OptVerse 求解器用户手册

发行版本 *2.5.1*

OptVerse 求解器团队

2025 年 07 月 23 日

目录:

CHAPTER 1

OptVerse(天筹) 求解器简介

OptVerse 天筹数学优化求解器是一款由华为云推出的高效通用数学规划求解器。本用户手册简要介绍了 OptVerse 求解器的功能概述、安装说明和使用帮助。OptVerse 求解器可在 Linux 环境下使用，并支持以下调用方式：

- 命令行方式: 读取标准格式模型文件进行求解, 支持常用参数配置。
- 多语言 API: 目前 OptVserse 提供*C++ API* , *Python API* 和*Java API* 接口。

1.1 概述

现阶段 OptVerse 求解器提供线性规划问题 (linear programming, LP)，混合整数线性规划问题 (mixed integer linear programming, MILP) 问题和二次约束二次规划 (quadratically constrained quadratic program, QCQP) 问题的求解能力，并致力于支持更多类型的优化问题求解。

考虑（混合整数）线性规划的一般形式：

$$\begin{array}{ll} \text{minimize} & c_0 + c^T x \\ \text{subject to} & L \leq Ax \leq U \\ & l \leq x \leq u \\ & x_i \in \mathbb{Z}, i \in \mathrm{I} \end{array}$$

这里

$$\begin{array}{l} c_0 \in \mathbb{R} \\ c, x, l, u \in \mathbb{R}^n \\ L, U \in \mathbb{R}^m \\ A \in \mathbb{R}^{m \times n} \end{array}$$

而且 I 为指定变量 x 取整的下标集。

OptVerse 求解器提供如下求解算法：

- 对于 LP 问题，OptVerse 求解器提供原始/对偶单纯形 (primal/dual simplex method) 和内点算法 (interior point method)；
- 对于满足特定结构特征的网络流规划问题, OptVerse 求解器会自动选择网络流单纯形法 (network simplex method) 进行求解；
- 对于混合整数规划问题 (集合 I 非空)，OptVerse 求解器主要通过分支定界方法 (branch and bound) 来求解，并融合了业界多种启发式策略用以提高求解效率；
- 对于二次约束二次规划问题 (目标或约束中包含二次项 $x'Qx$), OptVerse 求解器会识别并调用内点算法 (interior point method) 进行求解。

CHAPTER 2

安装说明

2.1 支持平台

OptVerse 目前支持 Linux 系统平台，在以下系统测试可正常运行：

- **Ubuntu**: Ubuntu 18.04, Ubuntu 20.04, Ubuntu 22.04
- **CentOS**: CentOS 8, CentOS 9
- 低版本 Linux 系统需要满足 *glibc >= 2.33* 和 *libstdc >= 6.0.26*
- Python 版本需要满足在 3.8 和 3.11 之间

2.2 获取渠道

2.2.1 线下版本

- **OptVerse 软件安装包**: 请登录 华为云天筹求解器官网 订阅 SDK 服务后下载 OptVerse 软件安装包。

2.3 安装指南

2.3.1 线下版本安装

1. 将 `optverse` 求解器安装包 `optverse.x.x.x.tar.gz` 拷贝到指定路径 (如 `/opt/`). 以下假设路径为 `<installdir>` .
2. 解压缩安装包：

```
tar -xzvf optverse.x.x.x.tar.gz
```

这将创建一个新的路径 `<installdir>/optverse` ，路径结构为：

```
./
├── bin/
│   └── optverse
├── dist/
│   ├── optvpy-X.X.X-cp310-cp310-linux_x86_64.whl
│   ├── optvpy-X.X.X-cp311-cp311-linux_x86_64.whl
│   ├── optvpy-X.X.X-cp38-cp38-linux_x86_64.whl
│   └── optvpy-X.X.X-cp39-cp39-linux_x86_64.whl
├── examples/
│   ├── c++/
│   │   ├── feasibility-relax.cpp
│   │   ├── lp-sensitivity-analysis.cpp
│   │   ├── lp1.cpp
│   │   ├── lp2.cpp
│   │   ├── lp3.cpp
│   │   ├── milp1.cpp
│   │   └── milp2.cpp
│   ├── CMakeLists.txt
│   ├── data/
│   │   ├── lp-example.lp
│   │   ├── milp-example.mps
│   │   └── testNetwork.mps
│   └── python/
│       ├── feasibility-relax.py
│       ├── lp-sensitivity-analysis.py
│       ├── lp1.py
│       ├── lp2.py
│       ├── lp3.py
│       ├── milp1.py
│       └── milp2.py
├── include/
│   └── optv_c++.h
└── lib/
   ├── libgfortran.so.4
   ├── libiomp5.so
   ├── liboptverse.so
   └── libquadmath.so.0
```

3. 进入 <installdir>/optverse/bin 路径，执行 ./optverse --version，如果显示以下信息则说明 OptVerse 求解器已成功安装：

```
OptVerse Optimizer version 2.5.1 (Internal Release)
Copyright (c) Huawei Technologies Co., Ltd. 2022-2023. All rights reserved.
```

2.3.2 Python SDK 安装指南

如果需要天筹求解器 Python 版本 SDK，可通过安装包中的 wheel 文件进行安装：

```
pip3 install optvpy-x.x.x-cp3x-cp3xm-linux_x86_64.whl
```

2.3.3 使用许可获取

SDK 需要使用许可 (license) 才能正常使用，请登录 华为云天筹求解器官网 订阅 SDK 服务后下载 License 文件。

2.3.4 License 使用

- 把 license 文件放在某个目录下，如 `/opt/Optverse.xml`
- 将 `OPTV_LICENSE_FILE` 环境变量指向 license 文件: `export OPTV_LICENSE_FILE=/opt/Optverse.xml`
- 将 `OPTVERSE_HOME` 环境变量指向求解器安装路径: `export OPTVERSE_HOME=<installdir>/optverse`

备注： 将此变量导出到.bashrc 文件中，可方便多次连续调用。

```
echo "export OPTV_LICENSE_FILE=/opt/Optverse.xml" >> ~/.bashrc
echo "export OPTVERSE_HOME=<installdir>/optverse" >> ~/.bashrc
source ~/.bashrc
```

执行如下命令查看 SDK 是否可以正常使用。.. code-block:: bash

    cd <installdir>/optverse/bin ./optverse ../examples/data/lp-example.lp

2.3.5 Docker 环境使用

当 OptVerse 软件安装包运行在 Docker 容器环境中时，若需正常使用 license 授权功能，须配置容器与宿主机共享网络栈。

使用 Docker run 命令时，必须显式指定 `--net=host` 参数，示例：

```
docker run --net=host [其他参数] 镜像名称
```

CHAPTER 3

命令行工具

3.1 使用方法

3.1.1 显示帮助及版本信息

用法：

```
optverse [flag]    # 显示帮助或版本信息
```

可选 flag 详见参数说明。

3.1.2 求解问题

用法：

```
optverse [options]* filename # 设置求解参数，读取模型文件并行求解
```

可选求解参数列表见参数说明。当前支持的文件格式有：

- `.mps`, `.lp` 文件格式
- `.gz`, `.z`, `.Z` 压缩文件格式

有关 `.mps` 格式的详细说明可参考*MPS 格式文件说明*，有关 `.lp` 格式的详细说明可参考*LP 格式文件说明*。

3.2 参数说明

表 1: OptVerse 命令行参数

选项	作用	类型	取值范围	默认值
-h/–help	显示帮助信息	-	-	-
-v/–version	显示版本信息	-	-	-
–method	求解算法	`int`	$[0, 1, 2]$, 0=auto, 1=primal, 2=dual	0
–timeLimit	求解时间上限(秒)	`double`	$[0.0, 1.79769 \times 10^{308}]$	1.79769×10^{308}
–memLimit	求解内存上限(MB)	`int`	$[100, 2147483647]$	2147483647
–threads	求解线程数	`int`	[1, platform threads]	平台物理核数
–gap	MIP 目标 gap 值	`double`	$[0.0, 1.79769 \times 10^{308}]$	0.0001
–logFile	求解日志路径	`string`	-	`./optv.log`
–resultFile	解文件路径	`string`	-	`./<filename>.sol`

- 如果用户没有指定任何输入参数，将默认返回帮助信息。
- 输入问题模型文件只能作为最后一个输入参数。
- 设置 `method` 为 `0`, `1`, `2` 将分别调用原始、对偶或内点法求解 LP 问题，或 MILP 问题的首个 LP 松弛问题。
- 如输入问题为网络流规划问题, OptVerse 求解器将自动调用网络流单纯形法（network simplex method）进行求解。

3.3 使用示例

- 不输入任何参数，默认显示帮助信息

```
./optverse
```

- 显示帮助信息

```
./optverse -h
./optverse --help
```

- 显示版本信息

```
./optverse -v
./optverse --version
```

- 按默认设置求解问题文件 `test-problem.mps` 中的问题

```
./optverse test-problem.mps
```

- 求解问题文件 `test-problem.mps`, 指定日志文件为 `./test-problem.log`, 结果文件为 `./test-problem.sol`

```
./optverse --logFile=./test-problem.log --resultFile=./test-problem.sol␣
↪test-problem.mps
```

- 求解压缩格式问题文件 test-problem.mps.gz 并设置时间上限 7200 秒，内存使用上限 32GB，目标 gap 值 0.01%，使用线程数 4 并行求解

```
./optverse --timeLimit=7200 --memLimit=32768 --gap=0.0001 --threads=4␣
↪test-problem.mps.gz
```

3.4 日志解析

3.4.1 LP 求解日志解析

求解 LP 问题文件 lp-example.lp 使用默认参数进行求解，指定结果文件为 lp-example.sol：

```
./optverse --resultFile=lp-example.sol examples/data/lp-example.lp
```

所得日志如下：

```
Set parameter "logFile" to "optv.log"
Set parameter "resultFile" to "lp-example.sol"
OptVerse Optimizer version "2.5.1"
Copyright (c) Huawei Technologies Co., Ltd. 2022-2023. All rights reserved.

Read problem /home/solver/optverse/examples/data/lp-example.lp
Read time: 0.00s

Optimize an LP model
  8 rows, 20 columns and 36 nonzeros

Presolve problem
Presolve time: 0.00s
After presolve:
  4 rows, 12 columns and 12 nonzeros

     Time    Iteration     Objective   Primal Num.Inf.  Dual Num.Inf.
     0.0s            0  0.000000e+00                4              0
     0.0s            4  1.380000e+02                0              0

Postsolving
Postsolve finished

     Time    Iteration     Objective   Primal Num.Inf.  Dual Num.Inf.
     0.0s            0  1.380000e+02                0              0

Solve results
  Status               Optimal solution found
  Objective            1.380000000000e+02
  Simplex iteration    4
  Time                 0.01

Write best solution /home/solver/optverse/lp-example.sol
```

上述日志包含了如下信息：

- 参数设置，包含: logFile、resultFile，可用参数请参考*OptVerse 参数*

- 版本和版权信息
- 问题读取时间
- 问题信息，包含行数、列数、非零元个数
- 预处理时间及预处理后的问题规模，包含新行数、列数和非零元个数
- 单纯形算法迭代日志，如时间、迭代步数，当前目标值，原始及对偶可行性的违背量
- 后处理信息
- 求解结果概览，包含
  - 求解状态，如 `Optimal solution found`, `Problem is infeasible`, `Problem is unbounded`, `Time limit reached`, `Numerical error`, `Symbolic error`, `Unknown`
  - 最优目标值
  - 迭代总数
  - 总耗时
- 运行期间的报警错误信息

3.4.2 MILP 求解日志解析

使用默认参数求解 MILP 问题文件 `milp-example.mps`，指定结果文件为 *milp-example.sol*：

```
./optverse --resultFile=milp-example.sol examples/data/milp-example.mps
```

所得日志如下：

```
Set parameter "logFile" to "optv.log"
Set parameter "resultFile" to "milp-example.sol"
OptVerse Optimizer version "2.5.1" (Internal Release)
Copyright (c) Huawei Technologies Co., Ltd. 2022-2023. All rights reserved.

Read problem /home/solver/optverse/examples/data/milp-example.mps
Read time: 0.00s

Optimize an MILP model
  8 rows, 20 columns (20 binary, 0 integer, 0 continuous) and 36 nonzeros

Presolve problem
Presolve time: 0.00s
After presolve:
  8 rows, 20 columns (20 binary, 0 integer, 0 continuous) and 36 nonzeros

Start parallel solving, using up to 6 threads

    Time     Solved       Open       LPIter     BestBound       BestSol        Gap
  H   0.0s          0          0          0   0.000000e+00   2.980000e+02     ␣
↪100%
  H   0.0s          0          0          0   1.380000e+02   2.880000e+02   52.
↪08%
      0.0s          0          0          0   1.380000e+02   2.880000e+02   52.
↪08%
```

```
      0.0s          0           0           9    2.420000e+02    2.880000e+02    15.
↪97%
  H   0.0s          1           0           9    2.420000e+02    2.420000e+02     0.
↪00%
      0.0s          1           0           9    2.420000e+02    2.420000e+02     0.
↪00%

Solve results
  Status                Optimal solution found
  Best solution         2.4200000000000e+02
  Best bound            2.4200000000000e+02
  Gap                   0.0000%
  Node                  1
  LP iteration          9
  Time                  0.01

Write best solution /home/solver/optverse/milp-example.sol
```

上述日志包含了如下信息:

- 参数设置, 包含: logFile、resultFile，可用参数请参考*OptVerse 参数*。
- 版本信息
- 读取问题时间
- 原问题规模
- 预处理时间和预处理后问题规模
- 开始串行、并行求解（线程数），如
  - Start sequential solving
  - Start parallel solving, using up to 4 threads
- 第一列（在 Time 列之前）仅可能出现 H 和 *
- Time, Solved, Open, LPIter, BestBound, BestSol, 和 Gap 信息
- 求解结果概览，包含
  - 求解状态，仅可能出现 `Optimal solution found`, `Time limit reached`, `Memory limit reached`, 或 `Problem is infeasible`
  - 最优解和最优界
  - Gap
  - 节点个数
  - LP 迭代总数
  - 总耗时
- 运行时期间的报警错误信息

CHAPTER 4

OptVerse 求解器快速入门

本章提供了一些基本的示例，用于演示如何调用 OptVerse 各语言接口进行求解数学规划问题建模和求解。

4.1 C++ 接口

本节将通过一个简单的 C++ 示例来演示 OptVerse 求解器 C++ 接口的使用，包括

- 设置求解器参数
- 构建模型
- 将构建模型写入文件
- 求解模型
- 打印最优解信息

待求解问题的数学公式如下所示：

$$\begin{aligned} \text{minimize} \quad & -x - 14y - 6z \\ \text{subject to} \quad & 2x + y + z \leq 3 \\ & 3y + z \leq 6 \\ & 0 \leq x \leq 1,\ 0 \leq y,\ 0 \leq z \leq 3 \\ & x,\ y \in \mathbb{Z}, z \in \mathbb{R}. \end{aligned}$$

```
/*
 * Copyright (c) Huawei Technologies Co., Ltd. 2022-2023. All rights reserved.
 * Description: OptVerse c++ interface example - build and solve an MILP problem
 */

#include "optv_c++.h"
using namespace std;
```

(续下页)

(接上页)

```
int main()
{
    try {
        OPTVEnv env("milp1.log");
        // Before model construction: set parameters through environment
        env.Set(OPTVDblParam::TIME_LIMIT, 7200);

        OPTVModel model(env);
        // After model construction: set parameters through model
        model.Set(OPTVStrParam::RESULT_FILE, "milp1.sol");

        // Add variables to the model
        OPTVVar x = model.AddVar(0, 1, -1, OPTV_BINARY, "x");
        OPTVVar y = model.AddVar(0, OPTV_INF, -14, OPTV_INTEGER, "y");
        OPTVVar z = model.AddVar(0, 3, -6, OPTV_CONTINUOUS, "z");

        // Add constraint to the model
        OPTVConstr c0 = model.AddConstr(2 * x + y + z, -OPTV_INF, 3, "c0");
        OPTVConstr c1 = model.AddConstr(3 * y + z, -OPTV_INF, 6, "c1");

        model.Write("milp1.lp");
        model.Optimize();

        if (model.Get(OPTVIntAttr::SOL_COUNT) > 0) {
            cout << "Model Status: " << model.Get(OPTVIntAttr::STATUS) << endl;
            cout << "Best solution: " << model.Get(OPTVDblAttr::OBJ_VAL) << endl;

            cout << "x = " << x.Get(OPTVDblAttr::X) << endl;
            cout << "y = " << model.GetVar(1).Get(OPTVDblAttr::X) << endl;
            cout << "z = " << model.GetVarByName("z").Get(OPTVDblAttr::X) << endl;
            cout << "c0 = " << model.GetConstr(0).Get(OPTVDblAttr::X) << endl;
            cout << "c1 = " << model.GetConstrByName("c1").Get(OPTVDblAttr::X) <<␣
↪endl;
        } else {
            cout << "No feasible solution available!" << endl;
        }

    } catch (const OPTVException& e) {
        cout << e.GetErrorCode() << ": " << e.GetMessage() << endl;
    } catch (...) {
        cerr << "Unknown exception" << endl;
    }
    return 0;
}
```

4.1.1 包含头文件

要使用 C++ 接口，首先要包含头文件 `optv_c++.h`.

```
#include "optv_c++.h"
```

4.1.2 创建环境

程序最开始创建了一个空的环境对象，并将日志文件名提供给构造函数。

```
OPTVEnv env("milp1.log");
```

4.1.3 模型创建前配置求解参数

若需要在调用*OPTVModel* 构造函数 **前**修改求解器参数，用户必须通过环境对象设置这些参数。在本例中，我们展示了在创建模型对象前如何通过环境对象设置求解器的时限参数。

```
env.Set(OPTVDblParam::TIME_LIMIT, 7200);
```

4.1.4 模型创建后的参数配置

若需要在调用*OPTVModel* 构造函数 **后**修改求解器参数，用户则必须通过模型对象设置这些参数。在本例中，我们展示了如何通过模型对象配置求解日志文件参数。

```
model.Set(OPTVStrParam::RESULT_FILE, "milp1.sol");
```

4.1.5 创建模型

创建了环境后，下一步就是创建模型。每个 OptVerse 模型是一个优化问题，它由一组变量，一组约束和关联属性 (变量边界、目标系数、约束右边值等) 组成。

创建空模型

如果用户需要通过 API 一步一步构造模型，首先需要做的是创建一个空模型。

```
OPTVModel model(env);
```

从模型文件加载

如果把 `.mps`[1] 或 `.lp`[2] 文件路径当做第二个参数传入，该文件则被读取，相应的模型可以通过 API 求解或修改。

```
OPTVEnv env("fileRead.log");
OPTVModel(env, fileName);
```

更多细节请参考 `milp2.cpp`

[1] MPS 格式说明见*MPS 格式文件说明* 。
[2] LP 格式说明见*LP 格式文件说明* 。

向模型中添加变量

如果上一步创建了一个空模型，那么用户 **必须**向该模型中添加变量和约束后，才可以进行求解。同样，也可以向通过文件读取的模型增加变量或约束。方法*OPTVModel::AddVar()* 提供了向模型对象添加变量的功能，前两个参数是变量的下界和上界，第三个参数是目标系数，第四个参数是变量类型，可选值为: *OPTV_CONTINUOUS*、*OPTV_INTEGER* 或*OPTV_BINARY*，最后一个参数是变量的名称。

```
OPTVVar x = model.AddVar(0, 1, -1, OPTV_BINARY, "x");
OPTVVar y = model.AddVar(0, OPTV_INF, -14, OPTV_INTEGER, "y");
OPTVVar z = model.AddVar(0, 3, -6, OPTV_CONTINUOUS, "z");
```

如果所有创建的变量全部都是*OPTV_CONTINUOUS*，那么该模型就是一个线性规划问题，程序会自动调用线性规划求解算法进行求解。可以参考样例代码 `lp1.cpp`.

向模型中添加约束

添加完变量后，用户可以继续添加约束。OptVerse 的 API 支持双边约束表达式，即用户创建约束对象时需要同时提供该约束的上下界。第一个参数是约束表达式的类型，如*OPTVLinExpr*，接下来两个参数分别是约束的上、下界（左、右边值），最后一个参数为约束名称。

```
OPTVConstr c0 = model.AddConstr(2 * x + y + z, -OPTV_INF, 3, "c0");
OPTVConstr c1 = model.AddConstr(3 * y + z, -OPTV_INF, 6, "c1");
```

4.1.6 求解模型

模型构造及相应的参数配置完成后，用户可以通过调用*OPTVModel::Optimize()* 来求解模型。求解完成后，求解状态、最优解等属性值都被保存在*OPTVModel* 对象中。可以调用 OPTVModel::Optimize() 方法执行求解。

```
model.Optimize();
```

4.1.7 查询最优解

模型求解结束后，用户可以通过*OPTVModel::Get()* 方法来查询模型的不同属性和输出所得解。首先，模型的*OPTVIntAttr::SOL_COUNT* 属性提示了可行解是否成功获取；然后，模型求解状态可以通过*OPTVIntAttr::STATUS* 查询；而且目前最佳的目标值可以通过*OPTVDblAttr::OBJ_VAL* 获取；最后，以下示例展示了查询变量或约束的三种不同方法。

```
if (model.Get(OPTVIntAttr::SOL_COUNT) > 0) {
    cout << "Model Status: " << model.Get(OPTVIntAttr::STATUS) << endl;
    cout << "Best solution: " << model.Get(OPTVDblAttr::OBJ_VAL) << endl;

    cout << "x = " << x.Get(OPTVDblAttr::X) << endl;
    cout << "y = " << model.GetVar(1).Get(OPTVDblAttr::X) << endl;
    cout << "z = " << model.GetVarByName("z").Get(OPTVDblAttr::X) << endl;
    cout << "c0 = " << model.GetConstr(0).Get(OPTVDblAttr::X) << endl;
    cout << "c1 = " << model.GetConstrByName("c1").Get(OPTVDblAttr::X) << endl;
} else {
    cout << "No feasible solution available!" << endl;
}
```

4.1.8 编译和运行

OptVerse 的软件包中已经包含了本节介绍的样例程序，路径在 `${OPTV_HOME}/examples/c++`，同时提供了 CMake 文件用于在 Linux 平台下编译所有样例。

编译时只需要先进入到 `${OPTV_HOME}/examples` 文件夹，然后执行如下操作即可。

```
mkdir build && cd build
cmake ..
make -j $(nproc)
```

编译完成后，上述示例可以通过调用以下命令运行：

```
./milp1
```

如果需要求解特定的 `.mps` 或 `.lp` 格式文件所含模型，用户可以通过以下命令求解和查询模型目标值及状态：

```
./milp2 xxx.mps
```

4.2 Python 接口

本节将通过前一章节的*C++ 接口示例* 来演示 OptVerse 求解器 Python 接口的使用。该示例在 Python 环境下构建了一个模型，并对其进行求解，最后输出最优解。

```
"""
   Copyright (c) Huawei Technologies Co., Ltd. 2024. All rights reserved.
   Description: OptVerse python interface example - build and solve an MILP problem

   MILP model built in this example:
     min    - x - 14 y - 6 z
     s.t.   2 x +    y +   z <= 3
                  3 y +   z <= 6
     where x is binary, y is integer, z is continuous
              0 <= x <= 1
              0 <= y
              0 <= z <= 3
"""

from optvpy import *

if __name__ == "__main__":
    # Create an environment and set log file
    env = OPTVEnv("milp1.log")
    env.Set(OPTVDblParam.TIME_LIMIT, 7200)

    # Create an empty model
    model = OPTVModel(env)
    model.Set(OPTVStrParam.RESULT_FILE, "milp1.sol")

    # Add variables to the model
    x = model.AddVar(0, 1, -1, OPTV_BINARY, "x")
    y = model.AddVar(0, OPTV_INF, -14, OPTV_INTEGER, "y")
    z = model.AddVar(0, 3, -6, OPTV_CONTINUOUS, "z")
```

(续下页)

(接上页)

```
    # Add constraint to the model
    c0 = model.AddConstr(2 * x + y + z, -OPTV_INF, 3, "c0")
    c1 = model.AddConstr(3 * y + z, -OPTV_INF, 6, "c1")

    model.Write('milp1.lp')
    # Optimize the model
    model.Optimize()

    # Output solution
    print(f"Best solution: {model.Get(OPTVDblAttr.OBJ_VAL)}")
    print(f"x = {x.Get(OPTVDblAttr.X)}")
    print(f"y = {model.GetVar(1).Get(OPTVDblAttr.X)}")
    print(f"z = {model.GetVarByName('z').Get(OPTVDblAttr.X)}")
```

4.2.1 引入模块

要使用 Python 接口，首先要安装并引入模块 `optvpy`.

```
from optvpy import *
```

4.2.2 创建环境

第一步需要创建一个空的环境对象，然后将日志文件名提供给该构造函数。

```
# Create an environment and set log file
env = OPTVEnv("milp1.log")
```

4.2.3 模型创建前的参数配置

若需要在调用*OPTVModel* 构造函数 **前**修改求解器参数，用户必须通过环境对象设置这些参数。在本例中，我们展示了在创建模型对象前如何通过环境对象设置求解器的时限参数。

```
env.Set(OPTVDblParam.TIME_LIMIT, 7200)
```

4.2.4 模型创建后的参数配置

若需要在调用*OPTVModel* 构造函数 **后**修改求解器参数，用户则必须通过模型对象设置这些参数。在本例中，我们展示了如何通过模型对象配置求解日志文件参数。

```
model.Set(OPTVStrParam.RESULT_FILE, "milp1.sol")
```

4.2.5 创建模型

创建了环境后，下一步就是创建模型。每个 OptVerse 模型是一个优化问题，它由一组变量，一组约束和关联属性 (变量边界、目标系数、约束右边值等) 组成。

创建空模型

如果用户需要通过 API 一步一步构造模型，首先需要做的是创建一个空模型。

```
# Create an empty model
model = OPTVModel(env)
```

从模型文件加载

如果把 `.mps`[1] 或 `.lp`[2] 文件路径当做第二个参数传入，该文件则被读取，相应的模型可以通过 API 求解或修改。

```
OPTVEnv env("fileRead.log")
OPTVModel(env, fileName)
```

更多细节请参考 `milp2.py`.

向模型中添加变量

如果上一步创建了一个空模型，那么用户 **必须**向该模型中添加变量和约束后，才可以进行求解。同样，也可以向通过文件读取的模型增加变量或约束。*OPTVModel.AddVar()* 方法提供了向模型对象添加变量的功能，前两个参数是变量的下界和上界，第三个参数是目标系数，第四个参数是变量类型，可选值为: *OPTV_CONTINUOUS*, *OPTV_INTEGER*, 或*OPTV_BINARY*，最后一个参数是变量的名称。

```
# Add variables to the model
x = model.AddVar(0, 1, -1, OPTV_BINARY, "x")
y = model.AddVar(0, OPTV_INF, -14, OPTV_INTEGER, "y")
z = model.AddVar(0, 3, -6, OPTV_CONTINUOUS, "z")
```

如果所有创建的变量全部都是*OPTV_CONTINUOUS*，那么该模型就是一个线性规划问题，程序会自动调用线性规划求解算法进行求解。更多细节请参考样例代码 `lp1.py`.

向模型中添加约束

添加完变量后，用户可以继续添加约束。OptVerse 的 API 支持双边约束表达式，即用户创建约束对象时需要同时提供该约束的上下界。第一个参数是约束表达式的类型，如*OPTVLinExpr*，接下来两个参数分别是约束的上、下界（左、右边值），最后一个参数为约束名称。

```
# Add constraint to the model
c0 = model.AddConstr(2 * x + y + z, -OPTV_INF, 3, "c0")
c1 = model.AddConstr(3 * y + z, -OPTV_INF, 6, "c1")
```

[1] MPS 格式说明见*MPS 格式文件说明*。
[2] LP 格式说明见*LP 格式文件说明*。

4.2.6 求解模型

模型构造及相应的参数配置完成后，用户可以通过调用*OPTVModel.Optimize()* 来求解模型。求解完成后，求解状态、最优解等属性值都被保存在*OPTVModel* 对象中。

```
# Optimize the model
model.Optimize()
```

4.2.7 查询最优解

模型求解结束后，用户可以通过*OPTVModel.Get()* 方法来查询模型的不同属性和输出所得解。例如，目前最佳的目标值可以通过*OPTVDblAttr.OBJ_VAL* 查询，以下示例展示了查询变量或约束的三种不同方法。

```
# Output solution
print(f"Best solution: {model.Get(OPTVDblAttr.OBJ_VAL)}")
print(f"x = {x.Get(OPTVDblAttr.X)}")
print(f"y = {model.GetVar(1).Get(OPTVDblAttr.X)}")
print(f"z = {model.GetVarByName('z').Get(OPTVDblAttr.X)}")
```

4.2.8 运行示例

OptVerse 的软件包中已经包含了本节介绍的样例程序，路径在 `${OPTV_HOME}/examples/python`，只需进入 `${OPTV_HOME}/examples` 路径并执行

```
python3 milp1.py
```

如果需要求解特定的 `.mps` 或 `.lp` 格式文件所含模型，用户可以通过以下命令求解和查询模型目标值及状态:

```
python3 milp2.py xxx.mps
```

4.3 Java 接口

本节将通过前一章节的*C++ 接口示例* 来演示 OptVerse 求解器 Java 接口的使用。该示例在 Java 环境下构建了一个模型，并对其进行求解，最后输出最优解。

```
import com.huaweicloud.optv.*;

public class Milp1 {
    public static void main(String[] args) {
        try {
            OPTVEnv env = new OPTVEnv("Milp1.log");
            env.Set(OPTVDblParam.TIME_LIMIT, 7200);


            OPTVModel model = new OPTVModel(env);
            model.set(OPTVStrParam.RESULT_FILE, "Milp1.sol");

            // add variables to the model
            OPTVVar x = model.addVar(0, 1, -1, OPTV.OPTV_BINARY, "x");
```

(续下页)

(接上页)

```
            OPTVVar y = model.addVar(0, OPTV.OPTV_INF, -14, OPTV.OPTV_INTEGER, "y");
            OPTVVar z = model.addVar(0, 3, -6, OPTV.OPTV_CONTINUOUS, "z");

            // add constraint to the model
            OPTVConstr c0 = model.addConstr(OPTV.add(OPTV.add(OPTV.mul(2, x), y), z),␣
↪-OPTV.OPTV_INF, 3, "c0");
            OPTVConstr c1 = model.addConstr(OPTV.add(OPTV.mul(3, y), z), -OPTV.OPTV_
↪INF, 6, "c1");

            model.write("Milp1.lp");
            model.optimize();

            if (model.get(OPTVIntAttr.SOL_COUNT) > 0) {
                System.out.println("Best solution: " + model.get(OPTVDblAttr.OBJ_
↪VAL));

                System.out.println("x = " + x.get(OPTVDblAttr.X));
                System.out.println("y = " + model.getVar(1).get(OPTVDblAttr.X));
                System.out.println("z = " + model.getVarByName("z").get(OPTVDblAttr.
↪X));
            } else {
                System.out.println("No feasible solution available!");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

4.3.1 引入模块

使用 Java 接口前需要导入安装包 `com.huaweicloud.optv`.

```
import com.huaweicloud.optv.*;
```

4.3.2 创建环境

第一步需要创建一个空的环境对象，然后将日志文件名提供给该构造函数。

```
OPTVEnv env = new OPTVEnv("Milp1.log");
```

4.3.3 模型创建前的参数配置

若需要在调用*OPTVModel* 构造函数 **前**修改求解器参数，用户必须通过环境对象设置这些参数。在本例中，我们展示了在创建模型对象前如何通过环境对象设置求解器的时限参数。

```
env.Set(OPTVDblParam.TIME_LIMIT, 7200);
```

4.3.4 模型创建后的参数配置

若需要在调用*OPTVModel* 构造函数 **后**修改求解器参数，用户则必须通过模型对象设置这些参数。在本例中，我们展示了如何通过模型对象配置求解日志文件参数。

```
model.set(OPTVStrParam.RESULT_FILE, "Milp1.sol");
```

4.3.5 创建模型

创建了环境后，下一步就是创建模型。每个 OptVerse 模型是一个优化问题，它由一组变量，一组约束和关联属性 (变量边界、目标系数、约束右边值等) 组成。

创建空模型

如果用户需要通过 API 一步一步构造模型，首先需要做的是创建一个空模型。

```
OPTVModel model = new OPTVModel(env);
```

从模型文件加载

如果把 `.mps`[1] 或 `.lp`[2] 文件路径当做第二个参数传入，该文件则被读取，相应的模型可以通过 API 求解或修改。

```
OPTVEnv env("fileRead.log")
OPTVModel(env, fileName)
```

更多细节请参考 `Milp2.java`

向模型中添加变量

如果上一步创建了一个空模型，那么用户 **必须**向该模型中添加变量和约束后，才可以进行求解。同样，也可以向通过文件读取的模型增加变量或约束。*OPTVModel.addVar()* 方法提供了向模型对象添加变量的功能，前两个参数是变量的下界和上界，第三个参数是目标系数，第四个参数是变量类型，可选值为: *OPTV_CONTINUOUS*, *OPTV_INTEGER*, 或*OPTV_BINARY*，最后一个参数是变量的名称。

```
// add variables to the model
OPTVVar x = model.addVar(0, 1, -1, OPTV.OPTV_BINARY, "x");
OPTVVar y = model.addVar(0, OPTV.OPTV_INF, -14, OPTV.OPTV_INTEGER, "y");
OPTVVar z = model.addVar(0, 3, -6, OPTV.OPTV_CONTINUOUS, "z");
```

如果所有创建的变量全部都是*OPTV_CONTINUOUS*，那么该模型就是一个线性规划问题，程序会自动调用线性规划求解算法进行求解。更多细节请参考样例代码 `Lp1.java`.

[1] MPS 格式说明见*MPS 格式文件说明*。
[2] LP 格式说明见*LP 格式文件说明*。

向模型中添加约束

添加完变量后，用户可以继续添加约束。OptVerse 的 API 支持双边约束表达式，即用户创建约束对象时需要同时提供该约束的上下界。第一个参数是约束表达式的类型，如*OPTVLinExpr*，接下来两个参数分别是约束的上、下界（左、右边值），最后一个参数为约束名称。

```
// add constraint to the model
OPTVConstr c0 = model.addConstr(OPTV.add(OPTV.add(OPTV.mul(2, x), y), z), -OPTV.OPTV_
↪INF, 3, "c0");
OPTVConstr c1 = model.addConstr(OPTV.add(OPTV.mul(3, y), z), -OPTV.OPTV_INF, 6, "c1");
```

4.3.6 求解模型

模型构造及相应的参数配置完成后，用户可以通过调用*OPTVModel.optimize()* 来求解模型。求解完成后，求解状态、最优解等属性值都被保存在*OPTVModel* 对象中。

```
model.optimize();
```

4.3.7 查询最优解

模型求解结束后，用户可以通过*OPTVModel.get()* 方法来查询模型的不同属性和输出所得解。例如，目前最佳的目标值可以通过*OPTVDblAttr.OBJ_VAL* 查询，以下示例展示了查询变量或约束的三种不同方法。

```
System.out.println("Best solution: " + model.get(OPTVDblAttr.OBJ_VAL));

System.out.println("x = " + x.get(OPTVDblAttr.X));
System.out.println("y = " + model.getVar(1).get(OPTVDblAttr.X));
System.out.println("z = " + model.getVarByName("z").get(OPTVDblAttr.X));
```

4.3.8 运行示例

OptVerse 的软件包中已经包含了本节介绍的样例程序，路径在 `${OPTV_HOME}/examples/java`，只需进入 `${OPTV_HOME}/examples` 路径并执行编译

```
javac -cp .:${OPTV_HOME}/lib/optv-java-sdk.jar Milp1.java
```

编译完成后，可执行以下命令

```
java -cp .:.${OPTV_HOME}/lib/optv-java-sdk.jar Milp1
```

如果需要求解特定的 `.mps` 或 `.lp` 格式文件所含模型，用户可以通过以下命令求解和查询模型目标值及状态：

```
java -cp .:${OPTV_HOME}/lib/optv-java-sdk.jar Milp2
```

4.4 LP 不可行诊断及修复 C++ 接口

本节将通过一个简单的 C++ 示例来演示 OptVerse 求解器 C++ 接口下不可行模型修复功能，即松弛不可行问题使其获取可行性。

待求解问题的数学公式如下所示：

$$\begin{aligned} \text{minimize} \quad & x + y \\ \text{subject to} \quad & 2x + y \quad \leq 2 \\ & -3x + 3y \quad \leq -3 \\ & x - 2y \quad \leq -2 \\ & 0 \leq x, y \\ & x, y \in \mathbb{R}. \end{aligned} \tag{4.1}$$

```cpp
/*
 * Copyright (c) Huawei Technologies Co., Ltd. 2022-2023. All rights reserved.
 * Description: OptVerse c++ interface example - build and solve an LP problem
 */

#include "optv_c++.h"
using namespace std;

int main()
{
    try {
        OPTVEnv env("feasibility-relax.log");
        OPTVModel model(env);
        model.Set(OPTVStrParam::RESULT_FILE, "feasibility-relax.sol");

        // Add variables to the model, all variables are continuous
        OPTVVar x = model.AddVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "x");
        OPTVVar y = model.AddVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "y");

        // Add constraint to the model
        OPTVConstr c0 = model.AddConstr(2 * x + y, -OPTV_INF, 2, "c0");
        OPTVConstr c1 = model.AddConstr(-3 * x + 2 * y, -OPTV_INF, -3, "c1");
        OPTVConstr c2 = model.AddConstr(x - 2 * y, -OPTV_INF, -2, "c2");

        model.FeasRelax();

        cout << "L1 violation: " << model.Get(OPTVDblAttr::OBJ_VAL) << endl;
        cout << "c2 upper bound = " << c2.Get(OPTVDblAttr::UB) << endl;
    } catch (const OPTVException& e) {
        cout << e.GetErrorCode() << ": " << e.GetMessage() << endl;
    } catch (...) {
        cerr << "Unknown exception" << endl;
    }
    return 0;
}
```

4.4.1 包含头文件

要使用 C++ 接口，首先要包含头文件 optv_c++.h.

```
#include "optv_c++.h"
```

4.4.2 创建模型

首先初始化一个空的环境对象，提供给*OPTVModel* 构造函数。

```
OPTVEnv env("feasibility-relax.log");
OPTVModel model(env);
```

添加变量和约束

```
// Add variables to the model, all variables are continuous
OPTVVar x = model.AddVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "x");
OPTVVar y = model.AddVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "y");

// Add constraint to the model
OPTVConstr c0 = model.AddConstr(2 * x + y, -OPTV_INF, 2, "c0");
OPTVConstr c1 = model.AddConstr(-3 * x + 2 * y, -OPTV_INF, -3, "c1");
OPTVConstr c2 = model.AddConstr(x - 2 * y, -OPTV_INF, -2, "c2");
```

4.4.3 使用不可行修复功能

调用 LP 不可行修复 C++ 接口函数*OPTVModel::FeasRelax()* 即可使用不可行修复功能。

```
model.FeasRelax();
```

4.4.4 查询不可行修复结果

用户可以通过调用*model.Get(OPTVDblAttr::OBJ_VAL)* 查询原模型的不可行违背量的 L^1-范数，以获取不可行修复的结果。同时，约束界限所需修改可以通过查询相应约束的属性获取，如*c2.Get(OPTVDblAttr::UB)*.

```
cout << "L1 violation: " << model.Get(OPTVDblAttr::OBJ_VAL) << endl;
cout << "c2 upper bound = " << c2.Get(OPTVDblAttr::UB) << endl;
```

4.5 LP 不可行诊断及修复 Python 接口

对于前一节展示的不可行问题 (4.1)，我们接下来展示在 python 环境下的 LP 不可行修复功能。具体细节请参考*Python 接口快速入门* 或*Python API 使用参考*，这里不再赘述中间步骤。

```
from optvpy import *

if __name__ == "__main__":
    env = OPTVEnv("feasibility-relax.log")
    model = OPTVModel(env)
    model.Set(OPTVStrParam.RESULT_FILE, "feasibility-relax.sol")
    obj = 0.0

    # Add variables to the model, all variables are continuous
    x = model.AddVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "x")
    y = model.AddVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "y")

    # Add constraint to the model
    c0 = model.AddConstr(2 * x + y, -OPTV_INF, 2, "c0")
    c1 = model.AddConstr(-3 * x + 2 * y, -OPTV_INF, -3, "c1")
    c2 = model.AddConstr(x - 2 * y, -OPTV_INF, -2, "c2")

    obj = model.FeasRelax()

    print(f"L1 violation: {model.Get(OPTVDblAttr.OBJ_VAL)}")
    print(f"c0 upper bound = {c0.Get(OPTVDblAttr.UB)}")
    print(f"c1 upper bound = {c1.Get(OPTVDblAttr.UB)}")
    print(f"c2 upper bound = {c2.Get(OPTVDblAttr.UB)}")
```

若不可行性被成功修复，用户可以通过以下方法查询原模型的不可行违背量的 L^1-范数，及原模型所需修改量。

```
print(f"L1 violation: {model.Get(OPTVDblAttr.OBJ_VAL)}")
print(f"c0 upper bound = {c0.Get(OPTVDblAttr.UB)}")
print(f"c1 upper bound = {c1.Get(OPTVDblAttr.UB)}")
print(f"c2 upper bound = {c2.Get(OPTVDblAttr.UB)}")
```

4.6 LP 不可行诊断及修复 Java 接口

本部分展示在同一示例 (4.1) 上，如何在 Java 环境中使用 LP 不可行修复功能。

```
import com.huaweicloud.optv.*;

public class FeasibilityRelax {
    public static void main(String[] args) {
        try {
            OPTVEnv env = new OPTVEnv("FeasibilityRelax.log");
            OPTVModel model = new OPTVModel(env);
            model.set(OPTVStrParam.RESULT_FILE, "FeasibilityRelax.sol");

            // add variables to the model, all variables are continuous
            OPTVVar x = model.addVar(0, OPTV.OPTV_INF, 1, OPTV.OPTV_CONTINUOUS, "x");
            OPTVVar y = model.addVar(0, OPTV.OPTV_INF, 1, OPTV.OPTV_CONTINUOUS, "y");

            // add constraint to the model
            // 2 * x + y
            OPTVConstr c0 = model.addConstr(OPTV.add(OPTV.mul(2, x), y), -OPTV.OPTV_
↪INF, 2, "c0");
            // -3 * x + 2 * y
```

(续下页)

(接上页)

```
            OPTVConstr c1 = model.addConstr(OPTV.add(OPTV.mul(-3, x), OPTV.mul(2, y)),
↪ -OPTV.OPTV_INF, -3, "c1");
            // x - 2 * y
            OPTVConstr c2 = model.addConstr(OPTV.sub(x, OPTV.mul(2, y)), -OPTV.OPTV_
↪INF, -2, "c2");

            model.feasRelax();

            System.out.println("L1 violation: " + model.get(OPTVDblAttr.OBJ_VAL));
            System.out.println("c2 upper bound = " + c2.get(OPTVDblAttr.UB));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

这里不再赘述不可行修复功能调用之前的准备工作，更多细节请参考*Java 接口快速入门* 或*Java API examples*.

4.6.1 不可行修复结果查询

若不可行性被成功修复，用户可以通过以下方法查询原模型的不可行违背量的 L^1-范数，及原模型所需修改量。

```
System.out.println("L1 violation: " + model.get(OPTVDblAttr.OBJ_VAL));
System.out.println("c2 upper bound = " + c2.get(OPTVDblAttr.UB));
```

4.7 不可约不一致子系统 C++ 接口

本节将通过一个简单的 C++ 示例来演示 OptVerse 求解器计算不可约不一致子系统 (IIS) 的使用。该示例构建了一个模型，并对其进行求解，最后输出 IIS 的计算结果。

待求解问题的数学公式如下所示：

$$
\begin{array}{lrrrl}
\text{minimize} & & x_1 & + x_2 + & x_3 & \\
\text{subject to} & 1 \leq & x_1 & - 4x_2 + & x_3 & \\
& & 3x_1 & - x_2 - & x_3 & \leq 2 \\
& 3 \leq & x_1 & + 2x_2 - & 3x_3 & \\
& & x_1 & - x_2 - & & \leq 5 \\
& & 5x_1 & + 3x_2 + & x_3 & \leq 20 \\
& & & 0 \leq x_1,\ x_2,\ x_3 & & \\
& & & x_1,\ x_2,\ x_3 \in \mathbb{R}. & &
\end{array}
\tag{4.2}
$$

```
/*
 * Copyright (c) Huawei Technologies Co., Ltd. 2022-2023. All rights reserved.
 * Description: OptVerse c++ interface example - build and solve an LP problem
 */

#include "optv_c++.h"
using namespace std;

int main()
```

(续下页)

(接上页)

```
{
    try {
        OPTVEnv env("iis.log");
        OPTVModel model(env);

        // Add variables to the model, all variables are continuous
        OPTVVar x1 = model.AddVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "x1");
        OPTVVar x2 = model.AddVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "x2");
        OPTVVar x3 = model.AddVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "x3");

        // Add constraints to the model
        OPTVConstr c1 = model.AddConstr(x1 - 4*x2 + x3, 1, OPTV_INF, "c1");
        OPTVConstr c2 = model.AddConstr(3*x1 - x2 - x3, -OPTV_INF, 2, "c2");
        OPTVConstr c3 = model.AddConstr(x1 + 2*x2 - 3*x3, 3, OPTV_INF, "c3");
        OPTVConstr c4 = model.AddConstr(x1 - x2, -OPTV_INF, 5, "c4");
        OPTVConstr c5 = model.AddConstr(5*x1 + 3*x2 + x3, -OPTV_INF, 20, "c5");

        if (model.ComputeIIS()) {
            model.WriteIIS("iis.ilp");
        } else {
            cout << "IIS computation failed" << endl;
        }

    } catch (const OPTVException& e) {
        cout << e.GetErrorCode() << ": " << e.GetMessage() << endl;
    } catch (...) {
        cerr << "Unknown exception" << endl;
    }
    return 0;
}
```

4.7.1 包含头文件

使用 C++ 接口之前要包含头文件 `optv_c++.h`.

```
#include "optv_c++.h"
```

4.7.2 创建模型

首先初始化一个空的环境对象，提供给*OPTVModel* 构造函数。

```
OPTVEnv env("iis.log");
OPTVModel model(env);
```

添加变量和约束

```
// Add variables to the model, all variables are continuous
OPTVVar x1 = model.AddVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "x1");
OPTVVar x2 = model.AddVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "x2");
OPTVVar x3 = model.AddVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "x3");

// Add constraints to the model
OPTVConstr c1 = model.AddConstr(x1 - 4*x2 + x3, 1, OPTV_INF, "c1");
OPTVConstr c2 = model.AddConstr(3*x1 - x2 - x3, -OPTV_INF, 2, "c2");
OPTVConstr c3 = model.AddConstr(x1 + 2*x2 - 3*x3, 3, OPTV_INF, "c3");
OPTVConstr c4 = model.AddConstr(x1 - x2, -OPTV_INF, 5, "c4");
OPTVConstr c5 = model.AddConstr(5*x1 + 3*x2 + x3, -OPTV_INF, 20, "c5");
```

4.7.3 使用 IIS 计算功能

IIS 可以通过调用*OPTVModel::ComputeIIS()* 函数获取。该函数返回一个布尔变量，表征 IIS 是否被成功计算。

```
if (model.ComputeIIS()) {
    model.WriteIIS("iis.ilp");
} else {
    cout << "IIS computation failed" << endl;
}
```

在该示例中，IIS 是 c_1, c_2, c_3 约束以及其涉及的变量 x_1, x_2, x_3 边界约束。具体地，$c_1 + (-c2) + c3 \implies 2 \leq -x_1 - x_2 - x_3$，但是由于涉及到的变量 x_1, x_2, x_3 都大于等于 0，导致矛盾，从而它们不能同时成立。

4.8 不可约不一致子系统 Python 接口

对于前一节展示的不可行问题 (4.2)，我们接下来展示在 python 环境下的 IIS 的计算，具体细节请参考*Python 接口快速入门* 或*Python API 使用参考*，这里不再赘述中间步骤。

```
from optvpy import *

if __name__ == "__main__":
    env = OPTVEnv("iis.log")
    model = OPTVModel(env)

    # Add variables to the model, all variables are continuous
    x1 = model.AddVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "x1")
    x2 = model.AddVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "x2")
    x3 = model.AddVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "x3")

    # Add constraints to the model
    c1 = model.AddConstr(x1 - 4*x2 + x3, 1, OPTV_INF, "c1")
    c2 = model.AddConstr(3*x1 - x2 - x3, -OPTV_INF, 2, "c2")
    c3 = model.AddConstr(x1 + 2*x2 - 3*x3, 3, OPTV_INF, "c3")
    c4 = model.AddConstr(x1 - x2, -OPTV_INF, 5, "c4")
    c5 = model.AddConstr(5*x1 + 3*x2 + x3, -OPTV_INF, 20, "c5")

    if model.ComputeIIS():
        model.WriteIIS("iis.ilp")
```

(续下页)

(接上页)

```
    else:
        print("IIS computation failed")
```

类似地，IIS 可以通过调用*OPTVModel.ComputeIIS()* 函数获取。该函数返回一个布尔变量，表征 IIS 是否被成功计算。

```
if model.ComputeIIS():
    model.WriteIIS("iis.ilp")
else:
    print("IIS computation failed")
```

4.9 不可约不一致子系统 Java 接口

本部分展示在同一示例 (4.2) 上，如何在 Java 环境中使用不可约不一致子系统功能。

```
import com.huaweicloud.optv.*;

public class FeasibilityRelax {
    public static void main(String[] args) {
        try {
            OPTVEnv env = new OPTVEnv("iis.log");
            OPTVModel model = new OPTVModel(env);

            // Add variables to the model, all variables are continuous
            OPTVVar x1 = model.addVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "x1");
            OPTVVar x2 = model.addVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "x2");
            OPTVVar x3 = model.addVar(0, OPTV_INF, 1, OPTV_CONTINUOUS, "x3");

            // Add constraints to the model
            OPTVConstr c1 = model.addConstr(x1 - 4*x2 + x3, 1, OPTV_INF, "c1");
            OPTVConstr c2 = model.addConstr(3*x1 - x2 - x3, -OPTV_INF, 2, "c2");
            OPTVConstr c3 = model.addConstr(x1 + 2*x2 - 3*x3, 3, OPTV_INF, "c3");
            OPTVConstr c4 = model.addConstr(x1 - x2, -OPTV_INF, 5, "c4");
            OPTVConstr c5 = model.addConstr(5*x1 + 3*x2 + x3, -OPTV_INF, 20, "c5");

            if (model.computeIIS()) {
                model.writeIIS("iis.ilp");
            } else {
                cout << "IIS computation failed" << endl;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

这里不再赘述不可行修复功能调用之前的准备工作，更多细节请参考*Java 接口快速入门* 或*Java API examples*.

4.10 LP 敏感度分析 C++ 接口

本节将通过一个简单的 C++ 示例来演示 OptVerse 求解器 LP 敏感度分析的使用。该示例构建了一个模型，并对其进行求解，最后输出敏感度分析信息。

待求解问题的数学公式如下所示：

$$\begin{aligned} \text{minimize} \quad & -x_1 - 2x_2 - 3x_3 \\ \text{subject to} \quad & -x_1 + x_2 + x_3 \leq 20 \\ & x_1 - 3x_2 + x_3 \leq 30 \\ & x_1 \leq 40, x_2 \geq 0, x_3 \geq 0 \\ & x_1, x_2, x_3 \in \mathbb{R}. \end{aligned} \tag{4.3}$$

```
/*
 * Copyright (c) Huawei Technologies Co., Ltd. 2022-2023. All rights reserved.
 * Description: OptVerse c++ interface example -LP sensitivity analysis␣
↪functionalities
 */

#include <iomanip>

#include "optv_c++.h"
using namespace std;

int main()
{
    try {
        OPTVEnv env("lp-sensitivity-analysis.log");
        OPTVModel model(env);
        model.Set(OPTVStrParam::RESULT_FILE, "lp-sensitivity-analysis.sol");

        // Construct the model
        OPTVVar x1 = model.AddVar(-OPTV_INF, 40, -1, OPTV_CONTINUOUS, "x1");
        OPTVVar x2 = model.AddVar(0, OPTV_INF, -2, OPTV_CONTINUOUS, "x2");
        OPTVVar x3 = model.AddVar(0, OPTV_INF, -3, OPTV_CONTINUOUS, "x3");
        std::vector<OPTVVar> x {x1, x2, x3};

        model.AddConstr(-x1 + x2 +x3, -OPTV_INF, 20, "c1");
        model.AddConstr(x1 - 3*x2 +x3, -OPTV_INF, 30, "c2");

        // Optimize the model
        model.Optimize();
        if (model.Get(OPTVIntAttr::SOL_COUNT) > 0) {
            std::vector<int> rows{0, 1};
            std::vector<int> cols{0, 1, 2};
            cout << "-----------------Display---------------" << endl;

            cout << "Optimal Solution:" << endl;
            cout << std::setprecision(2) << std::scientific;
            for (auto j: cols) {
                cout << "x"<< (j+1) << " = " << x[j].Get(OPTVDblAttr::X) << endl;
            }

            // Sensitivity analysis on variables
```

(续下页)

(接上页)

```
            // Reduced cost: how much the objective will change if we increase the␣
↪simple bound constraints by 1
            cout << "Reduced Cost:" << endl;
            for (auto j : cols) {
                cout << "x"<< (j+1) << " : " << model.GetVar(j).
↪Get(OPTVDblAttr::DUAL) << endl;
            }


            // Objective coefficient sensitivity info, within which the current␣
↪optimal basis would remain optimal
            cout << "objective SA:" << endl;
            std::vector<double> objLower{};
            std::vector<double> objUpper{};
            bool ok = model.GetObjSA(objLower, objUpper, cols);
            for (auto j : cols) {
                cout << "x"<< (j+1) << " : [" << objLower[j] <<", " << objUpper[j] <<
↪"]" << endl;
            }

            // Lower bound sensitivity info, within which the current optimal basis␣
↪would remain optimal
            cout << "Lower bound SA:" << endl;
            std::vector<double> lbLower{};
            std::vector<double> lbUpper{};
            ok = model.GetVarLbSA(lbLower, lbUpper, cols);
            for (auto j : cols) {
                cout << "x"<< (j+1) << " : [" << lbLower[j] <<", " << lbUpper[j] << "]
↪" << endl;
            }

            // Upper bound sensitivity info, within which the current optimal basis␣
↪would remain optimal
            cout << "Upper bound SA:" << endl;
            std::vector<double> ubLower{};
            std::vector<double> ubUpper{};
            ok = model.GetVarUbSA(ubLower, ubUpper, cols);
            for (auto j : cols) {
                cout << "x"<< (j+1) << " : [" << ubLower[j] <<", " << ubUpper[j] << "]
↪" << endl;
            }

            // Sensitivity analysis on constraints

            // Shadow price: how much the objective will change if we increase the␣
↪right-hand side of a structural constraint by 1
            cout << "Shadow Price:" << endl;
            for (auto i : rows) {
                cout << "c"<< (i+1) << " : " << model.GetConstr(i).
↪Get(OPTVDblAttr::DUAL) << endl;
            }

            // LHS sensitivity info, within which the current optimal basis would␣
↪remain optimal
            cout << "Constraint left-hand-side SA:" << endl;
            std::vector<double> lhsLower{};
            std::vector<double> lhsUpper{};
```

(续下页)

(接上页)

```
            ok = model.GetConstrLbSA(lhsLower, lhsUpper, rows);
            for (auto i : rows) {
                cout << "c"<< (i+1) << " : [" << lhsLower[i] <<", " << lhsUpper[i] <<
↪"]" << endl;
            }

            // RHS sensitivity info, within which the current optimal basis would␣
↪remain optimal
            cout << "Constraint right-hand-side SA:" << endl;
            std::vector<double> rhsLower{};
            std::vector<double> rhsUpper{};
            ok = model.GetConstrUbSA(rhsLower, rhsUpper, rows);
            for (auto i : rows) {
                cout << "c"<< (i+1) << " : [" << rhsLower[i] <<", " << rhsUpper[i] <<
↪"]" << endl;
            }
        } else {
            cout << std::defaultfloat << "No feasible solution available!" << endl;
        }

    } catch (const OPTVException& e) {
        cout << e.GetErrorCode() << ": " << e.GetMessage() << endl;
    } catch (...) {
        cerr << "Unknown exception" << endl;
    }
    return 0;
}
```

4.10.1 包含头文件

要使用 C++ 接口，首先要包含头文件 `optv_c++.h`.

