to the corresponding integer programs. (If z_0 is a feasible solution, the corresponding integer program is defined by putting the right-hand side to Az_0.) □

1.2.6 Markov Bases in Statistics

In this example you learn about the functions `markov` and `genmodel`.

Let us consider the following 4×4 table of non-negative integer numbers together with all row and column sums.

$$\begin{array}{cccc} \left(\begin{array}{cccc} 11 & 23 & 34 & 3 \\ 4 & 15 & 12 & 11 \\ 17 & 2 & 3 & 25 \\ 16 & 12 & 22 & 7 \end{array}\right) & \begin{array}{c} 71 \\ 42 \\ 47 \\ 57 \end{array} \\ 48 \quad 52 \quad 71 \quad 46 & \end{array}$$

In statistics, one wishes to sample among arrays that have fixed counts, say fixed row and column sums. In order to sample, one needs a set of moves that, in particular, do not change the counts when added to the current table. Clearly, these moves must have counts 0 and thus quite naturally lead us to the toric ideal

$$I_A = \langle x^u - x^v : Au = Av, u, v \in \mathbb{Z}_+^{16} \rangle,$$

where

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

It turns out that for any set of fixed counts, a (minimal) *Markov basis* is given by a minimal generating set of this toric ideal. Note that a Markov basis connects all non-negative tables with these counts in the sense that for any two non-negative tables T_1 and T_2 with these counts, there is a sequence of non-negative tables $T_1 = S_0, \ldots, S_N = T_2$ with the same counts as T_1 and T_2 and such that $S_i - S_{i-1}$ or $S_{i-1} - S_i$ is in the Markov basis for $i = 1, \ldots, N$.

For two-way tables the situation is still very simple as our computations with 4×4 tables will now demonstrate. Write the matrix that defines our toric ideal in the file `4x4.mat`:

4x4.mat															
8	16														
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0
0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1

Let us compute the Markov basis via the call

```
./markov 4x4
```

which creates a single output file `4x4.mar` containing the 36 Markov basis elements. Up to symmetry (swapping rows or columns), the Markov basis consists of the single move

$$\begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

In fact, this elementary move is (up to symmetry) the only representative of the minimal Markov moves for arbitrary $m \times n$ tables using row and column counts.

Creating the matrices for statistical models may be pretty cumbersome. `4ti2` provides a litte function, `genmodel`, that helps the user with creating matrices for hierarchical models defined by a complex.

The $m \times n$ tables problem above corresponds to the complex $\{\{1\},\{2\}\}$ on two nodes with levels m and n, respectively. Let us encode the complex for 3×6 tables with 1-marginals (row and column sums) in `3x6.mod`.

3x6.mod
3
3 6
2
1 1
1 2

and call

```
./genmodel 3x6
```

to produce the desired matrix file `3x6.mat`.

The encoding of the complex should be obvious from the example: first we state the number of nodes and their levels, then we give the number of maximal faces. Finally,

we list each maximal face by first specifying the number of nodes on it and then by listing these nodes.

Thus, a $3 \times 4 \times 6$ table with 2-marginals (that is, again only counts along coordinate axes) corresponds to the complex $\{\{(1,2)\},\{(2,3)\},\{(3,1)\}\}$ on 3 nodes with levels 3, 4, and 6, respectively. Thus, its encoding is in `4ti2` would look like:

3x4x6.mod
3
3 4 6
3
2 1 2
2 2 3
2 3 1

A binary model on the bipartite graph $K_{2,3}$ then reads as

k2_3.mod
5
2 2 2 2 2
6
2 1 3
2 1 4
2 1 5
2 2 3
2 2 4
2 2 5

Chapter 2

Advanced guide

In this part, we deal with several more advanced problem specifications in `4ti2`.

First we introduce *affine systems* and their encodings. In fact, affine systems are the basic objects used in `4ti2`, since every linear system is transformed into an affine system. However, in the integer situation, it is not always possible to transform an affine system back into a linear system without adding variables or modulo constraints.

2.1 Affine systems and their encodings

Let $a + \mathcal{L}_{\mathbb{Z}}$ be an "integer linear affine space" given by the vector $a \in \mathbb{Z}^n$ and by generators for the lattice $\mathcal{L}_{\mathbb{Z}} \subseteq \mathbb{Z}^n$. We wish to find a finite sign-compatible description for the set of all (integer) vectors $x \in a + \mathcal{L}_{\mathbb{Z}}$.

As an example, let consider the linear space $\mathcal{L}_{\mathbb{R}}$ and the lattice $\mathcal{L}_{\mathbb{Z}}$ both spanned by the two vectors $(1, -2, 1, 0)$ and $(2, -3, -0, 1)$. Moreover, consider the sign-constraints $(1, 2, 2, 0)$. Thus, we are looking for a finite explicit sign-compatible description for all x that can be written as

$$x = \begin{pmatrix} 1 & 2 \\ -2 & -3 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \lambda,$$

with $\lambda \in \mathbb{R}^2$ and $\lambda \in \mathbb{Z}^2$, respectively.

In order to solve this affine system using `zsolve`, we create the following input files to encode the affine system:

affine.lat	affine.sign
$\begin{array}{cccc} 2 & 4 & & \\ 1 & -1 & 1 & 0 \\ 2 & -3 & 0 & 1 \end{array}$	$\begin{array}{cccc} 1 & 4 & & \\ 1 & 2 & 2 & 0 \end{array}$

and then call

```
./zsolve affine
```

This creates the files `affine.zhom` and `affine.zinhom`.

Chapter 3

Command-line reference

3.1 circuits

```
Usage: circuits [options] PROJECT

Computes the circuits of a cone.

Input Files:
  PROJECT.mat         A matrix (compulsory).
  PROJECT.sign        The sign constraints of the variables ('1' means
                      non-negative, '0' means a free variable, and '2' means
                      both non-negative and non-positive).
                      It is optional, and the default is both.
  PROJECT.rel         The relations on the matrix rows ('<','>','=').
                      It is optional and the default is all '='.
                      The mat must be given with this file.
Output Files:
  PROJECT.cir         The circuits of the cone.
  PROJECT.qfree       A basis for the linear subspace of the cone.
                      If this file does not exist then the linear subspace
                      is trivial.

Options:
  -p, --precision=PREC       Select PREC as the integer arithmetic precision.
```

```
                          PREC is one of the following: ‘64’ (default),
                          ‘32’, and ‘arbitrary’ (only ‘arb‘ is needed).
  -m, --mat               Use the Matrix algorithm (default for 32 and 64).
  -s, --support           Use the Support algorithm (default for arbitrary).
  -o, --order=ORDERING    Set ORDERING as the ordering in which the columns
                          are chosen. The possible orderings are ‘maxinter’,
                          ‘minindex’, ‘maxcutoff’ (default), and ‘mincutoff’.
  -f, --output-freq=n     Set the frequency of output (default is 1000).
  -q, --quiet             Do not output anything to the screen.
  -h, --help              Display this help and exit.
```

