

杉数求解器用户手册

版本 *7.0.3*

上海杉数网络科技有限公司

葛冬冬　皇甫琦　王子卓　伍健　叶荫宇

2023 年 11 月 14 日

目录

1 杉数求解器简介 **1**

1.1 概述 . . . 1
1.2 许可类型 . . . 2
1.3 如何引用 . . . 3
1.4 联系信息 . . . 4

2 安装说明 **5**

2.1 软件申请与下载 . . . 5
2.2 软件安装 . . . 5
2.2.1 Windows . . . 5
2.2.2 MacOS . . . 8
2.2.3 Linux . . . 12
2.3 配置许可文件 . . . 14
2.3.1 获取许可 . . . 14
2.3.2 验证许可 . . . 15
2.3.3 安装许可 . . . 15
2.3.4 其他 . . . 19
2.4 验证安装配置完成 . . . 19
2.5 升级和更新 . . . 20

3 COPT 交互式命令行工具 **23**

3.1 概述 . . . 23
3.2 交互模式 . . . 23
3.3 脚本模式 . . . 24
3.4 工具命令 . . . 25
3.4.1 普通工具命令 . . . 25
3.4.2 COPT 工具命令 . . . 26
3.5 使用示例 . . . 29

4 COPT 浮动授权服务 **33**

4.1 服务器端安装说明 . . . 33
4.1.1 安装步骤 . . . 33
4.1.2 许可文件 . . . 34
4.1.3 配置文件 . . . 35
4.1.4 使用示例 . . . 36
4.2 客户端使用说明 . . . 36

4.2.1 安装和配置 . . . 37
4.2.2 使用示例 . . . 37
4.3 浮动授权服务器管理工具 . . . 39
4.3.1 工具说明 . . . 39
4.3.2 使用示例 . . . 40
4.4 服务方式启动 . . . 41
4.4.1 Linux 系统 . . . 42
4.4.2 MacOS 系统 . . . 44

5 COPT 计算集群服务 47
5.1 服务器端安装说明 . . . 47
5.1.1 安装步骤 . . . 47
5.1.2 许可文件 . . . 48
5.1.3 配置文件 . . . 49
5.1.4 使用示例 . . . 51
5.2 客户端使用说明 . . . 52
5.2.1 安装和配置 . . . 52
5.2.2 使用示例 . . . 53
5.3 集群服务器管理工具 . . . 54
5.3.1 工具说明 . . . 55
5.3.2 使用示例 . . . 56
5.4 服务方式启动 . . . 58
5.4.1 Linux 系统 . . . 59
5.4.2 MacOS 系统 . . . 60

6 COPT 在线授权服务 63

7 COPT 快速入门 65
7.1 C 接口 . . . 65
7.1.1 示例解析 . . . 65
7.1.2 编译与运行 . . . 74
7.2 C++ 接口 . . . 74
7.2.1 示例解析 . . . 75
7.2.2 编译与运行 . . . 79
7.3 C# 接口 . . . 80
7.3.1 示例解析 . . . 80
7.3.2 编译与运行 . . . 85
7.4 Java 接口 . . . 86
7.4.1 示例解析 . . . 86
7.4.2 编译与运行 . . . 91
7.5 Python 接口 . . . 92
7.5.1 安装说明 . . . 92
7.5.2 示例解析 . . . 94
7.5.3 最佳实践 . . . 99
7.6 AMPL 接口 . . . 100
7.6.1 安装说明 . . . 100

7.6.2 求解参数与返回值 . . . 101
7.6.3 使用示例 . . . 103
7.7 Pyomo 接口 . . . 106
7.7.1 安装说明 . . . 107
7.7.2 使用示例 . . . 107
7.8 PuLP 接口 . . . 114
7.8.1 安装说明 . . . 114
7.8.2 配置 PuLP 接口 . . . 114
7.8.3 功能介绍 . . . 115
7.9 CVXPY 接口 . . . 116
7.9.1 安装说明 . . . 116
7.9.2 配置 CVXPY 接口 . . . 117
7.9.3 功能介绍 . . . 117

8 一般常数 **119**
8.1 版本信息 . . . 119
8.2 优化方向 . . . 119
8.3 无边界和未定义 . . . 120
8.4 约束类型 . . . 120
8.5 变量类型 . . . 121
8.6 SOS 约束类型 . . . 121
8.7 二阶锥约束类型 . . . 121
8.8 基状态 . . . 122
8.9 解状态 . . . 122
8.10 回调函数的触发点 . . . 123
8.11 客户端配置参数 . . . 123
8.12 访问常数相关操作 . . . 123

9 属性 **125**
9.1 优化模型相关属性 . . . 125
9.2 求解结果相关属性 . . . 128
9.3 属性获取方式 . . . 131

10 信息 **133**
10.1 模型相关信息 . . . 134
10.2 求解结果相关信息 . . . 134
10.3 对偶 Farkas 和主元射线 . . . 135
10.4 可行化松弛结果相关信息 . . . 135
10.5 Callback 相关信息 . . . 136
10.6 信息获取方式 . . . 136

11 参数 **137**
11.1 限制和容差 . . . 137
11.2 预求解相关 . . . 140
11.3 线性规划相关 . . . 141
11.4 整数规划相关 . . . 143

11.5 半定规划相关 . . . 148
11.6 并行计算相关 . . . 148
11.7 IIS 计算相关 . . . 150
11.8 可行化松弛计算相关 . . . 150
11.9 参数调优相关 . . . 151
11.10 Callback 相关 . . . 153
11.11 其它参数 . . . 153
11.12 参数相关操作 . . . 154

12 不同类型优化问题建模求解 155
12.1 线性规划（LP） . . . 156
12.1.1 数学模型 . . . 156
12.1.2 建模 . . . 157
12.1.3 求解 . . . 159
12.1.4 相关属性和信息 . . . 159
12.2 二阶锥规划（SOCP） . . . 160
12.2.1 数学模型 . . . 160
12.2.2 建模 . . . 160
12.3 半定规划（SDP） . . . 163
12.3.1 数学模型 . . . 163
12.3.2 建模 . . . 164
12.3.3 求解 . . . 168
12.3.4 相关属性 . . . 168
12.4 二次规划（QP） . . . 168
12.4.1 数学模型 . . . 168
12.5 二次约束规划（QCP） . . . 169
12.5.1 数学模型 . . . 169
12.5.2 建模 . . . 170
12.5.3 相关属性 . . . 172
12.6 混合整数规划（MIP） . . . 172
12.6.1 建模 . . . 172
12.6.2 求解 . . . 173
12.6.3 相关属性 . . . 173
12.7 特殊约束 . . . 173
12.7.1 SOS 约束 . . . 174
12.7.2 Indicator 约束 . . . 175
12.7.3 特殊约束相关属性 . . . 177
12.7.4 特殊约束的 IIS 状态 . . . 177

13 不可行模型的处理 179
13.1 不可行模型的 IIS . . . 179
13.1.1 计算 IIS . . . 180
13.1.2 获取变量和约束的 IIS 状态 . . . 181
13.1.3 IIS 相关参数、属性和信息 . . . 183
13.2 不可行模型的可行化松弛 . . . 184

13.2.1 计算可行化松弛 . . . 184
13.2.2 可行化松弛的相关参数、属性和信息 . . . 185

14 整数规划初始解功能 187
14.1 相关函数 . . . 187
14.1.1 设置并加载初始解 . . . 187
14.1.2 初始解的读入/写入 . . . 188
14.2 相关参数 . . . 188
14.3 初始解日志输出 . . . 189
14.3.1 初始解被接受的情况 . . . 189
14.3.2 初始解被拒绝的情况 . . . 189

15 整数规划解池功能 191

16 COPT 调优工具 193
16.1 调优工具基本介绍 . . . 193
16.2 调优工具相关参数 . . . 193
16.3 参数调优功能 . . . 194
16.3.1 调优方法 . . . 194
16.3.2 调优模式 . . . 194
16.3.3 调优扰动次数 . . . 194
16.3.4 调优准则 . . . 195
16.3.5 调优目标 . . . 195
16.3.6 调优日志输出 . . . 195
16.3.7 调优时间限制 . . . 195
16.3.8 用户自定义部分 . . . 195
16.3.9 获取调优结果 . . . 196
16.4 参数调优示例 . . . 196

17 Callbacks 功能 197
17.1 获取 MIP 求解过程中间信息 . . . 197
17.2 控制 MIP 求解进程 . . . 198
17.2.1 添加惰性约束 . . . 198
17.2.2 添加割平面 . . . 199
17.2.3 设置自定义的可行解 . . . 199
17.3 不同 API 中使用 Callback 功能 . . . 200

18 矩阵建模方式 203
18.1 多维变量 . . . 203
18.2 多维数组运算及表达式 . . . 204
18.2.1 多维线性表达式 . . . 204
18.2.2 多维二次表达式 . . . 204
18.2.3 其他多维数组运算 . . . 204
18.3 矩阵约束 . . . 205
18.3.1 矩阵线性约束 . . . 205
18.3.2 二次约束 . . . 206
18.4 由多维变量构成的目标函数 . . . 206

19 求解日志 **209**
19.1 求解日志相关参数和函数 . . . 209
19.2 求解日志基础信息部分 . . . 210
19.3 单纯形法（Simplex）求解日志 . . . 210
19.3.1 预求解（Presolve） . . . 211
19.3.2 单纯形法求解过程 . . . 211
19.3.3 求解结果汇总 . . . 212
19.4 内点法（Barrier）求解日志 . . . 212
19.4.1 预求解（Presolve） . . . 212
19.4.2 模型信息部分 . . . 213
19.4.3 内点法（Barrier）求解过程 . . . 213
19.4.4 内点法（Barrier）求解总结 . . . 214
19.4.5 求解结果汇总 . . . 214
19.5 分支切割法（Branch-and-Cut）求解日志 . . . 215
19.5.1 预求解（Presolve） . . . 215
19.5.2 求解过程 . . . 216
19.5.3 求解结果汇总 . . . 217

20 文件格式 **219**
20.1 文件格式一览 . . . 219
20.2 文件读写操作 . . . 219
20.3 模型文件介绍 . . . 220

21 常见问题 **223**
21.1 安装和许可配置相关 . . . 223
21.1.1 MacOS 系统 . . . 224
21.1.2 Windows 系统 . . . 224
21.2 建模求解和功能使用相关 . . . 225

22 C API 参考手册 **227**
22.1 常量 . . . 227
22.1.1 优化方向 . . . 227
22.1.2 无边界 . . . 227
22.1.3 未定义 . . . 228
22.1.4 约束类型 . . . 228
22.1.5 变量类型 . . . 228
22.1.6 SOS 约束类型 . . . 229
22.1.7 Indicator 约束 . . . 229
22.1.8 二阶锥约束 . . . 229
22.1.9 基状态 . . . 230
22.1.10 LP 的解状态 . . . 230
22.1.11 MIP 的解状态 . . . 231
22.1.12 回调函数的触发点 . . . 232
22.1.13 API 函数的返回值 . . . 232
22.1.14 客户端配置参数 . . . 233
22.1.15 其他常量 . . . 233

22.2 属性 . . . 233
22.2.1 优化模型相关属性 . . . 233
22.2.2 求解结果相关属性 . . . 235
22.3 信息 . . . 238
22.3.1 模型相关信息 . . . 238
22.3.2 求解结果相关信息 . . . 238
22.3.3 对偶 Farkas 和主元射线 . . . 239
22.3.4 可行化松弛结果相关信息 . . . 239
22.4 Callback 相关信息 . . . 240
22.5 参数 . . . 241
22.5.1 限制和容差 . . . 241
22.5.2 预求解相关 . . . 243
22.5.3 线性规划相关 . . . 244
22.5.4 半定规划相关 . . . 246
22.5.5 整数规划相关 . . . 246
22.5.6 并行计算相关 . . . 250
22.5.7 IIS 计算相关 . . . 251
22.5.8 可行化松弛计算相关 . . . 251
22.5.9 参数调优相关 . . . 252
22.5.10 回调函数相关 . . . 254
22.5.11 其它参数 . . . 254
22.6 API 函数 . . . 255
22.6.1 创建求解环境和模型 . . . 255
22.6.2 构造和修改模型 . . . 259
22.6.3 读入与输出模型 . . . 287
22.6.4 求解和获取解 . . . 292
22.6.5 获取模型信息 . . . 298
22.6.6 设置和获取参数 . . . 324
22.6.7 获取属性 . . . 329
22.6.8 日志函数 . . . 330
22.6.9 整数规划初始解功能函数 . . . 330
22.6.10 不可行模型 IIS 计算功能函数 . . . 332
22.6.11 可行化松弛计算功能函数 . . . 335
22.6.12 参数调优功能函数 . . . 336
22.6.13 回调功能函数 . . . 338
22.6.14 其他 API 函数 . . . 349

23 Python API 参考手册 **351**
23.1 COPT 常数类 . . . 351
23.1.1 一般常数 . . . 351
23.1.2 属性 . . . 351
23.1.3 信息 . . . 351
23.1.4 Callback 信息 . . . 352
23.1.5 参数 . . . 352
23.2 优化建模类 . . . 352

23.2.1 EnvrConfig 类 . . . 352
23.2.2 Envr 类 . . . 353
23.2.3 Model 类 . . . 355
23.2.4 Var 类 . . . 424
23.2.5 VarArray 类 . . . 428
23.2.6 PsdVar 类 . . . 430
23.2.7 PsdVarArray 类 . . . 433
23.2.8 SymMatrix 类 . . . 435
23.2.9 SymMatrixArray 类 . . . 436
23.2.10 Constraint 类 . . . 437
23.2.11 ConstrArray 类 . . . 441
23.2.12 ConstrBuilder 类 . . . 443
23.2.13 ConstrBuilderArray 类 . . . 444
23.2.14 QConstraint 类 . . . 446
23.2.15 QConstrArray 类 . . . 449
23.2.16 QConstrBuilder 类 . . . 451
23.2.17 QConstrBuilderArray 类 . . . 453
23.2.18 PsdConstraint 类 . . . 455
23.2.19 PsdConstrArray 类 . . . 457
23.2.20 PsdConstrBuilder 类 . . . 459
23.2.21 PsdConstrBuilderArray 类 . . . 461
23.2.22 LmiConstraint 类 . . . 463
23.2.23 LmiConstrArray 类 . . . 467
23.2.24 SOS 类 . . . 469
23.2.25 SOSArray 类 . . . 470
23.2.26 SOSBuilder 类 . . . 472
23.2.27 SOSBuilderArray 类 . . . 475
23.2.28 Cone 类 . . . 477
23.2.29 ConeArray 类 . . . 477
23.2.30 ConeBuilder 类 . . . 479
23.2.31 ConeBuilderArray 类 . . . 482
23.2.32 GenConstr 类 . . . 483
23.2.33 GenConstrArray 类 . . . 484
23.2.34 GenConstrBuilder 类 . . . 486
23.2.35 GenConstrBuilderArray 类 . . . 488
23.2.36 Column 类 . . . 490
23.2.37 ColumnArray 类 . . . 495
23.2.38 MVar 类 . . . 497
23.2.39 MConstr 类 . . . 503
23.2.40 MConstrBuilder 类 . . . 507
23.2.41 MQConstrBuilder 类 . . . 508
23.2.42 MLinExpr 类 . . . 509
23.2.43 MQuadExpr 类 . . . 515
23.2.44 ExprBuilder 类 . . . 522
23.2.45 LinExpr 类 . . . 526

23.2.46 QuadExpr 类 . . . 532
23.2.47 PsdExpr 类 . . . 540
23.2.48 LmiExpr 类 . . . 547
23.2.49 CallbackBase 类 . . . 552
23.2.50 GenConstrX 类 . . . 559
23.2.51 CoptError 类 . . . 559
23.3 辅助函数与工具类 . . . 560
23.3.1 辅助函数 . . . 560
23.3.2 tuplelist 类 . . . 561
23.3.3 tupledict 类 . . . 562
23.3.4 ProbBuffer 类 . . . 564

24 C++ API 参考手册 **567**
24.1 常量 . . . 567
24.2 属性 . . . 567
24.3 参数 . . . 567
24.4 C++ 优化建模类 . . . 568
24.4.1 Envr 类 . . . 568
24.4.2 EnvrConfig 类 . . . 569
24.4.3 Model 类 . . . 570
24.4.4 Var 类 . . . 635
24.4.5 VarArray 类 . . . 637
24.4.6 Expr 类 . . . 638
24.4.7 Constraint 类 . . . 645
24.4.8 ConstrArray 类 . . . 647
24.4.9 ConstrBuilder 类 . . . 648
24.4.10 ConstrBuilderArray 类 . . . 650
24.4.11 Column 类 . . . 651
24.4.12 ColumnArray 类 . . . 654
24.4.13 Sos 类 . . . 655
24.4.14 SosArray 类 . . . 656
24.4.15 SosBuilder 类 . . . 657
24.4.16 SosBuilderArray 类 . . . 659
24.4.17 GenConstr 类 . . . 660
24.4.18 GenConstrArray 类 . . . 660
24.4.19 GenConstrBuilder 类 . . . 661
24.4.20 GenConstrBuilderArray 类 . . . 663
24.4.21 Cone 类 . . . 664
24.4.22 ConeArray 类 . . . 665
24.4.23 ConeBuilder 类 . . . 666
24.4.24 ConeBuilderArray 类 . . . 667
24.4.25 QuadExpr 类 . . . 668
24.4.26 QConstraint 类 . . . 676
24.4.27 QConstrArray 类 . . . 679
24.4.28 QConstrBuilder 类 . . . 680