```
#include "optv_c++.h"
```

4.10.2 创建模型

首先初始化一个空的环境对象，提供给*OPTVModel* 构造函数。

```
OPTVEnv env("lp-sensitivity-analysis.log");
OPTVModel model(env);
```

添加变量和约束

```
// Construct the model
OPTVVar x1 = model.AddVar(-OPTV_INF, 40, -1, OPTV_CONTINUOUS, "x1");
OPTVVar x2 = model.AddVar(0, OPTV_INF, -2, OPTV_CONTINUOUS, "x2");
OPTVVar x3 = model.AddVar(0, OPTV_INF, -3, OPTV_CONTINUOUS, "x3");
std::vector<OPTVVar> x {x1, x2, x3};

model.AddConstr(-x1 + x2 +x3, -OPTV_INF, 20, "c1");
model.AddConstr(x1 - 3*x2 +x3, -OPTV_INF, 30, "c2");
```

4.10.3 求解模型

完成模型创建及配置所需求解参数后，可以通过调用*OPTVModel::Optimize()* 求解该模型。

```
model.Optimize();
```

4.10.4 输出模型解的信息

```
cout << "Optimal Solution:" << endl;
cout << std::setprecision(2) << std::scientific;
for (auto j: cols) {
    cout << "x"<< (j+1) << " = " << x[j].Get(OPTVDblAttr::X) << endl;
}
```

4.10.5 递减成本

递减成本反映了目标值对变量界限的敏感度，具体地，它们对应变量界限每增加一个单位后目标值的改变量。

```
cout << "Optimal Solution:" << endl;
cout << std::setprecision(2) << std::scientific;
for (auto j: cols) {
    cout << "x"<< (j+1) << " = " << x[j].Get(OPTVDblAttr::X) << endl;
}
```

4.10.6 目标函数敏感度分析

对于指定的变量（通过 cols 参数设定），目标函数敏感性分析功能提供一个区间 [objLower , objUpper]，只要对应的目标系数在此区间内，无论如何变化，所得模型的最优基不变。

```
// Objective coefficient sensitivity info, within which the current optimal basis␣
↪would remain optimal
cout << "objective SA:" << endl;
std::vector<double> objLower{};
std::vector<double> objUpper{};
bool ok = model.GetObjSA(objLower, objUpper, cols);
for (auto j : cols) {
    cout << "x"<< (j+1) << " : [" << objLower[j] <<", " << objUpper[j] << "]" << endl;
}
```

4.10.7 变量上/下界敏感度分析

对于指定的变量（通过 cols 参数设定），上/下界敏感性分析功能提供一个区间（如下界的敏感度区间 [lbLower , lbUpper], 上界的敏感度区间 [ubLower , ubUpper]），只要对应的上/下界在此区间内，无论如何变化，所得模型的最优基不变。

```
// Lower bound sensitivity info, within which the current optimal basis would remain␣
↪optimal
cout << "Lower bound SA:" << endl;
```

(续下页)

(接上页)

```
std::vector<double> lbLower{};
std::vector<double> lbUpper{};
ok = model.GetVarLbSA(lbLower, lbUpper, cols);
for (auto j : cols) {
    cout << "x"<< (j+1) << " : [" << lbLower[j] <<", " << lbUpper[j] << "]" << endl;
}
```

```
// Upper bound sensitivity info, within which the current optimal basis would remain␣
↪optimal
cout << "Upper bound SA:" << endl;
std::vector<double> ubLower{};
std::vector<double> ubUpper{};
ok = model.GetVarUbSA(ubLower, ubUpper, cols);
for (auto j : cols) {
    cout << "x"<< (j+1) << " : [" << ubLower[j] <<", " << ubUpper[j] << "]" << endl;
}
```

4.10.8 影子价格

影子价格反映了目标值对约束右侧值的敏感度，具体地，它们对应约束右侧值每增加一个单位后目标值的改变量。

```
// Shadow price: how much the objective will change if we increase the right-hand␣
↪side of a structural constraint by 1
cout << "Shadow Price:" << endl;
for (auto i : rows) {
    cout << "c"<< (i+1) << " : " << model.GetConstr(i).Get(OPTVDblAttr::DUAL) << endl;
}
```

4.10.9 约束左/右侧值敏感度分析

对于指定的约束（通过 `rows` 参数设定），左/右侧值敏感性分析功能提供一个区间，具体地，左侧值的敏感度区间 [`lhsLower`, `lhsUpper`]，右侧值的敏感度区间 [`rhsLower`, `rhsUpper`]。只要对应的左/右侧值在此区间内，无论如何变化，所得模型的最优基不变。

```
// LHS sensitivity info, within which the current optimal basis would remain optimal
cout << "Constraint left-hand-side SA:" << endl;
std::vector<double> lhsLower{};
std::vector<double> lhsUpper{};
ok = model.GetConstrLbSA(lhsLower, lhsUpper, rows);
for (auto i : rows) {
    cout << "c"<< (i+1) << " : [" << lhsLower[i] <<", " << lhsUpper[i] << "]" << endl;
}
```

```
// RHS sensitivity info, within which the current optimal basis would remain optimal
cout << "Constraint right-hand-side SA:" << endl;
std::vector<double> rhsLower{};
std::vector<double> rhsUpper{};
ok = model.GetConstrUbSA(rhsLower, rhsUpper, rows);
for (auto i : rows) {
    cout << "c"<< (i+1) << " : [" << rhsLower[i] <<", " << rhsUpper[i] << "]" << endl;
}
```

4.11 LP 敏感度分析 Python 接口

对于前一节展示的问题 (4.3)，我们接下来展示在 python 环境下的 LP 敏感度分析，具体细节请参考*Python 接口快速入门* 或*Python API 使用参考*，这里不再赘述中间步骤。

```
from __future__ import print_function
from optvpy import *

def print(*args):
    """ Print globally in scientific notation with a given precision """
    __builtins__.print(*("%.15e" % arg if isinstance(arg, float) else arg for arg in␣
↪args))

if __name__ == "__main__":
    env = OPTVEnv("lp-sensitivity-analysis.log")
    model = OPTVModel(env)
    model.Set(OPTVStrParam.RESULT_FILE, "lp-sensitivity-analysis.sol")

    # Construct the model
    x1 = model.AddVar(-OPTV_INF, 40, -1, OPTV_CONTINUOUS, "x1")
    x2 = model.AddVar(0, OPTV_INF, -2, OPTV_CONTINUOUS, "x2")
    x3 = model.AddVar(0, OPTV_INF, -3, OPTV_CONTINUOUS, "x3")
    x = [x1, x2, x3]
    model.AddConstr(-x1 + x2 + x3, -OPTV_INF, 20, "c1")
    model.AddConstr(x1 - 3*x2 + x3, -OPTV_INF, 30, "c2")

    # Optimize the model
    model.Optimize()
    if model.Get(OPTVIntAttr.SOL_COUNT) > 0:
        rows = [0, 1]
        cols = [0, 1, 2]

        print("-----------------Display---------------")

        print("Optimal Solution:")
        for j in range(3):
            print("x%s = %s" % (j+1, x[j].Get(OPTVDblAttr.X)))

        # Sensitivity analysis on variables

        # Reduced cost: how much the objective will change if we increase the simple␣
↪bound constraints by 1
        print("Reduced Cost:")
        for i in cols:
            print("x%s : %s" % (i+1, model.GetVar(i).Get(OPTVDblAttr.DUAL)))

        # Objective coefficient sensitivity info, within which the current optimal␣
↪basis would remain optimal
        print("objective SA:")
        objLower = []
        objUpper = []
        ok = model.GetObjSA(objLower, objUpper, cols)
        for i in cols:
            print("x%s : [%s, %s]" % (i+1, objLower[i], objUpper[i]))

        # Lower bound sensitivity info, within which the current optimal basis would␣
↪remain optimal
```

(续下页)

(接上页)

```
        print("Lower bound SA:")
        lbLower = []
        lbUpper = []
        ok = model.GetVarLbSA(lbLower, lbUpper, cols)
        for i in cols:
            print("x%s : [%s, %s]" % (i+1, lbLower[i], lbUpper[i]))

        # Upper bound sensitivity info, within which the current optimal basis would␣
↪remain optimal
        print("Upper bound SA:")
        ubLower = []
        ubUpper = []
        ok = model.GetVarUbSA(ubLower, ubUpper, cols)
        for i in cols:
            print("x%s : [%s, %s]" % (i+1, ubLower[i], ubUpper[i]))

        # Sensitivity analysis on constraints

        # Shadow price: how much the objective will change if we increase the right-
↪hand side of a structural constraint by 1
        print("Shadow Price:")
        for i in rows:
            print("c%s : %s" % (i+1, model.GetConstr(i).Get(OPTVDblAttr.DUAL)))

        # LHS sensitivity info, within which the current optimal basis would remain␣
↪optimal
        print("Constraint left-hand-side SA:")
        lhsLower = []
        lhsUpper = []
        ok = model.GetConstrLbSA(lhsLower, lhsUpper, rows)
        for i in rows:
            print("c%s : [%s, %s]" % (i+1, lhsLower[i], lhsUpper[i]))

        # RHS sensitivity info, within which the current optimal basis would remain␣
↪optimal
        print("Constraint right-hand-side SA:")
        rhsLower = []
        rhsUpper = []
        ok = model.GetConstrUbSA(rhsLower, rhsUpper, rows)
        for i in rows:
            print("c%s : [%s, %s]" % (i+1, rhsLower[i], rhsUpper[i]))
    else:
        print("No feasible solution available!")
```

4.12 LP 敏感度分析 Java 接口

本部分展示在同一示例 (4.3) 上，如何在 Java 环境中使用 LP 敏感度分析功能。

```
import com.huaweicloud.optv.*;
import java.util.*;

public class LpSensitivityAnalysis {
    public static void main(String[] args) {
        try {
```

(续下页)

(接上页)

```
            OPTVEnv env = new OPTVEnv("LpSensitivityAnalysis.log");
            OPTVModel model = new OPTVModel(env);
            model.set(OPTVStrParam.RESULT_FILE, "LpSensitivityAnalysis.sol");

            // construct the model
            OPTVVar x1 = model.addVar(-OPTV.OPTV_INF, 40, -1, OPTV.OPTV_CONTINUOUS,
↪"x1");
            OPTVVar x2 = model.addVar(0, OPTV.OPTV_INF, -2, OPTV.OPTV_CONTINUOUS, "x2
↪");
            OPTVVar x3 = model.addVar(0, OPTV.OPTV_INF, -3, OPTV.OPTV_CONTINUOUS, "x3
↪");
            OPTVVar[] x = {x1, x2, x3};
            // -x1 + x2 +x3
            model.addConstr(OPTV.add(OPTV.add(OPTV.sub(x1), x2), x3), -OPTV.OPTV_INF,␣
↪20, "c1");
            // x1 - 3*x2 +x3
            model.addConstr(OPTV.add(OPTV.sub(x1, OPTV.mul(3, x2)), x3), -OPTV.OPTV_
↪INF, 30, "c2");

            // optimize the model
            model.optimize();
            model.write("LpSensitivityAnalysis.lp");

            if (model.get(OPTVIntAttr.SOL_COUNT) > 0) {
                int[] rows = {0, 1};
                int[] cols = {0, 1, 2};

                System.out.println("-----------------Display---------------");

                System.out.println("Optimal Solution:");
                for (int j: cols) {
                    System.out.printf("x%d = %.2e\n", j+1, x[j].get(OPTVDblAttr.X));
                }

                // // sensitivity analysis on variables

                // reduced cost: how much the objective will change if we increase␣
↪the simple bound constraints by 1
                System.out.println("Reduced Cost:");
                for (int j : cols) {
                    System.out.printf("x%d : %.2e\n", j+1, model.getVar(j).
↪get(OPTVDblAttr.DUAL));
                }

                // // objective coefficient sensitivity info, within which the␣
↪current optimal basis would remain optimal
                System.out.println("objective SA:");
                OPTVModel.SensitvityBounds objBounds = model.getObjSA(cols);
                for (int j : cols) {
                    System.out.printf("x%d : [ %.2e,  %.2e]\n", j+1, objBounds.
↪down[j], objBounds.up[j]);
                }

                // lower bound sensitivity info, within which the current optimal␣
↪basis would remain optimal
                System.out.println("Lower bound SA:");
                OPTVModel.SensitvityBounds lbBounds = model.getVarLbSA(cols);
```

(续下页)

(接上页)

```
                for (int j : cols) {
                    System.out.printf("x%d : [ %.2e,  %.2e]\n", j+1, lbBounds.down[j],
↪ lbBounds.up[j]);
                }

                // upper bound sensitivity info, within which the current optimal␣
↪basis would remain optimal
                System.out.println("Upper bound SA:");
                OPTVModel.SensitvityBounds ubBound = model.getVarUbSA(cols);
                for (int j : cols) {
                    System.out.printf("x%d : [ %.2e,  %.2e]\n", j+1, ubBound.down[j],␣
↪ubBound.up[j]);
                }

                // // sensitivity analysis on constraints

                // shadow price: how much the objective will change if we increase␣
↪the right-hand side of a structural constraint by 1
                System.out.println("Shadow Price:");
                for (int i : rows) {
                    System.out.printf("x%d : %.2e\n", i+1, model.getConstr(i).
↪get(OPTVDblAttr.DUAL));
                }

                // LHS sensitivity info, within which the current optimal basis would␣
↪remain optimal
                System.out.println("Constraint left-hand-side SA:");
                OPTVModel.SensitvityBounds lhsBounds = model.getConstrLbSA(rows);
                for (int i : rows) {
                    System.out.printf("x%d : [ %.2e,  %.2e]\n", i+1, lhsBounds.
↪down[i], lhsBounds.up[i]);
                }

                // RHS sensitivity info, within which the current optimal basis would␣
↪remain optimal
                System.out.println("Constraint right-hand-side SA:");
                OPTVModel.SensitvityBounds rhsBounds = model.getConstrUbSA(rows);
                for (int i : rows) {
                    System.out.printf("x%d : [ %.2e,  %.2e]\n", i+1, rhsBounds.
↪down[i], rhsBounds.up[i]);
                }
            } else {
                System.out.println("No feasible solution available!");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

这里不再赘述不可行修复功能调用之前的准备工作，更多细节请参考*Java 接口快速入门* 或*Java API examples*.

CHAPTER 5

文件格式

本章描述了 OptVerse 求解器支持的各种文件格式，如

- `.mps`
- `.lp`

同时，OptVerse 也支持以下压缩格式：

- `.gz`
- `.z`
- `.Z`

5.1 MPS 格式文件说明

MPS 文件是运筹优化求解器常用的文件格式之一，它采用列格式书写且区分大小写; 每行数据表示一条记录，记录分为两种类型：指示符记录和数据记录，记录中包含空格，且常采用空格作为分割符。根据书写的方式不同常分为 **Fixed** 格式 (fixed format) 和 **Free** 格式 (free format), 其中 Fixed 格式字段必须书写在固定的范围列内，而 Free 格式限制更为宽松常采用空格隔离字段。固定格式的行列名常为 8 个字符，空格也算作字符，变量名中可以存在空格；自由格式则无字符数量限制，但变量名最好不要超过 255 个字符且不能包含空格；指示符记录用来标识数据段的, 每条指示符记录从第一列开始书写，常见指示符类型包括五个必选项和四个可选项。

节名称/指示符记录	是否必须	用途
NAME	是	用于指定问题名称；不同于其他指示符记录，名称记录包含数据
OBJSENSE	否	选用, 制定模型优化方向为最小化（默认）或最大化
ROWS	是	用于指定每个约束的名称和含义
COLUMNS	是	用于指定分配给每个变量（列）的名称和对应于该变量的非零约束系数
RHS	是	用于指定右侧向量的名称和每个约束（行）的值
RANGES	否	用于指定限制为处于两个值之间的区间内的约束；还会指定区间端点
BOUNDS	否	用于指定每个变量（列）必须保持的限制
QUADOBJ/QMATRIX	否	选用, 指定目标函数中的二次项
QCMATRIX	否	选用, 指定二次约束函数中的二次项
ENDATA	是	用于表示数据结束；始终为 MPS 文件中的最后一个条目

5.1.1 MPS 文件样例

下面混合整数问题的例子来解释每个指示符和数据记录的含义与使用方式

所描述模型的数学公式如下所示

$$
\begin{aligned}
\text{minimize} \quad & x_1 + 2x_5 - x_8 \\
\text{subject to} \quad & \\
2.5 \leq & 3x_1 + x_2 - 2x_4 - x5 - x_8 \\
& 2x_2 + 1.1x_3 && \leq 2.1 \\
& x_3 + x_6 && = 4.0 \\
1.8 \leq & 2.8x_4 - 1.2x_7 && \leq 5.0 \\
3.0 \leq & 5.6x_1 + x_5 + 1.9x_8 && \leq 15.0
\end{aligned}
$$

同时,

$$
\begin{aligned}
2.5 &\leq x_1 \\
0 &\leq x_2 \quad \leq 4.1 \\
0 &\leq x_3 \\
0 &\leq x_4 \\
0.5 &\leq x_5 \quad \leq 4.0 \\
0 &\leq x_6 \\
0 &\leq x_7 \\
0 &\leq x_8 \quad \leq 4.3
\end{aligned}
$$

这里,

$$
\begin{aligned}
& x_1, x_2 \in \mathbb{R} \\
& x_3, x_4 \in \{0, 1\}.
\end{aligned}
$$

对应 MPS 文件

```
NAME          EXAMPLE
ROWS
 N  OBJ
 G  ROW01
 L  ROW02
 E  ROW03
 G  ROW04
 L  ROW05
COLUMNS
    COL01     OBJ                1.0
    COL01     ROW01              3.0   ROW05              5.6
    COL02     ROW01              1.0   ROW02              2.0
*
*  Mark COL03 and COL04 as integer variables.
*
    INT1      'MARKER'                 'INTORG'
    COL03     ROW02              1.1   ROW03              1.0
    COL04     ROW01             -2.0   ROW04              2.8
    INT1END   'MARKER'                 'INTEND'
*
    COL05     OBJ                2.0
    COL05     ROW01             -1.0   ROW05              1.0
    COL06     ROW03              1.0
    COL07     ROW04             -1.2
    COL08     OBJ               -1.0
    COL08     ROW01             -1.0   ROW05              1.9
RHS
    RHS1      ROW01              2.5
    RHS1      ROW02              2.1
    RHS1      ROW03              4.0
    RHS1      ROW04              1.8
    RHS1      ROW05             15.0
RANGES
    RNG1      ROW04              3.2
    RNG1      ROW05             12.0
BOUNDS
 LO BND1      COL01              2.5
 UP BND1      COL02              4.1
 LO BND1      COL05              0.5
 UP BND1      COL05              4.0
 UP BND1      COL08              4.3
ENDATA
```

备注: 以星号 (*) 开头的行为注释行

5.1.2 数据记录格式

分为 Fixed 格式和 Free 格式, Fixed 格式对参数出现的位置要求极其严格。

Fixed 格式

对于 Fixed 格式如果定义字符偏移值 Inumchar 等于 8，则数据记录对应的位置为

记录	字段 1	字段 2	字段 3	字段 4	字段 5	字段 6
内容	2-3	5-12	15-22	25-36	40-47	50-61

如果字符偏移量不等于 8，则数据记录对应的位置也会改变：

参数记录	开始位置	结束位置
字段 1	2	3
字段 2	5	5 + Inumchar - 1
字段 3	字段 2 后的第三列	开始位置 +Inumchar-1
字段 4	字段 3 后的第三列	开始位置 +11
字段 5	字段 4 后的第四列	开始位置 +Inumchar-1
字段 6	字段 5 后的第五列	开始位置 +11

备注:

1. 字段 2、3、5 字符段，其中的字符串中不应该带有空格，如果有空格也会被移除；
2. 字段 4、6 是数据段，其数值必须能在 12 个字符内完成书写，这六个字段外的列必须是空格否则无法识别。

Free 格式

对于 Free 格式其基本形式也采用 6 字段的方式书写，其形式略有不同：

1. Free 格式则无需固定位置，可以在除第二列外的任何位置，不同字段通过空格分割；
2. 但是字段对应的顺序必须和 Fixed 格式相同，超过 6 个字段的记录仅读取前 6 个字段后续不再处理；
3. 如不是从第一列开始且采用 `$$` 的字段，会被当成注释，该行记录后续部分不会被处理；
4. 第 72-80 列自动忽略。

5.1.3 NAME 节

MPS 文件第一个指示符记录是问题名字段，表示模型的名字，其对应的对应字符段为

字段 1	字段 2	字段 3	字段 4
NAME	空格	模型名字	BINARY、FREE 或者空格

对于 Fixed 格式 NAME 必须书写在 1-4 列, modelName 必须书写在列范围 $[5+\text{Inumchar}+2, 5+2\cdot\text{Inumchar}+1]$ 内，字段 4 必须书写在列范围 $[5+2\cdot\text{Inumchar}+4, 5+2\cdot\text{Inumchar}+15]$ 内。关于字段 3 的 modelName，对于 Fixed 格式是可选项，而对于 Free 格式则是必选的。如果文件是二进制格式，Fixed 格式则要求指定字段 4 为 `BINARY`，否则留空即可。

5.1.4 OBJSENSE 节

该指示符非标准 MPS 格式具有，常作为扩展格式使用。是可选指示符记录，Fixed 格式不能包含该节。若未显式指定，则默认此模型采用最小化优化方向；OptVerse 代码中可以使用以下两个节来扩展 MPS 标准: OBJSENSE 和 OBJNAME，两者可在 NAME 节之后指定。OBJSENSE 用于设置目标函数含义，而 OBJNAME 用于从文件内的可用行中选择目标函数。如果这些节均未出现在 MPS 文件中，那么默认问题为最小化问题，并且目标函数由 ROWS 节中遇到的第一个可用行决定。如果使用这些选项，那么它们必须按顺序出现 NAME 节之后作为第一个和第二个节。OBJSENSE 的可选值是 `MAX` 或 `MIN`. 例如

```
NAME          example.mps
OBJSENSE
  MAX
OBJNAME
  rowname
```

5.1.5 ROWS 节

标题字段 ROWS 表征该参数段的开始位置，自此需要从第一列开始书写；参数记录按行书写，每行包含名称和约束含义，每条记录占一行，格式为

字段 1	字段 2
约束指示符	约束名字

字段 1 必须在 2-3 列，字段 1 与字段 2 之间至少需要一个空格。约束指示符表示约束的类型，包含了一个用于指定该行的含义的单个字母。可接受的值为:

1. E 表示该行等于
2. L 表示该行小于等于
3. G 表示该行大于等于
4. N 表示自由行

字段 2 包含字符标识（最大长度为 255 个字符），字段 3-6 不可在 ROWS 节中留白。MPS 文件将目标函数也当成一条约束，以标志符类型 `N` 包含在 ROWS 节中。如果定义了多个自由行，则仅将第一个作为目标函数，其他自由行会被丢弃。

5.1.6 Multiple Objectives 节

通过标志符 N 来定义多个目标函数。为确保这些目标函数被正确解析，需在同一行中使用空格分隔的方式提供以下参数：

1. 优先级
2. 权重 (可以取负数)
3. 绝对容差
4. 相对容差

在 OptVerse 求解器中进行多目标优化时，所有的目标函数都要是线性的。

下面的示例定义了两个目标函数，其中第一个目标函数的优先级为 2，两个目标函数的权重均为 1：

```
N  OBJ0 2 1 0 0
N  OBJ1 1 1 0 0
```

5.1.7 COLUMNS 节

COLUMNS 用来定义各列（变量）名字。在 COLUMNS 节中，约束矩阵的所有列均使用其名称和所有非零元素进行指定。数据记录格式为：

字段 1	字段 2	字段 3	字段 4	字段 5	字段 6
空白	列的名字	约束 1 的名字	约束 1 中该列的系数（非零）	约束 2 的名字	约束 2 中该列的系数（非零）

如果字段 2 为空白，则说明该记录和上一行的记录具有相同的列名。字段 3 是约束或者目标函数中的非零系数; 字段 5、6 和字段 3、4 的含义相同，是可选项。记录中的 ROWS 约束名称无顺序要求。若出现‘MARKER’‘INTORG’及‘MARKER’‘INTEND’，则分别表示整数变量的起止，介于这些关键字之间的列都是整数变量。

5.1.8 RHS 节

在 RHS 节中，指定了约束的非零右侧值。数据记录格式为：

字段 1	字段 2	字段 3	字段 4	字段 5	字段 6
空白	RHS 的名字	约束 1 的名字	约束 1 的 RHS 非零值	约束 2 的名字	约束 2 的 RHS 非零值

这里，

1. 第一列要留空；
2. 第二列表示 RHS 名字，可以任取，若为空则和上一条记录名字相同；
3. 每个 RHS 必须具有唯一的名称；
4. 第四列表示所对应的约束的 RHS 值；
5. 第五列、第六列分别与第三列、第四列的含义相同，为可选字段。

此外，还可以在 MPS 格式的文件的 RHS 节中声明模型的目标偏移量。使用此功能时，将偏移量的负值声明为目标函数的 RHS. 例如，以下行声明目标偏移量为 3.1415：

```
rhs       obj     -3.1415
```

5.1.9 RANGES 节

RANGES 节包含了约束右侧值（RHS）的范围。数据记录格式为：

字段 1	字段 2	字段 3	字段 4	字段 5	字段 6
空白	右侧范围向量标识	约束 1 的名字	约束 1 标识的范围值	约束 2 的名字（可选）	约束 2 标识的范围值（可选）

指定右侧范围的效果取决于指定行的含义以及该范围具有正系数还是负系数。对于给定的行，*rhs* 为右侧值，而 *range* 为对其应的范围值。范围值的解释如下：

行类型	范围值符号	生成的 RHS 上限	生成的 RHS 下限
G	+ 或 -	rhs + |range|	rhs
L	+ 或 -	rhs	rhs - |range|
E	+	rhs + range	rhs
E	-	rhs	rhs + range

5.1.10 BOUNDS 节

变量的界限值可在 BOUNDS 节中指定。数据记录格式为

字段 1	字段 2	字段 3	字段 4	字段 5	字段 6
边界类型	边界名称	列的名字	列的边界值	空白	空白

备注:

1. 字段 1: 界限类型，从左侧第二列开始书写
2. OPTV 支持界限类型值为：
   - LO: 下限
   - LI: 下限，整数
   - UP: 上限
   - UI: 上限，整数
   - FX: 固定值（上下限相同）
   - FR: 自由变量（下限为 -∞ 且上限为 +∞）
   - MI: 负无穷大（下限为 -∞）
   - PL: 正无穷大（上限为 +∞）
   - BV: 二进制变量，整数（0 或 1）

3. OptVerse 暂不支持界类型值为 - SC: 半连续变量的上限
4. 字段 5 和 6 不可用在 BOUNDS 节中
5. 如果界类型不是 LO, UP, FX, 或者 UI 则忽略字段 4 的取值
6. 界中出现的变量的顺序无需和 COLUMNS 中相同

5.1.11 QUADOBJ/QMATRIX 节

QUADOBJ 节定义了目标函数中的二次型，是可选的。如果未提供该节，则问题被当作（混合整数）线性规划问题处理。数据记录格式为

字段 1	字段 2	字段 3	字段 4	字段 5	字段 6
空白	变量 1 名字	变量 2 名字	系数值	空白	空白

该节中的每行表示了 Q 矩阵的上三角部分的一个非零值，每个值隐含了 1/2 系数，例如

```
QUADOBJ
    X1        X1        2.0
    X1        X2        4.0
    X2        X2        6.0
```

描述的 Q 如下：

$$Q = \frac{1}{2}\begin{bmatrix} 2 & 4 \\ 4 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$$

QMATRIX 节是 QUADOBJ 节的替代，不像 QUADOBJ 节只描述上三角部分，它描述了完整的矩阵，例如，上述矩阵可以在 QMATRIX 节中描述为：

```
QMATRIX
    X1        X1        2.0
    X1        X2        4.0
    X2        X1        4.0
    X2        X2        6.0
```

5.1.12 QCMATRIX 节

QCMATRIX 节扩展了 MPS 的功能，使其可以兼容二次约束问题。具体地，每个含有二次项的约束，需要单独的 QCMATRIX 节，约束名需要在 QCMATRIX 行提供，矩阵格式类似于 QMATRIX，需要完整的矩阵。不过，这里不再有除以 2 的默认操作。

```
QCMATRIX c1
    X1        X1        2.0
    X1        X2        4.0
    X2        X1        4.0
    X2        X2        6.0

QCMATRIX c3
    X1        X1        1.0
```

5.2 LP 格式文件说明

LP 格式作为一种在学界与业界都十分常用的文件格式，具有比 MPS 格式更易阅读的优点。

与 MPS 文件是列格式不同，LP 文件采用行表达式描述问题，且没有固定的书写范围和长度。但从处理性能和阅读难易的角度出发，各家求解器都规范了每行的最字节长度。OptVerse 求解器以 16 KB 的 buffer 进行读取。虽然超过此规格的行仍然可以正确读取，但是建议将每行的规格控制在 16 KB 以内。下面我们给出一个标准的 LP 文件：

```
\ENCODING-ISO-8859-1

\Problem name:Maximize
Maximize
obj:x1 + 2 x2 + 3 x3 + x4
Subject To
 r1: - x1 + x2 + x3 + 10 x4 <= 20
 r2: x1 - 3 x2 + x3 <= 30
 r3: x2 - 3.5 x4 = 0
Bounds
 0 <= x1 <=40
 2 <= x4 <= 3
Generals
 x4
Binaries
 x3
End
```

`Problem`, `Maximize`, `Subject To`, `Bounds`, `Generals`, `Binaries`, `General Constraints`, 和 `End` 分别是各节的指示词。它们出现的顺序需要以本章节接下来的介绍为准，否则会影响读取效果。各节指示词必须是单独一行书写，且对大小写不敏感。

下面我们对文件的各部分给出详细的描述。

5.2.1 LP 格式 BNF 规范

本章节介绍 OptVerse 求解器 Backus-Naur 形式规范。

⟨LP File⟩ ::=[⟨Model Name⟩]
⟨Objective Section⟩
⟨Constraint Section⟩
[⟨Bounds Section⟩]
[⟨Var Type Section⟩]
[⟨SOS Section⟩]
[⟨GenCon Section⟩]
“END”

⟨Model Name⟩ ::=(“PROBLEM” |“PROB”)⟨IDENT⟩

⟨Objective Section⟩ ::=⟨Objective Sense⟩([⟨Objective⟩]|⟨Multi Objective⟩+)

⟨Objective Sense⟩ ::=⟨Min Sense⟩|⟨Max Sense⟩

⟨Min Sense⟩ ::=(“MINIMIZE” |“MINIMUM” |“MIN”)

⟨Max Sense⟩ ::=(“MAXIMIZE” |“MAXIMUM” |“MAX”)

⟨Objective⟩ ::=[⟨IDENT⟩‘:’][⟨Linear Term⟩]+

⟨Multi Objective⟩ ::=[⟨IDENT⟩‘:’]⟨Multi Params⟩[⟨Linear Objective⟩]

⟨Constraint Section⟩ ::=(“SUBJECT TO” |“SUBJECT” |“ST”)[⟨Constraint⟩|⟨Indicator Constraint⟩]+

⟨Bounds Section⟩ ::=(“BOUNDS” |“BOUND”)[⟨Bound⟩]+

⟨Var Type Section⟩ ::=(⟨Binary Section⟩|⟨General Section⟩)+

⟨Binary Section⟩ ::=(“BINARIES” |“BINARY” |“BIN”)[⟨IDENT⟩]+

⟨General Section⟩ ::=(“GENERALS” |“GENERAL” |“GEN”)[⟨IDENT⟩]+

⟨Constraint⟩ ::=[⟨IDENT⟩‘:’][⟨Linear Term⟩] + ⟨SENSE⟩⟨Signed Number⟩

⟨Linear Term⟩ ::=[‘+’ |‘-’] + [⟨Signed Number⟩]⟨IDENT⟩

⟨SOS Section⟩ ::=(“SOS”)⟨SOS Constraint⟩+

⟨SOS Constraint⟩ ::=[⟨IDENT⟩‘:’]‘S’ [‘1’ |‘2’]“::” ⟨SOS Weight⟩ + \n

⟨SOS Weight⟩ ::=⟨IDENT⟩‘:’ ⟨Signed Number⟩

⟨GenCon Section⟩ ::=⟨GenCon Indicator⟩[⟨General Constraint⟩]+

⟨Bound⟩ ::=⟨Num or Inf⟩⟨SENSE⟩⟨IDENT⟩⟨SENSE⟩⟨Num or Inf⟩
|⟨Num or Inf⟩⟨SENSE⟩⟨IDENT⟩
|⟨IDENT⟩⟨SENSE⟩⟨Num or Inf⟩
|⟨IDENT⟩"FREE"

⟨SENSE⟩ ::='<' ['=']
|'>' ['=']
|'=' ['=']

⟨Num or Inf⟩ ::=⟨NUMBER⟩
|['+' |'-']("INFINITY" |"INF")

⟨Signed Number⟩ ::=['+' |'-'] + ⟨NUMBER⟩

表 1: BNF 符号定义

符号	描述
⟨A⟩	非终结符
⟨A⟩ := ...	替代 ⟨A⟩ 为 '...'
[⟨A⟩]	⟨A⟩, 可选
'A'	终结符
"A"	终结字符串
(⟨A⟩)	⟨A⟩
(⟨A⟩|⟨B⟩)	⟨A⟩ 或 ⟨B⟩
[⟨A⟩|⟨B⟩]	⟨A⟩ 或 ⟨B⟩, 可选
⟨A⟩+	一个或多个 ⟨A⟩
[⟨A⟩]+	一个或多个 ⟨A⟩, 可选 (基本上 *)

5.2.2 LP 格式的注释

反斜杠 (\) 后面的内容被认定为注释，求解器不对其进行解析，而是将其忽略；对于文件中的空白行同样进行忽略。例如

```
\ENCODING-ISO-8859-1

\Problem name:Maximize
```

5.2.3 LP 格式的关键字

LP 格式中的关键字可选用以下三种格式之一:

1. 全部大写 (如 `PROBLEM`)
2. 全部小写 (如 `problem`)
3. 首字母大写 (如 `Problem`)

关键词 (指示词)	可选项	含义
problem	prob	问题名称
minimize	min, minimum	最小化问题
maximize	max, maximum	最大化问题
subject to	st, s.t., such that	约束满足
bounds	bound	表达式的界
binaries	bin, binary	0-1 变量
generals	gen, general, int, integer, integers	整数变量
sos		SOS 约束
general constraints	gencon, gencons, general constraint, g.c.	一般约束
and		一般函数 AND
or		一般函数 OR
inf	infinity	无穷大
free		自由变量
end		结束

问题最小 (大) 化关键词可以可选地作为格式的第一行有效字段，如果省略则默认为最小化问题；end 关键词标注问题描述的结束，end 之后即便还存在字段也不会被解析。

5.2.4 LP 格式的节

在介绍节之前，我们首先介绍两个概念：**令牌**和 **表达式**。

令牌是单次能够处理的最小单位，不可通过插入任意字符分割，如空格或换行符。我们将数值、字符串、关键词、操作符、关系符视为令牌。

表达式是加入空格进行组合的令牌组，我们仅支持采用空格作为令牌组的令牌分隔符。如表达式 `10 x1 + x2 + 2000` 中，10 和 2000 是数值令牌，变量名 x1 和 x2 是字符串令牌，令牌之间采用空格分隔；`10,x1,+,x2,+,2000` 则不是令牌组或所谓的表达式，而仅仅是个字符串（字符串是由零个或多个字符组成的有限序列，可以包含特殊字符，但不可以以这些符号 *, +, < >, =, (,), ^, :, /, , , 或 . , 以及数字开头，也不可以作为关键词)。同时，它也不是变量名，因为变量名、约束名、目标函数名需要满足如下条件：彼此不同，名字长度不超过 255 个字符，不能以数字或者操作符 +, −, *, ^ 及关系符 <, >, = 开头，此外，也不可以是所谓的关键词。变量名最好也不要采用 E + 数字形式，如 E9 或 e9，因为这样会被解析为数字。建议使用字符和数字组合，如 var001.

Problem Name 节 [可选]

用户可以在此节中为模型指定名称，若缺省，求解器会以 optv 作为临时的模型名称。

Objective Function 节

目标函数可采用 maximize 或 minimize 关键词开头，形式为：

```
minimize
-x1 + x2 + -x3 + 0.5 x1 + 100 + x4 + 20
```

如上形式可以看出：

1. *minmize* 单独一行。
2. 变量可以直接与负号 (−) 连接，如 x1 和 x3.

3. 变量可以多次出现，我们聚合多次系数，本例中为 x1 的系数最终为 -1 + 0.5.
4. 偏移量可以出现在表达式中间或结尾，如 100 和 20. 如果出现多个偏移量，它们会被自动加和。

在实际使用中目标函数可以提供一个名字，并且可以多行书写，但需注意一些细节，如

```
minimize
obj: -x1 + x2 + -x3 + 0.5 x1
+ 100 + x4 + x5 + 1.5 + 2.0
x6
```

目标函数名后需要加冒号 (:) 作为分割；换行时以令牌为最小的组成单位。

Multiple Objectives

OptVerse 求解器也支持求解多目标优化问题。多目标优化问题的模型必须使用线性目标函数（目标函数中不能包含二次项），但可以设置一个空的目标函数。一个多目标优化问题模型中的每个目标都必需有一个表头，该表头需包含以下信息：

1. [可选] 目标名称，名称后使用冒号表示名称结束
2. 优先级-Priority
3. 权重-Weight （可以取负数）
4. 绝对容差-AbsTol
5. 相对容差-RelTol

下面这个示例展示了各个参数的格式以及如何指定多个目标：

```
obj1: Priority=2 Weight=1.0 AbsTol=1.0e-6 RelTol=0.0
 x0 + x1 + 2
obj2: Priority=1 Weight=1.0 AbsTol=1.0e-6 RelTol=0.0
 3 x0 + 2 x1 - 3
```

Quadratic Objectives 节

目标函数可以含有二次项，该部分需放入方括弧 [<quad terms>] 并由 / 2 随后。这里，空格符是可选的。

一般来说，二次项有以下两种写法：

1. x^2 或 x ^ 2
2. x*x 或 x * x

这两种方式都可以被正确解析和读取，* 和 ^ 操作符左右的空格符都是可选的。其次，混合项由 * 标识，对角元可以使用 * 或 ^ .

注意，若未读取到 / 2 ,相应的数据将被转换并以警告形式通知，否则用户需要在相应的 .lp 文件中将数值乘以 2 以避免该转换和警告；另一方面，除以 2 以外的数值会导致解析错误。

如果二次项未被提供系数，那么默认值 1.0 会被用来当做其系数。

另外，建议将表达式中二次项按照下标由低到高排序以提高效率，例如，

```
minimize
obj: -2.0 x1 +3.0 x2 -1.0 x3 + [ x1^2 + x2^2 + x1 * x2 + x3^2 + x1 * x3 + x2 * x3 ] /␣
↪2
```

重新排列上述示例中目标函数表达式的二次项部分为

```
minimize
obj: -2.0 x1 +3.0 x2 -1.0 x3 + [ x1^2 + x1 * x2 + x1 * x3 + x2^2 + x2 * x3 + x3^2 ] /␣
↪2
```

Constraints 节

约束满足节的关键词 `subject to` 同样需要单独一行书写。每个新的约束必须另起一行。约束可以提供约束名，在约束名之后跟冒号，约束名必须遵循与变量名称相同的规则。若约束名缺省，求解器会自动为约束增加临时名称，`R_#`，这里 `#` 是当前的约束总计数。

约束被定义为通过变量、常数、操作符组成的线性表达式，后跟关系符和数字右侧系数。使用以下关系符之一可以直观地指示约束的意义：`>=`, `<=`, 或 `=`. 例如，这是一个命名约束：

```
subject to:
depts01: -x1 + 0.5 x2 <= 40
```

但不允许关系符两边都存在表达式，如

```
subject to:
depts02: -x1 + 0.5 x2 <= 40 + x1
```

和双侧关系符，如

```
subject to:
depts03: 0 <= -x1 + 0.5 x2 <= 40 + x1 <= 40
```

Quadratic Constraints 节

与目标函数表达式相同，约束表达式也可以多行书写，但需要保持一致性。

约束表达式也可以含有二次项，该部分需放入方括弧 `[ <quad terms> ]`，但与目标函数中的二次项不同，这里 **没有** `/ 2` 随后。以下两种二次项的写法都被支持：

1. `x^2` 或 `x ^ 2`
2. `x*x` 或 `x * x`

这两种方式都可以被正确解析和读取，`*` 和 `^` 操作符左右的空格符都是可选的。。其次，混合项由 `*` 标识，对角元可以使用 `*` 或 `^`.

如果二次项未被提供系数，那么默认值 `1.0` 会被用来当做其系数。

同样，将表达式中二次项按照下标由低到高排序可以提高效率，例如，对于下面的二次约束，

```
subject to:
c1: -2.0 x1 +3.0 x2 -1.0 x3 + [ x1^2 + x2^2 + x1 * x2 + x3^2 + x1 * x3 + x2 * x3 ]
```

建议重新排列二次项部分，重写为

```
subject to:
c1: -2.0 x1 +3.0 x2 -1.0 x3 + [ x1^2 + x1 * x2 + x1 * x3 + x2^2 + x2 * x3 + x3^2 ]
```

Indicator Constraints 节

指示约束（Indicator Constraint）是 OptVerse 求解器中用于表达逻辑逻辑关系的条件约束，用于刻画如果某个 0-1 变量取特定值，则某条线性约束必须被满足的逻辑关系。指示约束允许在满足某些条件的情况下，禁用或者激活某一线性约束。指示约束的格式为：约束名称（可选），随后依次是 0-1 变量、等号、变量的值（0 或 1）、箭头”->”，最后是未标记的线性约束。

例如:

```
ind0: x = 0 -> y + z <= 20
```

Bounds 节 [可选]

`bounds` 节是可选的，如果存在则表明需要对变量使用提供的变量界。根据界被提供的个数，每一行表达式可以分为单边界或双边界。单边界形如 `x >= 10` 或 `x < 10`，双边界形如 `10 <= x <= 15`. 这两种界都不允许多行书写。如果没有提供变量界，则默认采用求解器的内部值。如果为变量定义了单个边界，则将使用适当的默认边界作为第二个边界。这里，有几条准则：

1. 固定边界表达式不被允许，即不能写为 `x1 = 20` 而要统一写为 `20 <= x1 <= 20`.
2. 支持 `+(-)inf`，如 `x1 > -inf`.
3. 支持 `free`, 如 `x1 free`.

Generals 和 Binaries 节 [可选]

LP 文件的 `generals` 和 `binaries` 节用于指示在可行解中必须具有整数值的变量。鉴于两者之间的相似性，我们将其放在同一个部分介绍，实际上他们是否同时存在互不影响。注册在这两节的变量将具有的默认边界的定义。对于在 `generals` 部分注册的变量，默认范围是 $[0, 10^{20}]$，而 `binaries` 部分注册的变量，默认边界是 0 和 1. 变量注册的方式是通过将变量罗列到对应节，如

```
Generals
x4 x5 x6 x7
x8 x9

Binaries
x1 x2 x3 x4
```

可见，不同变量之间通过空格分割，允许多行书写, 同时我们允许变量注册到不同节，如变量 `x4`，最终的变量类型为其可行区间最小的类型。

变量赋值可以采用任意顺序，也可以多次赋值（只有最后一次赋值生效）。

SOS Constraints 节 [可选]

`sos` 节支持用户为模型增加 SOS 约束。这里，每个 SOS 约束必须单独新起一行，且以新行结束。可以向 SOS 约束提供约束名（若约束名缺省，求解器会自动为约束增加临时名称，`SOS_#`, 这里 `#` 是当前的 SOS 约束总计数），在约束名之后跟冒号。然后，SOS 约束名后需要跟随 SOS 类型指示符（`S1` 或 `S2` ）和两个冒号。最后，SOS 约束由变量及权重组合的列表表征和定义，细节请参考如下 SOS 节：

```
SOS
   sos1: S1::   x : 1.0    y : 2.0
   sos2: S2::   x : 1.0    z : 2.0
```

General Constraints 节 [可选]

表 2: 支持的一般约束类型

约束	关键字	描述
And	AND	只有所有变量非零时，结果才非零
Or	OR	只要有一个变量非零，结果就非零
Max	MAX	返回提供的所有变量取值中的最大者
Min	MIN	返回提供的所有变量取值中的最小者

类似于 subject to 节，general constraints 节支持用户为模型增加一般约束。同样地，每个一般函数需要单独新起一行，可以提供约束名，在约束名之后跟冒号，约束名必须遵循与变量名称相同的规则。若约束名缺省，求解器会自动为约束增加临时名称，GC_# ，这里 # 是当前的一般约束总计数。

.lp 格式中的一般约束需要遵循以下规范：首先，若指定约束名称，则名称后需要加 : ；然后，需要一个变量名和等号联合表征该约束的结果；其次，在等号后，用户需指定约束函数类型，如 AND 或 OR ； 最后，约束函数中的参数列表需要以 , 分隔。细节请参考以下示例：

```
General Constraints
    and1: r1 = AND ( x1, x2, x3 )
    and2: r2 = AND ( x3, x1 )
    or1: r3 = OR ( x1, x2, x3 )
    or2: r4 = OR ( x3, x1 )
    max1: r5 = MAX ( x1, x2, 10.0 )
    min1: r6 = MIN ( x1, x2, 10.0 )
```

备注: 现阶段我们求解器仅支持线性问题，对于非线性字段和半连续、半整型暂不支持。

CHAPTER 6

C++ API 参考手册

6.1 OptVerse 常量

OPTV_INF

取值

`1.0e+100`

描述

代表无穷大的量的最大值。

OPTV_UNDF

取值

`1.0e+101`

描述

未定义的数值。

6.1.1 变量类型

OPTV_BINARY

取值

`B`

描述

0-1 整数变量类型。

OPTV_INTEGER

取值

I

描述

整数变量类型。

OPTV_CONTINUOUS

取值

C

描述

连续变量类型。

6.1.2 模型求解状态

OPTV_UNKNOWN

取值

0

描述

未知模型求解状态（默认）。

OPTV_OPTIMAL

取值

1

描述

模型找到最优解，可以查询到最优解。

OPTV_INFEASIBLE

取值

2

描述

模型被证明无解。

OPTV_INF_OR_UNBD

取值

3

描述

模型被证明无解或者无界。

OPTV_UNBOUNDED

取值

4

描述

模型被证明无界。

OPTV_TIME_LIMIT

取值

5

描述

模型在指定时间*OPTVDblParam::TIME_LIMIT* 内没有完成求解。

OPTV_MEM_LIMIT

取值

6

描述

模型求解使用内存超过预设值*OPTVIntParam::MEM_LIMIT*.

OPTV_INTERRUPTED

取值

7

描述

用户终止模型求解（如通过 Ctrl+C）.

6.1.3 变量和约束的基状态

OPTV_BASIC

取值

0

描述

变量或约束为基。

OPTV_NONBASIC_LOWER

取值

-1

描述

变量或约束非基，取下界值。

OPTV_NONBASIC_UPPER

取值

-2

描述

变量或约束非基，取上界值。

OPTV_SUPERBASIC

取值

-3

描述

变量或约束为超基。

6.1.4 一般约束类型

OPTV_GENCONSTR_AND

取值

0

描述

只有所有变量非零时，结果才非零。

OPTV_GENCONSTR_OR

取值

1

描述

只要有一个变量非零，结果就非零。

OPTV_GENCONSTR_INDICATOR

取值

2

描述

返回值为布尔型，当且仅当相应的逻辑条件满足，取值为 1，否则取值为 0。

6.1.5 约束含义

OPTV_SENSE_LESS

取值

<

描述

小于或等于。

OPTV_SENSE_EQUAL

取值

=

描述

等于。

OPTV_SENSE_GREATER

取值

>

描述

大于或等于。

6.2 OptVerse 参数

参数对应着用户使用的求解器内部配置，可以通过*OPTVEnv*（模型构建前）或*OPTVModel*（模型构建后）来设置。

6.2.1 整数型参数

enum class **OPTVIntParam**

表 1: OptVerse 整数型参数

名称	描述
BRANCH_DIR	MIP 分支方向
CROSSOVER	内点算法 crossover 策略
CUT_PASSES	根节点 (MIP) 割平面方法最大轮数
CUTS	MIP 割平面方法强度
MEM_LIMIT	用户指定的内存使用上限 (MB)
METHOD	LP 求解算法（如单纯形、内点等）
MIP_FOCUS	MIP 高层级的解策略
OPTIMAL_BASIS	清理所得解获取最优基
OUTPUT_FLAG	禁止日志打印
POOL_SOLUTIONS	解池存储的 MIP 解个数
PRESOLVE_LEVEL	预处理程度控制
SEED	PRNG 随机数种子
SOL_FORMAT	输出解的形式（稀疏或者稠密）
SOLUTION_NUMBER	用来指定 MIP 解
START_NODE_LIMIT	MIP 分支定界节点个数上限
SUBMIP_NODES	MIP 启发式策略模块访问的节点个数上限
THREADS	线程数
VAR_BRANCH	(MIP) 变量分支方法

BRANCH_DIR 分支方向

enumerator *OPTVIntParam*::**BRANCH_DIR**

MIP 分支方向。

默认值: 0

最小值: -1

最大值: 1

该参数控制分支剪界搜索中子节点的选取，取值-1 表示优先尝试下行节点，取值 +1 表示优先尝试上行节点。默认值为 0，表示自动选取子节点。

CROSSOVER

enumerator *OPTVIntParam*::**CROSSOVER**

内点法 crossover 策略。

默认值: 2

最小值: 0

最大值: 4

crossover 将内点解（通常为可行解，但一般不是基解）转化为基本可行解，它由以下三步组成:

1) 原始推进步 - 将原始变量推至边界，
2) 对偶推进步 - 将对偶变量推至边界，
3) 清理 - 应用单纯形方法清理任何原始或对偶不可行性。

该参数用于配置前两步的步骤及清理方法的选项。

取值	第一种推进方法	第二种推进方法	不可行性清理方法
0	禁用	禁用	禁用
1	对偶	原始	原始
2 (默认)	对偶	原始	对偶
3	原始	对偶	原始
4	原始	对偶	对偶

CUT_PASSES 割平面轮数

enumerator *OPTVIntParam*::**CUT_PASSES**

根节点 (MIP) 割平面方法最大轮数。

默认值: 2147483647

最小值: 0

最大值: 2147483647

该参数用以控制根节点割平面方法，使得轮数不超过设定值。

CUTS 割平面方法强度

enumerator *OPTVIntParam*::**CUTS**

MIP 割平面方法强度。

默认值: -1

最小值: -1

最大值: 3

该参数全局地控制割平面方法强度。默认值为-1，允许求解器进行自适应的调整；取值 0 则会关闭割平面方法；取值 1 会应用轻量级的割平面方法，取值 2 对应中等强度的割平面方法，而取值 3 则应用高强度的割平面方法。

MEM_LIMIT 内存限制

enumerator *OPTVIntParam*::**MEM_LIMIT**

求解器使用的内存限制, 单位 MB

默认值: 2147483647

最小值: 100

最大值: 2147483647

该参数用来限制求解器运行占用的 RAM 空间不超过指定阈值，若超出该阈值，求解器将终止运行并返回状态*OPTV_MEM_LIMIT*.

