

Handbook of Magma Functions

Part XXII: Optimization

Version 2.30-1

Sydney

November 10, 2026

Editors

John Cannon *Wieb Bosma* *Claus Fieker* *Allan Steel*

Handbook Contributors

Eran Assaf *Geoff Bailey* *Wieb Bosma* *Gavin Brown* *Nils Bruin* *John Cannon* *Jon Carlson* *Scott Contini* *Edgar Costa* *Bruce Cox* *Brendan Creutz* *Steve Donnelly* *Tim Dokchitser* *Willem de Graaf* *Andreas-Stephan Elsenhans* *Claus Fieker* *Damien Fisher* *Volker Gebhardt* *Sergei Haller* *Michael Harrison* *Florian Hess* *Derek Holt* *David Howden* *Al Kasprzyk* *Markus Kirschmer* *David Kohel* *Axel Kohnert* *Dimitri Leemans* *Paulette Lieby* *Graham Matthews* *Scott Murray* *Eamonn O'Brien* *Dan Roozemond* *Ben Smith* *Bernd Souvignier* *William Stein* *Allan Steel* *Damien Stehlé* *Nicole Sutherland* *Don Taylor* *Bill Unger* *Alexa van der Waall* *Paul van Wamelen* *Helena Verrill* *John Voight* *Mark Watkins* *Greg White*

Production Editors

Wieb Bosma *Claus Fieker* *Allan Steel* *Nicole Sutherland*

HTML Production

Claus Fieker *Allan Steel*

Copyright © 1994–2026 The University of Sydney. All rights reserved.

Contents

173 Linear Programming **1**
- 173.1 Introduction . . . 1
- 173.2 Explicit LP Solving Functions . . . 2
- 173.3 Creation of LP objects . . . 3
- 173.4 Operations on LP objects . . . 3
- 173.5 Bibliography . . . 6

References **7**

Index **8**

Chapter 173

Linear Programming

173.1 Introduction

A Linear Program in n variables $x_1, \cdots, x_n$ with m constraints of the form $\sum_{j=1}^{n} a_j x_j \ \leq \ c$ (the relations in any of the constraints may also be $=$ or $\geq$) may be represented in matrix form as:

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \quad (REL) \quad \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}$$

where (REL) represents a componentwise relation between vectors, with each element $=$, $\leq$, or $\geq$.

Note that there is an additional implicit constraint, wherein all variables are assumed to be nonnegative.

We wish to find a solution (x_i) that maximises (or minimises) the objective function: $\sum_{i=1}^{n} o_i x_i$

Magma provides two methods for solving LP problems. The first is to set up suitable constraint matrices and then use an explicit LP solving function to solve the problem. The second involves creating an instance of the `LP` process, which is of category `LP`. Constraints are added and options set before calling `Solution` to get a solution to the problem.

All functions that actually solve an LP problem return a solution vector together with an integer code representing the state of the solution, provided by the lp_solve library. The codes are:

0 Optimal Solution

1 Failure

2 Infeasible problem

3 Unbounded problem

4 Failure

Magma supports LP problems over Integer, Rational, and Real rings. For Integer and Real problems, the solutions will be provided as Integer and Real vectors respectively. For LP problems provided in Rationals, the solution is a Real vector.

Linear programming in Magma is implemented using the lp_solve library written by Michel Berkelaar (michel@ics.ele.tue.nl). The library source may be found at `ftp://ftp.ics.ele.tue.nl/pub/lp_solve/`.

For further reference see [Naz87], [Chv83], [OH68], and [NW88].

173.2 Explicit LP Solving Functions

Each explicit LP solving function takes four arguments to represent an LP problem in n variables with m constraints:

1 `LHS` : $m \times n$ matrix, representing the left-hand-side coefficients of the m constraints.

2 `relations` : $m \times 1$ matrix over the same ring as `LHS`, representing the relations for each constraint, with a positive entry representing $\geq$, a zero entry representing $=$, and a negative entry representing $\leq$.

3 `RHS` : $m \times 1$ matrix over the same ring as `LHS`, representing the right-hand-side values of the m constraints.