3.2 genmodel

```
usage: genmodel [--options] FILENAME

Computes the problem matrix corresponding to graphical statistical models
given by a simplicial complex and levels on the nodes.

Options:
 -q, --quiet No output is written to the screen

Input file:
FILENAME.mod    Simplicial complex and levels on the nodes

Output file:
FILENAME.mat    Matrix file

Example: Consider the problem of 3x3x3 tables with 2-marginals. These
are given by K_3 as the simplicial complex on 3 nodes and with levels
of 3 on each node.  In '333.mod' write:
3
3 3 3
3
2 1 2
2 2 3
2 3 1
Calling 'genmodel 333' produces the following file '333.mat':
27 27
1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
[...]
1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[...]
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
[...]
```

3.3 gensymm

```
usage: gensymm [--options] A B C D FILENAME

Computes the generators for the symmetry group acting on 4-way tables
with 3-marginals. By putting one side length to 1, this includes
3-way tables with 2-marginals.

Options:
 -q, --quiet       No output is written to the screen

Output file:
 FILENAME.sym      generators for the symmetry group

Example:  Consider the problem of 3x3x3 tables with 2-marginals. Calling
  gensymm 3 3 3 1 333
produces the file '333.sym' containing the following lines.

9 27
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 1 2 3 4 5 6 7 8 9 19 20 21 22 23 24 25 26 27
4 5 6 7 8 9 1 2 3 13 14 15 16 17 18 10 11 12 22 23 24 25 26 27 19 20 21
4 5 6 1 2 3 7 8 9 13 14 15 10 11 12 16 17 18 22 23 24 19 20 21 25 26 27
2 3 1 5 6 4 8 9 7 11 12 10 14 15 13 17 18 16 20 21 19 23 24 22 26 27 25
2 1 3 5 4 6 8 7 9 11 10 12 14 13 15 17 16 18 20 19 21 23 22 24 26 25 27
1 2 3 10 11 12 19 20 21 4 5 6 13 14 15 22 23 24 7 8 9 16 17 18 25 26 27
1 10 19 4 13 22 7 16 25 2 11 20 5 14 23 8 17 26 3 12 21 6 15 24 9 18 27
1 4 7 2 5 8 3 6 9 10 13 16 11 14 17 12 15 18 19 22 25 20 23 26 21 24 27
```

3.4 graver

```
Usage: graver [options] PROJECT

Computes the Graver basis of a matrix or a given lattice.

Basic options:
 -p PREC, --precision=PREC  Use precision (32, 64, gmp). Default is 32 bit
 -m, --maxnorm              Write vectors with maximum norm to PROJECT.maxnorm
 -b [FREQ], --backup[=FREQ] Frequently backup status to PROJECT.backup
 -r, --resume               Resume from backup file PROJECT.backup
 -h, --help                 Display this help
 --version                  Display version information

Output options:
 -q, --quiet        Quiet mode
 -u, --update[=1]   Updated output on console (default)
 -uu, --update=2    More verbose updated output on console
 -v, --verbose[=1]  Output once every variable computation
 -vv, --verbose=2   Output once every norm sum computation
 -vvv, --verbose=3  Output once every norm computation

Logging options:
 -n, --log=0    Disable logging (default)
 -l, --log[=1]  Log once every variable computation to PROJECT.log
 -ll, --log=2   Log once every norm sum computation to PROJECT.log
 -lll, --log=3  Log once every norm computation to PROJECT.log

Input files:
PROJECT.mat     Matrix
PROJECT.lat     Lattice basis (can be provided instead of matrix)
PROJECT.sign    Sign of columns (optional)
PROJECT.lb      Lower bounds of columns (optional)
PROJECT.ub      Upper bounds of columns (optional)

Backup files:
```

```
PROJECT.backup  Backup file
PROJECT.backup~ Temporary backup file
                (if it exists, it may be newer than PROJECT.backup)

Output files:
PROJECT.gra     Graver basis
PROJECT.zfree   Free part of the solution
PROJECT.maxnorm Vectors with maximum norm (if -m, --maxnorm is in use)
```

3.5 groebner

```
Usage: groebner [options] PROJECT

Computes a Groebner basis of the toric ideal of a matrix,
or, more general, of the lattice ideal of a lattice.

Input Files:
  PROJECT.mat         A matrix (optional if lattice basis is given).
  PROJECT.lat         A lattice basis (optional if matrix is given).
  PROJECT.cost        The cost matrix, which determines the term ordering
                      (optional, default is degrevlex).
                           Ties are broken with degrevlex.
  PROJECT.sign        The sign constraints of the variables ('1' means
                      non-negative and '0' means a free variable).
                      It is optional, and the default is all non-negative.
  PROJECT.mar         The Markov basis/generating set of the lattice (optional).
  PROJECT.weights     The weight vectors used for truncation (optional).
  PROJECT.weights.max The maximum weights used for truncation.
                      This file is needed when PROJECT.weights exists.
  PROJECT.zsol        An integer solution to specify a fiber (optional).
                      The integer solution is used for truncation.
Output Files:
  PROJECT.gro         The Groebner basis of the lattice.

Options:
  -p, --precision=PREC       Select PREC as the integer arithmetic precision.
                             PREC is one of the following: '64' (default),
                             '32', and 'arbitrary' (only 'arb' is needed).
  -a, --algorithm=ALG        Select ALG as the completion procedure for
                             computing Groebner bases. ALG is one of
                             'fifo', 'weighted', or 'unbounded.'
  -g, --generation=ALG       Select ALG as the procedure for computing
                             a generating set or Markov basis. ALG is
                             one of 'hybrid' (default), 'project-and-lift',
                             'max-min', or 'saturation'.
```

```
-t, --truncation=TRUNC     Set TRUNC as the truncation method.  TRUNC is
                           of the following: ‘ip’, ‘lp’, ‘weight’ (default),
                           or ‘none’. Only relevant if ‘zsol’ is given.
-m, --minimal=STATE        If STATE is ‘yes’ (default), then 4ti2 will
                           compute a minimal Markov basis. If STATE is
                           ’no’, then the Markov basis will not
                           necessarily be minimal.
-r, --auto-reduce-freq=n   Set the frequency of auto reduction.
                           (default is 2500).
-f, --output-freq=n        Set the frequency of output (default is 1000).
-q, --quiet                Do not output anything to the screen.
-h, --help                 Display this help and exit.
```

3.6 hilbert

```
Usage: hilbert [options] PROJECT

Computes the Hilbert basis of a matrix or a given lattice.

Basic options:
 -p PREC, --precision=PREC  Use precision (32, 64, gmp). Default is 32 bit
 -m, --maxnorm              Write vectors with maximum norm to PROJECT.maxnorm
 -b [FREQ], --backup[=FREQ] Frequently backup status to PROJECT.backup
 -r, --resume               Resume from backup file PROJECT.backup
 -h, --help                 Display this help
 --version                  Display version information

Output options:
 -q, --quiet        Quiet mode
 -u, --update[=1]   Updated output on console (default)
 -uu, --update=2    More verbose updated output on console
 -v, --verbose[=1]  Output once every variable computation
 -vv, --verbose=2   Output once every norm sum computation
 -vvv, --verbose=3  Output once every norm computation

Logging options:
 -n, --log=0    Disable logging (default)
 -l, --log[=1]  Log once every variable computation to PROJECT.log
 -ll, --log=2   Log once every norm sum computation to PROJECT.log
 -lll, --log=3  Log once every norm computation to PROJECT.log

Input files:
PROJECT.mat     Matrix
PROJECT.lat     Lattice basis (can be provided instead of matrix)
PROJECT.rel     Relations (<, >, =)
PROJECT.sign    Sign of columns (optional)
PROJECT.ub      Upper bounds of columns (optional)

Backup files:
```

```
PROJECT.backup  Backup file
PROJECT.backup~ Temporary backup file
                (if it exists, it may be newer than PROJECT.backup)

Output files:
PROJECT.hil     Hilbert basis
PROJECT.zfree   Free part of the solution
PROJECT.maxnorm Vectors with maximum norm (if -m, --maxnorm is in use)
```