METHOD 优化算法

enumerator *OPTVIntParam*::**METHOD**

LP 问题优化算法选择。

默认值: -1

最小值: -1

最大值: 4

该参数用于指定 LP 问题的优化方法。

取值	方法
-1	自动
0	原始单纯形方法
1	对偶单纯形方法
2	内点方法 (IPM)
3	行生成方法
4	两阶段原始单纯形方法

MIP_FOCUS 解策略

enumerator *OPTVIntParam*::**MIP_FOCUS**

MIP 高层级的解策略。

默认值: 0

最小值: 0

最大值: 3

该参数控制 MIP 求解器的优先度策略，默认为均衡式，具体信息如下：

取值	方法
0	均衡式
1	优先快速找到可行解
2	优先证明最优性
3	优先改善下界

OPTIMAL_BASIS 最优基

enumerator *OPTVIntParam*::**OPTIMAL_BASIS**

求解结束时是否同时需要最优基。

默认值: 1 (开启)

最小值: 0 (关闭)

最大值: 1 (开启)

该高级参数允许用户获取可行解后跳过基状态清理过程，若不关心所得的可行解是基解与否，可关闭该选项。关闭该选项后，求解器不会执行基状态的清理操作。

警告: 若关闭该选项，所得解有基状态不正确的风险，而且这个参数会影响热启动的效果。仅适用于查询问题是否可行的情景，如果用户需要获取变量或约束的取值 (*var.Get(OPTVDblAttr::X)* 或 *constr.Get(OPTVDblAttr::X)*)，则需要 **开启**该选项 (默认环境下为开启状态)。

OUTPUT_FLAG 日志打印控制

enumerator *OPTVIntParam*::**OUTPUT_FLAG**

控制打印日志的详细程度。

默认值: 0

最小值: 0

最大值: 1

如果开启该选项，所得日志（屏幕和日志文件）将被缩减和简化。

POOL_SOLUTIONS 解池容量

enumerator *OPTVIntParam*::**POOL_SOLUTIONS**

MIP 解池的最大存储（解）数量。

默认值: 10

最小值: 1

最大值: 1000000

该参数用于设置 MIP 解池存储的解的数量上限。

PRESOLVE_LEVEL 预处理控制

enumerator *OPTVIntParam*::**PRESOLVE_LEVEL**

控制预处理的程度。

默认值: -1

最小值: -1

最大值: 3

用户通过该参数控制预处理的程度。

取值	程度
-1	自动
0	关闭预处理
1	保守倾向
2	中等
3	进取倾向（耗时多，处理后的模型更紧凑）

SEED 随机数种子

enumerator *OPTVIntParam*::**SEED**

PRNG 初始随机数种子

默认值: 0

最小值: 0

最大值: 2147483647

该参数用来设定随机数生成器的初始值。

SOL_FORMAT 解文件风格控制

enumerator *OPTVIntParam*::**SOL_FORMAT**

解文件稀疏/稠密控制。

默认值: 0

最小值: 0

最大值: 1

通过调整该参数，用户可以控制解文件中是否包含 0 值。若设置该参数为 0，解文件将省去 0 值，为稀疏格式；否则，解文件将包含 0 值，为稠密格式。

SOLUTION_NUMBER 解索引

enumerator *OPTVIntParam*::**SOLUTION_NUMBER**

用于查询指定（MIP）解。

默认值: 0

最小值: 0

最大值: 1000000

用户可以通过该参数指定要查询的解，从而可获悉更多相关信息，如*OPTVDblAttr::XN*或*OPTVDblAttr::POOL_OBJ_VAL*；取值为 0 时，上述查询信息等同于*OPTVDblAttr::X*或*OPTVDblAttr::OBJ_VAL*.

备注: 本参数仅适用于 MIP 问题。

START_NODE_LIMIT 分支定界节点数上限

enumerator *OPTVIntParam*::**START_NODE_LIMIT**

限制在 MIP 问题求解启动期间访问的分支定界节点数。

默认值: -1

最小值: -3

最大值: 2147483647

该参数用于控制 MIP 启动期间访问的分支定界节点数。默认选项允许求解器自动确定 MIP 启动期间计算节点的最大数目；取值为-2 时，禁用 MIP 部分启动评估而采用全部启动；取值为-3 将完全禁用 MIP 启动计算。

备注： 本参数仅适用于 MIP 问题。

SUBMIP_NODES 启发式节点数上限

enumerator *OPTVIntParam*::**SUBMIP_NODES**

MIP 启发式策略模块访问的节点个数上限。

默认值: 500

最小值: 0

最大值: 2147483647

该参数用以限制 MIP 启发式策略访问的节点个数，访问节点越多，越有可能产生更高质量的解，但通常也耗时更久。

备注： 本参数仅适用于 MIP 问题。

THREADS 线程数

enumerator *OPTVIntParam*::**THREADS**

求解器运行期间使用最大线程数。

默认值: 1

最小值: 1

最大值: 虚拟机可用的最大线程总数

VAR_BRANCH 变量分支方法

enumerator *OPTVIntParam*::**VAR_BRANCH**

(MIP) 变量分支方法

默认值: -1

最小值: -1

最大值:: 2

该参数控制变量分支策略，具体信息如下：

取值	方法
0	自动
0	Reliability 分支方法
1	Strong branching 方法
2	Pseudo cost branching 方法

6.2.2 双精度型参数

enum class **OPTVDblParam**

表 2: OptVerse 双精度型参数

名称	描述
FEAS_TOL	原始问题约束可行性精度
HEURISTICS	MIP 启发式策略模块占时比例
INT_TOL	整数性精度
MARKOWITZ_TOL	用于限制选择矩阵分解主元时的数值误差
GAP	MIP 相对偏差
TIME_LIMIT	最大运行时间（秒）

FEAS_TOL 约束可行性精度

enumerator *OPTVDblParam*::**FEAS_TOL**

原始问题约束可行性精度。

默认值: 10^{-6}

最小值: 10^{-9}

最大值: 0.01

若该参数取较小的值，所得解的约束违背量则越小，但是迭代步数可能更多。

HEURISTICS 启发式模块

enumerator *OPTVDblParam*::**HEURISTICS**

MIP 启发式策略模块占总求解时间的比例。

默认值: 0.05

最小值: 0

最大值: 1

该参数控制 MIP 求解器启发式策略模块的占用时间，以占时比例为测度，默认值为 0.05, 即占总时耗的 5%. 较高的该参数值可以使 MIP 求解器找到更多或更好的可行解，但是目标值的最佳边界进度也较慢。

INT_TOL 整数性精度

enumerator *OPTVDblParam*::**INT_TOL**

整数变量的整数性精度。

默认值: 10^{-6}

最小值: 10^{-9}

最大值: 0.01

若该参数取较小的值，所得解的整数性违背量则越小，但是求解性能（时耗）有可能被影响；取值较大一般不影响性能（时耗）。

MARKOWITZ_TOL 矩阵分解精度

enumerator *OPTVDblParam*::**MARKOWITZ_TOL**

适用于单纯形方法，用于限制选择矩阵分解主元时的数值误差。

默认值: 0.1

最小值: 10^{-4}

最大值: 0.999

取值较大将得到更好的数值稳定性，但可能影响求解性能（时耗）。

GAP 相对偏差

enumerator *OPTVDblParam*::**GAP**

MIP 相对偏差。

默认值: 0

最小值: 0

最大值: 10^{100}

若 f_P 和 f_D 分别为原始和对偶目标函数值，相对偏差值 g 的定义为 $g = \frac{|f_P - f_D|}{|f_P|}$. 在 MIP 求解过程中，一旦目标值的上下界相对偏差比 `gap` 小，求解器则会终止并返回状态*OPTV_OPTIMAL*.

TIME_LIMIT 求解时间上限

enumerator *OPTVDblParam*::**TIME_LIMIT**

模型求解时间上限（秒）。

默认值: 10^{20}

最小值: 0

最大值: 10^{20}

如果求解时间超出该数值，求解器则会终止并返回状态*OPTV_TIME_LIMIT*.

注意求解计时受制于单步迭代计时。对于耗时较久的优化问题，建议增加额外计时机制，如 `timeout`.

6.2.3 字符串型参数

enum class **OPTVStrParam**

表 3: OptVerse 字符串型参数

名称	描述
LOG_FILE	日志文件路径
PARAM_FILE	参数文件路径
REPLAY_FILE	SDK 命令回放文件路径
RESULT_FILE	结果文件路径

LOG_FILE 日志文件路径

enumerator *OPTVStrParam*::**LOG_FILE**

日志文件路径。

默认值: "./optv.log"

如果*OPTVIntParam::OUTPUT_FLAG* 被设置为 0，则本参数将被忽略。

PARAM_FILE 参数文件路径

enumerator *OPTVStrParam*::**PARAM_FILE**

参数文件路径。

默认值: ""

用于为求解器指定参数文件，默认值为空。

REPLAY_FILE SDK 命令回放文件

enumerator *OPTVStrParam*::**REPLAY_FILE**

SDK 命令回放文件路径。

默认值: ""

用于指定 SDK 命令回放文件存放的路径，若为空，则不创建命令回放文件。

备注: 该参数 **必须**在*OPTVModel* 对象创建前，通过*OPTVEnv* 对象设置。

RESULT_FILE 结果文件路径

enumerator *OPTVStrParam*::**RESULT_FILE**

结果文件路径。

默认值: "./<input_file>.sol"

备注: 通过 SDK 构建模型时，默认不进行输出结果文件，**必须**通过该参数指定生成解文件。

6.3 OptVerse 属性

属性对应着模型相关的各种数据，从属于模型组件对象 (*OPTVModel*, *OPTVVar*, *OPTVConstr*). 例如，任意给定变量或约束的下界可以通过*OPTVDblAttr::LB* 来获取。

备注: 有若干属性只能在*OPTVModel::Optimize()* 调用后才可以获取，如*OPTVDblAttr::X*, *OPTVDblAttr::XN*, *OPTVDblAttr::DUAL*, *OPTVDblAttr::OBJ_VAL*, *OPTVDblAttr::OBJ_BOUND*, 和*OPTVDblAttr::MIP_GAP*.

6.3.1 布尔属性

class **OPTVBoolAttr**

表 4: OptVerse 布尔型属性

名称	描述
IIS_CONSTR	指示该约束是否属于所得的 IIS

IIS_CONSTR 约束是否属于 IIS

enumerator *OPTVBoolAttr*::**IIS_CONSTR**

约束是否属于 IIS.

是否可修改: 否

表征指定约束是否属于计算所得的不可约不一致子系统 (IIS)；如果没有计算出 IIS，那么该属性不可获取。

6.3.2 整数型属性

class **OPTVIntAttr**

表 5: OptVerse 整数型属性

名称	描述
BASIS	变量或约束的基状态
BRANCH_PRIORITY	MIP 求解过程中变量的 branch(分支) 优先度
GENCONSTR_TYPE	一般约束类型
NUM_CONSTRS	约束总数
NUM_LIN_CONSTRS	线性约束总数
NUM_QUAD_CONSTRS	二次约束总数
NUM_INT_VARS	整数变量总数
NUM_VARS	变量总数
OBJ_SENSE	目标含义（最小化或最大化）
PROHIBITED	变量或约束是否被预处理屏蔽
SOL_COUNT	存储解的个数
SOS_TYPE	SOS 约束类型，1 或 2
STATUS	求解结果

BASIS 变量或约束的基状态

是否可修改: 是

变量或约束的基状态。

名称	取值	描述
OPTV_BASIC	0	基变量
OPTV_NONBASIC_LOWER	-1	非基，在下界
OPTV_NONBASIC_UPPER	-2	非基，在上界
OPTV_SUPERBASIC	-3	超基变量

BRANCH_PRIORITY 分支优先度

enumerator *OPTVIntAttr*::**BRANCH_PRIORITY**

是否可修改: 是

默认值: 0

该数值仅适用于 MIP 问题求解，用于引导 MIP 搜索过程的分支环节，以决定在哪些取值非整数的整数变量处进行分支，较大的取值表示较高的分支优先度。默认值为 0。如果存在优先度相同的情况，则采用原始的分支决策。

GENCONSTR_TYPE 一般约束类型

enumerator *OPTVIntAttr*::**GENCONSTR_TYPE**

一般约束类型。

是否可修改: 否

表 6: OptVerse 一般约束类型

名称	取值	描述
OPTV_GENCONSTR_AND	0	只有所有变量非零时，结果才非零
OPTV_GENCONSTR_OR	1	只要有一个变量非零，结果就非零
OPTV_GENCONSTR_INDICA	2	返回值为布尔型，当且仅当相应的逻辑条件满足，取值为 1，否则取值为 0

NUM_CONSTRS 约束总数

enumerator *OPTVIntAttr*::**NUM_CONSTRS**

是否可修改: 否

约束总数。

NUM_LIN_CONSTRS 线性约束总数

enumerator *OPTVIntAttr*::**NUM_LIN_CONSTRS**

是否可修改: 否

线性约束总数。

NUM_QUAD_CONSTRS 二次约束总数

enumerator *OPTVIntAttr*::**NUM_QUAD_CONSTRS**

是否可修改: 否

二次约束总数。

NUM_INT_VARS 整数变量总数

enumerator *OPTVIntAttr*::**NUM_INT_VARS**

是否可修改: 否

整数变量总数，包括 0/1 变量和普通整数变量。

NUM_VARS 变量总数

enumerator *OPTVIntAttr*::**NUM_VARS**

是否可修改: 否

变量总数。

OBJ_SENSE 目标含义

enumerator *OPTVIntAttr*::**OBJ_SENSE**

是否可修改: 是

优化问题目标含义，取值为-1 代表最大化问题，取值为 1 代表最小化问题。参考*OPTVSense*.

PROHIBITED 预处理屏蔽

enumerator *OPTVIntAttr*::**PROHIBITED**

是否可修改: 是

用于指示变量或约束是否被 LP 预处理屏蔽，取值包括 0（未屏蔽）和 1（屏蔽）。该参数不适用于 MIP 问题。

SOL_COUNT 存储解的个数

enumerator *OPTVIntAttr*::**SOL_COUNT**

是否可修改: 否

存储解的个数：对于 LP 问题，若获得最优解则取值为 1，否则取 0；对于 MIP 问题，该属性对应找到的解（包含最优和次最优）的个数。

SOS_TYPE SOS 类型

enumerator *OPTVIntAttr*::**SOS_TYPE**

是否可修改: 否

SOS 类型，可取值为 1 或 2.

STATUS 求解结果

enumerator *OPTVIntAttr*::**STATUS**

是否可修改: 否

模型求解状态。

名称	取值	描述
OPTV_OPTIMAL	1	模型求解成功（最优解）
OPTV_INFEASIBLE	2	模型被证明不可行
OPTV_INF_OR_UNBD	3	模型被证明不可行或无界
OPTV_UNBOUNDED	4	模型被证明无界
OPTV_TIME_LIMIT	5	求解时间超出给定时限*OPTVDblParam::TIME_LIMIT*.
OPTV_MEM_LIMIT	6	求解内存占用超过给定阈值*OPTVIntParam::MEM_LIMIT*.
OPTV_INTERRUPTED	7	用户自行终止（如通过 Ctrl+C）.
OPTV_UNKNOWN	0	未知状态。

6.3.3 字符型属性

class **OPTVCharAttr**

表 7: OptVerse 字符串型属性

名称	描述
VAR_TYPE	变量类型

VAR_TYPE 变量类型

enumerator *OPTVCharAttr*::**VAR_TYPE**

变量类型。

是否可修改: 是

表 8: OptVerse 变量类型

名称	取值	描述
OPTV_CONTINUOUS	'C'	连续型
OPTV_INTEGER	'I'	整数型
OPTV_BINARY	'B'	0/1 型

连续变量可取给定界限内的任意浮点数值；整数变量取值限定在给定界限内的所有整数；0/1 变量只可取 0 或者 1。

6.3.4 双精度型属性

class **OPTVDblAttr**

表 9: OptVerse 双精度型属性

名称	描述
DUAL	变量或约束的对偶解值。
LB	变量或约束的下界值。
MIP_GAP	MIP 相对偏差。
OBJ	变量的目标值系数。
OBJ_BOUND	目标值的最佳界限。
OBJ_OFFSET	目标函数的偏移量。
OBJ_VAL	原始模型目标值。
POOL_OBJ_VAL	第 N 个解的原始模型目标值。
RUN_TIME	运行时间。
START	MIP 热启动值。
UB	变量或约束的上界值。
X	变量或约束的原始解值。
XN	第 N 个解的变量或约束的原始解值。

DUAL 对偶解值

enumerator *OPTVDblAttr*::**DUAL**

是否可修改: 否

变量或约束的对偶解值。

LB 下界值

enumerator *OPTVDblAttr*::**LB**

是否可修改: 是

变量或约束的下界值。

备注: 对于 0/1 变量，该属性只允许取值为 `0` 或 `1`. 若赋值超出上述允许值 10^{-6}, 该操作会被禁止且相应的警告信息会被抛出。

MIP_GAP MIP 相对偏差

enumerator *OPTVDblAttr*::**MIP_GAP**

是否可修改: 否

MIP 相对偏差，仅适用于 MIP 求解范畴，具体定义可参考*相对偏差*。

OBJ 目标值系数

enumerator *OPTVDblAttr*::**OBJ**

是否可修改: 是

变量的目标值系数。

OBJ_BOUND 目标值的最佳界限

enumerator *OPTVDblAttr*::**OBJ_BOUND**

是否可修改: 否

目标值的最佳界限，仅适用于 MIP 求解范畴。

OBJ_OFFSET 目标函数偏移量

enumerator *OPTVDblAttr*::**OBJ_OFFSET**

是否可修改: 是

目标函数中的常数项。

OBJ_VAL 目标值

enumerator *OPTVDblAttr*::**OBJ_VAL**

是否可修改: 否

模型的原始解对应的目标值。

POOL_OBJ_VAL MIP 解的原始模型目标值

enumerator *OPTVDblAttr*::**POOL_OBJ_VAL**

是否可修改: 否

该属性仅适用于 MIP 求解范畴，当通过*OPTVIntParam::SOLUTION_NUMBER* 指定第 N 个 MIP 解时，*OPTVDblAttr::OBJ_VAL* 返回该解对应的原始模型目标值。

RUN_TIME 运行时间

enumerator *OPTVDblAttr*::**RUN_TIME**

是否可修改: 否

最近一次求解任务的执行时间。

START MIP 热启动值

enumerator *OPTVDblAttr*::**START**

是否可修改: 是

MIP 热启动值，该属性仅适用于 MIP 求解范畴，默认为*OPTV_UNDF*.

UB 上界值

enumerator *OPTVDblAttr*::**UB**

是否可修改: 是

变量或约束的上界值。

备注: 对于 0/1 变量，该属性只允许取值为 `0` 或 `1`．若赋值超出上述允许值 10^{-6}, 该操作会被禁止且相应的警告信息会被抛出。

X 原始解值

enumerator *OPTVDblAttr*::**X**

是否可修改: 否

变量或约束的原始解值。

XN MIP 解的原始解值

enumerator *OPTVDblAttr*::**XN**

是否可修改: 否

该属性仅适用于 MIP 求解范畴。类似于*OPTVDblAttr::X*, 当通过*OPTVIntParam::SOLUTION_NUMBER* 指定第 N 个 MIP 解时，*OPTVDblAttr::XN* 返回该解的原始解值。

6.3.5 长整型属性

class **OPTVLongAttr**

表 10: OptVerse 长整型属性

名称	描述
FINGERPRINT	模型指纹

FINGERPRINT （模型）指纹

enumerator *OPTVLongAttr*::**FINGERPRINT**

模型指纹

是否可修改: 否

该参数对应由模型内部数据和属性决定的 hash 值，不同的模型（包括变量或约束添加的顺序）应具有不同的模型指纹。

6.3.6 字符串型属性

class **OPTVStrAttr**

表 11: OptVerse 字符串型属性

名称	描述
NAME	变量或约束的名称。

NAME 变量/约束名称

enumerator *OPTVStrAttr*::**NAME**

变量或约束的名称，通常在建模环节根据问题定义设定。

是否可修改: 是

6.4 OptVerse 目标含义

enum class **OPTVSense**

目标含义，默认值为 1，表征最小化问题。

enumerator **MINIMIZE**

最小化问题 (+1)

enumerator **MAXIMIZE**

最大化问题 (-1)

6.5 OptVerse 环境

class **OPTVEnv**

OPTV 求解器环境类，用于管理 OPTV 求解器的参数配置。此外，许可文件需要通过环境变量 `OPTV_LICENSE_FILE` 来配置。

6.5.1 OPTVEnv() 环境类构造函数

OPTVEnv::**OPTVEnv**()

返回

环境变量，参数为默认。

备注: 默认配置下日志文件被设定为 `optv.log`.

OPTVEnv::**OPTVEnv**(const char *logFileName)

参数

logFileName –日志文件路径，格式为 C-风格字符串。

返回

环境变量，参数为默认，日志文件由参数指定。

OPTVEnv::**OPTVEnv**(const std::string &logFileName)

参数

logFileName –日志文件路径，格式为 C-风格字符串。

返回

环境变量，参数为默认，日志文件由参数指定。

OPTVEnv::**OPTVEnv**(const *OPTVEnv* &xenv)

参数
xenv –需要被复制的环境对象。

返回
环境对象 xenv 的副本。

OPTVEnv::**OPTVEnv**(const *OPTVEnv* *xenv)

参数
xenv –需要被复制的环境对象的指针。

返回
环境对象指针 xenv 所指对象的副本

6.5.2 Get() 参数查询

获取求解器的参数值，可用参数请参考*OptVerse 参数*。

OPTVEnv::**Get**(*OPTVIntParam* param)

参数
param –整数型参数。

返回
参数取值。

抛出
OPTVErrCode::*UNKNOWN_PARAMETER* –未知参数。

OPTVEnv::**Get**(*OPTVDblParam* param)

参数
param –双精度型参数。

返回
参数取值。

抛出
OPTVErrCode::*UNKNOWN_PARAMETER* –未知参数。

OPTVEnv::**Get**(*OPTVStrParam* param)

参数
param –字符串型参数。

返回
参数取值。

抛出
OPTVErrCode::*UNKNOWN_PARAMETER* –未知参数

6.5.3 Set() 参数设置

通过环境对象设置求解器参数，相应修改会作为默认，被应用到后续创建的所有模型。模型建立后，只有调用*OPTVModel::Update()* 才可以激活新的参数修改。具体细节可参考*OptVerse 参数*。

OPTVEnv::**Set**(*OPTVIntParam* param, int value)

参数

- **param** –整数型参数。
- **value** –新参数值。

返回
无。

抛出
OPTVErrCode::*UNKNOWN_PARAMETER* –未知参数。

OPTVEnv::**Set**(*OPTVDblParam* param, double value)

参数

- **param** –双精度型参数。
- **value** –新参数值。

返回
无。

抛出
OPTVErrCode::*UNKNOWN_PARAMETER* –未知参数。

OPTVEnv::**Set**(*OPTVStrParam* param, const std::string &value)

参数

- **param** –字符串型参数。
- **value** –新参数值。

返回
无。

抛出
OPTVErrCode::*UNKNOWN_PARAMETER* –未知参数。

6.5.4 ResetParams() 重置参数

将所有的参数重置为默认值。

OPTVEnv::**ResetParams**()

返回
无。

6.6 OptVerse 变量

class **OPTVVar**

OptVerse 变量对象，用于从模型访问变量。注意变量不可由其构造函数创建，而需要通过模型的*OPTVModel::AddVar()* 或*OPTVModel::AddVars()* 接口添加变量到模型。

6.6.1 Get() 查询变量属性

查询变量属性，关于可用属性请参考*OptVerse 属性*。

OPTVVar::**Get**(*OPTVIntAttr* attr)

参数
attr –整数型属性。

返回
属性值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知变量属性。

OPTVVar::**Get**(*OPTVCharAttr* attr)

参数
attr –字符型属性。

返回
属性值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知变量属性。

OPTVVar::**Get**(*OPTVDblAttr* attr)

参数
attr –双精度型属性。

返回
属性值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知变量属性。

OPTVVar::**Get**(*OPTVStrAttr* attr)

参数
attr –字符串型属性。

返回
属性值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知变量属性。

6.6.2 Set() 设置变量属性

设置变量属性，只有调用*OPTVModel::Update()* 后，新设置才可生效，更多细节请参考*OptVerse 属性*。

OPTVVar::**Set**(*OPTVIntAttr* attr, int value)

参数

- **attr** –整数型属性。
- **value** –新属性值。

返回

无。

抛出

OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知变量属性。

OPTVVar::**Set**(*OPTVCharAttr* attr, int value)

参数

- **attr** –字符型属性。
- **value** –新属性值。

返回

无。

抛出

OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知变量属性。

OPTVVar::**Set**(*OPTVDblAttr* attr, int value)

参数

- **attr** –双精度型属性。
- **value** –新属性值。

返回

无。

抛出

OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知变量属性。

OPTVVar::**Set**(*OPTVStrAttr* attr, int value)

参数

- **attr** –字符串型属性。
- **value** –新属性值。

返回

无。

抛出

OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知变量属性。

6.6.3 Index() 变量索引

获得变量在模型中的索引值。

OPTVVar::**Index**() const

返回
变量索引/下标。

返回值

- **-1** –该变量在模型中不存在。
- **-2** –该变量已从模型中删除。
- **>=0** –该变量在模型中的索引/下标。

6.6.4 SameAs() 变量对比

查询两变量是否相同。

bool **SameAs**(const *OPTVVar* &v2) const

参数
v2 –需要与当前变量对比的变量。

返回
变量是否相同。

返回值

- **true** –变量相同。
- **false** –变量不同。

6.7 OptVerse 列

class **OPTVColumn**

OptVerse 列对象，用于以列的形式向模型添加变量，用户可以使用该功能向模型同时新增变量并添加其至已有约束。

在下面的例子中，我们构造一个初始模型并展示如何通过列对象增加变量，本示例展示的模型为*Quick Start* 章节的*milp1* .

```
#include "optv_c++.h"

OPTVEnv env;
OPTVModel model(env);

OPTVVar x = model.AddVar(0, 1, -1, OPTV_BINARY, "x");
OPTVVar y = model.AddVar(0, OPTV_INF, -14, OPTV_INTEGER, "y");

OPTVConstr c0 = model.AddConstr(2 * x + y, -OPTV_INF, 3, "c0");
OPTVConstr c1 = model.AddConstr(3 * y, -OPTV_INF, 6, "c1");
```

具体地，构建的模型形式如下：

$$A = \begin{bmatrix} 2 & 1 \\ 0 & 3 \end{bmatrix}$$

```
OPTVColumn col;

col.AddTerm(1, c0);
col.AddTerm(1, c1);

OPTVVar z = model.AddVar(0, 3, -6, OPTV_CONTINUOUS, col, "z");
```

通过将 `col` 添加到模型，相应的约束矩阵变为：

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 0 & 3 & 1 \end{bmatrix}$$

6.7.1 AddTerm() 添加单项

向列对象（尾部）添加单项。

OPTVColumn::**AddTerm**(double coef, const *OPTVConstr* &constr)

该函数向列对象添加单项。

参数

- **coef** –系数值。
- **constr** –约束对象。

返回

无。

6.7.2 AddTerms() 增加多项

向列对象（尾部）添加多项。

OPTVColumn::**AddTerms**(const double *coefs, const *OPTVConstr* *constrs, int cnt)

该功能向列对象添加多项。

参数

- **coefs** –C-风格的系数序列。
- **constrs** –C-风格的约束序列。
- **cnt** –添加项的个数。

返回

无。

抛出

OPTVErrCode::*INVALID_ARGUMENT* –`constrs` 或 `coefs` 为空指针或 `cnt` 取负值。

6.7.3 Clear() 清空

清空与该列相关内部容器，使*OPTVColumn::Size()* 返回 0.

OPTVColumn::**Clear**()

返回
无。

Rtype
None

6.7.4 GetCoeff() 获取系数

据指定索引值获得列对象中对应的约束系数。

OPTVColumn::**GetCoef**(int i) const

参数
i –指定的索引值，对应列中的约束。

返回
列对象中对应该索引, i, 的约束系数。

Raises OPTVErrCode::INDEX_OUT_OF_RANGE
若 $i \notin [0, \texttt{this->Size()}]$.

6.7.5 GetConstr() 获取单个约束

根据指定索引值获得列对象中对应的约束。

OPTVColumn::**GetConstr**(int i) const

参数
i –指定的索引值，对应列中的约束。

返回
列对象中对应该索引, i, 的约束对象。

抛出
OPTVErrCode::*INDEX_OUT_OF_RANGE* –若 $i \notin [0, \texttt{this->Size()}]$.

6.7.6 Remove() 删除项

从列对象中删除单/多项。

OPTVColumn::**Remove**(int i)
根据指定索引值从该列对象中删除项。

参数
i –指定的索引值。

返回
无。

抛出
OPTVErrCode::*INDEX_OUT_OF_RANGE* –若 $i \notin [0, \texttt{this->Size()}]$.

OPTVColumn::**Remove**(const *OPTVConstr* &constr)

从列对象中删除指定约束的所有相关项。

参数
constr –指定的约束。

返回
是否删除任何项。

返回
无。

抛出
OPTVErrCode::*NOT_IN_MODEL* –指定的约束，constr, 在模型中不存在。

6.7.7 Size() 规模

获得该列的规模。

OPTVColumn::**Size**() const

该数值统计对应列中的元素个数，无论是否有重复项，例如，若某约束被多次添加，本数值依然相应增加。

返回
该列的规模。

6.8 OptVerse 约束

class **OPTVConstr**

OptVerse 约束对象，用于从模型访问约束。注意约束不可由其构造函数创建，而需要通过模型的*OPTVModel::AddConstr()* 或*OPTVModel::AddConstrs()* 接口添加约束至模型。此外，二次约束的相应接口为*OPTVModel::AddQConstr()*.

6.8.1 Get() 查询约束属性

查询约束属性，关于可用属性请参考*OptVerse 属性*。

OPTVConstr::**Get**(*OPTVIntAttr* attr)

参数
attr –整数型属性。

返回
属性值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

OPTVConstr::**Get**(*OPTVDblAttr* attr)

参数
attr –双精度型属性。

返回
属性值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

OPTVConstr::**Get**(*OPTVStrAttr* attr)

参数
attr –字符串型属性。

返回
属性值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

6.8.2 Set() 设置约束属性

设置约束属性，只有调用*OPTVModel::Update()* 后，新设置才可生效，更多细节请参考*OptVerse 属性*。

OPTVConstr::**Set**(*OPTVIntAttr* attr, int value)

参数

- **attr** –整数型属性。
- **value** –新属性值。

返回
无。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

OPTVConstr::**Set**(*OPTVDblAttr* attr, int value)

参数

- **attr** –双精度型属性。
- **value** –新属性值。

返回
无。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

OPTVConstr::**Set**(*OPTVStrAttr* attr, int value)

参数

- **attr** –字符串型属性。
- **value** –新属性值。

返回
无。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

6.8.3 Index() 约束索引

获得约束在模型中的索引值。

OPTVConstr::**Index**() const

返回
约束索引/下标。

返回值

- **-1** –该约束在模型中不存在。
- **-2** –该约束已从模型中删除。
- **>=0** –该约束在模型中的索引/下标。

6.8.4 SameAs() 约束对比

查询两约束是否相同。

OPTVConstr::**SameAs**(const *OPTVConstr* &c2) const

参数
c2 –需要与当前约束对比的约束。

返回
约束是否相同。

返回值

- **true** –约束相同。
- **false** –约束不同。

6.9 OptVerse 二次约束

class **OPTVQConstr**

二次约束类用于访问模型中的二次约束。注意二次约束也不可以由其构造函数直接创建，而是通过接口*OPTVModel::AddQConstr()* 将其添加到模型。

警告: 当求解的模型 **包含**二次约束时，QCQP 求解器要求模型的二次目标函数或二次约束的矩阵都是正半定的，否则会抛出*OPTVErrCode::MATRIX_NOT_PSD* 异常。

6.9.1 Get() 查询二次约束属性

查询二次约束属性，更多细节请参考*OptVerse 属性*。

OPTVQConstr::**Get**(*OPTVIntAttr* attr)

参数
attr –整数型属性。

返回
属性值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

OPTVQConstr::**Get**(*OPTVDblAttr* attr)

参数
attr –双精度型属性。

返回
属性值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

OPTVQConstr::**Get**(*OPTVStrAttr* attr)

参数
attr –字符串型属性。

返回
属性值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

6.9.2 Set() 设置二次约束属性

设置二次约束属性，只有调用*OPTVModel::Update()* 后，新设置才可生效，更多细节请参考*OptVerse 属性*。

OPTVQConstr::**Set**(*OPTVIntAttr* attr, int value)

参数

- **attr** –整数型属性。
- **value** –新属性值。

返回
无。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

OPTVQConstr::**Set**(*OPTVDblAttr* attr, int value)

参数

- **attr** –双精度型属性。
- **value** –新属性值。

返回
无。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

OPTVQConstr::**Set**(*OPTVStrAttr* attr, int value)

参数

- **attr** –字符串型属性。
- **value** –新属性值。

返回
无。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

6.9.3 Index() 二次约束索引

获取二次约束在模型中的索引值。

OPTVQConstr::**Index**() const

返回
二次约束索引/下标。

返回值

- **-1** –该二次约束在模型中不存在。
- **-2** –该二次约束已从模型中删除。
- **>=0** –该二次约束在模型中的索引/下标。

6.9.4 SameAs() 二次约束对比

查询两二次约束是否相同。

OPTVQConstr::**SameAs**(const *OPTVQConstr* &c2) const

参数
c2 –需要与当前二次约束对比的二次约束。

返回
二次约束是否相同。

返回值

- **true** –二次约束相同。
- **false** –二次约束不同。

6.10 OPTVSOS SOS 约束

class **OPTVSOS**

特殊顺序集，special ordered set (SOS)，是指一组有序集合里，至多有一个非零值（SOS1 型）或至多有两个非零值（SOS2 型）。SOS 约束是对一组变量的取值进行限制的特殊约束，可以额外地指定模型中的整数条件。这类约束单独列出来，可以加快线性规划的求解速度。

OptVerse SOS 约束对象，用以从模型中获取 SOS 约束数据。这里，用户需要通过*OPTVModel::AddSOS()*接口向模型中添加 SOS 约束，而非借助于构造函数。具体地，OptVerse 支持下列两种 SOS 约束：

- SOS 类型 1 要求指定的一组变量至多有一个变量可取非零值
- SOS 类型 2 指定的一组变量至多有两个变量可取非零值，且取非零值的变量顺序要求相邻

6.10.1 Get() 查询 SOS 约束属性

查询 SOS 约束的属性，更多细节请参考*OptVerse 属性*。

OPTVSOS::**Get**(*OPTVIntAttr* attr)

参数
attr –整数型属性。

返回
属性值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

OPTVSOS::**Get**(*OPTVStrAttr* attr)

参数
attr –字符串型属性。

返回
属性值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

6.10.2 Set() 设置 SOS 约束属性

设置 SOS 约束属性，只有调用*OPTVModel::Update()* 后，新设置才可生效，更多细节请参*OptVerse 属性*。

OPTVSOS::**Set**(*OPTVIntAttr* attr, int value)

参数
- **attr** –整数型属性。
- **value** –新属性值。

返回
无。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

OPTVSOS::**Set**(*OPTVStrAttr* attr, const std::string &value)

参数
- **attr** –字符串型属性。
- **value** –新属性值。

返回
无。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

6.10.3 Index() SOS 约束索引

获取 SOS 约束在模型中的索引值。

OPTVSOS::**Index**() const

返回
SOS 约束索引/下标。

返回值
- **-1** –该 SOS 约束在模型中不存在。
- **-2** –该 SOS 约束已从模型中删除。
- **>=0** –该 SOS 约束在模型中的索引/下标。

6.10.4 SameAs() SOS 约束对比

查询两 SOS 约束是否相同。

OPTVSOS::**SameAs**(const *OPTVSOS* &c2) const

参数
c2 –需要与当前 SOS 约束对比的 SOS 约束。

返回
SOS 约束是否相同。

返回值

- **true** –SOS 约束相同。
- **false** –SOS 约束不同。

6.11 OptVerse 一般约束

class **OPTVGenConstr**

OptVerse 一般约束，用于从模型中获取一般约束数据。特别地。一般约束需要通过 `OPTVModel::AddGenConstrXxx` 向模型中添加约束而构造，而不是借助于任何构造函数。

表 12: OptVerse 支持的一般约束类型

约束	类型	接口	描述
And	*OPTV_GENCONSTR_*	*OPTVModel::AddGenCons*	只有所有变量非零时，结果才非零
Or	*OPTV_GENCONSTR_*	*OPTVModel::AddGenCons*	只要有一个变量非零，结果就非零
Indicator	*OPTV_GENCONSTR_*	*OPTVModel::AddGenCons*	返回值为布尔型，当且仅当相应的逻辑条件满足，取值为 1，否则取值为 0

6.11.1 Get() 查询一般约束属性

查询一般约束属性，关于可用属性请参考*OptVerse 属性*。

OPTVGenConstr::**Get**(*OPTVIntAttr* attr)

参数
attr –整数型属性。

返回
属性值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

OPTVGenConstr::**Get**(*OPTVDblAttr* attr)

参数
attr –双精度型属性。

返回
属性值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

OPTVGenConstr::**Get**(*OPTVStrAttr* attr)

参数
attr –字符串型属性。

返回
属性值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

6.11.2 Set() 设置一般约束属性

设置一般约束属性，只有调用*OPTVModel::Update()* 后，新设置才可生效，更多细节请参考*OptVerse 属性*。

OPTVGenConstr::**Set**(*OPTVDblAttr* attr, int value)

参数

- **attr** –双精度型属性。
- **value** –新属性值。

返回
无。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

OPTVGenConstr::**Set**(*OPTVStrAttr* attr, int value)

参数

- **attr** –字符串型属性。
- **value** –新属性值。

返回
无。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知约束属性。

6.11.3 Index() 一般约束索引

获得一般约束在模型中的索引值。

OPTVGenConstr::**Index**() const

返回
约束索引/下标。

返回值

- **-1** –该约束在模型中不存在。
- **-2** –该约束已从模型中删除。
- **>=0** –该约束在模型中的索引/下标。

6.11.4 SameAs() 一般约束对比

查询两一般约束是否为同一对象。

OPTVGenConstr::**SameAs**(const *OPTVGenConstr* &c2) const

参数
c2 –需要与当前约束对比的约束。

返回
约束是否相同。

返回值

- **true** –约束相同。
- **false** –约束不同。

6.12 OptVerse 线性表达式

class **OPTVLinExpr**

线性表达式*OPTVLinExpr* 类用于构建线性的目标函数或者约束函数。线性表达式对象由变量和常量组成，边获取边构造，如下例所示的表达式 `1 * x + 2 * y + 3 * x`，它包含以下元素：

下标	系数	变量
0	1	x
1	2	y
2	3	x

6.12.1 AddTerms() 增加多项

该功能向当前表达式（后）添加一组新的项。

OPTVLinExpr::**AddTerms**(const double *coefs, const *OPTVVar* *vars, int cnt)

此函数将 $\text{coefs}_i * \text{var}_i, \forall i \in [0, \text{cnt}]$ 添加到当前线性表达式后。

参数

- **coefs** –C-风格的系数序列。
- **vars** –C-风格的变量序列。
- **cnt** –添加项的个数。

返回

无。

抛出

OPTVErrCode::*INVALID_ARGUMENT* –vars 或 coefs 为空指针或 cnt 取负值。

6.12.2 Clear() 清空

清空该表达式，*OPTVLinExpr::Size()* 将返回 0. 同时，该表达式从数值意义上将等同于 0.

OPTVLinExpr::**Clear**()

返回

无。

6.12.3 GetCoef() 获取系数

据指定索引值获得表达式中对应变量的系数。

OPTVLinExpr::**GetCoef**(int i) const

参数

i –指定的索引值。

返回

对应的变量系数。

抛出

OPTVErrCode::*INDEX_OUT_OF_RANGE* –若 i $\notin [0, \text{this->Size()}]$.

6.12.4 GetVar() 变量获取

根据指定索引值获得表达式中的对应变量。

OPTVLinExpr::**GetVar**(int i) const

参数

i –指定的索引值。

返回

对应的变量。

抛出
OPTVErrCode::*INDEX_OUT_OF_RANGE* –若 i ∉ [0, this->Size()].

6.12.5 GetConstant() 获取常数项

获得当前表达式中的常数项。

OPTVLinExpr::**GetConstant**() const

返回
表达式中的常量数值。

6.12.6 GetValue() 取值

获得该表达式的取值。

OPTVLinExpr::**GetValue**() const

返回
对应原始解的表达式的取值，即该表达式的变量值与系数的乘积之和。

抛出
OPTVErrCode::*DATA_NOT_AVAILABLE* –当前变量取值不存在。

6.12.7 Remove() 删除项

从表达式中删除项。

OPTVLinExpr::**Remove**(int i)
根据指定索引值从该表达式中删除项。

参数
i –指定的索引值。

返回
无。

抛出
OPTVErrCode::*INDEX_OUT_OF_RANGE* –若 i ∉ [0, this->Size()].

OPTVLinExpr::**Remove**(const *OPTVVar* &v)
从线性表达式中删除指定变量涉及的所有项。

参数
v –指定要删除的变量。

返回
是否成功删除任何项。

返回值
- **true** –成功删除任何项。
- **false** –未删除任何项。

抛出
OPTVErrCode::*NOT_IN_MODEL* –指定的变量（在模型中）不存在。

6.12.8 SameAs() 线性表达式对比

查询两线性表达式是否相同。

OPTVLinExpr::**SameAs**(const *OPTVLinExpr* &e2) const

参数
e2 –需要与当前线性表达式对比的线性表达式。

返回
线性表达式是否相同。

返回值

- **true** –线性表达式相同。
- **false** –线性表达式不同。

6.12.9 Size() 规模

获得该表达式的规模。

OPTVLinExpr::**Size**() const

该数值对应线性表达式中的变量个数，不论变量是否重复出现。例如表达式 `x+y` 对应 `Size = 2`，而 `x+y+x` 对应 `Size = 3`.

返回
该表达式的规模。

6.13 OptVerse 二次表达式

class **OPTVQuadExpr**

二次表达式类*OPTVQuadExpr* 用于构建二次的目标函数或约束。

二次表达式对象由变量和常量组成，通常包含三部分：二次项、线性项和常数项。其中，二次项依照第一个变量、第二个变量和系数的顺序保存在容器中，而线性部分以前一章节所述的线性表达式的形式保存。二次表达式的索引仅限于获取二次项，线性项部分的需单独获取。如下例所示的二次表达式 `1*x*x + 2*x*y + 3*x*y + 4*y*y + 5*x + 6*y + 7*x` 的描述如下：

索引	系数	变量 1	变量 2
0	1	x	x
1	2	x	y
2	3	x	y
3	4	y	y

它的线性项部分需要通过*OPTVQuadExpr::GetLinExpr()* 接口访问，使用方法可参考*线性表达式*：

索引	系数	变量
0	5	x
1	6	y
2	7	x

6.13.1 AddConstant() 增加常数项

向二次表达式增加常数项。

OPTVQuadExpr::**AddConstant**(double c)

此函数向当前二次表达式添加常数项，但不是覆盖。

参数

c –要添加的常数。

返回

无。

6.13.2 AddTerm() 增加单项

向当前二次表达式后添加一个线性项。

OPTVQuadExpr::**AddTerm**(double coeff, const *OPTVVar* &var)

该函数向当前二次表达式后添加 coeff ∗ var 项。

参数

- **coeff** –系数值。
- **var** –变量对象。

返回

无。

抛出

OPTVErrCode::*INVALID_ARGUMENT* –需添加的变量 `var` 为空。

向当前二次表达式后添加一个二次线性项。

OPTVQuadExpr::**AddTerm**(double coeff, const *OPTVVar* &var1, const *OPTVVar* &var2)

该函数向当前二次表达式后添加 coeff ∗ var1 ∗ var2 项。

参数

- **coeff** –系数值。
- **var1** –该项的第一个变量。
- **var2** –该项的第二个变量。

返回

无。

抛出

OPTVErrCode::*INVALID_ARGUMENT* –需添加的项中变量 `var1` 或 `var2` 为空。

6.13.3 AddTerms() 增加多项

向当前二次表达式后添加多个线性项。

OPTVQuadExpr::**AddTerms**(const double *coeff, const *OPTVVar* *var, int cnt)

该函数向当前二次表达式后添加 $\text{coeff}_i * \text{var}_i, \forall i \in [0, \text{cnt}]$.

参数

- **coeff** –C-风格的系数序列。
- **var** –C-风格的变量序列。
- **cnt** –需要添加的项的个数。

返回

无。

抛出

OPTVErrCode::*INVALID_ARGUMENT* –参数 `var` 或 `coeff` 为空指针，或 `cnt` 取负值。

向当前二次表达式后添加多个二次性项。

OPTVQuadExpr::**AddTerms**(const double *coeff, const *OPTVVar* *var1, const *OPTVVar* *var2, int cnt)

该函数向当前二次表达式后添加 $\text{coeff}_i * \text{var1}_i * \text{var2}_i, \forall i \in [0, \text{cnt}]$.

参数

- **coeff** –C-风格的系数序列。
- **var1** –C-风格的变量 1 序列。
- **var2** –C-风格的变量 2 序列。
- **cnt** –需要添加的项的个数。

返回

无。

抛出

OPTVErrCode::*INVALID_ARGUMENT* –参数 `var1`, `var2` 或 `coeff` 为空指针，或 `cnt` 取负值。

6.13.4 Clear() 清空

清空该表达式。

OPTVQuadExpr::**Clear**()

该函数清空该表达式涉及的所有存储容器，*OPTVQuadExpr::Size()* 将返回 `0`. 同时，该表达式也清空相应的线性表达式部分，重置常数项为 `0`.

返回

无。

6.13.5 GetCoeff() 获取系数

据指定索引值获得表达式中对应二次项的系数。

OPTVQuadExpr::**GetCoeff**(int i) const

参数
i –指定的索引值。

返回
对应的二次项系数。

抛出
OPTVErrCode::*DATA_NOT_AVAILABLE* –若 i $\notin [0, \texttt{this->Size()}]$.

6.13.6 GetConstant() 获取常量

获得当前表达式中的常数项。

OPTVQuadExpr::**GetConstant**() const

返回
表达式中的常量数值。

6.13.7 GetLinExpr() 线性部分

获取二次表达式的线性部分。

OPTVQuadExpr::**GetLinExpr**() const

返回
二次表达式的线性部分。

6.13.8 GetValue() 取值

获得该二次表达式的取值。

OPTVQuadExpr::**GetValue**() const

返回
对应原始解的表达式的取值。

抛出
OPTVErrCode::*DATA_NOT_AVAILABLE* –当前变量取值不存在。

6.13.9 GetVar1() 获取变量 1

根据指定索引值获得表达式中的对应二次项的第一个变量。

OPTVQuadExpr::**GetVar1**(int i) const

该函数返回表达式中的对应二次项的第一个变量，如表达式 `x*x + x*y + y*y` 的描述如下：

索引	变量 1	变量 2
0	x	x
1	x	y
2	y	y

参数

i –指定的二次项对应的索引值。

返回

对应二次项的第一个变量。

抛出

OPTVErrCode::*DATA_NOT_AVAILABLE* –若 i ∉ [0, this->Size()].

6.13.10 GetVar2() 获取变量 2

根据指定索引值获得表达式中的对应二次项的第二个变量。请参考*OPTVQuadExpr::GetVar1()* 和示例。

OPTVQuadExpr::**GetVar2**(int i) const

该函数返回表达式中的对应二次项的第二个变量。

参数

i –指定的二次项对应的索引值。

返回

对应二次项的第二个变量。

抛出

OPTVErrCode::*DATA_NOT_AVAILABLE* –若 i ∉ [0, this->Size()].

6.13.11 Remove() 删除项

从表达式中删除二次项。

OPTVQuadExpr::**Remove**(int i)

根据指定索引值从二次表达式中删除二次项，但不可通过此函数删除线性项的任何部分。

参数

i –指定的索引值。

返回

无。

抛出

OPTVErrCode::*DATA_NOT_AVAILABLE* –若 i ∉ [0, this->Size()].

从表达式中删除变量。

OPTVQuadExpr::**Remove**(const *OPTVVar* &v)

从二次表达式中（包括线性部分）删除指定变量涉及的所有项。

参数

v –指定要删除的变量。

返回

是否成功删除任何项。

返回值

- **true** –成功删除任何项。
- **false** –未删除任何项。

抛出

OPTVErrCode::*NOT_IN_MODEL* –指定的变量（在模型中）不存在。

6.13.12 SameAs() 二次表达式对比

查询两二次表达式是否相同。

OPTVQuadExpr::**SameAs**(const *OPTVQuadExpr* &e2) const

参数

e2 –需要与当前二次表达式对比的线性表达式。

返回

二次表达式是否相同。

返回值

- **true** –二次表达式相同。
- **false** –二次表达式不同。

6.13.13 Size() 规模

获得该表达式的规模。

OPTVQuadExpr::**Size**() const

该数值对应当前二次表达式中的 二**次**项的个数，不论重复与否。例如，表达式 `x*x + y*y + y` 对应 `Size = 2`，而表达式 `x*x + x*x + y*y + y` 对应 `Size = 3`.

返回

该表达式的规模。

6.14 OptVerse 模型

class **OPTVModel**

6.14.1 AddConstr() 增加单个约束

向模型中添加单个线性约束。

OPTVModel::**AddConstr**(const *OPTVLinExpr* &expr, double lhs, double rhs, const std::string &name = "")

该函数向模型中添加一个线性约束。当需要一次性添加大量约束时，建议使用批量模式（*OPTVModel::AddConstrs()*），虽然大多数情况下，效率差别不大。

参数

- **expr** –约束的线性表达式部分。
- **lhs** –约束的左端值。
- **rhs** –约束的右端值。
- **name** –约束的名字。

返回

添加到模型的约束对象。

抛出

OPTVErrCode::*INVALID_ARGUMENT* –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

6.14.2 AddConstrs() 批量增加约束

向模型中添加多个线性约束。

OPTVModel::**AddConstrs**(const *OPTVLinExpr* *expr, const double *lhs, const double *rhs, const std::string *name, int count)

该函数一次性向模型中添加多个线性约束，尤其适用于添加大量约束的情形。

参数

- **expr** –约束的线性表达式序列。
- **lhs** –约束的左端值序列。
- **rhs** –约束的右端值序列。
- **name** –约束的名字序列。
- **count** –添加约束的个数。

返回

添加到模型的约束序列。

抛出

OPTVErrCode::*INVALID_ARGUMENT* –name 为空，数值参数中存在 NaN 或者 name 包含的约束名与模型中的其他组件重名。

6.14.3 AddQConstr() 增加单个二次约束

向模型中添加单个二次约束。

OPTVModel::**AddQConstr**(const *OPTVQuadExpr* &expr, double lhs, double rhs, const std::string &name = "")

该函数向模型中添加一个二次约束。

参数

- **expr** –约束的二次表达式部分。
- **lhs** –约束的下界。
- **rhs** –约束的上界。
- **name** –约束的名字。

返回

添加到模型的约束对象。

抛出

OPTVErrCode::*INVALID_ARGUMENT* –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

6.14.4 AddGenConstrAnd() 增加一般约束 AND

向模型中添加单个 AND 一般约束。

OPTVModel::**AddGenConstrAnd**(const *OPTVVar* &resultant, const *OPTVVar* *vars, int len, const std::string &name = "")

该函数向模型中添加单个 AND 一般约束：只有所有变量非零时，结果才非零。

参数

- **resultant** –约束的布尔型返回值。
- **vars** –所有参与检验的变量的 C-风格的序列。
- **len** –vars 序列长度。
- **name** –约束的名字。

返回

添加到模型的一般约束对象。

抛出

OPTVErrCode::*INVALID_ARGUMENT* –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

6.14.5 AddGenConstrOr() 增加一般约束 OR

向模型中添加单个 OR 一般约束。

OPTVModel::**AddGenConstrOr**(const *OPTVVar* &resultant, const *OPTVVar* *vars, int len, const std::string &name = "")

该函数向模型中添加单个 OR 一般约束：只要有一个变量非零，结果就非零。

参数

- **resultant** –约束的布尔型返回值。

- **vars** –所有参与检验的变量的 C-风格的序列。
- **len** –vars 序列长度。
- **name** –约束的名字。

返回
添加到模型的一般约束对象。

抛出
OPTVErrCode::*INVALID_ARGUMENT* –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

6.14.6 AddGenConstrIndicator() 增加一般约束 INDICATOR

向模型中添加单个 INDICATOR 一般约束。

OPTVModel::**AddGenConstrIndicator**(const *OPTVVar* &binVar, int binVal, const *OPTVLinExpr* &expr, char sense, double rhs, const std::string &name = "")

该函数向模型中添加单个 INDICATOR 一般约束：当且仅当相应的逻辑条件满足，取值为 1，否则取值为 0.

