linprog

Solve linear programming problems

Syntax

```
x = linprog(f,A,b)
x = linprog(f,A,b,Aeq,beq)
x = linprog(f,A,b,Aeq,beq,lb,ub)
x = linprog(f,A,b,Aeq,beq,lb,ub,options)
x = linprog(problem)
[x,fval] = linprog( __ )
[x,fval,exitflag,output] = linprog( __ )
[x,fval,exitflag,output,lambda] = linprog( __ )
```

Description

Linear programming solver

Finds the minimum of a problem specified by

$$\min_x f^T x \text{ such that } \begin{cases} A \cdot x \le b, \\ Aeq \cdot x = beq, \\ lb \le x \le ub. \end{cases}$$

f, x, b, beq, lb, and ub are vectors, and A and Aeq are matrices.

> **Note**
>
> `linprog` applies only to the solver-based approach. For a discussion of the two optimization approaches, see First Choose Problem-Based or Solver-Based Approach.

`x = linprog(f,A,b)` solves min `f'*x` such that `A*x ≤ b`. example

`x = linprog(f,A,b,Aeq,beq)` includes equality constraints `Aeq*x = beq`. Set `A = []` and `b = []` if no inequalities exist. example

`x = linprog(f,A,b,Aeq,beq,lb,ub)` defines a set of lower and upper bounds on the design variables, `x`, so that the solution is always in the range `lb ≤ x ≤ ub`. Set `Aeq = []` and `beq = []` if no equalities exist. example

> **Note**
>
> If the specified input bounds for a problem are inconsistent, the output `fval` is `[]`.

`x = linprog(f,A,b,Aeq,beq,lb,ub,options)` minimizes with the optimization options specified by `options`. Use `optimoptions` to set these options. example

`x = linprog(problem)` finds the minimum for `problem`, a structure described in `problem`. example

You can import a `problem` structure from an MPS file using `mpsread`. You can also create a `problem` structure from an `OptimizationProblem` object by using `prob2struct`.

`[x,fval] = linprog( __ )`, for any input arguments, returns the value of the objective function `fun` at the solution `x`: `fval = f'*x`. example

`[x,fval,exitflag,output] = linprog( __ )` additionally returns a value `exitflag` that describes the exit condition, and a structure `output` that contains information about the optimization process. example

`[x,fval,exitflag,output,lambda] = linprog( __ )` additionally returns a structure `lambda` whose fields contain the Lagrange multipliers at the solution `x`. example

Examples

collapse all

Linear Program, Linear Inequality Constraints

Solve a simple linear program defined by linear inequalities.

For this example, use these linear inequality constraints:

$$x(1) + x(2) \le 2$$

$$x(1) + x(2)/4 \le 1$$

$$x(1) - x(2) \le 2$$

$$-x(1)/4 - x(2) \le 1$$

$$-x(1) - x(2) \le -1$$

$$-x(1) + x(2) \le 2.$$

Try This Example

View MATLAB Command

```
A = [1 1
    1 1/4
    1 -1
    -1/4 -1
    -1 -1
    -1 1];

b = [2 1 2 1 -1 2];
```

Use the objective function $-x(1) - x(2)/3$.

```
f = [-1 -1/3];
```

Solve the linear program.

```
x = linprog(f,A,b)
```

```
Optimal solution found.
x = 2×1

    0.6667
    1.3333
```

Linear Program with Linear Inequalities and Equalities

Solve a simple linear program defined by linear inequalities and linear equalities.

For this example, use these linear inequality constraints:

Try This Example

View MATLAB Command

$$x(1) + x(2) \le 2$$

$$x(1) + x(2)/4 \le 1$$

$$x(1) - x(2) \le 2$$

$$-x(1)/4 - x(2) \le 1$$

$$-x(1) - x(2) \le -1$$

$$-x(1) + x(2) \le 2.$$

```
A = [1 1
    1 1/4
    1 -1
    -1/4 -1
    -1 -1
    -1 1];

b = [2 1 2 1 -1 2];
```

Use the linear equality constraint $x(1) + x(2)/4 = 1/2$.

```
Aeq = [1 1/4];
beq = 1/2;
```

Use the objective function $-x(1) - x(2)/3$.

```
f = [-1 -1/3];
```

Solve the linear program.

```
x = linprog(f,A,b,Aeq,beq)
```

```
Optimal solution found.
x = 2×1

     0
     2
```

Linear Program with All Constraint Types

Solve a simple linear program with linear inequalities, linear equalities, and bounds.

