`linopt` — library for linear optimization

Table of contents

Introduction

The `linopt` library provides algorithms for linear and integer programming. The routines in this library can be used in order to minimize and maximize a linear function subject to the set of linear constraints. It is possible to get only integer solutions. The routines for linear optimization are based on the two phase simplex algorithm. The algorithm of Land-Doig is used to find integer solutions.

The library functions are called using the library name `linopt` and the name of the function. E.g., use

```
>> c := [{3*x + 4*y - 3*z <= 23, 5*x - 4*y - 3*z <= 10,
          7*x + 4*y + 11*z <= 30}, - x + y + 2*z, {x, y, z}]:
   linopt::maximize(c);

            [OPTIMAL, {x = 0, z = 1/2, y = 49/8}, 57/8]
```

to solve the linear optimization problem defined in the variable `c`. This mechanism avoids naming conflicts with other library functions. If this is found to be inconvenient, then the routines of the `linopt` package may be exported via `export`. E.g., after calling

```
>> export(linopt, maximize):
```

the function `linopt::maximize` may be called directly:

```
>> c := [{3*x + 4*y - 3*z <= 23, 5*x - 4*y - 3*z <= 10,
          7*x + 4*y + 11*z <= 30}, - x + y + 2*z, {x, y, z}]:
   maximize(c);

            [OPTIMAL, {x = 0, z = 1/2, y = 49/8}, 57/8]
```

All routines of the `linopt` package are exported simultaneously by

```
>> export(linopt):
```

The functions available in the `linopt` library can be listed with:

```
>> info(linopt)

 Library 'linopt': a package for linear optimization

 -- Interface:
 linopt::Transparent, linopt::corners,
 linopt::dual,        linopt::feasible,
 linopt::maximize,    linopt::minimize,
 linopt::plot_data,   linopt::standardForm
```

`linopt::corners` – return the feasible corners of a linear program

`linopt::corners([constr, obj], vars)` returns all valid corners of the linear program.

`linopt::corners([constr, obj], vars, All)` returns all corners of the linear program.

Call(s):

- `linopt::corners([constr, obj], vars<, All><, Logic>)`
- `linopt::corners([constr, obj<, NonNegative><, seti>], vars<, All><, Logic>)`
- `linopt::corners([constr, obj<, NonNegative><, All>], vars<, All><, Logic>)`
- `linopt::corners([constr, obj<, setn><, seti>], vars<, All><, Logic>)`
- `linopt::corners([constr, obj<, setn><, All>], vars<, All><, Logic>)`

Parameters:

`constr`	—	a set or list of linear constraints
`obj`	—	a linear expression
`seti`	—	a set which contains identifiers interpreted as indeterminants
`setn`	—	a set which contains identifiers interpreted as indeterminants
`vars`	—	a list containing the variables of the linear program described by `constr` and `obj` and the existing options

Options:

`All`	—	This option can appear at two different places in the call of `linopt::corners`. If it is part of the linear program it means that all variables are constrained to be integers. If it appears as the third or forth argument of the call it means that all corners, not only the valid ones should be computed by `linopt::corners`.
`NonNegative`	—	all variables are constrained to be nonnegative
`Logic`	—	This allows the algorithm to search for corners in planes like x=0 too, although $x \geq 0$ is not part of the linear program.

Return Value: a set or a list with 3 elements.

Related Functions: `linopt::maximize`, `linopt::minimize`, `linopt::plot_data`

Details:

- [`constr`, `obj`] is a linear program of the same structure like in `linopt::maximize`. The second parameter `vars` specifies the order in which the components of the corners found are printed; if e.g. {x=1, y=2} is a corner and [x,y] was entered, [1,2] will be returned.
- As options, for finding the corners, one may use `All` and/or `Logic`. `All` causes the output of non-feasible corners, too, `Logic` allows the algorithm to search for corners in planes like x=0, too, although $x \geq 0$ is not part of the input. This guarantees that for all non-empty feasible regions a corner will be found.
- As the result of `linopt::corners` a triple consisting of the set of corners, the maximal objective function value found and the corner associated to it is returned. If there is no feasible corner found, only the empty set is returned.

Example 1. We compute all valid corners of a small linear program:

```
>> k := [[4 <= 2*x + 2*y, -2 <= 4*y - 2*x, -8 <= y - 2*x,
          y - 2*x <= -2, y <= 6], x + y]:
   linopt::corners(k, [x, y])

 [{[5, 2], [4, 6], [7, 6], [4/3, 2/3], [5/3, 1/3]}, 13, [7, 6]]
```

Now we compute all corners, also the ones which are not valid. We see that we now get e.g. also the corner which is given by the cut of $-2x + 4y = 2$ and $-2 * x + y <= -2$. Here we see that the invalid corner (13,6) has the maximal objective function value 19:

```
>> k := [[4 <= 2*x + 2*y, -2 <= 4*y - 2*x, -8 <= y - 2*x,
          y - 2*x <= -2, y <= 6], x + y]:
   linopt::corners(k, [x, y], All)

 [{[1, 0], [5, 2], [-4, 6], [4, 6], [7, 6], [13, 6],

    [4/3, 2/3], [5/3, 1/3], [10/3, -4/3]}, 19, [13, 6]]

>> delete k:
```

Example 2. As one can see the linear program given by the constraints $x + y >= -1$ and $x + y <= 3$ and the linear objective function $x + 2y$ has no corners:

```
>> l := [[-1 <= x + y, x + y <= 3], x + 2*y]:
   linopt::corners(l,[x,y]), linopt::corners(l,[x,y], All)

                                {}, {}
```

If one also assumes a cut with a coordinate plane as a corner, some corners exist. One can use `linopt::plot_data` to visualize this problem:

```
>> linopt::corners(l, [x,y], Logic)

     [{[0, 0], [0, -1], [-1, 0], [0, 3], [3, 0]}, 6, [0, 3]]

>> delete l:
```

Background:

- References:

  Papadimitriou, Christos H; Steiglitz, Kenneth: Combinatorial Optimization; Algorithms and Complexity. Prentice-Hall, 1982.

  Nemhauser, George L; Wolsey, Laurence A: Integer and Combinatorial Optimization. New York, Wiley, 1988.

  Salkin, Harvey M; Mathur, Kamlesh: Foundations of Integer Programming. North-Holland, 1989.

  Neumann, Klaus; Morlock, Martin: Operations-Research. Munich, Hanser, 1993.

  Duerr, Walter; Kleibohm, Klaus: Operations Research; Lineare Modelle und ihre Anwendungen. Munich, Hanser, 1992.

  Suhl, Uwe H: MOPS - Mathematical OPtimization System. European Journal of Operational Research 72(1994)312-322. North-Holland, 1994.

  Suhl, Uwe H; Szymanski, Ralf: Supernode Processing of Mixed Integer Models. Boston, Kluwer Academic Publishers, 1994.

Changes:

- `linopt::corners` is a new function.
- The linopt library was completely rewritten in MuPAD version 2.0.

`linopt::maximize` – maximize a linear or mixed-integer program

`linopt::maximize([constr, obj])` returns the solution of the linear or mixed-integer program given by the constraints `constr` and the linear objective function `obj` which should be maximized.

Call(s):

- `linopt::maximize([constr, obj])`
- `linopt::maximize([constr, obj], DualPrices)`
- `linopt::maximize([constr, obj<, NonNegative><, seti>])`
- `linopt::maximize([constr, obj<, NonNegative><, All>])`
- `linopt::maximize([constr, obj<, setn><, seti>])`
- `linopt::maximize([constr, obj<, setn><, All>])`
- `linopt::maximize([constr, obj<, NonNegative>], DualPrices)`
- `linopt::maximize([constr, obj<, set>], DualPrices)`

Parameters:

`constr`	—	a set or list of linear constraints
`obj`	—	a linear expression
`seti`	—	a set which contains identifiers interpreted as indeterminates
`setn`	—	a set which contains identifiers interpreted as indeterminates

Options:

`All`	—	all variables are constrained to be integers
`NonNegative`	—	all variables are constrained to be nonnegative
`DualPrices`	—	This option is only available in the linear case. It causes the output of the dual-prices in addition to the solution-triple.

Return Value: a list or a sequence of a list and a set containing the solution of the linear or mixed-integer program.

Related Functions: `linopt::minimize`, `linopt::plot_data`, `linopt::corners`

Details:

- The expression `obj` is the linear objective function to be maximized subject to the linear constraints `constr`. The function `linopt::maximize` returns a triple consisting of the state of the output, `OPTIMAL`, `EMPTY` or

`UNBOUNDED`, a set of equations which describes the optimal solution of the specified linear program, which is empty or depends on a free variable Φ subject to the state, and finally the maximal objective function value, which can be either a number, `-infinity` or a linear function in Φ.

- The states `OPTIMAL`, `EMPTY` or `UNBOUNDED` have the following meanings. `OPTIMAL` means an optimal solution for the linear program was found. If the state is `EMPTY` no optimal solution was found and if it is `UNBOUNDED` then the solution has no upper bound.
- If the option *`NonNegative`* is used all variables are constrained to be nonnegative. If instead of *`NonNegative`* a set `setn` is given then only the variables from `setn` are constrained to be nonnegative.
- If the option *`All`* is used all variables are constrained to be integers. If instead of *`All`* a set `seti` is given, then only the variables from `seti` are constrained to be integers.
- As a second parameter for `linopt::maximize` the option *`DualPrices`* is provided for the linear case (the first parameter therefore must not have more than three elements). This option causes the output of the dual-prices in addition to the solution-tripel. In this case the result of `linopt::maximize` is a sequence of a list containing the solution-tripel and a set containing the dual prices. See example 4.