参数

- **binVar** –二进制指示变量。
- **binVal** –要求线性约束必须满足的二进制指示变量的值 (0 or 1).
- **expr** –参与检验的线性约束表达式。
- **sense** – 线性约束的含义，如*OPTV_SENSE_LESS*，*OPTV_SENSE_EQUAL*，或*OPTV_SENSE_GREATER*.
- **rhs** –参与检验的线性约束右端值。
- **name** –约束的名字。

返回
添加到模型的一般约束对象。

抛出
OPTVErrCode::*INVALID_ARGUMENT* –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

6.14.7 AddGenConstrAbs() 增加一般约束 ABS

向模型中添加单个 ABS 一般约束：$r = \text{abs}(x)$。此约束条件保证因变量 r 的值为自变量 x 的绝对值，即 $r = |x|$, 其中 $r \geq 0$。

OPTVModel::**AddGenConstrAbs**(const *OPTVVar* &resultant, const *OPTVVar* &inputVar, const std::string &name = "")

该函数向模型中添加单个 ABS 一般约束。

参数

- **res_var** –结果变量，该变量的值为另一变量的绝对值。
- **input_var** –该变量的绝对值将被结果变量采用。
- **str name** –约束的名字。

返回
添加到模型的一般约束对象。

抛出
OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他约束重名。

6.14.8 AddGenConstrNorm() 增加一般约束 NORM

向模型中添加单个 NORM 一般约束: $r = \text{norm}\{x_1, \ldots, x_n\}$，其中因变量 r 的值为向量 $x_1, \ldots, x_n$ 的范数。

OPTVModel::**AddGenConstrNorm**(const *OPTVVar* &resultant, const *OPTVVar* *vars, int len, int normType, const std::string &name = "")

该函数向模型中添加单个 NORM 一般约束。

参数

- **resultant** –结果变量，其值为参数向量的范数。
- **vars** –用于计算向量范数的变量。
- **len** –向量长度。
- **normType** –范数类型：-1 为 L^∞ 范数，0 为 L^0 范数，1 为 L^1 范数。
- **name** –约束的名字。

返回
添加到模型的一般约束对象。

抛出
OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他约束重名。

6.14.9 AddGenConstrMax() 添加一般约束 MAX

向模型中添加单个 MAX 约束：$r = \max\{x_1, \ldots, x_n\}$。

该约束确保因变量 r 的取值为变量向量 $x_1, \ldots, x_n$ 中的最大值。

OPTVModel::**AddGenConstrMax**(const *OPTVVar* &resultant, const *OPTVVar* *vars, int len, const std::string &name = "", double constant = -OPTV_INF)

该函数向模型中添加单个 MAX 约束。

参数

- **resultant** –结果变量，该变量的值为其他变量的最大值。
- **vars** –变量向量，该向量的最大值会被结果变量采用。
- **len** –vars 向量长度。
- **name** –约束名称。
- **constant** –[**可选**] 参与 MAX 操作的常量。

返回
一个添加到模型中的一般约束对象

抛出

OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他约束重名。

6.14.10 AddGenConstrMin() 添加一般约束 MIN

向模型中添加单个 MIN 约束：$r = \min\{x_1, \ldots, x_n\}$ 。

该约束确保因变量 r 的取值为变量向量 $x_1, \ldots, x_n$ 中的最小值。

OPTVModel::**AddGenConstrMin**(const *OPTVVar* &resultant, const *OPTVVar* *vars, int len, const std::string &name = "", double constant = OPTV_INF)

该函数向模型中添加单个 MIN 约束。

参数

- **resultant** –结果变量，该变量的值为其他变量的最小值。
- **vars** –变量向量，该向量的最小值会被结果变量采用。
- **len** –vars 向量长度。
- **name** –约束名称。
- **constant** –[**可选**] 参与 MIN 操作的常量。

返回

一个添加到模型中的一般约束对象。

抛出

OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他约束重名。

6.14.11 AddSOS() 增加单个 SOS 约束

向模型中添加单个 SOS 约束。

OPTVModel::**AddSOS**(const *OPTVVar* *vars, const double *weights, int len, int type, const std::string &name = "")

参数

- **vars** –SOS 约束所涉及变量的 C-风格序列。
- **weights** –SOS 的变量权重的 C-风格序列。
- **len** –vars 和 weights 的序列长度。
- **type** –SOS 类型，取值为 1 或 2.
- **name** –SOS 约束的名字。

返回

添加到模型的 SOS 约束对象。

抛出

- *OPTVErrCode*::*INVALID_ARGUMENT* –type 取值不是 1 或 2.
- *OPTVErrCode*::*NOT_IN_MODEL* –vars 序列中的某个变量不属于当前模型。
- *OPTVErrCode*::*INDEX_OUT_OF_RANGE* –当前模型存在无效下标，比如模型关联的环境发生溢出时，执行本操作有可能抛出此错误。

6.14.12 AddVar() 增加变量

向模型中添加单个变量。

OPTVModel::**AddVar**(double lb, double ub, double obj, char type, const std::string &name = "")

该函数向模型中添加一个变量。当需要一次性添加大量变量时，建议使用批量模式（*OPTVModel::AddVars()*），虽然大多数情况下，效率差别不大。

参数

- **lb** –变量的下界。
- **ub** –变量的上界。
- **obj** –变量在目标函数中的系数。
- **type** –变量类型，取值包括*OPTV_CONTINUOUS*, *OPTV_INTEGER*, *OPTV_BINARY*.
- **name** –变量名称。

返回

添加到模型的变量对象。

抛出

OPTVErrCode::*INVALID_ARGUMENT* –未知变量类型，name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

OPTVModel::**AddVar**(double lb, double ub, double obj, char type, const *OPTVColumn* &col, const std::string &name = "")

该函数向模型中添加一个变量，并可以设置现有约束中的相应系数。当需要一次性添加大量变量时，建议使用批量模式（*OPTVModel::AddVars()*），虽然大多数情况下，效率差别不大。

参数

- **lb** –变量的下界。
- **ub** –变量的上界。
- **obj** –变量在目标函数中的（线性）系数。
- **type** –变量类型，取值包括*OPTV_CONTINUOUS*, *OPTV_INTEGER*, *OPTV_BINARY*.
- **col** –设置约束系数的列对象。
- **name** –变量名称。

返回

添加到模型的变量对象。

抛出

OPTVErrCode::*INVALID_ARGUMENT* –未知变量类型，name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

6.14.13 AddVars() 批量增加变量

向模型中添加多个变量。

OPTVModel::**AddVars**(const double *lb, const double *ub, const double *obj, const char *type, const std::string *name, int count)

该函数一次性向模型中添加多个变量，尤其适用于添加大量变量的情形。

参数

- **lb** –变量的下界序列。
- **ub** –变量的上界序列。
- **obj** –变量序列对应的目标系数。
- **type** –变量类型序列，包括*OPTV_CONTINUOUS*, *OPTV_INTEGER*, *OPTV_BINARY*.
- **name** –变量名称序列。
- **count** –添加变量的个数。

返回

添加到模型的变量序列。

抛出

OPTVErrCode::*INVALID_ARGUMENT* –`type` 包含未知变量类型，`name` 为空，数值参数中存在 `NaN` 或者 `name` 包含的变量名与模型中的其他组件重名。

OPTVModel::**AddVars**(const double *lb, const double *ub, const double *obj, const char *type, const *OPTVColumn* *col, const std::string *names, int count)

该函数向模型中添加多个变量，并可以设置现有约束中的相应系数。当需要一次性添加大量变量时，建议使用此功能，虽然大多数情况下，效率差别不大。

参数

- **lb** –变量的下界序列。
- **ub** –变量的上界序列。
- **obj** –变量序列对应的（线性）目标系数。
- **type** –变量类型序列，包括*OPTV_CONTINUOUS*, *OPTV_INTEGER*, *OPTV_BINARY*.
- **col** –设置约束系数的列对象序列。
- **names** –变量名称序列。
- **count** –添加变量的个数。

返回

添加到模型的变量序列。

抛出

OPTVErrCode::*INVALID_ARGUMENT* –`type` 包含未知变量类型，`name` 为空，数值参数中存在 `NaN` 或者 `name` 包含的变量名与模型中的其他组件重名。

6.14.14 CheckSolution() 校验解

检验指定的解是否可行。

OPTVModel::**CheckSolution**(const std::vector<double> &solution, double tol = 1e-8) const

该函数供用户检验指定的解是否满足当前模型的可行性要求，但 **不可以**检验内部存储的解。

参数

- **solution** –(列）解向量。
- **tol** –约束可行性整数性检查精度（**可选**，默认值为 10^{-8})。

返回

解对模型是否可行。

返回值

- **false** –解对模型不可行。
- **true** –解对模型是可行的。

6.14.15 ComputeIIS() 不可约不一致子系统

计算不可约不一致子系统（IIS)，关于 IIS 更多细节请参考*不可约不一致子系统 (IIS) 快速入门*。

OPTVModel::**ComputeIIS**()

返回

IIS 是否计算成功。

返回值

- **false** –IIS 计算不成功。
- **true** –成功计算 IIS。

6.14.16 FeasRelax() 不可行问题的诊断及分析

进行不可行问题的诊断及分析，更多细节请参考*LP 不可行诊断及修复快速入门*。

OPTVModel::**FeasRelax**()

该函数进行不可行问题的诊断及分析，目前仅支持 L^1 范数。

返回

不可行问题修复的状态。

返回值

- **0** –当前问题是可行的。
- **-1** –不可行问题修复失败。
- **>0** –Phase 1 最小违背量的目标函数值。

6.14.17 GetCoef() 查询变量系数

查询给定约束中指定变量的系数。

OPTVModel::**GetCoef**(const *OPTVConstr* &con, const *OPTVVar* &var) const

参数

- **con** –需要查询的约束。
- **var** –需要查询的变量。

返回

查询变量在该约束中的系数。

抛出

- *OPTVErrCode*::*INVALID_ARGUMENT* –约束或变量不存在。
- *OPTVErrCode*::*DATA_NOT_AVAILABLE* –数据不存在，或约束的线性表达式部分不可用。

6.14.18 GetColumn() 获取列

获取指定变量对应的列对象。

OPTVModel::**GetColumn**(const *OPTVVar* &v) const

参数

v –需要查询的变量。

返回

指定变量对应的列对象。

抛出

- *OPTVErrCode*::*DATA_NOT_AVAILABLE* –该变量不存在于当前模型。
- *OPTVErrCode*::*INDEX_OUT_OF_RANGE* –变量索引值*Index* 超过了模型的变量总数。若当前模型需要执行更新，或者当前变量属于其它模型时，该错会被抛出。

6.14.19 GetConstrByName() 通过名称访问约束

通过名称获取约束。

OPTVModel::**GetConstrByName**(const std::string &name) const

参数

name –需要访问的约束名称。

返回

名称匹配 name 的约束。

抛出

OPTVErrCode::*DATA_NOT_AVAILABLE* –给定的名称为空，或模型不存在命名为 name 的约束。

6.14.20 GetConstr() 获取单个约束

根据指定索引值获得模型中对应的约束。

OPTVModel::**GetConstr**(int i) const

参数
i –指定的索引值。

返回
对应的约束。

抛出
OPTVErrCode::*INDEX_OUT_OF_RANGE* –指定的索引值超出了模型的约束总数。

此函数中的参数 *i* 是指全局下标，与约束的类型无关。例如，用户依次添加了一个线性约束和一个二次约束，那么该线性约束的索引值为 *0* 而二次约束的索引值为 *1*.

6.14.21 GetConstrs() 获取所有线性约束

获得模型中的所有线性约束。

OPTVModel::**GetConstrs**() const

返回
包含模型中所有线性约束的向量。

6.14.22 GetGenConstrAnd() 获取单个 AND 约束

获得模型中特定的 AND 一般约束数据。

OPTVModel::**GetGenConstrAnd**(const *OPTVGenConstr* &genConstr, *OPTVVar* &resultant,
std::vector<*OPTVVar*> &vars) const

参数

- **genConstr** –需要查询的一般约束。
- **resultant** –该一般约束对应的结果变量。
- **vars** –参与检验该一般约束的所有变量的列表。

抛出
OPTVErrCode::*NOT_IN_MODEL* –模型中不存在该一般约束，当该一般约束已从模型中被删除或者是由默认的构造函数创建时，会抛出此错误。

6.14.23 GetGenConstrOr() 获取单个 OR 约束

获得模型中特定的 OR 一般约束数据。

OPTVModel::**GetGenConstrOr**(const *OPTVGenConstr* &genConstr, *OPTVVar* &resultant,
std::vector<*OPTVVar*> &vars) const

参数

- **genConstr** –需要查询的一般约束。
- **resultant** –该一般约束对应的结果变量。

- **vars** –参与检验该一般约束的所有变量的列表。

抛出

OPTVErrCode::*NOT_IN_MODEL* –模型中不存在该一般约束，当该一般约束已从模型中被删除或者是由默认的构造函数创建时，会抛出此错误。

6.14.24 GetGenConstrIndicator() 获取单个 INDICATOR 约束

获得模型中特定的 INDICATOR 一般约束数据。

OPTVModel::**GetGenConstrIndicator**(const *OPTVGenConstr* &genConstr, *OPTVVar* &binVar, int &binVal, *OPTVLinExpr* &expr, char &sense, double &rhs) const

参数

- **genConstr** –需要查询的一般约束。
- **binVar** –二进制指示变量。
- **binVal** –要求线性约束必须满足的二进制指示变量的值 (0 or 1).
- **expr** –参与检验的线性约束表达式。
- **sense** – 线性约束的含义，如*OPTV_SENSE_LESS*, *OPTV_SENSE_EQUAL*, 或*OPTV_SENSE_GREATER*.
- **rhs** –参与检验的线性约束右端值。

抛出

OPTVErrCode::*NOT_IN_MODEL* –模型中不存在该一般约束，当该一般约束已从模型中被删除或者是由默认的构造函数创建时，会抛出此错误。

6.14.25 GetQConstr() 获取二次约束

根据指定索引值获得模型中对应的二次约束。

OPTVModel::**GetQConstr**(int i) const

参数

i –指定的索引值。

返回

对应的二次约束。

抛出

OPTVErrCode::*INDEX_OUT_OF_RANGE* –指定的索引值超出了模型的约束总数。

此函数中的参数 *i* 是指全局下标，与约束的类型无关。例如，用户依次添加了一个线性约束和一个二次约束，那么该线性约束的索引值为 *0* 而二次约束的索引值为 *1*.

6.14.26 GetQConstrs() 获取所有二次约束

获得模型中的所有二次约束。

OPTVModel::**GetQConstrs**() const

返回
包含模型中所有二次约束的向量。

6.14.27 GetSOS() 获取单个 SOS 约束

根据索引值获取指定类型的 SOS 约束。

OPTVModel::**GetSOS**(int i, int type) const

参数

- **i** –指定 SOS 约束在模型中的索引值。
- **type** –SOS 类型，取值为 `1` 或 `2`.

返回
对应索引值 `i` 的指定类型的 SOS 约束。

抛出

- *OPTVErrCode*::*INDEX_OUT_OF_RANGE* –索引值 `i` 超出了模型的该类型的 SOS 约束总数。
- *OPTVErrCode*::*INVALID_ARGUMENT* –`type` 取值不是 `1` 或 `2`.

这里，参数 `i` 特指某一类型 SOS 约束集中的索引值，例如，如果用户先后添加了类型为 `1` 和 `2` 的两个 SOS 约束，那么两者的索引值都是 `0`.

6.14.28 GetSOSData() 获取 SOS 约束数据

OPTVModel::**GetSOSData**(const *OPTVSOS* &sos, std::vector<*OPTVVar*> &vars, std::vector<double> &weights) const

获取 SOS 约束的内部数据。

参数

- **sos** –当前访问的 SOS 约束对象。
- **vars** –变量序列的空向量。如果非空，该容器在本次函数调用中会被清空。
- **weights** –变量权重序列的空向量。如果非空，该容器在本次函数调用中会被清空。

返回
无。

抛出

- *OPTVErrCode*::*DATA_NOT_AVAILABLE* –SOS 约束是由默认构造函数生成的。
- *OPTVErrCode*::*NOT_IN_MODEL* –SOS 约束属于另外一个模型，或已经从本模型中被删除。
- *OPTVErrCode*::*INDEX_OUT_OF_RANGE* –约束下表越界、超出容器范围，可能需要对当前模型执行更新。

6.14.29 GetSOSs() 获取多个 SOS 约束

获取指定类型的所有 SOS 约束。

OPTVModel::**GetSOSs**(int type) const

参数
type –SOS 类型，取值为 1 或 2.

返回
包含模型中所有类型为 type 的 SOS 约束的向量。

抛出

- *OPTVErrCode*::*INDEX_OUT_OF_RANGE* –SOS 约束索引值存在越界、超出了模型的该类型的 SOS 约束总数，可能需要对当前模型执行更新。
- *OPTVErrCode*::*INVALID_ARGUMENT* –type 取值不是 1 或 2.

6.14.30 GetObjective() 获取线性目标函数

获取线性目标函数。

OPTVModel::**GetObjective**() const

返回
目标函数的线性表达式。

6.14.31 GetQObjective() 获取二次目标函数

获取二次目标函数。

OPTVModel::**GetQObjective**() const

返回
目标函数的二次表达式。

6.14.32 GetRow() 获取行

获取指定约束对应的线性函数表达式。

OPTVModel::**GetRow**(const *OPTVConstr* &c) const

参数
c –指定的约束对象。

返回
该约束对应的线性表达式。

抛出

- *OPTVErrCode*::*DATA_NOT_AVAILABLE* –该约束不存在于当前模型。
- *OPTVErrCode*::*INDEX_OUT_OF_RANGE* –当前约束的索引 *Index* 超过了模型的约束总数。若当前模型需要执行更新，或者当前约束属于其它模型时，该错会被抛出。

6.14.33 Get() 获取参数和属性

获取参数

获取求解器的参数取值，可用参数请参考*OptVerse 参数*。

OPTVModel::**Get**(*OPTVIntParam* param)

参数
param –整数型参数。

返回
参数取值。

抛出
OPTVErrCode::*UNKNOWN_PARAMETER* –未知参数。

OPTVModel::**Get**(*OPTVDblParam* param)

参数
param –双精度型参数。

返回
参数取值。

抛出
OPTVErrCode::*UNKNOWN_PARAMETER* –未知参数。

OPTVModel::**Get**(*OPTVStrParam* param)

参数
param –字符串型参数。

返回
参数取值。

抛出
OPTVErrCode::*UNKNOWN_PARAMETER* –未知参数。

获取属性

获取属性的取值，可用属性请参考*OptVerse 属性*。

OPTVModel::**Get**(*OPTVIntAttr* attr)

参数
attr –整数型属性。

返回
属性取值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知属性。

OPTVModel::**Get**(*OPTVDblAttr* attr)

参数
attr –双精度型属性。

返回
属性取值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知属性。

OPTVModel::**Get**(*OPTVLongAttr* attr)

参数
attr –长整型属性。

返回
属性取值。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知属性。

6.14.34 GetVarByName() 通过名称访问变量

通过名称获取变量。

OPTVModel::**GetVarByName**(const std::string &name) const

参数
name –需要访问的变量名称。

返回
名称匹配 name 的变量。

抛出
OPTVErrCode::*DATA_NOT_AVAILABLE* –给定的名称为空，或模型不存在命名为 name 的变量。

6.14.35 GetVar() 获取单个变量

根据指定索引值获得模型中对应的变量。

OPTVModel::**GetVar**(int i) const

参数
i –指定的索引值。

返回
对应的变量。

抛出
OPTVErrCode::*INDEX_OUT_OF_RANGE* –指定的索引值超出了模型的变量总数。

6.14.36 GetVars() 获取所有变量

获得模型中的所有变量。

OPTVModel::**GetVars**() const

返回
包含模型中所有变量的向量。

6.14.37 Optimize() 求解模型

对模型进行求解。

OPTVModel::**Optimize**() const

返回
无。

抛出
OPTVErrCode::*MATRIX_NOT_PSD* –模型中包含二次目标函数或者二次约束，而该表达式的矩阵不满足正半定性质。当求解的模型 **包含**二次约束时，QCQP 求解器要求模型的二次目标函数或二次约束的矩阵都是正半定的。

求解器采用哪种优化算法取决于求解器参数配置和模型属性，例如变量类型被用以自动识别 LP, QP, MIP 及网络流问题。

6.14.38 Presolve() 预处理

对模型进行预处理。

OPTVModel::**Presolve**() const
该函数会应用预处理流程，返回一个新的*OPTVModel* 对象。原模型的状态被修改为 被预处理.

返回
预处理后的模型。

6.14.39 Read() 读取模型和基

从文件中读取读取模型和基。

OPTVModel::**Read**(const std::string &fileName)
该函数供读取基信息，也可以读取模型文件，然而建议由构造函数读取文件中的模型，参考文件格式。

表 13: 支持读入的文件格式

.mps	.lp	.bas

表 14: 支持读入的压缩格式

.gz	.z	.Z

参数
fileName –C++ 字符串文件名。

返回
无。

抛出
OPTVErrCode::*ERROR_FILE_READ* –文件名或后缀为空，文件存在错误格式。

6.14.40 Remove() 删除模型组件

从模型中删除变量。

OPTVModel::**Remove**(const *OPTVVar* &var)

该函数将变量从模型中删除，对应的索引会被相应地调整。例如，若下列模型 `model` 构建时包含以下变量

变量	索引
x0	0
x1	1
x2	2
x3	3

调用 `model.Remove(x0)` 后，相应变化为

变量	索引
x1	0
x2	1
x3	2

参数
var –指定要删除的变量。

返回
无。

抛出

- *OPTVErrCode*::*NOT_IN_MODEL* –模型不包含此变量。
- *OPTVErrCode*::*DATA_NOT_AVAILABLE* –该变量属于其他模型，而不属于当前模型。
- *OPTVErrCode*::*INDEX_OUT_OF_RANGE* –变量的下标超出了模型的变量总数，可能需要执行模型更新。

从模型中删除约束。

OPTVModel::**Remove**(const *OPTVConstr* &constr)

该函数将约束从模型中删除，对应的索引会被相应地调整。例如，若下列模型 `model` 构建时包含以下约束

约束	索引
c0	0
c1	1
c2	2
c3	3

调用 `model.Remove(c0)` 后，相应变化为

约束	索引
c1	0
c2	1
c3	2

参数

constr –指定要删除的约束。

返回

无。

抛出

- *OPTVErrCode*::*NOT_IN_MODEL* –模型不包含此约束。
- *OPTVErrCode*::*DATA_NOT_AVAILABLE* –该约束属于其他模型，而不属于当前模型。
- *OPTVErrCode*::*INDEX_OUT_OF_RANGE* –约束的下标超出了模型的变量总数，可能需要执行模型更新。

OPTVModel::**Remove**(const *OPTVGenConstr* &genConstr)

从模型中删除单个一般约束。执行后对应的相同类型的一般约束索引会被相应地调整。例如，若下列模型 `model` 构建时包含以下一般约束：

一般约束	类型	索引
and0	AND	0
and1	AND	1
or0	OR	0
or1	OR	1

调用 `model.Remove(and0)` 后，相应变化为

一般约束	类型	索引
and1	AND	0
or0	OR	0
or1	OR	1

参数

genConstr –指定要删除的一般约束。

返回
无。

抛出

- *OPTVErrCode*::*NOT_IN_MODEL* –模型不包含此一般约束。
- *OPTVErrCode*::*DATA_NOT_AVAILABLE* –该一般约束属于其他模型，而不属于当前模型。
- *OPTVErrCode*::*INDEX_OUT_OF_RANGE* –该一般约束的下标超出了模型的变量总数，可能需要执行模型更新。

OPTVModel::**Remove**(const *OPTVSOS* &sos)

从模型中删除单个 SOS 约束。与其他约束类型不同的是，删除该 SOS 约束后，相同类型的约束的索引**不会**改变。例如，若模型 `model` 创建时包含如下约束：

约束	类型	索引
sos1_1	1	0
sos1_2	1	1
sos1_3	1	2
sos2_1	2	0

调用 `model.Remove(sos1_2)` 后，相应变化为

约束	类型	索引
sos1_1	1	0
sos1_3	1	2
sos2_1	2	0

参数
sos –指定要删除的 SOS 约束。

返回
无。

抛出

- *OPTVErrCode*::*NOT_IN_MODEL* –SOS 约束不属于任何模型。
- *OPTVErrCode*::*DATA_NOT_AVAILABLE* –SOS 不属于当前模型。
- *OPTVErrCode*::*INDEX_OUT_OF_RANGE* –SOS 约束的索引值超出了模型的该类型的 SOS 约束总数，可能需要对当前模型执行更新。

6.14.41 Relax() 松弛模型

进行线性规划 (LP) 松弛。

OPTVModel`::Relax()` const

该函数将原模型中的所有整数类型变量替换为连续类型变量，生成和返回一个新的*OPTVModel* 对象，对原模型不做任何修改。

返回
模型的 LP 松弛副本。

6.14.42 Reset() 重置模型

将模型重置为 *未求解成功*状态。

OPTVModel`::Reset` (bool clearAll = false)

该函数会清除内部解的信息，默认情况下，包括解的值、LP 基和模型状态。`clearAll` 标志是否清除额外的模型属性值（包括 MIP 热启动值，MIP 分支优先度，LP 的禁用属性状态）。

参数
clearAll –清除额外信息（默认为 `false`）.

返回
无。

6.14.43 GetObjSA() 目标函数敏感度分析

目标函数敏感度分析，关于目标函数敏感度分析细节，请参考*LP 敏感度分析快速入门*。

OPTVModel`::GetObjSA` (std::vector<double> &down, std::vector<double> &up, const std::vector<int> &list) const

参数

- **down** –敏感度区间左端点。
- **up** –敏感度区间右端点。
- **list** –需要进行目标函数敏感度分析的变量索引集合。

返回
敏感度分析是否成功。

返回值

- **false** –敏感度分析失败。
- **true** –敏感度分析成功。

6.14.44 GetVarLbSA() 变量下界敏感度分析

变量下界敏感度分析，有关变量下界敏感度分析细节，请参考*LP 敏感度分析快速入门*。

OPTVModel::**GetVarLbSA**(std::vector<double> &down, std::vector<double> &up, const std::vector<int> &list) const

参数

- **down** –敏感度区间左端点。
- **up** –敏感度区间右端点。
- **list** –需要进行变量下界敏感度分析的约束索引集合。

返回
敏感度分析是否成功。

返回值

- **false** –敏感度分析失败。
- **true** –敏感度分析成功。

6.14.45 GetVarUbSA() 变量上界敏感度分析

变量上界敏感度分析，有关变量上界敏感度分析细节，请参考*LP 敏感度分析快速入门*。

OPTVModel::**GetVarUbSA**(std::vector<double> &down, std::vector<double> &up, const std::vector<int> &list) const

参数

- **down** –敏感度区间左端点。
- **up** –敏感度区间右端点。
- **list** –需要进行变量上界敏感度分析的约束索引集合。

返回
敏感度分析是否成功。

返回值

- **false** –敏感度分析失败。
- **true** –敏感度分析成功。

6.14.46 GetConstrLbSA() 约束左端值敏感度分析

约束左端值/下界敏感度分析，有关约束敏感度分析细节，请参考*LP 敏感度分析快速入门*。

OPTVModel::**GetConstrLbSA**(std::vector<double> &down, std::vector<double> &up, const std::vector<int> &list) const

参数

- **down** –敏感度区间左端点。
- **up** –敏感度区间右端点。
- **list** –需要进行左端值敏感度分析的约束索引集合。

返回
敏感度分析是否成功。

返回值

- **false** –敏感度分析失败。
- **true** –敏感度分析成功。

6.14.47 GetConstrUbSA() 约束右端值敏感度分析

约束右端值/上界敏感度分析，有关约束敏感度分析细节，请参考*LP 敏感度分析快速入门*。

OPTVModel::**GetConstrUbSA**(std::vector<double> &down, std::vector<double> &up, const std::vector<int> &list) const

参数

- **down** –敏感度区间左端点。
- **up** –敏感度区间右端点。
- **list** –需要进行右端值敏感度分析的约束索引集合。

返回
敏感度分析是否成功。

返回值

- **false** –敏感度分析失败。
- **true** –敏感度分析成功。

6.14.48 SetCoef() 设置系数

设置给定约束中指定变量的系数。

OPTVModel::**SetCoef**(const *OPTVConstr* &con, const *OPTVVar* &var, double newValue)

参数

- **con** –需要修改的约束。
- **var** –需要修改系数的变量。
- **newValue** –新的系数值。

返回
无。

抛出

- *OPTVErrCode*::*INVALID_ARGUMENT* –约束或变量不存在，或数值参数中存在 NaN.
- *OPTVErrCode*::*DATA_NOT_AVAILABLE* –数据不存在，或约束的线性表达式部分不可用。

6.14.49 SetObjective() 设置目标函数

设置线性目标函数。

OPTVModel::**SetObjective**(const *OPTVLinExpr* &expr, *OPTVSense* sense = *OPTVSense*::*MINIMIZE*)

该函数清除当前目标函数（相对于在其基础上进行叠加）。

参数

- **expr** –需要设定的目标函数线性表达式。
- **sense** – **可选**，最小化或最大化，默认值为*OPTVSense::MINIMIZE*，参考*OPTVSense*.

返回

无。

设置二次目标函数

OPTVModel::**SetObjective**(const *OPTVQuadExpr* &expr, *OPTVSense* sense = *OPTVSense*::*MINIMIZE*)

同样，该函数清除当前目标函数（相对于在其基础上进行叠加）。

参数

- **expr** –需要设定的目标函数二次表达式。
- **sense** – **可选**，最小化或最大化，默认值为*OPTVSense::MINIMIZE*，参考*OPTVSense*.

返回

无。

6.14.50 Set() 设置参数和属性

设置参数

设置求解器的参数取值，一旦模型被创建，该接口仅提供控制此模型专用参数的接口，直到调用*OPTVModel::Update()* 方法后，新修改才会在内部生效。可用参数请参考*OptVerse 参数*。

OPTVModel::**Set**(*OPTVIntParam* param, int value)

参数

- **param** –整数型参数。
- **value** –新参数值。

返回

无。

抛出

OPTVErrCode::*UNKNOWN_PARAMETER* –未知参数。

OPTVModel::**Set**(*OPTVDblParam* param, double value)

参数

- **param** –双精度型参数。
- **value** –新参数值。

返回
无。

抛出
OPTVErrCode::*UNKNOWN_PARAMETER* –未知参数。

OPTVModel::**Set**(*OPTVStrParam* param, const std::string &value)

参数

- **param** –字符串型参数。
- **value** –新参数值。

返回
无。

抛出
OPTVErrCode::*UNKNOWN_PARAMETER* –未知参数。

设置属性

设置求解器的属性值，同样，直到调用*OPTVModel::Update()* 方法后，新修改才会在内部生效。可用属性请参考*OptVerse 属性*。

OPTVModel::**Set**(*OPTVIntAttr* attr, int value)

参数

- **attr** –整数型属性。
- **value** –新属性值。

返回
无。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知属性。

OPTVModel::**Set**(*OPTVDblAttr* attr, double value)

参数

- **attr** –双精度型属性。
- **value** –新属性值。

返回
无。

抛出
OPTVErrCode::*UNKNOWN_ATTRIBUTE* –未知属性。

6.14.51 Update() 更新模型

执行模型更新。

OPTVModel::**Update**()

该函数会进行模型更新，或 lazy 构建（如必要）。只有调用该函数 **后**，模型的更新才会生效。SDK 构建的模型需要调用此函数来保证内部模型的即时性。例如，用户调用*OPTVModel::Optimize()* 前 **不需要**调用此函数，因为该函数在内部已被默认执行。

返回
无。

6.14.52 WriteIIS() IIS 结果导出

将不可约不一致子系统（IIS）写入文件（.ilp 格式），有关 IIS 的细节请参考*IIS 快速入门*。

OPTVModel::**WriteIIS**(const std::string &fileName)

参数
fileName –写出的文件名，必须包含 .ilp 后缀（例如 iis.ilp）.

返回
无。

抛出
OPTVErrCode::*ERROR_FILE_WRITE* –不支持的文件格式，或者未计算出 IIS.

备注: 此函数 **不会**计算 IIS，若 IIS 未被计算，本函数将抛出异常。

6.14.53 Write() 写出模型和基

将模型或基写出到文件，支持的文件格式请参考文件格式。

OPTVModel::**Write**(const std::string &fileName)

表 15: 支持写出的文件格式

.mps （暂不支持 QP 模型写出）	.lp	.bas	.ilp

表 16: 支持写出的压缩格式

.gz	.z	.Z

参数
fileName –C++ 字符串文件名。

返回
无。

抛出
OPTVErrCode::*ERROR_FILE_WRITE* –文件名或后缀为空或错误。

6.15 OptVerse 异常

class **OPTVException**

OptVerse 异常类。

OPTVException::**GetMessage**() const

获取异常的错误信息。

OPTVException::**GetErrorCode**() const

获取异常的错误码*OPTVErrCode*，关于错误码详细信息请参考*OptVerse 错误码*。

6.16 OptVerse 错误码

OPTVErrCode 用来产生特定的*OPTVException* 异常对象，并给出相应的详细信息。

enum class **OPTVErrCode**

OptVerse 错误码

enumerator **ERROR**

错误：错误信息，一般从求解器内部调用抛出。

enumerator **WARNING**

警告：警告信息，一般从求解器内部调用抛出。

enumerator **UNKNOWN**

未知：遇到未知错误。

enumerator **INVALID_ARGUMENT**

无效参数：传入函数的参数不满足条件，例如在设置变量类型*OPTVCharAttr::VAR_TYPE* 时，传入未定义的属性值，或者用 0 除线性表达式。

enumerator **UNKNOWN_ATTRIBUTE**

未知属性：访问或者设置未定义的属性，例如*OPTVConstr::Get(OPTVIntAttr::NUM_VARS)* 将抛出此错误码，因为*OPTVIntAttr::NUM_VARS* 不是模型的属性。

enumerator **DATA_NOT_AVAILABLE**

数据不可用：访问的属性值不可用，例如在调用*OPTVModel::Optimize()* 之前，试图获取解的信息*OPTVVar::Get(OPTVDblAttr::X)* 将会抛出此错误。

enumerator **INDEX_OUT_OF_RANGE**

下标越界：用于获取对象的索引值超出了允许范围，例如，当前模型仅有 2 个变量，调用*model.GetVar(10)* 会抛出此异常。

enumerator **UNKNOWN_PARAMETER**

未知参数：访问未定义的参数。

enumerator **VALUE_OUT_OF_RANGE**

数值越界：参数值超出了允许范围，例如尝试获取不存在的*OPTVSense*.

enumerator **ERROR_FILE_READ**

文件读取失败：读取文件时发生错误。

enumerator **ERROR_FILE_WRITE**

文件写出失败：写出文件时发生错误。

enumerator **LICENSE_CHECK_FAILED**

许可检查失败：未发现有效许可，注：许可文件 OPTV_LICENSE_FILE 需要通过环境变量设置，细节请参考 安装说明。

enumerator **INVALID_MODEL_INPUT**

模型输入无效：试图使用无效的模型输入信息。

enumerator **NOT_IMPLEMENTED**

功能不可用：试图使用的功能暂未开放。

enumerator **NOT_IN_MODEL**

模型组件不存在：访问的变量或约束在模型中不存在。

enumerator **MATRIX_NOT_PSD**

非正半定矩阵：二次表达式的矩阵不满足正半定性质。当求解的模型 **包含**二次约束时，QCQP 求解器要求模型的二次目标函数或二次约束的矩阵都是正半定的。

CHAPTER 7

Python API 参考手册

7.1 OptVerse 常量

OPTV_INF

代表无穷大的量的最大值。

类型

float

取值

`1.0e+100`

OPTV_UNDF

未定义的数值。

类型

float

取值

`1.0e+101`

7.1.1 变量类型

OPTV_BINARY

0-1 整数变量类型。

类型

str

取值

`'B'`

OPTV_INTEGER

整数变量类型。

类型

str

取值

'I'

OPTV_CONTINUOUS

连续变量类型。

类型

str

取值

'C'

7.1.2 模型求解状态

OPTV_UNKNOWN

未知模型求解状态（默认）。

类型

int

取值

0

OPTV_OPTIMAL

模型找到最优解，可以查询到最优解。

类型

int

取值

1

OPTV_INFEASIBLE

模型被证明无解。

类型

int

取值

2

OPTV_INF_OR_UNBD

模型被证明无解或者无界。

类型
int

取值
3

OPTV_UNBOUNDED

模型被证明无界。

类型
int

取值
4

OPTV_TIME_LIMIT

模型在指定时间*OPTVDblParam.TIME_LIMIT* 内没有完成求解。

类型
int

取值
5

OPTV_MEM_LIMIT

模型求解使用内存超过预设值*OPTVIntParam.MEM_LIMIT*.

类型
int

取值
6

OPTV_INTERRUPTED

用户终止模型求解（如通过 Ctrl+C）.

类型
int

取值
7

7.1.3 变量和约束的基状态

OPTV_BASIC

变量或约束为基。

类型
int

取值
0

OPTV_NONBASIC_LOWER

变量或约束非基，取下界值。

类型
int

取值
-1

OPTV_NONBASIC_UPPER

变量或约束非基，取上界值。

类型
int

取值
-2

OPTV_SUPERBASIC

变量或约束为超基。

类型
int

取值
-3

7.1.4 一般约束类型

OPTV_GENCONSTR_AND

只有所有变量非零时，结果才非零。

类型
int

取值
0

OPTV_GENCONSTR_OR

只要有一个变量非零，结果就非零。

类型
int

取值
1

OPTV_GENCONSTR_INDICATOR

返回值为布尔型，当且仅当相应的逻辑条件满足，取值为 1，否则取值为 0。

类型
int

取值
2

7.1.5 约束含义

OPTV_SENSE_LESS
小于或等于。

类型
str

取值
<

OPTV_SENSE_EQUAL
等于。

类型
str

取值
=

OPTV_SENSE_GREATER
大于或等于。

类型
str

取值
>

7.2 OptVerse 参数

参数对应着用户使用的求解器内部配置，可以通过*OPTVEnv* 对象（模型构建前）或*OPTVModel* 对象（模型构建后）来设置。

7.2.1 整数型参数

class OPTVIntParam

表 1: OptVerse 整数型参数

名称	描述
BRANCH_DIR	MIP 分支方向
CROSSOVER	内点算法 crossover 策略
CUT_PASSES	根节点 (MIP) 割平面方法最大轮数
CUTS	MIP 割平面方法强度
MEM_LIMIT	用户指定的内存使用上限 (MB)
METHOD	LP 求解算法（如单纯形、内点等）
MIP_FOCUS	MIP 高层级的解策略
OPTIMAL_BASIS	清理所得解获取最优基
OUTPUT_FLAG	禁止日志打印
POOL_SOLUTIONS	解池存储的 MIP 解个数
PRESOLVE_LEVEL	预处理程度控制
SEED	PRNG 随机数种子
SOL_FORMAT	输出解的形式（稀疏或者稠密）
SOLUTION_NUMBER	用来指定 MIP 解
START_NODE_LIMIT	MIP 分支定界节点个数上限
SUBMIP_NODES	MIP 启发式策略模块访问的节点个数上限
THREADS	线程数
VAR_BRANCH	(MIP) 变量分支方法

BRANCH_DIR 分支方向

`OPTVIntParam.`**`BRANCH_DIR`**

MIP 分支方向。

默认值: 0

最小值: -1

最大值: 1

该参数控制分支剪界搜索中子节点的选取，取值-1 表示优先尝试下行节点，取值 +1 表示优先尝试上行节点。默认值为 0，表示自动选取子节点。

CROSSOVER

`OPTVIntParam.`**`CROSSOVER`**

内点法 crossover 策略。

默认值: 2

最小值: 0

最大值: 4

crossover 将内点解（通常为可行解，但一般不是基解）转化为基本可行解，它由以下三步组成:

1) 原始推进步 - 将原始变量推至边界，
2) 对偶推进步 - 将对偶变量推至边界，
3) 清理 - 应用单纯形方法清理任何原始或对偶不可行性。

该参数用于配置前两步的步骤及清理方法的选项。

取值	第一种推进方法	第二种推进方法	不可行性清理方法
0	禁用	禁用	禁用
1	对偶	原始	原始
2 (默认)	对偶	原始	对偶
3	原始	对偶	原始
4	原始	对偶	对偶

CUT_PASSES 割平面轮数

`OPTVIntParam.CUT_PASSES`

根节点 (MIP) 割平面方法最大轮数。

默认值: 2147483647

最小值: 0

最大值: 2147483647

该参数用以控制根节点割平面方法，使得轮数不超过设定值。

CUTS 割平面方法强度

`OPTVIntParam.CUTS`

MIP 割平面方法强度。

*** 默认值:** -1

最小值: -1

最大值: 3

该参数全局地控制割平面方法强度。默认值为-1，允许求解器进行自适应的调整；取值 0 则会关闭割平面方法；取值 1 会应用轻量级的割平面方法，取值 2 对应中等强度的割平面方法，而取值 3 则应用高强度的割平面方法。

MEM_LIMIT 内存限制

`OPTVIntParam.MEM_LIMIT`

求解器使用的内存限制, 单位 MB

默认值: 2147483647

最小值: 100

最大值: 2147483647

该参数用来限制求解器运行占用的 RAM 空间不超过指定阈值，若超出该阈值，求解器将终止运行并返回状态:py;data:*OPTV_MEM_LIMIT*.

METHOD 优化算法

OPTVIntParam.**METHOD**

LP 问题优化算法选择。

默认值: -1

最小值: -1

最大值: 4

该参数用于指定 LP 问题的优化方法。

取值	方法
-1	自动
0	原始单纯形方法
1	对偶单纯形方法
2	内点方法 (IPM)
3	行生成方法
4	两阶段原始单纯形方法

MIP_FOCUS 解策略

OPTVIntParam.**MIP_FOCUS**

MIP 高层级的解策略。

默认值: 0

最小值: 0

最大值: 3

该参数控制 MIP 求解器的优先度策略，默认为均衡式，具体信息如下：

取值	方法
0	均衡式
1	优先快速找到可行解
2	优先证明最优性
3	优先改善下界

OUTPUT_FLAG 日志打印控制

OPTVIntParam.**OUTPUT_FLAG**

控制打印日志的详细程度。

默认值: 0

最小值: 0

最大值: 1

如果开启该选项，所得日志（屏幕和日志文件）将被缩减和简化。

POOL_SOLUTIONS 解池容量

OPTVIntParam.**POOL_SOLUTIONS**

MIP 解池的最大存储（解）数量。

默认值: 10

最小值: 1

最大值: 1000000

该参数用于设置 MIP 解池存储的解的数量上限。

PRESOLVE_LEVEL 预处理控制

OPTVIntParam.**PRESOLVE_LEVEL**

控制预处理的程度。

默认值: -1

最小值: -1

最大值: 3

用户通过该参数控制预处理的程度。

取值	程度
-1	自动
0	关闭预处理
1	保守倾向
2	中等
3	进取倾向（耗时多，处理后的模型更紧凑）

SEED 随机数种子

OPTVIntParam.**SEED**

PRNG 初始随机数种子

默认值: 0

最小值: 0

最大值: 2147483647

该参数用来设定随机数生成器的初始值。

SOL_FORMAT 解文件风格控制

OPTVIntParam.**SOL_FORMAT**

解文件稀疏/稠密控制。

默认值: 0

最小值: 0

最大值: 1

通过调整该参数，用户可以控制解文件中是否包含 0 值。若设置该参数为 0，解文件将省去 0 值，为稀疏格式；否则，解文件将包含 0 值，为稠密格式。

SOLUTION_NUMBER 解索引

OPTVIntParam.**SOLUTION_NUMBER**

用于查询指定（MIP）解。

默认值: 0

最小值: 0

最大值: 1000000

用户可以通过该参数指定要查询的解，从而可获悉更多相关信息，如*OPTVDblAttr.XN* 或*OPTVDblAttr.POOL_OBJ_VAL*；取值为 0 时，上述查询信息等同于*OPTVDblAttr.X* 或*OPTVDblAttr.OBJ_VAL*.

备注: 本参数仅适用于 MIP 问题。

START_NODE_LIMIT 分支定界节点数上限

OPTVIntParam.**START_NODE_LIMIT**

限制在 MIP 问题求解启动期间访问的分支定界节点数。

默认值: -1

最小值: -3

最大值: 2147483647

该参数用于控制 MIP 启动期间访问的分支定界节点数。默认选项允许求解器自动确定 MIP 启动期间计算节点的最大数目；取值为-2 时，禁用 MIP 部分启动评估而采用全部启动；取值为-3 将完全禁用 MIP 启动计算。

备注: 本参数仅适用于 MIP 问题。

SUBMIP_NODES 启发式节点数上限

`OPTVIntParam.SUBMIP_NODES`

MIP 启发式策略模块访问的节点个数上限。

默认值 e: 500

最小值: 0

最大值: 2147483647

该参数用以限制 MIP 启发式策略访问的节点个数，访问节点越多，越有可能产生更高质量的解，但通常也耗时更久。

备注: 本参数仅适用于 MIP 问题。

THREADS 线程数

`OPTVIntParam.THREADS`

求解器运行期间使用最大线程数。

默认值: 1

最小值: 1

最大值: 虚拟机可用的最大线程总数

VAR_BRANCH 变量分支方法

`OPTVIntParam.VAR_BRANCH`

(MIP) 变量分支方法

默认值: -1

最小值: -1

最大值:: 2

该参数控制变量分支策略，具体信息如下：

取值	方法
0	自动
0	Reliability 分支方法
1	Strong branching 方法
2	Pseudo cost branching 方法

7.2.2 双精度型参数

class OPTVDblParam

表 2: OptVerse 双精度型参数

名称	描述
FEAS_TOL	原始问题约束可行性精度
HEURISTICS	MIP 启发式策略模块占时比例
INT_TOL	整数性精度
MARKOWITZ_TOL	用于限制选择矩阵分解主元时的数值误差
GAP	MIP 相对偏差
TIME_LIMIT	最大运行时间（秒）

FEAS_TOL 约束可行性精度

OPTVDblParam.**FEAS_TOL**

原始问题约束可行性精度。

默认值: 10^{-6}

最小值: 10^{-9}

最大值: 0.01

若该参数取较小的值，所得解的约束违背量则越小，但是迭代步数可能更多。

HEURISTICS 启发式模块

OPTVDblParam.**HEURISTICS**

MIP 启发式策略模块占总求解时间的比例。

默认值: 0.05

最小值: 0

最大值: 1

该参数控制 MIP 求解器启发式策略模块的占用时间，以占时比例为测度，默认值为 0.05, 即占总时耗的 5%。较高的该参数值可以使 MIP 求解器找到更多或更好的可行解，但是目标值的最佳边界进度也较慢。

INT_TOL 整数性精度

OPTVDblParam.**INT_TOL**

整数变量的整数性精度。

默认值: 10^{-6}

最小值: 10^{-9}

最大值: 0.01

若该参数取较小的值，所得解的整数性违背量则越小，但是求解性能（时耗）有可能被影响；取值较大一般不影响性能（时耗）。

MARKOWITZ_TOL 矩阵分解精度

`OPTVDblParam.`**`MARKOWITZ_TOL`**

适用于单纯形方法，用于限制选择矩阵分解主元时的数值误差。

默认值: 0.1

最小值: 10^{-4}

最大值: 0.999

取值较大将得到更好的数值稳定性，但可能影响求解性能（时耗）。

GAP 相对偏差

`OPTVDblParam.`**`GAP`**

MIP 相对偏差。

默认值: 0

最小值: 0

最大值: 10^{100}

若 f_P 和 f_D 分别为原始和对偶目标函数值，相对偏差值 g 的定义为 $g = \frac{|f_P - f_D|}{|f_P|}$. 在 MIP 求解过程中，一旦目标值的上下界相对偏差比 `gap` 小，求解器则会终止并返回状态*OPTV_OPTIMAL*.

TIME_LIMIT 求解时间上限

`OPTVDblParam.`**`TIME_LIMIT`**

模型求解时间上限（秒）。

默认值: 10^{20}

最小值: 0

最大值: 10^{20}

如果求解时间超出该数值，求解器则会终止并返回状态*OPTV_TIME_LIMIT*.

注意求解计时受制于单步迭代计时。对于耗时较久的优化问题，建议增加额外计时模块，如 `timeout`.

7.2.3 字符串型参数

`class OPTVStrParam`

表 3: OptVerse 字符串型参数

名称	描述
LOG_FILE	日志文件路径
PARAM_FILE	参数文件路径
REPLAY_FILE	SDK 命令回放文件路径
RESULT_FILE	结果文件路径

LOG_FILE 日志文件路径

OPTVStrParam.**LOG_FILE**

日志文件路径。

默认值: "./optv.log"

如果*OPTVIntParam.OUTPUT_FLAG* 被设置为 0，则本参数将被忽略。

PARAM_FILE 参数文件路径

OPTVStrParam.**PARAM_FILE**

参数文件路径。

默认值: ""

用于为求解器指定参数文件，默认值为空。

REPLAY_FILE SDK 命令回放文件

OPTVStrParam.**REPLAY_FILE**

SDK 命令回放文件路径。

默认值: ""

用于指定 SDK 命令回放文件存放的路径，若为空，则不创建命令回放文件。

备注: 该参数 **必须**在*OPTVModel* 对象创建前，通过*OPTVEnv* 对象设置。

RESULT_FILE 结果文件路径

OPTVStrParam.**RESULT_FILE**

结果文件路径。

默认值: "./<input_file>.sol"

备注: 通过 SDK 构建模型时，默认不进行输出结果文件，**必须**通过该参数指定生成解文件。

7.3 OptVerse 属性

属性对应着模型相关的各种数据，从属于模型组件对象 (*OPTVModel*, *OPTVVar*, *OPTVConstr*). 例如，任意给定变量或约束的下界可以通过*OPTVDblAttr.LB* 来获取。

备注: 有若干属性只能在*OPTVModel.Optimize()* 调用后才可以获取，如*OPTVDblAttr.X*, *OPTVDblAttr.XN*, *OPTVDblAttr.DUAL*, *OPTVDblAttr.OBJ_VAL*, *OPTVDblAttr.OBJ_BOUND*, 和*OPTVDblAttr.MIP_GAP*.

7.3.1 布尔属性

class OPTVBoolAttr

表 4: OptVerse 布尔型属性

名称	描述
IIS_CONSTR	指示该约束是否属于所得的 IIS

IIS_CONSTR 约束是否属于 IIS

OPTVBoolAttr.**IIS_CONSTR**

约束是否属于 IIS.

是否可修改: 否

表征指定约束是否属于计算所得的不可约不一致子系统 (IIS)；如果没有计算出 IIS，那么该属性不可获取。

7.3.2 整数型属性

class OPTVIntAttr

表 5: OptVerse 整数型属性

名称	描述
BASIS	变量或约束的基状态
BRANCH_PRIORITY	MIP 求解过程中变量的 branch(分支) 优先度
GENCONSTR_TYPE	一般约束类型
NUM_CONSTRS	约束总数
NUM_LIN_CONSTRS	线性约束总数
NUM_QUAD_CONSTRS	二次约束总数
NUM_INT_VARS	整数变量总数
NUM_VARS	变量总数
OBJ_SENSE	目标含义（最小化或最大化）
PROHIBITED	变量或约束是否被预处理屏蔽
SOL_COUNT	存储解的个数
SOS_TYPE	SOS 约束类型，1 或 2
STATUS	求解结果

BASIS 变量或约束的基状态

是否可修改: 是

变量或约束的基状态。

名称	取值	描述
OPTV_BASIC	0	基变量
OPTV_NONBASIC_LOWER	-1	非基，在下界
OPTV_NONBASIC_UPPER	-2	非基，在上界
OPTV_SUPERBASIC	-3	超基变量

BRANCH_PRIORITY 分支优先度

OPTVIntAttr.**BRANCH_PRIORITY**

是否可修改: 是

默认值: 0

该数值仅适用于 MIP 问题求解，用于引导 MIP 搜索过程的分支环节，以决定在哪些取值非整数的整数变量处进行分支，较大的取值表示较高的分支优先度。默认值为 0。如果存在优先度相同的情况，则采用原始的分支决策。

GENCONSTR_TYPE 一般约束类型

OPTVIntAttr.**GENCONSTR_TYPE**

一般约束类型。

是否可修改: 否

表 6: OptVerse 一般约束类型

名称	取值	描述
OPTV_GENCONSTR_AND	0	只有所有变量非零时，结果才非零
OPTV_GENCONSTR_OR	1	只要有一个变量非零，结果就非零
OPTV_GENCONSTR_INDICA	2	返回值为布尔型，当且仅当相应的逻辑条件满足，取值为 1，否则取值为 0

NUM_CONSTRS 约束总数

OPTVIntAttr.**NUM_CONSTRS**

是否可修改: 否

约束总数。

NUM_LIN_CONSTRS 线性约束总数

OPTVIntAttr.**NUM_LIN_CONSTRS**

是否可修改: 否

线性约束总数。

NUM_QUAD_CONSTRS 二次约束总数

OPTVIntAttr.**NUM_QUAD_CONSTRS**

是否可修改: 否

二次约束总数。

NUM_INT_VARS 整数变量总数

OPTVIntAttr.**NUM_INT_VARS**

是否可修改: 否

整数变量总数，包括 0/1 变量和普通整数变量。

NUM_VARS 变量总数

OPTVIntAttr.**NUM_VARS**

是否可修改: 否

变量总数。

OBJ_SENSE 目标含义

OPTVIntAttr.**OBJ_SENSE**

是否可修改: 是

优化问题目标含义，取值为-1 代表最大化问题，取值为 1 代表最小化问题。参考*OPTVSense*.

PROHIBITED 预处理屏蔽

OPTVIntAttr.**PROHIBITED**

是否可修改: 是

用于指示变量或约束是否被 LP 预处理屏蔽，取值包括 0（未屏蔽）和 1（屏蔽）。该参数不适用于 MIP 问题。

SOL_COUNT 存储解的个数

OPTVIntAttr.**SOL_COUNT**

是否可修改: 否

存储解的个数：对于 LP 问题，若获得最优解则取值为 1，否则取 0；对于 MIP 问题，该属性对应找到的解（包含最优和次最优）的个数。

SOS_TYPE SOS 类型

OPTVIntAttr.**SOS_TYPE**

是否可修改: 否

SOS 类型，可取值为 1 或 2.