For this example, use these linear inequality constraints:

Try This Example

View MATLAB Command

$$x(1) + x(2) \le 2$$

$$x(1) + x(2)/4 \le 1$$

$$x(1) - x(2) \le 2$$

$$-x(1)/4 - x(2) \le 1$$

$$-x(1) - x(2) \le -1$$

$$-x(1) + x(2) \le 2.$$

```
A = [1 1
    1 1/4
    1 -1
    -1/4 -1
    -1 -1
    -1 1];

b = [2 1 2 1 -1 2];
```

Use the linear equality constraint $x(1) + x(2)/4 = 1/2$.

```
Aeq = [1 1/4];
beq = 1/2;
```

Set these bounds:

$$-1 \le x(1) \le 1.5$$

$$-0.5 \le x(2) \le 1.25.$$

```
lb = [-1,-0.5];
ub = [1.5,1.25];
```

Use the objective function $-x(1) - x(2)/3$.

```
f = [-1 -1/3];
```

Solve the linear program.

```
x = linprog(f,A,b,Aeq,beq,lb,ub)
```

```
Optimal solution found.
x = 2×1

    0.1875
    1.2500
```

Linear Program Using the `'interior-point'` Algorithm

Solve a linear program using the `'interior-point'` algorithm.

Try This Example

View MATLAB Command

For this example, use these linear inequality constraints:

$$x(1) + x(2) \le 2$$

$$x(1) + x(2)/4 \le 1$$

$$x(1) - x(2) \le 2$$

$$-x(1)/4 - x(2) \le 1$$

$$-x(1) - x(2) \le -1$$

$$-x(1) + x(2) \le 2.$$

```
A = [1 1
    1 1/4
    1 -1
    -1/4 -1
    -1 -1
    -1 1];

b = [2 1 2 1 -1 2];
```

Use the linear equality constraint $x(1) + x(2)/4 = 1/2$.

```
Aeq = [1 1/4];
beq = 1/2;
```

Set these bounds:

$$-1 \le x(1) \le 1.5$$

$$-0.5 \le x(2) \le 1.25.$$

```
lb = [-1,-0.5];
ub = [1.5,1.25];
```

Use the objective function $-x(1) - x(2)/3$.

```
f = [-1 -1/3];
```

Set options to use the `'interior-point'` algorithm.

```
options = optimoptions('linprog','Algorithm','interior-point');
```

Solve the linear program using the `'interior-point'` algorithm.

```
x = linprog(f,A,b,Aeq,beq,lb,ub,options)
```

```
Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the selected value of the function tolerance,
and constraints are satisfied to within the selected value of the constraint
tolerance.
x = 2×1

    0.1875
    1.2500
```

Solve LP Using Problem-Based Approach for `linprog`

This example shows how to set up a problem using the problem-based approach and then solve it using the solver-based approach. The problem is

Try This Example

View MATLAB Command

$$\max_x (x + y/3) \text{ subject to } \begin{cases} x + y \le 2 \\ x + y/4 \le 1 \\ x - y \le 2 \\ x/4 + y \ge -1 \\ x + y \ge 1 \\ -x + y \le 2 \\ x + y/4 = 1/2 \\ -1 \le x \le 1.5 \\ -1/2 \le y \le 1.25 \end{cases}$$

Create an `OptimizationProblem` object named `prob` to represent this problem.

```
x = optimvar('x','LowerBound',-1,'UpperBound',1.5);
y = optimvar('y','LowerBound',-1/2,'UpperBound',1.25);
prob = optimproblem('Objective',x + y/3,'ObjectiveSense','max');
prob.Constraints.c1 = x + y <= 2;
prob.Constraints.c2 = x + y/4 <= 1;
prob.Constraints.c3 = x - y <= 2;
prob.Constraints.c4 = x/4 + y >= -1;
prob.Constraints.c5 = x + y >= 1;
prob.Constraints.c6 = -x + y <= 2;
prob.Constraints.c7 = x + y/4 == 1/2;
```

Convert the problem object to a problem structure.

```
problem = prob2struct(prob);
```

Solve the resulting problem structure.

```
[sol,fval,exitflag,output] = linprog(problem)
```

```
Optimal solution found.
sol = 2×1

    0.1875
    1.2500

fval = -0.6042
exitflag = 1
output = struct with fields:
         iterations: 0
    constrviolation: 0
            message: 'Optimal solution found.'
          algorithm: 'dual-simplex'
      firstorderopt: 0
```

The returned `fval` is negative, even though the solution components are positive. Internally, `prob2struct` turns the maximization problem into a minimization problem of the negative of the objective function. See Maximizing an Objective.