3.7 markov

```
Usage: markov [options] PROJECT

Computes a Markov basis (generating set) of the toric ideal
of a matrix or, more general, of the lattice ideal of a lattice.

Input Files:
  PROJECT             A matrix (optional only if lattice basis is given).
  PROJECT.lat         A lattice basis (optional only if matrix is given).
  PROJECT.sign        The sign constraints of the variables ('1' means
                      non-negative and '0' means a free variable).
                      It is optional, and the default is all non-negative.
  PROJECT.weights     The weight vectors used for truncation (optional).
  PROJECT.weights.max The maximum weights used for truncation.
                      This file is needed when PROJECT.weights exists.
  PROJECT.zsol        An integer solution to specify a fiber (optional).
                      The integer solution is used for truncation.
Output Files:
  PROJECT.mar         The Markov basis/generating set of the lattice.
Options:
  -p, --precision=PREC       Select PREC as the integer arithmetic precision.
                             PREC is one of the following: '64' (default),
                             '32', and 'arbitrary' (only 'arb' is needed).
  -a, --algorithm=ALG        Select ALG as the completion procedure for
                             computing Groebner bases. ALG is one of
                             'fifo', 'weighted', or 'unbounded.'
  -g, --generation=ALG       Select ALG as the procedure for computing
                             a generating set or Markov basis. ALG is
                             one of 'hybrid' (default), 'project-and-lift',
                             'max-min', or 'saturation'.
  -t, --truncation=TRUNC     Set TRUNC as the truncation method.  TRUNC is
                             of the following: 'ip', 'lp', 'weight' (default),
                             or 'none'. Only relevant if 'zsol' is given.
  -m, --minimal=STATE        If STATE is 'yes' (default), then 4ti2 will
                             compute a minimal Markov basis. If STATE is
```

```
                          'no', then the Markov basis will not
                          necessarily be minimal.
  -r, --auto-reduce-freq=n  Set the frequency of auto reduction.
                          (default is 2500).
  -f, --output-freq=n       Set the frequency of output (default is 1000).
  -q, --quiet               Do not output anything to the screen.
  -h, --help                Display this help and exit.
```

3.8 minimize

```
Usage: minimize [options] PROJECT

Computes the minimal solution of an integer linear program
or, more general, a lattice program, using a Groebner basis.

Input Files:
  PROJECT.mat         A matrix (optional only if lattice basis is given).
  PROJECT.lat         A lattice basis (optional only if matrix is given).
  PROJECT.cost        The cost vector. Exactly one vector allowed.
  PROJECT.zsol        An integer solution to specify a fiber (needed).
  PROJECT.sign        The sign constraints of the variables ('1' means
                      non-negative and '0' means a free variable).
                      It is optional, and the default is all non-negative.
Output Files:
  PROJECT.min         The minimal solution for the given fiber.

Options:
  -p, --precision=PREC       Select PREC as the integer arithmetic precision.
                             PREC is one of the following: `64' (default),
                             `32', and `arbitrary' (only `arb` is needed).
  -a, --algorithm=ALG        Select ALG as the completion procedure for
                             computing Groebner bases. ALG is one of
                             `fifo', `weighted', or 'unbounded.'
  -t, --truncation=TRUNC     Set TRUNC as the truncation method.  TRUNC is
                             of the following: `ip', `lp', `weight' (default),
                             or `none'. Only relevant if `zsol' is given.
  -r, --auto-reduce-freq=n   Set the frequency of auto reduction.
                             (default is 2500).
  -f, --output-freq=n        Set the frequency of output (default is 1000).
  -q, --quiet                Do not output anything to the screen.
  -h, --help                 Display this help and exit.
```

3.9 normalform

```
Usage: normalform [options] PROJECT

Computes the normal form of a list of feasible points.

Input Files:
  PROJECT.mat         A matrix (optional if lattice basis is given).
  PROJECT.lat         A lattice basis (optional if matrix is given).
  PROJECT.gro         The Groebner basis of the lattice (needed).
  PROJECT.cost        The cost matrix (optional, default is degrevlex).
                      Ties are broken with degrevlex.
  PROJECT.feas        An list of integer feasible solutions (needed).
  PROJECT.sign        The sign constraints of the variables ('1' means
                      non-negative and '0' means a free variable).
                      It is optional, and the default is all non-negative.
Output Files:
  PROJECT.nf          The normal forms of the feasible solutions.

Options:
  -p, --precision=PREC       Select PREC as the integer arithmetic precision.
                             PREC is one of the following: `64' (default),
                             `32', and `arbitrary' (only `arb` is needed).
  -q, --quiet                Do not output anything to the screen.
  -h, --help                 Display this help and exit.
```

3.10 output

```
usage: output [--options] FILENAME.EXT

Transforms a 4ti2 matrix file to something else.

General options:
 --quiet        No output is written to the screen.

Options that control what to output:                 their output files:
 --binomials    Write vectors as binomials.          FILENAME.EXT.bin
                Use an optional input file
                'FILENAME.EXT.vars'
                to specify variable names.

 --maple        Write vectors as Maple list.         FILENAME.EXT.maple
                This format is suitable also for
                CoCoA, Mathematica, Macaulay2.

 --0-1          Extract vectors with 0-1
                components only.                     FILENAME.EXT.0-1

 --transpose    Transpose matrix and write it        FILENAME.EXT.tra
                in 4ti2 format.

 --degree       Print 1-norms of all vectors.

 --degree N     Extract all vectors of 1-norm        FILENAME.EXT.deg.N
                equal to N.

 --degree N1 N2 Extract all vectors of 1-norm        FILENAME.EXT.deg.N1-N2
                between N1 and N2 (inclusive).

 --support      Print supports of all vectors.

 --support S    Extract all vectors of support       FILENAME.EXT.supp.S
```

```
               size equal to S.

 --support S1 S2    Extract all vectors of support   FILENAME.EXT.supp.S1-S2
                    size between S1 and S2 (incl.)

 --positive         Extract positive parts of vectors.  FILENAME.EXT.pos
                    Corresponds to leading terms of
                    binomials.

 --3way A B C       Write vectors as 3-way tables       FILENAME.EXT.3way
                    of size A x B x C.

 --nonzero-at K Extract all vectors that have           FILENAME.EXT.nonzero.K
                    nonzero K-th coordinate.

Undocumented or obscure options for experts:
 --representatives
 --dominated        Extract all non-dominated vectors   FILENAME.EXT.nondom
 --maximal-non-dominated                                FILENAME.EXT.maxnondom
 --expand-representatives-to-full-orbits
 --type T
 --AxB              Computes a matrix-vector product.
 --macaulay2
 --mathematica
 --cocoa
 --sum              Print the sum of the columns.
 --submatrix LISTFILENAME                               FILENAME.EXT.submat
 --remove-column I                                      FILENAME.EXT.remcol
 --remcol I                                             FILENAME.EXT.remcol
 --stabilizer SYMMFILENAME                              FILENAME.EXT.stab
 --fill-column                                          FILENAME.EXT.fil
 --add-column                                           FILENAME.EXT.addcol
 --fix I1 ... IK    Extract fixed vectors,              FILENAME.EXT.fix
                    that is, those vectors that
                    have x[i]=i for the given i.
 --fox I1 ... IK    Extract relaxed fixed vectors.      FILENAME.EXT.fox
```

```
 --initial-forms    Extract initial forms.          FILENAME.ini
                    (Call with FILENAME rather      FILENAME.ini.bin
                    than FILENAME.EXT.  Reads
                    FILENAME.gro and
                    optionally FILENAME.cost and
                    FILENAME.vars.

Examples:
 'output --binomials file.gra' writes the Graver basis elements as
 binomials in 'file.gra.bin'.

 'output --0-1 foo.gra' extracts the 0-1 elements from the Graver basis
 elements and writes them into 'foo.gra.0-1'.
```