Example 1. We try to solve the linear program

$$\begin{array}{rcl} 2c_1 & \leq & 1 \\ 2c_2 & \leq & 1 \end{array}$$

with the linear objective function $c_1 + 2c_2$:

```
>> linopt::maximize([{2*c1 <= 1, 2*c2 <= 1}, c1 + 2*c2])

              [OPTIMAL, {c1 = 1/2, c2 = 1/2}, 3/2]
```

Example 2. Now let's have a look at the linear program

$$\begin{array}{rcl} 3x + 4y - 3z & \leq & 23 \\ 5x - 4y - 3z & \leq & 10 \\ 7x + 4y + 11z & \leq & 30 \end{array}$$

with the linear objective function $-x + y + 2z$. If we make no restriction to the variables the result is unbounded:

```
>> c := [{3*x + 4*y - 3*z <= 23, 5*x - 4*y - 3*z <= 10,
          7*x + 4*y + 11*z <= 30}, -x + y + 2*z]:
   linopt::maximize(c)
```

```
 --            {                          7 PHI1         }
 |  UNBOUNDED, { y = 0, x = -PHI1, z = ------ + 30/11 },
 --            {                            11           }

    25 PHI1         --
    ------- + 60/11  |
      11            --
```

But if all variables are constrained to be nonnegative, we get a result. That's also the case if only x and y are constrained to be nonnegative:

```
>> linopt::maximize(append(c, NonNegative));
   linopt::maximize(append(c, {x, y}))
           [OPTIMAL, {x = 0, z = 1/2, y = 49/8}, 57/8]

           [OPTIMAL, {x = 0, z = 1/2, y = 49/8}, 57/8]
>> delete c:
```

Example 3. The following linear program do not have a solution:

```
>> linopt::maximize([{x <= -1, x >= 0}, x])
                     [EMPTY, {}, -infinity]
```

Example 4. The output of the dual prices can be enforced with the option *DualPrices*:

```
>> linopt::maximize([{2*c1 <= 1, 2*c2 <= 1},c1 + 2*c2],
                    DualPrices)
 [OPTIMAL, {c1 = 1/2, c2 = 1/2}, 3/2],

    {[c1 <= 1/2, 1], [c2 <= 1/2, 2]}
```

Example 5. We have a look at the knapsack problem with $x1, x2, x3, x4 \in \{0, 1\}$:

```
>> c := {20*x1 + 15*x2 + 20*x3 + 5*x4 <= 25}:
   c := c union {x1 <= 1, x2 <= 1, x3 <= 1, x4 <= 1}:
   f := 10*x1 + 15*x2 + 16*x3 + x4:
   linopt::maximize([c, f, NonNegative, All])
         [OPTIMAL, {x1 = 0, x2 = 0, x3 = 1, x4 = 1}, 17]
>> delete c, f:
```

Background:

- References:

  Papadimitriou, Christos H; Steiglitz, Kenneth: Combinatorial Optimization; Algorithms and Complexity. Prentice-Hall, 1982.

  Nemhauser, George L; Wolsey, Laurence A: Integer and Combinatorial Optimization. New York, Wiley, 1988.

  Salkin, Harvey M; Mathur, Kamlesh: Foundations of Integer Programming. North-Holland, 1989.

  Neumann, Klaus; Morlock, Martin: Operations-Research. Munich, Hanser, 1993.

  Duerr, Walter; Kleibohm, Klaus: Operations Research; Lineare Modelle und ihre Anwendungen. Munich, Hanser, 1992.

  Suhl, Uwe H: MOPS - Mathematical OPtimization System. European Journal of Operational Research 72(1994)312-322. North-Holland, 1994.

  Suhl, Uwe H; Szymanski, Ralf: Supernode Processing of Mixed Integer Models. Boston, Kluwer Academic Publishers, 1994.

Changes:

- The order of the parameters changed. It's now also possible to solve a linear integer program.

`linopt::minimize` – minimize a linear or mixed-integer program

`linopt::minimize([constr, obj])` returns the solution of the linear or mixed-integer program given by the constraints `constr` and the linear objective function `obj` which should be minimized.

Call(s):

- `linopt::minimize([constr, obj])`
- `linopt::minimize([constr, obj], DualPrices)`
- `linopt::minimize([constr, obj<, NonNegative><, seti>])`
- `linopt::minimize([constr, obj<, NonNegative><, All>])`
- `linopt::minimize([constr, obj<, setn><, seti>])`
- `linopt::minimize([constr, obj<, setn><, All>])`
- `linopt::minimize([constr, obj<, NonNegative>], DualPrices)`
- `linopt::minimize([constr, obj<, set>], DualPrices)`

Parameters:

`constr`	—	a set or list of linear constraints
`obj`	—	a linear expression
`seti`	—	s set which contains identifiers interpreted as indeterminates
`setn`	—	a set which contains identifiers interpreted as indeterminates

Options:

`All`	—	all variables are constrained to be integer
`NonNegative`	—	all variables are constrained to be nonnegative
`DualPrices`	—	This option is only available in the linear case. It causes the output of the dual-prices in addition to the solution-tripel.

Return Value: a list or a sequence of a list and a set containing the solution of the linear or mixed-integer program.

Related Functions: `linopt::maximize`, `linopt::plot_data`, `linopt::corners`

Details:

- The expression `obj` is the linear objective function to be minimized subject to the linear constraints `constr`. The function `linopt::minimize` returns a triple consisting of the state of the output, `OPTIMAL`, `EMPTY` or `UNBOUNDED`, a set of equations which describes the optimal solution of the specified linear program, which is empty or depends on a free variable Φ subject to the state, and finally the minimal objective function value, which can be either a number, `infinity` or a linear function in Φ.

- The states `OPTIMAL`, `EMPTY` or `UNBOUNDED` have the following meanings. `OPTIMAL` means an optimal solution for the linear program was found. If the state is `EMPTY` no optimal solution was found and if it is `UNBOUNDED` then the solution has no upper bound.

- If the option *`NonNegative`* is used all variables are constrained to be nonnegative. If instead of *`NonNegative`* a set `setn` is given then only the variables from `setn` are constrained to be nonnegative.

- If the option *`All`* is used all variables are constrained to be integers. If instead of *`All`* a set `seti` is given, then only the variables from `seti` are constrained to be integers.

- As a second parameter for `linopt::minimize` the option *`DualPrices`* is provided for the linear case (the first parameter therefore must not have more than three elements). This option causes the output of the

dual-prices in addition to the solution-tripel. In this case the result of `linopt::minimize` is a sequence of a list containing the solution-tripel and a set containing the dual prices. See example 4.

Example 1. We try to solve the linear program

$$\begin{array}{rcl} c_1 + c_2 & \leq & 3 \\ c_2 & \leq & 9 \end{array}$$

with the linear objective function $-c_1 - c_2$:

```
>> linopt::minimize([{c1 + c2 <= 3, c2 <= 9}, -c1 - c2])

                [OPTIMAL, {c1 = -6, c2 = 9}, -3]
```

Example 2. Now let's have a look at the linear program

$$\begin{array}{rcl} 3x + 4y - 3z & \leq & 23 \\ 5x - 4y - 3z & \leq & 10 \\ 7x + 4y + 11z & \leq & 30 \end{array}$$

with the linear objective function $-x + y + 2z$. If we make no restriction to the variables the result is unbounded:

```
>> c := [{3*x + 4*y - 3*z <= 23, 5*x - 4*y - 3*z <= 10,
          7*x + 4*y + 11*z <= 30}, -x + y + 2*z]:
   linopt::minimize(c)

 --             {
 |  UNBOUNDED, { y = -PHI1, x = - 4 PHI1 - 13/2,
 --             {

           16 PHI1         }    23 PHI1          --
    z = - ------- - 85/6 }, - ------- - 131/6   |
             3             }       3             --
```

But if all variables are constrained to be nonnegative, we get a result. That's also the case if only x and y are constrained to be nonnegative:

```
>> linopt::minimize(append(c, NonNegative));
   linopt::minimize(append(c, {x, y}))

              [OPTIMAL, {y = 0, z = 0, x = 2}, -2]

          [OPTIMAL, {x = 0, z = -11/2, y = 13/8}, -75/8]

>> delete c:
```

Example 3. The following linear program do not have a solution:

```
>> linopt::minimize([{x <= -1, x >= 0}, x])

                      [EMPTY, {}, infinity]
```

Example 4. The output of the dual prices can be enforced with the option *DualPrices*:

```
>> linopt::minimize([{c1 + c2 <= 3, c2 <= 9}, -c1 - c2],
                    DualPrices)

 [OPTIMAL, {c1 = -6, c2 = 9}, -3],

    {[c2 <= 9, 0], [c1 + c2 <= 3, 1]}
```

Background:

- References:

  Papadimitriou, Christos H; Steiglitz, Kenneth: Combinatorial Optimization; Algorithms and Complexity. Prentice-Hall, 1982.

  Nemhauser, George L; Wolsey, Laurence A: Integer and Combinatorial Optimization. New York, Wiley, 1988.

  Salkin, Harvey M; Mathur, Kamlesh: Foundations of Integer Programming. North-Holland, 1989.

