

Namespace Methods

Namespaces

[Methods.Enums](#)

[Methods.Interfaces](#)

[Methods.MathObjects](#)

[Methods.Models](#)

[Methods.Solvers](#)

Namespace Methods.Enums

Enums

[ConstraintType](#)

Represents the type of a constraint in a LPP.

Enum ConstraintType

Namespace: [Methods.Enums](#)

Assembly: Methods.dll

Represents the type of a constraint in a LPP.

```
public enum ConstraintType
```

Fields

`Equal = 2`

Equality constraint.

`GreaterThanOrEqual = 1`

Greater than or equal to constraint.

`LessThanOrEqual = 0`

Less than or equal to constraint.

Namespace Methods.Interfaces

Interfaces

[ILinearSolver](#)

Interface that represents the essential methods for solving Linear Programming Problems (LPPs).

Interface ILinearSolver

Namespace: [Methods.Interfaces](#)

Assembly: Methods.dll

Interface that represents the essential methods for solving Linear Programming Problems (LPPs).

```
public interface ILinearSolver
```

Methods

CalculateReducedCosts()

Calculates the reduced costs (delta values) for the current simplex tableau.

```
void CalculateReducedCosts()
```

IsOptimal()

Determines whether the current solution is optimal.

```
bool IsOptimal()
```

Returns

[bool](#)

true if the problem has an optimal solution; otherwise, **false**.

IsUnbounded()

Determines whether the LPP has a solution.

```
bool IsUnbounded()
```

Returns

`bool` 

`true` if the problem has no solution; otherwise, `false`.

Pivot()

Finds the pivot row and column for the next step of solving.

```
void Pivot()
```

Solve()

Performs the steps required to solve the linear programming problem.

```
void Solve()
```

Namespace Methods.MathObjects

Classes

[Constraint](#)

Class that represent a constraint parts

[LinearProgrammingProblem](#)

Represents a LPP.

Class Constraint

Namespace: [Methods.MathObjects](#)

Assembly: Methods.dll

Class that represent a constraint parts

```
public class Constraint : ICloneable
```

Inheritance

[object](#) ← Constraint

Implements

[ICloneable](#)

Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Properties

Coefficients

List of coefficients.

```
public List<string> Coefficients { get; set; }
```

Property Value

[List](#) <[string](#)>

RightHandSide

Represents the expression on the right-hand side of the constraint.

```
public string RightHandSide { get; set; }
```


Property Value

[string](#)

Type

Represents the type of constraint, such as greater than or equal, less than or equal, or equal.

```
public ConstraintType Type { get; set; }
```

Property Value

[ConstraintType](#)

Methods

Clone()

Creates a clone of the current object.

```
public object Clone()
```

Returns

[object](#)

A new instance of the [Constraint](#) object.

Class LinearProgrammingProblem

Namespace: [Methods.MathObjects](#)

Assembly: Methods.dll

Represents a LPP.

```
public class LinearProgrammingProblem : ICloneable
```

Inheritance

[object](#) ← LinearProgrammingProblem

Implements

[ICloneable](#)

Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Properties

ArtificialVariableCoefficients

Coefficients for artificial variables.

```
public List<string> ArtificialVariableCoefficients { get; set; }
```

Property Value

[List](#) <[string](#)>

Constraints

List of constraints defining the problem.

```
public required List<Constraint> Constraints { get; set; }
```

Property Value

[List](#) <[Constraint](#)>

IsMaximization

Indicates whether the problem is a maximization ([true](#)) or minimization ([false](#)).

```
public bool IsMaximization { get; init; }
```

Property Value

[bool](#)

ObjectiveFunctionCoefficients

Coefficients of the objective function.

```
public required List<string> ObjectiveFunctionCoefficients { get; set; }
```

Property Value

[List](#) <[string](#)>

SlackVariableCoefficients

Coefficients for slack variables.

```
public List<string> SlackVariableCoefficients { get; set; }
```

Property Value

[List](#) <[string](#)>

VariablesCount

Gets the total number of variables, including primary, slack, and artificial variables.

```
public int VariablesCount { get; }
```

Property Value

[int](#)

Methods

Clone()

Creates a copy of the linear programming problem.

```
public object Clone()
```

Returns

[object](#)

A clone of the current [LinearProgrammingProblem](#) object.