24.4.29 QConstrBuilderArray 类 . . . 681
24.4.30 PsdVar 类 . . . 682
24.4.31 PsdVarArray 类 . . . 684
24.4.32 PsdExpr 类 . . . 685
24.4.33 PsdConstraint 类 . . . 693
24.4.34 PsdConstrArray 类 . . . 695
24.4.35 PsdConstrBuilder 类 . . . 696
24.4.36 PsdConstrBuilderArray 类 . . . 698
24.4.37 LmiConstraint 类 . . . 699
24.4.38 LmiConstrArray 类 . . . 701
24.4.39 LmiExpr 类 . . . 702
24.4.40 SymMatrix 类 . . . 709
24.4.41 SymMatrixArray 类 . . . 709
24.4.42 SymMatExpr 类 . . . 710
24.4.43 CallbackBase 类 . . . 716
24.4.44 ProbBuffer 类 . . . 723

25 C# API 参考手册 725

25.1 C# 常量类 . . . 725
25.1.1 一般常数 . . . 725
25.1.2 解状态 . . . 725
25.1.3 属性 . . . 725
25.1.4 信息 . . . 726
25.1.5 Callback 信息 . . . 726
25.1.6 参数 . . . 726
25.2 C# 优化建模类 . . . 726
25.2.1 Envr 类 . . . 726
25.2.2 EnvrConfig 类 . . . 728
25.2.3 Model 类 . . . 728
25.2.4 Var 类 . . . 797
25.2.5 VarArray 类 . . . 800
25.2.6 Expr 类 . . . 801
25.2.7 Constraint 类 . . . 806
25.2.8 ConstrArray 类 . . . 808
25.2.9 ConstrBuilder 类 . . . 809
25.2.10 ConstrBuilderArray 类 . . . 811
25.2.11 Column 类 . . . 812
25.2.12 ColumnArray 类 . . . 815
25.2.13 Sos 类 . . . 816
25.2.14 SosArray 类 . . . 817
25.2.15 SosBuilder 类 . . . 818
25.2.16 SosBuilderArray 类 . . . 821
25.2.17 GenConstr 类 . . . 822
25.2.18 GenConstrArray 类 . . . 823
25.2.19 GenConstrBuilder 类 . . . 824

25.2.20 GenConstrBuilderArray 类 . . . 825
25.2.21 Cone 类 . . . 826
25.2.22 ConeArray 类 . . . 827
25.2.23 ConeBuilder 类 . . . 828
25.2.24 ConeBuilderArray 类 . . . 829
25.2.25 QuadExpr 类 . . . 831
25.2.26 QConstraint 类 . . . 839
25.2.27 QConstrArray 类 . . . 841
25.2.28 QConstrBuilder 类 . . . 842
25.2.29 QConstrBuilderArray 类 . . . 843
25.2.30 PsdVar 类 . . . 844
25.2.31 PsdVarArray 类 . . . 846
25.2.32 PsdExpr 类 . . . 847
25.2.33 PsdConstraint 类 . . . 855
25.2.34 PsdConstrArray 类 . . . 857
25.2.35 PsdConstrBuilder 类 . . . 858
25.2.36 PsdConstrBuilderArray 类 . . . 860
25.2.37 LmiConstraint 类 . . . 861
25.2.38 LmiConstrArray 类 . . . 862
25.2.39 LmiExpr 类 . . . 864
25.2.40 SymMatrix 类 . . . 869
25.2.41 SymMatrixArray 类 . . . 870
25.2.42 SymMatExpr 类 . . . 871
25.2.43 CallbackBase 类 . . . 875
25.2.44 ProbBuffer 类 . . . 884
25.2.45 CoptException 类 . . . 885

26 Java API 参考手册 **887**
26.1 Java 常量类 . . . 887
26.1.1 一般常数 . . . 887
26.1.2 解状态 . . . 887
26.1.3 属性 . . . 887
26.1.4 信息 . . . 888
26.1.5 Callback 信息 . . . 888
26.1.6 参数 . . . 888
26.2 Java 建模类 . . . 888
26.2.1 Envr 类 . . . 888
26.2.2 EnvrConfig 类 . . . 890
26.2.3 Model 类 . . . 890
26.2.4 Var 类 . . . 958
26.2.5 VarArray 类 . . . 961
26.2.6 Expr 类 . . . 962
26.2.7 Constraint 类 . . . 968
26.2.8 ConstraArray 类 . . . 970
26.2.9 ConstraBuilder 类 . . . 971

26.2.10 ConstrBuilderArray 类 . . . 973
26.2.11 Column 类 . . . 974
26.2.12 ColumnArray 类 . . . 977
26.2.13 Sos 类 . . . 979
26.2.14 SosArray 类 . . . 979
26.2.15 SosBuilder 类 . . . 981
26.2.16 SosBuilderArray 类 . . . 983
26.2.17 GenConstr 类 . . . 984
26.2.18 GenConstrArray 类 . . . 985
26.2.19 GenConstrBuilder 类 . . . 986
26.2.20 GenConstrBuilderArray 类 . . . 987
26.2.21 Cone 类 . . . 988
26.2.22 ConeArray 类 . . . 989
26.2.23 ConeBuilder 类 . . . 990
26.2.24 ConeBuilderArray 类 . . . 991
26.2.25 QuadExpr 类 . . . 993
26.2.26 QConstraint 类 . . . 1001
26.2.27 QConstrArray 类 . . . 1003
26.2.28 QConstrBuilder 类 . . . 1005
26.2.29 QConstrBuilderArray 类 . . . 1006
26.2.30 PsdVar 类 . . . 1007
26.2.31 PsdVarArray 类 . . . 1008
26.2.32 PsdExpr 类 . . . 1009
26.2.33 PsdConstraint 类 . . . 1018
26.2.34 PsdConstrArray 类 . . . 1019
26.2.35 PsdConstrBuilder 类 . . . 1020
26.2.36 PsdConstrBuilderArray 类 . . . 1022
26.2.37 LmiConstraint 类 . . . 1023
26.2.38 LmiConstrArray 类 . . . 1025
26.2.39 LmiExpr 类 . . . 1026
26.2.40 SymMatrix 类 . . . 1031
26.2.41 SymMatrixArray 类 . . . 1032
26.2.42 SymMatExpr 类 . . . 1033
26.2.43 CallbackBase 类 . . . 1038
26.2.44 ProbBuffer 类 . . . 1046
26.2.45 CoptException 类 . . . 1047

第 1 章　杉数求解器简介

杉数求解器是一款针对大规模优化问题的高效数学规划求解器。本文档简要介绍了杉数求解器的相关内容，包括：

- 如何安装杉数求解器
- 如何设置许可文件
- 如何使用杉数求解器交互式命令行工具

我们推荐所有用户在使用杉数求解器之前仔细阅读上述前两节内容。

用户在正确安装杉数求解器和配置许可后，如果想通过已有模型对杉数求解器进行求解与测试，我们推荐用户仔细阅读*COPT 交互式命令行工具* 章节。如果用户有偏爱的编程语言并希望使用编程接口调用杉数求解器，可以从目前支持的以下几种接口中进行选择：

- *C 接口*
- *C++ 接口*
- *C# 接口*
- *Java 接口*
- *Python 接口*
- *AMPL 接口*
- *Pyomo 接口*
- *PuLP 接口*
- *CVXPY 接口*

1.1 概述

杉数求解器目前支持求解线性规划 (LP) 问题、二阶锥规划 (SOCP) 问题、凸二次规划（QP）问题、凸二次约束规划（QCP）问题、半定规划（SDP）问题、混合整数线性规划（MILP）、混合整数二阶锥规划（MISOCP）、混合整数凸二次规划 (MIQP)、混合整数凸二次约束规划（MIQCP）问题。在以后发布的版本中会逐渐支持求解更多类型的优化问题。目前支持求解的问题类型与相应的算法如下表所示：

表 1.1: 问题类型与求解算法

问题类型	求解算法
线性规划 (LP)	对偶单纯形法、内点法
二阶锥规划 (SOCP)	内点法
凸二次规划 (QP)	内点法
凸二次约束规划 (QCP)	内点法
半定规划 (SDP)	内点法、交替乘子下降法
混合整数线性规划 (MILP)	分支切割法
混合整数二阶锥规划 (MISOCP)	分支切割法
混合整数凸二次规划 (MIQP)	分支切割法
混合整数凸二次约束规划 (MIQCP)	分支切割法

杉数求解器支持所有主流的操作系统（均为 64 位系统），包括：Windows、Linux（包括 ARM64 平台）和 MacOS（包括 ARM64 平台），目前主要提供了以下编程接口：

- C 接口
- C++ 接口
- C# 接口
- Java 接口
- Python 接口
- AMPL 接口
- GAMS 接口
- AIMMS 接口
- Julia 接口
- Pyomo 接口
- PuLP 接口
- CVXPY 接口

我们将在以后发布的版本中提供更多的编程接口以满足不同场景下用户的不同需要。

1.2 许可类型

目前，我们提供了 4 种许可类型，分别为：个人许可、服务器许可、浮动许可和集群许可。各种许可的功能如表 表 1.2 所示：

表 1.2: 许可类型

许可类型	功能
个人许可	该许可绑定个人电脑的用户账号，供单个用户使用。求解运算在本地，不限制 CPU 核数和线程数。
服务器许可	该许可绑定单台服务器的 MAC 地址和 CPUID，在该服务器上可并发求解多个优化任务。求解运算在绑定的服务器上进行，不限制用户数量，不限制 CPU 核数。
浮动许可	该许可绑定单台机器的 MAC 地址和 CPUID 作为浮动许可服务器，连接到该服务器的客户机都可以完成许可的获取与释放，可同时进行优化计算的客户机上限为浮动许可的令牌数量。求解运算在客户机上进行，每个进程需要一个令牌，允许多任务并发求解。
集群许可	该许可绑定服务器的 MAC 地址和 CPUID，通过一台或多台服务器搭建求解集群进行集中求解，其它客户机可以通过局域网将求解任务发送到集群来实现远程求解功能，集群中的每个服务器允许多任务并发求解，但每个客户进程只能顺序递交任务，不能并发。不限制 CPU 核数。

1.3 如何引用

如果您在研究工作中使用了杉数求解器 COPT，请您在论文中引用我们。例如：

- 我们研究中使用了 COPT [1]
- 为了求解混合整数规划问题，我们使用了 Cardinal Optimizer [1]

相应的引用条目为：

```
[1] D. Ge, Q. Huangfu, Z. Wang, J. Wu and Y. Ye. Cardinal Optimizer (COPT) user guide. https://
↪guide.coap.online/copt/en-doc, 2023.
```

相应的 BibTex 引用为：

```
@misc{copt,
  author={Dongdong Ge and Qi Huangfu and Zizhuo Wang and Jian Wu and Yinyu Ye},
  title={Cardinal {O}ptimizer {(COPT)} user guide},
  howpublished={https://guide.coap.online/copt/en-doc},
  year=2023
}
```

1.4 联系信息

杉数求解器由 上海杉数网络科技有限公司 独立开发，用户可以通过 表 6.1 中的联系方式获取更多信息：

表 1.3: 联系信息

联系方式	具体信息	描述
官方网站	https://www.shanshu.ai	
联系电话	400-680-5680	
电子邮件	coptsales@shanshu.ai	商务支持
电子邮件	coptsupport@shanshu.ai	技术支持

第 2 章　安装说明

本章介绍了如何在所有支持的操作系统上安装杉数求解器，以及如何获取与配置许可文件的过程。我们推荐所有用户在使用杉数求解器之前仔细阅读本章内容。

2.1 软件申请与下载

使用杉数求解器之前，用户需要下载并安装杉数求解器到计算机中。如果还未下载软件，请用户访问软件 官方申请页面 按照说明进行申请。目前，网页申请是对个人许可的试用，申请信息除基本信息外，只需要填写计算机用户登录账号。

个人许可申请通过后，申请者邮箱会收到一封来自 coptsales@shanshu.ai 的邮件，邮件中会提供杉数求解器安装包的下载链接、许可文件、以及授权通过的密钥信息，该密钥信息与用户申请信息一一对应。用户需要根据本章内容安装杉数求解器并配置许可文件。

如果用户在使用杉数求解器时遇到任何问题，请联系 coptsupport@shanshu.ai 获取更多帮助。

2.2 软件安装

2.2.1 Windows

我们为 Windows 平台提供了两种安装包，用户 **二选其一即可**。一种是可执行安装程序（下载链接名称中含有 `installer`），另一种是 ZIP 格式压缩包。

可执行安装程序提供可视化安装提示窗口，并且会自动配置环境变量，用户只需按照指引，点击依次完成安装步骤即可；ZIP 格式安装包则需要用户先解压安装包，并手动配置环境变量。



图 2.1: Windows 系统两种不同安装包下载链接

我们 **推荐大部分用户下载使用可执行安装程序**，专家级用户可以下载 ZIP 格式压缩包。

可执行安装程序安装

对于下载可执行安装程序的用户，例如：适用于 64 位 Windows 系统的 COPT 7.0.2 版本软件 CardinalOptimizer-7.0.2-win64-installer.exe ，双击该安装程序并按照下述说明进行操作：

- 步骤一：双击运行安装程序，进入安装语言选择页面。默认语言为 `English` ，用户可以通过下拉菜单进行选择，详见 图 2.2 。本文选择安装语言为“简体中文”。

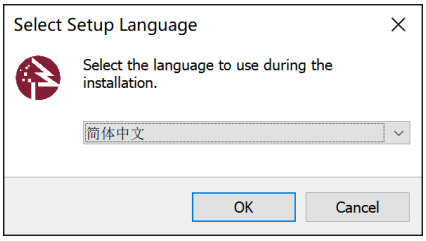


图 2.2: 安装语言选择页面

- 步骤二：如果用户同意“用户协议”，请选择“我授受协议”，然后选择“下一步”。如果用户不同意用户协议，并选择“取消”，则软件将退出安装，详见 图 2.3 。



图 2.3: 用户协议页面

- 步骤三：默认设置下，杉数求解器安装路径为：C:\Program Files\copt70，用户可以修改为任意路径。确认安装路径后，点击“下一步”，详见 图 2.4 。



图 2.4: 安装路径选择页面

- 步骤四：选择开始菜单文件夹，建议使用默认设置并点击“下一步”，详见 图 2.5 。

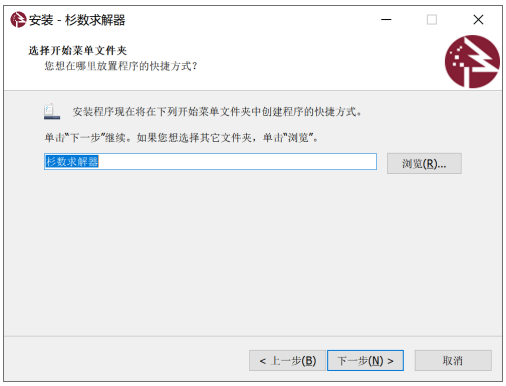


图 2.5: 选择开始菜单文件夹

- 步骤五：软件准备安装，点击“安装”，详见 图 2.6 。

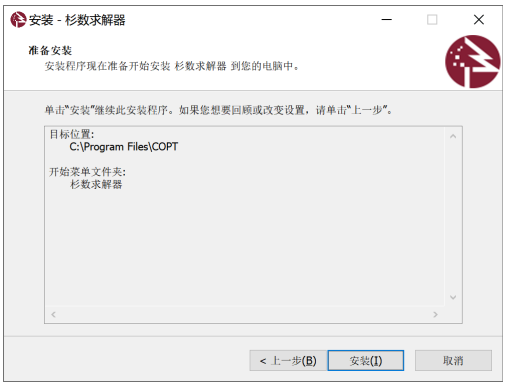


图 2.6: 软件准备安装

- 步骤六：安装过程中将自动配置必要的环境变量。安装完成后，请按照要求重启计算机。重启之前，请保存正在使用的文档等资料并关闭正在运行的其它应用，然后点击“完成”，详见 图 2.7 。



图 2.7: 安装完成并重启计算机

ZIP 压缩包安装

对于下载 ZIP 格式压缩包的用户，请使用任意解压软件将软件解压至任意目录，并按照下述操作设置环境变量。以软件解压路径为 C:\Program Files\copt70 为例进行说明。

- 步骤一：使用 **“管理员权限”** 打开 Windows 命令行，并执行下述命令弹出环境设置面板。

```
rundll32 sysdm.cpl,EditEnvironmentVariables
```

- 步骤二: 修改系统环境变量 `PATH` ，将路径 C:\Program Files\copt70\bin 添加至环境变量 `PATH` 的值中。
- 步骤三：新建系统环境变量 `COPT_HOME` ，变量的值为：C:\Program Files\copt70。
- 步骤四：新建系统环境变量 `COPT_LICENSE_DIR` ，变量的值为：C:\Program Files\copt70。

至此，环境变量配置已完成。请用户仔细阅读软件安装目录下的 `copt-eula_cn.pdf` 用户协议文件，如果用户同意该用户协议，请继续阅读如何申请与配置许可文件 章节。

2.2.2 MacOS

对于 MacOS 平台，我们提供了 DMG 格式的安装程序和 GZIP 格式的压缩包，**用户二选其一即可**。我们 **推荐大多数用户选择 DMG 格式的安装程序**，专家级用户选择 GZIP 格式压缩包。

此外，针对 MacOS 系统，我们提供 MacOS-Universal 的安装包，对于 Apple 芯片和旧版 Intel 芯片都是通用的（安装包后缀为 `universal_mac` ）。

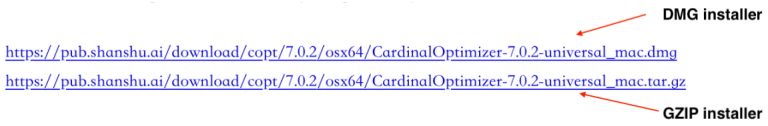


图 2.8: MacOS 系统两种不同安装包下载链接

DMG 安装程序安装

对于下载 DMG 格式安装程序的用户，例如：适用于 MacOS-Universal 的 COPT 7.0.2 版本软件 CardinalOptimizer-7.0.2-universal_mac.dmg ，请按照下述步骤进行操作：