STATUS 求解结果

OPTVIntAttr.**STATUS**

是否可修改: 否

模型求解状态。

名称	取值	描述
OPTV_OPTIMAL	1	模型求解成功（最优解）
OPTV_INFEASIBLE	2	模型被证明不可行
OPTV_INF_OR_UNBD	3	模型被证明不可行或无界
OPTV_UNBOUNDED	4	模型被证明无界
OPTV_TIME_LIMIT	5	求解时间超出给定时限*OPTVDblParam.TIME_LIMIT*.
OPTV_MEM_LIMIT	6	求解内存占用超过给定阈值*OPTVIntParam.MEM_LIMIT*.
OPTV_INTERRUPTED	7	用户自行终止（如通过 Ctrl+C）.
OPTV_UNKNOWN	0	未知状态。

7.3.3 字符型属性

class OPTVCharAttr

表 7: OptVerse 字符串型属性

名称	描述
VAR_TYPE	变量类型

VAR_TYPE 变量类型

OPTVCharAttr.**VAR_TYPE**

变量类型。

是否可修改: 是

表 8: OptVerse 变量类型

名称	取值	描述
OPTV_CONTINUOUS	'C'	连续型
OPTV_INTEGER	'I'	整数型
OPTV_BINARY	'B'	0/1 型

连续变量可取给定界限内的任意浮点数值；整数变量取值限定在给定界限内的所有整数；0/1 变量只可取 0 或者 1。

7.3.4 双精度型属性

class OPTVDblAttr

表 9: OptVerse 双精度型属性

名称	描述
DUAL	变量或约束的对偶解值。
LB	变量或约束的下界值。
MIP_GAP	MIP 相对偏差。
OBJ	变量的目标值系数。
OBJ_BOUND	目标值的最佳界限。
OBJ_OFFSET	目标函数的偏移量。
OBJ_VAL	原始模型目标值。
POOL_OBJ_VAL	第 N 个解的原始模型目标值。
RUN_TIME	运行时间。
START	MIP 热启动值。
UB	变量或约束的上界值。
X	变量或约束的原始解值。
XN	第 N 个解的变量或约束的原始解值。

DUAL 对偶解值

OPTVDblAttr.**DUAL**

是否可修改: 否

变量或约束的对偶解值。

LB 下界值

`OPTVDblAttr.`**`LB`**

是否可修改: 是

变量或约束的下界值。

备注: 对于 0/1 变量，该属性只允许取值为 `0` 或 `1`. 若赋值超出上述允许值 10^{-6}, 该操作会被禁止且相应的警告信息会被抛出。

MIP_GAP MIP 相对偏差

`OPTVDblAttr.`**`MIP_GAP`**

是否可修改: 否

MIP 相对偏差，仅适用于 MIP 求解范畴，具体定义可参考*相对偏差*。

OBJ 目标值系数

`OPTVDblAttr.`**`OBJ`**

是否可修改: 是

变量的目标值系数。

OBJ_BOUND 目标值的最佳界限

`OPTVDblAttr.`**`OBJ_BOUND`**

是否可修改: 否

目标值的最佳界限，仅适用于 MIP 求解范畴。

OBJ_OFFSET 目标函数偏移量

`OPTVDblAttr.`**`OBJ_OFFSET`**

是否可修改: 是

目标函数中的常数项。

OBJ_VAL 目标值

`OPTVDblAttr.`**`OBJ_VAL`**

是否可修改: 否

模型的原始解对应的目标值。

POOL_OBJ_VAL MIP 解的原始模型目标值

OPTVDblAttr.**POOL_OBJ_VAL**

是否可修改: 否

该属性仅适用于 MIP 求解范畴，当通过*OPTVIntParam.SOLUTION_NUMBER* 指定第 N 个 MIP 解时，*OPTVDblAttr.OBJ_VAL* 返回该解对应的原始模型目标值。

RUN_TIME 运行时间

OPTVDblAttr.**RUN_TIME**

是否可修改: 否

最近一次求解任务的执行时间。

START MIP 热启动值

OPTVDblAttr.**START**

是否可修改: 是

MIP 热启动值，该属性仅适用于 MIP 求解范畴，默认为*OPTV_UNDF*.

UB 上界值

OPTVDblAttr.**UB**

是否可修改: 是

变量或约束的上界值。

备注: 对于 0/1 变量，该属性只允许取值为 `0` 或 `1`．若赋值超出上述允许值 10^{-6}, 该操作会被禁止且相应的警告信息会被抛出。

X 原始解值

OPTVDblAttr.**X**

是否可修改: 否

变量或约束的原始解值。

XN MIP 解的原始解值

OPTVDblAttr.**XN**

是否可修改: 否

该属性仅适用于 MIP 求解范畴，当通过*OPTVIntParam.SOLUTION_NUMBER* 指定第 N 个 MIP 解时，*OPTVDblAttr.XN* 返回该解的原始解值。

7.3.5 长整型属性

class OPTVLongAttr

表 10: OptVerse 长整型属性

名称	描述
FINGERPRINT	模型指纹

FINGERPRINT （模型）指纹

OPTVLongAttr.**FINGERPRINT**

模型指纹

是否可修改: 否

该参数对应由模型内部数据和属性决定的 hash 值，不同的模型（包括变量或约束添加的顺序）应具有不同的模型指纹。

7.3.6 字符串型属性

class OPTVStrAttr

表 11: OptVerse 字符串型属性

名称	描述
NAME	变量或约束的名称。

NAME 变量/约束名称

OPTVStrAttr.**NAME**

变量或约束的名称，通常在建模环节根据问题定义设定。

是否可修改: 是

7.4 OptVerse 目标含义

class OPTVSense

目标含义，默认值为 1，表征最小化问题。

MINIMIZE

最小化问题

类型
int

取值
1

MAXIMIZE

最大化问题

类型
int

取值
-1

7.5 OptVerse 环境

class OPTVEnv

OPTV 求解器环境类，用于管理 OPTV 求解器的参数配置。此外，许可文件需要通过环境变量 `OPTV_LICENSE_FILE` 来配置。

7.5.1 OPTVEnv() 环境类构造函数

`OPTVEnv.`**OPTVEnv**`()`

返回
环境变量，参数为默认。

返回类型
OPTVEnv

备注： 默认配置下日志文件被设定为 `optv.log`.

`OPTVEnv.`**OPTVEnv**`(`*logFileName*`)`

参数
logFileName (*str*) – 日志文件路径。

返回
环境变量，参数为默认，日志文件由参数指定。

返回类型
OPTVEnv

OPTVEnv.**OPTVEnv**(*xenv*)

参数
xenv(OPTVEnv)–需要被复制的环境。

返回
环境对象 xenv 的副本。

返回类型
OPTVEnv

7.5.2 Get() 参数查询

获取求解器的参数值，可用参数请参考*OptVerse 参数*。

OPTVEnv.**Get**(*param*)

参数
param(OPTVIntParam/OPTVDblParam/OPTVStrParam)–需要查询的参数，包括整数型、双精度型、字符串型

返回
参数取值。

返回类型
int | float | str

抛出
OPTVErrCode.UNKNOWN_PARAMETER–未知参数。

示例

```
env = OPTVEnv()
env.Set(OPTVDblParam.TIME_LIMIT, 7200.0)
model.Update()
print(env.Get(OPTVDblParam.TIME_LIMIT)) # 7200.0
```

7.5.3 Set() 参数设置

通过环境对象设置求解器参数，相应修改会作为默认被应用到后续创建的所有模型。模型建立后，只有调用*OPTVModel.Update()*才可以激活新的参数修改。具体细节可参考*OptVerse 参数*。

OPTVEnv.**Set**(*param*, *value*)

参数

- **param**(OPTVIntParam/OPTVDblParam/OPTVStrParam)–需要设置的参数，包括整数型、双精度型、字符串型
- **value**(*int*/*float*/*str*)–新参数值。

返回
无。

返回类型
None

抛出
OPTVErrCode.UNKNOWN_PARAMETER –未知参数。

示例

```
env = OPTVEnv()
env.Set(OPTVDblParam.TIME_LIMIT, 7200.0)
env.Set(OPTVIntParam.THREADS, 2)
env.Set(OPTVStrParam.LOG_FILE, 'optv_new.log')

model.Update()
print(env.Get(OPTVDblParam.TIME_LIMIT)) # 7200.0
print(env.Get(OPTVIntParam.THREADS))    # 2
print(env.Get(OPTVStrParam.LOG_FILE))   # optv_new.log
```

7.5.4 ResetParams() 重置参数

将所有的参数重置为默认值。

OPTVEnv.**ResetParams**()

返回
无。

返回类型
None

示例

```
env = OPTVEnv()
env.Set(OPTVDblParam.TIME_LIMIT, 7200.0)
env.Set(OPTVIntParam.THREADS, 1)
env.Set(OPTVStrParam.LOG_FILE, 'optv_new.log')

# 重置设置的参数为默认值
env.ResetParams()

print(env.Get(OPTVDblParam.TIME_LIMIT)) # 1e+20
print(env.Get(OPTVIntParam.THREADS))    # 1
print(env.Get(OPTVStrParam.LOG_FILE))   # ""
```

7.6 OptVerse 变量

class OPTVVar

OptVerse 变量对象，用于从模型访问变量。注意变量不可由其构造函数创建，而需要通过模型的*OPTVModel.AddVar()* 或*OPTVModel.AddVars()* 接口添加变量到模型。

7.6.1 Get() 查询变量属性

查询变量属性，关于可用属性请参考*OptVerse 属性*。

OPTVVar.**Get**(*attr*)

参数
attr (OPTVIntAttr / OPTVDblAttr / OPTVCharAttr / OPTVStrAttr) –需要查询的变量属性，包括整数型、双精度型、字符和字符串型

返回
属性值。

返回类型
int | float | str

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知变量属性。

示例

```
x = model.AddVar(0, 1, 0, OPTV_BINARY, "x")
model.Update()
# 查询变量名称
result = x.Get(OPTVStrAttr.NAME)
print(result) # x
```

7.6.2 Set() 设置变量属性

设置变量属性，只有调用*OPTVModel.Update()* 后，新设置才可生效，更多细节请参考*OptVerse 属性*。

OPTVVar.**Set**(*attr*, *value*)

参数

- **attr** (OPTVIntAttr / OPTVCharAttr / OPTVDblAttr / OPTVStrAttr) –需要设置的变量属性，包括整数型、双精度型、字符和字符串型
- **value** (*int* / *float* / *str*) –新属性值。

返回
无。

返回类型
None

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知变量属性。

示例

```
x = model.AddVar(0, OPTV_INF, 0, OPTV_CONTINUOUS, "x")
x.Set(OPTVDblAttr.Ub, 100)
```

7.6.3 Index() 变量索引

获得变量在模型中的索引值。

OPTVVar.**Index**()

返回
变量索引/下标。

返回类型
int

返回值

- **-1** –该变量在模型中不存在。
- **-2** –该变量已从模型中删除。
- **>=0** –该变量在模型中的索引/下标。

示例

```
x = model.AddVar(0, 1, 0, OPTV_BINARY, "x")
y = model.AddVar(0, 1, 0, OPTV_BINARY, "y")
x_ind = x.Index()
y_ind = y.Index()
print(x_ind, y_ind) # 0, 1
```

7.6.4 SameAs() 变量对比

查询两变量是否相同。

OPTVVar.**SameAs**(*v2*)

参数
v2 –需要与当前变量对比的变量。

返回
变量是否相同。

返回类型
bool

返回值

- **true** –变量相同。
- **false** –变量不同。

示例

```
x = model.AddVar(0, OPTV_INF, 0, OPTV_CONTINUOUS, "x")
y = model.AddVar(0, OPTV_INF, 0, OPTV_CONTINUOUS, "y")
z = x
print(x.SameAs(y)) # False
print(x.SameAs(x)) # True
```

7.7 OptVerse 矩阵变量

class OPTVMVar

OptVerse 矩阵变量对象。OPTVMVar 类是一个构建多维变量，并且支持 NumPy 多维数组运算的类。我们建议通过模型的*OPTVModel.AddMVar()* 方法添加多维变量，而不是通过内置类方法转换生成。

7.7.1 Copy() 拷贝

深度复制一个 OPTVMVar 对象

OPTVMVar.**Copy**()

返回
原 OPTVMVar 对象的拷贝。

返回类型
OPTVMVar

示例

```
matrix_vars = model.AddMVar((2, 2), name="x")
copy_matrix_vars = matrix_vars.Copy()
```

7.7.2 Diagonal() 获取指定对角线上的元素

创建一个包含原 OPTVMVar 对象指定对角线上的变量的新 OPTVMVar 对象。

OPTVMVar.**Diagonal**(*offset=0*, *axis1=0*, *axis2=1*)

参数

- **offset** (*int*) –可选参数，表示对角线的偏移量，默认值为 0。若是大于 0，表示对角线向上的偏移量；若是小于 0，表示对角线向下的偏移量
- **axis1** (*int*) –可选参数，作为二维子 OPTVMVar 的第一轴的轴，对角线从这里开始。默认值为 0。
- **axis2** (*int*) –可选参数，作为二维子 OPTVMVar 的第二轴的轴，对角线从这里开始。默认值为 1。

返回
一个包含原 OPTVMVar 对象指定对角线上的变量的新 OPTVMVar 对象。

返回类型
OPTVMVar

示例

```
matrix_vars = model.AddMVar((8, 8), name="x")
diagonal_main = matrix_vars.Diagonal()    # 主对角线上的变量
diagonal_sub1 = matrix_vars.Diagonal(1)   #␣
↪主对角线向上偏移1个单位的对角线上的变量
diagonal_sub1 = matrix_vars.Diagonal(-2)  #␣
↪主对角线向下偏移2个单位的对角线上的变量
```

7.7.3 FromList() 将列表转化为 OPTVMVar 对象

将元素为 OPTVVar 的列表转化为 OPTVMVar 对象。该方法为类生成方法，无需 OPTVMVar 对象。

OPTVMVar.**FromList**(*varlist*)

参数
varlist (*list*) –元素为 OPTVVar 的列表。

返回
新的 OPTVMVar 对象，其维度取决于参数 varlist 的维度。

返回类型
OPTVMVar

示例

```
x0 = model.AddVar(0, 1, 0, OPTV_BINARY, name="x0")
x1 = model.AddVar(0, 1, 0, OPTV_BINARY, name="x1")
x2 = model.AddVar(0, 1, 0, OPTV_BINARY, name="x2")
x3 = model.AddVar(0, 1, 0, OPTV_BINARY, name="x3")
matrix_1d = OPTVMMVar.FromList([x0, x1, x2, x3])      # 1维矩阵
matrix_2d = OPTVMMVar.FromList([[x0, x1], [x2, x3]]) # 2维矩阵
```

7.7.4 FromVar() 将 OPTVVar 对象转化为 OPTVMVar 对象

将 OPTVVar 对象转化为一个 0 维的 OPTVMVar 对象。该方法为类生成方法，无需 OPTVMVar 对象。

OPTVMVar.**FromVar**(*var*)

参数
var (OPTVVar) –OPTVVar 对象。

返回
新的 0 维 OPTVMVar 对象。

返回类型
OPTVMVar

示例

```
x = model.AddVar(0, 1, 0, OPTV_BINARY, name="x")
matrix_1d = OPTVMMVar.FromVar(x)
```

7.7.5 Get() 查询矩阵变量属性

查询 OPTVMVar 对象中每个变量的属性，关于可用属性请参考*OptVerse 属性*。

OPTVMVar.**Get**(*attr*)

参数
attr (OPTVIntAttr / OPTVDblAttr / OPTVCharAttr / OPTVStrAttr) –需要查询的变量属性，包括整数型、双精度型、字符和字符串型。

返回
一个 NumPy 数组，维度与 OPTVMVar 的维度相同，其元素为对应变量相关属性的取值。

返回类型
numpy.ndarray

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知变量属性。

示例

```
matrix_vars = model.AddMVar((2, 2), name="x")
print(matrix_vars.Get(OPTVCharAttr.VAR_TYPE))
```

7.7.6 Set() 设置矩阵变量属性

设置 OPTVMVar 对象中所有变量的属性，只有调用*OPTVModel.Update()* 后，新设置才可生效，更多细节请参考*OptVerse 属性*。

OPTVMVar.**Set**(*attr*, *newvalue*)

参数

- **attr** (OPTVIntAttr / OPTVCharAttr / OPTVDblAttr / OPTVStrAttr) –需要设置的变量属性，包括整数型、双精度型、字符和字符串型。
- **newvalue** (*int* / *float* / *str*) –需要设置的属性的值，其形状必须与 OPTVMVar 对象的形状相同。也可以是一个标量，它将自动扩展为正确的形状。

返回
无。

返回类型
None

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知变量属性。

示例

```
matrix_vars = model.AddMVar((2, 2), name="x")
matrix_vars.Set(OPTVIntAttr.BASIS, 1)
```

7.7.7 Item() 获取 OPTVMVar 对象中的变量

若是 OPTVMVar 中只有一个 OPTVVar 对象，返回该 OPTVVar 的拷贝。若是在 OPTVMVar 有多个变量时调用该方法，会产生 ValueError。

OPTVMVar.**Item**()

返回
OPTVMVar 的唯一元素 OPTVVar 的拷贝。

返回类型
OPTVVar

抛出
ValueError –OPTVMVar 中有多个元素。

示例

```
matrix_vars = model.AddMVar((2, 2), name="x")
matrix_var = matrix_vars[0, 1] #␣
↪一个0维的OPTVMVar对象，只有一个OPTVVar
var = matrix_vars[0, 1].Item() # 对应位置的OPTVVar对象
```

7.7.8 Reshape() 转换形状

返回一个新的 OPTVMVar 对象，其元素与元 OPTVMVar 相同，形状与参数 shape 相同。

OPTVMVar.**Reshape**(*shape*)

参数
shape –一个整数或者元组，表示新 OPTVMVar 对象的形状。

返回
OPTVMVar 对象，其形状与给定形状相同。

返回类型
OPTVMVar

示例

```
matrix_vars = model.AddMVar((2, 2), name="x")
new_matrix_vars = matrix_vars.Reshape(4) # 维度变为1维
```

7.7.9 Tolist() 转换为列表

将 OPTVMVar 对象转换为包含其所有变量的列表。

OPTVMVar.**ToList**()

返回
包含 OPTVMVar 所有变量的列表。

返回类型
list[*OPTVVar*]

示例

```
matrix_vars = model.AddMVar(3, name="x")
vars_list = matrix_vars.ToList()
print(vars_list) # [<optv.Var x_0> <optv.Var x_1> <optv.Var x_2>]
```

7.7.10 Transpose() 转置矩阵

生成一个新的 OPTVMVar 对象，它是原 OPTVMVar 对象的转置。

OPTVMVar.**Transpose**()

返回
新的 OPTVMVar 对象，为原 OPTVMVar 对象的转置。

返回类型
OPTVMVar

示例

```
matrix_vars = model.AddMVar((3, 1), name="x")
matrix_vars_T = matrix_vars.Transpose()
print(matrix_vars_T.Shape) # (1, 3)
matrix_vars = model.AddMVar((2, 3), name="x")
matrix_vars_T = matrix_vars.Transpose()
print(matrix_vars_T.Shape) # (3, 2)
```

7.7.11 Ndim 维度

获取 OPTVMVar 对象的维度。

OPTVMVar.**Ndim**

返回
OPTVMVar 对象的维度。

返回类型
int

示例

```
matrix_vars = model.AddMVar((2, 2), name="x")
print(matrix_vars.Ndim) # "2"
matrix_vars = model.AddMVar((2), name="x")
print(matrix_vars.Ndim) # "1"
```

7.7.12 Shape 形状

获取 OPTVMVar 对象的形状。

OPTVMVar.**Shape**

返回
OPTVMVar 对象的形状。

返回类型
tuple[int, ···]

示例

```
matrix_vars = model.AddMVar((2, 2), name="x")
print(matrix_vars.Shape) # (2, 2)
matrix_vars = model.AddMVar(3, name="x")
print(matrix_vars.Shape) # (3,)
```

7.7.13 Size 元素个数

获取 OPTVMVar 对象中的元素个数。

`OPTVMVar.Size`

返回
OPTVMVar 对象中的元素个数。

返回类型
int

示例

```
matrix_vars = model.AddMVar((2, 2), name="x")
print(matrix_vars.Size) # 4
matrix_vars = model.AddMVar(3, name="x")
print(matrix_vars.Size) # 3
```

7.7.14 T 转置矩阵

类似于类方法 Transpose()。

`OPTVMVar.T`

返回
新的 OPTVMVar 对象，为原 OPTVMVar 对象的转置。

返回类型
OPTVMVar

示例

```
matrix_vars = model.AddMVar((3, 1), name="x")
matrix_vars_T = matrix_vars.T
print(matrix_vars_T.Shape) # (1, 3)
matrix_vars = model.AddMVar((2, 3), name="x")
matrix_vars_T = matrix_vars.T
print(matrix_vars_T.Shape) # (3, 2)
```

7.8 OptVerse 列

class OPTVColumn

OptVerse 列对象，用于以列的形式向模型添加变量，用户可以使用该功能向模型同时新增变量并添加其至已有约束。

在下面的例子中，我们构造一个初始模型并展示如何通过列对象增加变量，本示例展示的模型为*Quick Start* 章节的*milp1* .

```
from optvpy import *

env = OPTVEnv()
model = OPTVModel(env)
```

(续下页)

(接上页)

```
x = model.AddVar(0, 1, -1, OPTV_BINARY, "x")
y = model.AddVar(0, OPTV_INF, -14, OPTV_INTEGER, "y")

c0 = model.AddConstr(2 * x + y, -OPTV_INF, 3, "c0")
c1 = model.AddConstr(3 * y, -OPTV_INF, 6, "c1")
```

具体地，构建的模型形式如下：

$$A = \begin{bmatrix} 2 & 1 \\ 0 & 3 \end{bmatrix}$$

```
col = OPTVColumn()

col.addTerm(1, c0)
col.addTerm(1, c1)

z = model.AddVar(0, 3, -6, OPTV_CONTINUOUS, col, "z");
```

通过将 `col` 添加到模型，相应的约束矩阵变为：

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 0 & 3 & 1 \end{bmatrix}$$

7.8.1 AddTerm() 添加单项

向列对象（尾部）添加单项。

`OPTVColumn.`**`AddTerm`**(*coef*, *constr*)

该函数向列对象添加单项。

参数

- **coef** (*float*) –系数值。
- **constr** (OPTVConstr) –约束对象。

返回

无。

返回类型

None

示例

```
col = OPTVColumn()

col.AddTerm(1.0, x)
col.AddTerm(3.0, y)
col.AddTerm(2.0, z)
```

7.8.2 AddTerms() 增加多项

向列对象（尾部）添加多项。

OPTVColumn.**AddTerms**(*coefs*, *constrs*)

该功能向列对象添加多项。

参数

- **coefs**(*list[float]*) –系数值列表。
- **constrs**(*list[OPTVConstr]*) –约束列表。

返回

无。

返回类型

None

抛出

OPTVErrCode.INVALID_ARGUMENT –constrs 或 coefs 为空。

示例

```
constrs = self.model.GetConstrs()

coefs = [1.0, 3.0, 2.0]

col = OPTVColumn()
col.AddTerms(coefs, constrs)
```

7.8.3 Clear() 清空

清空与该列相关内部容器，使*OPTVColumn.Size()* 返回 0.

OPTVColumn.**Clear**()

返回

无。

返回类型

None

示例

```
model.Update()

constrs = model.GetConstrs()
coefs = [1.0, 2.0, 3.0]

col = OPTVColumn()
col.AddTerms(coefs, constrs)
print(col.Size()) # 3

col.Clear()
print(col.Size()) # 0
```

7.8.4 GetCoef() 获取系数

据指定索引值获得列对象中对应的约束系数。

OPTVColumn.**GetCoef**(*i*)

参数
i (*int*) –指定的索引值，对应列中的约束。

返回
列对象中对应该索引, i, 的约束系数。

返回类型
float

抛出
OPTVErrCode.INDEX_OUT_OF_RANGE –若 $i \notin [0, \texttt{this->Size()}]$.

示例

```
col = OPTVColumn()

col.AddTerm(1.0, x)
col.AddTerm(3.0, y)
col.AddTerm(2.0, z)

print(col.GetCoef(0)) # 1.0
```

7.8.5 GetConstr() 获取单个约束

根据指定索引值获得列对象中对应的约束。

OPTVColumn.**GetConstr**(*i*)

参数
i (*int*) –指定的索引值，对应列中的约束。

返回
列对象中对应该索引, i, 的约束对象。

返回类型
OPTVConstr

抛出
OPTVErrCode.INDEX_OUT_OF_RANGE –若 $i \notin [0, \texttt{this->Size()}]$.

示例

```
constrs = model.GetConstrs()

coefs = [1.0, 3.0, 2.0]

col = OPTVColumn()
col.AddTerms(coefs, constrs)

col.GetConstr(0)
```

7.8.6 Remove() 删除项

从列对象中删除单/多项。

OPTVColumn.**Remove**(*i*)

根据指定索引值从该列对象中删除项。

参数

i (*int*) –指定的索引值。

返回

无。

返回类型

None

抛出

OPTVErrCode.INDEX_OUT_OF_RANGE –若 i ∉ [0, this->Size()].

示例

```
model.Update()

constrs = model.GetConstrs()

coefs = [1.0, 3.0, 2.0]

col = OPTVColumn()
col.AddTerms(coefs, constrs)
print(col.Size()) # 3

col.Remove(1)
print(col.Size()) # 2
```

OPTVColumn.**Remove**(*constr*)

从列对象中删除指定约束的所有相关项。

参数

constr (OPTVConstr) –指定的约束。

返回

是否删除任何项。

返回

无。

返回类型

None

抛出

OPTVErrCode.NOT_IN_MODEL –指定的约束， constr, 在模型中不存在。

示例

```
model.Update()

constrs = model.GetConstrs()

coefs = [1.0, 3.0, 2.0]
```

(续下页)

(接上页)

```
col = OPTVColumn()
col.AddTerms(coefs, constrs)
print(col.Size()) # 3

col.Remove(constrs[1])
print(col.Size()) # 2
```

7.8.7 Size() 规模

获得该列的规模。

OPTVColumn.**Size**()

该数值统计对应列中的元素个数，无论是否有重复项，例如，若某约束被多次添加，本数值依然相应增加。

返回

该列的规模。

返回类型

int

示例

```
model.Update()

constrs = model.GetConstrs()

coefs = [1.0, 3.0, 2.0]

col = OPTVColumn()
col.AddTerms(coefs, constrs)
print(col.Size()) # 3
```

7.9 OptVerse 约束

class OPTVConstr

OptVerse 约束对象，用于从模型访问约束。注意约束不可由其构造函数创建，而需要通过模型的*OPTVModel.AddConstr()* 或*OPTVModel.AddConstrs()* 接口添加约束至模型。此外，二次约束的相应接口为*OPTVModel.AddQConstr()*.

7.9.1 Get() 查询约束属性

查询约束属性，关于可用属性请参考*OptVerse 属性*。

OPTVConstr.**Get**(*attr*)

参数
attr (OPTVIntAttr / OPTVDblAttr / OPTVStrAttr) –需要查询的约束属性，包括整数型、双精度型和字符串型。

返回
属性值。

返回类型
int | float | str

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

示例

```
c0 = model.AddConstr(x+y+z, 0, 3, "c0")
model.Update()
print(c0.Get(OPTVIntAttr.PROHIBITED)) # 0
```

7.9.2 Set() 设置约束属性

设置约束属性，只有调用*OPTVModel.Update()* 后，新设置才可生效，更多细节请参考*OptVerse 属性*。

OPTVConstr.**Set**(*attr*, *value*)

参数

- **attr** (OPTVIntAttr / OPTVDblAttr / OPTVStrAttr) –需要设置的约束属性，包括整数型、双精度型和字符串型
- **value** (*int* / *float* / *str*) –新属性值。

返回
无。

返回类型
None

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

示例

```
c0 = model.AddConstr(x+y+z, 0, 3, "c0")

c0.Set(OPTVDblAttr.UB, 999)
model.Update()

print(c0.Get(OPTVDblAttr.UB)) # 999
```

7.9.3 Index() 约束索引

获得约束在模型中的索引值。

OPTVConstr.**Index**()

返回
约束索引/下标。

返回类型
int

返回值

- **-1** –该约束在模型中不存在。
- **-2** –该约束已从模型中删除。
- **>=0** –该约束在模型中的索引/下标。

示例

```
c0 = model.AddConstr(x+y+z, 0, 3, "c0")
c1 = model.AddConstr(2*x+z, 2, 6, "c1")

print(c0.Index()) # 0
print(c1.Index()) # 1
```

7.9.4 SameAs() 约束对比

查询两约束是否相同。

OPTVConstr.**SameAs**(*c2*)

参数
c2 (OPTVConstr) –需要与当前约束对比的约束。

返回
约束是否相同。

返回类型
bool

返回值

- **true** –约束相同。
- **false** –约束不同。

示例

```
c0 = model.AddConstr(x+y+z, 0, 3, "c0")
c1 = model.AddConstr(2*x+z, 2, 6, "c1")
copy_c0 = c0

print(c0.SameAs(copy_c0)) # True
print(c1.SameAs(c1))      # False
```

7.10 OPTVMConstr 线性约束矩阵

class OPTVMConstr

线性约束矩阵对象。线性约束矩阵对象在数据结构上等效于存储线性约束的 numpy.ndarray 数组。推荐用户使用*OPTVModel.AddMConstr()* 添加线性约束矩阵对象。

7.10.1 FromList() 从约束列表创建 OPTVMConstr 对象

将元素为*OPTVConstr* 的列表转化为 OPTVMConstr 对象。该方法为类生成方法，无需 OPTVMConstr 对象。

OPTVMConstr.**FromList**(*constr_list*)

参数
constr_list (*list*) –list[OPTVConstr]

返回
新的 OPTVMConstr 对象，其维度取决于参数 constr_list 的维度。

返回类型
OPTVMConstr

示例

```
constrs_list = model.GetConstrs()
matrix_constrs = OPTVMConstr.FromList(constrs_list)
```

7.10.2 Get() 查询线性约束矩阵属性

查询 OPTVMConstr 对象中每条约束的属性，关于可用属性请参考*OptVerse 属性*。

OPTVMConstr.**Get**(*attr*)

参数
attr (OPTVIntAttr / OPTVDblAttr / OPTVCharAttr / OPTVStrAttr) –需要查询的约束属性，包括整数型、双精度型、字符和字符串型。

返回
一个 NumPy 数组，维度与 OPTVMConstr 的维度相同，其元素为对应约束相关属性的取值。

返回类型
numpy.ndarray

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

示例

```
x = model.AddMVar(3, ub=1, vtype=OPTV_BINARY, name=['x1', 'y1', 'z1'])
constrs = model.AddMConstr(np.array([[1, 2, 3], [1, 1, 0]]), x, [-
↪OPTV_INF, 1], [4, OPTV_INF], name='c')
result = constrs.Get(OPTVDblAttr.UB)
```

7.10.3 Set() 设置线性约束矩阵属性

设置 OPTVMConstr 对象中所有约束的属性，只有调用*OPTVModel.Update()* 后，新设置才可生效，更多细节请参考*OptVerse 属性*。

OPTVMConstr.**Set**(*attrname*, *newvalue*)

参数

- **attrname** (OPTVIntAttr / OPTVCharAttr / OPTVDblAttr / OPTVStrAttr) – 需要设置的约束属性，包括整数型、双精度型、字符和字符串型。
- **newvalue** (*int* / *float* / *str*) – 需要设置的属性的值，其形状必须与 OPTVMConstr 对象的形状相同。也可以是一个标量，它将自动扩展为正确的形状。

返回
无。

返回类型
None

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE – 未知约束属性。

示例

```
x = model.AddMVar(3, ub=1, vtype=OPTV_BINARY, name=['x1', 'y1', 'z1'])
constrs =  model.AddMConstr(np.array([[1, 2, 3], [1, 1, 0]]), x, [-
↪OPTV_INF, 1], [4, OPTV_INF], name='c')
constrs.Set(OPTVDblAttr.LB, -10)
```

7.10.4 Tolist() 转换为列表

将 OPTVMConstr 对象转换为包含其所有约束的列表。

OPTVMConstr.**ToList**()

返回
包含 OPTVMConstr 所有约束的列表。

返回类型
list[*OPTVConstr*]

示例

```
x = model.AddMVar(3, ub=1, vtype=OPTV_BINARY, name=['x1', 'y1', 'z1'])
constrs =  model.AddMConstr(np.array([[1, 2, 3], [1, 1, 0]]), x, [-
↪OPTV_INF, 1], [4, OPTV_INF], name='c')
constrs_list = constrs.ToList()
```

7.11 OptVerse 二次约束

class OPTVQConstr

二次约束类用于访问模型中的二次约束。注意二次约束也不可以由其构造函数直接创建，而是通过接口*OPTVModel.AddQConstr()* 将其添加到模型。

警告： 当求解的模型 **包含**二次约束时，QCQP 求解器要求模型的二次目标函数或二次约束的矩阵都是正半定的，否则会抛出*OPTVErrCode.MATRIX_NOT_PSD* 异常。

7.11.1 Get() 查询二次约束属性

查询二次约束属性，更多细节请参考*OptVerse 属性*。

OPTVQConstr.**Get**(*attr*)

参数
attr (OPTVIntAttr / OPTVDblAttr / OPTVStrAttr) –需要查询的二次约束属性，包括整数型、双精度型、字符和字符串型

返回
属性值。

返回类型
int | float | str

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

示例

```
c0 = model.AddQConstr(2*x*x + y*y + z, -OPTV_INF, 3, "c0")
model.Update()
result = c0.Get(OPTVIntAttr.PROHIBITED)
```

7.11.2 Set() 设置二次约束属性

设置二次约束属性，只有调用*OPTVModel.Update()* 后，新设置才可生效，更多细节请参考*OptVerse 属性*。

OPTVQConstr.**Set**(*attr*, *value*)

参数

- **attr** (OPTVIntAttr / OPTVDblAttr / OPTVStrAttr) –需要设置的二次约束属性，包括整数型、双精度型、字符和字符串型
- **value** (*int* / *float* / *str*) –新属性值。

返回
无。

返回类型
None

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

示例

```
c0 = model.AddQConstr(2*x*x + y*y + z, -OPTV_INF, 3, "c0")
c0.Set(OPTVStrAttr.NAME, "dummy")
model.Update()
print(c0.Get(OPTVStrAttr.NAME)) # dummy
```

7.11.3 Index() 二次约束索引

获取二次约束在模型中的索引值。

OPTVQConstr.**Index**()

返回
二次约束索引/下标。

返回类型
int

返回值

- **-1** –该二次约束在模型中不存在。
- **-2** –该二次约束已从模型中删除。
- **>=0** –该二次约束在模型中的索引/下标。

示例

```
c0 = model.AddQConstr(2*x*x + y*y + z, -OPTV_INF, 3, "c0")
c1 = model.AddQConstr(3*y*y + z, -OPTV_INF, 6, "c1")
print(c0.Index()) # 0
print(c1.Index()) # 1
```

7.11.4 SameAs() 二次约束对比

查询两二次约束是否相同。

OPTVQConstr.**SameAs**(*c2*)

参数
c2 (OPTVQConstr) –需要与当前二次约束对比的二次约束。

返回
二次约束是否相同。

返回类型
bool

返回值

- **true** –二次约束相同。
- **false** –二次约束不同。

示例

```
c0 = model.AddQConstr(2*x*x + y*y + z, -OPTV_INF, 3, "c0")
c1 = c0
invalid = model.AddQConstr(2*x*x + y*y + z, -OPTV_INF, 3, "Invalid")

c0.SameAs(c1)       # True
c0.SameAs(invalid)  # False
```

7.12 OPTVSOS SOS 约束

class OPTVSOS

特殊顺序集，special ordered set (SOS)，是指一组有序集合里，至多有一个非零值（SOS1 型）或至多有两个非零值（SOS2 型）。SOS 约束是对一组变量的取值进行限制的特殊约束，可以额外地指定模型中的整数条件。这类约束单独列出来，可以加快线性规划的求解速度。

OptVerse SOS 约束对象，用以从模型中获取 SOS 约束数据。这里，用户需要通过*OPTVModel.AddSOS()* 接口向模型中添加 SOS 约束，而非借助于构造函数。具体地，OptVerse 支持下列两种 SOS 约束：

7.12.1 Get() 查询 SOS 约束属性

查询 SOS 约束的属性，更多细节请参考*OptVerse 属性*。

OPTVSOS.**Get**(*attr*)

参数

attr (OPTVIntAttr / OPTVStrAttr) –需要查询的 SOS 约束属性，包括整数型和字符串型。

返回

属性值。

返回类型

int|str

Throws OPTVErrCode.UNKNOWN_ATTRIBUTE

未知约束属性。

示例

```
sos = model.AddSOS([x, y, z], [1, 2, 3], 1, name="sos")
# 查询SOS约束类型
result = sos.Get(OPTVIntAttr.SOS_TYPE)
```

7.12.2 set() 设置 SOS 约束属性

设置 SOS 约束属性，只有调用*OPTVModel.Update()* 后，新设置才可生效，更多细节请参*OptVerse 属性*。

OPTVSOS.**set**(*attr*, *value*)

参数

- **attr** (OPTVIntAttr / OPTVStrAttr) –需要设置的 SOS 约束属性，包括整数型和字符串型。
- **value** (*int* / *str*) –新属性值。

返回
无。

返回类型
None

Throws OPTVErrCode.UNKNOWN_ATTRIBUTE
未知约束属性。

示例

```
sos = model.AddSOS([x, y, z], [1, 2, 3], 1, name="sos")
sos.Set(OPTVStrAttr.NAME, "dummy")
model.Update()
print(sos.Get(OPTVStrAttr.NAME)) # dummy
```

7.12.3 Index() SOS 约束索引

获取 SOS 约束在模型中的索引值。

OPTVSOS.**Index**()

返回
SOS 约束索引/下标。

返回类型
int

返回值

- **-1** –该 SOS 约束在模型中不存在。
- **-2** –该 SOS 约束已从模型中删除。
- **>=0** –该 SOS 约束在模型中的索引/下标。

示例

```
sos = model.AddSOS([x, y, z], [1, 2, 3], 1, name="sos")
result = sos.Index()
```

7.12.4 SameAs() SOS 约束对比

查询两 SOS 约束是否相同。

OPTVSOS.**SameAs**(*c2*)

参数
c2 (OPTVSOS) –需要与当前 SOS 约束对比的 SOS 约束。

返回
SOS 约束是否相同。

返回类型
bool

返回值

- **true** –SOS 约束相同。
- **false** –SOS 约束不同。

示例

```
sos = model.AddSOS([x, y, z], [1, 2, 3], 1, name="sos")
sos1 = sos
invalid = OPTVSOS()

sos.SameAs(sos1)     # True
sos.SameAs(invalid) # False
```

7.13 OptVerse 一般约束

class OPTVGenConstr

OptVerse 一般约束，用于从模型中获取一般约束数据。特别地。一般约束需要通过 OPTVModel.AddGenConstrXxx 向模型中添加约束而构造，而不是借助于任何构造函数。

表 12: OptVerse 支持的一般约束类型

约束	类型	接口	描述
And	*OPTV_GENCONSTR_A*	*OPTVModel.AddGenConstrAnd()*	只有所有变量非零时，结果才非零
Or	*OPTV_GENCONSTR_O*	*OPTVModel.AddGenConstrOr()*	只要有一个变量非零，结果就非零
Indicator	*OPTV_GENCONSTR_I*	*OPTVModel.AddGenConstrIndicator*	返回值为布尔型，当且仅当相应的逻辑条件满足，取值为 1，否则取值为 0

7.13.1 Get() 查询一般约束属性

查询一般约束属性，关于可用属性请参考*OptVerse 属性*。

OPTVGenConstr.**Get**(*attr*)

参数
attr (OPTVIntAttr/OPTVDblAttr/OPTVStrAttr) –需要查询的约束属性，包括整数型、双精度型、字符和字符串型

返回
属性值。

返回类型
intlfloatlstr

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

示例

```
c0 = model.AddGenConstrAnd(x3, [x1, x2], "and")
# 查询一般约束类型
print(c0.Get(OPTVIntAttr.GENCONSTR_TYPE))
```

7.13.2 Set() 设置一般约束属性

设置一般约束属性，只有调用*OPTVModel.Update()* 后，新设置才可生效，更多细节请参考*OptVerse 属性*。

OPTVGenConstr.**set**(*attr*, *value*)

参数

- **attr** (OPTVDblAttr/OPTVStrAttr) –需要设置的约束属性，包括双精度型和字符串型
- **value** (*float*/*str*) –新属性值。

返回
无。

返回类型
None

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

示例

```
c0 = model.AddGenConstrAnd(x3, [x1, x2], "and")
c0.Set(OPTVStrAttr.NAME, "dummy")
model.Update()
print(c0.Get(OPTVStrAttr.NAME)) # dummy
```

7.13.3 Index() —般约束索引

获得一般约束在模型中的索引值。

OPTVGenConstr.**Index**()

返回
约束索引/下标。

返回类型
int

返回值

- **-1** –该约束在模型中不存在。
- **-2** –该约束已从模型中删除。
- **>=0** –该约束在模型中的索引/下标。

示例

```
c0 = model.AddGenConstrAnd(x3, [x1, x2], "and")
c0.Index()
```

7.13.4 SameAs() —般约束对比

查询两一般约束是否为同一对象。

OPTVGenConstr.**SameAs**(*c2*)

参数
c2 (OPTVGenConstr) –需要与当前约束对比的约束。

返回
约束是否相同。

返回类型
bool

返回值

- **true** –约束相同。
- **false** –约束不同。

示例

```
c0 = model.AddGenConstrAnd(x3, [x1, x2], "and")
c1 = c0
invalid = model.AddGenConstrOr(x3, [x1, x2], "Or")

c0.SameAs(c1)       # True
c0.SameAs(invalid)  # False
```

7.14 OptVerse 线性表达式

class OPTVLinExpr

线性表达式*OPTVLinExpr*类用于构建线性的目标函数或者约束函数。线性表达式对象由变量和常量组成，边获取边构造，如下例所示的表达式 `1 * x + 2 * y + 3 * x`，它包含以下元素：

下标	系数	变量
0	1	x
1	2	y
2	3	x

7.14.1 AddTerms() 增加多项

该功能向当前表达式（后）添加一组新的项。

OPTVLinExpr.**AddTerms**(*coefs*, *vars*, *cnt*)

参数

- **coefs** (*list[float]*) –新添加项的变量系数
- **vars** (*list[OPTVVar]*) –新添加项的变量列表
- **cnt** (*int*) –添加项的个数。

返回
无。

返回类型
None

抛出
OPTVErrCode.INVALID_ARGUMENT –vars 或 coefs 为空，或 cnt 取负值。

示例

```
expr = x + 1
expr.AddTerms([2.0, 3.0], [y, z], 2)
model.Update()
print(expr) # x + 2*y + 3*z + 1
```

7.14.2 Clear() 清空

清空该表达式，*OPTVLinExpr.Size()* 将返回 0. 同时，该表达式从数值意义上将等同于 0.

OPTVLinExpr.**Clear**()

返回
无。

返回类型
None

示例

```
expr = x + 1
expr.Clear()
print(expr)       # 0
print(expr.Size)  # 0
```

7.14.3 GetCoef() 获取系数

据指定索引值获得表达式中对应变量的系数。

OPTVLinExpr.**GetCoef**(*i*)

参数
i (*int*) –指定的索引值。

返回
对应的变量系数。

返回类型
float

抛出
OPTVErrCode.INDEX_OUT_OF_RANGE –若 i $\notin [0, \texttt{this->Size()}]$.

示例

```
expr = 2*x + 3*y + z + 1
result = expr.GetCoef(1)
print(result) # 3
```

7.14.4 GetVar() 变量获取

根据指定索引值获得表达式中的对应变量。

OPTVLinExpr.**GetVar**(*i*)

参数
i (*int*) –指定的索引值。

返回
对应的变量。

返回类型
OPTVVar

抛出
OPTVErrCode.INDEX_OUT_OF_RANGE –若 i $\notin [0, \texttt{this->Size()}]$.

示例

```
expr = 2*x + 3*y + 1
result = expr.GetVar(0)
print(result) # x
```

7.14.5 GetConstant() 获取常数项

获得当前表达式中的常数项。

OPTVLinExpr.**GetConstant**()

返回
表达式中的常量数值。

返回类型
float

示例

```
expr = 2*x + 3*y + z + 6
result = expr.GetConstant()
print(result) # 6
```

7.14.6 GetValue() 取值

获得该表达式的取值。

OPTVLinExpr.**GetValue**()

返回
对应原始解的表达式的取值，即该表达式的变量值与系数的乘积之和。

返回类型
float

抛出
OPTVErrCode.DATA_NOT_AVAILABLE –当前变量取值不存在。

示例

```
model.Optimize()
obj = model.GetObjective()
result = obj.GetValue()
```

7.14.7 Remove() 删除项

从表达式中删除项。

OPTVLinExpr.**Remove**(*i*)
根据指定索引值从该表达式中删除项。

参数
i (*int*) –指定的索引值。

返回
无。

返回类型
None

抛出
OPTVErrCode.INDEX_OUT_OF_RANGE –若 i $\notin [0,$ this->Size()$]$.

示例

```
expr = 2*x + 3*y + 1
expr.Remove(0)
print(expr) # 3*y + 1
```

OPTVLinExpr.**Remove**(*v*)

从线性表达式中删除指定变量涉及的所有项。

参数
v (OPTVVar) –指定要删除的变量。

返回
是否成功删除任何项。

返回类型
bool

返回值

- **true** –成功删除任何项。
- **false** –未删除任何项。

抛出
OPTVErrCode.NOT_IN_MODEL –指定的变量（在模型中）不存在。

示例

```
expr = 2*x + 3*y + 1
expr.Remove(y)
print(expr) # 2*x + 1
```

7.14.8 SameAs() 线性表达式对比

查询两线性表达式是否相同。

OPTVLinExpr.**SameAs**(*e2*)

参数
e2 (OPTVLinExpr) –需要与当前线性表达式对比的线性表达式。

返回
线性表达式是否相同。

返回类型
bool

返回值

- **true** –线性表达式相同。
- **false** –线性表达式是不同。

示例

```
expr1 = 2*x + 3*y + 1
expr2 = x + y
expr3 = 2*x + 3*y + 1
print(expr1.SameAs(expr3)) # True
print(expr1.SameAs(expr2)) # False
```

7.14.9 Size() 规模

获得该表达式的规模。

OPTVLinExpr.**Size**()

该数值对应线性表达式中的变量个数，不论变量是否重复出现。例如表达式 x+y 对应 Size = 2，而 x+y+x 对应 Size = 3.

返回
该表达式的规模。

返回类型
int

示例

```
expr = 2*x + 3*y + 1
print(expr.Size) # 2
```

7.15 OPTVMLinExpr 线性矩阵表达式

class OPTVMLinExpr

线性矩阵表达式对象。线性矩阵表达式对象是对*OPTVMVar* 对象进行线性计算后得到的。用户可以使用重载运算符来构建线性矩阵表达式，例如 A @ x 或 x + 1（x 为*OPTVMVar* 对象）。用户也可以使用*OPTVMLinExpr.Zeros()* 来构建一个所有表达式均为零的 OPTVMLinExpr 对象。

7.15.1 Copy() 拷贝

深度复制一个 OPTVMLinExpr 对象。

OPTVMLinExpr.**Copy**()

返回
原 OPTVMLinExpr 对象的拷贝。

返回类型
OPTVMLinExpr

示例

```
expr = model.AddMVar(3, name="x") + 1
expr_copy = expr.Copy()
```

7.15.2 Clear() 清空表达式

将 OPTVMLinExpr 对象中的所有表达式重置为 0。

OPTVMLinExpr.**Clear**()

返回
无。

返回类型
None

示例

```
expr = model.AddMVar(3, name="x") + 1
expr.Clear()
# 矩阵中所有表达式被重置为0.0
print(expr) # ['0.0' '0.0' '0.0']
```

7.15.3 GetValue() 获取线性表达式的值

获取 OPTVMLinExpr 对象中线性表达式的值。

OPTVMLinExpr.**GetValue**()

返回
一个 NumPy 数组，其维度和形状与 OPTVMLinExpr 对象相同，其元素为对应表达式的值。

返回类型
numpy.ndarray

示例

```
expr = model.AddMVar(3, name="x") + 1
model.AddConstrs((expr[i].Item() <= 1 for i in range(3)), name='c')
model.Optimize()
values = expr.GetValue()
```

7.15.4 Item() 获取 OPTVMLinExpr 对象中的线性表达式

若是 OPTVMLinExpr 中只有一个 OPTVLinExpr 对象，返回该 OPTVLinExpr 对象的拷贝。若是在 OPTVMLinExpr 有多个线性表达式时调用该方法，会产生 ValueError。

OPTVMLinExpr.**Item**()

返回
OPTVMLinExpr 的唯一元素 OPTVLinExpr 的拷贝。

返回类型
OPTVLinExpr

抛出
ValueError –OPTVMLinExpr 中有多个元素。

示例

```
expr = model.AddMVar(3, name="x") + 1
expr_sub = expr[0, 1]        #␣
↪一个0维的线性表达式矩阵，只有一个线性表达式对象
expr_le = expr[0, 1].Item() # 相应位置线性表达式的拷贝
```

7.15.5 Zeros() 创建元素全为零的线性矩阵表达式

创建一个元素全为零的 OPTVMLinExpr 对象。

OPTVMLinExpr.**Zeros**(*shape*)

参数

shape (*int | tuple[int, ...]*) –表示创建的 OPTVMLinExpr 对象的形状。

返回

元素全为零的 OPTVMLinExpr 对象。

返回类型

OPTVMLinExpr

示例

```
expr_zeros = OPTVMLinExpr.Zeros((2, 2)) #␣
↪创建一个元素全为0的OPTVMLinExpr对象
```

7.15.6 Sum() 求和

对 OPTVMLinExpr 中的元素进行求和。

OPTVMLinExpr.**Sum**(*axis=None*)

参数

axis (*int | None*) –若是整数，对给定轴进行求和；若为 None，对所有元素进行求和。

返回

按照给定轴求和后的 OPTVMLinExpr 对象。

返回类型

OPTVMLinExpr

示例

```
expr = model.AddMVar((2, 3), name='x') + 1
sum_row = expr.Sum(axis=0) # 对每一行的OPTVLinExpr进行求和
sum_col = expr.Sum(axis=1) # 对每一列的OPTVLinExpr进行求和
sum_all = expr.Sum()       #␣
↪对所有的OPTVLinExpr进行求和，返回一个0维的OPTVMLinExpr对象
```

7.15.7 Ndim 维度

获取 OPTVMLinExpr 对象的维度。

OPTVMLinExpr.**Ndim**

返回
OPTVMLinExpr 对象的维度数目。

返回类型
int

示例

```
expr1 = model.AddMVar((3,), name="x") + 1
print(expr1.Ndim) # 1
expr2 = model.AddMVar((1, 3), name='y') + 1
print(expr2.Ndim) # 2
```

7.15.8 Shape 形状

获取 OPTVMLinExpr 对象的形状。

OPTVMLinExpr.**Shape**

返回
一个表示 OPTVMLinExpr 对象形状的整数元组。

返回类型
tuple[int, ···]

示例

```
expr1 = model.AddMVar((3,), name="x") + 1
print(expr1.Shape) # (3,)
expr2 = model.AddMVar((1, 3), name='y') + 1
print(expr2.Shape) # (1, 3)
```

7.15.9 Size 元素个数

获取 OPTVMLinExpr 对象元素的个数。

OPTVMLinExpr.**Size**

返回
OPTVMLinExpr 对象元素的个数。

返回类型
int

示例

```
expr1 = model.AddMVar((3,), name="x") + 1
print(expr1.Size) # 3
expr2 = model.AddMVar((2, 3), name='y') + 1
print(expr2.Size) # 6
```

7.16 OptVerse 二次表达式

class OPTVQuadExpr

二次表达式类*OPTVQuadExpr* 用于构建二次的目标函数或约束。二次表达式对象由变量和常量组成，通常包含三部分：二次项、线性项和常数项。其中，二次项依照第一个变量、第二个变量和系数的顺序保存在容器中，而线性部分以前一章节所述的线性表达式的形式保存。二次表达式的索引仅限于获取二次项，线性项部分的需单独获取。如下例所示的二次表达式 `1*x*x + 2*x*y + 3*x*y + 4*y*y + 5*x + 6*y + 7*x` 的描述如下：

索引	系数	变量 1	变量 2
0	1	x	x
1	2	x	y
2	3	x	y
3	4	y	y

它的线性项部分需要通过*OPTVQuadExpr.GetLinExpr()* 接口访问，使用方法可参考线性表达式 。

索引	系数	变量
0	5	x
1	6	y
2	7	x

7.16.1 AddConstant() 增加常数项

向二次表达式增加常数项。

OPTVQuadExpr.**AddConstant**(*c*)

此函数向当前二次表达式添加常数项，但不是覆盖。

参数

c (*float*) –要添加的常数。

返回

无。

返回类型

None

示例

```
quad_expr = x*y + x
quad_expr.AddConstant(3)
print(quad_expr) # x*y + x + 3
```

7.16.2 AddTerm() 增加单项

向当前二次表达式后添加一项。

OPTVQuadExpr.**AddTerm**(*coeff*, *var*)

该函数向当前二次表达式后添加 coeff * var 项。

参数

- **coeff** (*float*) –系数值。
- **var** (OPTVVar) –变量对象。

返回

无。

返回类型

None

抛出

OPTVErrCode.INVALID_ARGUMENT –需添加的变量 var 为空。

示例

```
quad_expr = x*y + x
quad_expr.AddTerm(3, z)
print(quad_expr) # x*y + x + 3*z
```

下列函数也会向当前二次表达式后添加一项。

OPTVQuadExpr.**AddTerm**(*coeff*, *var1*, *var2*)

该函数向当前二次表达式后添加 coeff * var1 * var2 项。

参数

- **coeff** (*float*) –系数值。
- **var1** (OPTVVar) –该项的第一个变量。
- **var2** (OPTVVar) –该项的第二个变量。

返回

无。

返回类型

None

抛出

OPTVErrCode.INVALID_ARGUMENT –需添加的项中变量 var1 或 var2 为空。

示例

```
quad_expr = x*y + x
quad_expr.AddTerm(3, z, x)
print(quad_expr) # x*y + x + 3*z*x
```

7.16.3 AddTerms() 增加多项

向当前二次表达式后添加多项。

OPTVQuadExpr.**AddTerms**(*coeff*, *var*, *cnt*)

该函数向当前二次表达式后添加 $\text{coeff}_i * \text{var}_i, \forall i \in [0, \text{cnt}]$.