Namespace Methods.Models

Classes

[BranchCut](#)

Represents a parts of Gomory cut.

[ExpressionValue](#)

Represents a text expression for the table and its actual numeric value.

[GomoryHistory](#)

Represents the current state of finding an integer solution using the Gomory cut method.

[SimplexHistory](#)

Represents the history of the simplex algorithm solving process.

[SimplexStep](#)

Represents the current state of the simplex table during pivot selection.

[SimplexTable](#)

Represents a simplex tableau used during the simplex algorithm iterations.

Class BranchCut

Namespace: [Methods.Models](#)

Assembly: Methods.dll








Represents a parts of Gomory cut.

```
public class BranchCut
```

Inheritance

[object](#)  ← BranchCut

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#) 

Properties

CutExpression

List of Gomory cut values, calculated as $x_{20} - [x_{20}]$.

```
public List<Fraction> CutExpression { get; set; }
```

Property Value

[List](#)  <Fraction>

FractionalElements

Dictionary containing fractional elements and their integer parts.

```
public Dictionary<string, (Fraction intPart, Fraction valueOfOriginalFraction)>  
FractionalElements { get; set; }
```

Property Value

[Dictionary](#) <[string](#), (Fraction [intPart](#), Fraction [valueOfOriginalFraction](#))>

Remarks

Each entry maps a fractional variable name (e.g., y20) to a tuple, where **intPart** is the integer part [x20] and **valueOfOriginalFraction** is the original fraction value x20.

Class ExpressionValue

Namespace: [Methods.Models](#)

Assembly: Methods.dll

Represents a text expression for the table and its actual numeric value.

```
public class ExpressionValue
```

Inheritance

[object](#)  ← ExpressionValue

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#) 

Constructors

ExpressionValue(string, Fraction)

Represents a text expression for the table and its actual numeric value.

```
public ExpressionValue(string expressionText, Fraction value)
```

Parameters

expressionText [string](#) 

value Fraction

Properties

ExpressionText

Formatted text for the table, used in the Big M method to represent expressions like "7M - 11".


```
public string ExpressionText { get; set; }
```

Property Value

[string](#)

Value

Actual numeric value of the expression.

```
public Fraction Value { get; set; }
```

Property Value

Fraction

Class GomoryHistory

Namespace: [Methods.Models](#)

Assembly: Methods.dll








Represents the current state of finding an integer solution using the Gomory cut method.

```
public class GomoryHistory
```

Inheritance

[object](#)  ← GomoryHistory

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#) 

Properties

Cut

Current branch cut applied in the Gomory method.

```
public BranchCut Cut { get; set; }
```

Property Value

[BranchCut](#)

MaxFracValue

Maximum fractional part found from the optimal (non-integer) solution. The tuple contains the row index and the fractional value.

```
public (int rowIndex, Fraction value) MaxFracValue { get; set; }
```

Property Value

([int](#) [rowIndex](#), Fraction [value](#))

Steps

List of simplex steps taken during the Gomory cut process.

```
public List<SimplexStep> Steps { get; set; }
```

Property Value

[List](#) <[SimplexStep](#)>

Class SimplexHistory

Namespace: [Methods.Models](#)

Assembly: Methods.dll








Represents the history of the simplex algorithm solving process.

```
public class SimplexHistory
```

Inheritance

[object](#)  ← SimplexHistory

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#) 

Properties

ArtificialProblemProblem

LLP that including artificial variables.

```
public LinearProgrammingProblem? ArtificialProblemProblem { get; set; }
```

Property Value

[LinearProgrammingProblem](#)

InitialBasis

Initial basis variables with their values. Key — variable name; Value — RHS taken from the corresponding constraint.

```
public Dictionary<string, string> InitialBasis { get; set; }
```