Which component of `sol` corresponds to which optimization variable? Examine the `Variables` property of `prob`.

```
prob.Variables
```

```
ans = struct with fields:
```

```
    x: [1x1 optim.problemdef.OptimizationVariable]
    y: [1x1 optim.problemdef.OptimizationVariable]
```

As you might expect, `sol(1)` corresponds to `x`, and `sol(2)` corresponds to `y`. See Algorithms.

Return the Objective Function Value

Calculate the solution and objective function value for a simple linear program.

Try This Example

View MATLAB Command

The inequality constraints are

$$x(1) + x(2) \le 2$$

$$x(1) + x(2)/4 \le 1$$

$$x(1) - x(2) \le 2$$

$$-x(1)/4 - x(2) \le 1$$

$$-x(1) - x(2) \le -1$$

$$-x(1) + x(2) \le 2.$$

```
A = [1 1
    1 1/4
    1 -1
    -1/4 -1
    -1 -1
    -1 1];

b = [2 1 2 1 -1 2];
```

The objective function is $-x(1) - x(2)/3$.

```
f = [-1 -1/3];
```

Solve the problem and return the objective function value.

```
[x,fval] = linprog(f,A,b)
```

```
Optimal solution found.
x = 2×1

    0.6667
    1.3333

fval = -1.1111
```

Obtain More Output to Examine the Solution Process

Obtain the exit flag and output structure to better understand the solution process and quality.

Try This Example

For this example, use these linear inequality constraints:

View MATLAB Command

$$x(1) + x(2) \le 2$$

$$x(1) + x(2)/4 \le 1$$

$$x(1) - x(2) \le 2$$

$$-x(1)/4 - x(2) \le 1$$

$$-x(1) - x(2) \le -1$$

$$-x(1) + x(2) \le 2.$$

```
A = [1 1
    1 1/4
    1 -1
    -1/4 -1
    -1 -1
    -1 1];

b = [2 1 2 1 -1 2];
```

Use the linear equality constraint $x(1) + x(2)/4 = 1/2$.

```
Aeq = [1 1/4];
beq = 1/2;
```

Set these bounds:

$$-1 \le x(1) \le 1.5$$

$$-0.5 \le x(2) \le 1.25.$$

```
lb = [-1,-0.5];
ub = [1.5,1.25];
```

Use the objective function $-x(1) - x(2)/3$.

```
f = [-1 -1/3];
```

Set options to use the `'dual-simplex'` algorithm.

```
options = optimoptions('linprog','Algorithm','dual-simplex');
```

Solve the linear program and request the function value, exit flag, and output structure.

```
[x,fval,exitflag,output] = linprog(f,A,b,Aeq,beq,lb,ub,options)
```

```
Optimal solution found.
x = 2×1

    0.1875
    1.2500

fval = -0.6042
exitflag = 1
output = struct with fields:
         iterations: 0
    constrviolation: 0
```

```
            message: 'Optimal solution found.'
          algorithm: 'dual-simplex'
      firstorderopt: 0
```

- `fval`, the objective function value, is larger than Return the Objective Function Value, because there are more constraints.
- `exitflag` = 1 indicates that the solution is reliable.
- `output.iterations` = 0 indicates that `linprog` found the solution during presolve, and did not have to iterate at all.

Obtain Solution and Lagrange Multipliers

Solve a simple linear program and examine the solution and the Lagrange multipliers.