参数

- **coeff** (*list[float]*) –C-风格的系数序列。
- **var** (*list[OPTVVar]*) –C-风格的变量序列。
- **cnt** (*int*) –需要添加的项的个数。

返回

无。

返回类型

None

抛出

OPTVErrCode.INVALID_ARGUMENT –变量 var 或 coeff 为空，或 cnt 取负值。

示例

```
quad_expr = y*y
quad_expr.AddTerms([1, 2], [x, z], 2)
print(quad_expr) # y*y + x + 2*z
```

下列函数也会向当前二次表达式后添加多项。

OPTVQuadExpr.**AddTerms**(*coeff*, *var1*, *var2*, *cnt*)

该函数向当前二次表达式后添加 $\text{coeff}_i * \text{var1}_i * \text{var2}_i, \forall i \in [0, \text{cnt}]$.

参数

- **coeff** (*list[float]*) –C-风格的系数序列。
- **var1** (*list[OPTVVar]*) –C-风格的变量 1 序列。
- **var2** (*list[OPTVVar]*) –C-风格的变量 2 序列。
- **cnt** (*int*) –需要添加的项的个数。

返回

无。

返回类型

None

抛出

OPTVErrCode.INVALID_ARGUMENT –变量 var1，var2 或 coeff 为空，或 cnt 取负值。

示例

```
quad_expr = y*y
quad_expr.AddTerms([1, 2], [x, z], [x, z], 2)
print(quad_expr) # y*y + x*x + 2*z*z
```

7.16.4 Clear() 清空

清空该表达式。

OPTVQuadExpr.**Clear**()

该函数清空该表达式涉及的所有存储容器，*OPTVQuadExpr.Size()* 将返回 0。同时，该表达式也清空相应的线性表达式部分，重置常数项为 0。

返回
无。

返回类型
None

示例

```
quad_expr = x*y + z
quad_expr.Clear()
print(quad_expr.Size()) # 0
```

7.16.5 GetCoeff() 获取系数

据指定索引值获得表达式中对应二次项的系数。

OPTVQuadExpr.**GetCoeff**(*i*)

参数
i (*int*) –指定的索引值。

返回
对应的二次项系数。

返回类型
float

抛出
OPTVErrCode.DATA_NOT_AVAILABLE –若 i $\notin [0, \texttt{this->Size()}]$。

示例

```
quad_expr = x*y - 2*y*z
coeff1 = quad_expr.GetCoeff(0)
coeff2 = quad_expr.GetCoeff(1)
print(coeff1) # 1
print(coeff2) # -2
```

7.16.6 GetConstant() 获取常量

获得当前表达式中的常数项。

OPTVQuadExpr.**GetConstant**()

返回
表达式中的常量数值。

返回类型
float

示例

```
quad_expr = x*y - 2*y*z + 3
constant = quad_expr.GetConstant()
print(constant) # 3
```

7.16.7 GetLinExpr() 线性部分

获取二次表达式的线性部分。

OPTVQuadExpr.**GetLinExpr**()

返回
二次表达式的线性部分。

返回类型
OPTVLinExpr

示例

```
quad_expr = x*y - 2*z + 3
lin_expr = quad_expr.GetLinExpr()
print(lin_expr) # -2*z
```

7.16.8 GetValue() 取值

获得该二次表达式的取值。

OPTVQuadExpr.**GetValue**()

返回
对应原始解的表达式的取值。

返回类型
float

抛出
OPTVErrCode.DATA_NOT_AVAILABLE –当前变量取值不存在。

示例

```
model.Optimize()
result = model.GetQObjective()
print(result.GetValue())
```

7.16.9 GetVar1() 获取变量 1

根据指定索引值获得表达式中的对应二次项的第一个变量。

OPTVQuadExpr.**GetVar1**(*i*)

该函数返回表达式中的对应二次项的第一个变量，如表达式 `x*x + x*y + y*y` 的描述如下：

索引	变量 1	变量 2
0	x	x
1	x	y
2	y	y

参数
i (*int*) –指定的二次项对应的索引值。

返回
对应二次项的第一个变量。

返回类型
OPTVVar

抛出
OPTVErrCode.DATA_NOT_AVAILABLE –若 i ∉ [0, this->Size()]。

示例

```
quad_expr = x1*z + *x2*y
print(quad_expr.GetVar1(1)) # x2
print(quad_expr.GetVar2(1)) # y
```

7.16.10 GetVar2() 获取变量 2

根据指定索引值获得表达式中的对应二次项的第二个变量。请参考*OPTVQuadExpr.GetVar1()* 和示例。

OPTVQuadExpr.**GetVar2**(*i*)

该函数返回表达式中的对应二次项的第二个变量。

参数
i (*int*) –指定的二次项对应的索引值。

返回
对应二次项的第二个变量。

返回类型
OPTVVar

抛出
OPTVErrCode.DATA_NOT_AVAILABLE –若 i ∉ [0, this->Size()]。

示例

```
quad_expr = x1*z + *x2*y
print(quad_expr.GetVar1(1)) # x2
print(quad_expr.GetVar2(1)) # y
```

7.16.11 Remove() 删除项

从二次表达式中删除项。

OPTVQuadExpr.**Remove**(*i*)

根据指定索引值从二次表达式中删除二次项，但不可通过此函数删除线性项的任何部分。

参数

i (*int*) –指定的索引值。

返回

无。

返回类型

None

抛出

OPTVErrCode.DATA_NOT_AVAILABLE –若 i $\notin [0, \texttt{this->Size()}]$。

示例

```
quad_expr = x*x + x*y + y*y
quad_expr.Remove(1)
print(quad_expr) # x*x + y*y
```

OPTVQuadExpr.**Remove**(*v*)

从二次表达式中（包括线性部分）删除指定变量涉及的所有项。

参数

v (OPTVVar) –指定要删除的变量。

返回

无。

返回类型

None

抛出

OPTVErrCode.NOT_IN_MODEL –指定的变量（在模型中）不存在。

示例

```
quad_expr = x*x + x + y*y
quad_expr.Remove(x)
print(quad_expr) # y*y
```

7.16.12 SameAs() 二次表达式对比

查询两二次表达式是否相同。

OPTVQuadExpr.**SameAs**(*e2*)

参数

e2 (OPTVQuadExpr) –需要与当前二次表达式对比的线性表达式。

返回

二次表达式是否相同。

返回类型
bool

返回值

- **true** –二次表达式相同。
- **false** –二次表达式不同。

示例

```
quad_expr = x*x + x + y*y
quad_expr1 = x*z + y + 1
quad_expr2 = quad_expr
print(quad_expr.SameAs(quad_expr1)) # False
print(quad_expr.SameAs(quad_expr2)) # True
```

7.16.13 Size() 规模

获得该表达式的规模。

OPTVQuadExpr.**Size**()

该数值对应当前二次表达式中的 二**次**项的个数，不论重复与否。例如，表达式 `x*x + y*y + y`` 对应 ``Size = 2`，而表达式 `x*x + x*x + y*y + y` 对应 `Size = 3`.

返回
该表达式的规模。

返回类型
int

示例

```
quad_expr = x*x + x + y*y
print(quad_expr.Size()) # 2
```

7.17 OptVerse 模型

class OPTVModel

7.17.1 AddConstr() 增加线性约束

向模型中添加单个线性约束。

OPTVModel.**AddConstr**(*expr*, *lhs*, *rhs*, *name=''*)

该函数向模型中添加一个线性约束。当需要一次性添加大量约束时，建议使用批量模式（*OPTVModel.AddConstrs()*），虽然大多数情况下，效率差别不大。

参数

- **expr** (OPTVLinExpr) –约束的线性表达式部分。
- **lhs** (*float*) –约束的下界。

- **rhs** (*float*) –约束的上界。
- **name** (*str*) –约束的名字。

返回
添加到模型的约束对象。

返回类型
OPTVConstr

抛出
OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

示例

```
# 添加线性约束：1 <= x+y <= 10
model.AddConstr(x+y, 1, 10, "c0")
```

7.17.2 AddConstrs() 批量增加线性约束

向模型中添加多个线性约束。

OPTVModel.**AddConstrs**(*expr*, *lhs*, *rhs*, *name*, *count*)

该函数一次性向模型中添加多个线性约束，尤其适用于添加大量约束的情形。

参数

- **expr** (*list[OPTVLinExpr]*) –约束的线性表达式序列。
- **lhs** (*list[float]*) –约束的左端值序列
- **rhs** (*list[float]*) –约束的右端值序列。
- **name** (*list[str]*) –约束的名字序列。
- **count** (*int*) –**可选**, 添加约束的个数，默认值为参数序列的长度。v1.2.6 前为必须参数，其后用户可省略此参数，该选项可以通过其他参数序列推断获取。

返回
包含所添加到模型的约束的列表。

返回类型
list[*OPTVConstr*]

抛出
OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 包含的约束名与模型中的其他组件重名。

示例

```
# 批量添加线性约束
model.AddConstrs([x+y, x+z, 2*x], [1, 2, 3], [6, 7, 8], ["c0", "c1",
↪"c2"])
```

7.17.3 AddQConstr() 增加单个二次约束

向模型中添加单个二次约束。

OPTVModel.**AddQConstr**(*expr*, *lhs*, *rhs*, *name=''*)

该函数向模型中添加一个二次约束。

参数

- **expr** (OPTVQuadExpr) –约束的二次表达式部分。
- **lhs** (*float*) –约束的下界。
- **rhs** (*float*) –约束的上界。
- **name** (*str*) –约束的名字。

返回

添加到模型的约束对象。

返回类型

OPTVQConstr

抛出

OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

示例

```
# 添加二次约束：x*x + 2*y +1 <= 100
model.AddQConstr(x*x+2*y+1, -OPTV_INF, 100, name="qconstr")
```

7.17.4 AddGenConstrAnd() 增加一般约束 AND

向模型中添加单个 AND 一般约束。

OPTVModel.**AddGenConstrAnd**(*resultant*, *conVars*, *name=''*)

该函数向模型中添加单个 AND 一般约束：只有所有变量非零时，结果才非零。

参数

- **resultant** (OPTVVar) –约束的布尔型返回值。
- **conVars** (*list[OPTVVar]*) –所有参与检验的变量列表。
- **name** (*str*) –约束的名字。

返回

添加到模型的一般约束对象。

返回类型

OPTVGenConstr

抛出

OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

示例

```
model.AddGenConstrAnd(x3, [x1, x2], "and")
```

7.17.5 AddGenConstrOr() 增加一般约束 OR

向模型中添加单个 OR 一般约束。

OPTVModel.**AddGenConstrOr**(*resultant*, *conVars*, *name=''*)

该函数向模型中添加单个 OR 一般约束：只要有一个变量非零，结果就非零。

参数

- **resultant** (OPTVVar) –约束的布尔型返回值。
- **conVars** (*list[OPTVVar]*) –所有参与检验的变量的序列。
- **name** (*str*) –约束的名字。

返回

添加到模型的一般约束对象。

返回类型

OPTVGenConstr

抛出

OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

示例

```
model.AddGenConstrOr(x3, [x1, x2], "or")
```

7.17.6 AddGenConstrIndicator() 增加一般约束 INDICATOR

向模型中添加单个 INDICATOR 一般约束。

OPTVModel.**AddGenConstrIndicator**(*binVar*, *binVal*, *expr*, *sense*, *rhs*, *name=''*)

该函数向模型中添加单个 INDICATOR 一般约束：当且仅当相应的逻辑条件满足，取值为 1，否则取值为 0.

参数

- **binVar** (OPTVVar) –二进制指示变量。
- **binVal** (*float*) –要求线性约束必须满足的二进制指示变量的值 (0 or 1).
- **expr** (OPTVLinExpr) –参与检验的线性约束表达式。
- **sense** (*str*) –线性约束的含义，如*OPTV_SENSE_LESS*, *OPTV_SENSE_EQUAL*, 或*OPTV_SENSE_GREATER*.
- **rhs** (*float*) –参与检验的线性约束右端值。
- **name** (*str*) –约束的名字。

返回

添加到模型的一般约束对象。

返回类型

OPTVGenConstr

抛出

OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

示例

```
# x3 = 1 -> x2 + x1 + 1 <= 10
model.AddGenConstrIndicator(x3, 1, x2+x1+1, OPTV_SENSE_LESS, 10, name=
↪"indicator")
```

7.17.7 AddGenConstrAbs() 增加一般约束 ABS

向模型中添加单个 ABS 一般约束：$r = \text{abs}(x)$。此约束条件保证因变量 r 的值为自变量 x 的绝对值，即 $r = |x|$, 其中 $r \geq 0$。

OPTVModel.**AddGenConstrAbs**(*res_var*, *input_var*, *name=''*)

该函数向模型中添加单个 ABS 一般约束。

参数

- **res_var** (OPTVVar) –结果变量，该变量的值为另一变量的绝对值。
- **input_var** (OPTVVar) –该变量的绝对值将被另一变量采用。
- **name** (*str*) –约束的名字。

返回

添加到模型的一般约束对象。

返回类型

OPTVGenConstr

抛出

OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他约束重名。

示例

```
model.AddGenConstrAbs(x1, x2, "abs")
```

7.17.8 AddGenConstrNorm() 增加一般约束 NORM

向模型中添加单个 NORM 一般约束: $r = \text{norm}\{x_1, \ldots, x_n\}$，其中因变量 r 的值为向量 $x_1, \ldots, x_n$ 的范数。

OPTVModel.**AddGenConstrNorm**(*res_var*, *vars_list*, *norm_type*, *name=''*)

该函数向模型中添加单个 NORM 一般约束。

参数

- **res_var** (OPTVVar) –结果变量，其值为参数向量的范数。
- **vars_list** (*list[OPTVVar]*) –用于计算向量范数的变量。
- **norm_type** (*int*) –范数类型：-1 为 L^∞ 范数，0 为 L^0 范数，1 为 L^1 范数。
- **name** (*str*) –约束的名字。

返回

添加到模型的一般约束对象。

返回类型

OPTVGenConstr

抛出

OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他约束重名。

示例

```
model.AddGenConstrNorm(x1, [x2, x3, x4], 0, "norm")
```

7.17.9 AddGenConstrMax() 添加一般约束 MAX

向模型中添加单个 MAX 约束：$r = \max\{x_1, \ldots, x_n\}$ 。

该约束确保因变量 r 的取值为变量向量 $x_1, \ldots, x_n$ 中的最大值。

OPTVModel.**AddGenConstrMax**(*resultant*, *var_list*, *name=''*, *constant=-OPTV_INF*)

该函数向模型中添加单个 MAX 约束。

参数

- **resultant** (OPTVVar) –结果变量，该变量的值为其他变量的最大值。
- **var_list** (*list[OPTVVar]*) –变量向量，该向量的最大值会被结果变量采用。
- **name** (*str*) –约束名称。
- **constant** (*double*) –[**可选**] 参与 MAX 操作的常量。

返回

一个添加到模型中的一般约束对象

返回类型

OPTVGenConstr

抛出

OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他约束重名。

示例

```
# x1 = max([x2, x3, x4])
model.AddGenConstrMax(x1, [x2, x3, x4] name="max")
```

7.17.10 AddGenConstrMin() 添加一般约束 MIN

向模型中添加单个 MIN 约束：$r = \min\{x_1, \ldots, x_n\}$ 。

该约束确保因变量 r 的取值为变量向量 $x_1, \ldots, x_n$ 中的最小值。

OPTVModel.**AddGenConstrMin**(*resultant*, *var_list*, *name=''*, *constant=OPTV_INF*)

该函数向模型中添加单个 MIN 约束。

参数

- **resultant** (OPTVVar) –结果变量，该变量的值为其他变量的最小值。
- **var_list** (*list[OPTVVar]*) –变量向量，该向量的最小值会被结果变量采用。
- **name** (*str*) –约束名称。
- **constant** (*double*) –[**可选**] 参与 MIN 操作的常量。

返回

一个添加到模型中的一般约束对象。

返回类型
OPTVGenConstr

抛出
OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他约束重名。

示例

```
# x1 = min([x2, x3, x4])
model.AddGenConstrMin(x1, [x2, x3, x4] name="min")
```

7.17.11 AddSOS() 增加单个 SOS 约束

向模型中添加单个 SOS 约束。

OPTVModel.**AddSOS**(*vars*, *weights*, *type*, *name=''*)

参数

- **vars** (*list[OPTVVar]*) –SOS 约束所涉及变量的序列。
- **weights** (*list[float]*) –SOS 的变量权重的序列。
- **type** (*int*) –SOS 类型，取值为 1 或 2.
- **name** (*str*) –SOS 约束的名字。

返回
添加到模型的 SOS 约束对象。

返回类型
OPTVSOS

Throws OPTVErrCode.INVALID_ARGUMENT
type 取值不是 1 或 2.

Throws OPTVErrCode.NOT_IN_MODEL
vars 序列中的某个变量不属于当前模型。

Throws OPTVErrCode.INDEX_OUT_OF_RANGE
当前模型存在无效下标，比如模型关联的环境发生溢出时，执行本操作有可能抛出此错误。

示例

```
# 添加SOS1约束：最多只能有一个变量非零
model.AddSOS([x, y, z], [1, 2, 3], 1, name="sos1")
# 添加SOS2约束：最多只能有两个相邻的变量非零
model.AddSOS([x, y, z], [1, 2, 3], 2, name="sos2")
```

7.17.12 AddVar() 增加变量

向模型中添加单个变量。

OPTVModel.**AddVar**(*lb*, *ub*, *obj*, *type*, *name=''*)

该函数向模型中添加一个变量。当需要一次性添加大量变量时，建议使用批量模式（*OPTVModel.AddVars()*），虽然大多数情况下，效率差别不大。

参数

- **lb** (*float*) –变量的下界。
- **ub** (*float*) –变量的上界。
- **obj** (*float*) –变量在目标函数中的系数。
- **type** (*str*) – 变量类型，取值包括*OPTV_CONTINUOUS*，*OPTV_INTEGER*，*OPTV_BINARY*.
- **name** (*str*) –变量名称。

返回

添加到模型的变量对象。

返回类型

OPTVVar

抛出

OPTVErrCode.INVALID_ARGUMENT –未知变量类型，name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

示例

```
# 添加一个0-1变量
model.AddVar(0, 1, 0, OPTV_BINARY, name="x")
```

OPTVModel.**AddVar**(*lb*, *ub*, *obj*, *type*, *col*, *name=''*)

该函数向模型中添加一个变量，并可以设置现有约束中的相应系数。当需要一次性添加大量变量时，建议使用批量模式(*OPTVModel.AddVars()*)，虽然大多数情况下，效率差别不大。

参数

- **lb** (*float*) –变量的下界。
- **ub** (*float*) –变量的上界。
- **obj** (*float*) –变量在目标函数中的（线性）系数。
- **type** (*str*) – 变量类型，取值包括*OPTV_CONTINUOUS*，*OPTV_INTEGER*，*OPTV_BINARY*。
- **col** (OPTVColumn) –设置约束系数的列对象。
- **name** (*str*) –变量名称。

返回

添加到模型的变量对象。

返回类型

OPTVVar

抛出

OPTVErrCode.INVALID_ARGUMENT –未知变量类型，name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

示例

```
# 添加一个0-1变量，并设置该变量在现有约束当中的系数
col = OPTVColumn()
col.addTerm(1, c0)
model.AddVar(0, 1, 0, OPTV_BINARY, col, name="x")
```

7.17.13 AddMVar() 添加矩阵变量

向模型中添加一个 OPTVMVar 对象。

OPTVModel.**AddMVar**(*shape*, *lb=0.0*, *ub=OPTV_INF*, *obj=0.0*, *vtype=OPTV_CONTINUOUS*, *name=”*)

参数

- **shape** –OPTVMVar 对象的形状，可以是一个整数或者元组。
- **lb** –可选参数。变量下界。
- **ub** –可选参数。变量上界。
- **obj** –可选参数。变量在目标函数中的系数。
- **vtype** –可选参数。变量类型取值包括*OPTV_CONTINUOUS*, *OPTV_INTEGER*, *OPTV_BINARY*。
- **name** –可选参数。变量名称。

变量 lb、ub、obj 以及 vtype 可以是一个标量、列表或者 Numpy 数组。它们的形状与 OPTVMVar 的形状相同，或者可以通过 NumPy 的广播机制形成给定的形状。

变量 name 可以是一个字符串，不同变量的名称会根据其索引在 name 后面添加后缀，或者是一个与 OPTVMVar 对象形状相同的 NumPy 数组。

返回

一个新的 OPTVMVar 对象。

返回类型

OPTVMVar

示例

```
# 添加一个2行3列的连续变量矩阵
model.AddMVar((2, 3), name="x")
# 添加一个连续变量向量，指定变量下界
model.AddMVar((3), lb=[1, 2, 3], name="y")
```

7.17.14 AddMConstr() 添加线性约束矩阵

通过矩阵表达方式向模型中添加约束。添加约束的形式为 `lhs <= Ax <= rhs`. 约束中的 A 为 `numpy.ndarray`.

`OPTVModel.`**`AddMConstr`**(*A*, *x*, *lhs*, *rhs*, *name=''*)

参数

- **A** –一个二维的 `numpy.ndarray`.
- **x** –该参数可以是*OPTVMVar* 对象, 或者是元素为*OPTVVar* 的列表, 也可以为 None(这时会用模型中的*OPTVVar* 填充该参数). 变量个数与参数 A 中的列数相同。
- **lhs** –约束的左端项向量, 一个一维的 `numpy.ndarray`。向量中元素个数与参数 A 的行数相同。
- **rhs** –约束的右端项向量, 一个一维的 `numpy.ndarray`。向量中元素个数与参数 A 的行数相同。
- **name** –新增约束的名称。若是该参数为字符串标量, 会使用约束在矩阵中的下标填充该标量；若是该参数为一个字符串列表或者一维的 `numpy.ndarray` ，那么元素个数一定要与参数 A 的行数相同。

返回

一个新的 OPTVMConstr 对象

返回类型

OPTVMConstr

抛出

- **TypeError** –x 不是列表、*OPTVMVar* 或者 None
- **IndexError** –x 的维度不符合要求
- **ValueError** –x 的形状不符合要求

示例

```
vars = model.AddMVar(3, vtype=OPTV_BINARY, name=['x', 'y', 'z'])
model.Update()
obj = vars @ np.array([1, 1, 2])
A = np.array([[1, 2, 3], [1, 1, 0]])
constrs = model.AddMConstr(A, vars, [-OPTV_INF, 1], [4, OPTV_INF],␣
↪name='c')
```

7.17.15 AddVars() 批量增加变量

向模型中添加多个变量。

`OPTVModel.`**`AddVars`**(*lb*, *ub*, *obj*, *type*, *name*, *count=None*)

该函数一次性向模型中添加多个变量，尤其适用于添加大量变量的情形。

参数

- **lb** (*list[float]*) –变量的下界列表。
- **ub** (*list[float]*) –变量的上界列表。
- **obj** (*list[float]*) –变量对应的目标系数列表。

- **type** (*list[str]*) – 变量类型，包括 py:data:*OPTV_CONTINUOUS*, py:data:*OPTV_INTEGER*, py:data:*OPTV_BINARY*.
- **name** (*list[str]*) –变量名称列表。
- **count** (*int*) –**可选**, 添加变量的个数，默认值为参数序列的长度。v1.2.6 前为必须参数，其后用户可省略此参数，该选项可以通过其他参数序列推断获取。

返回
添加到模型的变量列表。

返回类型
list[*OPTVVar*]

抛出
OPTVErrCode.INVALID_ARGUMENT –`type` 包含未知变量类型，`name` 为空，数值参数中存在 `NaN` 或者 `name` 包含的变量名与模型中的其他组件重名。

示例

```
# 批量添加0-1变量
model.AddVars([0]*3, [1]*3, [0]*3, [OPTV_BINARY]*3, name=["x", "y", "z
↪"])
```

`OPTVModel.`**`AddVars`**(*lb*, *ub*, *obj*, *type*, *col*, *names*, *count=None*)

该函数向模型中添加多个变量，并可以设置现有约束中的相应系数。当需要一次性添加大量变量时，建议使用此功能，虽然大多数情况下，效率差别不大。

参数

- **lb** (*list[float]*) –变量的下界列表。
- **ub** (*list[float]*) –变量的上界列表。
- **obj** (*list[float]*) –变量对应的目标系数列表。
- **type** (*list[str]*) – 变量类型，包括*OPTV_CONTINUOUS*, *OPTV_INTEGER*, *OPTV_BINARY*.
- **col** (*list[OPTVColumn]*) –设置约束系数的列对象序列。
- **names** (*list[float]*) –变量名称列表。
- **count** (*int*) –添加变量的个数。

返回
添加到模型的变量序列。

返回类型
OPTVVar[]

Throws OPTVErrCode.INVALID_ARGUMENT
`type` 包含未知变量类型，`name` 为空，数值参数中存在 `NaN` 或者 `name` 包含的变量名与模型中的其他组件重名。

示例

```
# 批量添加0-1变量，并设置变量在现有约束当中的系数
col_x = OPTVColumn()
col_x.addTerm(1, c0)
col_y = OPTVColumn()
```

(续下页)

(接上页)

```
col_y.addTerm(1, c0)
col_z = OPTVColumn()
col_z.addTerm(1, c0)
model.AddVars([0]*3, [1]*3, [0]*3, [OPTV_BINARY]*3, [col_x, col_y,␣
↪col_z], name=["x", "y", "z"])
```

7.17.16 CheckSolution() 校验解

检验指定的解是否可行。

OPTVModel.**CheckSolution**(*solution*, *tol=1e-8*)

该函数供用户检验指定的解是否满足当前模型的可行性要求，但 **不可以**检验内部存储的解。

参数

- **solution** (*list[float]*) –(列) 解向量。
- **tol** (*float*) –约束可行性整数性检查精度（**可选**，默认值为 10^{-8}）。

返回

解对模型是否可行。

返回类型

bool

返回值

- **false** –解对模型不可行。
- **true** –解对模型是可行的。

示例

```
result = model.CheckSolution(solution)
```

7.17.17 ComputeIIS() 不可约不一致子系统

计算不可约不一致子系统（IIS），关于 IIS 更多细节请参考*不可约不一致子系统 (IIS) 快速入门*。

OPTVModel.**ComputeIIS**()

返回

IIS 是否计算成功。

返回类型

bool

返回值

- **false** –IIS 计算不成功。
- **true** –成功计算 IIS。

示例

```
if model.ComputeIIS():
    model.WriteIIS("iis.ilp")
else:
    print("IIS computation failed")
```

7.17.18 FeasRelax() 不可行问题的诊断及分析

进行不可行问题的诊断及分析，更多细节请参考*LP 不可行诊断及修复快速入门*。

OPTVModel.**FeasRelax**()

该函数进行不可行问题的诊断及分析，目前仅支持 L^1 范数。

返回
不可行问题修复的状态。

返回类型
int

返回值

- **0** –当前问题是可行的。
- **-1** –不可行问题修复失败。
- **>0** –Phase 1 最小违背量的目标函数值。

示例

```
result = model.FeasRelax()
```

7.17.19 GetCoef() 查询变量系数

查询给定约束中指定变量的系数。

OPTVModel.**GetCoef**(*con*, *var*)

参数

- **con** (OPTVConstr) –需要查询的约束。
- **var** (OPTVVar) –需要查询的变量。

返回
查询变量在该约束中的系数。

返回类型
float

抛出

- ***OPTVErrCode.INVALID_ARGUMENT*** –约束或变量不存在。
- ***OPTVErrCode.DATA_NOT_AVAILABLE*** –数据不存在，或约束的线性表达式部分不可用。

示例

```
c0 = model.AddConstr(x+2*y, 1, 10, "c0")
# 获取变量y在约束c0中的系数
result = model.GetCoef(c0, y)
```

7.17.20 GetColumn() 获取列

获取指定变量对应的列对象。

OPTVModel.**GetColumn**(*v*)

参数
v –需要查询的变量。

返回
指定变量对应的列对象。

返回类型
OPTVColumn

抛出

- ***OPTVErrCode.DATA_NOT_AVAILABLE*** –该变量不存在于当前模型。
- ***OPTVErrCode.INDEX_OUT_OF_RANGE*** –变量索引值*Index* 超过了模型的变量总数。若当前模型需要执行更新，或者当前变量属于其它模型时，该错会被抛出。

示例

```
result = model.GetColumn(x)
```

7.17.21 GetConstrByName() 通过名称访问约束

通过名称获取约束。

OPTVModel.**GetConstrByName**(*name*)

参数
name (*str*) –需要访问的约束名称。

返回
名称匹配 name 的约束。

返回类型
OPTVConstr

抛出
OPTVErrCode.DATA_NOT_AVAILABLE –给定的名称为空，或模型不存在命名为 name 的变量。

示例

```
# 获取名为c0的约束
result = model.GetConstrByName("c0")
```

7.17.22 GetConstr() 获取单个约束

根据指定索引值获得模型中对应的约束。

OPTVModel.**GetConstr**(*i*)

参数
i (*int*) –指定的索引值。

返回
对应的约束。

返回类型
OPTVConstr

抛出
OPTVErrCode.INDEX_OUT_OF_RANGE –指定的索引值超出了模型的约束总数。

示例

```
result = model.GetConstr(1)
```

7.17.23 GetConstrs() 获取所有约束

获得模型中的所有约束。

OPTVModel.**GetConstrs**()

返回
包含模型中所有约束的列表。

返回类型
list[*OPTVConstr*]

示例

```
result = model.GetConstrs()
c0 = result[0]
```

7.17.24 GetGenConstrAnd() 获取单个 AND 约束

获得模型中特定的 AND 一般约束数据。

OPTVModel.**GetGenConstrAnd**(*genConstr*)

参数
genConstr (OPTVGenConstr) –需要查询的一般约束。

返回
元组，索引值 0 对应该一般约束对应的结果变量，索引值 1 对应参与检验该一般约束的所有变量的列表。

返回类型
(*OPTVVar*, list[*OPTVVar*])

抛出
OPTVErrCode.NOT_IN_MODEL –模型中不存在该一般约束，当该一般约束被删除或者由默认的构造函数创建时，会抛出此错误。

示例

```
c0 = model.AddGenConstrAnd(x1, [x2, x3, x4], "and")
model.Update()
resvar, vars_list = model.GetGenConstrAnd(c0)
```

7.17.25 GetGenConstrOr() 获取单个 OR 约束

获得模型中特定的 OR 一般约束数据。

OPTVModel.**GetGenConstrOr**(*genConstr*)

参数

genConstr (OPTVGenConstr) –需要查询的一般约束。

返回

元组，索引值 0 对应该一般约束对应的结果变量，索引值 1 对应参与检验该一般约束的所有变量的列表。

返回类型

(*OPTVVar*, list[*OPTVVar*])

抛出

OPTVErrCode.NOT_IN_MODEL –模型中不存在该一般约束，当该一般约束已从模型中被删除或者是由默认的构造函数创建时，会抛出此错误。

示例

```
c0 = model.AddGenConstrAnd(x3, [x1, x2], "or")
model.Update()
resvar, vars_list = mdoel.GetGenConstrOr(c0)
```

7.17.26 GetGenConstrIndicator() 获取单个 INDICATOR 约束

获得模型中特定的 INDICATOR 一般约束数据。

OPTVModel.**GetGenConstrIndicator**(*genConstr*)

参数

genConstr (OPTVGenConstr) –需要查询的一般约束。

返回

元组，索引值 0 对应二进制指示变量, 索引值 1 对应要求线性约束必须满足的二进制指示变量的值, 索引值 2 对应参与检验的线性约束表达式, 索引值 3 对应线性约束的含义, 索引值 4 对应参与检验的线性约束右端值。

返回类型

(*OPTVVar*, float, *OPTVLinExpr*, str, float)

抛出

OPTVErrCode.NOT_IN_MODEL –模型中不存在该一般约束，当该一般约束已从模型中被删除或者是由默认的构造函数创建时，会抛出此错误。

示例

```
# x3 = 1 -> x2+x1+1 <= 10
c0 = model.AddGenConstrIndicator(x3, 1, x2+x1+1, OPTV_SENSE_LESS, 10,␣
↪name="indicator")
model.Update()
binVar, binVal, expr, sense, rhs = mdoel.GetGenConstrIndicator(c0)
```

7.17.27 GetLinMatrix() 获取线性约束矩阵

获得模型线性约束的矩阵。

OPTVModel.**GetLinMatrix**()

该函数用于获得模型线性约束的 CSR 形式的矩阵。

返回
模型线性约束的 CSR 形式的矩阵。

返回类型
scipy.sparse.csr_matrix

示例

```
result = model.GetLinMatrix()
```

7.17.28 GetObjective() 获取线性目标函数

获取线性目标函数。

OPTVModel.**GetObjective**()

返回
目标函数的线性表达式。

返回类型
OPTVLinExpr

示例

```
result = model.GetObjective()
```

7.17.29 GetQObjective() 获取二次目标函数

获取二次目标函数。

OPTVModel.**GetQObjective**()

返回
目标函数的二次表达式。

返回类型
OPTVQuadExpr

示例

```
result = model.GetQObjective()
```

7.17.30 GetQConstr() 获取二次约束

根据指定索引值获得模型中对应的二次约束。

OPTVModel.**GetQConstr**(*i*)

参数
i (*int*) –指定的索引值。

返回
对应的二次约束。

返回类型
OPTVQConstr

抛出
OPTVErrCode.INDEX_OUT_OF_RANGE –Index is larger than number of constraints in the model.

示例

```
result = model.GetQConstr(1)
```

此函数中的参数 *i* 是指全局下标，与约束的类型无关。例如，用户依次添加了一个线性约束和一个二次约束，那么该线性约束的索引值为 *0* 而二次约束的索引值为 *1*.

7.17.31 GetQConstrs() 获取所有二次约束

获得模型中的所有二次约束。

OPTVModel.**GetQConstrs**()

返回
包含模型中所有二次约束的列表。

返回类型
list[*OPTVQConstr*]

示例

```
result = model.GetQConstrs()
qc0 = result[0]
```

7.17.32 GetSOS() 获取单个 SOS 约束

根据索引值获取指定类型的 SOS 约束。

OPTVModel.**GetSOS**(*i*, *type*)

参数

- **i** (*int*) –指定 SOS 约束在模型中的索引值。
- **type** (*int*) –SOS 类型，取值为 1 或 2.

返回
对应索引值 i 的指定类型的 SOS 约束。

返回类型
OPTVSOS

抛出

- ***OPTVErrCode.INDEX_OUT_OF_RANGE*** –索引值 `i` 超出了模型的该类型的 SOS 约束总数。
- ***OPTVErrCode.INVALID_ARGUMENT*** –`type` 取值不是 `1` 或 `2`。

示例

```
c0 = model.AddSOS([x, y, z], [1, 2, 3], 1, name="sos1")
result = model.GetSOS(0, 1)
```

这里，参数 `i` 特指某一类型 SOS 约束集中的索引值，例如，如果用户先后添加了类型为 `1` 和 `2` 的两个 SOS 约束，那么两者的索引值都是 `0`.

7.17.33 GetSOSData() 获取 SOS 约束数据

OPTVModel.**GetSOSData**(*sos*, *sosVars*, *weights*)

获取 SOS 约束的内部数据。

参数

- **sos** (OPTVSOS) –当前访问的 SOS 约束对象。
- **sosVars** (*list[OPTVVar]*) –变量序列的空向量。如果非空，该容器在本次函数调用中会被清空。
- **weights** (*list[float]*) –变量权重序列的空向量。如果非空，该容器在本次函数调用中会被清空。

返回
SOS 涉及的变量及权重的元组。

返回类型
Tuple[*OPTVVar*, float]

抛出

- ***OPTVErrCode.DATA_NOT_AVAILABLE*** –SOS 约束是由默认构造函数生成的。
- ***OPTVErrCode.NOT_IN_MODEL*** –SOS 约束属于另外一个模型，或已经从本模型中被删除。
- ***OPTVErrCode.INDEX_OUT_OF_RANGE*** –约束下表越界、超出容器范围，可能需要对当前模型执行更新。

示例

```
c0 = model.AddSOS([x, y, z], [1, 2, 3], 1, name="sos1")
(var, weight) = model.GetSOSData(c0, [x, y, z], [1, 2, 3])[0]
```

7.17.34 GetSOSs() 获取多个 SOS 约束

获取指定类型的所有 SOS 约束。

OPTVModel.**GetSOSs**(*type*)

参数
type (*int*) –SOS 类型，取值为 1 或 2.

返回
包含模型中所有类型为 type 的 SOS 约束的列表。

返回类型
list[*OPTVSOS*]

抛出

- ***OPTVErrCode.INDEX_OUT_OF_RANGE*** –SOS 约束索引值存在越界、超出了模型的该类型的 SOS 约束总数，可能需要对当前模型执行更新。
- ***OPTVErrCode.INVALID_ARGUMENT*** –type 取值不是 1 或 2。

示例

```
result = model.GetSOSs(1)
```

7.17.35 GetRow() 获取行

获取指定约束对应的线性函数表达式。

OPTVModel.**GetRow**(*c*)

参数
c (OPTVConstr) –指定的约束对象。

返回
该约束对应的线性表达式。

返回类型
OPTVLinExpr

抛出

- ***OPTVErrCode.DATA_NOT_AVAILABLE*** –该约束不存在于当前模型。
- ***OPTVErrCode.INDEX_OUT_OF_RANGE*** –*Index* of the constraint is larger than the number of constraints in model. This may occur if the constraint needs to be flushed to the model. This exception may also be thrown if the constraint is from another model.

示例

```
constrs = model.GetConstrs()
linexpr = model.GetRow(constrs[0])
```

7.17.36 Get() 获取参数和属性

获取参数

获取求解器的 param 取值，可用 param 请参考*OptVerseparam*。

OPTVModel.**Get**(*param*)

参数
param (OPTVIntParam / OPTVDblParam / OPTVStrParam) –需要查询的 param，包括整数型、双精度型、字符串型

返回
param 取值。

返回类型
int | float | str

抛出
OPTVErrCode.UNKNOWN_PARAMETER –未知 param。

示例

```
result1 = model.Get(OPTVIntParam.CUTS)
result2 = model.Get(OPTVDblParam.GAP)
result3 = model.Get(OPTVStrParam.LOG_FILE)
```

获取属性

获取属性的取值，可用属性请参考*OptVerse 属性*。

OPTVModel.**Get**(*attr*)

参数
attr (OPTVIntAttr / OPTVDblAttr / OPTVLongAttr) –需要查询的属性，包括整数型、双精度型

返回
属性取值。

返回类型
int | float

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知属性。

示例

```
result1 = model.Get(OPTVIntAttr.NUM_CONSTRS)
result2 = model.Get(OPTVDblAttr.MIP_GAP)
result3 = model.Get(OPTVLongAttr.FINGERPRINT)
```

7.17.37 GetVarByName() 通过名称访问变量

通过名称获取变量。

OPTVModel.**GetVarByName**(*name*)

参数
name (*str*) –需要访问的变量名称。

返回
名称匹配 name 的变量。

返回类型
OPTVVar

抛出
OPTVErrCode.DATA_NOT_AVAILABLE –给定的名称为空，或模型不存在命名为 name 的变量。

示例

```
var = model.GetVarByName("x")
```

7.17.38 GetVar() 获取单个变量

根据指定索引值获得模型中对应的变量。

OPTVModel.**GetVar**(*i*)

参数
i (*int*) –指定的索引值。

返回
对应的变量。

返回类型
OPTVVar

抛出
OPTVErrCode.INDEX_OUT_OF_RANGE –指定的索引值超出了模型的变量总数。

示例

```
result = model.GetVar(0)
```

7.17.39 GetVars() 获取所有变量

获得模型中的所有变量。

OPTVModel.**GetVars**()

返回
包含模型中所有变量的向量。

返回类型
list[*OPTVVar*]

示例

```
result = model.GetVars()
x = result[0]
```

7.17.40 Optimize() 求解模型

对模型进行求解。

OPTVModel.**Optimize**()

返回
无。

返回类型
None

抛出
OPTVErrCode.MATRIX_NOT_PSD–模型中包含二次目标函数或者二次约束，而该表达式的矩阵不满足正半定性质。当求解的模型 **包含**二次约束时，QCQP 求解器要求模型的二次目标函数或二次约束的矩阵都是正半定的。

示例

```
model.Optimize()
```

求解器采用哪种优化算法取决于求解器参数配置和模型属性，例如变量类型被用以自动识别 LP, QP, MIP 及网络流问题。

7.17.41 Presolve() 预处理

对模型进行预处理。

OPTVModel.**Presolve**()

该函数会应用预处理流程，返回一个新的*OPTVModel* 对象。原模型的状态被修改为 被预处理.

返回
预处理后的模型。

返回类型
OPTVModel

示例

```
result = mdoel.Presolve()
```

7.17.42 Read() 读取模型和基

从文件中读取读取模型和基。

OPTVModel.**Read**(*fileName*)

该函数供读取基信息，也可以读取模型文件，然而建议由构造函数读取文件中的模型，参考文件格式。

表 13: 支持读入的文件格式

.mps	.lp	.bas

表 14: 支持读入的压缩格式

.gz	.z	.Z

参数
fileName (*str*) –C++ 字符串文件名。

返回
无。

返回类型
None

抛出
OPTVErrCode.ERROR_FILE_READ –文件名或后缀为空，文件存在格式错误。

示例

```
model.Read("input.mps")
model.Read("input.gz")
```

7.17.43 Remove() 删除模型组件

从模型中删除变量。

OPTVModel.**Remove**(*var*)

该函数将变量从模型中删除，对应的索引会被相应地调整。例如，若下列模型 model 构建时包含以下变量

变量	索引
x0	0
x1	1
x2	2
x3	3

调用 model.Remove(x0) 后，相应变化为

变量	索引
x1	0
x2	1
x3	2

参数
var (OPTVVar) –指定要删除的变量。

返回
无。

返回类型
None

抛出

- ***OPTVErrCode.NOT_IN_MODEL*** –模型不包含此变量。

- ***OPTVErrCode.DATA_NOT_AVAILABLE*** –该变量属于其他模型，而不属于当前模型。
- ***OPTVErrCode.INDEX_OUT_OF_RANGE*** –变量的下标超出了模型的变量总数，可能需要执行模型更新。

从模型中删除约束。

OPTVModel.**Remove**(*constr*)

该函数将约束从模型中删除，对应的索引会被相应地调整。例如，若下列模型 model 构建时包含以下约束

约束	索引
c0	0
c1	1
c2	2
c3	3

调用 model.Remove(c0) 后，相应变化为

约束	索引
c1	0
c2	1
c3	2

参数

constr(OPTVConstr)–指定要删除的约束。

返回

无。

返回类型

None

抛出

- ***OPTVErrCode.NOT_IN_MODEL*** –模型不包含此约束。
- ***OPTVErrCode.DATA_NOT_AVAILABLE*** –该约束属于其他模型，而不属于当前模型。
- ***OPTVErrCode.INDEX_OUT_OF_RANGE*** –约束的下标超出了模型的变量总数，可能需要执行模型更新。

OPTVModel.**Remove**(*genConstr*)

从模型中删除单个一般约束。执行后对应的相同类型的一般约束索引会被相应地调整。例如，若下列模型 model 构建时包含以下一般约束：

一般约束	类型	索引
and0	AND	0
and1	AND	1
or0	OR	0
or1	OR	1

调用 `model.Remove(and0)` 后，相应变化为

一般约束	类型	索引
and1	AND	0
or0	OR	0
or1	OR	1

参数
genConstr (OPTVGenConstr) –指定要删除的一般约束。

返回
无。

返回类型
None

抛出

- ***OPTVErrCode.NOT_IN_MODEL*** –模型不包含此一般约束。
- ***OPTVErrCode.DATA_NOT_AVAILABLE*** –该一般约束属于其他模型，而不属于当前模型。
- ***OPTVErrCode.INDEX_OUT_OF_RANGE*** –该一般约束的下标超出了模型的变量总数，可能需要执行模型更新。

OPTVModel.**Remove**(*sos*)

从模型中删除单个 SOS 约束。与其他约束类型不同的是，删除该 SOS 约束后，相同类型的约束的索引**不会**改变。例如，若模型 `model` 创建时包含如下约束：

约束	类型	索引
sos1_1	1	0
sos1_2	1	1
sos1_3	1	2
sos2_1	2	0

调用 `model.Remove(sos1_2)` 后，相应变化为

约束	类型	索引
sos1_1	1	0
sos1_3	1	2
sos2_1	2	0

参数
sos (OPTVSOS) –指定要删除的 SOS 约束。

返回
无。

返回类型
None

抛出

- ***OPTVErrCode.NOT_IN_MODEL*** –SOS 约束不属于任何模型。
- ***OPTVErrCode.DATA_NOT_AVAILABLE*** –SOS 不属于当前模型。
- ***OPTVErrCode.INDEX_OUT_OF_RANGE*** –SOS 约束的索引值超出了模型的该类型的 SOS 约束总数，可能需要对当前模型执行更新。

7.17.44 Relax() 松弛模型

进行线性规划 (LP) 松弛。

`OPTVModel.`**`Relax`**`()`

该函数将原模型中的所有整数类型变量替换为连续类型变量，生成和返回一个新的*`OPTVModel`* 对象，对原模型不做任何修改。

返回
模型的 LP 松弛副本。

返回类型
OPTVModel

示例

```
r = model.Relax()
```

7.17.45 Reset() 重置模型

将模型重置为 *未求解成功*状态。

`OPTVModel.`**`Reset`**`(`*clearAll=false*`)`

该函数会清除内部解的信息，默认情况下，包括解的值、LP 基和模型状态。`clearAll` 标志是否清除额外的模型属性值（包括 MIP 热启动值，MIP 分支优先度，LP 的禁用属性状态）。

参数
`clearAll` (*bool*) –清除额外信息（默认为 `false`）。

返回
无。

返回类型
None

示例

```
model.Reset()
```

7.17.46 GetObjSA() 目标函数敏感度分析

目标函数敏感度分析，关于目标函数敏感度分析细节，请参考*LP 敏感度分析快速入门*。

OPTVModel.**GetObjSA**(*down*, *up*, *list*)

参数

- **down** (*list[float]*) –敏感度区间左端点。
- **up** (*list[float]*) –敏感度区间右端点。
- **list** (*list[int]*) –需要进行目标函数敏感度分析的约束索引集合。

返回
敏感度分析是否成功。

返回类型
bool

返回值

- **false** –敏感度分析失败。
- **true** –敏感度分析成功。

示例

```
# 目标系数的敏感性信息
x1 = model.AddVar(-OPTV_INF, 40, -1, OPTV_CONTINUOUS, "x1")
x2 = model.AddVar(0, OPTV_INF, -2, OPTV_CONTINUOUS, "x2")
x3 = model.AddVar(0, OPTV_INF, -3, OPTV_CONTINUOUS, "x3")
model.Optimize()
cols = [0, 1, 2]
objLower = []
objUpper = []
result = model.GetObjSA(objLower, objUpper, cols)
for i in cols:
    print("x%s : [%s, %s]" % (i+1, objLower[i], objUpper[i]))
```

7.17.47 GetVarLbSA() 变量下界敏感度分析

变量下界敏感度分析，有关变量下界敏感度分析细节，请参考*LP 敏感度分析快速入门*。

OPTVModel.**GetVarLbSA**(*down*, *up*, *list*)

参数

- **down** (*list[float]*) –敏感度区间左端点。
- **up** (*list[float]*) –敏感度区间右端点。
- **list** (*list[int]*) –需要进行变量下界敏感度分析的约束索引集合。

返回
敏感度分析是否成功。

返回类型
bool

返回值

- **false** –敏感度分析失败。

- **true** –敏感度分析成功。

示例

```
# 下界敏感性信息
x1 = model.AddVar(-OPTV_INF, 40, -1, OPTV_CONTINUOUS, "x1")
x2 = model.AddVar(0, OPTV_INF, -2, OPTV_CONTINUOUS, "x2")
x3 = model.AddVar(0, OPTV_INF, -3, OPTV_CONTINUOUS, "x3")
model.Optimize()
cols = [0, 1, 2]
lbLower = []
lbUpper = []
result = model.GetVarLbSA(lbLower, lbUpper, cols)
for i in cols:
    print("x%s : [%s, %s]" % (i+1, lbLower[i], lbUpper[i]))
```

7.17.48 GetVarUbSA() 变量上界敏感度分析

变量上界敏感度分析，有关变量上界敏感度分析细节，请参考*LP 敏感度分析快速入门*。

OPTVModel.**GetVarUbSA**(*down*, *up*, *list*)

参数

- **down** (*list[float]*) –敏感度区间左端点。
- **up** (*list[float]*) –敏感度区间右端点。
- **list** (*list[int]*) –需要进行变量上界敏感度分析的约束索引集合。

返回
敏感度分析是否成功。

返回类型
bool

返回值

- **false** –敏感度分析失败。
- **true** –敏感度分析成功。

示例

```
# 上界敏感性信息
x1 = model.AddVar(-OPTV_INF, 40, -1, OPTV_CONTINUOUS, "x1")
x2 = model.AddVar(0, OPTV_INF, -2, OPTV_CONTINUOUS, "x2")
x3 = model.AddVar(0, OPTV_INF, -3, OPTV_CONTINUOUS, "x3")
model.Optimize()
cols = [0, 1, 2]
ubLower = []
ubUpper = []
result = model.GetVarUbSA(ubLower, ubUpper, cols)
for i in cols:
    print("x%s : [%s, %s]" % (i+1, ubLower[i], ubUpper[i]))
```

7.17.49 GetConstrLbSA() 约束左端值敏感度分析

约束左端值/下界敏感度分析，有关约束敏感度分析细节，请参考*LP 敏感度分析快速入门*。

OPTVModel.**GetConstrLbSA**(*down*, *up*, *list*)

参数

- **down** (*list[float]*) –敏感度区间左端点。
- **up** (*list[float]*) –敏感度区间右端点。
- **list** (*list[int]*) –需要进行左端值敏感度分析的约束索引集合。

返回
敏感度分析是否成功。

返回类型
bool

返回值

- **false** –敏感度分析失败。
- **true** –敏感度分析成功。

示例

```
# 左端项(LHS)敏感性信息
x1 = model.AddVar(-OPTV_INF, 40, -1, OPTV_CONTINUOUS, "x1")
x2 = model.AddVar(0, OPTV_INF, -2, OPTV_CONTINUOUS, "x2")
x3 = model.AddVar(0, OPTV_INF, -3, OPTV_CONTINUOUS, "x3")
model.AddConstr(-x1 + x2 + x3, -OPTV_INF, 20, "c1")
model.AddConstr(x1 - 3*x2 + x3, -OPTV_INF, 30, "c2")
model.Optimize()
rows = [0, 1]
lhsLower = []
lhsUpper = []
result = model.GetConstrLbSA(lhsLower, lhsUpper, rows)
for i in rows:
    print("c%s : [%s, %s]" % (i+1, lhsLower[i], lhsUpper[i]))
```

7.17.50 GetConstrUbSA() 约束右端值敏感度分析

约束右端值/上界敏感度分析，有关约束敏感度分析细节，请参考*LP 敏感度分析快速入门*。

OPTVModel.**GetConstrUbSA**(*down*, *up*, *list*)

参数

- **down** (*list[float]*) –敏感度区间左端点。
- **up** (*list[float]*) –敏感度区间右端点。
- **list** (*list[int]*) –需要进行右端值敏感度分析的约束索引集合。

返回
敏感度分析是否成功。

返回类型
bool

返回值

- **false** –敏感度分析失败。
- **true** –敏感度分析成功。

示例

```
# 右端项 (RHS) 敏感性信息
x1 = model.AddVar(-OPTV_INF, 40, -1, OPTV_CONTINUOUS, "x1")
x2 = model.AddVar(0, OPTV_INF, -2, OPTV_CONTINUOUS, "x2")
x3 = model.AddVar(0, OPTV_INF, -3, OPTV_CONTINUOUS, "x3")
model.AddConstr(-x1 + x2 + x3, -OPTV_INF, 20, "c1")
model.AddConstr(x1 - 3*x2 + x3, -OPTV_INF, 30, "c2")
model.Optimize()
rows = [0, 1]
rhsLower = []
rhsUpper = []
result = model.GetConstrUbSA(rhsLower, rhsUpper, rows)
for i in rows:
    print("c%s : [%s, %s]" % (i+1, rhsLower[i], rhsUpper[i]))
```

7.17.51 SetCoef() 设置系数

设置给定约束中指定变量的系数。

OPTVModel.**SetCoef**(*con*, *var*, *newValue*)

参数

- **con** (OPTVConstr) –需要修改的约束。
- **var** (OPTVVar) –需要修改系数的变量。
- **newValue** (*float*) –新的系数值。

返回

无。

返回类型

None

抛出

- ***OPTVErrCode.INVALID_ARGUMENT*** –约束或变量不存在，或数值参数中存在 NaN.
- ***OPTVErrCode.DATA_NOT_AVAILABLE*** –数据不存在，或约束的线性表达式部分不可用。

示例

```
c0 = mdoel.AddConstr(x+y, 1, 10, "c0")
# c0: 1 <= x+2*y <= 10
model.SetCoef(c0, y, 2)
```

7.17.52 SetObjective() 设置目标函数

设置线性目标函数

OPTVModel.**SetObjective**(*expr*, *sense=OPTVSense.MINIMIZE*)

该函数清除当前目标函数（相对于在其基础上进行叠加）。

参数

- **expr** (OPTVLinExpr) –需要设定的目标函数线性表达式。
- **sense** (OPTVSense) –**可选**，最小化或最大化，默认值为*OPTVSense.MINIMIZE*，参考*OPTVSense*。

返回

无。

返回类型

None

示例

```
model.SetObjective(x+y+z, OPTVSense.MINIMIZE)
model.SetObjective(x*y+z, OPTVSense.MAXIMIZE)
```

7.17.53 Set() 设置参数和属性

设置参数

设置求解器的参数取值，一旦模型被创建，该接口仅提供控制此模型专用参数的接口，直到调用*OPTVModel.Update()* 方法后，新修改才会在内部生效。可用参数请参考*OptVerse 参数*。

OPTVModel.**Set**(*param*, *value*)

参数

- **param** (OPTVIntParam/OPTVDblParam/OPTVStrParam) –需要设置的参数，包括整数型、双精度型、字符串型
- **value** (*int*/*float*/*str*) –新参数值。

返回

无。

返回类型

None

抛出

OPTVErrCode.UNKNOWN_PARAMETER –未知参数。

示例

```
model.Set(OPTVIntParam.PRESOLVE_LEVEL, 0)
model.Set(OPTVStrParam.RESULT_FILE, sol_file)
model.Set("resultFile", "example2.sol")
```

设置属性

设置求解器的属性值，同样，直到调用*OPTVModel.Update()* 方法后，新修改才会在内部生效。可用属性请参考*OptVerse 属性*。

OPTVModel.**Set**(*attr*, *value*)

参数

- **attr** (OPTVIntAttr / OPTVDblAttr) –需要设置的属性。
- **value** (*int*) –新属性值。

返回
无。

返回类型
None

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知属性。

示例

```
model.Set(OPTVIntAttr.OBJ_SENSE, OPTVSense.MAXIMIZE)
```

7.17.54 Update() 更新模型

执行模型更新。

OPTVModel.**Update**()

该函数会进行模型更新，或 lazy 构建（如必要）。只有调用该函数 **后**，模型的更新才会生效。SDK 构建的模型需要调用此函数来保证内部模型的即时性。例如，用户调用*OPTVModel.Optimize()* 前 **不需要**调用此函数，因为该函数在内部已被默认执行。

返回
无。

返回类型
None

示例

```
model.Update()
```

7.17.55 WriteIIS() IIS 结果导出

将不可约不一致子系统（IIS）写入文件（.ilp 格式），有关 IIS 的细节请参考*IIS 快速入门*。

OPTVModel.**WriteIIS**(*fileName*)

参数
fileName (*str*) –写出的文件名，必须包含 .ilp 后缀（例如 iis.ilp）.