Property Value

[Dictionary](#) <[string](#), [string](#)>

InitialLinearProgrammingProblem

Initial LPP before any transformations.

```
public LinearProgrammingProblem? InitialLinearProgrammingProblem { get; set; }
```

Property Value

[LinearProgrammingProblem](#)

OptimalTable

Optimal simplex table.

```
public SimplexTable? OptimalTable { get; set; }
```

Property Value

[SimplexTable](#)

SlackVariableProblem

LLP that including slack variables.

```
public LinearProgrammingProblem? SlackVariableProblem { get; set; }
```

Property Value

[LinearProgrammingProblem](#)

Steps

List of simplex steps performed during the solving process.

```
public List<SimplexStep> Steps { get; set; }
```

Property Value

[List](#) <[SimplexStep](#)>

Class SimplexStep


Namespace: [Methods.Models](#)

Assembly: Methods.dll








Represents the current state of the simplex table during pivot selection.

```
public class SimplexStep
```

Inheritance

[object](#)  ← SimplexStep

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#) 

Properties

PivotColumn

Index of the pivot column in the current simplex table.

```
public int PivotColumn { get; set; }
```

Property Value

[int](#) 

PivotRow

Index of the pivot row in the current simplex table.

```
public int PivotRow { get; set; }
```

Property Value

[int](#)

Table

Simplex tableau at the current step.

```
public required SimplexTable Table { get; set; }
```

Property Value

[SimplexTable](#)

Class SimplexTable

Namespace: [Methods.Models](#)

Assembly: Methods.dll

Represents a simplex tableau used during the simplex algorithm iterations.

```
public class SimplexTable : ICloneable
```

Inheritance

[object](#) ← SimplexTable

Implements

[ICloneable](#)

Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Properties

ColumnVariables

Dictionary of column variables, where the key is the variable name (e.g., "A0"), and the value is the corresponding coefficient from the objective function.

```
public Dictionary<string, string> ColumnVariables { get; set; }
```

Property Value

[Dictionary](#) <[string](#), [string](#)>

DeltaRow

Delta row values representing the results after computing reduced costs.

```
public List<ExpressionValue> DeltaRow { get; set; }
```

Property Value

[List](#) <[ExpressionValue](#)>

RowVariables

Dictionary of row variables, where the key is the variable name (e.g., "x1"), and the value is the corresponding coefficient from the objective function.

```
public Dictionary<string, string> RowVariables { get; set; }
```

Property Value

[Dictionary](#) <[string](#), [string](#)>

ThetaRow

Theta row values used to identify pivot elements during the Gomory method.

```
public List<string> ThetaRow { get; set; }
```

Property Value

[List](#) <[string](#)>

Values

Numerical values of the simplex tableau.

```
public Fraction[,] Values { get; set; }
```

Property Value

Fraction[,]

Methods

Clone()

Creates a clone of the current object.

```
public object Clone()
```

Returns

[object](#)

A new instance of the [SimplexTable](#) object.

Namespace Methods.Solvers

Classes

[GomorySolver](#)

Represents a solver for the Gomory Cutting Plane method.

[SimplexSolver](#)

Solver for Linear Programming Problems using the Simplex method.

Class GomorySolver


Namespace: [Methods.Solvers](#)

Assembly: Methods.dll

Represents a solver for the Gomory Cutting Plane method.

```
public class GomorySolver : ILinearSolver
```








Inheritance

[object](#)  ← GomorySolver

Implements

[ILinearSolver](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#) 

Constructors

GomorySolver(SimplexTable, LinearProgrammingProblem)

Represents a solver for the Gomory Cutting Plane method.

```
public GomorySolver(SimplexTable Table, LinearProgrammingProblem Problem)
```

Parameters

Table [SimplexTable](#)

The simplex table obtained after solving by the Simplex method.

Problem [LinearProgrammingProblem](#)

The initial linear programming problem.

Properties

GomoryHistory

History of each Gomory cut step.

```
public List<GomoryHistory> GomoryHistory { get; set; }
```

Property Value

[List](#) [<GomoryHistory>](#)

Methods

CalculateReducedCosts()

Calculates the reduced costs (delta values) for the current simplex tableau.

```
public void CalculateReducedCosts()
```

IsOptimal()

Determines whether the current solution in the simplex table is optimal.

```
public bool IsOptimal()
```

Returns

[bool](#)

true if the solution is optimal; otherwise, **false**.