3.11 ppi

```
usage: ppi [--binary-output] N

Computes the primitive partition identities, that is, the Graver basis of [1 2 3 ... N].

Options:
 -b, --binary-output       Create a binary file ppiN.dat instead of text file ppiN.gra
```

3.12 qsolve

```
Usage: qsolve [options] PROJECT

Computes a generator description of a cone.

Input Files:
  PROJECT.mat         A matrix (compulsory).
  PROJECT.sign        The sign constraints of the variables ('1' means
                      non-negative, '0' means a free variable, and '2' means
                      both non-negative and non-positive).
                      It is optional, and the default is all free.
  PROJECT.rel         The relations on the matrix rows ('<','>','=').
                      It is optional and the default is all '='.
                      The mat must be given with this file.
Output Files:
  PROJECT.qhom        The homogeneous generators of the linear system.
  PROJECT.qfree       A basis for the linear subspace of the cone.
                      If this file does not exist then the linear subspace
                      is trivial.

Options:
  -p, --precision=PREC       Select PREC as the integer arithmetic precision.
                             PREC is one of the following: `64' (default),
                             `32', and `arbitrary' (only `arb` is needed).
  -m, --mat                  Use the Matrix algorithm (default for 32 and 64).
  -s, --support              Use the Support algorithm (default for arbitrary).
  -o, --order=ORDERING       Set ORDERING as the ordering in which the columns
                             are chosen. The possible orderings are `maxinter',
                             `minindex', `maxcutoff' (default), and `mincutoff'.
  -f, --output-freq=n        Set the frequency of output (default is 1000).
  -q, --quiet                Do not output anything to the screen.
  -h, --help                 Display this help and exit.
```

3.13 rays

```
Usage: rays [options] PROJECT

Computes the extreme rays of a cone.

Input Files:
  PROJECT.mat         A matrix (compulsory).
  PROJECT.sign        The sign constraints of the variables ('1' means
                      non-negative, '0' means a free variable, and '2' means
                      both non-negative and non-positive).
                      It is optional, and the default is all non-negative.
  PROJECT.rel         The relations on the matrix rows ('<','>','=').
                      It is optional and the default is all '='.
                      The mat must be given with this file.
Output Files:
  PROJECT.ray         The extreme rays of the cone.
  PROJECT.qfree       A basis for the linear subspace of the cone.
                      If this file does not exist then the linear subspace
                      is trivial.

Options:
  -p, --precision=PREC       Select PREC as the integer arithmetic precision.
                             PREC is one of the following: '64' (default),
                             '32', and 'arbitrary' (only 'arb' is needed).
  -m, --mat                  Use the Matrix algorithm (default for 32 and 64).
  -s, --support              Use the Support algorithm (default for arbitrary).
  -o, --order=ORDERING       Set ORDERING as the ordering in which the columns
                             are chosen. The possible orderings are 'maxinter',
                             'minindex', 'maxcutoff' (default), and 'mincutoff'.
  -f, --output-freq=n        Set the frequency of output (default is 1000).
  -q, --quiet                Do not output anything to the screen.
  -h, --help                 Display this help and exit.
```

3.14 walk

```
Usage: walk [options] PROJECT

Computes the minimal solution of an integer linear program
or, more general, a lattice program using a Groebner basis.

Input Files:
  PROJECT.mat         A matrix (optional only if lattice basis is given).
  PROJECT.lat         A lattice basis (optional only if matrix is given).
  PROJECT.gro.start   The starting Groebner basis (needed).
  PROJECT.gro.cost    The starting cost vector (optional, default is degrevlex).
                      Ties are broken with degrevlex.
  PROJECT.cost        The target cost vector (optional, default is degrevlex).
                      Ties are broken with degrevlex.
  PROJECT.zsol        An integer solution to specify a fiber (needed).
  PROJECT.sign        The sign constraints of the variables ('1' means
                      non-negative and '0' means a free variable).
                      It is optional, and the default is all non-negative.
Output Files:
  PROJECT.gro         The Groebner basis of the lattice.

Options:
  -p, --precision=PREC       Select PREC as the integer arithmetic precision.
                             PREC is one of the following: `64' (default),
                             `32', and `arbitrary' (only `arb' is needed).
  -t, --truncation=TRUNC     Set TRUNC as the truncation method.  TRUNC is
                             of the following: `ip', `lp', `weight' (default),
                             or `none'. Only relevant if `zsol' is given.
  -f, --output-freq=n        Set the frequency of output (default is 1000).
  -q, --quiet                Do not output anything to the screen.
  -h, --help                 Display this help and exit.
```

3.15 zbasis

```
Usage: zbasis [options] PROJECT

Computes an integer lattice basis.

Input Files:
  PROJECT             A matrix (needed).
Output Files:
  PROJECT.lat         A lattice basis.
Options:
  -p, --precision=PREC       Select PREC as the integer arithmetic precision.
                             PREC is one of the following: ‘64’ (default),
                             ‘32’, and ‘arbitrary’ (only ‘arb‘ is needed).
  -q, --quiet                Do not output anything to the screen.
  -h, --help                 Display this help and exit.
```