- 步骤一：双击 DMG 格式安装程序, 操作系统将自动挂载 DMG 文件。
- 步骤二：将 copt70 文件夹图标拖动至 `'Applications'` 文件夹, 如 图 2.9 所示。

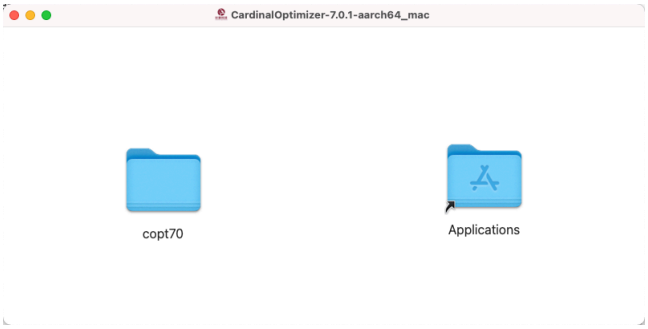


图 2.9: 拖动 copt70 至 'Applications'

上述步骤完成后，需要配置必要的环境变量。请继续参阅：环境变量配置 章节。

GZIP 压缩包安装

对于下载 GZIP 格式压缩包的用户，在终端上输出下述命令解压软件：

```
tar -xzf CardinalOptimizer-7.0.2-universal_mac.tar.gz
```

解压完成后在当前目录得到 copt70 文件夹，用户可以将其移动到任意路径。我们推荐用户移动到 '/Applications' 目录下，在终端中输入下述命令：

```
mv copt70 /Applications/
```

若无法安装至 "/Applications" 目录，也可以将 copt70 文件夹移动至用户目录，在终端下输入下述命令：

```
mv copt70 ~/
```

注意：后续设置 COPT 相关环境变量时 COPT 的相关路径也需要调整。

上述步骤完成后，需要配置必要的环境变量。请继续参阅：环境变量配置 章节。

配置环境变量

上一步安装完成后，还需要配置环境变量。首先，在终端执行下述命令，确定当前使用的 Shell 版本：

```
echo $SHELL
```

若输出为：'/bin/bash'，则表示当前使用终端为 bash；若输出为：'/bin/zsh'，则表示当前使用终端为 zsh。

接着，请打开终端 terminal，并确认处于用户目录下，(若不是，则需在终端执行下述命令切换到用户目录)：

```
cd ~/
```

请用户根据终端的情况，选择参考对应的环境变量配置指南。

- *BASH 终端*
- *ZSH 终端*

环境变量配置完成后，后续可以验证是否配置成功，参考验证环境变量 。

BASH 终端

在 BASH 终端下配置环境变量，主要有三个步骤：

1. 检查'bash_profile' 文件

输入下述命令，输出用户目录下所有文件，查看是否存在 '.bash_profile' 隐藏文件：

```
ls -a
```

若不存在该文件，请执行下述命令，创建空的 '.bash_profile' 文件（若文件已存在，请忽略该步骤）：

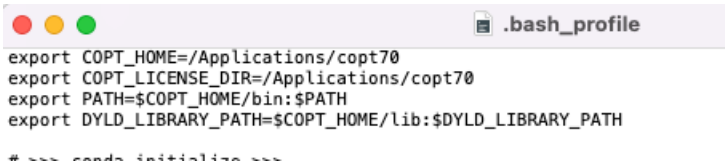
```
touch ~/.bash_profile
```

2. 在'.bash_profile' 文件中，添加环境变量内容

首先，请在终端执行下述命令，打开 '.bash_profile' 文件。

```
open ~/.bash_profile
```

接着，请使用任意文本编辑器编辑现有 '.bash_profile' 文件，并添加如下内容：



注意: 等号两边不能有空格。

3. 检查和修改环境变量生效

保存上述修改并退出，用户可以在终端执行下述命令查看修改后的 '.bash_profile' 文件：

```
cat ~/.bash_profile
```

若修改成功，则终端输出的文件内容中应该包括上述新添加的信息。

然后，用户需要在终端执行下述命令，使得上述环境变量修改生效。

```
source ~/.bash_profile
```

ZSH 终端

在 ZSH 终端下配置环境变量，同样有如下三个步骤：

1. 检查'zshrc' 文件

输入下述命令，输出用户目录下所有文件，执行下述命令查看是否存在 '.zshrc' 隐藏文件：

```
ls -a
```

若不存在该文件，则在终端上执行下述命令创建空的 '.zshrc' 文件：

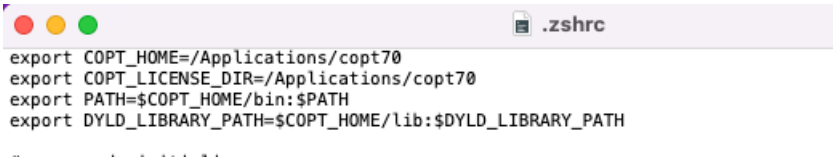
```
touch ~/.zshrc
```

2. 在'.zshrc' 文件中，添加环境变量内容

首先，请在终端执行下述命令，打开 '.zshrc' 文件。

```
open ~/.zshrc
```

若已存在该文件，则用户可以使用任意文本编辑器编辑现有 '.zshrc' 文件，并添加如下内容：



注意： 等号两边不能有空格。

3. 检查和修改环境变量生效

保存上述修改并退出，用户可以在终端执行下述命令查看修改后的 '.zshrc' 文件：

```
cat ~/.zshrc
```

若修改成功，则终端输出的文件内容中应该包括上述新添加的信息。

然后，用户需要在终端执行下述命令，使得上述环境变量修改生效。

```
source ~/.zshrc
```

验证环境变量

对于 bash 终端和 zsh 终端，可以分别输入下述命令检查 COPT 相关环境变量是否设置成功：

```
echo $COPT_HOME
echo $COPT_LICENSE_DIR

echo $PATH
echo $DYLD_LIBRARY_PATH
```

若终端分别输出：

/Applications/copt70

/Applications/copt70

/Applications/copt70/bin:$PATH

/Applications/copt70/lib:$DYLD_LIBRARY_PATH

则表示 COPT 相关环境变量配置成功。

注意： 不同计算机上环境变量 $PATH 和 $DYLD_LIBRARY_PATH 显示的内容可能不同，但设置的 COPT 相关环境变量应正常显示才表示配置成功。

如果用户检查确认已成功添加 COPT 相关环境变量，请仔细阅读安装目录下的 'copt-eula_cn.pdf' 用户协议文件。若同意该用户协议，请继续阅读如何申请与配置许可文件 章节。

MacOS 安全性检查

对于使用 MacOS 10.15 或以上版本系统的用户，可能在后续执行 COPT 相关程序时报告安全性错误，如下图所示：



此时在终端上执行命令：

```
xattr -dr com.apple.quarantine /Applications/copt70
```

去掉 MacOS 系统关于程序的安全性相关的检查，再执行 COPT 相关程序即可。

2.2.3 Linux

对于 Linux 平台，目前我们只提供了 GZIP 格式的压缩包。例如，用户下载了 64 位 COPT 7.0.2 的 Linux 版本软件 CardinalOptimizer-7.0.2-lnx64.tar.gz ，在 Linux 终端上输入下述命令将软件解压到任意路径：

```
tar -xzf CardinalOptimizer-7.0.2-lnx64.tar.gz
```

解压后在当前目录下得到 copt70 文件夹，用户可以将它移动到任意路径。我们推荐移动到 "/opt" 目录下，在终端下输入下述命令：

```
sudo mv copt70 /opt/
```

注意：执行该命令需要 **“root 权限”**。

若无法安装至 "/opt" 目录，也可以将 copt70 文件夹移动至用户目录，在终端下输入下述命令：

```
mv copt70 ~/
```

注意：后续设置 COPT 相关环境变量时 COPT 的相关路径也需要调整。

上述步骤完成后，需要配置必要的环境变量。

在终端执行下述命令切换到用户目录：

```
cd  ~/
```

再执行下述命令查看是否存在 '.bashrc' 隐藏文件：

```
ls -a
```

若不存在该文件，则在终端上执行下述命令创建空的 '.bashrc' 文件：

```
touch ~/.bashrc
```

若已存在该文件，则用户可以使用任意文本编辑器编辑现有 '.bashrc' 文件，并添加如下内容：

```
export COPT_HOME=/opt/copt70
export COPT_LICENSE_DIR=/opt/copt70
export PATH=$COPT_HOME/bin:$PATH
export DYLD_LIBRARY_PATH=$COPT_HOME/lib:$DYLD_LIBRARY_PATH
```

注意： 等号两边不能有空格。

保存上述修改并退出，用户可以在终端执行下述命令查看修改后的 '.bashrc' 文件：

```
cat ~/.bashrc
```

若修改成功，则终端输出的文件内容应该包括上述信息。

然后在终端输入下述命令使得上述环境变量修改生效。

```
source ~/.bashrc
```

最后分别输入下述命令检查 COPT 相关环境变量是否设置成功：

```
echo $COPT_HOME
echo $COPT_LICENSE_DIR

echo $PATH
echo $LD_LIBRARY_PATH
```

若终端分别输出：

/opt/copt70

/opt/copt70

/opt/copt70/bin:$PATH

/opt/copt70/lib:$LD_LIBRARY_PATH

则表示 COPT 相关环境变量配置成功。

注意： 不同计算机上环境变量 $PATH 和 $LD_LIBRARY_PATH 显示的内容可能不同，但设置的 COPT 相关环境变量应正常显示才表示配置成功。

如果用户检查确认已成功添加 COPT 相关环境变量，请仔细阅读安装目录下的 'copt-eula_cn.pdf' 用户协议文件。若同意该用户协议，请继续阅读如何申请与配置许可文件 章节。

2.3 配置许可文件

杉数求解器需要配置相应的许可文件才能正常使用，我们根据不同的用户需求提供了多种授权方式。如果在使用中遇到任何和授权相关的问题，请及时联系 coptsupport@shanshu.ai 。

许可文件具体包括：`license.dat` 和 `license.key` 这两个文件。从 COPT6.5 版本开始，在杉数官网申请获取的个人许可，我们将这两个文件作为附件直接发送（无需自行操作获取了）。用户可以 **直接下载许可文件至本地，跳过下面获取和验证许可的步骤**，直接执行安装许可 这一步。

许可文件的配置主要包括如下三个步骤：

- 获取许可（可跳过）
- 验证许可（可跳过）
- 安装许可

2.3.1 获取许可

注册完成后，用户会获得唯一授权凭证 `key`，该凭证和用户注册信息一一对应。用户可以运行杉数求解器自带的 `copt_licgen` 工具，从杉数授权服务器自动获得授权文档（需要网络连接）。

注意：如果用户已经获取到 `license.dat` 和 `license.key` 这两个授权文档，则需重复执行下面步骤再次获取了。

下面是关于如何使用 `copt_licgen` 工具的简单说明。

对于 Windows 系统，打开一个新的命令行窗口，此时当前路径为用户目录，路径形如：`"C:\Users\shanshu"` 。

对于 MacOS 和 Linux 系统，打开一个新的终端，此时当前路径为用户目录，以符号 ~ 表示。

假设用户的 key 为 `'19200817f147gd9f60abc791def047fb'` ，请输入如下命令获取杉数求解器的许可文件。

```
copt_licgen -key 19200817f147gd9f60abc791def047fb
```

另一种方式是把凭证保存在文件 `key.txt` 中，格式为 `'KEY=xxx'`，输入下述命令以获取许可文件。

```
copt_licgen -file key.txt
```

我们推荐 **第一种方式**获取许可。

若授权服务器验证和用户注册凭证关联的注册信息通过，则生成 `license.dat` 和 `license.key` 授权文档并下载到用户计算机，默认下载目录为当前工作目录。

```
copt_licgen -key 19200817f147gd9f60abc791def047fb
  [Info] Cardinal Optimizer   COPT v7.0.2 20231014
  [Info] Use specific key 19200817f147gd9f60abc791def047fb
  [Info] * get new COPT license from licensing server *
  [Info] Write to license.dat
  [Info] Write to license.key
```

```
  [Info] Received new license files from server
  [Info] Done !!!
```

注意： copt_licgen 工具需要联网才能和授权服务器交互。如果在使用中遇到任何问题，请及时联系 coptsupport@shanshu.ai 。

2.3.2 验证许可

用户可以通过下述命令验证当前工作目录下的授权文档是否支持当前安装的杉数求解器版本。

```
copt_licgen -v
```

如果一切正常，则显示如下日志信息，表明许可文件可以正常使用。

```
copt_licgen -v
  [Info] Cardinal Optimizer   COPT v7.0.2 20231014
  [Info] Run local validation
  [Info] Read license.dat
  [Info] Read license.key
  [Info] Expiry : Tue 2030-12-31 00:00:00 +0800
  [Info] Local validation result: Succeeded
  [Info] Done !!!
```

2.3.3 安装许可

当用户获取并验证了授权文档 license.dat 和 license.key 后，可以采用如下两种方式安装许可文件，任意选择一种方式即可。我们推荐使用 **用户目录方式**。

用户目录方式（推荐）

该方式要求首先在系统的用户目录下 **新建文件夹** copt ，并移动已验证的授权文档 license.dat 和 license.key 至新建文件夹 copt 中。

注意： 新建文件夹 "copt" 必须为小写。

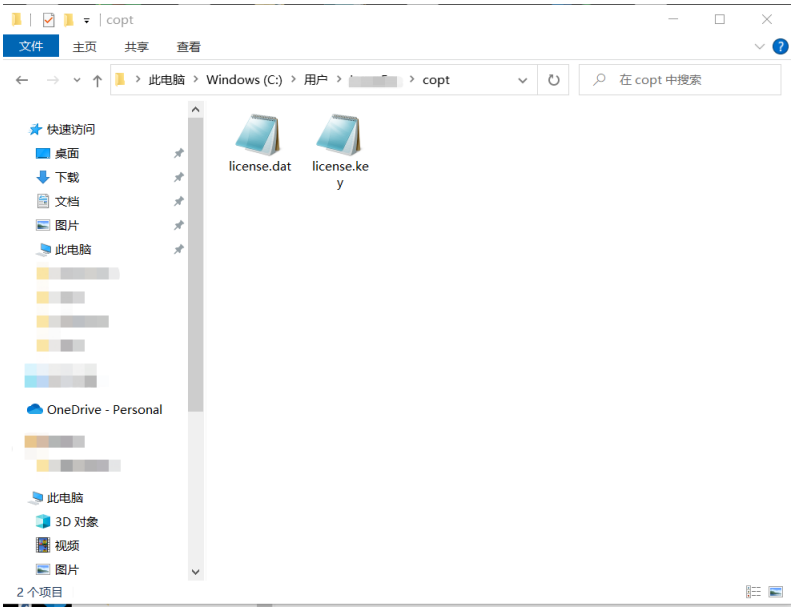
不同系统下的用户目录名称略有不同，请用户根据自己的操作系统参考对应的许可安装步骤：

- *Windows 系统*
- *MacOS 系统*
- *Linux 系统*

Windows 系统

对于 Windows 系统，用户目录形如："C:\Users\username" ，用户可手动将 2 个授权文档 license.dat 和 license.key 移动至目录 "C:\Users\username\copt" 中。

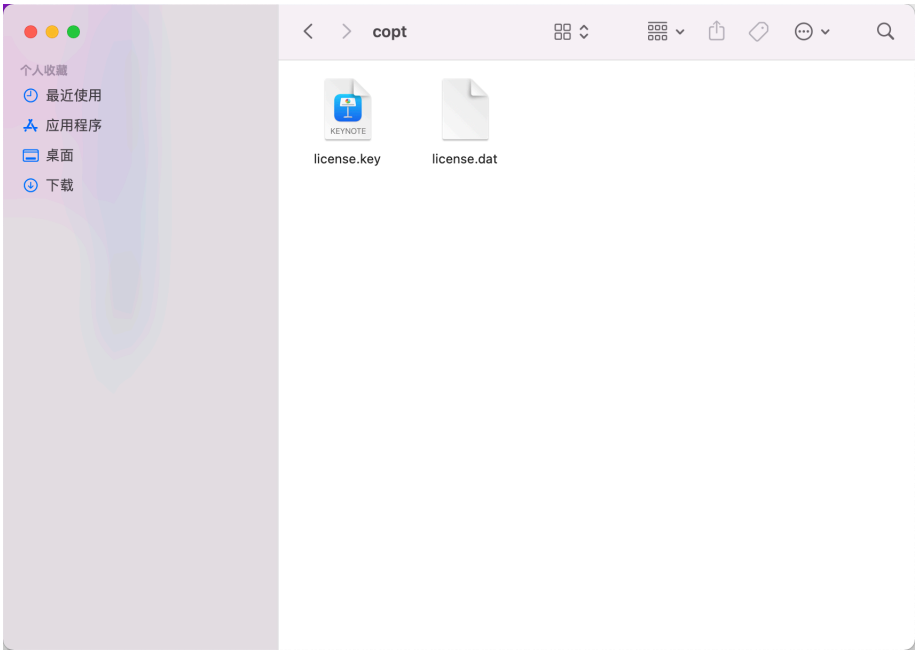
最后，请检查许可文件 license.dat 和 license.key 是否位于该目录下即可，如下图所示，说明已正确安装许可：



MacOS 系统

对于 MacOS 系统，用户目录形如："/home/username"，用户可手动将 2 个授权文档 license.dat 和 license.key 移动至目录 "/home/username/copt" 中。