4 `objective` : $1 \times n$ matrix over the same ring as `LHS`, representing the coefficients of the objective function to be optimised.

Each function returns a vector representing an optimal solution to the problem, and an integer indicating the state of the solution, as described in the introduction.

`MaximalSolution(LHS, relations, RHS, objective): Mtrx, Mtrx, Mtrx, Mtrx -> Mtrx, RngIntElt`

The vector maximising the LP problem, with an integer describing the state of the solution.

`MinimalSolution(LHS, relations, RHS, objective): Mtrx, Mtrx, Mtrx, Mtrx -> Mtrx, RngIntElt`

The vector minimising the LP problem, with an integer describing the state of the solution.

`MaximalIntegerSolution(LHS, relations, RHS, objective): Mtrx, Mtrx, Mtrx, Mtrx -> Mtrx, RngIntElt`

The integer vector maximising the LP problem, with an integer describing the state of the solution.

`MinimalIntegerSolution(LHS, relations, RHS, objective): Mtrx, Mtrx, Mtrx, Mtrx -> Mtrx, RngIntElt`

The integer vector minimising the LP problem, with an integer describing the state of the solution.

`MaximalZeroOneSolution(LHS, relations, RHS, objective): Mtrx, Mtrx, Mtrx, Mtrx -> Mtrx, RngIntElt`

The vector with each entry either zero or one maximising the LP problem, with an integer describing the state of the solution.

`MinimalZeroOneSolution(LHS, relations, RHS, objective): Mtrx, Mtrx, Mtrx, Mtrx -> Mtrx, RngIntElt`

The vector with each entry either zero or one minimising the LP problem, with an integer describing the state of the solution.

Example: Linear Program in Two Variables (ex-807d41)

We solve the LP maximising $F(x, y) = 8x+15y \quad x, y \in \mathbf{R}$ subject to the constraints $10x+21y \leq 156$ $2x + y \leq 22$

```
> R := RealField( );
> lhs := Matrix(R, 2, 2, [10, 21, 2, 1]);
> rhs := Matrix(R, 2, 1, [156, 22]);
> rel := Matrix(R, 2, 1, [-1, -1]); // negative values - less-or-equal relation
> obj := Matrix(R, 1, 2, [8, 15]);
> MaximalSolution(lhs, rel, rhs, obj);
[9.562500000000000000 2.875000000000000888]
0
```

Example: Linear Program in Seven Variables (ex-33e772)

We find solutions to the LP maximising $F(x_1, \cdots, x_7) = 592x_1+381x_2+273x_3+55x_4+48x_5+37x_6+23x_7$ subject to the constraint $3534x_1+2356x_2+2767x_3+589x_4+528x_5+451x_6+304x_7 \leq 119567$ with $(x_1, \cdots, x_7)$ taking real values, integer values, and zero/one values.

```
> R := RealField( );
> lhs := Matrix(R, 1, 7, [3534, 2356, 2767, 589, 528, 451, 304]);
> rhs := Matrix(R, 1, 1, [119567]);
> rel := Matrix(R, 1, 1, [-1]);
> obj := Matrix(R, 1, 7, [592, 381, 273, 55, 48, 37, 23]);
> MaximalSolution(lhs, rel, rhs, obj);
[33.83333333333333570 0.E-92 0.E-92 0.E-92 0.E-92 0.E-92 0.E-92]
0
> MaximalIntegerSolution(lhs, rel, rhs, obj);
[33.00000000000000000 1.00000000000000000 0.E-92 1.00000000000000000 0.E-92
    0.E-92 0.E-92]
0
> MaximalZeroOneSolution(lhs, rel, rhs, obj);
[1.00000000000000000 1.00000000000000000 1.00000000000000000
    1.00000000000000000 1.00000000000000000 1.00000000000000000
    1.00000000000000000]
0
```

173.3 Creation of LP objects

`LPProcess(R, n): Rng, RngIntElt -> LP`

A Linear Program over the ring R in n variables.