Remarks

A solution is optimal if all values in the first column are non-negative.

IsUnbounded()

Determines whether the LPP has a solution. If the chosen row does not contain fractional values, no solution is found.

```
public bool IsUnbounded()
```

Returns

[bool](#)

`true` if the problem has no solution; otherwise, `false`.

Pivot()

Finds the pivot row and column for the next solving step.

```
public void Pivot()
```

Remarks

The pivot row is determined by the minimum value in column A0. Then, the theta row is calculated as $|\Delta * x_{ij}|$, and the pivot column is chosen by the minimum theta value. After that, the reduced costs are recalculated.

Exceptions

[InvalidOperationException](#)

Arises if the pivot column is not found.

Solve()

Performs the steps required to solve the LPP.

```
public void Solve()
```

Remarks

First, checks whether the current solution contains non-integer values or is not optimal.
If the solution is not optimal, calls the pivot method and repeats the check.
If it is optimal, verifies whether a feasible solution exists.
Then, finds the most fractional value in the solution, constructs the Gomory cut,
adds it to the simplex table, and determines the pivot element.

Exceptions

[InvalidOperationException](#) 

"Arises if no solution exists or the given initial values are optimal."

Class SimplexSolver


Namespace: [Methods.Solvers](#)

Assembly: Methods.dll

Solver for Linear Programming Problems using the Simplex method.

```
public class SimplexSolver : ILinearSolver
```








Inheritance

[object](#)  ← SimplexSolver

Implements

[ILinearSolver](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#) 

Constructors

SimplexSolver(LinearProgrammingProblem)

Solver for Linear Programming Problems using the Simplex method.

```
public SimplexSolver(LinearProgrammingProblem problem)
```

Parameters

problem [LinearProgrammingProblem](#)

The LPP to be solved.

Properties

SimplexHistory

Stores the history of the Simplex solving process.

```
public SimplexHistory SimplexHistory { get; set; }
```

Property Value

[SimplexHistory](#)

Table

The current Simplex table used during the solution process.

```
public SimplexTable Table { get; set; }
```

Property Value

[SimplexTable](#)

Methods

CalculateReducedCosts()

Calculates the reduced costs (delta values) for the current simplex tableau.

```
public void CalculateReducedCosts()
```

IsOptimal()

Determines whether the current solution in the simplex table is optimal.

```
public bool IsOptimal()
```

Returns

[bool](#)

`true` if the solution is optimal; otherwise, `false`.

Remarks

The solution is optimal if all values in the delta row are non-negative when maximizing the objective function, or all values are non-positive when minimizing.

IsUnbounded()

Determines whether the LPP has a solution.

```
public bool IsUnbounded()
```

Returns

[bool](#)

`true` if the problem has no solution; otherwise, `false`.

Remarks

Returns true if the basis still contains artificial variables even though the delta row indicates an optimal solution.

Pivot()

Finds the pivot row and column for the next solving step.

```
public void Pivot()
```

Remarks

The pivot column is determined by the minimum or maximum value, depending on the objective function. Then, in the pivot column, the minimum positive row value is found and selected as the pivot row. Finally, recalculates all table values accordingly.

Exceptions

[InvalidOperationException](#)

Thrown if the pivot row cannot be determined (all values in the pivot column are less than or equal to 0).

Solve()

Performs the steps required to solve the linear programming problem.

```
public void Solve()
```

Remarks

First, if the right-hand side (RHS) of any constraint is less than 0, the constraints are converted into canonical form. Then, slack and artificial variables (if needed) are added. After that, the initial simplex table is initialized. The solver proceeds by calculating reduced costs, checking for solution existence, determining if the solution is optimal, and searching for a pivot element. Finally, artificial variables are removed if they were added.

Exceptions

[InvalidOperationException](#) 

Thrown if artificial variables remain in the basis after solving.