  Neumann, Klaus; Morlock, Martin: Operations-Research. Munich, Hanser, 1993.

  Duerr, Walter; Kleibohm, Klaus: Operations Research; Lineare Modelle und ihre Anwendungen. Munich, Hanser, 1992.

  Suhl, Uwe H: MOPS - Mathematical OPtimization System. European Journal of Operational Research 72(1994)312-322. North-Holland, 1994.

  Suhl, Uwe H; Szymanski, Ralf: Supernode Processing of Mixed Integer Models. Boston, Kluwer Academic Publishers, 1994.

Changes:

- The order of the parameters changed. It's now also possible to solve a linear integer program.

`linopt::plot_data` – plot the feasible region of a linear program

`linopt::plot_data([constr, obj], vars)` returns a graphical description of the feasible region of the linear program [constr, obj], and the line vertical to the objective function vector through the corner with the maximal objective function value found.

Call(s):

- `linopt::plot_data([constr, obj], vars)`
- `linopt::plot_data([constr, obj<, NonNegative><, seti>], vars)`
- `linopt::plot_data([constr, obj<, NonNegative><, All>], vars)`
- `linopt::plot_data([constr, obj<, setn><, seti>], vars)`
- `linopt::plot_data([constr, obj<, setn><, All>], vars)`

Parameters:

`constr`	—	a set or list of linear constraints
`obj`	—	a linear expression
`seti`	—	a set which contains identifiers interpreted as indeterminates
`setn`	—	a set which contains identifiers interpreted as indeterminates
`vars`	—	a list containing the two variables of the linear program described by `constr` and `obj` and the existing options

Options:

`All`	—	all variables are constrained to be integer
`NonNegative`	—	all variables are constrained to be nonnegative

Return Value: an expression describing a graphical object which can be used by `plot2d`.

Related Functions: `linopt::maximize`, `linopt::minimize`, `linopt::corners`

Details:

- [`constr`, `obj`] is a linear program with exactly two variables. The problem has the same structure like in `linopt::maximize`. The second parameter `vars` specifies which variable belongs to the horizontal and vertical axis.

Example 1. We plot the feasible region of the given linear program. Here the valid corners of the linear program are easy to see:

```
>> k := [[2*x + 2*y >= 4, -2*x + 4*y >= -2, -2*x + y >= -8,
          -2*x + y <= -2, y <= 6], x + y, NonNegative]:
   g := linopt::plot_data(k, [x, y]):
   plot2d(g):
```

In this example there is no difference if the Option *NonNegative* is given for the linear program or not:

```
>> k := [[2*x + 2*y >= 4, -2*x + 4*y >= -2, -2*x + y >= -8,
          -2*x + y <= -2, y <= 6], x + y]:
   g := linopt::plot_data(k, [x, y]):
   plot2d(g):

>> delete k, g:
```

Example 2. Now we give an example where one can see a difference if the variables are constrained to be nonnegative:

```
>> k := [[x + y >= -1, x + y <= 3], x + 2*y]:
   g := linopt::plot_data(k, [x, y]):
   plot2d(g):

>> k := [[x + y >= -1, x + y <= 3], x + 2*y, NonNegative]:
   g := linopt::plot_data(k, [x, y]):
   plot2d(g):

>> delete k, g:
```

Background:

- References:

  Papadimitriou, Christos H; Steiglitz, Kenneth: Combinatorial Optimization; Algorithms and Complexity. Prentice-Hall, 1982.

  Nemhauser, George L; Wolsey, Laurence A: Integer and Combinatorial Optimization. New York, Wiley, 1988.

  Salkin, Harvey M; Mathur, Kamlesh: Foundations of Integer Programming. North-Holland, 1989.

  Neumann, Klaus; Morlock, Martin: Operations-Research. Munich, Hanser, 1993.

  Duerr, Walter; Kleibohm, Klaus: Operations Research; Lineare Modelle und ihre Anwendungen. Munich, Hanser, 1992.

Suhl, Uwe H: MOPS - Mathematical OPtimization System. European Journal of Operational Research 72(1994)312-322. North-Holland, 1994.

Suhl, Uwe H; Szymanski, Ralf: Supernode Processing of Mixed Integer Models. Boston, Kluwer Academic Publishers, 1994.

Changes:

- `linopt::plot_data` is a new function.
- The linopt library was completely rewritten in MuPAD version 2.0.

`linopt::Transparent` – return the ordinary simplex tableau of a linear program

`linopt::Transparent([constr, obj])` returns the ordinary simplex tableau of the given linear program given by the constraints `constr` and the linear objective function `obj`.

Call(s):

- `linopt::Transparent([constr, obj])`
- `linopt::Transparent([constr, obj<, NonNegative><, seti>])`
- `linopt::Transparent([constr, obj<, NonNegative><, All>])`
- `linopt::Transparent([constr, obj<, setn><, seti>])`
- `linopt::Transparent([constr, obj<, setn><, All>])`

Parameters:

`constr`	—	a set or list of linear constraints
`obj`	—	a linear expression
`seti`	—	a set which contains identifiers interpreted as indeterminates
`setn`	—	a set which contains identifiers interpreted as indeterminates

Options:

`All`	—	all variables are constrained to be integer
`NonNegative`	—	all variables are constrained to be nonnegative

Return Value: a simplex tableau of domain type `linopt::Transparent`.

Related Functions: `linopt::Transparent::autostep`, `linopt::Transparent::convert`, `linopt::Transparent::dual_prices`, `linopt::Transparent::phaseI_tableau`, `linopt::Transparent::phaseII_tableau`, `linopt::Transparent::result`, `linopt::Transparent::simplex`, `linopt::Transparent::suggest`, `linopt::Transparent::userstep`

Details:

- [`constr`, `obj`] is a Linear Optimization Problem of the same structure like in `linopt::maximize`. As the result the ordinary simplex tableau of the given problem is returned; this means that equations will be replaced by two unequations and unbounded variables will be replaced by two new variables.
- Internally the tableau returned consists of more information than viewable on the screen. Therefore `linopt::Transparent::convert` is provided to perform the transduction into the structure of the screen-tableau. (This can be necessary if the returned tableau shall serve as an input-parameter for another function — e.g. a user defined procedure for the selection of pivot elements.) If an ordinary simplex with two phases is wished, the next step should be the call of `linopt::Transparent::phaseI_tableau`.
- All functions of the linopt library using the tableau returned by `linopt::Transparent` try to minimize the problem! Therefore it can be necessary to multiply the objective function with -1 first.
- In the simplex tableau returned a special notation is used. "linopt" stands for the tableau them self, "obj" describes the linear objective function, "restr" stands for the vector of restrictions, slk[1], slk[2], ... are the slack variables and the names of the other variables stand for themselves. Variables which are given as row labels indicate that they are part of the base.

Example 1. First a small example, returning the ordinary simplex tableau of the given linear program. One can see that the slack variables are forming the basis:

```
>> k := [[x + y >= -1, x + y <= 3], x + 2*y, NonNegative]:
   linopt::Transparent(k)
```

```
+-                                                  -+
|  "linopt", "restr", slk[1], slk[2],  x,  y  |
|                                                    |
|    "obj",     0,       0,      0,    1,  2  |
|                                                    |
```

```
         |   slk[1],     1,       1,      0,   -1, -1  |
         |                                              |
         |   slk[2],     3,       0,      1,    1,  1  |
         +-                                           -+
```

It follows a little bit larger example:

```
>> k := [{3*x + 4*y - 3*z <= 23, 5*x - 4*y - 3*z <= 10,
          7*x + 4*y + 11*z <= 30}, -x + y + 2*z, NonNegative]:
   linopt::Transparent(k)

   +-                                                        -
+
   |  "linopt", "restr", slk[1], slk[2], slk[3],  z,  x,  y  |
   |                                                         |
   |    "obj",     0,       0,      0,      0,    2, -1,  1  |
   |                                                         |
   |   slk[1],     30,      1,      0,      0,   11,  7,  4  |
   |                                                         |
   |   slk[2],     10,      0,      1,      0,   -3,  5, -
4  |
   |                                                         |
   |   slk[3],     23,      0,      0,      1,   -3,  3,  4  |
   +-                                                       -
+
```

The result of `linopt::Transparent` is of domain type `linopt::Transparent`. So it can be used as input for other `linopt::Transparent::*` function, e.g. for
`linopt::Transparent::suggest`:

```
>> k := [[x + y >= -1, x + y <= 3], x + 2*y, NonNegative]:
   t := linopt::Transparent(k):
   domtype(t), linopt::Transparent::suggest(t)

                  linopt::Transparent, OPTIMAL

>> delete k, t:
```

Background:

- References:

  Papadimitriou, Christos H; Steiglitz, Kenneth: Combinatorial Optimization; Algorithms and Complexity. Prentice-Hall, 1982.

  Nemhauser, George L; Wolsey, Laurence A: Integer and Combinatorial Optimization. New York, Wiley, 1988.

Salkin, Harvey M; Mathur, Kamlesh: Foundations of Integer Programming. North-Holland, 1989.

Neumann, Klaus; Morlock, Martin: Operations-Research. Munich, Hanser, 1993.

Duerr, Walter; Kleibohm, Klaus: Operations Research; Lineare Modelle und ihre Anwendungen. Munich, Hanser, 1992.