Try This Example

View MATLAB Command

Use the objective function

$$f(x) = -5x_1 - 4x_2 - 6x_3.$$

```
f = [-5; -4; -6];
```

Use the linear inequality constraints

$$x_1 - x_2 + x_3 \le 20$$

$$3x_1 + 2x_2 + 4x_3 \le 42$$

$$3x_1 + 2x_2 \le 30.$$

```
A =  [1 -1  1
      3  2  4
      3  2  0];
b = [20;42;30];
```

Constrain all variables to be positive:

$$x_1 \ge 0$$

$$x_2 \ge 0$$

$$x_3 \ge 0.$$

```
lb = zeros(3,1);
```

Set `Aeq` and `beq` to `[]`, indicating that there are no linear equality constraints.

```
Aeq = [];
beq = [];
```

Call `linprog`, obtaining the Lagrange multipliers.

```
[x,fval,exitflag,output,lambda] = linprog(f,A,b,Aeq,beq,lb);
```

```
 Optimal solution found.
```

Examine the solution and Lagrange multipliers.

```
x,lambda.ineqlin,lambda.lower
```

```
x = 3×1

         0
   15.0000
    3.0000

ans = 3×1

         0
    1.5000
    0.5000

ans = 3×1

    1.0000
         0
         0
```

`lambda.ineqlin` is nonzero for the second and third components of `x`. This indicates that the second and third linear inequality constraints are satisfied with equalities:

$$3x_1 + 2x_2 + 4x_3 = 42$$

$$3x_1 + 2x_2 = 30.$$

Check that this is true:

```
A*x
```

```
ans = 3×1

  -12.0000
   42.0000
   30.0000
```

`lambda.lower` is nonzero for the first component of `x`. This indicates that `x(1)` is at its lower bound of 0.

Input Arguments

collapse all

`f` — Coefficient vector

real vector | real array

Coefficient vector, specified as a real vector or real array. The coefficient vector represents the objective function `f'*x`. The notation assumes that `f` is a column vector, but you can use a row vector or array. Internally, `linprog` converts `f` to the column vector `f(:)`.

Example: `f = [1,3,5,-6]`

Data Types: `double`

`A` — Linear inequality constraints
real matrix

Linear inequality constraints, specified as a real matrix. `A` is an `M`-by-`N` matrix, where `M` is the number of inequalities, and `N` is the number of variables (length of `f`). For large problems, pass `A` as a sparse matrix.

`A` encodes the `M` linear inequalities

```
A*x <= b,
```

where `x` is the column vector of `N` variables `x(:)`, and `b` is a column vector with `M` elements.

For example, consider these inequalities:

$$x_1 + 2x_2 \le 10$$
$$3x_1 + 4x_2 \le 20$$
$$5x_1 + 6x_2 \le 30.$$

Specify the inequalities by entering the following constraints.

```
A = [1,2;3,4;5,6];
b = [10;20;30];
```

Example: To specify that the x-components add up to 1 or less, take `A = ones(1,N)` and `b = 1`.

Data Types: `double`

`Aeq` — Linear equality constraints
real matrix

Linear equality constraints, specified as a real matrix. `Aeq` is an `Me`-by-`N` matrix, where `Me` is the number of equalities, and `N` is the number of variables (length of `f`). For large problems, pass `Aeq` as a sparse matrix.

`Aeq` encodes the `Me` linear equalities

```
Aeq*x = beq,
```

where `x` is the column vector of `N` variables `x(:)`, and `beq` is a column vector with `Me` elements.

For example, consider these equalities:

$$x_1 + 2x_2 + 3x_3 = 10$$
$$2x_1 + 4x_2 + x_3 = 20.$$

Specify the equalities by entering the following constraints.

```
Aeq = [1,2,3;2,4,1];
beq = [10;20];
```

Example: To specify that the x-components sum to 1, take `Aeq = ones(1,N)` and `beq = 1`.

Data Types: `double`

`b` — Linear inequality constraints
real vector

Linear inequality constraints, specified as a real vector. `b` is an `M`-element vector related to the `A` matrix. If you pass `b` as a row vector, solvers internally convert `b` to the column vector `b(:)`. For large problems, pass `b` as a sparse vector.

`b` encodes the `M` linear inequalities

```
A*x <= b,
```

where `x` is the column vector of `N` variables `x(:)`, and `A` is a matrix of size `M`-by-`N`.

For example, consider these inequalities:

$x_1 + 2x_2 \le 10$
$3x_1 + 4x_2 \le 20$
$5x_1 + 6x_2 \le 30$.

Specify the inequalities by entering the following constraints.

```
A = [1,2;3,4;5,6];
b = [10;20;30];
```

Example: To specify that the `x` components sum to 1 or less, use `A = ones(1,N)` and `b = 1`.