最后，请检查许可文件 license.dat 和 license.key 是否位于该目录下即可，如下图所示，说明已正确安装许可：



Linux 系统

对于 Linux 系统，用户目录形如："/home/username"，可在终端下执行下述命令将 2 个授权文档 license.dat 和 license.key 移动到目录 "/home/username/copt" 中。

```
mv license.*  ~/copt/
```

接着，请检查许可文件 license.dat 和 license.key 是否位于 "/home/username/copt" 目录下，命令为：

```
ls ~/copt/
```

若终端输出显示存在 license.dat 和 license.key 2 个文件，则表示已正确安装许可，否则安装失败。

环境变量方式

该方式通过设置环境变量 COPT_LICENSE_DIR 实现，其值为许可文件放置的文件目录，默认值为杉数求解器的安装目录。

不同系统下环境变量的查看和操作略有不同，请用户对应自己的操作系统参考如下安装步骤：

- *Windows 系统*
- *MacOS 系统*
- *Linux 系统*

Windows 系统

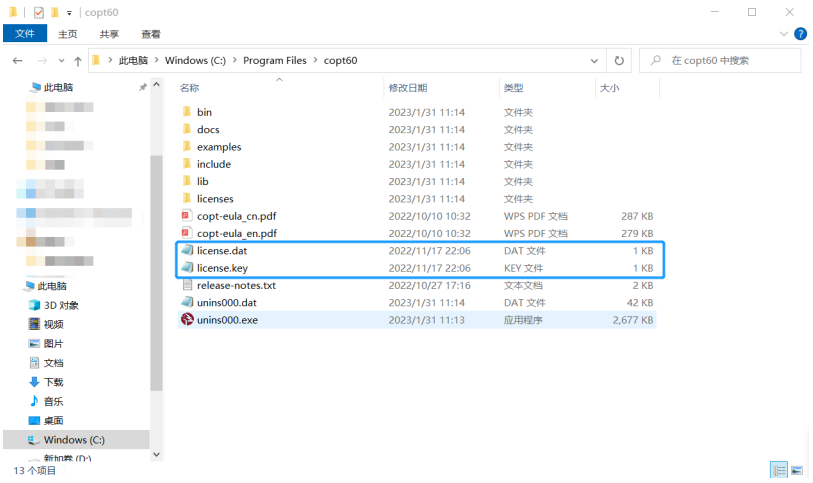
对于 Windows 系统，执行下述命令查看环境变量 COPT_LICENSE_DIR 指向的路径：

```
echo %COPT_LICENSE_DIR%
```

注意：若终端无输出，则表示 COPT 相关环境变量还未配置好，请用户查看软件安装 部分配置杉数求解器。

然后将授权文档 license.dat 和 license.key 移动到 COPT_LICENSE_DIR 指向的路径下。

最后，请用户检查许可文件 license.dat 和 license.key 是否位于环境变量 COPT_LICENSE_DIR 所指向的路径下即可。假设环境变量 COPT_LICENSE_DIR 指向 COPT 的默认安装目录，则如图所示，说明已正确安装许可：

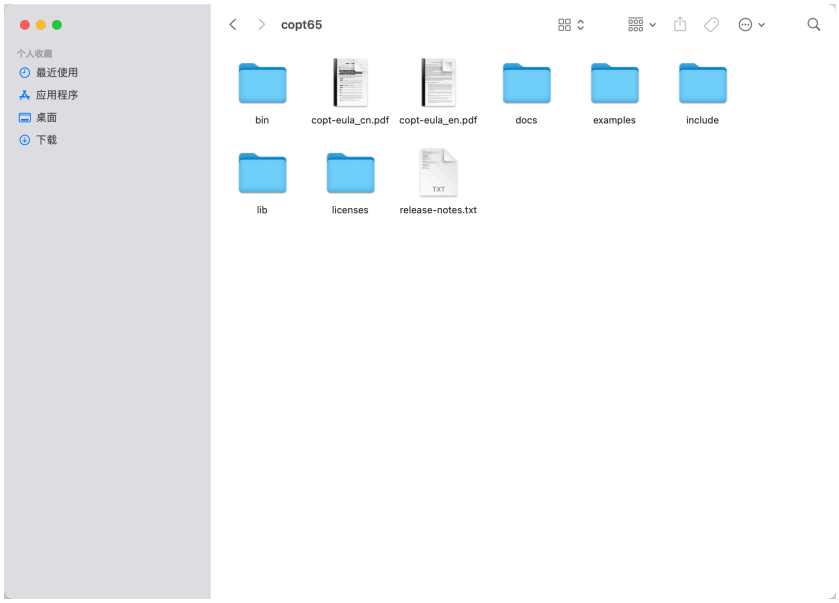


MacOS 系统

对于 MacOS 系统，可通过下述命令查看环境变量 COPT_LICENSE_DIR 指向的路径：

```
echo $COPT_LICENSE_DIR
```

最后，请用户检查许可文件 license.dat 和 license.key 是否位于环境变量 COPT_LICENSE_DIR 所指向的路径下即可。假设环境变量 COPT_LICENSE_DIR 指向 COPT 的安装目录，则如图所示，说明已正确安装许可：



Linux 系统

对于 Linux 系统，执行下述命令查看环境变量 COPT_LICENSE_DIR 指向的路径：

```
echo $COPT_LICENSE_DIR
```

注意：若终端无输出，则表示 COPT 相关环境变量还未配置好，请用户查看软件安装 部分配置杉数求解器。

然后执行下述命令将授权文档 license.dat 和 license.key 移动到 COPT_LICENSE_DIR 指向的路径下：

```
mv license.* $COPT_LICENSE_DIR/
```

注意：对于 Linux 系统，若环境变量 COPT_LICENSE_DIR 指向的路径为 /opt/copt70，则需要以 **root 权限**执行移动操作，命令为：

```
sudo mv license.* $COPT_LICENSE_DIR/
```

最后，请用户检查许可文件 license.dat 和 license.key 是否位于环境变量 COPT_LICENSE_DIR 所指向的路径下，命令为：

```
ls $COPT_LICENSE_DIR/
```

若终端显示存在文件 `license.dat` 和 `license.key` ，则表示已正确安装许可，否则安装失败。

注意：若用户目录的 copt 文件夹和环境变量指向的目录下均存在许可文件 `license.dat` 和 `license.key` ，将优先检查并使用前者（即 copt 文件夹中）的许可。

2.3.4 其他

一般来说，任何一种许可都包括两个授权文件：`license.dat` 和 `license.key` 。它们都有数字签名用来确保文件内容没有改动过。在使用 COPT 交互命令行工具或者直接调用杉数求解器编程接口来求解问题模型时，`license.key` 里的 RSA 公钥首先会被用来验证 `license.dat` 里的签名。之后，才是验证具体的保存为键值形式的授权内容。下面的例子就是具体的授权内容。

```
## SHANSHU LICENSE FILE ##

USER = Trial User
MAC = 44:05:99:31:41:C2
CPUID = BFEBFBFF000706E5
EXPIRY = 2030-12-31
VERSION = 7.0.2
NOTE = Free For Trial
```

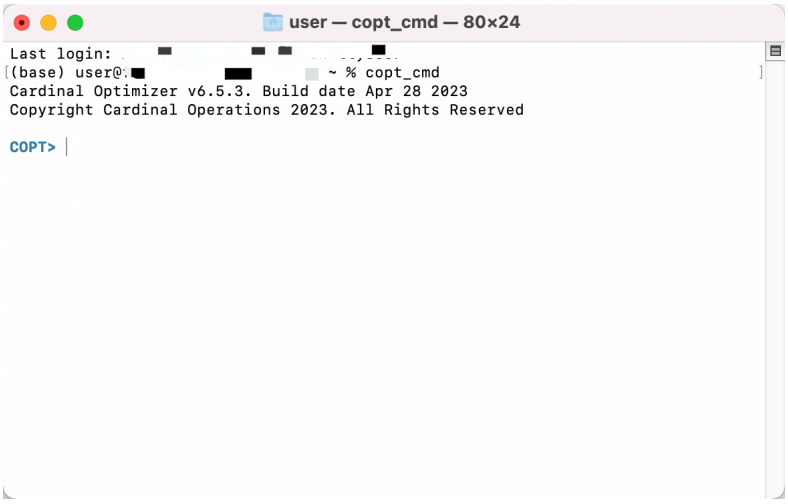
注意：请确保许可文件中 `VERSION` 字段的版本号，和机器上所安装的软件是同一个大版本。否则，在求解会产生：`Invalid signature in public key file` 报错。

2.4 验证安装配置完成

用户完成软件安装和许可配置这两个步骤后，可以进入 COPT 命令行工具，来验证是否安装成功。

在 MacOS 和 Linux 系统中，请打开终端 terminal（在 Windows 系统中，请打开 cmd 命令行窗口），输入 `copt_cmd` 命令，即可使用 COPT 命令行工具。

以 MacOS 为例（Linux 和 Windows 系统也同样），如果用户看到如下所示的窗口界面（无任何报错输出），可以正常进入 COPT 命令行工具，即说明之前的软件和许可配置已成功完成。



如果未能正常进入 COPT 命令行工具，请根据报错信息，参考常见问题 并检查上述软件安装和许可配置的步骤是否正确完成。

成功完成安装后，请根据需要调用 COPT 的接口，继续参考快速入门，查阅各个接口的安装和使用方法。

2.5 升级和更新

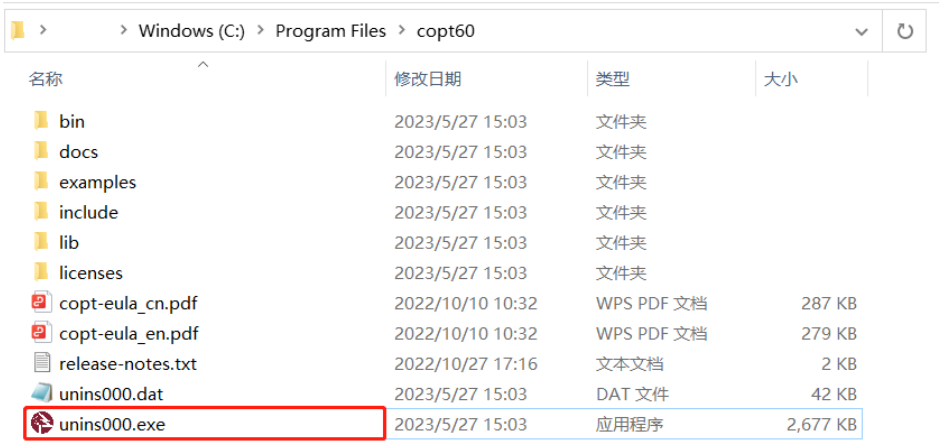
COPT 的升级操作主要分为两部分：1. 软件的升级；2. 许可文件的更新。

1. 卸载旧版 COPT

首先，如果用户没有多个版本同时运行的需要，可以先卸载旧版本的软件。

不同操作系统下，卸载 COPT 的方法略有差别：在 Windows 系统中，请打开旧版本的 COPT 安装包（名称形如 `copt60` ），如下图所示，双击运行 `unins000.exe` 程序，即可自动完成软件卸载；

在 MacOS 和 Linux 系统中，请直接删除旧版本的 COPT 安装包，即可完成软件卸载。



2. 安装新版本 COPT

如果是大版本的迭代升级（例如：COPT 6.5 -> COPT 7.0），用户需要前往 杉数官网求解器 COPT 申请页 重新申请获取新版本试用，并下载邮件中的新版本安装包和许可文件。接着，用户需要重新安装新版本的 COPT，步骤同样请参考软件安装 。

3. 更新许可文件

此外，软件更新完成后，还需要更新许可文件。我们推荐大家通过用户目录的方式 配置许可文件。用户只需删除 copt 文件夹下旧版本的许可文件，并将新的许可文件移入其中，即完成许可的更新。

完成上述步骤后，用户可参考验证安装配置完成，观察 COPT 命令行工具输出的 banner 信息中，版本号是否为新版本。

注意

1. 对于 COPT 的 Python 接口，用户同时还需要 **升级对应的** `coptpy` 。如果 Python 是 Anaconda 发行版，可以通过 `pip install --upgrade coptpy` 升级至最新版本。
2. 以上是针对大版本迭代的升级步骤（例如：COPT 6.5 -> COPT 7.0）。此外，在每个大版本的基础上，COPT 会发布小版本的迭代（例如：COPT 7.0.1 -> COPT 7.0.2），也会有一些功能的修复和更新。对于小版本间的升级，无需更新许可文件。

第 3 章　COPT 交互式命令行工具

COPT 交互式命令行工具 `copt_cmd` 是一个由杉数求解器提供的，与 Windows、Linux 和 MacOS 下终端无缝连接的、可以直接调用杉数求解器编程接口进行模型求解与分析的工具。使用该工具前，请用户确保已正确安装杉数求解器，并成功配置授权文档。

3.1 概述

简单来说，COPT 交互式命令行工具是一个对从终端输入或脚本读入的命令行进行解释并执行的程序。它调用相应的杉数求解器编程接口进行求解与分析，并提供了以下启动参数：

- `-c`: 该选项读取并执行指定的内联脚本内容。注意，内联脚本需要带双引号。
- `-i`: 该选项读取并执行指定的脚本文件。
- `-o`: 该选项提供了脚本录制功能。任何在“交互模式”下“执行正确”的命令都会输出到指定的脚本文件中，文件后缀无特殊要求。

若不提供任何启动参数，则进入“交互模式”。在操作上，该工具目前支持多种键盘功能键、组合键和自定义键，以尽量满足用户的便捷使用需求，达到类似系统终端的使用体验。

3.2 交互模式

命令行工具定义了一些命令和功能键以方便用户进行快速光标定位，也定义了一些组合键以方便用户编辑命令行。目前这些定义是基于标准键盘的，而不是 MAC 键盘。下面对具体功能进行介绍。

- **基本命令**
  1. `<Insert>`: 切换“插入”和“覆盖”模式。
  2. `<Esc>`: 取消当前行的所有输入，且光标回到起始位置，等待输入。Linux 和 MacOS 下使用需连续按两次 `<Esc>` 。
- **光标定位**
  1. `<Home>/<End>`: 光标跳转到输入行首/尾后，等待输入。
  2. `<Left>/<Right>` 方向键: 光标前/后移动一个字符，等待输入。
  3. `<CTRL>+<Left>/<Right>` 方向键: 光标跳转到前一个单词和后一个单词，等待输入。
- **简单的剪切和粘贴**

  以下剪切操作需要用到内部用来粘贴的缓存。

1. <Delete>：剪切当前光标下字符，并缓存留待后续粘贴操作。
2. <Backspace>：剪切当前光标前一位字符，并缓存留待后续粘贴操作。
3. <CTRL>+<H>：剪切当前光标至行首的所有字符，并缓存留待后续粘贴操作。
4. <CTRL>+<E>：剪切当前光标至行尾的所有字符，并缓存留待后续粘贴操作。
5. <CTRL>+<Y>：粘贴当前缓存中的剪切内容。

每个剪切操作都定义了一个剪切方向，即向前剪切或者向后剪切。显然，<Delete> 和 <CTRL>+<H> 属于向前剪切；<Backspace> 和 <CTRL>+<E> 属于向后剪切。如果当前剪切操作的方向和前一个剪切操作的方向一致，剪切部分会追加到粘贴缓存尾部，否者则覆盖原来的缓存内容。

- **命令行历史记录**

  1. <Up>/<Down>：相对于当前位置，显示上一个或下一个历史命令。如果上一次执行命令是已有的历史记录，当前位置为该历史记录所在条目，否则当前位置为最近执行的命令。
  2. <CTRL>+<R> 或 <F8>：如果用户在已知命令前缀的情形下调用历史记录，则可以通过前缀遍历匹配的历史记录进行查找。即首先输入命令前缀，然后按 <CTRL>+<R> 组合键，或者 Windows 系统下标准键盘的 <F8> 键，遍历所有匹配前缀的历史记录。

- **Tab 补全功能**

  以当前光标位置的前一个字符到光标所在单词的字首作为前缀，可以补全交互式工具命令、COPT 参数名或属性名、当前或指定路径所在的文件和文件夹。如果有多个匹配结果，重复按 <Tab> 以遍历所有匹配。

  1. 补全功能会根据光标所在位置进行简单识别，在两种情形下，补全工具命令名。一种情形是当前光标位于命令行的第一个单词，另一种情形是使用 help 命令查阅命令的基本介绍。
  2. 其余情形下，根据匹配前缀补全 COPT 参数名或属性名、当前或指定路径所在的文件和文件夹。特别地，如果用户输入的前缀同时匹配当前工作路径下的某个文件名和某个 COPT 参数名或属性名，则按 COPT 参数名或属性名进行补全。若用户需要补全文件名，可通过添加 './' 前缀，再用 <Tab> 补全，避免优先补全 COPT 参数名或属性名。
  3. 易用性方面，工具支持补全的前缀检验忽略大小写，也支持以字符 (*) 作为通配符筛选文件和文件夹。例如：命令 'set t<Tab>' ，补全为 'set TimeLimit' 。命令 'read ~/*.gz<Tab>' ，补全为用户目录下所有以 '.gz' 为后缀的文件名。
  4. <Shift>+<Tab>：补全到前一个匹配，和 <Tab> 方向相反。

3.3 脚本模式

命令行工具的脚本模式主要有两种形式：一是脚本文件，二是带双引号的内联脚本，即以 ';' 分隔的命令行组成的文本。这两种脚本都可以在启动交互式工具时，使用参数选项进行加载，也可以在交互模式下随时加载。下面介绍如何加载脚本文件。