Suhl, Uwe H: MOPS - Mathematical OPtimization System. European Journal of Operational Research 72(1994)312-322. North-Holland, 1994.

Suhl, Uwe H; Szymanski, Ralf: Supernode Processing of Mixed Integer Models. Boston, Kluwer Academic Publishers, 1994.

Changes:

- `linopt::Transparent` is a new function.
- The linopt library was completely rewritten in MuPAD version 2.0.

`linopt::Transparent::autostep` – perform the next simplex step

`linopt::Transparent::autostep(tableau)` perform the next step of the simplex algorithm for the given simplex tableau `tableau`.

Call(s):

- `linopt::Transparent::autostep(tableau)`

Parameters:

`tableau` — a simplex tableau of domain type `linopt::Transparent`

Return Value: a simplex tableau of domain type `linopt::Transparent` or a set which contains the solution of the linear program.

Related Functions: `linopt::Transparent`, `linopt::Transparent::convert`, `linopt::Transparent::dual_prices`, `linopt::Transparent::phaseI_tableau`, `linopt::Transparent::result`, `linopt::Transparent::simplex`, `linopt::Transparent::suggest`, `linopt::Transparent::userstep`

Details:

- `linopt::Transparent::autostep(tableau)` performs the next step of the simplex algorithm. This is the same step that `linopt::Transparent::suggest` would suggest for the given simplex tableau `tableau`.
- Normally `linopt::Transparent::autostep` returns the next simplex tableau. If the calculation of the simplex algorithm is finished `linopt::Transparent::au` returns a set containing a solution of the given linear program described by `tableau`.

Example 1. The ordinary simplex tableau of a given linear program is created:

```
>> k := [[x + y >= 2], x, NonNegative]:
   t := linopt::Transparent(k)
```

```
+-                                      -+
|  "linopt", "restr", slk[1],  x,  y  |
|                                        |
|    "obj",     0,       0,    1,  0  |
|                                        |
|   slk[1],     -2,      1,   -1, -1  |
+-                                      -+
```

The next two steps of the simplex algorithm are executed for the given simplex tableau:

```
>> linopt::Transparent::autostep(t);
   linopt::Transparent::autostep(%)
```

```
+-                                     -+
|  "linopt", "restr", slk[1], x,  y  |
|                                       |
|    "obj",     -2,      1,   0, -1  |
|                                       |
|      x,       2,      -1,   1,  1  |
+-                                     -+

 +-                                    -+
 |  "linopt", "restr", slk[1], x, y  |
 |                                     |
 |    "obj",     0,       0,   1, 0  |
 |                                     |
 |      y,       2,      -1,   1, 1  |
 +-                                    -+
```

```
>> delete k, t:
```

Example 2. The ordinary simplex tableau of a given linear program is created:

```
>> k := [[x + y >= -1, x + y <= 3], x + 2*y, NonNegative]:
   t := linopt::Transparent(k)
```

```
+-                                              -+
|  "linopt", "restr", slk[1], slk[2],  x,  y  |
|                                                |
|    "obj",     0,       0,      0,    1,  2  |
|                                                |
|   slk[1],     1,       1,      0,   -1, -1  |
|                                                |
|   slk[2],     3,       0,      1,    1,  1  |
+-                                              -+
```

If the end of the simplex algorithm is reached, `linopt::Transparent::autostep` returns a solution of the given linear program:

```
>> linopt::Transparent::suggest(t),
   linopt::Transparent::autostep(t)
```

```
                    OPTIMAL, {x = 0, y = 0}
```

```
>> delete k, t:
```

Background:

- References:

Papadimitriou, Christos H; Steiglitz, Kenneth: Combinatorial Optimization; Algorithms and Complexity. Prentice-Hall, 1982.

Nemhauser, George L; Wolsey, Laurence A: Integer and Combinatorial Optimization. New York, Wiley, 1988.

Salkin, Harvey M; Mathur, Kamlesh: Foundations of Integer Programming. North-Holland, 1989.

Neumann, Klaus; Morlock, Martin: Operations-Research. Munich, Hanser, 1993.

Duerr, Walter; Kleibohm, Klaus: Operations Research; Lineare Modelle und ihre Anwendungen. Munich, Hanser, 1992.

Suhl, Uwe H: MOPS - Mathematical OPtimization System. European Journal of Operational Research 72(1994)312-322. North-Holland, 1994.

Suhl, Uwe H; Szymanski, Ralf: Supernode Processing of Mixed Integer Models. Boston, Kluwer Academic Publishers, 1994.

Changes:

- `linopt::Transparent::autostep` is a new function.
- The linopt library was completely rewritten in MuPAD version 2.0.

`linopt::Transparent::clean_basis` – delete all slack variables of the first phase from the basis

`linopt::Transparent::clean_basis(tableau)` removes the additional slack variables of the phase one of the simplex algorithm from the optimal basic (described by `tableau`) calculated by `linopt::Transparent::phaseI_tableau` and `linopt::Transparent::simplex`.

Call(s):

- `linopt::Transparent::clean_basis(tableau)`

Parameters:

`tableau` — a simplex tableau of domain type `linopt::Transparent`

Return Value: a simplex tableau of domain type `linopt::Transparent`.

Related Functions: `linopt::Transparent`, `linopt::Transparent::autostep`, `linopt::Transparent::convert`, `linopt::Transparent::dual_prices`, `linopt::Transparent::phaseII_tableau`, `linopt::Transparent::result`, `linopt::Transparent::simplex`, `linopt::Transparent::suggest`, `linopt::Transparent::userstep`

Details:

- At the end of the phase one of the 2-phase simplex algorithm, explicitly started by using `linopt::Transparent::phaseI_tableau`, it is necessary to eliminate all artificial variables from the optimal basis before phase two can be started by using `linopt::Transparent::phaseII_tableau`. `linopt::Transparent::clean_basis` performs some pivot steps until all phase one slack variables do not occur in the basis any longer.

Example 1. In this example we first compute an optimal basis for the first phase of the simplex algorithm:

```
>> t := linopt::Transparent([[x <= 1,y <= 1,x + y >= 2],
                             0,NonNegative]):
   t := linopt::Transparent::phaseI_tableau(t):
   t := linopt::Transparent::simplex(t)
```

```
                        array(1..5, 1..10,
                          (1, 1) = "linopt",
                          (1, 2) = "restr",
                          (1, 3) = slk[4],
                          (1, 4) = slk[5],
                          (1, 5) = slk[6],
                          (1, 6) = slk[1],
                          (1, 7) = slk[2],
                          (1, 8) = slk[3],
                          (1, 9) = x,
                          (1, 10) = y,
                          (2, 1) = "obj",
                          (2, 2) = 0,
                                 .
                                 .
                          (4, 10) = 1,
                          (5, 1) = slk[6],
                          (5, 2) = 0,
                          (5, 3) = -1,
                          (5, 4) = -1,
                          (5, 5) = 1,
                          (5, 6) = -1,
                          (5, 7) = -1,
                          (5, 8) = -1,
                          (5, 9) = 0,
                          (5, 10) = 0
                        )
```

As we can see the artificial slack variable slk[6] is an element of the optimal basis. An error message is returned if we apply `linopt::Transparent::phaseII_tableau` or `linopt::Transparent::simplex` to this simplex tableau:

```
>> linopt::Transparent::phaseII_tableau(t);

 Error:  Clean the basis from phase I slack variables first! [l\
 inopt::Transparent::phaseII_tableau]
```

So we have to use `linopt::Transparent::clean_basis` before continuing with the appropriate function. To get a smarter output in this example one should use at least a `TEXTWIDTH` of 80:

```
>> t := linopt::Transparent::clean_basis(t);
   linopt::Transparent::phaseII_tableau(t)

                      array(1..5, 1..10,
                        (1, 1) = "linopt",
                        (1, 2) = "restr",
                        (1, 3) = slk[4],
                        (1, 4) = slk[5],
                        (1, 5) = slk[6],
                        (1, 6) = slk[1],
                        (1, 7) = slk[2],
                        (1, 8) = slk[3],
                        (1, 9) = x,
                        (1, 10) = y,
                        (2, 1) = "obj",
                        (2, 2) = 0,
                               .
                               .
                        (4, 10) = 1,
                        (5, 1) = slk[1],
                        (5, 2) = 0,
                        (5, 3) = 1,
                        (5, 4) = 1,
                        (5, 5) = -1,
                        (5, 6) = 1,
                        (5, 7) = 1,
                        (5, 8) = 1,
                        (5, 9) = 0,
                        (5, 10) = 0
                      )


      +-                                                    -
+
      |  "linopt", "restr", slk[1], slk[2], slk[3], x, y  |
      |                                                    |
      |    "obj",     0,       0,      0,      0,   0, 0  |
      |                                                    |
      |      x,       1,       0,     -1,     -1,   1, 0  |
      |                                                    |
      |      y,       1,       0,      1,      0,   0, 1  |
      |                                                    |
      |   slk[1],     0,       1,      1,      1,   0, 0  |
      +-                                                    -
+

>> delete t:
```

Background:

- References:

  Papadimitriou, Christos H; Steiglitz, Kenneth: Combinatorial Optimization; Algorithms and Complexity. Prentice-Hall, 1982.

  Nemhauser, George L; Wolsey, Laurence A: Integer and Combinatorial Optimization. New York, Wiley, 1988.

  Salkin, Harvey M; Mathur, Kamlesh: Foundations of Integer Programming. North-Holland, 1989.