Data Types: `double`

`beq` — Linear equality constraints
real vector

Linear equality constraints, specified as a real vector. `beq` is an `Me`-element vector related to the `Aeq` matrix. If you pass `beq` as a row vector, solvers internally convert `beq` to the column vector `beq(:)`. For large problems, pass `beq` as a sparse vector.

`beq` encodes the `Me` linear equalities

```
Aeq*x = beq,
```

where `x` is the column vector of `N` variables `x(:)`, and `Aeq` is a matrix of size `Me`-by-`N`.

For example, consider these equalities:

$x_1 + 2x_2 + 3x_3 = 10$
$2x_1 + 4x_2 + x_3 = 20$.

Specify the equalities by entering the following constraints.

```
Aeq = [1,2,3;2,4,1];
beq = [10;20];
```

Example: To specify that the `x` components sum to 1, use `Aeq = ones(1,N)` and `beq = 1`.

Data Types: `double`

`lb` — Lower bounds
real vector | real array

Lower bounds, specified as a real vector or real array. If the length of `f` is equal to the length of `lb`, then `lb` specifies that

```
x(i) >= lb(i) for all i.
```

If `numel(lb) < numel(f)`, then `lb` specifies that

`x(i) >= lb(i)` for `1 <= i <= numel(lb)`.

In this case, solvers issue a warning.

Example: To specify that all x-components are positive, use `lb = zeros(size(f))`.

Data Types: `double`

`ub` — Upper bounds
real vector | real array

Upper bounds, specified as a real vector or real array. If the length of `f` is equal to the length of `ub`, then `ub` specifies that

`x(i) <= ub(i)` for all `i`.

If `numel(ub) < numel(f)`, then `ub` specifies that

`x(i) <= ub(i)` for `1 <= i <= numel(ub)`.

In this case, solvers issue a warning.

Example: To specify that all x-components are less than `1`, use `ub = ones(size(f))`.

Data Types: `double`

`options` — Optimization options
output of `optimoptions` | structure as `optimset` returns

Optimization options, specified as the output of `optimoptions` or a structure as `optimset` returns.

Some options apply to all algorithms, and others are relevant for particular algorithms. See Optimization Options Reference for detailed information.

Some options are absent from the `optimoptions` display. These options appear in italics in the following table. For details, see View Options.

All Algorithms	
`Algorithm`	Choose the optimization algorithm: • `'dual-simplex'` (default) • `'interior-point-legacy'` • `'interior-point'` For information on choosing the algorithm, see Linear Programming Algorithms.
Diagnostics	Display diagnostic information about the function to be minimized or solved. Choose `'off'` (default) or `'on'`.
`Display`	Level of display (see Iterative Display): • `'final'` (default) displays just the final output. • `'off'` or `'none'` displays no output. • `'iter'` displays output at each iteration.

`MaxIterations`	Maximum number of iterations allowed, a positive integer. The default is: • `85` for the `'interior-point-legacy'` algorithm • `200` for the `'interior-point'` algorithm • `10*(numberOfEqualities + numberOfInequalities + numberOfVariables)` for the `'dual-simplex'` algorithm See Tolerances and Stopping Criteria and Iterations and Function Counts. For `optimset`, the name is `MaxIter`. See Current and Legacy Option Names.
`OptimalityTolerance`	Termination tolerance on the dual feasibility, a positive scalar. The default is: • `1e-8` for the `'interior-point-legacy'` algorithm • `1e-7` for the `'dual-simplex'` algorithm • `1e-6` for the `'interior-point'` algorithm For `optimset`, the name is `TolFun`. See Current and Legacy Option Names.
interior-point Algorithm	
`ConstraintTolerance`	Feasibility tolerance for constraints, a scalar from `1e-10` through `1e-3`. `ConstraintTolerance` measures primal feasibility tolerance. The default is `1e-6`. For `optimset`, the name is `TolCon`. See Current and Legacy Option Names.
Preprocess	Level of LP preprocessing prior to algorithm iterations. Specify `'basic'` (default) or `'none'`.
Dual-Simplex Algorithm	
`ConstraintTolerance`	Feasibility tolerance for constraints, a scalar from `1e-10` through `1e-3`. `ConstraintTolerance` measures primal feasibility tolerance. The default is `1e-4`. For `optimset`, the name is `TolCon`. See Current and Legacy Option Names.
`MaxTime`	Maximum amount of time in seconds that the algorithm runs. The default is `Inf`.
Preprocess	Level of LP preprocessing prior to dual simplex algorithm iterations. Specify `'basic'` (default) or `'none'`.