前文概述中，我们介绍了在启动工具时，使用选项 '-i' 可以加载指定脚本文件，下面进一步阐述相关细节。

- 加载脚本文件时，除了检查指定文件是否打开成功，还会检查文件内容中首个非空行是否以 '#COPT script file' 起始，以避免执行无关的文件。事实上，工具仅检查以否以 '#COPT' 起始。任意首个非空格字符为 '#' 的中间行都会被忽略，一般用于添加注释。
- 加载脚本文件成功后，会逐条执行脚本中的命令，直到文件结尾，或者读到字符 '?' 。工具以字符 '?' 表示暂停脚本执行。如果用户需要继续执行，可以通过手动输入命令 load 继续执行当前脚本，直到文件结尾或者下一个 '?' 。只有在当前脚本中命令全部执行后（有提示），才可以加载其它脚本文件。

工具支持加载一种特殊的脚本，即带引号的内联脚本。内联脚本与脚本文件的区别是两者的命令行分隔符分别是 ';' 和 '\n' 。对于内联脚本，可以在启动时使用选项 '-c' 指定的方式加载，也可以在交互模式下使用命令 load 随时加载，字符 '?' 仍表示暂停脚本执行。

用户还可以通过选项 '-o' 指定脚本输出文件，用于记录用户在交互模式下的所有“执行正确”的命令行。其中，“正确”是指命令行使用的是已知的工具命令，且命令语法使用正确。

特别地，load 命令本身不被写入输出脚本。由于被加载脚本的具体内容会在执行过程中展开，并写入到输出脚本，不写入 load 命令可以避免可能的脚本执行死循环。

3.4 工具命令

COPT 交互式命令行工具目前支持以下命令，方便用户求解与分析优化模型。所有命令均大小写无关，并支持 <Tab> 补全。

3.4.1 普通工具命令

普通工具命令提供了命令帮助说明查阅、脚本加载等通用功能，不直接调用杉数求解器编程接口进行操作，包括以下命令：

- cd: 类似 DOS 命令 'cd' ，该命令改变 '当前工作目录' 。这里要求其参数是一个有效的目录路径，可以是相对目录路径，也可以是绝对目录路径。注意下面工具命令中用到的相对路径或路径补全功能都是基于当前工作目录的。其初始值设为 copt_cmd 所在的目录，之后用户可以通过命令 cd <dirpath> 来改变工作路径。例如，如果用户把当前工作目录设为求解问题模型所在目录，那么读取模型就变得很简洁，只需要模型文件名即可。
- dir/ls: 类似 DOS 命令 dir 和 Bash 命令 ls ，该命令列出给定的相对或绝对路径下的所有文件和文件夹。设计该命令是为了方便用户定位待求解的模型文件、脚本文件等。例如：使用 dir 命令或者 ls 命令列出当前工作目录下的所有文件和文件夹；使用 'dir ../' 列出上一层目录下的所有文件和文件夹；使用 'dir ~/' 列出用户目录下的所有文件和文件夹。

  该命令还支持使用通配符 (*) 筛选出符合条件的文件和文件夹。例如：使用 'dir ~/net' 命令，列出用户目录下所有前缀为 'net' 的文件；使用 'dir d:/problems/*.mps.gz' 命令，列出 'd:/problems' 目录下的所有后缀为 '.mps.gz' 的模型文件。
- exit/quit: 退出 COPT 命令行模式。
- help: 提供 COPT 命令的使用说明。查阅单个命令的详细使用说明，需要在 help 命令后指定 COPT 命令名称，例如：使用 'help display' 命令，查看 display 命令的使用说明。特别地，对于不带参数的 help 命令调用，将列出所有 COPT 命令以及对应的简单说明，方便用户查看，并在

新的提示符所在行，自动加上 `help` 命令调用，用户只需补充待查询的 COPT 命令名即可。输入 COPT 命令时，可以 `<Tab>` 补全，且不区分大小写。

- `load`: 读入给定路径下的脚本文件或者带引号的内联脚本，执行直到文件结尾或暂停字符 `'?'` 。暂停后，如果继续输入 `load` 命令，无论是否带参数（实际上任何参数都会被忽略），均继续执行被暂停的脚本。之前加载的脚本执行完成后，才可以加载其它脚本。
- `pwd`: 类似 Bash 命令 `'pwd'` ，该命令显示当前工作目录，可以让用户查看目前在什么位置。
- `status`: 交互工具通过状态机表示交互过程中的状态 (参看 图 3.1)。其一是为了避免用户的误操作，同时也可以提示用户下一步可进行的操作。输入 `status` 命令返回交互工具当前的状态，分为以下几种：

  1. `Initial` ： 未读取模型，可能在工具刚启动时，或者用户调用 `reset` 命令后。
  2. `Read` ： 成功读取给定模型文件。
  3. `SetParam` ： 用户通过 `set` 命令成功设置了求解参数。
  4. `Optimize` ： 用户调用 `optimize/optimizelp` 等求解命令求解模型。

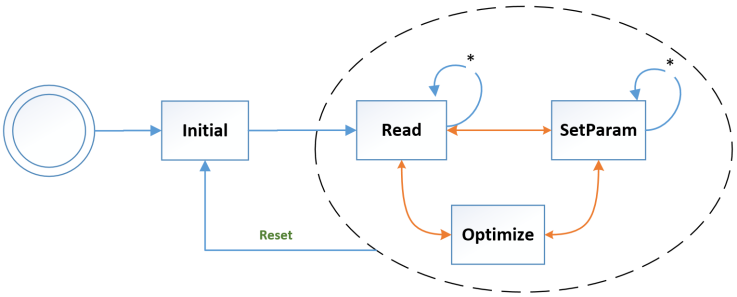


图 3.1: 命令行工具的内部状态

3.4.2 COPT 工具命令

COPT 工具命令是求解与分析优化模型的核心命令集，它直接调用杉数求解器编程接口进行相关操作，包括以下命令：

- `display/get`: 获取给定的参数或属性的当前值。特别地，对于不带参数名的调用，会列出目前所有支持的参数和属性名称，方便不熟悉 COPT 参数和属性名的用户查看其所需的信息，并在新的提示符所在行，自动添加 `display` 命令调用，用户只需补充待查询的参数或属性名即可。在输入参数或属性名称时，支持 `<Tab>` 补全，且不区分大小写，自动补全显示的名字是标准形式。例如：使用 `'get so<Tab>'` 命令，补全后的命令是 `'get SolvingTime'`。
- `opt/optimize`: 求解已读入的模型，并将工具状态置为 `'Optimize'`。
- `optlp/optimizelp`: 将已读入的模型当成 LP 问题求解，并将工具状态置为 `'Optimize'`。
- `iis`: 计算已读入模型的 IIS。
- `feasrelax`: 计算已读入模型的可行化松弛。其中，`'feasrelax'` 或 `'feasrelax all'` 表示松弛模型中所有的变量和约束，则惩罚因子均为 1；`'feasrelax vars'` 表示仅松弛模型中变量的上下界，且惩罚因子均为 1；`'feasrelax cons'` 表示仅松弛模型中约束的上下界，且惩罚因子均为 1。
- `tune`: 对已读入模型进行自动参数调优。

- loadtuneparam: 加载指定的调优结果到当前已读入模型。调用命令为 'loadtuneparam idx'，其中 'idx' 为指定的调优结果编号。通过命令 'get TuneResults' 获取当前调优后获取的调优结果数目。
- read: 读取给定的相对或绝对路径下的模型文件、结果文件、基文件、初始解文件和参数设置文件。目前支持 '.mps' 、'.mps.gz' 、'.lp' 和 '.lp.gz' 、'.dat-s' 和 '.dat-s.gz' 、'.cbf' 和 '.cbf.gz' 、'.bin' 和 '.bin.gz' 模型文件格式；以 '.sol' 为后缀的结果文件；以 '.bas' 为后缀的基解文件；以 '.mst' 为后缀的初始解文件；以 '.par' 为后缀的参数设置文件；以 '.tune' 为后缀的调参文件。若文件类型不匹配，则报告相应错误信息。
- readmps: 读取给定的相对或绝对路径下的模型文件。该命令不要求待读取的文件后缀为 '.mps' 或 '.mps.gz' ，只要文件本身符合 MPS 格式即可。命令调用完成后将工具状态置为 'Read' 。
- readlp: 读取给定的相对或绝对路径下的模型文件。该命令不要求待读取的文件后缀为 '.lp' 或 '.lp.gz' ，只要文件本身符合 LP 格式即可。命令调用完成后将工具状态置为 'Read' 。
- readsdpa: 读取给定的相对或绝对路径下的模型文件。该命令不要求待读取的文件后缀为 '.dat-s' 或 '.dat-s.gz' ，只要文件本身符合 SDPA 格式即可。命令调用完成后将工具状态置为 'Read' 。
- readcbf: 读取给定的相对或绝对路径下的模型文件。该命令不要求待读取的文件后缀为 '.cbf' 或 '.cbf.gz' ，只要文件本身符合 CBF 格式即可。命令调用完成后将工具状态置为 'Read' 。
- readbin: 读取给定的相对或绝对路径下的模型文件。该命令不要求待读取的文件后缀为 '.bin' 或 '.bin.gz' ，只要文件本身符合 COPT 二进制格式即可。命令调用完成后工具状态置为 'Read' 。
- readsol: 读取给定的相对或绝对路径下的结果文件。该命令不要求待读取的文件后缀为 '.sol' ，只要文件本身符合 COPT 的结果文件格式即可。
- readbasis: 读取给定的相对或绝对路径下的基解文件。该命令不要求待读取的文件后缀为 '.bas' ，只要文件本身符合 COPT 的基解文件格式即可。
- readmst: 读取给定的相对或绝对路径下的初始解文件。该命令不要求待读取的文件后缀为 '.mst' ，只要文件本身符合 COPT 的初始解文件格式即可。
- readparam: 读取给定相对或绝对路径下的参数设置文件。该命令不要求待读取的文件后缀为 '.par' ，只要文件本身符合 COPT 的参数设置文件格式即可。
- readtune: 读取给定相对或绝对路径下的调参文件。该命令不要求待读取的文件后缀为 '.tune' ，只要文件本身符合 COPT 的调参文件格式即可。
- reset: 重置模型中的解信息，以及初始解等其它信息。工具的状态设为 'Initial' 。
- resetparam: 重置优化参数为默认值。工具的状态设为 'Initial' 。
- set: 设置给定的 COPT 参数的当前值。该命令的完整语法形如：'set TimeLimit 100' 。特别地，对于不带参数名的调用，将列出目前所有参数的名称和简介，方便不熟悉 COPT 参数的用户查看其所需信息，并在新的提示符所在行，自动添加 set 命令调用，用户只需要补充待查询的参数名称即可。在输入参数名称时，支持 <Tab> 补全，且不区分大小写，自动补全的名字是标准形式。

  例如：用户需要设置求解器的求解时间为 1000 秒，完整命令为：'set TimeLimit 1e3' ，若用户不确定待设置参数的允许范围，可在输入命令 'set TimeLimit' 后按 <Enter> 键，交互工具会反馈 TimeLimit 参数的当前值、默认值、最小值和最大值，以便用户正确输入。

  对于不允许设置的属性，或者输入了错误的参数名称，则报告相应的错误。命令正确完成后将工具的状态置为 'SetParam' 。

- write: 输出模型文件、结果文件、IIS 文件、可行化松弛文件、基文件、初始解文件和参数设置文件。该命令根据指定的输出文件的后缀执行相应的输出任务，即如果文件后缀是 '.mps' ，则自动将当前加载的模型输出为 MPS 格式的模型文件，功能相当于命令 writemps ； 如果文件后缀是 '.lp' ，则自动将当前加载的模型输出为 LP 格式的模型文件，功能相当于命令 writelp ； 如果文件后缀是 '.bin' ，则自动将当前加载的模型输出为 COPT 二进制格式的模型文件，功能相当于命令 writebin ； 如果文件后缀是 '.cbf' ，则自动将当前加载的模型输出为 CBF 格式的模型文件，功能相当于命令 writecbf ； 如果文件后缀是 '.iis'，则将输出计算得到的 IIS 模型到 IIS 文件，功能相当于命令 writeiis ； 如果文件后缀是 '.relax'，则将输出计算得到的松弛模型到 Relax 文件，功能相当于命令 writerelax ； 如果文件后缀是 '.sol' ，则自动将模型求解的最终结果输出到结果文件，功能相当于命令 writesol ； 如果文件后缀为 '.bas' ，则自动将模型求解后最终的基状态输出到基文件，功能相当于命令 writebasis ； 如果文件后缀为 '.mst' ，则自动输出初始解信息到文件，功能相当于命令 writemst ； 如果文件后缀为 '.par' ，则自动将求解当前模型时修改过的 COPT 参数设置输出到文件，功能相当于命令 writeparam 。除此之外，不匹配的文件类型会提示出错。
- writemps: 将当前求解问题输出为给定的相对或绝对路径下的 MPS 格式模型文件。如果文件后缀名不是 '.mps' ，则自动添加后缀 '.mps' 。
- writelp: 将当前求解问题输出为给定的相对或绝对路径下的 LP 格式模型文件。如果文件后缀名不是 '.lp' ，则自动添加后缀 '.lp' 。
- writecbf: 把（半定）问题输出到'.cbf' 文件。如果文件后缀名不是 '.cbf' , 则自动添加后缀 '.cbf' 。注意不支持输出带有二次目标或者二次约束的问题。
- writebin: 将当前求解问题输出为给定的相对或绝对路径下的 COPT 二进制格式模型文件。如果文件后缀名不是 '.bin' ，则自动添加后缀 '.bin' 。
- writeiis: 将当前读入问题的 IIS 模型保存至给定的相对或绝对路径下的 IIS 文件。如果文件后缀名不是 '.iis' ，则自动添加后缀 '.iis' 。
- writerelax: 把具有可行解的松弛问题输出到'.relax' 文件。如果文件后缀名不是 '.relax' ，则自动添加后缀 '.relax' 。
- writesol: 将当前求解问题的结果保存至给定的相对或绝对路径下的结果文件。如果文件后缀名不是 '.sol' ，则自动添加后缀 '.sol' 。
- writepoolsol: 将当前求解问题解池中指定编号的结果保存至给定的相对或绝对路径下的结果文件。如果文件后缀名不是 '.sol' ，则自动添加后缀 '.sol' 。调用命令为 'writepoolsol idx pool_idx.sol'，其中 'idx' 为指定的解池结果编号。通过命令 'get PoolSols' 获取当前解池结果数目。
- writebasis: 将当前求解问题模型的基状态信息保存至给定的相对或绝对路径下的基文件。如果文件后缀名不是 '.bas' ，则自动添加后缀 '.bas' 。
- writemst: 将当前求解问题模型的初始解信息保存至给定的相对或绝对路径下的初始解文件。如果文件后缀名不是 '.mst' ，则自动添加后缀 '.mst' 。
- writeparam: 将当前修改过的 COPT 参数设置保存至给定的相对或绝对路径下的参数设置文件。如果文件后缀名不是 '.par' ，则自动添加后缀 '.par' 。
- writetuneparam: 将指定的调优结果保存至给定的相对或绝对路径下的参数设置文件。调用命令为 'writetuneparam idx tune_idx.par'，其中 'idx' 为指定的调优结果编号。通过命令 'get

TuneResults' 获取当前调优后获取的调优结果数目。

3.5 使用示例

本节以著名的“Diet 问题”为例，演示如何使用 copt_cmd 交互式命令行工具求解优化问题。关于该问题的具体描述，请参考*AMPL 接口 - 使用示例*。

首先请用户按照*安装说明* 安装杉数求解器并配置许可文件，然后在 Windows 的命令行或者 Linux 和 MacOS 的终端中输入下述命令：

```
copt_cmd
```

若屏幕输出如下，则表示调用成功。用户可在交互式界面中输入 help 查看命令使用说明。

```
Cardinal Optimizer v7.0.2. Build date Oct 14 2023
Copyright Cardinal Operations 2023. All Rights Reserved

COPT>
```

假定待求解问题模型为 diet.mps ，且位于当前工作目录下，在交互式界面中输入下述命令读取待求解模型：

```
COPT> read diet.mps
Reading from '/home/username/copt/diet.mps'
Reading finished (0.00s)
```

求解具体问题模型之前，用户可以任意设置 COPT 的求解参数。例如：设置求解时间限制为 10 秒，则在交互界面输入下述命令：

```
COPT> set timelimit 10
Setting parameter 'TimeLimit' to 10
```

如果用户不熟悉 COPT 支持的求解参数，可以输入 set 命令列出所有 COPT 支持的参数和属性名，以及相应的简单说明。如果在 set 命令后加上参数名，比如：'set TimeLimit' ，则会显示指定参数的当前值、默认值、最小值和最大值。

设置完求解参数后，可以使用工具命令 opt 或 optimize 求解当前优化模型，求解过程中显示的日志信息如下：

```
COPT> opt
Model fingerprint: 129c032d

Hardware has 4 cores and 8 threads. Using instruction set X86_NATIVE (1)
Minimizing an LP problem

The original problem has:
    4 rows, 8 columns and 31 non-zero elements
The presolved problem has:
```

(续下页)

(接上页)

```
    4 rows, 8 columns and 31 non-zero elements

Starting the simplex solver using up to 8 threads

Method   Iteration           Objective  Primal.NInf   Dual.NInf        Time
Dual             0    0.0000000000e+00            4           0       0.01s
Dual             1    8.8201991646e+01            0           0       0.01s
Postsolving
Dual             1    8.8200000000e+01            0           0       0.01s

Solving finished
Status: Optimal  Objective: 8.8200000000e+01  Iterations: 1  Time: 0.01s
```

求解完成后，用户可以通过工具命令 get 加上具体的参数名或属性名查看相关信息。与工具命令 set 类似，直接使用 get 命令将列出所有 COPT 支持的参数名和属性名，及其简单介绍。特别地，使用 'get all' 命令将显示所有 COPT 支持的参数名和属性名，及其当前值。

```
COPT> get TimeLimit
  DblParam: [Current]  10s
COPT> get LpObjval
  DblAttr:  [Current] 88.2
COPT> get LpStatus
  IntAttr:  [Current] 1 optimal
```