  Neumann, Klaus; Morlock, Martin: Operations-Research. Munich, Hanser, 1993.

  Duerr, Walter; Kleibohm, Klaus: Operations Research; Lineare Modelle und ihre Anwendungen. Munich, Hanser, 1992.

  Suhl, Uwe H: MOPS - Mathematical OPtimization System. European Journal of Operational Research 72(1994)312-322. North-Holland, 1994.

  Suhl, Uwe H; Szymanski, Ralf: Supernode Processing of Mixed Integer Models. Boston, Kluwer Academic Publishers, 1994.

Changes:

- `linopt::Transparent::clean_basis` is a new function.
- The linopt library was completely rewritten in MuPAD version 2.0.

`linopt::Transparent::convert` – transform the given tableau into a structure printable on screen

`linopt::Transparent::convert(tableau)` converts the simplex tableau, into a two dimensional array.

Call(s):

- `linopt::Transparent::convert(tableau)`

Parameters:

`tableau` — a simplex tableau of domain type

Return Value: a two dimensional array, representing the given simplex tableau `tableau`.

Related Functions: `linopt::Transparent`, `linopt::Transparent::autostep`, `linopt::Transparent::phaseI_tableau`, `linopt::Transparent::phaseII_tableau`, `linopt::Transparent::suggest`, `linopt::Transparent::userstep`

Details:

- Internally the given `tableau` of domain type `linopt::Transparent` contains a lot of more information than the simplex tableau which is printed by some functions of the `linopt` library, e.g. `linopt::Transparent::simplex`, and which is visible on the screen. Furthermore it is not possible to access the element in the i-th row and j-th column of `tableau` to get the corresponding element from the simplex tableau which is visible on the screen.

  `linopt::Transparent::convert` converts `tableau` into a two dimensional array which corresponds with the screen-tableau. One can now access the element in the i-th row and j-th column of the simplex tableau by accessing the corresponding element of the array.

  While the internal structure of `tableau` is not known the structure of the two dimensional array is well defined. So it can be easily used in own procedures. See example 2.

Example 1. We convert a simplex tableau of domain type `linopt::Transparent` into a two dimensional array:

```
>> k := [[x + y >= 2], x, NonNegative]:
   t := linopt::Transparent(k):
   a := linopt::Transparent::convert(t):
   t, domtype(t);
   a, domtype(a)
```

```
  +-                                    -+
  |  "linopt", "restr", slk[1],  x,  y   |
  |                                       |
  |    "obj",     0,       0,    1,  0  |, linopt::Transparent
  |                                       |
  |   slk[1],     -2,      1,   -1, -1   |
  +-                                    -+

       +-                                    -+
       |  "linopt", "restr", slk[1],  x,  y   |
       |                                       |
       |    "obj",     0,       0,    1,  0  |, DOM_ARRAY
       |                                       |
```

```
       |   slk[1],     -2,      1,   -1, -1  |
       +-                                   -+

>> delete a, k, t:
```

Example 2. We will write another simplex routine `mysimplex` for solving a linear program. For this we define the function `eigenpivot` for finding the pivot element of a given simplex tableau. `eigenpivot` assumes that the simplex tableau is given as a two dimensional array.

Here is the procedure `eigenpivot`, which is not coded in every detail, e.g., the error checking isn't implemented completely:

```
>> eigenpivot := proc(T: DOM_ARRAY)
      local i,j,m,n,k,l,mini;
      begin
        m := op(T,[0,2,2]):
        n := op(T,[0,3,2]):
        k := 0:
        l := 0:
        mini := unbesetzt:

        for j from 3 to n do
          if T[2,j] < 0 then
            l := j:
            break
          end_if:
        end_for:
        if l=0 then return(OPTIMAL) end_if:
        for i from 3 to m do
          if T[i,l] > 0 and (mini=unbesetzt or T[i,2]/T[i,l] < mini) then
            k := i:
            mini := T[k,2]/T[k,l]
          end_if
        end_for:
        if k=0 then return(UNBOUNDED) end_if:
        return(T[k,1],T[1,l]):
   end_proc:
```

This is the new simplex algorithm `mysimplex` which uses `eigenpivot` and some function from the `linopt` library:

```
>> mysimplex := proc(P)
      local T;
      begin
        T := linopt::Transparent(P):
        T := linopt::Transparent::phaseI_tableau(T):
```

```
        piv := eigenpivot(linopt::Transparent::convert(T)):
        while piv <> OPTIMAL and piv <> UNBOUNDED do
          T := linopt::Transparent::userstep(T,piv):
          piv := eigenpivot(linopt::Transparent::convert(T))
        end_while:

        if piv = UNBOUNDED then
          error(" Phase I unbounded ?!?")
        end_if:
        if T[2,2] <> 0
           then return(EMPTY)
        end_if:
        T := linopt::Transparent::clean_basis(T):

        T := linopt::Transparent::phaseII_tableau(T):
        piv := eigenpivot(linopt::Transparent::convert(T)):
        while piv <> OPTIMAL and piv <> UNBOUNDED do
          T := linopt::Transparent::userstep(T,piv):
          piv := eigenpivot(linopt::Transparent::convert(T))
        end_while:

        if piv = OPTIMAL
          then return(linopt::Transparent::result(T))
          else return(UNBOUNDED)
        end_if
   end_proc:
```

We now apply `mysimplex` to a linear program:

```
>> k := [[2*x + 2*y >= 4, -2*x + 4*y >= -2, -2*x + y>= -8,
          -2*x + y <= -2, y <= 6], -x - y]:
   k := append(k, NonNegative):
   mysimplex(k);

                          {x = 7, y = 6}

>> delete k, eigenpivot, mysimplex:
```

Background:

- References:

  Papadimitriou, Christos H; Steiglitz, Kenneth: Combinatorial Optimization; Algorithms and Complexity. Prentice-Hall, 1982.

  Nemhauser, George L; Wolsey, Laurence A: Integer and Combinatorial Optimization. New York, Wiley, 1988.

Salkin, Harvey M; Mathur, Kamlesh: Foundations of Integer Programming. North-Holland, 1989.

Neumann, Klaus; Morlock, Martin: Operations-Research. Munich, Hanser, 1993.

Duerr, Walter; Kleibohm, Klaus: Operations Research; Lineare Modelle und ihre Anwendungen. Munich, Hanser, 1992.

Suhl, Uwe H: MOPS - Mathematical OPtimization System. European Journal of Operational Research 72(1994)312-322. North-Holland, 1994.

Suhl, Uwe H; Szymanski, Ralf: Supernode Processing of Mixed Integer Models. Boston, Kluwer Academic Publishers, 1994.

Changes:

- `linopt::Transparent::convert` is a new function.
- The linopt library was completely rewritten in MuPAD version 2.0.

`linopt::Transparent::dual_prices` – get the dual solution belonging to the given tableau

`linopt::Transparent::dual_prices(tableau)` returns the dual solution of the linear optimization problem given by `tableau`.

Call(s):

- `linopt::Transparent::dual_prices(tableau)`

Parameters:

`tableau` — a simplex tableau of domain type `linopt::Transparent`

Return Value: a set of lists, each containing 2 elements.

Related Functions: `linopt::Transparent`, `linopt::Transparent::result`

Details:

- This procedure returns the dual solution belonging to the given tableau in form of a set of lists containing two elements, the first one is a restriction and the second one is the value belonging to the slack variable connected to the restriction in the dual solution.

Example 1. Here it is demonstrated that the dual solution of the final tableau is similar to the second element of the result of `linopt::minimize` using the option `DualPrices`:

First we compute the final tableau of the simplex algorithm:

```
>> k := [[x<=2, y<=2, x+2*y>=4], -x+y, NonNegative]:
   t := linopt::Transparent(k):
   t := linopt::Transparent::simplex(t)

      +-                                                  -
+
      |  "linopt", "restr", slk[1], slk[2], slk[3], x, y  |
      |                                                   |
      |    "obj",     1,      3/2,     0,     1/2,  0, 0  |
      |                                                   |
      |      x,       2,       1,      0,      0,   1, 0  |
      |                                                   |
      |   slk[2],     1,      1/2,     1,     1/2,  0, 0  |
      |                                                   |
      |      y,       1,     -1/2,     0,    -1/2,  0, 1  |
      +-                                                  -
+
```

Now we compute the solutions:

```
>> linopt::Transparent::dual_prices(t);
   linopt::minimize(k, DualPrices)[2]

 {[0 <= x, 0], [0 <= y, 0], [y <= 2, 0], [x <= 2, 3/2],

    [4 <= x + 2 y, 1/2]}

 {[0 <= x, 0], [0 <= y, 0], [y <= 2, 0], [x <= 2, 3/2],

    [4 <= x + 2 y, 1/2]}

>> delete k, t:
```

Example 2. We compute the dual solution of another linear program:

```
>> k := [[x <= 2, y <= 2, x + 2*y >= 4], -x + y, NonNegative]:
   t := linopt::Transparent(k);
   linopt::Transparent::dual_prices(t)
```

```
     +-                                                        -
+
     |  "linopt", "restr", slk[1], slk[2], slk[3],  x,  y  |
     |                                                       |
     |    "obj",     0,       0,      0,      0,   -1,  1  |
     |                                                       |
     |   slk[1],     2,       1,      0,      0,    1,  0  |
     |                                                       |
     |   slk[2],     2,       0,      1,      0,    0,  1  |
     |                                                       |
     |   slk[3],     -4,      0,      0,      1,   -1, -2  |
     +-                                                        -
+

 {[0 <= x, -1], [0 <= y, 1], [x <= 2, 0], [y <= 2, 0],

    [4 <= x + 2 y, 0]}

>> delete k, t:
```

Background:

- References:

  Papadimitriou, Christos H; Steiglitz, Kenneth: Combinatorial Optimization; Algorithms and Complexity. Prentice-Hall, 1982.