Example: `options = optimoptions('linprog','Algorithm','interior-point','Display','iter')`

`problem` — Problem structure

structure

Problem structure, specified as a structure with the following fields.

Field Name	Entry
`f`	Linear objective function vector `f`
`Aineq`	Matrix for linear inequality constraints
`bineq`	Vector for linear inequality constraints
`Aeq`	Matrix for linear equality constraints
`beq`	Vector for linear equality constraints
`lb`	Vector of lower bounds
`ub`	Vector of upper bounds
`solver`	`'linprog'`
`options`	Options created with `optimoptions`

You must supply at least the `solver` field in the `problem` structure.

Data Types: `struct`

Output Arguments

collapse all

`x` — Solution

real vector | real array

Solution, returned as a real vector or real array. The size of `x` is the same as the size of `f`.

`fval` — Objective function value at the solution

real number

Objective function value at the solution, returned as a real number. Generally, `fval` = `f'*x`.

`exitflag` — Reason `linprog` stopped

integer

Reason `linprog` stopped, returned as an integer.

`3`	The solution is feasible with respect to the relative `ConstraintTolerance` tolerance, but is not feasible with respect to the absolute tolerance.
`1`	Function converged to a solution `x`.
`0`	Number of iterations exceeded `options.MaxIterations` or solution time in seconds exceeded `options.MaxTime`.
`-2`	No feasible point was found.
`-3`	Problem is unbounded.
`-4`	`NaN` value was encountered during execution of the algorithm.
`-5`	Both primal and dual problems are infeasible.
`-7`	Search direction became too small. No further progress could be made.
`-9`	Solver lost feasibility.

Exitflags `3` and `-9` relate to solutions that have large infeasibilities. These usually arise from linear constraint matrices that have large condition number, or problems that have large solution components. To correct these issues, try to scale the coefficient matrices, eliminate redundant linear constraints, or give tighter bounds on the variables.

`output` — Information about the optimization process

structure

Information about the optimization process, returned as a structure with these fields.

`iterations`	Number of iterations

`algorithm`	Optimization algorithm used
`cgiterations`	0 (interior-point algorithm only, included for backward compatibility)
`message`	Exit message
`constrviolation`	Maximum of constraint functions
`firstorderopt`	First-order optimality measure

`lambda` — Lagrange multipliers at the solution
structure

Lagrange multipliers at the solution, returned as a structure with these fields.

`lower`	Lower bounds corresponding to `lb`
`upper`	Upper bounds corresponding to `ub`
`ineqlin`	Linear inequalities corresponding to `A` and `b`
`eqlin`	Linear equalities corresponding to `Aeq` and `beq`

The Lagrange multipliers for linear constraints satisfy this equation with `length(f)` components:

$$\mathrm{f} + \mathrm{A}^T\lambda_{\mathrm{ineqlin}} + \mathrm{Aeq}^T\lambda_{\mathrm{eqlin}} + \lambda_{\mathrm{upper}} - \lambda_{\mathrm{lower}} = 0,$$

based on the Lagrangian

$$\mathrm{f}^T x + \lambda_{\mathrm{ineqlin}}^T(\mathrm{A}x - \mathrm{b}) + \lambda_{\mathrm{eqlin}}^T(\mathrm{Aeq}\,x - \mathrm{beq}) + \lambda_{\mathrm{upper}}^T(x - \mathrm{ub}) + \lambda_{\mathrm{lower}}^T(\mathrm{lb} - x).$$

This sign convention matches that of nonlinear solvers (see Constrained Optimality Theory). However, this sign is the opposite of the sign in much linear programming literature, so a `linprog` Lagrange multiplier is the negative of the associated "shadow price."

Algorithms

collapse all

Dual-Simplex Algorithm

For a description, see Dual-Simplex Algorithm.

Interior-Point-Legacy Algorithm

The `'interior-point-legacy'` method is based on LIPSOL (Linear Interior Point Solver, [3]), which is a variant of Mehrotra's predictor-corrector algorithm [2], a primal-dual interior-point method. A number of preprocessing steps occur before the algorithm begins to iterate. See Interior-Point-Legacy Linear Programming.