3.16 zsolve

```
Usage: zsolve [options] PROJECT

Solves linear inequality and equation systems over the integers.

Basic options:
 -p PREC, --precision=PREC  Use precision (32, 64, gmp). Default is 32 bit
 -m, --maxnorm              Write vectors with maximum norm to PROJECT.maxnorm
 -b [FREQ], --backup[=FREQ] Frequently backup status to PROJECT.backup
 -r, --resume               Resume from backup file PROJECT.backup
 -h, --help                 Display this help
 --version                  Display version information

Output options:
 -q, --quiet        Quiet mode
 -u, --update[=1]   Updated output on console (default)
 -uu, --update=2    More verbose updated output on console
 -v, --verbose[=1]  Output once every variable computation
 -vv, --verbose=2   Output once every norm sum computation
 -vvv, --verbose=3  Output once every norm computation

Logging options:
 -n, --log=0    Disable logging (default)
 -l, --log[=1]  Log once every variable computation to PROJECT.log
 -ll, --log=2   Log once every norm sum computation to PROJECT.log
 -lll, --log=3  Log once every norm computation to PROJECT.log

Input files:
PROJECT.mat     Matrix
PROJECT.lat     Lattice basis (can be provided instead of matrix)
PROJECT.rhs     Right hand side
PROJECT.rel     Relations (<, >, =)
PROJECT.sign    Sign of columns (optional)
PROJECT.lb      Lower bounds of columns (optional)
PROJECT.ub      Upper bounds of columns (optional)
```

```
Backup files:
PROJECT.backup  Backup file
PROJECT.backup~ Temporary backup file
                (if it exists, it may be newer than PROJECT.backup)

Output files:
PROJECT.zinhom  Inhomogeneous part of the solution
PROJECT.zhom    Homogeneous part of the solution
PROJECT.zfree   Free part of the solution
PROJECT.maxnorm Vectors with maximum norm (if -m, --maxnorm is in use)
```

Chapter 4

4ti2 as a callable library

Some portions of `4ti2` can be used as a callable library to avoid I/O and process overhead. It has a simple C API that closely mirrors the commands: `qsolve`, `rays`, `circuits`, `zsolve`, `hilbert`, `graver`.

Depending on its configuration, `4ti2` builds and installs several libraries, either as static or shared libraries, using `libtool`.

The functions equivalent to `zsolve`, `hilbert`, `graver` require the use of `libzsolve`. The functions equivalent to `qsolve`, `rays`, `circuits` require the use of `lib4ti2common` and, depending on the arithmetic precision requested, the use of `lib4ti2int32`, `lib4ti2int64`, or `lib4ti2gmp`.

4.1 C API header file: 4ti2/4ti2.h

A single header file, `<4ti2/4ti2.h>`, provides the C API. It is reproduced below for reference.

```
/*
4ti2 -- A software package for algebraic, geometric and combinatorial
problems on linear spaces.

Copyright(C) 2008 4ti2 team.
Main author(s): Peter Malkin

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
```

```
as published by the Free Software Foundation; either version 2
of the License, or(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
*/

#ifndef _4ti2_H
#define _4ti2_H

#include <inttypes.h>

#include "4ti2/4ti2_config.h"

#ifdef _4ti2_HAVE_GMP
#include <gmp.h>
#endif

#ifdef __cplusplus
extern "C"
{
#endif

// Enum representing the possible arithmetic precision settings available.
typedef enum { _4ti2_PREC_INT_32 = 32, _4ti2_PREC_INT_64 = 64, _4ti2_PREC_INT_ARB = 0 } _4ti2_precision;

// Enum representing values describing the constraints on a variable or row of the constraint matrix.
typedef enum { _4ti2_FR = 0, _4ti2_LB = 1, _4ti2_UB = -1, _4ti2_DB = 2, _4ti2_FX = 3 } _4ti2_constraint;

// Enum representing the exit status of an API call to 4ti2.
typedef enum { _4ti2_OK = 0, _4ti2_ERROR = 1 } _4ti2_status;

// 4ti2 data structures.
typedef struct _4ti2_state _4ti2_state;
typedef struct _4ti2_matrix _4ti2_matrix;

// Create a QSolve 4ti2 state object.
_4ti2_state* _4ti2_qsolve_create_state(_4ti2_precision prec);

// Create a QSolve 4ti2 rays object.
_4ti2_state* _4ti2_rays_create_state(_4ti2_precision prec);

// Create a QSolve 4ti2 circuits object.
_4ti2_state* _4ti2_circuits_create_state(_4ti2_precision prec);

// Create a ZSolve 4ti2 state object.
```

```
_4ti2_state* _4ti2_zsolve_create_state(_4ti2_precision prec);

// Create a ZSolve 4ti2 state object.
_4ti2_state* _4ti2_hilbert_create_state(_4ti2_precision prec);

// Create a ZSolve 4ti2 state object.
_4ti2_state* _4ti2_graver_create_state(_4ti2_precision prec);

// Read in options for the 4ti2 state object.
// These options are exactly the same as the command line options without the project filename at the end.
// Note that argv[0] is ignored!
_4ti2_status _4ti2_state_set_options(_4ti2_state* state, int argc, char** argv);

#if 0
  /* Implementation does not exist. --mkoeppe */
// Read in the state object from "project"
_4ti2_status _4ti2_state_read(_4ti2_state* state, const char* project);

// Write out the state object to "project"
_4ti2_status _4ti2_state_write(_4ti2_state* state, const char* project);
#endif

// Deletes a 4ti2 state object.
void _4ti2_state_delete(_4ti2_state* state);

// Runs the main algorithm of the 4ti2 state object.
_4ti2_status _4ti2_state_compute(_4ti2_state* state);

// Create a 4ti2 matrix. Previous matrix is deleted if it exists.  Pointer is 0 if "name" is not valid.
_4ti2_status _4ti2_state_create_matrix(_4ti2_state* state, int num_rows, int num_cols, const char* name, _4ti2_matrix** mat

// Read a 4ti2 matrix from a file.  Previous matrix is deleted if it exists. Returns 0 if "name" is not valid.
_4ti2_status _4ti2_state_read_matrix(_4ti2_state* state, const char* filename, const char* name, _4ti2_matrix** matrix);

// Get a 4ti2 matrix.  Returns 0 if "name" is not valid or if matrix has not been created.
_4ti2_status _4ti2_state_get_matrix(_4ti2_state* state, const char* name, _4ti2_matrix** matrix);

// Returns the number of rows of the matrix.
int _4ti2_matrix_get_num_rows(const _4ti2_matrix*  matrix);

// Returns the number of columns of the matrix.
int _4ti2_matrix_get_num_cols(const _4ti2_matrix*  matrix);

// Write the 4ti2 matrix to stdout.
void _4ti2_matrix_write_to_stdout(const _4ti2_matrix*  matrix);

// Write the 4ti2 matrix to sterr.
void _4ti2_matrix_write_to_stderr(const _4ti2_matrix*  matrix);

// Write the 4ti2 matrix to the file called "filename".
void _4ti2_matrix_write_to_file(const _4ti2_matrix*  matrix, const char* filename);
```

```
// Operations on the matrix.
_4ti2_status _4ti2_matrix_set_entry_int32_t(_4ti2_matrix*  matrix, int r, int c, int32_t value);

_4ti2_status _4ti2_matrix_get_entry_int32_t(const _4ti2_matrix*  matrix, int r, int c, int32_t* value);

_4ti2_status _4ti2_matrix_set_entry_int64_t(_4ti2_matrix*  matrix, int r, int c, int64_t value);

_4ti2_status _4ti2_matrix_get_entry_int64_t(const _4ti2_matrix*  matrix, int r, int c, int64_t* value);

#ifdef _4ti2_HAVE_GMP
_4ti2_status _4ti2_matrix_set_entry_mpz_ptr(_4ti2_matrix*  matrix, int r, int c, mpz_ptr value);

_4ti2_status _4ti2_matrix_get_entry_mpz_ptr(const _4ti2_matrix*  matrix, int r, int c, mpz_ptr value);
#endif

#ifdef __cplusplus
} // extern "C"
#endif

#endif
```

4.2 Example program: zsolve

Example programs using the C API can be found in the source tree of `4ti2`, in the directories `test/qsolve/api` and `test/zsolve/api`.