Example: Creating an LP Object (ex-a6b954)

We create an LP representing a problem in 2 real variables:

```
> R := RealField( );
> L := LPProcess(R, 2);
> L;
LP <Real Field, 2 variables>
Minimising objective function: [0 0]
Subject to constraints:
Variables bounded above by: [ ]
Variables bounded below by: [ ]
Solving in integers for variables [ ]
```

173.4 Operations on LP objects

`AddConstraints(L, lhs, rhs): LP, Mtrx, Mtrx`

`Rel: MonStgElt` Default: `"eq"`

Add some constraints to the LP problem L. All constraints will have the same relation, given by `Rel`, which may be set to `"eq"` for strict equality (the default), `"le"` for less-or-equal constraints, or `"ge"` for greater-or-equal constraints.

Constraints are of the form

$$\sum_{j=1}^{n} \texttt{lhs}_{ij} \quad \texttt{Rel} \quad \texttt{rhs}_{i1}$$

where *lhs* and *rhs* are described in Section *Explicit LP Solving Functions*.

`NumberOfConstraints(L): LP -> RngIntElt`

The number of constraints in the LP problem L.

`NumberOfVariables(L): LP -> RngIntElt`

The number of variables in the LP problem L.

`EvaluateAt(L, p): LP, Mtrx -> RngIntElt`

Evaluate the objective function of the LP problem L at the point p given by a matrix.

`Constraint(L, n): LP, RngIntElt -> Mtrx, Mtrx, RngIntElt`

The LHS, RHS, and relation (-1 for $\leq$, 0 for $=$, 1 for $\geq$) of the n-th constraint of the LP problem L.

`IntegerSolutionVariables(L): LP -> SeqEnum`

Sequence of indices of the variables in the LP problem L to be solved in integers.

`ObjectiveFunction(L): LP -> Mtrx`

The objective function of the LP problem L.

`IsMaximisingFunction(L): LP -> BoolElt`

Returns `true` if the LP problem L is set to maximise its objective function, `false` if set to minimise.

`RemoveConstraint(L, n): LP, RngIntElt`

Remove the n-th constraint from the LP problem L.

`SetIntegerSolutionVariables(L, I, m): LP, SeqEnum[RngIntElt], BoolElt`

Set the variables of the LP problem L indexed by elements of the sequence I to be solved in integers if m is `true`, or in the usual ring if `false`.

`SetLowerBound(L, n, b): LP, RngIntElt, RngElt`

Set the lower bound on the n-th variable in the LP problem L to b.

Note that for all LP problems in Magma there is an implicit constraint that all variables are ≥ 0. This constraint is overridden if a lower bound is specified by using this function (e.g., specifying a lower bound of -5 works as expected), but the lower bound can currently not be completely removed.

`SetMaximiseFunction(L, m): LP, BoolElt`

Set the LP problem L to maximise its objective function if m is `true`, or to minimise the objective function if m is false.

`SetObjectiveFunction(L, F): LP, Mtrx`

Set the objective function of the LP problem L to the matrix F.

`SetUpperBound(L, n, b): LP, RngIntElt, RngElt`

Set the upper bound on the n-th variable in the LP problem L to b.

`Solution(L): LP -> Mtrx, RngIntElt`

Solve the LP problem L; returns a point representing an optimal solution, and an integer representing the state of the solution.

`UnsetBounds(L): LP`

Remove any bounds on all variables in the LP problem L.

Note that this reactivates the implicit constraint that all variables are ≥ 0.