退出交互界面之前，用户可以选择把求解结果、修改过的参数和基状态输出到指定文件。下面演示了将求解结果输出到当前路径下的 diet.sol 文件中，然后退出交互界面。由于未指定后缀为 '.sol' ，则 copt_cmd 在输出时自动追加后缀 '.sol' 。

```
COPT> writesol diet
  Writing solutions to /home/username/copt/diet.sol
COPT> quit
  Leaving COPT...
```

用户也可以将上述步骤对应的命令写入脚本文件 diet.in 中，详见 代码 3.1:

代码 3.1: diet.in

```
#COPT script-in file

read diet.mps
set timelimit 10
opt
writesol diet
quit
```

然后在启动 copt_cmd 时使用选项 '-i' 加载上述脚本文件，如下所示：

```
copt_cmd -i diet.in
```

也可以在交互模式下使用工具命令 load 直接加载，如下所示：

```
COPT> load diet.in
```

第 4 章　COPT 浮动授权服务

COPT 浮动授权服务器，COPT Floating Token Server，是一个由杉数求解器提供的，能对局域网内用户提供临时授权的服务。服务器端程序可以在 Windows、Linux 和 MacOS 三大系统下运行。

如果 COPT 浮动授权服务器成功启动，根据许可文件上设定的 Token 数量，可以对本地装有同版本或者适配低版本的杉数求解器，但未获得单机授权的用户进行临时授权。用户求解结束后，可以通过释放 COPT 环境对象来返回 Token 给服务器。

4.1 服务器端安装说明

COPT 浮动授权服务器对应的可执行文件是 `copt_flserver`。其启动时需要验证具有浮动类型的许可文件，读取服务器运行所需的配置文件 `fls.ini`，以及在杉数授权服务器进行远程验证，例如验证浮动授权服务器的 IP 符合注册时提供的 IP 范围。浮动版的许可文件的获取可以通过 `copt_licgen` 工具和注册后得到的许可凭证，由杉数授权服务器远程自动生成。详情描述如下或者参考前文如何申请与配置许可文件。

4.1.1 安装步骤

杉数提供的 COPT 远程服务安装包，包含了 COPT 浮动服务器端的程序。用户需先通过客服申请注册并获取 COPT 远程服务安装包，然后可直接将安装包解压缩并放置于自定义路径下。过程属于绿色安装，无需配置环境变量，具体安装步骤演示如下：

对于 Windows 用户

请使用 ZIP 解压软件将软件解压至任意目录，推荐软件解压缩到 `C:\Program Files` 目录下。

对于 Linux 用户

鼠标操作解压缩 COPT 远程服务安装包，或在终端上输入下述命令解压缩：

```
tar -xzf CardinalOptimizer-Remote-7.0.2-lnx64.tar.gz
```

解压缩后在当前目录下得到 copt_remote70 文件夹，用户可以将它移动到其它自定义路径下。对于 `root` 用户我们推荐移动到 `/opt` 目录下，对于非 `root` 用户可以放在 `$HOME` 目录下。即在终端下输入下述命令（以 `root` 用户为例）

```
sudo mv copt_remote70 /opt
```

注意，执行该命令需要 `root` 权限。

对于 MacOS 用户

鼠标操作解压缩 COPT 远程服务安装包，或在终端上输入下述命令解压缩：

```
tar -xzf CardinalOptimizer-Remote-7.0.2-universal_mac.tar.gz
```

解压缩后在当前目录下得到 copt_remote70 文件夹，用户可以将它移动到其它自定义路径下。对于 root 用户我们推荐移动到 /Applications 目录下，对于非 root 用户可以放在 $HOME 目录下。即在终端下输入下述命令（以 root 用户为例）

```
mv copt_remote70 /Applications
```

4.1.2 许可文件

在成功安装 COPT 远程服务包后，还需配置浮动许可文件。我们推荐把获得的许可文件 license.dat 和 license.key 放置在 COPT 远程服务安装路径下的 floating 目录。

下面讲解在不同系统下，如何通过 copt_licgen 工具和许可凭证信息 key 来获取许可文件。

注意

如果用户已经获取到 license.dat 和 license.key 这两个授权文档，则无需重复获取了。可以跳过如下获取许可文件的步骤，直接参阅配置文件 。

对于 Windows 用户

需要打开命令行工具，也就是 cmd 。假如 COPT 远程服务已安装在 "C:\Program Files" ，在命令行工具窗口输入如下命令后，进入 COPT 远程服务安装路径下的 floating 目录：

```
cd "C:\Program Files\copt_remote70\floating"
```

假如用户的许可凭证信息为 7483dff0863ffdae9fff697d3573e8bc ，因为生成许可文件的 copt_licgen 在 COPT 远程服务安装路径下的 tools 文件夹内，则可以在命令行工具窗口输入下述命令获取杉数 COPT 的许可文件 license.dat 和 license.key 。

```
..\tools\copt_licgen -key 7483dff0863ffdae9fff697d3573e8bc
```

对于 Linux 和 MacOS 用户

假如 COPT 远程服务安装在 "/Applications" 目录下，在终端输入以下命令进入浮动授权服务端目录（以 MacOS 系统为例）：

```
cd /Applications/copt_remote70/floating
```

进入 floating 目录后，假如用户的许可凭证信息为 7483dff0863ffdae9fff697d3573e8bc ，则可以在终端输入下述命令获取杉数 COPT 的许可文件 license.dat 和 license.key 。

```
../tools/copt_licgen -key 7483dff0863ffdae9fff697d3573e8bc
```

特别地，如果用户不是在 COPT 远程服务安装路径下的 floating 目录下配置的许可文件，建议将 license.dat 和 license.key 两个许可文件移至此路径下。可手动将这两个文件放到 floating 目录下，也可以输入下述命令移动这两个许可文件（以 MacOS 系统为例）。

```
mv license.* /Application/copt_remote70/floating
```

4.1.3 配置文件

COPT 浮动授权服务器的配置文件 fls.ini 同样位于 COPT 远程服务安装路径下的 floating 目录。内容如下面所示。

```
[Main]
Port=7979

[Licensing]
# if empty or default license name, it is from binary folder
# to get license files from cwd, add prefix "./"
# full path is supported as well
LicenseFile = license.dat
PubkeyFile = license.key

[Filter]
# default policy 0 indicates accepting all connections, except for ones in blacklist
# otherwise, denying all connections except for ones in whitelist
DefaultPolicy = 0
UseBlackList = true
UseWhiteList = true
FilterListFile = flsfilters.ini
```

这里，配置文件可以设置浮动授权服务器的端口，客户端需要连接这个端口才能通讯和获得授权。在下面 Licensing 部分，配置项设定许可文件的路径。就如上面注释描述的，空缺或者默认的许可文件名表示 COPT 浮动授权服务器从可执行文件所在目录下读取许可文件。

如果用户在当前目录下启动服务器，并且许可文件也在当前目录下，通过修改 LicenseFile = ./license.dat 和 PubkeyFile = ./license.key，可以让 COPT 浮动授权服务器从当前目录读取许可文件。

最后是 Filter 部分，DefaultPolicy 默认设为 0，表示除了在黑名单里的用户，其他均允许连接到本授权服务器；反之，除了在白名单中的用户，其他设备均无法连接到本授权服务器。UseBlackList 为 True 表示使用黑名单列表，UseWhiteList 为 True 表示使用白名单列表。FilterListFile 表示浮动授权服务器过滤配置文件名为 flsfilters.ini 。其格式举例如下：

```
[BlackList]
# 127.0.*.* + user@machine*

[WhiteList]
# 127.0.1.2/16 - user@machine*

[ToolList]
# only tool client at server side can access by default
127.0.0.1/32
```

BlackList 和 WhiteList 表示可添加规则到黑名单以及白名单里。规则的格式是以 IP 地址开头，这里 IP 可以用通配符匹配，或者固定位数的方式。另外，IP 地址后面还可以包括 (+) 或者排除 (-) 给定用户名和给定机器名。其中用户名 user 是指 username ， MacOS/Linux 系统用户可以在终端通过命令 whoami

来查询；机器名 machine 是指 hostname ，MacOS/Linux 系统用户可以在终端通过命令 hostname 来查询。ToolList 是对工具使用者的访问限制，添加相关 IP 地址表示允许此远程机器访问。需注意的是，配置文件 flsfilters.ini 更改之后，可使用 ResetFilters 命令把当前的规则重置为过滤配置文档里的规则。也可使用 WriteFilters 命令把当前的规则输出到过滤配置文档。

我们知道杉数求解器还允许客户通过环境变量 COPT_LICENSE_DIR 设置授权文档所在的路径。详情可参考如何安装杉数求解器。在这种情况下，需要修改配置文件为完整的环境变量路径，COPT 浮动授权服务器才能读取环境变量下的授权文档。

4.1.4 使用示例

如果合法的浮动许可文件安装在浮动授权服务器可执行文件所在的目录下，要启动 COPT 浮动授权服务器，只需在 Windows 的命令行或者 Linux 和 MacOS 的终端中输入下述命令：

```
copt_flserver
```

若屏幕输出如下，则表示成功启动浮动授权服务器，启动后会监视和管理客户端的授权申请。如果用户输入字符 Q，则浮动授权服务器会关闭退出。

```
> copt_flserver
  [ Info] Floating Token Server, COPT v7.0.2 20231014
  [ Info] server started at port 7979
```

如果本地授权验证不通过，或者远程服务器连接错误，则可能出现如下输出。

```
> copt_flserver
  [ Info] Floating Token Server, COPT v7.0.2 20231014
  [Error] Invalid signature in public key file
  [Error] Fail to verify local license
```

和

```
> copt_flserver
  [ Info] Floating Token Server, COPT v7.0.2 20231014
  [Error] Error to connect license server
  [Error] Fail to verify floating license by server
```

4.2 客户端使用说明

浮动授权的客户端可以是 COPT 命令行交互工具，或者通过接口调用杉数求解器动态库的方式，比如 Python 接口。如果客户没有获得单机许可，那么浮动许可是一种更灵活的替代方式。所有使用 COPT 的客户端都可以通过浮动授权的方式合法使用杉数求解器。

4.2.1 安装和配置

首先用户需要确保已在客户端正确安装杉数求解器，详情可参考如何安装杉数求解器。成功安装 COPT 客户端后，用户不需要配置本地的许可文件，但需要配置 COPT 浮动授权客户端的配置文件 `client.ini`，如下表所示。

```
Host = 192.168.1.11
Port = 7979
QueueTime = 600
```

上述配置文件表示客户端会尝试和 `192.168.1.11:7979` 连接，且排队等待时间最多为 600 秒。这里，如果 `Host` 或者 `Floating` 为空，则自动设为 `localhost` 。`QueueTime` 空缺则认为是 0，即如果服务器的 Token 都已经被占用就不等待，直接错误退出。这里只有 `Port` 必须设置，不能空缺，而且需和浮动授权服务器的配置文件里设的端口一致。注意客户端配置文件中的关键词是不区分大小写的。

使用浮动授权的前提是客户端首先发现了配置文件 `client.ini`，而不是本地许可文件。但和寻找本地许可的途径一样，客户端会从当前目录，环境变量目录 `COPT_LICENSE_DIR` 和客户端可执行文件所在目录，依次查看是否有浮动许可的配置文件 `client.ini`。所以，如果用户想使用浮动授权，即使用户已经在环境目录下有了本地许可，用户还可以在当前目录（不同于环境目录）下，建立配置文件 `client.ini` 来激活浮动授权。

如果是调用杉数求解器动态库方式，比如 Python 接口，在建模中创建 COPT 优化求解环境时，可以进入浮动授权模式。只要当前目录下没有授权文档，但有配置好的 `client.ini`。建模中最后释放优化求解环境时，会依据最后一个释放的环境对象来决定是否退出浮动授权模式，并通知服务器释放占用 Token。

4.2.2 使用示例

假如用户在当前目录下设置了客户端配置文件 `client.ini` （同时没有本地授权许可），下面以交互式命令行工具 `copt_cmd` 为例，演示如何使用浮动许可获得临时授权。首先在 Windows 的命令行或者 Linux 和 MacOS 的终端中输入下述命令：

```
copt_cmd
```

若屏幕输出如下，则表示客户端成功获得临时浮动许可，可以使用 COPT 求解优化问题。

```
> copt_cmd
  Cardinal Optimizer v7.0.2. Build date Oct 14 2023
  Copyright Cardinal Operations 2023. All Rights Reserved

  [ Info] initialize floating client: ./client.ini

  [ Info] connecting to server ...
  [ Info] connection established
COPT>
```

若屏幕输出如下，则表示客户端连接到浮动授权服务器，但由于 Token 数目有限，需要排队等待。

```
> copt_cmd
  Cardinal Optimizer v7.0.2. Build date Oct 14 2023
```

```
  Copyright Cardinal Operations 2023. All Rights Reserved

  [ Info] Initialize floating client: ./client.ini

  [ Info] connecting to server ...
  [Error] empty license and queue size 1
  [ Info] Wait for license in  2 / 39 secs
  [ Info] Wait for license in  4 / 39 secs
  [ Info] Wait for license in  6 / 39 secs
  [ Info] Wait for license in  8 / 39 secs
  [ Info] Wait for license in 10 / 39 secs
  [ Info] Wait for license in 20 / 39 secs
  [ Info] Wait for license in 30 / 39 secs
```

若屏幕输出如下，则表示客户端连接到浮动授权服务器，客户拒绝排队等待。

```
> copt_cmd
  Cardinal Optimizer v7.0.2. Build date Oct 14 2023
  Copyright Cardinal Operations 2023. All Rights Reserved

  [ Info] Initialize floating client: ./client.ini

  [ Info] connecting to server ...
  [Error] Server error: "no more token available", code = 2
  [Error] Fail to open: ./license.dat

  [Error] Fail to initialize cmdline
```

若屏幕输出如下，则表示客户端无法连接浮动授权服务器，超时后退出。

```
> copt_cmd
  Cardinal Optimizer v7.0.2. Build date Oct 14 2023
  Copyright Cardinal Operations 2023. All Rights Reserved

  [ Info] initialize floating client: ./client.ini

  [ Info] connecting to server ...
  [ Info] wait for license in  2 / 10 secs
  [ Info] wait for license in  4 / 10 secs
  [ Info] wait for license in  6 / 10 secs
  [ Info] wait for license in  8 / 10 secs
  [ Info] wait for license in 10 / 10 secs
  [Error] timeout at waiting for license
  [Error] fail to open: ./license.dat

  [Error] Fail to initialize cmdline
```

4.3 浮动授权服务器管理工具

COPT 浮动授权服务器同时提供了一个对服务器端进行管理的工具 `copt_flstool` ，用来查看服务器的配置信息，并且具有在线修改的功能。

4.3.1 工具说明

在 Windows 的命令行或者 Linux 和 MacOS 的终端中输入下述命令：

```
> copt_flstool
```

屏幕输出如下：

```
> copt_flstool
  COPT Floating Token Server Managing Tool

  copt_flstool [-s server ip] [-p port] [-x passwd] command <param>

  commands are:   addblackrule <127.0.0.1/20[-user@machine]>
                  addwhiterule <127.0.*.*[+user@machine]>
                  getfilters
                  getinfo
                  resetfilters
                  setpasswd <xxx>
                  toggleblackrule <n-th>
                  togglewhiterule <n-th>
                  writefilters
```

上面是这个工具的使用格式说明。用户通过 `-s` 和 `-p` 来设置目标服务器的 IP 地址和端口。如果不写，则默认是本地服务器和端口 7979。如果服务器设置了非空密码，那么需要通过 `-x` 来设置密码后才能成功连接。

这个工具提供的命令包括：

- `AddBlackRule`: 添加一个规则到黑名单里。规则的格式是以 IP 地址开头，这里 IP 可以用通配符匹配，或者固定位数的方式。另外，IP 地址后面还可以包括（+）或者排除（-）给定用户名和给定机器名。
- `AddWhiteRule`: 添加一个规则到白名单里。白名单规则的格式和黑名单一样。
- `GetFilters`: 获取当前所有规则，包括黑名单，白名单以及关于工具的。每个规则前面有个序号，用来作为命令 `ToggleBlackRule` 和 `ToggleWhiteRule` 的参数。
- `GetInfo`: 获取服务器的基本信息，包括 Token 占用数和总数，连接的客户，以及支持的 COPT 版本号。
- `ResetFilters`: 把当前的规则重置为过滤配置文档里的规则。
- `SetPasswd`: 动态更新当前服务器的连接密码。
- `ToggleBlackRule`: 激活或者禁止一条黑名单规则，用从 `GetFilters` 获取的序列号作为参数。

- ToggleWhiteRule: 激活或者禁止一条白名单规则，用从 GetFilters 获取的序列号作为参数。
- WriteFilters: 把当前的规则输出到过滤配置文档。

4.3.2 使用示例

下面的命令列出了服务器为 192.168.1.11 的基本信息。

```
> copt_flstool GetInfo

[ Info] COPT Floating Token Server Managing Tool, COPT v7.0.2 20231014
[ Info] connecting to localhost:7979
[ Info] [command] wait for connecting to floating token server
[ Info] [floating] general info
  # of available tokens is 3 / 3, queue size is 0
  # of active clients is 0
```

如果要在别的机器上使用工具来获取本服务器的信息，需在服务器的过滤配置文件 flsfilters.ini 中，将允许机器的 IP 地址加入到 TooList 部分。下面我们以在允许的机器上获取服务器 192.128.1.11 的信息为例：

```
> copt_flstool -s 192.168.1.11 GetInfo

[ Info] COPT Floating Token Server Managing Tool, COPT v7.0.2 20231014
[ Info] connecting to 192.168.1.11:7979
[ Info] [command] wait for connecting to floating token server
[ Info] [floating] general info
  # of available tokens is 3 / 3, queue size is 0
  # of active clients is 0
```