Below we reproduce the example program `test/zsolve/api/test_zsolve_api.cpp`.

```
/*
4ti2 -- A software package for algebraic, geometric and combinatorial
problems on linear spaces.

Copyright (C) 2006 4ti2 team.
Main author(s): Peter Malkin.

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
```

```
*/

#include "4ti2/4ti2.h"

int
main()
{
    // Input data.
    const int m = 4;
    const int n = 3;
    int64_t mat[m][n] = {
                { 2,  3, -6 },
                { 2, -1, -4 },
                { 1,  2, -11 },
                { 1, -1,  1 }
            };
    int64_t rel[m] = { _4ti2_LB, _4ti2_UB, _4ti2_UB, _4ti2_LB };
    int64_t sign[n] = { _4ti2_LB, _4ti2_LB, _4ti2_LB };

///     // Output data.
///     const int k = 4;
///     int64_t zhom[k][n] = {
///                 { 3, 4, 1 },
///                 { 3, 8, 5 },
///                 { 9, 2, 4 },
///                 {19,18, 5 }
///             };

    _4ti2_state* zsolve_api = _4ti2_zsolve_create_state(_4ti2_PREC_INT_64);
    const int argc = 2;
    char*argv[2] = { (char*)"zsolve", (char*)"-q" };
    _4ti2_state_set_options(zsolve_api, argc, argv);

    _4ti2_matrix* cons_matrix;
    _4ti2_state_create_matrix(zsolve_api, m, n, "mat", &cons_matrix);
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            _4ti2_matrix_set_entry_int64_t(cons_matrix, i, j,  mat[i][j]);
        }
    }
    //_4ti2_matrix_write_to_stdout(cons_matrix);

    _4ti2_matrix* rel_matrix;
    _4ti2_state_create_matrix(zsolve_api, 1, m, "rel", &rel_matrix);
    for (int i = 0; i < m; ++i) {
        _4ti2_matrix_set_entry_int64_t(rel_matrix, 0, i, rel[i]);
    }
    //_4ti2_matrix_write_to_stdout(rel_matrix);

    _4ti2_matrix* sign_matrix;
    _4ti2_state_create_matrix(zsolve_api, 1, n, "sign", &sign_matrix);
    for (int i = 0; i < n; ++i) {
```

```
        _4ti2_matrix_set_entry_int64_t(sign_matrix, 0, i, sign[i]);
    }
    //_4ti2_matrix_write_to_stdout(sign_matrix);

    _4ti2_state_compute(zsolve_api);

    _4ti2_matrix* zhom_matrix;
    _4ti2_state_get_matrix(zsolve_api, "zhom", &zhom_matrix);
    //_4ti2_matrix_write_to_stdout(zhom_matrix);

    // Check the output
///     if (_4ti2_matrix_get_num_rows(zhom_matrix) != k) { return 1; }
///     if (_4ti2_matrix_get_num_cols(zhom_matrix) != n) { return 1; }
///     for (int i = 0; i < k; ++i) {
///         for (int j = 0; j < n; ++j) {
///             int64_t value;
///             _4ti2_matrix_get_entry_int64_t(zhom_matrix, i, j, &value);
///             if (value != zhom[i][j]) { return 1; }
///         }
///     }

    _4ti2_state_delete(zsolve_api);
    return 0;
}
```

4.3 Example program: qsolve

Below we reproduce the example program `test/qsolve/api/qsolve_main.cpp`.

```
/*
4ti2 -- A software package for algebraic, geometric and combinatorial
problems on linear spaces.

Copyright (C) 2006 4ti2 team.
Main author(s): Peter Malkin.

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
```

```
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
*/

#include "4ti2/4ti2.h"

int
main()
{
    // Input data.
    const int m = 4;
    const int n = 3;
    int64_t mat[m][n] = {
                { 2,  3, -6 },
                { 2, -1, -4 },
                { 1,  2, -11 },
                { 1, -1,  1 }
            };
    int64_t rel[m] = { _4ti2_LB, _4ti2_UB, _4ti2_UB, _4ti2_LB };
    int64_t sign[n] = { _4ti2_LB, _4ti2_LB, _4ti2_LB };

    // Output data.
    const int k = 4;
    int64_t qhom[k][n] = {
                { 3, 4, 1 },
                { 3, 8, 5 },
                { 9, 2, 4 },
                {19,18, 5 }
            };

    _4ti2_state* qsolve_api = _4ti2_qsolve_create_state(_4ti2_PREC_INT_64);
    const int argc = 2;
    char*argv[2] = { (char*) "qsolve", (char*) "-q" };
    _4ti2_state_set_options(qsolve_api, argc, argv);

    _4ti2_matrix* cons_matrix;
    _4ti2_state_create_matrix(qsolve_api, m, n, "mat", &cons_matrix);
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            _4ti2_matrix_set_entry_int64_t(cons_matrix, i, j,  mat[i][j]);
        }
    }
    //_4ti2_matrix_write_to_stdout(cons_matrix);

    _4ti2_matrix* rel_matrix;
    _4ti2_state_create_matrix(qsolve_api, 1, m, "rel", &rel_matrix);
    for (int i = 0; i < m; ++i) {
        _4ti2_matrix_set_entry_int64_t(rel_matrix, 0, i, rel[i]);
    }
    //_4ti2_matrix_write_to_stdout(rel_matrix);

    _4ti2_matrix* sign_matrix;
```

```
    _4ti2_state_create_matrix(qsolve_api, 1, n, "sign", &sign_matrix);
    for (int i = 0; i < n; ++i) {
        _4ti2_matrix_set_entry_int64_t(sign_matrix, 0, i, sign[i]);
    }
    //_4ti2_matrix_write_to_stdout(sign_matrix);

    _4ti2_state_compute(qsolve_api);

    _4ti2_matrix* qhom_matrix;
    _4ti2_state_get_matrix(qsolve_api, "qhom", &qhom_matrix);
    //_4ti2_matrix_write_to_stdout(qhom_matrix);

    // Check the output
    if (_4ti2_matrix_get_num_rows(qhom_matrix) != k) { return 1; }
    if (_4ti2_matrix_get_num_cols(qhom_matrix) != n) { return 1; }
    for (int i = 0; i < k; ++i) {
        for (int j = 0; j < n; ++j) {
            int64_t value;
            _4ti2_matrix_get_entry_int64_t(qhom_matrix, i, j, &value);
            if (value != qhom[i][j]) { return 1; }
        }
    }

    _4ti2_state_delete(qsolve_api);
    return 0;
}
```