  Nemhauser, George L; Wolsey, Laurence A: Integer and Combinatorial Optimization. New York, Wiley, 1988.

  Salkin, Harvey M; Mathur, Kamlesh: Foundations of Integer Programming. North-Holland, 1989.

  Neumann, Klaus; Morlock, Martin: Operations-Research. Munich, Hanser, 1993.

  Duerr, Walter; Kleibohm, Klaus: Operations Research; Lineare Modelle und ihre Anwendungen. Munich, Hanser, 1992.

  Suhl, Uwe H: MOPS - Mathematical OPtimization System. European Journal of Operational Research 72(1994)312-322. North-Holland, 1994.

  Suhl, Uwe H; Szymanski, Ralf: Supernode Processing of Mixed Integer Models. Boston, Kluwer Academic Publishers, 1994.

Changes:

- `linopt::Transparent::dual_prices` is a new function.
- The linopt library was completely rewritten in MuPAD version 2.0.

`linopt::Transparent::phaseI_tableau` – start an ordinary phase one of a 2-phase simplex algorithm

`linopt::Transparent::phaseI_tableau(tableau)` starts an (ordinary) phase one of the 2-phase simplex algorithm on the given simplex tableau `tableau`.

Call(s):

- `linopt::Transparent::phaseI_tableau(tableau)`

Parameters:

`tableau` — a simplex tableau of domain type `linopt::Transparent`

Return Value: a simplex tableau of domain type `linopt::Transparent`.

Related Functions: `linopt::Transparent`, `linopt::Transparent::autostep`, `linopt::Transparent::convert`, `linopt::Transparent::clean_basis`, `linopt::Transparent::dual_prices`, `linopt::Transparent::phaseII_tableau`, `linopt::Transparent::result`, `linopt::Transparent::simplex`, `linopt::Transparent::suggest`, `linopt::Transparent::userstep`

Details:

- `linopt::Transparent::phaseI_tableau` explicitly starts an (ordinary) phase one of the simplex algorithm , i.e. rows associated with infeasible basic variables are multiplied with -1 and another identity matrix with new slack variables is added to the given tableau. As soon as an optimal tableau with relative costs 0 is found the calculation can be continued with `linopt::Transparent::clean_basis` and the second phase of the simplex algorithm (`linopt::Transparent::phaseII_tableau`).

Example 1. The first simplex tableau is created and the first phase of the simplex algorithm is started:

```
>> t := linopt::Transparent([[x + y >= 2], x, NonNegative]);
   t := linopt::Transparent::phaseI_tableau(t)
```

```
+-                                      -+
|  "linopt", "restr", slk[1],  x,  y  |
|                                       |
|    "obj",     0,       0,    1,  0  |
|                                       |
|   slk[1],     -2,      1,   -1, -1  |
+-                                      -+

+-                                           -+
|  "linopt", "restr", slk[2], slk[1],  x,  y  |
|                                              |
|    "obj",     -2,      0,      1,   -1, -1  |
|                                              |
|   slk[2],     2,       1,     -1,    1,  1  |
+-                                           -+
```

We can see that a new slack variable, slk[2] was added to the tableau. And if we now execute `linopt::Transparent::simplex` we can see that we have just finished the first phase of the simplex algorithm:

```
>> linopt::Transparent::suggest(t);
   t := linopt::Transparent::simplex(t):
   linopt::Transparent::suggest(t)
```

```
                        slk[2], x

            "linopt::Transparent::phaseII_tableau"
```

We continue the simplex algorithm by executing `linopt::Transparent::clean_basis`, `linopt::Transparent::phaseII_tableau` and `linopt::Transparent::simplex`. Observe in this special case `linopt::Transparent::clean_basis` is not necessary:

```
>> t := linopt::Transparent::clean_basis(t):
   t := linopt::Transparent::phaseII_tableau(t):
   t := linopt::Transparent::simplex(t);
   linopt::Transparent::suggest(t)
```

```
            +-                                -+
            |  "linopt", "restr", slk[1], x, y  |
            |                                   |
            |    "obj",     0,       0,   1, 0  |
            |                                   |
            |      y,       2,      -1,   1, 1  |
            +-                                -+

                        OPTIMAL
```

```
>> delete t:
```

Background:

- References:

  Papadimitriou, Christos H; Steiglitz, Kenneth: Combinatorial Optimization; Algorithms and Complexity. Prentice-Hall, 1982.

  Nemhauser, George L; Wolsey, Laurence A: Integer and Combinatorial Optimization. New York, Wiley, 1988.

  Salkin, Harvey M; Mathur, Kamlesh: Foundations of Integer Programming. North-Holland, 1989.

  Neumann, Klaus; Morlock, Martin: Operations-Research. Munich, Hanser, 1993.

  Duerr, Walter; Kleibohm, Klaus: Operations Research; Lineare Modelle und ihre Anwendungen. Munich, Hanser, 1992.

  Suhl, Uwe H: MOPS - Mathematical OPtimization System. European Journal of Operational Research 72(1994)312-322. North-Holland, 1994.

  Suhl, Uwe H; Szymanski, Ralf: Supernode Processing of Mixed Integer Models. Boston, Kluwer Academic Publishers, 1994.

Changes:

- `linopt::Transparent::phaseI_tableau` is a new function.
- The linopt library was completely rewritten in MuPAD version 2.0.

`linopt::Transparent::phaseII_tableau` – start phase two of a 2-phase simplex algorithm

`linopt::Transparent::phaseII_tableau(tableau)` starts the second phase of the simplex algorithm on the given simplex tableau `tableau`.

Call(s):

- `linopt::Transparent::phaseII_tableau(tableau)`

Parameters:

`tableau` — a simplex tableau of domain type `linopt::Transparent`

Return Value: a simplex tableau of domain type `linopt::Transparent`.

Related Functions: `linopt::Transparent`,
`linopt::Transparent::autostep`,
`linopt::Transparent::convert`,
`linopt::Transparent::clean_basis`,
`linopt::Transparent::dual_prices`,
`linopt::Transparent::phaseI_tableau`,
`linopt::Transparent::result`, `linopt::Transparent::simplex`,
`linopt::Transparent::suggest`,
`linopt::Transparent::userstep`

Details:

- After the explicitly started first phase of the simplex algorithm (see `linopt::Transparent::ph` terminated with an optimal tableau with associated costs 0 and no phase one slack variables in the basis (see `linopt::Transparent::clean_basis`) this procedure can be used to start phase II. The procedure eliminates all artificial variables of phase I and their associated columns and reenters the old objective function modified for the new basis.

Example 1. The first simplex tableau is created and the first phase of the simplex algorithm is finished:

```
>> t := linopt::Transparent([[x + y >= 2], x, NonNegative]):
   t := linopt::Transparent::simplex(
          linopt::Transparent::phaseI_tableau(t))
```

```
+-                                          -+
|  "linopt", "restr", slk[2], slk[1], x, y  |
|                                           |
|    "obj",     0,       1,      0,   0, 0  |
|                                           |
|      x,       2,       1,     -1,   1, 1  |
+-                                          -+
```

One sees that the artificial slack variable slk[2] of the first phase is removed by `linopt::Transparent::phaseII_tableau`. In this example it is not necessary to use
`linopt::Transparent::clean_basis` for cleaning the basis:

```
>> linopt::Transparent::phaseII_tableau(t)
```

```
+-                                    -+
|  "linopt", "restr", slk[1], x,  y   |
|                                      |
|    "obj",     -2,      1,   0, -1   |
|                                      |
|      x,       2,      -1,   1,  1   |
+-                                    -+
```

```
>> delete t:
```

Example 2. Again the first simplex tableau is created and the first phase of the simplex algorithm is finished:

```
>> t := linopt::Transparent([[x <= 1, y <= 1, x + y >= 2],
                            0, NonNegative]):
   t := linopt::Transparent::phaseI_tableau(t):
   t := linopt::Transparent::simplex(t)

                        array(1..5, 1..10,
                          (1, 1) = "linopt",
                          (1, 2) = "restr",
                          (1, 3) = slk[4],
                          (1, 4) = slk[5],
                          (1, 5) = slk[6],
                          (1, 6) = slk[1],
                          (1, 7) = slk[2],
                          (1, 8) = slk[3],
                          (1, 9) = x,
                          (1, 10) = y,
                          (2, 1) = "obj",
                          (2, 2) = 0,
                                 .
                                 .
                          (4, 10) = 1,
                          (5, 1) = slk[6],
                          (5, 2) = 0,
                          (5, 3) = -1,
                          (5, 4) = -1,
                          (5, 5) = 1,
                          (5, 6) = -1,
                          (5, 7) = -1,
                          (5, 8) = -1,
                          (5, 9) = 0,
                          (5, 10) = 0
                        )
```

In this example the artificial slack variable `slk[6]` is an element of the optimal basis. So we have to use `linopt::Transparent::clean_basis` before continuing with `linopt::Transparent::phaseII_tableau`, otherwise we will get an error message. To get a smarter output in this example one should use at least a `TEXTWIDTH` of 80:

```
>> linopt::Transparent::phaseII_tableau(t)
```

```
 Error:  Clean the basis from phase I slack variables first! [l\
 inopt::Transparent::phaseII_tableau]
```

```
>> t := linopt::Transparent::clean_basis(t):
   linopt::Transparent::phaseII_tableau(t)
```

```
      +-                                                  -
+
      |  "linopt", "restr", slk[1], slk[2], slk[3], x, y  |
      |                                                   |
      |    "obj",     0,       0,      0,      0,   0, 0  |
      |                                                   |
      |      x,       1,       0,     -1,     -1,   1, 0  |
      |                                                   |
      |      y,       1,       0,      1,      0,   0, 1  |
      |                                                   |
      |   slk[1],     0,       1,      1,      1,   0, 0  |
      +-                                                  -
+
```

```
>> delete t:
```

Background:

- References:

  Papadimitriou, Christos H; Steiglitz, Kenneth: Combinatorial Optimization; Algorithms and Complexity. Prentice-Hall, 1982.