下面的命令列出了服务器为 192.168.1.11 的所有过滤规则，包括黑名单，白名单以及关于工具的规则。

```
> copt_flstool -s 192.168.1.11 GetFilters

[ Info] COPT Floating Token Server Managing Tool, COPT v7.0.2 20231014
[ Info] connecting to 192.168.1.11:7979
[ Info] [command] wait for connecting to floating token server
[ Info] [floating] filters info
[BlackList]

[WhiteList]

[ToolList]
  [1]  127.0.0.1
```

下面的命令演示了把 IP 地址为 192.168.3.133 用户加入到黑名单中。

```
> copt_flstool -s 192.168.1.11 AddBlackRule 192.168.3.133
```

```
[ Info] COPT Floating Token Server Managing Tool, COPT v7.0.2 20231014
[ Info] connecting to 192.168.1.11:7979
[ Info] [command] wait for connecting to floating token server
[ Info] [floating] server added new black rule (succeeded)
```

此时 GetFilters 可得到本服务器的过滤规则，可以看到已经发生了改变。

```
> copt_flstool -s 192.168.1.11 GetFilters

[ Info] COPT Floating Token Server Managing Tool, COPT v7.0.2 20231014
[ Info] connecting to 192.168.1.11:7979
[ Info] [command] wait for connecting to floating token server
[ Info] [floating] filters info
[BlackList]
  [1] 192.168.3.133

[WhiteList]

[ToolList]
  [1]  127.0.0.1
```

下面的命令演示了切换黑名单规则，把 192.168.3.133 从黑名单中去掉。

```
> copt_flstool -s 192.168.1.11 ToggleBlackRule 1

[ Info] COPT Floating Token Server Managing Tool, COPT v7.0.2 20231014
[ Info] connecting to 192.168.1.11:7979
[ Info] [command] wait for connecting to floating token server
[ Info] [floating] server toggle black rule [1] (succeeded)
```

4.4 服务方式启动

以系统服务的方式启动浮动授权服务器，配置文件以 copt_flserver.service 为例，可通过查看 floating 文件夹下的 readme.txt 来进行配置和操作。readme.txt 中显示如下，分别列出了 Linux 和 MacOS 系统下的操作方式：

```
[Linux] To run copt_flserver as a service with systemd

Add a systemd file
    copy copt_flserver.service to /lib/systemd/system/
    sudo systemctl daemon-reload

Enable new service
    sudo systemctl start copt_flserver.service
    or
    sudo systemctl enable copt_flserver.service
```

(续下页)

(接上页)

```
Restart service
    sudo systemctl restart copt_flserver.service

Stop service
    sudo systemctl stop copt_flserver.service
    or
    sudo systemctl disable copt_flserver.service

Verify service is running
    sudo systemctl status copt_flserver.service

[MacOS] To run copt_flserver as a service with launchctrl

Add a plist file
    copy copt_flserver.plist to /Library/LaunchAgents as current user
    or
    copy copt_flserver.plist to /Library/LaunchDaemons with the key 'UserName'

Enable new service
    sudo launchctl load -w /Library/LaunchAgents/copt_flserver.plist
    or
    sudo launchctl load -w /Library/LaunchDaemons/copt_flserver.plist

Stop service
    sudo launchctl unload -w /Library/LaunchAgents/copt_flserver.plist
    or
    sudo launchctl unload -w /Library/LaunchDaemons/copt_flserver.plist

Verify service is running
    sudo launchctl list shanshu.copt.flserver
```

4.4.1 Linux 系统

下面以 Linux 系统为例，演示通过系统服务方式启动浮动授权服务器。打开终端，输入以下命令进入服务所在路径（以 COPT 远程服务安装在 home 目录下的 eleven 路径为例）。

```
cd /home/eleven/copt_remote70/floating
```

通过任意文本文件打开系统服务的配置文件 copt_flserver.service ，内容如下所示：

```
[Unit]
Description=COPT Floating Token Server

[Service]
WorkingDirectory=/path/to/service
ExecStart=/path/to/service/copt_flserver
Restart=always
```

(续下页)

(接上页)

```
RestartSec=1

[Install]
WantedBy=multi-user.target
```

将路径修改为 copt_flserver 所在的实际路径，修改之后 copt_flserver.service 内容显示如下：

```
[Unit]
Description=COPT Floating Token Server

[Service]
WorkingDirectory=/home/eleven/copt_remote70/floating
ExecStart=/home/eleven/copt_remote70/floating/copt_flserver
Restart=always
RestartSec=1

[Install]
WantedBy=multi-user.target
```

将 copt_flserver.service 文件复制到 /lib/systemd/system/ 路径下，命令如下：

```
sudo cp copt_flserver.service /lib/systemd/system/
```

初次启动系统服务，需要更新系统配置（之后再启动时无需更新），命令如下：

```
sudo systemctl daemon-reload
```

启动系统服务，命令如下：

```
sudo systemctl start copt_flserver.service
```

输入如下命令来验证下系统服务是否在运行：

```
sudo systemctl status copt_flserver.service
```

输入之后显示如下，表示浮动授权服务器的系统服务已启动：

```
copt_flserver.service - COPT Floating Token Server
Loaded: loaded (/lib/systemd/system/copt_flserver.service; enabled; vendor preset:␣
↪enabled)
Active: active (running) since Tue 2021-06-29 11:46:10 CST; 3s ago
Main PID: 3054 (copt_flserver)
    Tasks: 6 (limit: 4915)
CGroup: /system.slice/copt_flserver.service
         3054 /home/eleven/copt_remote70/floating/copt_flserver

eleven-ubuntu systemd[1]: Started COPT Floating Token Server.
eleven-ubuntu COPTCLS[3054]: LWS: 4.1.4-b2011a00, loglevel 1039
```

```
eleven-ubuntu COPTCLS[3054]: NET CLI SRV H1 H2 WS IPv6-absent
eleven-ubuntu COPTCLS[3054]: server started at port 7979
```

若要停止浮动授权服务器系统服务，则输入以下命令：

```
sudo systemctl stop copt_flserver.service
```

4.4.2 MacOS 系统

下面以 MacOS 系统为例，演示通过系统服务方式启动浮动授权服务器。打开终端，输入以下命令进入服务所在路径（以 COPT 远程服务安装在 /Applications 路径为例）。

```
cd /Applications/copt_remote70/floating
```

通过任意文本文件打开系统服务的配置文件 copt_flserver.plist ，内容如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
    <dict>
        <key>Label</key>
        <string>shanshu.copt.flserver</string>
        <key>Program</key>
        <string>/path/to/service/copt_flserver</string>
        <key>RunAtLoad</key>
        <true/>
        <key>KeepAlive</key>
        <true/>
    </dict>
</plist>
```

将路径修改为 copt_flserver 所在的实际路径，修改之后 copt_flserver.plist 内容显示如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
    <dict>
        <key>Label</key>
        <string>shanshu.copt.flserver</string>
        <key>Program</key>
        <string>/Applications/copt_remote70/floating/copt_flserver</string>
        <key>RunAtLoad</key>
        <true/>
        <key>KeepAlive</key>
        <true/>
    </dict>
</plist>
```

将 copt_flserver.plist 文件复制到 /Library/LaunchAgents 路径下，命令如下：

```
sudo cp copt_flserver.plist /Library/LaunchAgents
```

启动系统服务，命令如下：

```
sudo launchctl load -w /Library/LaunchAgents/copt_flserver.plist
```

输入如下命令来验证下系统服务是否在运行：

```
sudo launchctl list shanshu.copt.flserver
```

输入之后显示如下，表示浮动授权服务器的系统服务已启动：

```
{
    "LimitLoadToSessionType" = "System";
    "Label" = "shanshu.copt.flserver";
    "OnDemand" = false;
    "LastExitStatus" = 0;
    "PID" = 16406;
    "Program" = "/Applications/copt_remote70/floating/copt_flserver";
};
```

若要停止浮动授权服务器系统服务，则输入以下命令：

```
sudo launchctl unload -w /Library/LaunchAgents/copt_flserver.plist
```

若是需要指定此设备下的某一用户才有权限启动浮动授权服务器的系统服务，则需要将 UserName 添加到 copt_flserver.plist 中，以 UserName 为 eleven 为例，添加之后 copt_flserver.plist 内容显示如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
    <dict>
        <key>Label</key>
        <string>shanshu.copt.flserver</string>
        <key>Program</key>
        <string>/Applications/copt_remote70/floating/copt_flserver</string>
        <key>UserName</key>
        <string>eleven</string>
        <key>RunAtLoad</key>
        <true/>
        <key>KeepAlive</key>
        <true/>
    </dict>
</plist>
```

将添加 UserName 后的 copt_flserver.plist 文件复制到 /Library/LaunchDaemons 路径下，命令如下：

```
sudo cp copt_flserver.plist /Library/LaunchDaemons
```

启动系统服务，命令如下：

```
sudo launchctl load -w /Library/LaunchDaemons/copt_flserver.plist
```

若要停止浮动授权服务器系统服务，则输入以下命令：

```
sudo launchctl unload -w /Library/LaunchDaemons/copt_flserver.plist
```

第 5 章　COPT 计算集群服务

COPT 计算集群服务器，COPT Compute Cluster，是一个由杉数求解器提供的，允许用户通过局域网在服务器端执行优化计算的服务。该服务支持在 Windows、Linux 和 MacOS 三大系统下运行。

如果 COPT 计算集群服务器成功启动，可以对装有适配版本的杉数求解器客户端提供优化计算服务，不需要客户端具有任何授权文档。也就是说，COPT 计算集群客户端可以在本地建模，然后远程求解，最后交互式地从服务器端获得求解结果。

显然，服务器端同时允许的优化任务数目受限于服务器自己的算力。同时，多台服务器允许组成树状网络拓扑结构的 COPT 计算集群，从而实现在服务器端横向拓展集群算力。

5.1 服务器端安装说明

COPT 计算集群服务器对应的可执行文件是 `copt_cluster` 。其启动时需要验证具有集群类型的许可文件，读取服务器运行所需的配置文件 `cls.ini` ，以及在杉数授权服务器进行远程验证，例如需要验证服务器 IP 符合注册信息中要求的 IP 范围。集群版的许可文件需要通过 `copt_licgen` 工具和注册后得到的许可凭证，由杉数授权服务器远程自动生成。详情描述如下或者参考前文如何申请与配置许可文件 。

5.1.1 安装步骤

杉数提供的 COPT 远程服务安装包，包含了 COPT 计算集群服务器端的程序。用户需先通过客服申请注册并获取 COPT 远程服务安装包后，可直接将安装包解压缩并放置于自定义路径下。过程属于绿色安装，无需配置环境变量，具体安装步骤演示如下：

对于 Windows 用户

请使用 ZIP 解压软件将软件解压至任意目录，推荐软件解压缩到 `C:\Program Files` 目录下。

对于 Linux 用户

鼠标操作解压缩 COPT 远程服务安装包，或在终端上输入下述命令解压缩：

```
tar -xzf CardinalOptimizer-Remote-7.0.2-lnx64.tar.gz
```

解压缩后在当前目录下得到 copt_remote70 文件夹，用户可以将它移动到其它自定义路径下。对于 Admin 用户我们推荐移动到 `/opt` 目录下，对于非 Admin 用户可以放在 `$HOME` 目录下。即在终端下输入下述命令（以 Admin 用户为例）

```
sudo mv copt_remote70 /opt
```

注意，执行该命令需要 `root` 权限。

对于 MacOS 用户

鼠标操作解压缩 COPT 远程服务安装包，或在终端上输入下述命令解压缩：

```
tar -xzf CardinalOptimizer-Remote-7.0.2-universal_mac.tar.gz
```

解压缩后在当前目录下得到 copt_remote70 文件夹，用户可以将它移动到其它自定义路径下。对于 root 用户我们推荐移动到 /Applications 目录下，对于非 root 用户可以放在 $HOME 目录下。即在终端下输入下述命令（以 root 用户为例）

```
mv copt_remote70 /Applications
```

如果在安装或使用杉数求解器时出现如下错误：

```
"libcopt.dylib" cannot be opened because the developer cannot be verified.
macOS cannot verify that this app is free from malware.
```

或者出现类似的动态库签名问题。则使用管理员权限，在终端上执行如下命令，以绕过 MacOS 系统关于动态库的加载检查。

```
xattr -d com.apple.quarantine CardinalOptimizer-Remote-7.0.2-universal_mac.tar.gz
```

或者：

```
xattr -dr com.apple.quarantine /Applications/copt_remote70
```

5.1.2 许可文件

在成功安装 COPT 远程服务包后，还需配置集群许可文件。我们推荐把获得的许可文件 license.dat 和 license.key 放置在 COPT 远程服务安装路径下的 cluster 目录。

下面讲解在不同系统下，如何通过 copt_licgen 工具和许可凭证信息 key 来获取许可文件。

注意

如果用户已经获取到 license.dat 和 license.key 这两个授权文档，则无需重复获取了。可以跳过如下获取许可文件的步骤，直接参阅配置文件 。

对于 Windows 用户

需要打开命令行工具，也就是 cmd 。假如 COPT 远程服务已安装在 "C:\Program Files" ，在命令行工具窗口输入如下命令后，进入 COPT 远程服务安装路径下的 cluster 目录：

```
cd "C:\Program Files\copt_remote70\cluster"
```

假如用户的许可凭证信息为 7483dff0863ffdae9fff697d3573e8bc ，因为生成许可文件的 copt_licgen 在 COPT 远程服务安装路径下的 tools 文件夹内，则可以在命令行工具窗口输入下述命令获取杉数 COPT 的许可文件 license.dat 和 license.key 。

```
..\tools\copt_licgen -key 7483dff0863ffdae9fff697d3573e8bc
```

对于 Linux 和 MacOS 用户

假如 COPT 远程服务安装在 "/Applications" 目录下，在终端输入以下命令进入集群服务端目录（以 MacOS 系统为例）：

```
cd /Applications/copt_remote70/cluster
```

假如用户的许可凭证信息为 7483dff0863ffdae9fff697d3573e8bc ，则可以在终端输入下述命令获取杉数 COPT 的许可文件 license.dat 和 license.key 。

```
../tools/copt_licgen -key 7483dff0863ffdae9fff697d3573e8bc
```

特别地，如果用户不是在 COPT 远程服务安装路径下的 cluster 目录下配置的许可文件，建议将 license.dat 和 license.key 两个许可文件移至此路径下。可手动将这两个文件放到 cluster 目录下，也可以输入下述命令移动这两个许可文件（以 MacOS 系统为例）。

```
mv license.* /Application/copt_remote70/cluster
```

5.1.3 配置文件

COPT 计算集群服务器的配置文件 cls.ini ，如下表所示。

```
[Main]
Port = 7878
# Number of total tokens, what copt jobs can run simutaneously up to.
NumToken = 3
# Password is case-sensitive and default is emtpy;
# It applies to both copt clients and cluster nodes.
PassWd =
# Data folder of cluster relative to its binary folder,
# where multiple versions of copt libraries and related licenses reside.
DataFolder = ./data

[SSL]
# Needed if connecting using SSL/TLS
CaFile =
CertFile =
CertkeyFile =

[Licensing]
# If empty or default license name, it is from binary folder;
# To get license files from cwd, add prefix "./";
# Full path is supported as well.
LicenseFile = license.dat
PubkeyFile = license.key

[Cluster]
# Host name and port of parent node in cluster.
# Default is empty, meaning not connecting to other node.
Parent =
PPort = 7878
```

(续下页)

(接上页)

```
[Filter]
# default policy 0 indicates accepting all connections, except for ones in blacklist
# otherwise, denying all connections except for ones in whitelist
DefaultPolicy = 0
UseBlackList = true
UseWhiteList = true
FilterListFile = clsfilters.ini
```

这里，配置文件可以设置集群服务器连接端口，客户端需要连接这个端口才能通讯和获得服务。用户可以通过 NumToken = 3 来设置 COPT 计算集群服务器同时允许的优化任务数目，这个数目可以按本服务器的算力来估算。连接密码可以通过 PassWd 来改变，默认是空，即表示客户连接不需要设定密码。更安全的连接方式是下面 SSL 部分，用户可以通过 RSA 证书来加密通讯内容。

COPT 计算集群服务器可以预先安装多个版本的 COPT 求解器，只有 COPT 版本匹配的客户才会被批准。服务器端安装的 COPT 求解器所在路径则由 DataFolder 设定。COPT 计算集群服务预装了和服务器程序版本匹配的 COPT 求解器，用户可以参照着来设置需要的不同版本。

比如，如果集群服务器端预安装版本号为 v7.0.2 的 COPT 求解器，还想安装版本为 v4.0.7 的 COPT 求解器，只需要将 COPT v4.0.7 的 C 算法库，放到 DataFolder 的子目录./data/copt/4.0.7/即可。并且这个安装过程不需要重启服务器。具体地说，以 Linux 平台为例，就是从 COPT v4.0.7 的安装目录 $COPT_HOME/lib/ 复制 libcopt.so 到集群服务器程序的子目录./data/copt/4.0.7/。

值得注意的是，如果集群服务器的版本是 v7.0.2，但用户想安装更新的 COPT 求解器，比如 COPT v7.5.0 求解器，也是允许的。这时候，除了需要把 COPT v7.5.0 的 C 算法库放入./data/copt/7.5.0/，还需要对应的单机版授权文档才能加载 COPT v7.5.0 求解器，所以再复制一份 v7.5.0 的单机授权文档到子目录./data/copt/7.5.0/。当然如果杉数求解器有重大改动，上述简单安装的方式可能不可行，那么用户需要升级集群服务器到最新版本才可以。

以下示例是 Linux 系统里的集群服务器的安装目录结构，假设支持三个版本的 COPT 求解器，默认的 v7.0.2，旧版本 v4.0.7，和新版本 v7.5.0。