Example: Filling and Solving an LP Object `(ex-b5bf57)`

We use an LP object to solve the LP maximising $F(x, y) = 3x + 13y$ subject to constraints $2x + 9y <= 40$ $11x - 8y <= 82$

```
> R := RealField( );
> L := LPProcess(R, 2);
> SetObjectiveFunction(L, Matrix(R, 1, 2, [3,13]));
> lhs := Matrix(R, 2, 2, [2, 9, 11, -8]);
> rhs := Matrix(R, 2, 1, [40, 82]);
> AddConstraints(L, lhs, rhs : Rel := "le");
> SetMaximiseFunction(L, true);
> L;
LP <Real Field, 2 variables>
Maximising objective function: [ 3 13]
Subject to constraints:
1 : [2 9] <= [40]
2 : [11 -8] <= [82]
Variables bounded above by: [ ]
Variables bounded below by: [ ]
Solving in integers for variables [ ]
> Solution(L);
[9.199999999999999289 2.400000000000000355]
0
```

Now, we place some bounds on y:

```
> SetUpperBound(L, 2, R!2);
> SetLowerBound(L, 2, R!1);
> Solution(L);
[8.909090909090908283 2.000000000000000000]
0
```

And find integer solutions:

```
> SetIntegerSolutionVariables(L, [1,2], true);
> Solution(L);
[8.000000000000000000 2.000000000000000000]
0
```

Now, removing the 2nd constraint:

```
> RemoveConstraint(L, 2);
> L;
LP <Real Field, 2 variables>
Maximising objective function: [ 3 13]
Subject to constraints:
1 : [2 9] <= [40]
Variables bounded above by: [ 2:2 ]
Variables bounded below by: [ 2:1 ]
Solving in integers for variables [ 1, 2 ]
> Solution(L);
[11.000000000000000000 2.000000000000000000]
0
```

And removing the restriction to Integer values for y,

```
> SetIntegerSolutionVariables(L, [2], false);
> Solution(L);
[15.000000000000000000 1.111111111111111160]
0
```

173.5 Bibliography

- [Chv83] V. Chvatal. *Linear Programming*. W.H. Freeman and Company, 1983.
- [Naz87] John Lawrence Nazareth. *Computer Solution of Linear Programs*. Oxford University Press, 1987.
- [NW88] G.L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc., 1988.
- [OH68] W. Orchard–Hays. *Advanced linear–programming computing techniques*. McGraw–Hill, 1968.

References

[Chv83] V. Chvatal. *Linear Programming*. W.H. Freeman and Company, 1983.

[Naz87] John Lawrence Nazareth. *Computer Solution of Linear Programs*. Oxford University Press, 1987.

[NW88] G.L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc., 1988.

[OH68] W. Orchard–Hays. *Advanced linear–programming computing techniques*. McGraw–Hill, 1968.

Index

A

`AddConstraints(L, lhs, rhs)` (*procedure*), 3

C

`Constraint(L, n)` (*function*), 4
`Creating an LP Object` (*example*), 3

E

`EvaluateAt(L, p)` (*function*), 4

F

`Filling and Solving an LP Object` (*example*), 4

I

`IntegerSolutionVariables(L)` (*function*), 4
`IsMaximisingFunction(L)` (*function*), 4

L

`Linear Program in Seven Variables` (*example*), 2
`Linear Program in Two Variables` (*example*), 2
`LPProcess(R, n)` (*function*), 3

M

`MaximalIntegerSolution(LHS, relations, RHS, objective)` (*function*), 2
`MaximalSolution(LHS, relations, RHS, objective)` (*function*), 2
`MaximalZeroOneSolution(LHS, relations, RHS, objective)` (*function*), 2
`MinimalIntegerSolution(LHS, relations, RHS, objective)` (*function*), 2
`MinimalSolution(LHS, relations, RHS, objective)` (*function*), 2
`MinimalZeroOneSolution(LHS, relations, RHS, objective)` (*function*), 2

N

`NumberOfConstraints(L)` (*function*), 3
`NumberOfVariables(L)` (*function*), 4

O

`ObjectiveFunction(L)` (*function*), 4

R

`RemoveConstraint(L, n)` (*procedure*), 4

S

`SetIntegerSolutionVariables(L, I, m)` (*procedure*), 4
`SetLowerBound(L, n, b)` (*procedure*), 4
`SetMaximiseFunction(L, m)` (*procedure*), 4
`SetObjectiveFunction(L, F)` (*procedure*), 4
`SetUpperBound(L, n, b)` (*procedure*), 4
`Solution(L)` (*function*), 4

U

`UnsetBounds(L)` (*procedure*), 4