  Nemhauser, George L; Wolsey, Laurence A: Integer and Combinatorial Optimization. New York, Wiley, 1988.

  Salkin, Harvey M; Mathur, Kamlesh: Foundations of Integer Programming. North-Holland, 1989.

  Neumann, Klaus; Morlock, Martin: Operations-Research. Munich, Hanser, 1993.

  Duerr, Walter; Kleibohm, Klaus: Operations Research; Lineare Modelle und ihre Anwendungen. Munich, Hanser, 1992.

  Suhl, Uwe H: MOPS - Mathematical OPtimization System. European Journal of Operational Research 72(1994)312-322. North-Holland, 1994.

  Suhl, Uwe H; Szymanski, Ralf: Supernode Processing of Mixed Integer Models. Boston, Kluwer Academic Publishers, 1994.

Changes:

- `linopt::Transparent::phaseII_tableau` is a new function.
- The linopt library was completely rewritten in MuPAD version 2.0.

`linopt::Transparent::result` – get the basic feasible solution belonging to the given simplex tableau

`linopt::Transparent::result(tableau)` returns the basic feasible solution belonging to the given simplex tableau `tableau`.

Call(s):

- `linopt::Transparent::result(tableau)`

Parameters:

`tableau` — a simplex tableau of domain type `linopt::Transparent`

Return Value: a set containing the values of the user defined variables for the feasible solution.

Related Functions: `linopt::Transparent`, `linopt::Transparent::dual_prices`

Details:

- This procedure returns the basic feasible solution belonging to the given tableau in form of a set of equations. Only the user defined variables are taken into account - the dual prices can be achieved by use of `linopt::Transparent::dual_p`

Example 1. We first compute an edge for an initial simplex tableau:

```
>> k := [[x <= 1, y <= 1, x + y >= 2], 0, NonNegative]:
   t := linopt::Transparent(k):
   linopt::Transparent::result(t)

                          {x = 0, y = 0}
```

Now we compute the edge for the final tableau, which is identical to the optimal solution of the linear program given by `k`. We get the final simplex tableau by using `linopt::Transparent::simplex`:

```
>> t := linopt::Transparent(k):
   t := linopt::Transparent::simplex(t):
   linopt::Transparent::result(t)

                          {x = 1, y = 1}
```

```
>> linopt::minimize(k)

                  [OPTIMAL, {x = 1, y = 1}, 0]

>> delete k, t:
```

Background:

- References:

  Papadimitriou, Christos H; Steiglitz, Kenneth: Combinatorial Optimization; Algorithms and Complexity. Prentice-Hall, 1982.

  Nemhauser, George L; Wolsey, Laurence A: Integer and Combinatorial Optimization. New York, Wiley, 1988.

  Salkin, Harvey M; Mathur, Kamlesh: Foundations of Integer Programming. North-Holland, 1989.

  Neumann, Klaus; Morlock, Martin: Operations-Research. Munich, Hanser, 1993.

  Duerr, Walter; Kleibohm, Klaus: Operations Research; Lineare Modelle und ihre Anwendungen. Munich, Hanser, 1992.

  Suhl, Uwe H: MOPS - Mathematical OPtimization System. European Journal of Operational Research 72(1994)312-322. North-Holland, 1994.

  Suhl, Uwe H; Szymanski, Ralf: Supernode Processing of Mixed Integer Models. Boston, Kluwer Academic Publishers, 1994.

Changes:

- `linopt::Transparent::result` is a new function.
- The linopt library was completely rewritten in MuPAD version 2.0.

`linopt::Transparent::simplex` – finish the current phase of the 2-phase simplex algorithm

`linopt::Transparent::simplex(tableau)` finishes the current phase of the 2-phase simplex algorithm using the given tableau `tableau`.

Call(s):

- `linopt::Transparent::simplex(tableau)`

Parameters:

`tableau` — a simplex tableau of domain type `linopt::Transparent`

Return Value: a simplex tableau of domain type `linopt::Transparent` or the empty set if there was no feasible solution found.

Related Functions: `linopt::Transparent`, `linopt::Transparent::autostep`, `linopt::Transparent::convert`, `linopt::Transparent::dual_prices`, `linopt::Transparent::phaseI_tableau`, `linopt::Transparent::result`, `linopt::Transparent::suggest`, `linopt::Transparent::userstep`

Details:

- `linopt::Transparent::simplex` runs the current phase of the 2-phase simplex algorithm until the end, i.e. if phase I was explicitly started (see `linopt::Transparent::phaseI_tableau`) the first phase will lead the optimal tableau. Sometimes it can be necessary to eliminate some slack variables of phase one by using `linopt::Transparent::clean_basis`.
- If there was no phase I started by the user, `(linopt::Transparent)::simplex` returns the last optimal tableau or the empty set if there was no feasible solution found.

Example 1. We apply `linopt::Transparent::simplex` to an ordinary simplex tableau of a linear program and we get the optimal tableau:

```
>> k := [[x + y >= 2], x, NonNegative]:
   t := linopt::Transparent(k);
   t := linopt::Transparent::simplex(t)
```

```
+-                                      -+
|  "linopt", "restr", slk[1],  x,  y    |
|                                        |
|    "obj",     0,       0,    1,  0    |
|                                        |
|   slk[1],     -2,      1,   -1, -1    |
+-                                      -+
```

```
 +-                                    -+
 |  "linopt", "restr", slk[1], x, y     |
 |                                      |
 |    "obj",     0,       0,   1, 0     |
 |                                      |
 |      y,       2,      -1,   1, 1     |
 +-                                    -+
```

Let us proof the obtained result:

```
>> linopt::Transparent::suggest(t)

                              OPTIMAL

>> delete k, t:
```

Example 2. If the first phase of the simplex algorithm was started explicitly, `linopt::Transparent::simplex` returns only the optimal tableau of the first phase:

```
>> k := [[x + y >= 2], x, NonNegative]:
   t := linopt::Transparent(k):
   t := linopt::Transparent::phaseI_tableau(t);
   t := linopt::Transparent::simplex(t)
```

```
+-                                                -+
|  "linopt", "restr", slk[2], slk[1],  x,  y  |
|                                                  |
|    "obj",     -2,      0,      1,   -1, -1  |
|                                                  |
|   slk[2],     2,       1,     -1,    1,  1  |
+-                                                -+

 +-                                              -+
 |  "linopt", "restr", slk[2], slk[1], x, y  |
 |                                              |
 |    "obj",     0,       1,      0,   0, 0  |
 |                                              |
 |      x,       2,       1,     -1,   1, 1  |
 +-                                              -+
```

The next step of the simplex algorithm is computed:

```
>> linopt::Transparent::suggest(t)

            "linopt::Transparent::phaseII_tableau"
```

With `linopt::Transparent::autostep` we execute the first step of the second phase of the simplex algorithm. One can see that the simplex algorithm is not finished yet:

```
>> t := linopt::Transparent::autostep(t):
   linopt::Transparent::suggest(t);

                              x, y
```

If we then apply `linopt::Transparent::simplex` again we get the optimal solution. Here we don't had to use `linopt::Transparent::clean_basis`, before using `linopt::Transparent::autostep`, because there are no artificial variables in the basis computed by the first `linopt::Transparent::simplex` call above:

```
>> t := linopt::Transparent::simplex(t);
   linopt::Transparent::suggest(t)
```

```
+-                                    -+
|  "linopt", "restr", slk[1], x, y     |
|                                      |
|    "obj",     0,       0,   1, 0     |
|                                      |
|      y,       2,      -1,   1, 1     |
+-                                    -+

                    OPTIMAL
```

```
>> delete k, t:
```

Background:

- References:

  Papadimitriou, Christos H; Steiglitz, Kenneth: Combinatorial Optimization; Algorithms and Complexity. Prentice-Hall, 1982.

  Nemhauser, George L; Wolsey, Laurence A: Integer and Combinatorial Optimization. New York, Wiley, 1988.

  Salkin, Harvey M; Mathur, Kamlesh: Foundations of Integer Programming. North-Holland, 1989.

  Neumann, Klaus; Morlock, Martin: Operations-Research. Munich, Hanser, 1993.

  Duerr, Walter; Kleibohm, Klaus: Operations Research; Lineare Modelle und ihre Anwendungen. Munich, Hanser, 1992.