返回
无。

返回类型
None

抛出
OPTVErrCode.ERROR_FILE_WRITE –不支持的文件格式，或者未计算出 IIS.

备注： 此函数 **不会**计算 IIS，若 IIS 未被计算，本函数将抛出异常。

示例

```
if model.ComputeIIS():
    model.WriteIIS("iis.ilp")
else:
    print("IIS computation failed")
```

7.17.56 Write() 写出模型和基

将模型或基写出到文件，支持的文件格式请参考文件格式。

`OPTVModel.`**`Write`**(*fileName*)

表 15: 支持写出的文件格式

.mps （暂不支持 QP 模型写出）	.lp	.bas	.ilp

表 16: 支持写出的压缩格式

.gz	.z	.Z

参数
fileName (*str*) –C++ 字符串文件名。

返回
无。

返回类型
None

抛出
OPTVErrCode.ERROR_FILE_WRITE –文件名或后缀为空或错误。

示例

```
model.Write("out.mps")
mdoel.Write("out.gz")
```

7.18 OptVerse 错误码

OPTVErrCode 用来产生特定的*OPTVException* 异常对象，并给出相应的详细信息。

class OPTVErrCode

OptVerse 错误码

ERROR

错误：错误信息，一般从求解器内部调用抛出。

WARNING

警告：警告信息，一般从求解器内部调用抛出。

UNKNOWN

未知：遇到未知错误。

INVALID_ARGUMENT

无效参数：传入函数的参数不满足条件，例如在设置变量类型*OPTVCharAttr.VAR_TYPE* 时，传入未定义的属性值，或者用 0 除线性表达式。

UNKNOWN_ATTRIBUTE

未知属性：访问或者设置未定义的属性，例如*OPTVConstr.Get(OPTVIntAttr.NUM_VARS)* 将抛出此错误码，因为*OPTVIntAttr.NUM_VARS* 不是模型的属性。

DATA_NOT_AVAILABLE

数据不可用：访问的属性值不可用，例如在调用*OPTVModel.Optimize()* 之前，试图获取解的信息*OPTVVar.Get(OPTVDblAttr.X)* 将会抛出此错误。

INDEX_OUT_OF_RANGE

下标越界：用于获取对象的索引值超出了允许范围，例如，当前模型仅有 2 个变量，调用*model.GetVar(10)* 会抛出此异常。

UNKNOWN_PARAMETER

未知参数：访问未定义的参数。

VALUE_OUT_OF_RANGE

数值越界：参数值超出了允许范围，例如尝试获取不存在的*OPTVSense*.

ERROR_FILE_READ

文件读取失败：读取文件时发生错误。

ERROR_FILE_WRITE

文件写出失败：写出文件时发生错误。

LICENSE_CHECK_FAILED

许可检查失败：未发现有效许可，注：许可文件 OPTV_LICENSE_FILE 需要通过环境变量设置，细节请参考 安装说明。

INVALID_MODEL_INPUT

模型输入无效：试图使用无效的模型输入信息。

NOT_IMPLEMENTED

功能不可用：试图使用的功能暂未开放。

NOT_IN_MODEL

模型组件不存在：访问的变量或约束在模型中不存在。

MATRIX_NOT_PSD

非正半定矩阵：二次表达式的矩阵不满足正半定性质。当求解的模型 **包含**二次约束时，QCQP 求解器要求模型的二次目标函数或二次约束的矩阵都是正半定的。

7.19 OptVerse 异常

class OPTVException

OptVerse 异常类。

OPTVException.**GetMessage**()

获取异常的错误信息。

返回

异常的错误信息。

返回类型

str

OPTVException.**GetErrorCode**()

获取异常的错误码

返回

异常的错误码。

返回类型

OPTVErrCode

关于*OPTVErrCode* 更多细节请参考*OptVerse Error Code*.

7.20 OptVerse 调参工具

该类用来为测例构造最佳参数组合。

class OPTVTune

7.20.1 OPTVTune()

构造函数。

OPTVTune(*targets:* OPTVModel | *List[*OPTVModel*]*, *definedSpaces: List[*OPTVTuneSpace*]*, *maxTime: int*, *timeLimit: int*, *numSuggestion: int*, *indicator: str*, *bkptFile: str*, *bkptDir: str*, *logDir: str*, *resultDir: str*)

参数

- **targets** –需要进行调参的模型
- **definedSpaces** –参数搜索空间
- **maxTime** –调参时限
- **timeLimit** –模型单次求解时间上限
- **numSuggestion** –每轮尝试的最多参数组合个数
- **inidcator** –调参指标，包括

- time: 求解时间
- gap: 相对偏差（仅适用于 MIP 问题）
- solve: 可解性（批量调参任务）

- **bkptFile** –断点文件读取路径，用以继续暂停的调参任务；若为空，则开始全新的调参任务
- **bkptDir** –断点文件保存路径，保存当前调参进程的断点；若为空，则不保存任何断点
- **logDir** –调参日志保存路径；若为空，则不保存任何日志
- **resultDir** –调参结果保存路径；若为空，则不保存任何信息

返回
无。

7.20.2 SetFixedParams() 设置固定参数

设置调参任务的固定参数

OPTVTune.**SetFixedParams**(*self*, *fixedParams: Dict*)

参数
fixedParams (*Dict*) –指定固定的参数

返回
无。

7.20.3 Run() 调参

执行调参任务

OPTVTune.**Run**()

返回
无。

7.20.4 GetBestParams() 获取最佳参数

获取最佳参数组合

OPTVTune.GetBestParams(self):

可获取最佳参数组合信息，包括

nRound: int: 获取最佳参数组合所在的调参轮数

paramID: int: 获取最佳参数组合的索引

bestParams: Dict: 获取最佳参数组合值

7.20.5 OPTVTuneSpace 调参空间

调参任务的预设参数搜索空间

class OPTVTuneSpace

- PRESOLVE: 预处理参数的搜索空间
- SEARCH: Branch 方法参数的搜索空间
- SEPARATOR: Separation 方法参数的搜索空间
- HEURISTIC: 启发式方法参数的搜索空间
- CONFLICT: 冲突分析方法参数的搜索空间
- PROPAGATION: 邻域传播方法参数的搜索空间
- LP: LP 算法参数的搜索空间

CHAPTER 8

Java API 参考手册

8.1 OptVerse 常量

OPTV_INF
代表无穷大的量的最大值。

类型
float

取值
`1.0e+100`

OPTV_UNDF
未定义的数值。

类型
float

取值
`1.0e+101`

8.1.1 变量类型

OPTV_BINARY
0-1 整数变量类型。

类型
char

取值
`'B'`

OPTV_INTEGER

整数变量类型。

类型

char

取值

'I'

OPTV_CONTINUOUS

连续变量类型。

类型

char

取值

'C'

8.1.2 模型求解状态

OPTV_UNKNOWN

未知模型求解状态（默认）。

类型

int

取值

0

OPTV_OPTIMAL

模型找到最优解，可以查询到最优解。

类型

int

取值

1

OPTV_INFEASIBLE

模型被证明无解。

类型

int

取值

2

OPTV_INF_OR_UNBD

模型被证明无解或者无界。

类型
int

取值
3

OPTV_UNBOUNDED

模型被证明无界。

类型
int

取值
4

OPTV_TIME_LIMIT

模型在指定时间*OPTVDblParam.TIME_LIMIT* 内没有完成求解。

类型
int

取值
5

OPTV_MEM_LIMIT

模型求解使用内存超过预设值*OPTVIntParam.MEM_LIMIT*.

类型
int

取值
6

OPTV_INTERRUPTED

用户终止模型求解（如通过 Ctrl+C）.

类型
int

取值
7

8.1.3 变量和约束的基状态

OPTV_BASIC

变量或约束为基。

类型
int

取值
0

OPTV_NONBASIC_LOWER

变量或约束非基，取下界值。

类型
int

取值
-1

OPTV_NONBASIC_UPPER

变量或约束非基，取上界值。

类型
int

取值
-2

OPTV_SUPERBASIC

变量或约束为超基。

类型
int

取值
-3

8.1.4 一般约束类型

OPTV_GENCONSTR_AND

只有所有变量非零时，结果才非零。

类型
int

取值
0

OPTV_GENCONSTR_OR

只要有一个变量非零，结果就非零。

类型
int

取值
1

OPTV_GENCONSTR_INDICATOR

返回值为布尔型，当且仅当相应的逻辑条件满足，取值为 1，否则取值为 0。

类型
int

取值
2

8.1.5 约束含义

OPTV_SENSE_LESS
小于或等于。

类型
char

取值
<

OPTV_SENSE_EQUAL
等于。

类型
char

取值
=

OPTV_SENSE_GREATER
大于或等于。

类型
char

取值
>

8.2 OptVerse 参数

参数对应着用户使用的求解器内部配置，可以通过*OPTVEnv*（模型构建前）或*OPTVModel*（模型构建后）来设置。

8.2.1 整数型参数

class OPTVIntParam

表 1: OptVerse 整数型参数

名称	描述
BRANCH_DIR	MIP 分支方向
CROSSOVER	内点算法 crossover 策略
CUT_PASSES	根节点 (MIP) 割平面方法最大轮数
CUTS	MIP 割平面方法强度
MEM_LIMIT	用户指定的内存使用上限 (MB)
METHOD	LP 求解算法（如单纯形、内点等）
MIP_FOCUS	MIP 高层级的解策略
OUTPUT_FLAG	禁止日志打印
OPTIMAL_BASIS	清理所得解获取最优基
POOL_SOLUTIONS	解池存储的 MIP 解个数
PRESOLVE_LEVEL	预处理程度控制
SEED	PRNG 随机数种子
SOL_FORMAT	输出解的形式（稀疏或者稠密）
SOLUTION_NUMBER	用来指定 MIP 解
START_NODE_LIMIT	MIP 分支定界节点个数上限
SUBMIP_NODES	MIP 启发式策略模块访问的节点个数上限
THREADS	线程数
VAR_BRANCH	(MIP) 变量分支方法

BRANCH_DIR 分支方向

`OPTVIntParam.`**`BRANCH_DIR`**

MIP 分支方向。

默认值: 0

最小值: -1

最大值: 1

该参数控制分支剪界搜索中子节点的选取，取值-1 表示优先尝试下行节点，取值 +1 表示优先尝试上行节点。默认值为 0，表示自动选取子节点。

CROSSOVER

`OPTVIntParam.`**`CROSSOVER`**

内点法 crossover 策略。

默认值: 2

最小值: 0

最大值: 4

crossover 将内点解（通常为可行解，但一般不是基解）转化为基本可行解，它由以下三步组成:

1) 原始推进步 - 将原始变量推至边界，
2) 对偶推进步 - 将对偶变量推至边界，
3) 清理 - 应用单纯形方法清理任何原始或对偶不可行性。

该参数用于配置前两步的步骤及清理方法的选项。

取值	第一种推进方法	第二种推进方法	不可行性清理方法
0	禁用	禁用	禁用
1	对偶	原始	原始
2 (默认)	对偶	原始	对偶
3	原始	对偶	原始
4	原始	对偶	对偶

CUT_PASSES 割平面轮数

`OPTVIntParam.`**`CUT_PASSES`**

根节点 (MIP) 割平面方法最大轮数。

默认值: 2147483647

最小值: 0

最大值: 2147483647

该参数用以控制根节点割平面方法，使得轮数不超过设定值。

CUTS 割平面方法强度

`OPTVIntParam.`**`CUTS`**

MIP 割平面方法强度。

默认值: -1

最小值: -1

最大值: 3

该参数全局地控制割平面方法强度。默认值为-1，允许求解器进行自适应的调整；取值 0 则会关闭割平面方法；取值 1 会应用轻量级的割平面方法，取值 2 对应中等强度的割平面方法，而取值 3 则应用高强度的割平面方法。

MEM_LIMIT 内存限制

`OPTVIntParam.`**`MEM_LIMIT`**

求解器使用的内存限制, 单位 MB

默认值: 2147483647

最小值: 100

最大值: 2147483647

该参数用来限制求解器运行占用的 RAM 空间不超过指定阈值，若超出该阈值，求解器将终止运行并返回状态*OPTV_MEM_LIMIT*.

METHOD 优化算法

OPTVIntParam.**METHOD**

LP 问题优化算法选择。

默认值: -1

最小值: -1

最大值: 4

该参数用于指定 LP 问题的优化方法。

取值	方法
-1	自动
0	原始单纯形方法
1	对偶单纯形方法
2	内点方法 (IPM)
3	行生成方法
4	两阶段原始单纯形方法

MIP_FOCUS 解策略

OPTVIntParam.**MIP_FOCUS**

MIP 高层级的解策略。

默认值: 0

最小值: 0

最大值: 3

该参数控制 MIP 求解器的优先度策略，默认为均衡式，具体信息如下：

取值	方法
0	均衡式
1	优先快速找到可行解
2	优先证明最优性
3	优先改善下界

OPTIMAL_BASIS 最优基

OPTVIntParam.**OPTIMAL_BASIS**

求解结束时是否同时需要最优基。

默认值: 1 (开启)

最小值: 0 (关闭)

最大值: 1 (开启)

该高级参数允许用户获取可行解后跳过基状态清理过程，若不关心所得的可行解是基解与否，可关闭该选项。关闭该选项后，求解器不会执行基状态的清理操作。

警告: 若关闭该选项，所得解有基状态不正确的风险，而且这个参数会影响热启动的效果。仅适用于查询问题是否可行的情景，如果用户需要获取变量或约束的取值 (*var.get(OPTVDblAttr.X)* 或 *constr.get(OPTVDblAttr.X)*)，则需要 **开启**该选项 (默认环境下为开启状态)。

OUTPUT_FLAG 日志打印控制

OPTVIntParam.**OUTPUT_FLAG**

控制打印日志的详细程度。

默认值: 0

最小值: 0

最大值: 1

如果开启该选项，所得日志（屏幕和日志文件）将被缩减和简化。

POOL_SOLUTIONS 解池容量

OPTVIntParam.**POOL_SOLUTIONS**

MIP 解池的最大存储（解）数量。

默认值: 10

最小值: 1

最大值: 1000000

该参数用于设置 MIP 解池存储的解的数量上限。

PRESOLVE_LEVEL 预处理控制

OPTVIntParam.**PRESOLVE_LEVEL**

控制预处理的程度。

默认值: -1

最小值: -1

最大值: 3

用户通过该参数控制预处理的程度。

取值	程度
-1	自动
0	关闭预处理
1	保守倾向
2	中等
3	进取倾向（耗时多，处理后的模型更紧凑）

SEED 随机数种子

OPTVIntParam.**SEED**

PRNG 初始随机数种子

默认值: 0

最小值: 0

最大值: 2147483647

该参数用来设定随机数生成器的初始值。

SOL_FORMAT 解文件风格控制

OPTVIntParam.**SOL_FORMAT**

解文件稀疏/稠密控制。

默认值: 0

最小值: 0

最大值: 1

通过调整该参数，用户可以控制解文件中是否包含 0 值。若设置该参数为 0，解文件将省去 0 值，为稀疏格式；否则，解文件将包含 0 值，为稠密格式。

SOLUTION_NUMBER 解索引

OPTVIntParam.**SOLUTION_NUMBER**

用于查询指定（MIP）解。

默认值: 0

最小值: 0

最大值: 1000000

用户可以通过该参数指定要查询的解，从而可获悉更多相关信息，如*OPTVDblAttr.XN* 或*OPTVDblAttr.POOL_OBJ_VAL*；取值为 0 时，上述查询信息等同于*OPTVDblAttr.X* 或*OPTVDblAttr.OBJ_VAL*.

备注: 本参数仅适用于 MIP 问题。

START_NODE_LIMIT 分支定界节点数上限

OPTVIntParam.**START_NODE_LIMIT**

限制在 MIP 问题求解启动期间访问的分支定界节点数。

默认值: -1

最小值: -3

最大值: 2147483647

该参数用于控制 MIP 启动期间访问的分支定界节点数。默认选项允许求解器自动确定 MIP 启动期间计算节点的最大数目；取值为-2 时，禁用 MIP 部分启动评估而采用全部启动；取值为-3 将完全禁用 MIP 启动计算。

备注： 本参数仅适用于 MIP 问题。

SUBMIP_NODES 启发式节点数上限

`OPTVIntParam.`**`SUBMIP_NODES`**

MIP 启发式策略模块访问的节点个数上限。

默认值: 500

最小值: 0

最大值: 2147483647

该参数用以限制 MIP 启发式策略访问的节点个数，访问节点越多，越有可能产生更高质量的解，但通常也耗时更久。

备注： 本参数仅适用于 MIP 问题。

THREADS 线程数

`OPTVIntParam.`**`THREADS`**

求解器运行期间使用最大线程数。

默认值: 1

最小值: 1

最大值: 虚拟机可用的最大线程总数

VAR_BRANCH 变量分支方法

`OPTVIntParam.`**`VAR_BRANCH`**

(MIP) 变量分支方法

默认值: -1

最小值: -1

最大值:: 2

该参数控制变量分支策略，具体信息如下：

取值	方法
0	自动
0	Reliability 分支方法
1	Strong branching 方法
2	Pseudo cost branching 方法

8.2.2 双精度型参数

class OPTVDblParam

表 2: OptVerse 双精度型参数

名称	描述
FEAS_TOL	原始问题约束可行性精度
HEURISTICS	MIP 启发式策略模块占时比例
INT_TOL	整数性精度
MARKOWITZ_TOL	用于限制选择矩阵分解主元时的数值误差
GAP	MIP 相对偏差
TIME_LIMIT	最大运行时间（秒）

FEAS_TOL 约束可行性精度

OPTVDblParam.**FEAS_TOL**

原始问题约束可行性精度。

默认值: 10^{-6}

最小值: 10^{-9}

最大值: 0.01

若该参数取较小的值，所得解的约束违背量则越小，但是迭代步数可能更多。

HEURISTICS 启发式模块

OPTVDblParam.**HEURISTICS**

MIP 启发式策略模块占总求解时间的比例。

默认值: 0.05

最小值: 0

最大值: 1

该参数控制 MIP 求解器启发式策略模块的占用时间，以占时比例为测度，默认值为 0.05, 即占总时耗的 5%. 较高的该参数值可以使 MIP 求解器找到更多或更好的可行解，但是目标值的最佳边界进度也较慢。

INT_TOL 整数性精度

OPTVDblParam.**INT_TOL**

整数变量的整数性精度。

默认值: 10^{-6}

最小值: 10^{-9}

最大值: 0.01

若该参数取较小的值，所得解的整数性违背量则越小，但是求解性能（时耗）有可能被影响；取值较大一般不影响性能（时耗）。

MARKOWITZ_TOL 矩阵分解精度

`OPTVDblParam.`**`MARKOWITZ_TOL`**

适用于单纯形方法，用于限制选择矩阵分解主元时的数值误差。

默认值: 0.1

最小值: 10^{-4}

最大值: 0.999

取值较大将得到更好的数值稳定性，但可能影响求解性能（时耗）。

GAP 相对偏差

`OPTVDblParam.`**`GAP`**

MIP 相对偏差。

默认值: 0

最小值: 0

最大值: 10^{100}

若 f_P 和 f_D 分别为原始和对偶目标函数值，相对偏差值 g 的定义为 $g = \frac{|f_P - f_D|}{|f_P|}$. 在 MIP 求解过程中，一旦目标值的上下界相对偏差比 `gap` 小，求解器则会终止并返回状态*OPTV_OPTIMAL*.

TIME_LIMIT 求解时间上限

`OPTVDblParam.`**`TIME_LIMIT`**

模型求解时间上限（秒）。

默认值: 10^{20}

最小值: 0

最大值: 10^{20}

如果求解时间超出该数值，求解器则会终止并返回状态*OPTV_TIME_LIMIT*.

注意求解计时受制于单步迭代计时。对于耗时较久的优化问题，建议增加额外计时机制，如 `timeout`.

8.2.3 字符串型参数

`class OPTVStrParam`

表 3: OptVerse 字符串型参数

名称	描述
LOG_FILE	日志文件路径
PARAM_FILE	参数文件路径
REPLAY_FILE	SDK 命令回放文件路径
RESULT_FILE	结果文件路径

LOG_FILE 日志文件路径

OPTVStrParam.**LOG_FILE**

日志文件路径。

默认值: "./optv.log"

如果*OPTVIntParam.OUTPUT_FLAG* 被设置为 0，则本参数将被忽略。

PARAM_FILE 参数文件路径

OPTVStrParam.**PARAM_FILE**

参数文件路径。

默认值: ""

用于为求解器指定参数文件，默认值为空。

REPLAY_FILE SDK 命令回放文件

OPTVStrParam.**REPLAY_FILE**

SDK 命令回放文件路径。

默认值: ""

用于指定 SDK 命令回放文件存放的路径，若为空，则不创建命令回放文件。

备注: 该参数 **必须**在*OPTVModel* 对象创建前，通过*OPTVEnv* 对象设置。

RESULT_FILE 结果文件路径

OPTVStrParam.**RESULT_FILE**

结果文件路径。

默认值: "./<input_file>.sol"

备注: 通过 SDK 构建模型时，默认不进行输出结果文件，**必须**通过该参数指定生成解文件。

8.3 OptVerse 属性

属性对应着模型相关的各种数据，从属于模型组件对象 (*OPTVModel*, *OPTVVar*, *OPTVConstr*). 例如，任意给定变量或约束的下界可以通过*OPTVDblAttr.LB* 来获取。

备注: 有若干属性只能在*OPTVModel.optimize()* 调用后才可以获取，如*OPTVDblAttr.X*, *OPTVDblAttr.XN*, *OPTVDblAttr.DUAL*, *OPTVDblAttr.OBJ_VAL*, *OPTVDblAttr.OBJ_BOUND*, 和*OPTVDblAttr.MIP_GAP*.

8.3.1 布尔属性

class OPTVBoolAttr

表 4: OptVerse 布尔型属性

名称	描述
IIS_CONSTR	指示该约束是否属于所得的 IIS

IIS_CONSTR 约束是否属于 IIS

OPTVBoolAttr.**IIS_CONSTR**

约束是否属于 IIS.

是否可修改: 否

表征指定约束是否属于计算所得的不可约不一致子系统 (IIS)；如果没有计算出 IIS，那么该属性不可获取。

8.3.2 整数型属性

class OPTVIntAttr

表 5: OptVerse 整数型属性

名称	描述
BASIS	变量或约束的基状态
BRANCH_PRIORITY	MIP 求解过程中变量的 branch(分支) 优先度
GENCONSTR_TYPE	一般约束类型
NUM_CONSTRS	约束总数
NUM_LIN_CONSTRS	线性约束总数
NUM_QUAD_CONSTRS	二次约束总数
NUM_INT_VARS	整数变量总数
NUM_VARS	变量总数
OBJ_SENSE	目标含义（最小化或最大化）
PROHIBITED	变量或约束是否被预处理屏蔽
SOL_COUNT	存储解的个数
SOS_TYPE	SOS 约束类型，1 或 2
STATUS	求解结果

BASIS 变量或约束的基状态

是否可修改: 是

变量或约束的基状态。

名称	取值	描述
OPTV_BASIC	0	基变量
OPTV_NONBASIC_LOWER	-1	非基，在下界
OPTV_NONBASIC_UPPER	-2	非基，在上界
OPTV_SUPERBASIC	-3	超基变量

BRANCH_PRIORITY 分支优先度

OPTVIntAttr.**BRANCH_PRIORITY**

是否可修改: 是

默认值: 0

该数值仅适用于 MIP 问题求解，用于引导 MIP 搜索过程的分支环节，以决定在哪些取值非整数的整数变量处进行分支，较大的取值表示较高的分支优先度。默认值为 0。如果存在优先度相同的情况，则采用原始的分支决策。

GENCONSTR_TYPE 一般约束类型

OPTVIntAttr.**GENCONSTR_TYPE**

一般约束类型。

是否可修改: 否

表 6: OptVerse 一般约束类型

名称	取值	描述
OPTV_GENCONSTR_AND	0	只有所有变量非零时，结果才非零
OPTV_GENCONSTR_OR	1	只要有一个变量非零，结果就非零
OPTV_GENCONSTR_INDICA	2	返回值为布尔型，当且仅当相应的逻辑条件满足，取值为 1，否则取值为 0

NUM_CONSTRS 约束总数

OPTVIntAttr.**NUM_CONSTRS**

是否可修改: 否

约束总数。

NUM_LIN_CONSTRS 线性约束总数

OPTVIntAttr.**NUM_LIN_CONSTRS**

是否可修改: 否

线性约束总数。

NUM_QUAD_CONSTRS 二次约束总数

OPTVIntAttr.**NUM_QUAD_CONSTRS**

是否可修改: 否

二次约束总数。

NUM_INT_VARS 整数变量总数

OPTVIntAttr.**NUM_INT_VARS**

是否可修改: 否

整数变量总数，包括 0/1 变量和普通整数变量。

NUM_VARS 变量总数

OPTVIntAttr.**NUM_VARS**

是否可修改: 否

变量总数。

OBJ_SENSE 目标含义

OPTVIntAttr.**OBJ_SENSE**

是否可修改: 是

优化问题目标含义，取值为-1 代表最大化问题，取值为 1 代表最小化问题。参考*OPTVSense*.

PROHIBITED 预处理屏蔽

OPTVIntAttr.**PROHIBITED**

是否可修改: 是

用于指示变量或约束是否被 LP 预处理屏蔽，取值包括 0（未屏蔽）和 1（屏蔽）。该参数不适用于 MIP 问题。

SOL_COUNT 存储解的个数

OPTVIntAttr.**SOL_COUNT**

是否可修改: 否

存储解的个数：对于 LP 问题，若获得最优解则取值为 1，否则取 0；对于 MIP 问题，该属性对应找到的解（包含最优和次最优）的个数。

SOS_TYPE SOS 类型

OPTVIntAttr.**SOS_TYPE**

是否可修改: 否

SOS 类型，可取值为 1 或 2.

STATUS 求解结果

OPTVIntAttr.**STATUS**

是否可修改: 否

模型求解状态。

名称	取值	描述
OPTV_OPTIMAL	1	模型求解成功（最优解）
OPTV_INFEASIBLE	2	模型被证明不可行
OPTV_INF_OR_UNBD	3	模型被证明不可行或无界
OPTV_UNBOUNDED	4	模型被证明无界
OPTV_TIME_LIMIT	5	求解时间超出给定时限*OPTVDblParam.TIME_LIMIT*.
OPTV_MEM_LIMIT	6	求解内存占用超过给定阈值*OPTVIntParam.MEM_LIMIT*.
OPTV_INTERRUPTED	7	用户自行终止（如通过 Ctrl+C）.
OPTV_UNKNOWN	0	未知状态。

8.3.3 长整型属性

class OPTVLongAttr

表 7: OptVerse 长整型属性

名称	描述
FINGERPRINT	模型指纹

FINGERPRINT （模型）指纹

OPTVLongAttr.**FINGERPRINT**

模型指纹

是否可修改: 否

该参数对应由模型内部数据和属性决定的 hash 值，不同的模型（包括变量或约束添加的顺序）应具有不同的模型指纹。

8.3.4 字符型属性

class OPTVCharAttr

表 8: OptVerse 字符串型属性

名称	描述
VAR_TYPE	变量类型

VAR_TYPE 变量类型

OPTVCharAttr.**VAR_TYPE**

变量类型。

是否可修改: 是

表 9: OptVerse 变量类型

名称	取值	描述
OPTV_CONTINUOUS	'C'	连续型
OPTV_INTEGER	'I'	整数型
OPTV_BINARY	'B'	0/1 型

连续变量可取给定界限内的任意浮点数值；整数变量取值限定在给定界限内的所有整数；0/1 变量只可取 0 或者 1。

8.3.5 双精度型属性

class OPTVDblAttr

表 10: OptVerse 双精度型属性

名称	描述
DUAL	变量或约束的对偶解值。
LB	变量或约束的下界值。
MIP_GAP	MIP 相对偏差。
OBJ	变量的目标值系数。
OBJ_BOUND	目标值的最佳界限。
OBJ_OFFSET	目标函数的偏移量。
OBJ_VAL	原始模型目标值。
POOL_OBJ_VAL	第 N 个解的原始模型目标值。
RUN_TIME	运行时间。
START	MIP 热启动值。
UB	变量或约束的上界值。
X	变量或约束的原始解值。
XN	第 N 个解的变量或约束的原始解值。

DUAL 对偶解值

`OPTVDblAttr.`**`DUAL`**

是否可修改: 否

变量或约束的对偶解值。

LB 下界值

`OPTVDblAttr.`**`LB`**

是否可修改: 是

变量或约束的下界值。

备注: 对于 0/1 变量，该属性只允许取值为 `0` 或 `1`。若赋值超出上述允许值 10^{-6}, 该操作会被禁止且相应的警告信息会被抛出。

MIP_GAP MIP 相对偏差

`OPTVDblAttr.`**`MIP_GAP`**

是否可修改: 否

MIP 相对偏差，仅适用于 MIP 求解范畴，具体定义可参考相对偏差。

OBJ 目标值系数

OPTVDblAttr.**OBJ**

是否可修改: 是

变量的目标值系数。

OBJ_BOUND 目标值的最佳界限

OPTVDblAttr.**OBJ_BOUND**

是否可修改: 否

目标值的最佳界限，仅适用于 MIP 求解范畴。

OBJ_OFFSET 目标函数偏移量

OPTVDblAttr.**OBJ_OFFSET**

是否可修改: 是

目标函数中的常数项。

OBJ_VAL 目标值

OPTVDblAttr.**OBJ_VAL**

是否可修改: 否

模型的原始解对应的目标值。

POOL_OBJ_VAL MIP 解的原始模型目标值

OPTVDblAttr.**POOL_OBJ_VAL**

是否可修改: 否

该属性仅适用于 MIP 求解范畴，当通过*OPTVIntParam.SOLUTION_NUMBER* 指定第 N 个 MIP 解时，*OPTVDblAttr.OBJ_VAL* 返回该解对应的原始模型目标值。

RUN_TIME 运行时间

OPTVDblAttr.**RUN_TIME**

是否可修改: 否

最近一次求解任务的执行时间。

START MIP 热启动值

`OPTVDblAttr.START`

是否可修改: 是

MIP 热启动值，该属性仅适用于 MIP 求解范畴，默认为*OPTV_UNDF*.

UB 上界值

`OPTVDblAttr.UB`

是否可修改: 是

变量或约束的上界值。

备注: 对于 0/1 变量，该属性只允许取值为 `0` 或 `1`. 若赋值超出上述允许值 10^{-6}, 该操作会被禁止且相应的警告信息会被抛出。

X 原始解值

`OPTVDblAttr.X`

是否可修改: 否

变量或约束的原始解值。

XN MIP 解的原始解值

`OPTVDblAttr.XN`

是否可修改: 否

该属性仅适用于 MIP 求解范畴。类似于*OPTVDblAttr.X*, 当通过*OPTVIntParam.SOLUTION_NUMBER* 指定第 N 个 MIP 解时，*OPTVDblAttr.XN* 返回该解的原始解值。

8.3.6 字符串型属性

class OPTVStrAttr

表 11: OptVerse 字符串型属性

名称	描述
NAME	变量或约束的名称。

NAME 变量/约束名称

OPTVStrAttr.**NAME**

变量或约束的名称，通常在建模环节根据问题定义设定。

是否可修改: 是

8.4 OptVerse 目标含义

class OPTVSense

目标含义，默认值为 1，表征最小化问题。

OPTVSense.**MINIMIZE**

最小化问题

类型
int

取值
1

OPTVSense.**MAXIMIZE**

最大化问题

类型
int

取值
-1

8.5 OptVerse 环境

class OPTVEnv

OPTVConstr.get()

OPTV 求解器环境类，用于管理 OPTV 求解器的参数配置。此外，许可文件需要通过环境变量 OPTV_LICENSE_FILE 来配置。

8.5.1 OPTVEnv() 环境类构造函数

OPTVEnv.**OPTVEnv**()

返回
环境变量，参数为默认。

返回类型
OPTVEnv

备注： 默认配置下日志文件被设定为 `optv.log`.

`OPTVEnv.`**`OPTVEnv`**(String *logFileName*)

参数

- **`logFileName`** –日志文件路径，格式为 C-风格字符串。

返回
环境变量，参数为默认，日志文件由参数指定。

返回类型
OPTVEnv

`OPTVEnv.`**`OPTVEnv`**(OPTVEnv *xenv*)

参数

- **`xenv`** –需要被复制的环境对象。

返回
环境对象 `xenv` 的副本。

返回类型
OPTVEnv

8.5.2 get() 参数查询

获取求解器的参数值，可用参数请参考*OptVerse 参数*。

`OPTVEnv.`**`get`**(OPTVIntParam *param*)

参数

- **`param`** –整数型参数。

返回
参数取值。

返回类型
int

抛出
`OPTVErrCode.UNKNOWN_PARAMETER` –未知参数。

`OPTVEnv.`**`get`**(OPTVDblParam *param*)

参数

- **`param`** –双精度型参数。

返回
参数取值。

返回类型
double

抛出
OPTVErrCode.UNKNOWN_PARAMETER –未知参数。

OPTVEnv.**get**(OPTVStrParam *param*)

参数

- **param** –字符串型参数。

返回
参数取值。

返回类型
String

抛出
OPTVErrCode.UNKNOWN_PARAMETER –未知参数。

8.5.3 set() 参数设置

通过环境对象设置求解器参数，相应修改会作为默认，被应用到后续创建的所有模型。模型建立后，只有调用*OPTVModel.update()* 才可以激活新的参数修改。具体细节可参考*OptVerse 参数*。

OPTVEnv.**set**(OPTVIntParam *param*, int *value*)

参数

- **param** –整数型参数。
- **value** –新参数值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_PARAMETER –未知参数。

OPTVEnv.**set**(OPTVDblParam *param*, double *value*)

参数

- **param** –双精度型参数。
- **value** –新参数值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_PARAMETER –未知参数。

OPTVEnv.**set** (OPTVStrParam *param*, String *value*)

参数

- **param** –字符串型参数。
- **value** –新参数值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_PARAMETER –未知参数。

8.6 OptVerse 变量

class OPTVVar

OptVerse 变量对象，用于从模型访问变量。注意变量不可由其构造函数创建，而需要通过模型的*OPTVModel.addVar()* 或*OPTVModel.addVars()* 接口添加变量到模型。

8.6.1 get() 查询变量属性

查询变量属性，关于可用属性请参考*OptVerse 属性*。

OPTVVar.**get** (OPTVIntAttr *attr*)

参数

- **attr** –整数型属性。

返回
属性值。

返回类型
int

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知变量属性。

OPTVVar.**get** (OPTVCharAttr *attr*)

参数

- **attr** –字符型属性。

返回
属性值。

返回类型
char

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知变量属性。

OPTVVar.**get**(OPTVDblAttr *attr*)

参数
- **attr** –双精度型属性。

返回
属性值。

返回类型
double

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知变量属性。

OPTVVar.**get**(OPTVStrAttr *attr*)

参数
- **attr** –字符串型属性。

返回
属性值。

返回类型
String

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知变量属性。

8.6.2 set() 设置变量属性

设置变量属性，只有调用*OPTVModel.update()* 后，新设置才可生效，更多细节请参考*OptVerse 属性*。

OPTVVar.**set**(OPTVIntAttr *attr*, int *value*)

参数
- **attr** –整数型属性。
- **value** –新属性值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知变量属性。

OPTVVar.**set** (OPTVCharAttr *attr*, int *value*)

参数

- **attr** –字符型属性。
- **value** –新属性值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知变量属性。

OPTVVar.**set** (OPTVDblAttr *attr*, int *value*)

参数

- **attr** –双精度型属性。
- **value** –新属性值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知变量属性。

OPTVVar.**set** (OPTVStrAttr *attr*, int *value*)

参数

- **attr** –字符串型属性。
- **value** –新属性值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知变量属性。

8.6.3 index() 变量索引

获得变量在模型中的索引值。

OPTVVar.**index**()

返回
变量索引/下标。

返回类型
int

返回值

- **-1** –该变量在模型中不存在。
- **-2** –该变量已从模型中删除。
- **>=0** –该变量在模型中的索引/下标。

8.6.4 sameAs() 变量对比

查询两变量是否相同。

OPTVVar.**sameAs**(OPTVVar *v2*)

参数

- **v2** –需要与当前变量对比的变量。

返回
变量是否相同。

返回类型
boolean

返回值

- **true** –变量相同。
- **false** –变量不同。

8.7 OptVerse 列

class OPTVColumn

OptVerse 列对象，用于以列的形式向模型添加变量，用户可以使用该功能向模型同时新增变量并添加其至已有约束。

在下面的例子中，我们构造一个初始模型并展示如何通过列对象增加变量，本示例展示的模型为*Quick Start* 章节的*milp1* .

```
import com.huaweicloud.optv.*;

OPTVEnv env;
OPTVModel model(env);

OPTVVar x = model.addVar(0, 1, -1, OPTV_BINARY, "x");
OPTVVar y = model.addVar(0, OPTV_INF, -14, OPTV_INTEGER, "y");
```

(续下页)

(接上页)

```
OPTVConstr c0 = model.addConstr(2 * x + y, -OPTV_INF, 3, "c0");
OPTVConstr c1 = model.addConstr(3 * y, -OPTV_INF, 6, "c1");
```

具体地，构建的模型形式如下：

$$A = \begin{bmatrix} 2 & 1 \\ 0 & 3 \end{bmatrix}$$

```
OPTVColumn col;

col.addTerm(1, c0);
col.addTerm(1, c1);

OPTVVar z = model.addVar(0, 3, -6, OPTV_CONTINUOUS, col, "z");
```

通过将 `col` 添加到模型，相应的约束矩阵变为：

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 0 & 3 & 1 \end{bmatrix}$$

8.7.1 AddTerm() 添加单项

向列对象（尾部）添加单项。

OPTVColumn.**addTerm**(double *coef*, OPTVConstr *constr*)

该函数向列对象添加单项。

参数

- **coef** –系数值。
- **constr** –约束对象。

返回

无。

返回类型

void

8.7.2 AddTerms() 增加多项

向列对象（尾部）添加多项。

OPTVColumn.**addTerms**(double[] *coefs*, OPTVConstr[] *constrs*, int *cnt*)

该功能向列对象添加多项。

参数

- **coefs** –系数序列。
- **constrs** –约束序列。

- **cnt** –添加项的个数。

返回
无。

返回类型
void

抛出
OPTVErrCode.INVALID_ARGUMENT –constrs 或 coefs 为空，或 cnt 取值为负。

8.7.3 Clear() 清空

清空与该列相关内部容器，使*OPTVColumn.size()* 返回 0.

OPTVColumn.**clear**()

返回
无。

返回类型
void

8.7.4 GetCoeff() 获取系数

据指定索引值获得列对象中对应的约束系数。

OPTVColumn.**getCoef**(int *i*)

参数

- **i** –指定的索引值，对应列中的约束。

返回
列对象中对应该索引, i, 的约束系数。

返回类型
double

抛出
OPTVErrCode.INDEX_OUT_OF_RANGE –若 $i \notin [0, this->Size()]$.

8.7.5 GetConstr() 获取单个约束

根据指定索引值获得列对象中对应的约束。

OPTVColumn.**getConstr**(int *i*)

参数

- **i** –指定的索引值，对应列中的约束。

返回
列对象中对应该索引, i, 的约束对象。

返回类型
OPTVConstr

Throw OPTVErrCode.INDEX_OUT_OF_RANGE
若 `i` $\notin [0, $ `this->Size()` $]$.

8.7.6 Remove() 删除项

从列对象中删除单/多项。

`OPTVColumn.`**`remove`**(int *i*)

根据指定索引值从该列对象中删除项。

参数

- **i** –指定的索引值。

返回
无。

返回类型
void

抛出
OPTVErrCode.INDEX_OUT_OF_RANGE –若 `i` $\notin [0, $ `this->Size()` $]$.

`OPTVColumn.`**`remove`**(OPTVConstr *constr*)

从列对象中删除指定约束的所有相关项。

参数

- **constr** –指定的约束。

返回
是否删除任何项。

返回
无。

返回类型
void

抛出
OPTVErrCode.NOT_IN_MODEL –指定的约束，`constr`, 在模型中不存在。

8.7.7 Size() 规模

获得该列的规模。

`OPTVColumn.`**`size`**()

该数值统计对应列中的元素个数，无论是否有重复项，例如，若某约束被多次添加，本数值依然相应增加。

返回
该列的规模。

返回类型
int

8.8 OptVerse 约束

class OPTVConstr

OptVerse 约束对象，用于从模型访问约束。注意约束不可由其构造函数创建，而需要通过模型的*OPTVModel.addConstr()* 或*OPTVModel.addConstrs()* 接口添加约束至模型。此外，二次约束的相应接口为*OPTVModel.addQConstr()*.

8.8.1 get() 查询约束属性

查询约束属性，关于可用属性请参考*OptVerse 属性*。

OPTVConstr.**get**(OPTVIntAttr *attr*)

参数

- **attr** –整数型属性。

返回
属性值。

返回类型
int

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

OPTVConstr.**get**(OPTVDblAttr *attr*)

参数

- **attr** –双精度型属性。

返回
属性值。

返回类型
double

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

OPTVConstr.**get**(OPTVStrAttr *attr*)

参数

- **attr** –字符串型属性。

返回
属性值。

返回类型
String

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

8.8.2 set() 设置约束属性

设置约束属性，只有调用*OPTVModel.update()* 后，新设置才可生效，更多细节请参考*OptVerse 属性*。

OPTVConstr.**set**(OPTVIntAttr *attr*, int *value*)

参数

- **attr** –整数型属性。
- **value** –新属性值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

OPTVConstr.**set**(OPTVDblAttr *attr*, int *value*)

参数

- **attr** –双精度型属性。
- **value** –新属性值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

OPTVConstr.**set**(OPTVStrAttr *attr*, int *value*)

参数

- **attr** –字符串型属性。
- **value** –新属性值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

8.8.3 index() 约束索引

获得约束在模型中的索引值。

OPTVConstr.**index**()

返回
约束索引/下标。

返回值

- **-1** –该约束在模型中不存在。
- **-2** –该约束已从模型中删除。
- **>=0** –该约束在模型中的索引/下标。

8.8.4 sameAs() 约束对比

查询两约束是否相同。

OPTVConstr.**sameAs**(OPTVConstr *c2*)

参数

- **c2** –需要与当前约束对比的约束。

返回
约束是否相同。

返回类型
boolean

返回值

- **true** –约束相同。
- **false** –约束不同。

8.9 OptVerse 二次约束

class OPTVQConstr

二次约束类用于访问模型中的二次约束。注意二次约束也不可以由其构造函数直接创建，而是通过接口*OPTVModel.addQConstr()* 将其添加到模型。

警告: 当求解的模型 **包含**二次约束时，QCQP 求解器要求模型的二次目标函数或二次约束的矩阵都是正半定的，否则会抛出*OPTVErrCode.MATRIX_NOT_PSD* 异常。

8.9.1 get() 查询二次约束属性

查询二次约束属性，更多细节请参考*OptVerse 属性*。

`OPTVQConstr.get(OPTVIntAttr attr)`

参数

- **attr** –整数型属性。

返回
属性值。

返回类型
int

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

`OPTVQConstr.get(OPTVDblAttr attr)`

参数

- **attr** –双精度型属性。

返回
属性值。

返回类型
double

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

`OPTVQConstr.get(OPTVStrAttr attr)`

参数

- **attr** –字符串型属性。

返回
属性值。

返回类型
String

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

8.9.2 set() 设置二次约束属性

设置二次约束属性，只有调用*OPTVModel.update()* 后，新设置才可生效，更多细节请参考*OptVerse 属性*。

`OPTVQConstr.set(OPTVIntAttr attr, int value)`

参数

- **attr** –整数型属性。

- **value** –新属性值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

OPTVQConstr.**set**(OPTVDblAttr *attr*, int *value*)

参数

- **attr** –双精度型属性。
- **value** –新属性值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

OPTVQConstr.**set**(OPTVStrAttr *attr*, int *value*)

参数

- **attr** –字符串型属性。
- **value** –新属性值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

8.9.3 index() 二次约束索引

获取二次约束在模型中的索引值。

OPTVQConstr.**index**()

返回
二次约束索引/下标。

返回类型
int

返回值

- **-1** –该二次约束在模型中不存在。

- **-2** –该二次约束已从模型中删除。
- **>=0** –该二次约束在模型中的索引/下标。

8.9.4 sameAs() 二次约束对比

查询两二次约束是否相同。

`OPTVQConstr.sameAs` (OPTVConstr *c2*)

参数

- **c2** –需要与当前二次约束对比的二次约束。

返回
二次约束是否相同。

返回类型
boolean

返回值

- **true** –二次约束相同。
- **false** –二次约束不同。

8.10 OPTVSOS SOS 约束

class OPTVSOS

特殊顺序集，special ordered set (SOS)，是指一组有序集合里，至多有一个非零值（SOS1 型）或至多有两个非零值（SOS2 型）。SOS 约束是对一组变量的取值进行限制的特殊约束，可以额外地指定模型中的整数条件。这类约束单独列出来，可以加快线性规划的求解速度。

OptVerse SOS 约束对象，用以从模型中获取 SOS 约束数据。这里，用户需要通过*`OPTVModel.addSOS()`* 接口向模型中添加 SOS 约束，而非借助于构造函数。具体地，OptVerse 支持下列两种 SOS 约束：

- SOS 类型 1 要求指定的一组变量至多有一个变量可取非零值
- SOS 类型 2 指定的一组变量至多有两个变量可取非零值，且取非零值的变量顺序要求相邻

8.10.1 get() 查询 SOS 约束属性

查询 SOS 约束的属性，更多细节请参考*OptVerse 属性*。

`OPTVSOS.get` (OPTVIntAttr *attr*)

参数

- **attr** –整数型属性。

返回
属性值。

返回类型
int

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

OPTVSOS.**get** (OPTVStrAttr *attr*)

参数

- **attr** –字符串型属性。

返回
属性值。

返回类型
String

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

8.10.2 set() 设置 SOS 约束属性

设置 SOS 约束属性，只有调用*OPTVModel.update()* 后，新设置才可生效，更多细节请参*OptVerse 属性*。

OPTVSOS.**set** (OPTVIntAttr *attr*, int *value*)

参数

- **attr** –整数型属性。
- **value** –新属性值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

OPTVSOS.**set** (OPTVStrAttr *attr*, String *value*)

参数

- **attr** –整数型属性。
- **value** –新属性值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

8.10.3 index() SOS 约束索引

获取 SOS 约束在模型中的索引值。

OPTVSOS.**index**()

返回
SOS 约束索引/下标。

返回类型
int

返回值

- **-1** –该 SOS 约束在模型中不存在。
- **-2** –该 SOS 约束已从模型中删除。
- **>=0** –该 SOS 约束在模型中的索引/下标。

8.10.4 sameAs() SOS 约束对比

查询两 SOS 约束是否相同。

OPTVSOS.**sameAs**(OPTVSOS *c2*)

参数

- **c2** –需要与当前 SOS 约束对比的 SOS 约束。

返回
SOS 约束是否相同。

返回类型
boolean

返回值

- **true** –SOS 约束相同。
- **false** –SOS 约束不同。

8.11 OptVerse 一般约束

class OPTVGenConstr

OptVerse 一般约束，用于从模型中获取一般约束数据。特别地。一般约束需要通过 OPTVModel.addGenConstrXxx 向模型中添加约束而构造，而不是借助于任何构造函数。

表 12: OptVerse 支持的一般约束类型

约束	类型	接口	描述
And	*OPTV_GENCONSTR_*	*OPTVModel.addGenConstrAnd()*	只有所有变量非零时，结果才非零
Or	*OPTV_GENCONSTR_*	*OPTVModel.addGenConstrOr()*	只要有一个变量非零，结果就非零
Indicator	*OPTV_GENCONSTR_*	*OPTVModel.addGenConstrIndicator*	返回值为布尔型，当且仅当相应的逻辑条件满足，取值为 1，否则取值为 0

8.11.1 get() 查询一般约束属性

查询一般约束属性，关于可用属性请参考*OptVerse 属性*。

OPTVGenConstr.**get**(OPTVIntAttr *attr*)

参数

- **attr** –整数型属性。

返回
属性值。

返回类型
int

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

OPTVGenConstr.**get**(OPTVDblAttr *attr*)

参数

- **attr** –双精度型属性。

返回
属性值。

返回类型
double

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

OPTVGenConstr.**get**(OPTVStrAttr *attr*)

参数

- **attr** –字符串型属性。

返回
属性值。

返回类型
String

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

8.11.2 set() 设置一般约束属性

设置一般约束属性，只有调用*OPTVModel.update()* 后，新设置才可生效，更多细节请参考*OptVerse 属性*。

OPTVGenConstr.**set**(OPTVDblAttr *attr*, int *value*)

参数

- **attr** –双精度型属性。
- **value** –新属性值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

OPTVGenConstr.**set**(OPTVStrAttr *attr*, int *value*)

参数

- **attr** –字符串型属性。
- **value** –新属性值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知约束属性。

8.11.3 index() 一般约束索引

获得一般约束在模型中的索引值。

OPTVGenConstr.**index**()

返回
约束索引/下标。

返回类型
int

返回值

- **-1** –该约束在模型中不存在。
- **-2** –该约束已从模型中删除。
- **>=0** –该约束在模型中的索引/下标。

8.11.4 sameAs() 一般约束对比

查询两一般约束是否为同一对象。

`OPTVGenConstr.sameAs` (OPTVGenConstr *c2*)

参数

- **c2** –需要与当前约束对比的约束。

返回
约束是否相同。

返回类型
boolean

返回值

- **true** –约束相同。
- **false** –约束不同。

8.12 OptVerse 线性表达式

class OPTVLinExpr

线性表达式*OPTVLinExpr* 类用于构建线性的目标函数或者约束函数。线性表达式对象由变量和常量组成，边获取边构造，如下例所示的表达式 `1 * x + 2 * y + 3 * x`，它包含以下元素：

下标	系数	变量
0	1	x
1	2	y
2	3	x

8.12.1 addTerms() 增加多项

该功能向当前表达式（后）添加一组新的项。

`OPTVLinExpr.addTerms` (double[] *coefs*, OPTVVar[] *vars*, int *cnt*)

此函数将 $\text{coefs}_i * \text{vars}_i, \forall i \in [0, \text{cnt}]$ 添加到当前线性表达式后。

参数

- **coefs** –系数序列。
- **vars** –变量序列。
- **cnt** –添加项的个数。

返回
无。

返回类型
void

抛出
OPTVErrCode.INVALID_ARGUMENT –vars 或 coefs 为空指针或 cnt 取负值。

8.12.2 clear() 清空

清空该表达式，*OPTVLinExpr.size()* 将返回 0. 同时，该表达式从数值意义上将等同于 0.

OPTVLinExpr.**clear**()

返回
无。

返回类型
void

8.12.3 getCoef() 获取系数

据指定索引值获得表达式中对应变量的系数。

OPTVLinExpr.**getCoef**(int *i*)

参数
- **i** –指定的索引值。

返回
对应的变量系数。

返回类型
double

抛出
OPTVErrCode.INDEX_OUT_OF_RANGE –若 i ∉ [0, this->Size()].