Chapter 5

README: Instructions on configuring and building 4ti2

```
4ti2 -- A software package for algebraic, geometric and combinatorial
problems on linear spaces.

Copyright (C) 1998, 2002, 2006, 2015, 2025 4ti2 team.

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

COMPILING 4ti2
==============
```

```
If you checked out the sources using git, first, to generate the build system,
you need Automake, Autoconf, and Libtool. Run the following command:

 ./autogen.sh

Run the following commands with the 4ti2 directory:

 ./configure --prefix=INSTALLATION-DIRECTORY
 make
 make check
    make install-exec

The final command will install 4ti2 in a directory tree below the
INSTALLATION-DIRECTORY that you gave with the first command.  If you
omit the --prefix option, ‘make install’ will install 4ti2 in the
/usr/local hierarchy.

You will need glpk and gmp installed first (see below).

The first command, 'make', compiles all the executables. The second
command, 'make check', runs a lot of automatic checks. This will take
a while.  If a check fails, then please notify the 4ti2 team.

You will need gcc version 3.4 or higher.

You will need an installed version of glpk (linear programming software). See
the website http://www.gnu.org/software/glpk for more information. The
version 4.7 has been tested. If you do not have glpk installed or 4ti2 cannot
find glpk, then the compilation will fail saying that it cannot find the file
"glpk.h".  If you have installed glpk but not in a location that 4ti2 finds by
default, then you will need to invoke

     ./configure --with-glpk=/ROOT/OF/GLPK/INSTALLATION/HIERARCHY

You will also need an installed version of gmp, The GNU MP
```

```
Bignum Library, with c++ support enabled (see http://www.swox.com/gmp/ for more
details).  Versions 4.2.1 and 4.1.4 have been tested. If you are compiling a
version of gmp from the source, make sure that you enable c++ support
(--enable-cxx configure option).  If you have
installed gmp but not in a location that 4ti2 finds by default, then you
will need to invoke

     ./configure --with-gmp=/ROOT/OF/GMP/INSTALLATION/HIERARCHY

If you have gmp but not with c++ support, then ./configure will
fail with an error saying that the file "gmpxx.h" cannot be found.


USING MACPORTS ON MAC OS X
==========================

Use the following commands.

     sudo port install gmp glpk
     ./configure --with-gmp=/opt/local --with-glpk=/opt/local
     make
     sudo make install


INSTALLATION ON WINDOWS USING CYGWIN
====================================

1. Install Cygwin from https://www.cygwin.com/

   In the installer, select the following packages:

   Devel: gcc-core gcc-g++ make
   Math: glpk gmp libglpk-devel libgmp-devel

2. Make sure you unpack the 4ti2 sources into a
   C:\DIRECTORY\WITHOUT\SPACES\IN\IT
```

```
   (for example, C:\4ti2)

3. Open the Cygwin terminal

4. Type:

  cd /cygdrive/c/DIRECTORY/WITHOUT/SPACES/IN/IT
  ./configure
  make
  make install

5. Now you can run 4ti2's commands from the Cygwin terminal.


DOCUMENTATION
=============

See the manual or the website https://4ti2.github.io/ for information on using 4ti2.


PYTHON INTERFACES
=================

- https://github.com/alfsan/Py4ti2

- https://pypi.org/project/passagemath-latte-4ti2/
```

Chapter 6

NEWS: Changes to 4ti2 since version 1.2

```
See https://github.com/4ti2/4ti2/releases for newer news.

News in 4ti2 version 1.6.9, compared to 1.6.8:

 * Updates to test suite.

 * qsolve/groebner code: More detailed warnings for PROJECT
   vs. PROJECT.mat

 * Fix out of bounds vector access in circuits.
   Reported by Jerry James for Fedora.

News in 4ti2 version 1.6.8, compared to 1.6.7:

 * Updates to test suite and build.

 * Fix _4ti2_rays_create_state, _4ti2_circuits_create_state.
   Reported by Alfredo Sánchez-R. Navarro.

 * Merge Debian patch for PATH_MAX on Hurd.
   Patch from Jerome Benoit.
```

```
News in 4ti2 version 1.6.7, compared to 1.6.6:

 * Add missing amsabbrvurl.bst file required for rebuilding the documentation.
   Reported by Jerome Benoit for Debian.

 * Add tests for "walk -p arb" to testsuite

 * Build fix for Debian bug 801117 (underlinked library).
   Patch from Jerome Benoit.

 * Fix division-by-zero in "walk -p arb" for testcase 344.
   Reported by Jerome Benoit for Debian.

News in 4ti2 version 1.6.6, compared to 1.6.5:

 * Fix segfault in graver when a matrix with trivial kernel is input
   (testcase graver/trivial-kernel).
   Reported by Alfredo Sanchez.

News in 4ti2 version 1.6.5, compared to 1.6.4:

 * Fix build failure with gcc 4.9.2.

News in 4ti2 version 1.6.4, compared to 1.6.3:

 * Improved error checking while reading zsolve input files.
   Reported by Sebastian Gutsche.

 * The PDF manual has been updated to include a reference to commands
   and their options and a reference to the API.  The command
   reference on www.4ti2.de has also been updated.

 * Better option handling.  Make long options available in non-GNU
   platforms such as Mac OS X.  All commands now support the
   standard --help and --version options.
```

```
 * Minor fix to the test suite.
   Reported by Luis David Garcia-Puente.

News in 4ti2 version 1.6.3, compared to 1.6.2:

 * The manual has been updated.

 * Minor build fixes.

News in 4ti2 version 1.6.2, compared to 1.6.1:

 * Use GLPK's new glp_* API instead of the old lpx_* API (declared
   obsolete in glpk 4.47 and removed in 4.52).
   (Patch by Jerry James for Fedora.)

News in 4ti2 version 1.6.1, compared to 1.6:

 * Compile fix for XCode 5.0.2, Apple LLVM version 5.0 (clang-500.2.79).

News in 4ti2 version 1.6, compared to 1.5.2:

 * Restore the functionality of "hilbert" in versions up to 1.3.2 to
   accept "rel" files.  This signalled an error in the 1.4 and 1.5 series.
   (Note that "zsolve" did accept "rel" files in the 1.4 and 1.5 series.)

 * When the cone is not pointed, "hilbert" now outputs a "zfree" file,
   containing a lattice basis, in addition to the "hil" file.

   Note that in the non-pointed case, Hilbert bases are not uniquely
   determined.  Let zfree_1, ..., zfree_k be the vectors in the "zfree"
   file and hil_1, ..., hil_l be the vectors in the "hil" file.
   Then a Hilbert basis of the non-pointed cone is
   hil_1, ..., hil_l, -(hil_1 + ... + hil_l), zfree_1, ..., zfree_k.

   (In the 1.3 series, "hilbert" silently appended the lattice
```

```
   generators to the "hil" file.  Thus the list of vectors in the
   "hil" file was not a Hilbert basis of the non-pointed cone; this
   was a bug.  Note that "zsolve" did work correctly in the 1.3
   series.)

 * Fix a bug of zsolve and hilbert on 64-bit platforms (where
   sizeof(unsigned long) > sizeof(int)), which affected problems with
   more than 32 variables and could lead to wrong results. (Testcases
   a1, dutour-testcase-2013-08-21).

 * Accept longer filenames.

 * Enable shared library builds on the Cygwin platform (using the
   libtool -no-undefined flag).  (However, this requires that shared
   libraries of GMP, GLPK are available.)

 * Use gnulib to provide getopt_long if not available in the
   system libraries.

 * If the C++ compiler does not have int32_t and int64_t, use int and
   long int instead.

 * Fixed bug in lattice transformation with too few rows.
   (Reported by Jerry James for Fedora.)

 * Fix a build failure with gcc 4.7.
   (Patch by Jerry James for Fedora.)


News in 4ti2 version 1.5.2, compared to 1.5.1:

 * Build a GMP-only 4ti2 if the C++ compiler does not have int32_t and int64_t.


News in 4ti2 version 1.5.1, compared to 1.5:
```

```
 * Fix a build problem with --enable-shared.


News in 4ti2 version 1.5, compared to 1.4:

 * Latest version of new qsolve.


News in 4ti2 version 1.4, compared to 1.3.2:

 * Portability fixes

 * New abstract C and C++ API (callable library), header files in 4ti2/

 * New implementation of zsolve in C++


News in 4ti2 version 1.3.2, compared to 1.3.1:

 * New build system, using GNU Autoconf, Automake, and Libtool.

   This allows 4ti2 to be built using the standard "./configure &&
   make && make install" sequence.

 * Bug fixes

 * Portability fixes (for GCC versions 4.3.x and 4.4.x)


News in 4ti2 version 1.3.1, compared to 1.2:

 * 'groebner' and 'markov' are again heavily improved.

 * 'groebner' and 'markov' allow non-homogeneous lattice ideals.

 * 'groebner' and 'markov' allow truncation.
```