  Suhl, Uwe H: MOPS - Mathematical OPtimization System. European Journal of Operational Research 72(1994)312-322. North-Holland, 1994.

  Suhl, Uwe H; Szymanski, Ralf: Supernode Processing of Mixed Integer Models. Boston, Kluwer Academic Publishers, 1994.

Changes:

- `linopt::Transparent::simplex` is a new function.
- The linopt library was completely rewritten in MuPAD version 2.0.

`linopt::Transparent::suggest` – suggest the next step in the simplex algorithm

`linopt::Transparent::suggest(tableau)` suggests the next step in the simplex algorithm for the given simplex tableau `tableau`.

Call(s):

- `linopt::Transparent::suggest(tableau)`

Parameters:

`tableau` — a simplex tableau of domain type `linopt::Transparent`

Return Value: a sequence of 2 identifiers, the identifier `OPTIMAL` or the string `"linopt::Transparent::phaseII_tableau"`.

Related Functions: `linopt::Transparent`, `linopt::Transparent::autostep`, `linopt::Transparent::convert`, `linopt::Transparent::dual_prices`, `linopt::Transparent::phaseI_tableau`, `linopt::Transparent::result`, `linopt::Transparent::simplex`, `linopt::Transparent::userstep`

Details:

- The function suggests the next step in the simplex algorithm for `tableau`. Normally this suggestion will be a pivot element, i.e. a sequence of a basic and a non-basic variable. If a phase I of the 2-phase simplex algorithm was started explicitly (see `linopt::Transparent::phaseI_tableau`) and the current tableau belongs to a feasible solution the suggestion will be the string `"linopt::Transparent::phaseII_tableau"`. At the end of the calculation the 'suggestion' is the identifier `OPTIMAL`.
- The result of `linopt::Transparent::suggest` can be influenced if the global identifier `OPTIMAL` has a value. For this reason the identifier `OPTIMAL` is protected.

Example 1. We have a look at a linear program where the ordinary simplex tableau of the given problem is not the last tableau during the computation of the simplex algorithm. Looking at the ordinary simplex tableau we see that the element of the slk[2]-labeled row and the x-labeled column is a pivot element:

```
>> k := [{3*x + 4*y - 3*z <= 23, 5*x - 4*y - 3*z <= 10,
          7*x + 4*y + 11*z <= 30}, -x + y + 2*z, NonNegative]:
   t := linopt::Transparent(k);
   linopt::Transparent::suggest(t)

   +-                                                        -
+
   |  "linopt", "restr", slk[1], slk[2], slk[3],  z,  x,  y  |
   |                                                          |
   |    "obj",     0,       0,      0,      0,    2, -1,  1  |
   |                                                          |
   |   slk[1],     30,      1,      0,      0,   11,  7,  4  |
   |                                                          |
   |   slk[2],     10,      0,      1,      0,   -3,  5, -
4  |
   |                                                          |
   |   slk[3],     23,      0,      0,      1,   -3,  3,  4  |
   +-                                                       -
+

                            slk[2], x

>> delete k, t:
```

Example 2. Here the ordinary simplex tableau still contains the solution of the linear program if the linear objective function is to minimize (see `linopt::Transparent` for more information):

```
>> k := [[x+y>=-1, x+y<=3], x+2*y, NonNegative]:
   t := linopt::Transparent(k);
   linopt::Transparent::suggest(t)

         +-                                          -+
         |  "linopt", "restr", slk[1], slk[2],  x,  y  |
         |                                             |
         |    "obj",     0,       0,      0,    1,  2  |
         |                                             |
         |   slk[1],     1,       1,      0,   -1, -1  |
         |                                             |
         |   slk[2],     3,       0,      1,    1,  1  |
         +-                                          -+

                             OPTIMAL
```

```
>> delete k, t:
```

Example 3. Here we explicitly start the first phase of the simplex algorithm. If we want a solution of the original linear program we have to apply the second phase of the simplex algorithm:

```
>> k := [{3*x + 4*y - 3*z <= 23, 5*x -4*y -3*z <= 10,
          7*x + 4*y + 11*z <= 30}, -x + y + 2*z, NonNegative]:
   t := linopt::Transparent(k):
   t := linopt::Transparent::phaseI_tableau(t):
   t := linopt::Transparent::simplex(t):
   linopt::Transparent::suggest(t)

              "linopt::Transparent::phaseII_tableau"

>> delete k, t:
```

Background:

- References:

  Papadimitriou, Christos H; Steiglitz, Kenneth: Combinatorial Optimization; Algorithms and Complexity. Prentice-Hall, 1982.

  Nemhauser, George L; Wolsey, Laurence A: Integer and Combinatorial Optimization. New York, Wiley, 1988.

  Salkin, Harvey M; Mathur, Kamlesh: Foundations of Integer Programming. North-Holland, 1989.

  Neumann, Klaus; Morlock, Martin: Operations-Research. Munich, Hanser, 1993.

  Duerr, Walter; Kleibohm, Klaus: Operations Research; Lineare Modelle und ihre Anwendungen. Munich, Hanser, 1992.

  Suhl, Uwe H: MOPS - Mathematical OPtimization System. European Journal of Operational Research 72(1994)312-322. North-Holland, 1994.

  Suhl, Uwe H; Szymanski, Ralf: Supernode Processing of Mixed Integer Models. Boston, Kluwer Academic Publishers, 1994.

Changes:

- `linopt::Transparent::suggest` is a new function.
- The linopt library was completely rewritten in MuPAD version 2.0.

`linopt::Transparent::userstep` – perform a user defined simplex step

`linopt::Transparent::userstep(tableau, basvar, nonbasbar)` performs a user defined simplex step in the `tableau` with the pivot element defined by `basvar` and `nonbasvar`.

Call(s):

- `linopt::Transparent::userstep(tableau, basvar, nonbasbar)`

Parameters:

`tableau`	—	a simplex tableau of domain type `linopt::Transparent`
`basvar`	—	a basic variable represented by an identifier that has to leave the basis
`nonbasvar`	—	a non-basic variable represented by an identifier that has to enter the basis

Return Value: a simplex tableau of domain type `linopt::Transparent`.

Related Functions: `linopt::Transparent`, `linopt::Transparent::autostep`, `linopt::Transparent::convert`, `linopt::Transparent::dual_prices`, `linopt::Transparent::phaseI_tableau`, `linopt::Transparent::result`, `linopt::Transparent::simplex`, `linopt::Transparent::suggest`

Details:

- `linopt::Transparent::userstep` returns the next simplex tableau which is calculated by the user defined simplex step.

Example 1. We execute the simplex step given by the pivot element `(slk[1],x)`:

```
>> k := [[x + y >= 2], x, NonNegative]:
   t:= linopt::Transparent(k);
   linopt::Transparent::userstep(t, slk[1], x)
```

```
+-                                      -+
|  "linopt", "restr", slk[1],  x,  y  |
|                                       |
|    "obj",     0,       0,    1,  0  |
|                                       |
|   slk[1],     -2,      1,   -1, -1  |
+-                                      -+

+-                                     -+
|  "linopt", "restr", slk[1], x,  y  |
|                                      |
|    "obj",     -2,      1,   0, -1  |
|                                      |
|      x,       2,      -1,   1,  1  |
+-                                     -+
```

Example 2. If we specify a wrong pivot element, we will get an error message:

```
>> k := [{3*x + 4*y - 3*z <= 23, 5*x - 4*y - 3*z <= 10,
          7*x + 4*y + 11*z <= 30}, -x + y + 2*z, NonNegative]:
   t:= linopt::Transparent(k);
   linopt::Transparent::userstep(t, x, y)

   +-                                                        -
+
   |  "linopt", "restr", slk[1], slk[2], slk[3],  z,  x,  y  |
   |                                                          |
   |    "obj",     0,       0,      0,      0,    2, -1,  1  |
   |                                                          |
   |   slk[1],     30,      1,      0,      0,   11,  7,  4  |
   |                                                          |
   |   slk[2],     10,      0,      1,      0,   -3,  5, -
4  |
   |                                                          |
   |   slk[3],     23,      0,      0,      1,   -3,  3,  4  |
   +-                                                        -
+
 Error:  No correct pivot element specified  [linopt::Transpare\
 nt::userstep]

>> delete k, t:
```

Background:

- References:

Papadimitriou, Christos H; Steiglitz, Kenneth: Combinatorial Optimization; Algorithms and Complexity. Prentice-Hall, 1982.

Nemhauser, George L; Wolsey, Laurence A: Integer and Combinatorial Optimization. New York, Wiley, 1988.

Salkin, Harvey M; Mathur, Kamlesh: Foundations of Integer Programming. North-Holland, 1989.

Neumann, Klaus; Morlock, Martin: Operations-Research. Munich, Hanser, 1993.

Duerr, Walter; Kleibohm, Klaus: Operations Research; Lineare Modelle und ihre Anwendungen. Munich, Hanser, 1992.

Suhl, Uwe H: MOPS - Mathematical OPtimization System. European Journal of Operational Research 72(1994)312-322. North-Holland, 1994.

Suhl, Uwe H; Szymanski, Ralf: Supernode Processing of Mixed Integer Models. Boston, Kluwer Academic Publishers, 1994.

Changes:

- `linopt::Transparent::userstep` is a new function.
- The linopt library was completely rewritten in MuPAD version 2.0.