The first stage of the algorithm might involve some preprocessing of the constraints (see Interior-Point-Legacy Linear Programming). Several conditions might cause `linprog` to exit with an infeasibility message. In each case, `linprog` returns a negative `exitflag`, indicating to indicate failure.

- If a row of all zeros is detected in `Aeq`, but the corresponding element of `beq` is not zero, then the exit message is

  ```
  Exiting due to infeasibility: An all-zero row in the
  constraint matrix does not have a zero in corresponding
  right-hand-side entry.
  ```

- If one of the elements of `x` is found not to be bounded below, then the exit message is

  ```
  Exiting due to infeasibility: Objective f'*x is unbounded below.
  ```

- If one of the rows of `Aeq` has only one nonzero element, then the associated value in `x` is called a *singleton* variable. In this case, the value of that component of `x` can be computed from `Aeq` and `beq`. If the value computed violates another constraint, then the exit message is

  ```
  Exiting due to infeasibility: Singleton variables in
  equality constraints are not feasible.
  ```

- If the singleton variable can be solved for, but the solution violates the upper or lower bounds, then the exit message is

  ```
  Exiting due to infeasibility: Singleton variables in
  the equality constraints are not within bounds.
  ```

> **Note**
>
> The preprocessing steps are cumulative. For example, even if your constraint matrix does not have a row of all zeros to begin with, other preprocessing steps can cause such a row to occur.

When the preprocessing finishes, the iterative part of the algorithm begins until the stopping criteria are met. (For more information about residuals, the primal problem, the dual problem, and the related stopping criteria, see Interior-Point-Legacy Linear Programming.) If the residuals are growing instead of getting smaller, or the residuals are neither growing nor shrinking, one of the two following termination messages is displayed, respectively,

```
One or more of the residuals, duality gap, or total relative error
has grown 100000 times greater than its minimum value so far:
```

or

```
One or more of the residuals, duality gap, or total relative error
has stalled:
```

After one of these messages is displayed, it is followed by one of the following messages indicating that the dual, the primal, or both appear to be infeasible.

- `The dual appears to be infeasible (and the primal unbounded). (The primal residual < OptimalityTolerance.)`
- `The primal appears to be infeasible (and the dual unbounded). (The dual residual < OptimalityTolerance.)`
- `The dual appears to be infeasible (and the primal unbounded) since the dual residual > sqrt(OptimalityTolerance). (The primal residual < 10*OptimalityTolerance.)`
- `The primal appears to be infeasible (and the dual unbounded) since the primal residual > sqrt(OptimalityTolerance). (The dual residual < 10*OptimalityTolerance.)`
- `The dual appears to be infeasible and the primal unbounded since the primal objective < -1e+10 and the dual objective < 1e+6.`
- `The primal appears to be infeasible and the dual unbounded since the dual objective > 1e+10 and the primal objective > -1e+6.`
- `Both the primal and the dual appear to be infeasible.`

For example, the primal (objective) can be unbounded and the primal residual, which is a measure of primal constraint satisfaction, can be small.

Interior-Point Algorithm

The `'interior-point'` algorithm is similar to `'interior-point-legacy'`, but with a more efficient factorization routine, and with different preprocessing. See Interior-Point linprog Algorithm.

Alternative Functionality

App

The **Optimize** Live Editor task provides a visual interface for `linprog`.

References

[1] Dantzig, G.B., A. Orden, and P. Wolfe. “Generalized Simplex Method for Minimizing a Linear Form Under Linear Inequality Restraints.” *Pacific Journal Math.,* Vol. 5, 1955, pp. 183–195.

[2] Mehrotra, S. “On the Implementation of a Primal-Dual Interior Point Method.” *SIAM Journal on Optimization*, Vol. 2, 1992, pp. 575–601.

[3] Zhang, Y., “Solving Large-Scale Linear Programs by Interior-Point Methods Under the MATLAB Environment.” *Technical Report TR96-01*, Department of Mathematics and Statistics, University of Maryland, Baltimore County, Baltimore, MD, July 1995.

See Also

`intlinprog` | `mpsread` | Optimize | `optimoptions` | `prob2struct` | `quadprog`

Topics

Set Up a Linear Program, Solver-Based

Typical Linear Programming Problem

Maximize Long-Term Investments Using Linear Programming: Solver-Based

Solver-Based Optimization Problem Setup

Linear Programming Algorithms

Introduced before R2006a