* There is a new function 'walk' performing a Gröbner walk.

* There are new functions 'qsolve' and 'zsolve' for solving linear systems over the reals or the integers, respectively.

* There are new functions 'rays' and 'circuits' to compute extreme rays and circuits.

* The functions 'circuits' and 'graver' allow to fix certain orthants.

* One may compute with projections by specifying variables to be ignored.

* There is a new function 'minimize' to solve integer linear programs.

AUTHORS: The 4ti2 team

Ralf Hemmecke
Raymond Hemmecke
Matthias Koeppe
Peter Malkin
Matthias Walter

THANKS

Many thanks to Kris Nairn for suggesting the name '4ti2'. Cool. :-)

Moreover, lots of thanks to many 4ti2 users for suggesting many interesting and useful improvements.

Bibliography

[1] A. M. Bigatti, R. LaScala, and L. Robbiano, *Computing toric ideals*, Journal of Symbolic Computation **27** (1999), 351–365.

[2] J. A. De Loera, R. Hemmecke, and M. Köppe, *Algebraic and geometric ideas in the theory of discrete optimization*, MOS–SIAM Series on Optimization, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2013, doi:10.1137/1.9781611972443, ISBN 978-1-61197-243-6.

[3] R. Gebauer and H. M. Möller, *On an installation of Buchberger's algorithm*, Journal of Symbolic Computation **6** (1988), 275–286.

[4] U.-U. Haus, M. Köppe, and R. Weismantel, *A primal all-integer algorithm based on irreducible solutions*, Math. Programming, Series B **96** (2003), no. 2, 205–246.

[5] R. Hemmecke, *Exploiting symmetries in the computation of Graver bases*, 2004, arXiv:math.CO/0410334.

[6] R. Hemmecke, *On the computation of Hilbert bases of cones*, Mathematical Software, ICMS 2002 (A. M. Cohen, X. S. Gao, and N. Takayama, eds.), World Scientific, 2002.

[7] ———, *On the positive sum property and the computation of Graver test sets*, Math. Programming, Series B **96** (2003), 247–269.

[8] R. Hemmecke and P. N. Malkin, *Computing generating sets of lattice ideals and Markov bases of lattices*, Journal of Symbolic Computation **44** (2009), 1463–1476.

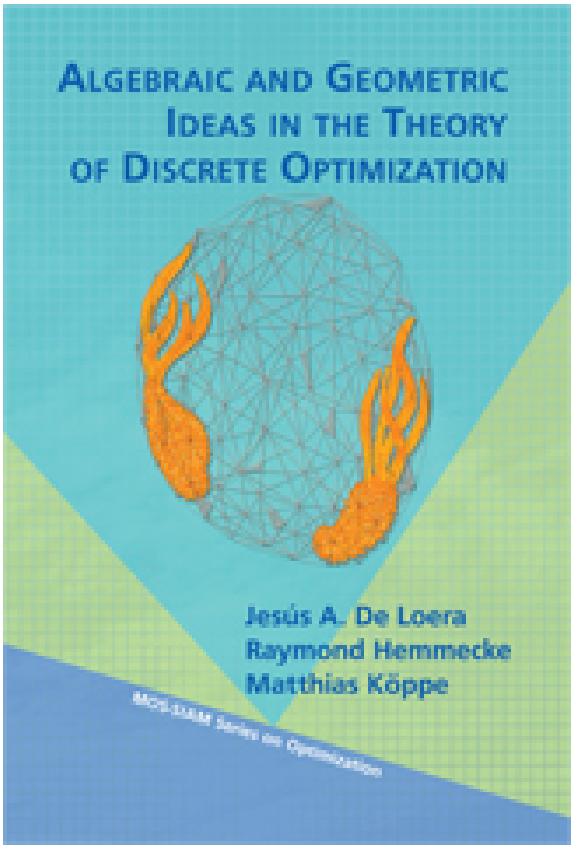
[9] S. Hoşten and B. Sturmfels, *GRIN: An implementation of Gröbner bases for integer programming*, Integer Programming and Combinatorial Optimization (E. Balas and J. Clausen, eds.), Lecture Notes in Computer Science, vol. 920, Springer Berlin Heidelberg, 1995, pp. 267–276, doi:10.1007/3-540-59408-6_57, ISBN 978-3-540-59408-6.

[10] M. Köppe, *Erzeugende Mengen für gemischt-ganzzahlige Programme*, Diploma thesis, Otto-von-Guericke-Universität Magdeburg, 1999, available from URL http://www.math.ucdavis.edu/~mkoeppe/art/mkoeppe-diplom.ps.

[11] P. N. Malkin, *Truncated Markov bases and Gröbner bases for integer programming*, 2006, arXiv:math/0612615.

[12] B. Sturmfels, *Algebraic recipes for integer programming*, 2003, arXiv:math.OC/0310194.

[13] R. Urbaniak, *Decomposition of generating sets and of integer programs*, Dissertation, Technische Universität Berlin, 1998.


ALGEBRAIC AND GEOMETRIC
IDEAS IN THE THEORY
OF DISCRETE OPTIMIZATION
Jesús A. De Loera
Raymond Hemmecke
Matthias Köppe
MOS-SIAM Series on Optimization