8.12.4 getVar() 变量获取

根据指定索引值获得表达式中的对应变量。

OPTVLinExpr.**getVar**(int *i*)

参数
- **i** –指定的索引值。

返回
对应的变量。

返回类型
OPTVVar

抛出
OPTVErrCode.INDEX_OUT_OF_RANGE –若 i ∉ [0, this->Size()].

8.12.5 getConstant() 获取常数项

获得当前表达式中的常数项。

`OPTVLinExpr.`**`getConstant`**()

返回
表达式中的常量数值。

返回类型
double

8.12.6 getValue() 取值

获得该表达式的取值。

`OPTVLinExpr.`**`getValue`**()

返回
对应原始解的表达式的取值，即该表达式的变量值与系数的乘积之和。

返回类型
double

抛出
OPTVErrCode.DATA_NOT_AVAILABLE –当前变量取值不存在。

8.12.7 remove() 删除项

从表达式中删除项。

`OPTVLinExpr.`**`remove`**(int *i*)

根据指定索引值从该表达式中删除项。

参数

- **i** –指定的索引值。

返回
无。

返回类型
void

抛出
OPTVErrCode.INDEX_OUT_OF_RANGE –若 $i \notin [0, \texttt{this->Size()}]$.

`OPTVLinExpr.`**`remove`**(OPTVVar *v*)

从线性表达式中删除指定变量涉及的所有项。

参数

- **v** –指定要删除的变量。

返回
是否成功删除任何项。

返回类型
boolean

返回值

- **true** –成功删除任何项。
- **false** –未删除任何项。

抛出
OPTVErrCode.NOT_IN_MODEL –指定的变量（在模型中）不存在。

8.12.8 sameAs() 线性表达式对比

查询两线性表达式是否相同。

`OPTVLinExpr.`**`sameAs`**(OPTVLinExpr *e2*)

参数

- **e2** –需要与当前线性表达式对比的线性表达式。

返回
线性表达式是否相同。

返回类型
boolean

返回值

- **true** –线性表达式相同。
- **false** –线性表达式不同。

8.12.9 size() 规模

获得该表达式的规模。

`OPTVLinExpr.`**`size`**()

该数值对应线性表达式中的变量个数，不论变量是否重复出现。例如表达式 `x+y` 对应 `Size = 2`，而 `x+y+x` 对应 `Size = 3`.

返回
该表达式的规模。

返回类型
int

8.13 OptVerse 二次表达式

`class OPTVQuadExpr`

二次表达式类*OPTVQuadExpr* 用于构建二次的目标函数或约束。

二次表达式对象由变量和常量组成，通常包含三部分：二次项、线性项和常数项。其中，二次项依照第一个变量、第二个变量和系数的顺序保存在容器中，而线性部分以前一章节所述的线性表达式的形式保存。二次表达式的索引仅限于获取二次项，线性项部分的需单独获取。如下例所示的二次表达式 `1*x*x + 2*x*y + 3*x*y + 4*y*y + 5*x + 6*y + 7*x` 的描述如下：

索引	系数	变量 1	变量 2
0	1	x	x
1	2	x	y
2	3	x	y
3	4	y	y

它的线性项部分需要通过*OPTVQuadExpr.getLinExpr()* 接口访问，例如上述表达式的线性部分由以下部分组成：

索引	系数	变量
0	5	x
1	6	y
2	7	x

8.13.1 addConstant() 增加常数项

向二次表达式增加常数项。

OPTVQuadExpr.**addConstant**(double *c*)

此函数向当前二次表达式添加常数项，但不是覆盖。

参数

- **c** –要添加的常数。

返回

无。

返回类型

void

8.13.2 addTerm() 增加单项

向当前二次表达式后添加一个线性项。

OPTVQuadExpr.**addTerm**(double *coeff*, OPTVVar *var*)

该函数向当前二次表达式后添加 coeff ∗ var 项。

参数

- **coeff** –系数值。
- **var** –变量对象。

返回

无。

返回类型

void

抛出

OPTVErrCode.INVALID_ARGUMENT –需添加的变量 var 为空。

向当前二次表达式后添加一个二次线性项。

`OPTVQuadExpr.addTerm`(double *coeff*, OPTVVar *var1*, OPTVVar *var2*)

该函数向当前二次表达式后添加 coeff ∗ var1 ∗ var2 项。

参数

- **coeff** –系数值。
- **var1** –该项的第一个变量。
- **var2** –该项的第二个变量。

返回

无。

返回类型

void

抛出

OPTVErrCode.INVALID_ARGUMENT –需添加的项中变量 `var1` 或 `var2` 为空。

8.13.3 addTerms() 增加多项

向当前二次表达式后添加多个线性项。

`OPTVQuadExpr.addTerms`(double[] *coeff*, OPTVVar[] *var*, int *cnt*)

该函数向当前二次表达式后添加 $\text{coeff}_i * \text{var}_i$, $\forall i \in [0, \text{cnt}]$.

参数

- **coeff** –C-风格的系数序列。
- **var** –C-风格的变量序列。
- **cnt** –需要添加的项的个数。

返回

无。

返回类型

void

抛出

OPTVErrCode.INVALID_ARGUMENT –参数 `var` 或 `coeff` 为空指针，或 `cnt` 取负值。

向当前二次表达式后添加多个二次性项。

`OPTVQuadExpr.addTerms`(double[] *coeff*, OPTVVar[] *var1*, OPTVVar[] *var2*, int *cnt*)

该函数向当前二次表达式后添加 $\text{coeff}_i * \text{var1}_i * \text{var2}_i$, $\forall i \in [0, \text{cnt}]$.

参数

- **coeff** –C-风格的系数序列。
- **var1** –C-风格的变量 1 序列。
- **var2** –C-风格的变量 2 序列。
- **cnt** –需要添加的项的个数。

返回
无。

返回类型
void

抛出
OPTVErrCode.INVALID_ARGUMENT –参数 var1，var2 或 coeff 为空指针，或 cnt 取负值。

8.13.4 clear() 清空

清空该表达式。

OPTVQuadExpr.**clear**()

该函数清空该表达式涉及的所有存储容器，*OPTVQuadExpr.size()* 将返回 0. 同时，该表达式也清空相应的线性表达式部分，重置常数项为 0.

返回
无。

返回类型
void

8.13.5 getCoeff() 获取系数

据指定索引值获得表达式中对应二次项的系数。

OPTVQuadExpr.**getCoeff**(int *i*)

参数

- **i** –指定的索引值。

返回
对应的二次项系数。

返回类型
double

抛出
OPTVErrCode.DATA_NOT_AVAILABLE –若 i $\notin [0, \texttt{this->Size()}]$.

8.13.6 getConstant() 获取常量

获得当前表达式中的常数项。

OPTVQuadExpr.**getConstant**()

返回
表达式中的常量数值。

返回类型
double

8.13.7 getLinExpr() 线性部分

获取二次表达式的线性部分。

OPTVQuadExpr.**getLinExpr**()

返回
二次表达式的线性部分。

返回类型
OPTVLinExpr

8.13.8 getValue() 取值

获得该二次表达式的取值。

OPTVQuadExpr.**getValue**()

返回
对应原始解的表达式的取值。

返回类型
double

抛出
OPTVErrCode.DATA_NOT_AVAILABLE –当前变量取值不存在。

8.13.9 getVar1() 获取变量 1

根据指定索引值获得表达式中的对应二次项的第一个变量。

OPTVQuadExpr.**getVar1**(int *i*)

该函数返回表达式中的对应二次项的第一个变量，如表达式 `x*x + x*y + y*y` 的描述如下：

索引	变量 1	变量 2
0	x	x
1	x	y
2	y	y

参数

- **i** –指定的二次项对应的索引值。

返回
对应二次项的第一个变量。

返回类型
OPTVVar

抛出
OPTVErrCode.DATA_NOT_AVAILABLE –若 $i \notin [0, \texttt{this->Size()}]$.

8.13.10 getVar2() 获取变量 2

根据指定索引值获得表达式中的对应二次项的第二个变量。请参考*OPTVQuadExpr.getVar1()* 和示例。

OPTVQuadExpr.**getVar2**(int *i*)

该函数返回表达式中的对应二次项的第二个变量。

参数

- **i** –指定的二次项对应的索引值。

返回

对应二次项的第二个变量。

返回类型

OPTVVar

抛出

OPTVErrCode.DATA_NOT_AVAILABLE –若 i $\notin [0, \texttt{this->Size()}]$.

8.13.11 remove() 删除项

从表达式中删除二次项。

OPTVQuadExpr.**remove**(int *i*)

根据指定索引值从二次表达式中删除二次项，但不可通过此函数删除线性项的任何部分。

参数

- **i** –指定的索引值。

返回

无。

返回类型

void

抛出

OPTVErrCode.DATA_NOT_AVAILABLE –若 i $\notin [0, \texttt{this->Size()}]$.

从表达式中删除变量。

OPTVQuadExpr.**remove**(OPTVVar *v*)

从二次表达式中（包括线性部分）删除指定变量涉及的所有项。

参数

- **v** –指定要删除的变量。

返回

无。

返回类型

void

抛出

OPTVErrCode.NOT_IN_MODEL –指定的变量（在模型中）不存在。

8.13.12 sameAs() 二次表达式对比

查询两二次表达式是否相同。

`OPTVQuadExpr.sameAs` (OPTVQuadExpr *e2*)

参数

- **e2** –需要与当前二次表达式对比的线性表达式。

返回
二次表达式是否相同。

返回类型
boolean

返回值

- **true** –二次表达式相同。
- **false** –二次表达式不同。

8.13.13 size() 规模

获得该表达式的规模。

`OPTVQuadExpr.size`()

该数值对应当前二次表达式中的 二**次**项的个数，不论重复与否。例如，表达式 `x*x + y*y + y` 对应 `Size = 2`，而表达式 `x*x + x*x + y*y + y` 对应 `Size = 3`.

返回
该表达式的规模。

返回类型
int

8.14 OptVerse 模型

class OPTVModel

8.14.1 addConstr() 增加单个约束

向模型中添加单个线性约束。

`OPTVModel.addConstr` (OPTVLinExpr *expr*, double *lhs*, double *rhs*, String *name*="")

该函数向模型中添加一个线性约束。当需要一次性添加大量约束时，建议使用批量模式 (*OPTVModel.addConstrs()*)，虽然大多数情况下，效率差别不大。

参数

- **expr** –约束的线性表达式部分。
- **lhs** –约束的左端值。
- **rhs** –约束的右端值。

- **name** –约束的名字。

返回
添加到模型的约束对象。

返回类型
OPTVConstr

抛出
OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

8.14.2 addConstrs() 批量增加约束

向模型中添加多个线性约束。

OPTVModel.**addConstrs**(OPTVLinExpr[] *expr*, double[] *lhs*, double[] *rhs*, String[] *name*, int *count*)

该函数一次性向模型中添加多个线性约束，尤其适用于添加大量约束的情形。

参数

- **expr** –约束的线性表达式序列。
- **lhs** –约束的左端值序列。
- **rhs** –约束的右端值序列。
- **name** –约束的名字序列。
- **count** –添加约束的个数。

返回
添加到模型的约束列表。

返回类型
OPTVConstr[]

抛出
OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 包含的约束名与模型中的其他组件重名。

8.14.3 addQConstr() 增加单个二次约束

向模型中添加单个二次约束。

OPTVModel.**addQConstr**(OPTVQuadExpr *expr*, double *lhs*, double *rhs*, String *name*="")

该函数向模型中添加一个二次约束。

参数

- **expr** –约束的二次表达式部分。
- **lhs** –约束的下界。
- **rhs** –约束的上界。
- **name** –约束的名字。

返回
添加到模型的约束对象。

返回类型
OPTVQConstr

抛出
OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

8.14.4 addGenConstrAnd() 增加一般约束 AND

向模型中添加单个 AND 一般约束。

OPTVModel.**addGenConstrAnd**(OPTVVar *resultant*, OPTVVar[] *vars*, int *len*, String *name* = "")

该函数向模型中添加单个 AND 一般约束：只有所有变量非零时，结果才非零。

参数

- **resultant** –约束的布尔型返回值。
- **vars** –所有参与检验的变量序列。
- **len** –vars 序列长度。
- **name** –约束的名字。

返回
添加到模型的一般约束对象。

返回类型
OPTVGenConstr

抛出
OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

8.14.5 addGenConstrOr() 增加一般约束 OR

向模型中添加单个 OR 一般约束。

OPTVModel.**addGenConstrOr**(OPTVVar *resultant*, OPTVVar[] *vars*, int *len*, String *name* = "")

该函数向模型中添加单个 OR 一般约束：只要有一个变量非零，结果就非零。

参数

- **resultant** –约束的布尔型返回值。
- **vars** –所有参与检验的变量的序列。
- **len** –vars 序列长度。
- **name** –约束的名字。

返回
添加到模型的一般约束对象。

返回类型
OPTVGenConstr

抛出
OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

8.14.6 addGenConstrIndicator() 增加一般约束 INDICATOR

向模型中添加单个 INDICATOR 一般约束。

OPTVModel.**addGenConstrIndicator**(OPTVVar *binVar*, int *binVal*, OPTVLinExpr *expr*, char *sense*, double *rhs*, String *name* = "")

该函数向模型中添加单个 INDICATOR 一般约束：当且仅当相应的逻辑条件满足，取值为 1，否则取值为 0.

参数

- **binVar** –二进制指示变量。
- **binVal** –要求线性约束必须满足的二进制指示变量的值 (0 or 1).
- **expr** –参与检验的线性约束表达式。
- **sense** – 线性约束的含义，如*OPTV_SENSE_LESS*，*OPTV_SENSE_EQUAL*，或*OPTV_SENSE_GREATER*.
- **rhs** –参与检验的线性约束右端值。
- **name** –约束的名字。

返回

添加到模型的一般约束对象。

返回类型

OPTVGenConstr

抛出

OPTVErrCode.INVALID_ARGUMENT –`name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

8.14.7 AddGenConstrAbs() 增加一般约束 ABS

向模型中添加单个 ABS 一般约束：$r = \text{abs}(x)$。此约束条件保证因变量 r 的值为自变量 x 的绝对值，即 $r = |x|$, 其中 $r \geq 0$。

OPTVModel.**addGenConstrAbs**(OPTVVar *resultant*, OPTVVar *inputVar*, String *name*)

该函数向模型中添加单个 ABS 一般约束。

参数

- **resultant** –结果变量，该变量的值为另一变量的绝对值。
- **inputVar** –该变量的绝对值将被结果变量采用。
- **name** –约束的名字。

返回

添加到模型的一般约束对象。

返回类型

OPTVGenConstr

抛出

OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他约束重名。

8.14.8 AddGenConstrNorm() 增加一般约束 NORM

向模型中添加单个 NORM 一般约束: $r = \text{norm}\{x_1, \ldots, x_n\}$，其中因变量 r 的值为向量 $x_1, \ldots, x_n$ 的范数。

OPTVModel.**addGenConstrNorm** (OPTVVar *resultant*, OPTVVar[] *vars*, int *len*, int *normType*, String *name*)

该函数向模型中添加单个 NORM 一般约束。

参数

- **resultant** –结果变量，其值为参数向量的范数。
- **vars** –用于计算向量范数的变量。
- **len** –向量长度。
- **normType** –范数类型：-1 为 L^∞ 范数，0 为 L^0 范数，1 为 L^1 范数。
- **name** –约束的名字。

返回

添加到模型的一般约束对象。

返回类型

OPTVGenConstr

抛出

OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他约束重名。

8.14.9 addGenConstrMax() 添加一般约束 MAX

向模型中添加单个 MAX 约束： $r = \max\{x_1, \ldots, x_n\}$ 。

该约束确保因变量 r 的取值为变量向量 $x_1, \ldots, x_n$ 中的最大值。

OPTVModel.**addGenConstrMax** (OPTVVar *resultant*, OPTVVar[] *vars*, int *len*, String *name* = *""*, double *constant* = *-OPTV_INF*)

该函数向模型中添加单个 MAX 约束。

参数

- **resultant** –结果变量，该变量的值为其他变量的最大值。
- **vars** –变量向量，该向量的最大值会被结果变量采用。
- **len** –vars 向量长度。
- **name** –约束名称。
- **constant** –[**可选**] 参与 MAX 操作的常量。

返回

一个添加到模型中的一般约束对象

返回类型

OPTVGenConstr

抛出

OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他约束重名。

8.14.10 addGenConstrMin() 添加一般约束 MIN

向模型中添加单个 MIN 约束：$r = \min\{x_1, \ldots, x_n\}$ 。

该约束确保因变量 r 的取值为变量向量 $x_1, \ldots, x_n$ 中的最小值。

OPTVModel.**addGenConstrMin**(OPTVVar *resultant*, OPTVVar[] *vars*, int *len*, String *name* = *""*, double *constant* = *OPTV_INF*)

该函数向模型中添加单个 MIN 约束。

参数

- **resultant** –结果变量，该变量的值为其他变量的最小值。
- **vars** –变量向量，该向量的最小值会被结果变量采用。
- **len** –vars 向量长度。
- **name** –约束名称。
- **constant** –[**可选**] 参与 MIN 操作的常量。

返回

一个添加到模型中的一般约束对象。

返回类型

OPTVGenConstr

抛出

OPTVErrCode.INVALID_ARGUMENT –name 为空，数值参数中存在 NaN 或者 name 与模型中的其他约束重名。

8.14.11 AddSOS() 增加单个 SOS 约束

向模型中添加单个 SOS 约束。

OPTVModel.**addSOS**(OPTVVar[] *vars*, double[] *weights*, int *len*, int *type*, String *name*=*""*)

参数

- **vars** –SOS 约束所涉及变量的序列。
- **weights** –SOS 的变量权重的序列。
- **len** –vars 和 weights 的序列长度。
- **type** –SOS 类型，取值为 1 或 2.
- **name** –SOS 约束的名字。

返回

添加到模型的 SOS 约束对象。

返回类型

OPTVSOS

抛出

- *OPTVErrCode.INVALID_ARGUMENT* –type 取值不是 1 或 2.
- *OPTVErrCode.NOT_IN_MODEL* –vars 序列中的某个变量不属于当前模型。
- *OPTVErrCode.INDEX_OUT_OF_RANGE* –当前模型存在无效下标，比如模型关联的环境发生溢出时，执行本操作有可能抛出此错误。

8.14.12 addVar() 增加变量

向模型中添加单个变量。

OPTVModel.**addVar**(double *lb*, double *ub*, double *obj*, OPTVCharAttr *type*, String *name* = "")

该函数向模型中添加一个变量。当需要一次性添加大量变量时，建议使用批量模式（*OPTVModel.addVars()*），虽然大多数情况下，效率差别不大。

参数

- **lb** –变量的下界。
- **ub** –变量的上界。
- **obj** –变量在目标函数中的系数。
- **type** –变量类型，取值包括*OPTV_CONTINUOUS*, *OPTV_INTEGER*, *OPTV_BINARY*.
- **name** –变量名称。

返回

添加到模型的变量对象。

返回类型

OPTVVar

抛出

OPTVErrCode.INVALID_ARGUMENT –未知变量类型，name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

OPTVModel.**addVar**(double *lb*, double *ub*, double *obj*, char *type*, OPTVColumn *col*, String *name* = "")

该函数向模型中添加一个变量，并可以设置现有约束中的相应系数。当需要一次性添加大量变量时，建议使用批量模式（*OPTVModel.addVars()*），虽然大多数情况下，效率差别不大。

参数

- **lb** –变量的上界。
- **ub** –变量的下界。
- **obj** –变量在目标函数中的（线性）系数。
- **type** –变量类型，取值包括*OPTV_CONTINUOUS*, *OPTV_INTEGER*, *OPTV_BINARY*.
- **col** –设置约束系数的列对象。
- **name** –变量名称。

返回

添加到模型的变量对象。

返回类型

OPTVVar

抛出

OPTVErrCode.INVALID_ARGUMENT –未知变量类型，name 为空，数值参数中存在 NaN 或者 name 与模型中的其他组件重名。

8.14.13 addVars() 批量增加变量

向模型中添加多个变量。

OPTVModel.**addVars**(double[] *lb*, double[] *ub*, double[] *obj*, char[] *type*, String[] *name*, int *count*)

该函数一次性向模型中添加多个变量，尤其适用于添加大量变量的情形。

参数

- **lb** –变量的下界序列。
- **ub** –变量的上界序列。
- **obj** –变量序列对应的目标系数。
- **type** –变量类型序列，包括*OPTV_CONTINUOUS*, *OPTV_INTEGER*, *OPTV_BINARY*.
- **name** –变量名称序列。
- **count** –添加变量的个数。

返回

添加到模型的变量序列。

返回类型

OPTVVar[]

抛出

OPTVErrCode.INVALID_ARGUMENT –type 包含未知变量类型，name 为空，数值参数中存在 NaN 或者 name 包含的变量名与模型中的其他组件重名。

OPTVModel.**addVars**(double[] *lb*, double[] *ub*, double[] *obj*, char[] *type*, OPTVColumn[] *col*, String[] *names*, int *count*)

该函数向模型中添加多个变量，并可以设置现有约束中的相应系数。当需要一次性添加大量变量时，建议使用此功能，虽然大多数情况下，效率差别不大。

参数

- **lb** –变量的下界序列。
- **ub** –变量的上界序列。
- **obj** –变量序列对应的目标系数。
- **type** –变量类型序列，包括*OPTV_CONTINUOUS*, *OPTV_INTEGER*, *OPTV_BINARY*.
- **col** –设置约束系数的列对象序列。
- **names** –变量名称序列。
- **count** –添加变量的个数。

返回

添加到模型的变量序列。

返回类型

OPTVVar[]

抛出

OPTVErrCode.INVALID_ARGUMENT –type 包含未知变量类型，name 为空，数值参数中存在 NaN 或者 name 包含的变量名与模型中的其他组件重名。

8.14.14 checkSolution() 校验解

检验指定的解是否可行。

`OPTVModel.checkSolution`(double[] *solution*, double *tol=1e-8*)

该函数供用户检验指定的解是否满足当前模型的可行性要求，但 **不可以**检验内部存储的解。

参数

- **`solution`** –(列）解向量。
- **`tol`** –约束可行性整数性检查精度（**可选**，默认值为 10^{-8}）。

返回
解对模型是否可行。

返回类型
boolean

返回值

- **`false`** –解对模型不可行。
- **`true`** –解对模型是可行的。

8.14.15 computeIIS() 不可约不一致子系统

计算不可约不一致子系统（IIS），关于 IIS 更多细节请参考不可约不一致子系统 *(IIS)* 快速入门。

`OPTVModel.computeIIS`()

返回
IIS 是否计算成功。

返回类型
boolean

返回值

- **`false`** –IIS 计算不成功。
- **`true`** –成功计算 IIS。

8.14.16 feasRelax() 不可行问题的诊断及分析

进行不可行问题的诊断及分析，更多细节请参考*LP* 不可行诊断及修复快速入门。

`OPTVModel.feasRelax`()

该函数进行不可行问题的诊断及分析，目前仅支持 L^1 范数。

返回
不可行问题修复的状态。

返回类型
int

返回值

- **`0`** –当前问题是可行的。
- **`-1`** –不可行问题修复失败。

- **>0** –Phase 1 最小违背量的目标函数值。

8.14.17 getCoef() 查询变量系数

查询给定约束中指定变量的系数。

OPTVModel.**getCoef**(OPTVConstr *con*, OPTVVar *var*)

参数

- **con** –需要查询的约束。
- **var** –需要查询的变量。

返回
查询变量在该约束中的系数。

返回类型
double

抛出

- *OPTVErrCode.INVALID_ARGUMENT* –约束或变量不存在。
- *OPTVErrCode.DATA_NOT_AVAILABLE* –数据不存在，或约束的线性表达式部分不可用。

8.14.18 GetColumn() 获取列

获取指定变量对应的列对象。

OPTVModel.**getColumn**(OPTVVar *v*)

参数

- **v** –需要查询的变量。

返回
指定变量对应的列对象。

返回类型
OPTVColumn

Throw OPTVErrCode.DATA_NOT_AVAILABLE
该变量不存在于当前模型

Throw OPTVErrCode.INDEX_OUT_OF_RANGE
变量索引值*Index*超过了模型的变量总数。若当前模型需要执行更新，或者当前变量属于其它模型时，该错会被抛出。

8.14.19 getConstrByName() 通过名称访问约束

通过名称获取约束。

OPTVModel.**getConstrByName**(String *name*)

参数

- **name** –需要访问的约束名称。

返回
名称匹配 name 的约束。

返回类型
OPTVConstr

抛出
OPTVErrCode.DATA_NOT_AVAILABLE –给定的名称为空，或模型不存在命名为 name 的约束。

8.14.20 getConstr() 获取单个约束

根据指定索引值获得模型中对应的约束。

OPTVModel.**getConstr**(int *i*)

参数

- **i** –指定的索引值。

返回
对应的约束。

返回类型
OPTVConstr

抛出
OPTVErrCode.INDEX_OUT_OF_RANGE –指定的索引值超出了模型的约束总数。

8.14.21 getConstrs() 获取所有线性约束

获得模型中的所有线性约束。

OPTVModel.**getConstrs**()

返回
包含模型中所有线性约束的向量。

返回类型
OPTVConstr[]

8.14.22 getGenConstrAnd() 获取单个 AND 约束

获得模型中特定的 AND 一般约束数据。

OPTVModel.**getGenConstrAnd**(OPTVGenConstr *genConstr*, OPTVVar *resultant*, OPTVVar[] *vars*)

参数

- **genConstr** –需要查询的一般约束。
- **resultant** –该一般约束对应的结果变量。
- **vars** –参与检验该一般约束的所有变量的序列。

返回
vars 序列中变量的个数。

返回类型
int

抛出
OPTVErrCode.NOT_IN_MODEL –模型中不存在该一般约束，当该一般约束已从模型中被删除或者是由默认的构造函数创建时，会抛出此错误。

8.14.23 getGenConstrOr() 获取单个 OR 约束

获得模型中特定的 OR 一般约束数据。

OPTVModel.**getGenConstrOr**(OPTVGenConstr *genConstr*, OPTVVar *resultant*, OPTVVar[] *vars*)

参数

- **genConstr** –需要查询的一般约束。
- **resultant** –该一般约束对应的结果变量。
- **vars** –参与检验该一般约束的所有变量的列表。

返回
vars 序列中变量的个数。

返回类型
int

抛出
OPTVErrCode.NOT_IN_MODEL –模型中不存在该一般约束，当该一般约束已从模型中被删除或者是由默认的构造函数创建时，会抛出此错误。

8.14.24 getGenConstrIndicator() 获取单个 INDICATOR 约束

获得模型中特定的 INDICATOR 一般约束数据。

OPTVModel.**getGenConstrIndicator**(OPTVGenConstr *genConstr*, OPTVVar *binVar*, int *binVal*, OPTVLinExpr *expr*, char *sense*, double *rhs*)

参数

- **genConstr** –需要查询的一般约束。
- **binVar** –二进制指示变量。

- **binVal** –要求线性约束必须满足的二进制指示变量的值 (0 or 1).
- **expr** –参与检验的线性约束表达式。
- **sense** – 线性约束的含义，如*OPTV_SENSE_LESS*, *OPTV_SENSE_EQUAL*, 或*OPTV_SENSE_GREATER*.
- **rhs** –参与检验的线性约束右端值。

返回
无。

返回类型
void

抛出
OPTVErrCode.NOT_IN_MODEL –模型中不存在该一般约束，当该一般约束已从模型中被删除或者是由默认的构造函数创建时，会抛出此错误。

8.14.25 getQConstr() 获取二次约束

根据指定索引值获得模型中对应的二次约束。

OPTVModel.**getQConstr(int i) const**()

参数

- **i** –指定的索引值。

返回
对应的二次约束。

返回类型
OPTVQConstr

抛出
OPTVErrCode.INDEX_OUT_OF_RANGE –指定的索引值超出了模型的约束总数。

此函数中的参数 *i* 是指全局下标，与约束的类型无关。例如，用户依次添加了一个线性约束和一个二次约束，那么该线性约束的索引值为 *0* 而二次约束的索引值为 *1*.

8.14.26 getQConstrs() 获取所有二次约束

获得模型中的所有二次约束。

OPTVModel.**getQConstrs() const**()

返回
包含模型中所有二次约束的向量。

8.14.27 getSOS() 获取单个 SOS 约束

根据索引值获取指定类型的 SOS 约束。

`OPTVModel.getSOS(int i, int type) const`()

参数

- **i** –指定 SOS 约束在模型中的索引值。
- **type** –SOS 类型，取值为 1 或 2.

返回
对应索引值 i 的指定类型的 SOS 约束。

返回类型
OPTVSOS

抛出

- *OPTVErrCode.INDEX_OUT_OF_RANGE* –索引值 i 超出了模型的该类型的 SOS 约束总数。
- *OPTVErrCode.INVALID_ARGUMENT* –type 取值不是 1 或 2.

这里，参数 i 特指某一类型 SOS 约束集中的索引值，例如，如果用户先后添加了类型为 1 和 2 的两个 SOS 约束，那么两者的索引值都是 0.

8.14.28 getSOSData() 获取 SOS 约束数据

`OPTVModel.getSOSData`(OPTVSOS *sos*, OPTVVar[] *vars*, double[] *weights*)

获取 SOS 约束的内部数据。

参数

- **sos** –当前访问的 SOS 约束对象。
- **vars** –变量序列的空向量。如果非空，该容器在本次函数调用中会被清空。
- **weights** –变量权重序列的空向量。如果非空，该容器在本次函数调用中会被清空。

返回
无。

返回类型
void

抛出

- *OPTVErrCode.DATA_NOT_AVAILABLE* –SOS 约束是由默认构造函数生成的。
- *OPTVErrCode.NOT_IN_MODEL* –SOS 约束属于另外一个模型，或已经从本模型中被删除。
- *OPTVErrCode.INDEX_OUT_OF_RANGE* –约束下表越界、超出容器范围，可能需要对当前模型执行更新。

8.14.29 getSOSs() 获取多个 SOS 约束

获取指定类型的所有 SOS 约束。

`OPTVModel.getSOSs(int type) const`()

参数

- **type** –SOS 类型，取值为 `1` 或 `2`.

返回
包含模型中所有类型为 `type` 的 SOS 约束的序列。

返回类型
OPTVSOS[]

抛出

- *OPTVErrCode.INDEX_OUT_OF_RANGE* –SOS 约束索引值存在越界、超出了模型的该类型的 SOS 约束总数，可能需要对当前模型执行更新。
- *OPTVErrCode.INVALID_ARGUMENT* –`type` 取值不是 `1` 或 `2`.

8.14.30 getObjective() 获取线性目标函数

获取线性目标函数。

`OPTVModel.getObjective`()

返回
目标函数的线性表达式。

返回类型
OPTVLinExpr

8.14.31 getQObjective() 获取二次目标函数

获取二次目标函数。

`OPTVModel.getQObjective() const`()

返回
目标函数的二次表达式。

8.14.32 getRow() 获取行

获取指定约束对应的线性函数表达式。

`OPTVModel.getRow`(OPTVConstr *c*)

参数

- **c** –指定的约束对象。

返回
该约束对应的线性表达式。

返回类型
OPTVLinExpr

抛出

- *OPTVErrCode.DATA_NOT_AVAILABLE* –该约束不存在于当前模型。
- *OPTVErrCode.INDEX_OUT_OF_RANGE* –当前约束的索引*Index* 超过了模型的约束总数。若当前模型需要执行更新，或者当前约束属于其它模型时，该错会被抛出。

8.14.33 get() 获取参数和属性

获取参数

获取求解器的参数取值，可用参数请参考*OptVerse 参数*。

OPTVModel.**get**(OPTVIntParam *param*)

参数

- **param** –整数型参数。

返回
参数取值。

抛出
OPTVErrCode.UNKNOWN_PARAMETER –未知参数。

OPTVModel.**get**(OPTVDblParam *param*)

参数

- **param** –双精度型参数。

返回
参数取值。

抛出
OPTVErrCode.UNKNOWN_PARAMETER –未知参数。

OPTVModel.**get**(OPTVStrParam *param*)

参数

- **param** –字符串型参数。

返回
参数取值。

抛出
OPTVErrCode.UNKNOWN_PARAMETER –未知参数。

获取属性

获取属性的取值，可用属性请参考*OptVerse 属性*。

`OPTVModel.get(OPTVIntAttr attr)`

参数

- **attr** –整数型属性。

返回
属性取值。

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知属性。

`OPTVModel.get(OPTVDblAttr attr)`

参数

- **attr** –双精度型属性。

返回
属性取值。

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知属性。

`OPTVModel.get(OPTVLongAttr attr)`

参数

- **attr** –长整型属性。

返回
属性取值。

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知属性。

8.14.34 getVarByName() 通过名称访问变量

通过名称获取变量。

`OPTVModel.getVarByName(String name)`

参数

- **name** –需要访问的变量名称。

返回
名称匹配 `name` 的变量。

返回类型
OPTVVar

抛出
OPTVErrCode.DATA_NOT_AVAILABLE –给定的名称为空，或模型不存在命名为 `name` 的变量。

8.14.35 getVar() 获取单个变量

根据指定索引值获得模型中对应的变量。

OPTVModel.**getVar**(int *i*)

参数

- **i** –指定的索引值。

返回
对应的变量。

返回类型
OPTVVar

抛出
OPTVErrCode.INDEX_OUT_OF_RANGE –指定的索引值超出了模型的变量总数。

8.14.36 getVars() 获取所有变量

获得模型中的所有变量。

OPTVModel.**getVars**()

返回
包含模型中所有变量的向量。

返回类型
OPTVVar[]

8.14.37 optimize() 求解模型

对模型进行求解。

OPTVModel.**optimize**()

返回
无。

返回类型
void

抛出
OPTVErrCode.MATRIX_NOT_PSD –模型中包含二次目标函数或者二次约束，而该表达式的矩阵不满足正半定性质。当求解的模型 **包含**二次约束时，QCQP 求解器要求模型的二次目标函数或二次约束的矩阵都是正半定的。

求解器采用哪种优化算法取决于求解器参数配置和模型属性，例如变量类型被用以自动识别 LP, QP, MIP 及网络流问题。

8.14.38 presolve() 预处理

对模型进行预处理。

`OPTVModel.`**`presolve`**()

该函数会应用预处理流程，返回一个新的*OPTVModel* 对象。原模型的状态被修改为 被预处理.

返回
预处理后的模型。

返回类型
OPTVModel

8.14.39 read() 读取模型和基

从文件中读取读取模型和基。

`OPTVModel.`**`read`**(String *fileName*)

该函数供读取基信息，也可以读取模型文件，然而建议由构造函数读取文件中的模型，参考文件格式。

表 13: 支持读入的文件格式

.mps	.lp	.bas

表 14: 支持读入的压缩格式

.gz	.z	.Z

参数

- **`fileName`** –C++ 字符串文件名。

返回
无。

返回类型
void

抛出
OPTVErrCode.ERROR_FILE_READ –文件名或后缀为空，文件存在错误格式。

8.14.40 remove() 删除模型组件

从模型中删除变量。

`OPTVModel.`**`remove`**(OPTVVar *var*)

该函数将变量从模型中删除，对应的索引会被相应地调整。例如，若下列模型 `model` 构建时包含以下变量

变量	索引
x0	0
x1	1
x2	2
x3	3

调用 `model.Remove(x0)` 后，相应变化为

变量	索引
x1	0
x2	1
x3	2

参数

- **var** –指定要删除的变量。

返回
无。

返回类型
void

抛出

- *OPTVErrCode.NOT_IN_MODEL* –模型不包含此变量。
- *OPTVErrCode.DATA_NOT_AVAILABLE* –该变量属于其他模型，而不属于当前模型。
- *OPTVErrCode.INDEX_OUT_OF_RANGE* –变量的下标超出了模型的变量总数，可能需要执行模型更新。

从模型中删除约束。

`OPTVModel.`**`remove`**`(`OPTVConstr *constr*`)`

该函数将约束从模型中删除，对应的索引会被相应地调整。例如，若下列模型 `model` 构建时包含以下约束

约束	索引
c0	0
c1	1
c2	2
c3	3

调用 `model.Remove(c0)` 后，相应变化为

约束	索引
c1	0
c2	1
c3	2

参数

- **constr** –指定要删除的约束。

返回
无。

返回类型
void

抛出

- *OPTVErrCode.NOT_IN_MODEL* –模型不包含此约束。
- *OPTVErrCode.DATA_NOT_AVAILABLE* –该约束属于其他模型，而不属于当前模型。
- *OPTVErrCode.INDEX_OUT_OF_RANGE* –约束的下标超出了模型的变量总数，可能需要执行模型更新。

`OPTVModel.`**`remove`**`(`OPTVGenConstr *genConstr*`)`

从模型中删除单个一般约束。执行后对应的相同类型的一般约束索引会被相应地调整。例如，若下列模型 `model` 构建时包含以下一般约束：

一般约束	类型	索引
and0	AND	0
and1	AND	1
or0	OR	0
or1	OR	1

调用 `model.remove(and0)` 后，相应变化为

一般约束	类型	索引
and1	AND	0
or0	OR	0
or1	OR	1

参数

- **`genConstr`** –指定要删除的一般约束。

返回
无。

返回类型
void

抛出

- *OPTVErrCode.NOT_IN_MODEL* –模型不包含此一般约束。
- *OPTVErrCode.DATA_NOT_AVAILABLE* –该一般约束属于其他模型，而不属于当前模型。
- *OPTVErrCode.INDEX_OUT_OF_RANGE* –该一般约束的下标超出了模型的变量总数，可能需要执行模型更新。

OPTVModel.**remove**(OPTVSOS *sos*)

从模型中删除单个 SOS 约束。与其他约束类型不同的是，删除该 SOS 约束后，相同类型的约束的索引**不会**改变。例如，若模型 model 创建时包含如下约束：

约束	类型	索引
sos1_1	1	0
sos1_2	1	1
sos1_3	1	2
sos2_1	2	0

调用 model.remove(sos1_2) 后，相应变化为

约束	类型	索引
sos1_1	1	0
sos1_3	1	2
sos2_1	2	0

参数

- **sos** –指定要删除的 SOS 约束。

返回
无。

返回类型
void

抛出

- *OPTVErrCode.NOT_IN_MODEL* –SOS 约束不属于任何模型。
- *OPTVErrCode.DATA_NOT_AVAILABLE* –SOS 不属于当前模型。
- *OPTVErrCode.INDEX_OUT_OF_RANGE* –SOS 约束的索引值超出了模型的该类型的 SOS 约束总数，可能需要对当前模型执行更新。

8.14.41 relax() 松弛模型

进行线性规划 (LP) 松弛。

OPTVModel.**relax**()

该函数将原模型中的所有整数类型变量替换为连续类型变量，生成和返回一个新的*OPTVModel* 对象，对原模型不做任何修改。

返回
模型的 LP 松弛副本。

返回类型
OPTVModel

8.14.42 reset() 重置模型

将模型重置为 *未求解成功*状态。

`OPTVModel.`**`reset`**(boolean *clearAll=false*)

该函数会清除内部解的信息，默认情况下，包括解的值、LP 基和模型状态。`clearAll` 标志是否清除额外的模型属性值（包括 MIP 热启动值，MIP 分支优先度，LP 的禁用属性状态）。

参数

- **`clearAll`** –清除额外信息（默认为 `false`）.

返回
无。

返回类型
void

8.14.43 getObjSA() 目标函数敏感度分析

目标函数敏感度分析，关于目标函数敏感度分析细节，请参考*LP 敏感度分析快速入门*。

`OPTVModel.`**`getObjSA`**(double[] *down*, double[] *up*, int[] *list*)

参数

- **`down`** –敏感度区间左端点。
- **`up`** –敏感度区间右端点。
- **`list`** –需要进行目标函数敏感度分析的变量索引集合。

返回
敏感度分析是否成功。

返回类型
boolean

返回值

- **`false`** –敏感度分析失败。
- **`true`** –敏感度分析成功。

8.14.44 getVarLbSA() 变量下界敏感度分析

变量下界敏感度分析，有关变量下界敏感度分析细节，请参考*LP 敏感度分析快速入门*。

`OPTVModel.`**`getVarLbSA`**(double[] *down*, double[] *up*, int[] *list*)

参数

- **`down`** –敏感度区间左端点。
- **`up`** –敏感度区间右端点。
- **`list`** –需要进行变量下界敏感度分析的约束索引集合。

返回
敏感度分析是否成功。

返回类型
boolean

返回值

- **false** –敏感度分析失败。
- **true** –敏感度分析成功。

8.14.45 getVarUbSA() 变量上界敏感度分析

变量上界敏感度分析，有关变量上界敏感度分析细节，请参考*LP 敏感度分析快速入门*。

OPTVModel.**getVarUbSA**(double[] *down*, double[] *up*, int[] *list*)

参数

- **down** –敏感度区间左端点。
- **up** –敏感度区间右端点。
- **list** –需要进行变量上界敏感度分析的约束索引集合。

返回
敏感度分析是否成功。

返回类型
boolean

返回值

- **false** –敏感度分析失败。
- **true** –敏感度分析成功。

8.14.46 getConstrLbSA() 约束左端值敏感度分析

约束左端值/下界敏感度分析，有关约束敏感度分析细节，请参考*LP 敏感度分析快速入门*。

OPTVModel.**getConstrLbSA**(double[] *down*, double[] *up*, int[] *list*)

参数

- **down** –敏感度区间左端点。
- **up** –敏感度区间右端点。
- **list** –需要进行左端值敏感度分析的约束索引集合。

返回
敏感度分析是否成功。

返回类型
boolean

返回值

- **false** –敏感度分析失败。
- **true** –敏感度分析成功。

8.14.47 getConstrUbSA() 约束右端值敏感度分析

约束右端值/上界敏感度分析，有关约束敏感度分析细节，请参考*LP 敏感度分析快速入门*。

`OPTVModel.getConstrUbSA`(double[] *down*, double[] *up*, int[] *list*)

参数

- **down** –敏感度区间左端点。
- **up** –敏感度区间右端点。
- **list** –需要进行右端值敏感度分析的约束索引集合。

返回
敏感度分析是否成功。

返回类型
boolean

返回值

- **false** –敏感度分析失败。
- **true** –敏感度分析成功。

8.14.48 setCoef() 设置系数

设置给定约束中指定变量的系数。

`OPTVModel.setCoef`(OPTVConstr *con*, OPTVVar *var*, double *newValue*)

参数

- **con** –需要修改的约束。
- **var** –需要修改系数的变量。
- **newValue** –新的系数值。

返回
无。

返回类型
void

抛出

- *OPTVErrCode.INVALID_ARGUMENT* –约束或变量不存在，或数值参数中存在 NaN.
- *OPTVErrCode.DATA_NOT_AVAILABLE* –数据不存在，或约束的线性表达式部分不可用。

8.14.49 setObjective() 设置目标函数

设置线性目标函数。

OPTVModel.**setObjective**(OPTVLinExpr *expr*, OPTVSense *sense=OPTVSense.MINIMIZE*)

该函数清除当前目标函数（相对于在其基础上进行叠加）。

参数

- **expr** –需要设定的目标函数线性表达式。
- **sense** – **可选**，最小化或最大化，默认值为:*OPTVSense.MINIMIZE*，参考*OPTVSense*.

返回

无。

返回类型

void

设置二次目标函数

OPTVModel.**setObjective**(OPTVQuadExpr *expr*, OPTVSense *sense=OPTVSense.MINIMIZE*)

同样，该函数清除当前目标函数（相对于在其基础上进行叠加）。若未指定 sense 参数，求解器会采用默认的.

参数

- **expr** –需要设定的目标函数二次表达式。
- **sense** – **可选**，最小化或最大化，默认值为:*OPTVSense.MINIMIZE*，参考*OPTVSense*.

返回

无。

返回类型

void

8.14.50 set() 设置参数和属性

设置参数

设置求解器的参数取值，一旦模型被创建，该接口仅提供控制此模型专用参数的接口，直到调用*OPTVModel.update()* 方法后，新修改才会在内部生效。可用参数请参考*OptVerse 参数*。

OPTVModel.**set**(OPTVIntParam *param*, int *value*)

参数

- **param** –整数型参数。
- **value** –新参数值。

返回

无。

返回类型

void

抛出
OPTVErrCode.UNKNOWN_PARAMETER –未知参数。

OPTVModel.**set**(OPTVDblParam *param*, double *value*)

参数

- **param** –双精度型参数。
- **value** –新参数值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_PARAMETER –未知参数。

OPTVModel.**set**(OPTVStrParam *param*, String *value*)

参数

- **param** –字符串型参数。
- **value** –新参数值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_PARAMETER –未知参数。

设置属性

设置求解器的属性值，同样，直到调用*OPTVModel.update()* 方法后，新修改才会在内部生效。可用属性请参考*OptVerse 属性*。

OPTVModel.**set**(OPTVIntAttr *attr*, int *value*)

参数

- **attr** –整数型属性。
- **value** –新属性值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知属性。

OPTVModel.**set**(OPTVDblAttr *attr*, double *value*)

参数

- **attr** –双精度型属性。
- **value** –新属性值。

返回
无。

返回类型
void

抛出
OPTVErrCode.UNKNOWN_ATTRIBUTE –未知属性。

8.14.51 update() 更新模型

执行模型更新。

OPTVModel.**update**()

该函数会进行模型更新，或 lazy 构建（如必要）。只有调用该函数 **后**，模型的更新才会生效。SDK 构建的模型需要调用此函数来保证内部模型的即时性。例如，用户调用*OPTVModel.optimize()* 前 **不需要**调用此函数，因为该函数在内部已被默认执行。

返回
无。

返回类型
void

8.14.52 writeIIS() IIS 结果导出

将不可约不一致子系统（IIS）写入文件（.ilp 格式），有关 IIS 的细节请参考*IIS 快速入门*。

OPTVModel.**writeIIS**(String *fileName*)

参数

- **fileName** –写出的文件名，必须包含 .ilp 后缀（例如 iis.ilp）.

返回
无。

返回类型
void

抛出
OPTVErrCode.ERROR_FILE_WRITE –不支持的文件格式，或者未计算出 IIS.

备注： 此函数 **不会**计算 IIS，若 IIS 未被计算，本函数将 throw 异常。

8.14.53 write() 写出模型和基

将模型或基写出到文件，支持的文件格式请参考文件格式。

OPTVModel.**write**(String *fileName*)

表 15: 支持写出的文件格式

.mps（暂不支持 QP 模型写出）	.lp	.bas	.ilp

表 16: 支持写出的压缩格式

.gz	.z	.Z

参数

- **fileName** –C++ 字符串文件名。

返回
无。

返回类型
void

抛出
OPTVErrCode.ERROR_FILE_WRITE –文件名或后缀为空或错误。

8.15 OptVerse 异常

class OPTVException
OptVerse 异常类。

OPTVException.**GetMessage() const**()
获取异常的错误信息。

OPTVException.**GetErrorCode() const**()
获取异常的错误码*OPTVErrCode*，关于错误码详细信息请参考*OptVerse 错误码*。

8.16 OptVerse 错误码

OPTVErrCode 用来产生特定的*OPTVException*异常对象，并给出相应的详细信息。

class OPTVErrCode
OptVerse 错误码

OPTVErrCode.**ERROR**
错误：错误信息，一般从求解器内部调用抛出。

OPTVErrCode.**WARNING**
警告：警告信息，一般从求解器内部调用抛出。

OPTVErrCode.**UNKNOWN**

未知：遇到未知错误。

OPTVErrCode.**INVALID_ARGUMENT**

无效参数：传入函数的参数不满足条件，例如在设置变量类型*OPTVCharAttr.VAR_TYPE*时，传入未定义的属性值，或者用 0 除线性表达式。

OPTVErrCode.**UNKNOWN_ATTRIBUTE**

未知属性：访问或者设置未定义的属性，例如*OPTVConstr.get(OPTVIntAttr.NUM_VARS)*将抛出此错误码，因为*OPTVIntAttr.NUM_VARS*不是模型的属性。

OPTVErrCode.**DATA_NOT_AVAILABLE**

数据不可用：访问的属性值不可用，例如在调用*OPTVModel.optimize()*之前，试图获取解的信息*OPTVVar.get(OPTVDblAttr.X)*将会抛出此错误。

OPTVErrCode.**INDEX_OUT_OF_RANGE**

下标越界：用于获取对象的索引值超出了允许范围，例如，当前模型仅有 2 个变量，调用*model.getVar(10)*会抛出此异常。

OPTVErrCode.**UNKNOWN_PARAMETER**

未知参数：访问未定义的参数。

OPTVErrCode.**VALUE_OUT_OF_RANGE**

数值越界：参数值超出了允许范围，例如尝试获取不存在的*OPTVSense*.

OPTVErrCode.**ERROR_FILE_READ**

文件读取失败：读取文件时发生错误。

OPTVErrCode.**ERROR_FILE_WRITE**

文件写出失败：写出文件时发生错误。

OPTVErrCode.**LICENSE_CHECK_FAILED**

许可检查失败：未发现有效许可，注：许可文件 OPTV_LICENSE_FILE 需要通过环境变量设置，细节请参考 安装说明。

OPTVErrCode.**INVALID_MODEL_INPUT**

模型输入无效：试图使用无效的模型输入信息。

OPTVErrCode.**NOT_IMPLEMENTED**

功能不可用：试图使用的功能暂未开放。

OPTVErrCode.**NOT_IN_MODEL**

模型组件不存在：访问的变量或约束在模型中不存在。

OPTVErrCode.**MATRIX_NOT_PSD**

非正半定矩阵：二次表达式的矩阵不满足正半定性质。当求解的模型 **包含**二次约束时，QCQP 求解器要求模型的二次目标函数或二次约束的矩阵都是正半定的。

CHAPTER 9

SDK 回放功能

OptVerse 的 SDK 功能支持创建”回放”文件，用户可以将一些列的 SDK 命令写入该文件。”回放”文件会压缩所有必须的输入文件（例如 `.mps` 格式数据），因而用户仅需上述”回放”文件即可”回放”所有相应的 API 调用。

使用这些文件和可执行文件 `optverse-replay`，用户即可实现”回放”。使用该功能，用户不仅可以轻松测试部署场景，还可以为开发者提供重要的技术支持。

9.1 Record 记录

只有设置*OPTVStrParam::REPLAY_FILE* 参数为非空时，相应的”回放”文件才会被创建；否则，不会创建任何”回放”文件（默认）.

与其他参数不同的是，该参数必须通过环境对象在*OPTVModel* 创建前设置，而 **不可以**通过*OPTVModel* 对象设置。

```
/**
 * This script illustrates how to generate a recording file.
 * This is accomplished through the `OPTVStrParam::REPLAY_FILE` parameter in the␣
↪OPTVEnv object.
 *
 * The recording file produced by this script can be rerun using optverse-replay.
 */
#include "optv_c++.h"
#include <iostream>

using namespace std;

int main()
{
    try {
        OPTVEnv env("lp1.log");
        env.Set(OPTVStrParam::REPLAY_FILE, "optv.rec");
```

(续下页)

(接上页)

```
        OPTVModel model(env);

        // add variables to the model, all variables are continuous
        OPTVVar x = model.AddVar(0, 1, -1, OPTV_CONTINUOUS, "x");
        OPTVVar y = model.AddVar(0, OPTV_INF, -14, OPTV_CONTINUOUS, "y");
        OPTVVar z = model.AddVar(0, 3, -6, OPTV_CONTINUOUS, "z");

        // add constraint to the model
        OPTVConstr c0 = model.AddConstr(2 * x + y + z, -OPTV_INF, 3, "c0");
        OPTVConstr c1 = model.AddConstr(3 * y + z, -OPTV_INF, 6, "c1");

        model.Optimize();
        model.Write("lp1.lp");

        if (model.Get(OPTVIntAttr::SOL_COUNT) > 0) {
            cout << "Best solution: " << model.Get(OPTVDblAttr::OBJ_VAL) << endl;

            cout << "x = " << x.Get(OPTVDblAttr::X) << endl;
            cout << "y = " << y.Get(OPTVDblAttr::X) << endl;
            cout << "z = " << z.Get(OPTVDblAttr::X) << endl;
        } else {
            cout << "No feasible solution available!" << endl;
        }

    } catch (const OPTVException& e) {
        cout << e.GetErrorCode() << ": " << e.GetMessage() << endl;
    } catch (...) {
        cerr << "Unknown exception" << endl;
    }
    return 0;
}
```

备注： 当前 OptVerse 仅支持单个环境关联单个”回放”应用。

9.2 Replay 回放

使用 OptVerse 的”回放”功能时，只需将”回放”文件的路径提供给可执行文件 `optverse-replay`，相应的 SDK 命令即被顺序执行。

备注： 当前 OptVerse 仅支持单个环境关联单个”回放”应用。

CHAPTER 10

网络流规划问题

网络流规划问题 (network programming) 是一类特殊的线性规划问题，本章将对这类问题进行简要说明。OptVerse 求解器可以自动识别网络流规划问题结构，从而使用更高效的算法（网络流单纯形法，network simplex method）进行求解。

10.1 问题定义

一个满足以下条件的线性规划问题被称为 **网络流规划问题**:

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & l \leq x \leq u \end{aligned}$$

这里

$$\begin{aligned} c &\in \mathbb{R}^m \\ b &\in \mathbb{R}^n \\ A &\in \{-1, 0, 1\}^{n \times m} \end{aligned}$$

而且问题满足下列条件:

- 系数矩阵 A 的每列只有一个 1 和 -1 (即其余系数都为零)
- $\sum_i b_i = 0, i \in \{1, \ldots, n\}$

这里, n 为约束数量, m 为变量数量。

10.2 网络流规划样例

- 目标函数为最小化:

$$x_1 + x_2 + x_3 + x_4$$

- 约束条件:

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ -5 \end{bmatrix}$$

- 变量边界:

$$\begin{aligned} 0 \leq x_1 \leq 4, \\ 0 \leq x_2 \leq 2, \\ 0 \leq x_3 \leq 4, \\ 0 \leq x_4 \leq 10 \end{aligned}$$