```
~/copt_remote70/cluster
|  cls.ini
|  copt_cluster
|  license.dat -> cluster license v7.0.2
|  license.key
|
|_ data
    |_ copt
        |_ 4.0.7
            libcopt.so
        |_ 7.0.2
            libcopt.so
        |_ 7.5.0
            libcopt.so
            license.dat -> license v7.5.0
```

```
license.key
```

在下面 Licensing 部分，配置文件需要设定授权文档的读取路径。就如上面注释描述的，空缺或者默认的授权文档名表示 COPT 计算集群服务器从可执行文件所在目录下读取授权文档。

如果用户在当前目录下启动集群服务器，并且许可文件也在当前目录下，通过修改 LicenseFile = ./license.dat 和 PubkeyFile = ./license.key ，可以让 COPT 计算集群服务器从当前目录读取授权文档。

我们知道杉数求解器还允许客户通过环境变量 COPT_LICENSE_DIR 设置授权文档所在的路径。详情可参考如何安装杉数求解器 。在这种情况下，需要修改配置文件为完整的环境变量路径，COPT 计算集群服务器才能读取环境变量下的授权文档。

在 Cluster 部分，用户可以设定连接的父节点（IP 和端口），本服务器可以通过这个设置来加入到父节点所在的集群中去。

最后是 Filter 部分，DefaultPolicy 默认设为 0，表示除了在黑名单里的用户，其他均允许连接到本授权服务器；反之，除了在白名单中的用户，其他设备均无法连接到本授权服务器。UseBlackList 为 True 表示使用黑名单列表，UseWhiteList 为 True 表示使用白名单列表。FilterListFile 表示集群服务器过滤配置文件名为 clsfilters.ini 。其格式举例如下：

```
[BlackList]
# 127.0.*.* + user@machine*

[WhiteList]
# 127.0.1.2/16 - user@machine*

[ToolList]
# only tool client at server side can access by default
127.0.0.1/32
```

BlackList 和 WhiteList 表示可添加规则到黑名单以及白名单里。规则的格式是以 IP 地址开头，这里 IP 可以用通配符匹配，或者固定位数的方式。另外，IP 地址后面还可以包括 (+) 或者排除 (-) 给定用户名和给定机器名。其中用户名 user 是指 username ，MacOS/Linux 系统用户可以在终端通过命令 whoami 来查询；机器名 machine 是指 hostname ，MacOS/Linux 系统用户可以在终端通过命令 hostname 来查询。ToolList 是对工具使用者的访问限制，添加相关 IP 地址表示允许此远程机器访问。需注意的是，配置文件 clsfilters.ini 更改之后，可使用 ResetFilters 命令把当前的规则重置为过滤配置文档里的规则。也可使用 WriteFilters 命令把当前的规则输出到过滤配置文档。

5.1.4 使用示例

如果正确的集群版授权文档安装在集群服务器可执行文件所在的目录下，要启动 COPT 计算集群服务器，只需在 Windows 的命令行或者 Linux 和 MacOS 的终端中输入下述命令：

```
copt_cluster
```

若屏幕输出如下，则表示成功启动集群服务器，启动后会监视和管理客户端的申请。如果用户输入字符 Q ，则集群服务器会关闭退出。

```
> copt_cluster
```

```
  [ Info] start COPT Compute Cluster, v7.0.2 20231014
  [ Info] [NODE] node has been initialized
  [ Info] server started at port 7878
```

如果本地集群版授权文档验证不通过，或者远程杉数授权服务器连接错误，则可能出现如下输出。

```
> copt_cluster
  [ Info] start COPT Compute Cluster, v7.0.2 20231014
  [Error] Invalid signature in public key file
  [Error] Fail to verify local license
```

和

```
> copt_cluster
  [ Info] start COPT Compute Cluster, v7.0.2 20231014
  [Error] Error to connect license server
  [Error] Fail to verify cluster license by server
```

5.2 客户端使用说明

COPT 计算集群客户端可以是 COPT 命令行交互工具，或者通过接口调用杉数求解器动态库的方式，比如 COPT Python 接口。如果客户没有任何单机许可，那么集群许可是一种更灵活的替代方式。所有使用 COPT 的客户端都可以通过集群服务器来合法使用杉数求解器。

5.2.1 安装和配置

首先用户需要确保已在客户端正确安装杉数求解器，详情可参考如何安装杉数求解器。成功安装 COPT 客户端，用户不需要配置本地的许可文件，但需要配置 COPT 计算集群客户端的配置文件 `client.ini`，如下表所示。

```
Cluster = 192.168.1.11
Port = 7878
QueueTime = 600
Passwd =
```

上述配置文件表示客户端会尝试和 192.168.1.11:7878 连接，且排队等待时间最多为 600 秒。这里，如果 `Cluster` 内容空缺，默认设为 `localhost` 。`QueueTime` 或者 `WaitTime` 空缺则认为是 0，即如果集群服务器的 Token 数目已经被占满就不等待，直接错误退出。`Port` 如果空缺，则设为默认的 7878，但要确保和集群服务器的配置文件里设的端口一致。注意客户端配置文件中的关键词是并不区分大小写。

使用 COPT 计算集群服务的前提是客户端首先发现的是配置文本文件 `client.ini` ，而不是本地许可文件。但和寻找本地许可文件的途径一样，客户端会从当前目录，环境变量目录 `COPT_LICENSE_DIR` 和客户端可执行文件所在目录，依次查看是否有连接服务器的配置文件 `client.ini` 。所以，如果用户想使用 COPT 计算集群服务，需要确保 COPT 客户端优先找到的是配置文件，而不是本地授权许可（如果有的话）。比如，用户已经在环境目录下有了本地许可，那么用户还可以在当前目录（不同于环境目录）下，建立配置文件 `client.ini` 来优先使用集群服务的方式。

如果是调用杉数求解器动态库方式，比如 COPT Python 接口，在建模中创建 COPT 优化求解环境时，会按照配置文件 `client.ini` 的设置，从集群服务器获取求解服务。这里，每个 COPT 求解环境对象需要建立和集群服务器的连接并占用一个 Token，建模结束后释放 COPT 求解环境对象时，会通知服务器释放占用的 Token。

5.2.2 使用示例

假如用户在当前目录下创建了客户端配置文件 `client.ini` （同时没有本地许可文件），下面以交互式命令行工具 `copt_cmd` 为例，演示如何建立与集群服务器的连接。首先，在 Windows 的命令行或者 Linux 和 MacOS 的终端中输入下述命令：

```
copt_cmd
```

若屏幕输出如下，则表示客户端成功连接到集群服务器，可以在本地建模，然后远程求解。

```
> copt_cmd
  Cardinal Optimizer v7.0.2. Build date Oct 14 2023
  Copyright Cardinal Operations 2023. All Rights Reserved

  [ Info] initialize cluster client with ./client.ini

  [ Info] wait for server in 0 / 39 secs
  [ Info] connecting to cluster server 192.168.1.11:7878
COPT>
```

若屏幕输出如下，则表示客户端连接到集群服务器，但由于 Token 数目有限，需要排队等待。

```
> copt_cmd
  Cardinal Optimizer v7.0.2. Build date Oct 14 2023
  Copyright Cardinal Operations 2023. All Rights Reserved

  [ Info] initialize cluster client with ./client.ini

  [ Info] wait for server in 0 / 39 secs
  [ Info] connecting to cluster server 192.168.1.11:7878

  [ Warn] wait in queue of size 5
  [ Info] wait for license in  2 / 39 secs
  [ Info] wait for license in  4 / 39 secs
  [ Info] wait for license in  6 / 39 secs
  [ Info] wait for license in  8 / 39 secs
  [ Info] wait for license in 10 / 39 secs
  [ Info] wait for license in 20 / 39 secs
  [ Info] wait for license in 30 / 39 secs
  [Error] timeout at waiting for server approval
  [Error] Fail to initialize copt command-line tool
```

若屏幕输出如下，则表示客户端连接到集群服务器，但用户拒绝排队等待。然后客户端立即返回错误后退出。

```
> copt_cmd
  Cardinal Optimizer v7.0.2. Build date Oct 14 2023
  Copyright Cardinal Operations 2023. All Rights Reserved

  [ Info] initialize cluster client with ./client.ini

  [ Info] wait for server in 0 / 9 secs
  [ Info] connecting to cluster server 192.168.1.11:7878
  [ Warn] server error: "no more token available", code = 129
  [Error] Fail to initialize copt command-line tool
```

若屏幕输出如下，则表示客户端无法连接集群服务器，超时后退出。

```
> copt_cmd
  Cardinal Optimizer v7.0.2. Build date Oct 14 2023
  Copyright Cardinal Operations 2023. All Rights Reserved

  [ Info] initialize cluster client with ./client.ini

  [ Info] wait for server in 0 / 39 secs
  [ Info] connecting to cluster server 192.168.1.11:7878
  [ Info] wait for license in  2 / 39 secs
  [ Info] wait for license in  4 / 39 secs
  [ Info] wait for license in  6 / 39 secs
  [ Info] wait for license in  8 / 39 secs
  [ Info] wait for license in 10 / 39 secs
  [ Info] wait for license in 20 / 39 secs
  [ Info] wait for license in 30 / 39 secs
  [Error] timeout at waiting for server approval
  [Error] Fail to initialize copt command-line tool
```

5.3 集群服务器管理工具

COPT 计算集群服务器同时提供了一个对服务器端进行管理的工具 `copt_clstool` ，用来查看集群服务器的配置信息，并且具有在线修改的功能。

5.3.1 工具说明

在 Windows 的命令行或者 Linux 和 MacOS 的终端中输入下述命令：

```
> copt_clstool
```

屏幕输出如下：

```
> copt_clstool
  COPT Cluster Managing Tool

  copt_clstool [-s server ip] [-p port] [-x passwd] command <param>

  commands are:   addblackrule <127.0.0.1/20[-user@machine]>
                  addwhiterule <127.0.*.*[+user@machine]>
                  getfilters
                  getinfo
                  getnodes
                  reload
                  resetfilters
                  setparent <xxx:7878>
                  setpasswd <xxx>
                  settoken  <num>
                  toggleblackrule <n-th>
                  togglewhiterule <n-th>
                  writefilters
```

上面是这个工具的使用格式说明。用户通过 `-s` 和 `-p` 来设置目标服务器的 IP 地址和端口。如果不写，则默认是本地服务器和端口 7878。如果服务器设置了非空密码，那么需要通过 `-x` 来设置密码后才能成功连接。

这个工具提供的命令包括：

- `AddBlackRule`: 添加一个规则到黑名单里。规则的格式是以 IP 地址开头，这里 IP 可以用通配符匹配，或者固定位数的方式。另外，IP 地址后面还可以包括（+）或者排除（-）给定用户名和给定机器名。
- `AddWhiteRule`: 添加一个规则到白名单里。白名单规则的格式和黑名单一样。
- `GetFilters`: 获取当前所有规则，包括黑名单，白名单以及关于工具的。每个规则前面有个序号，用来作为命令 `ToggleBlackRule` 和 `ToggleWhiteRule` 的参数。
- `GetInfo`: 获取服务器的基本信息，包括 Token 占用数和总数，连接的客户，以及支持的 COPT 版本号。
- `GetNodes`: 获取服务器作为集群节点的连接信息，包括父节点和子节点。
- `Reload`: 动态更新当前服务器的所有子节点的占用情况。在发生占用数据不一致的情形时，手动更新保持一致。
- `ResetFilters`: 把当前的规则重置为过滤配置文档里的规则。
- `SetParent`: 动态更新当前服务器的父节点并连接。调整网络结构时避免等待客户端任务的结束。

- SetPasswd: 动态更新当前服务器的连接密码。
- SetToken: 动态更新当前服务器的 Token 数。
- ToggleBlackRule: 激活或者禁止一条黑名单规则，用从 GetFilters 获取的序列号作为参数。
- ToggleWhiteRule: 激活或者禁止一条白名单规则，用从 GetFilters 获取的序列号作为参数。
- WriteFilters: 把当前的规则输出到过滤配置文档。

5.3.2 使用示例

下面的命令列出了服务器为 192.168.1.11 的基本信息。

```
> copt_clstool -s 192.168.1.11 GetInfo

[ Info] COPT Cluster Managing Tool, COPT v7.0.2 20231014
[ Info] connecting to 192.168.1.11:7878
[ Info] [command] wait for connecting to cluster
[ Info] [cluster] general info
  # of available tokens is 3 / 3, queue size is 0
  # of active clients is 0
  # of installed COPT versions is 1
    COPT v7.0.2
```

下面的命令列出了服务器为 192.168.1.11 的集群连接信息。

```
> copt_clstool -s 192.168.1.11 GetNodes

[ Info] COPT Cluster Managing Tool, COPT v7.0.2 20231014
[ Info] connecting to 192.168.1.11:7878
[ Info] [command] wait for connecting to cluster
[ Info] [cluster] node info
  [Parent] (null):7878 (Lost)
  [Child] Node_192.168.1.12:7878_N0001, v2.0=3
  Total num of child nodes is 1
```

下面的命令把服务器为 192.168.1.11 的 Token 数目从原来的 3，在线修改为目前的 0。

```
> copt_clstool -s 192.168.1.11 SetToken 0

[ Info] COPT Cluster Managing Tool, COPT v7.0.2 20231014
[ Info] connecting to 192.168.1.11:7878
[ Info] [command] wait for connecting to cluster
[ Info] [cluster] total token was 3 and now set to 0
```

下面的命令列出了服务器为 192.168.1.11 的所有过滤规则，包括黑名单，白名单以及关于工具的规则。

```
> copt_clstool -s 192.168.1.11 GetFilters

[ Info] COPT Cluster Managing Tool, COPT v7.0.2 20231014
```

```
[ Info] connecting to 192.168.1.11:7979
[ Info] [command] wait for connecting to cluster
[ Info] [cluster] filters info
[BlackList]

[WhiteList]

[ToolList]
  [1]  127.0.0.1
```

下面的命令演示了把 IP 地址为 192.168.3.133 用户加入到黑名单中。

```
> copt_clstool -s 192.168.1.11 AddBlackRule 192.168.3.133

[ Info] COPT Cluster Managing Tool, COPT v7.0.2 20231014
[ Info] connecting to 192.168.1.11:7979
[ Info] [command] wait for connecting to cluster
[ Info] [cluster] server added new black rule (succeeded)
```

此时 GetFilters 可得到本服务器的过滤规则，可以看到已经发生了改变。

```
> copt_clstool -s 192.168.1.11 GetFilters

[ Info] COPT Cluster Managing Tool, COPT v7.0.2 20231014
[ Info] connecting to 192.168.1.11:7979
[ Info] [command] wait for connecting to cluster
[ Info] [cluster] filters info
[BlackList]
  [1] 192.168.3.133

[WhiteList]

[ToolList]
  [1]  127.0.0.1
```

下面的命令演示了切换黑名单规则，把 192.168.3.133 从黑名单中去掉。

```
> copt_clstool -s 192.168.1.11 ToggleBlackRule 1

[ Info] COPT Cluster Managing Tool, COPT v7.0.2 20231014
[ Info] connecting to 192.168.1.11:7979
[ Info] [command] wait for connecting to cluster
[ Info] [cluster] server toggle black rule [1] (succeeded)
```

5.4 服务方式启动

以系统服务的方式启动集群服务器，配置文件以 `copt_cluster.service` 为例，可通过查看 `cluster` 文件夹下的 `readme.txt` 来进行配置和操作。`readme.txt` 中显示如下，分别列出了 Linux 和 MacOS 系统下的操作方式：

```
[Linux] To run copt_cluster as a service with systemd

Add a systemd file
    copy copt_cluster.service to /lib/systemd/system/
    sudo systemctl daemon-reload

Enable new service
    sudo systemctl start copt_cluster.service
    or
    sudo systemctl enable copt_cluster.service

Restart service
    sudo systemctl restart copt_cluster.service

Stop service
    sudo systemctl stop copt_cluster.service
    or
    sudo systemctl disable copt_cluster.service

Verify service is running
    sudo systemctl status copt_cluster.service

[MacOS] To run copt_cluster as a service with launchctrl

Add a plist file
    copy copt_cluster.plist to /Library/LaunchAgents as current user
    or
    copy copt_cluster.plist to /Library/LaunchDaemons with the key 'UserName'

Enable new service
    sudo launchctl load -w /Library/LaunchAgents/copt_cluster.plist
    or
    sudo launchctl load -w /Library/LaunchDaemons/copt_cluster.plist

Stop service
    sudo launchctl unload -w /Library/LaunchAgents/copt_cluster.plist
    or
    sudo launchctl unload -w /Library/LaunchDaemons/copt_cluster.plist

Verify service is running
    sudo launchctl list shanshu.copt.cluster
```

5.4.1 Linux 系统

下面以 Linux 系统为例，演示通过系统服务方式启动集群服务器。打开终端，输入以下命令进入集群服务所在路径（以 COPT 远程服务安装在 home 目录下的 eleven 路径为例）。

```
cd /home/eleven/copt_remote70/cluster
```

通过任意文本文件打开系统服务的配置文件 copt_cluster.service ，内容如下所示：

```
[Unit]
Description=COPT Cluster Server

[Service]
WorkingDirectory=/path/to/service
ExecStart=/path/to/service/copt_cluster
Restart=always
RestartSec=1

[Install]
WantedBy=multi-user.target
```

将路径修改为 copt_cluster 所在的实际路径，修改之后 copt_cluster.service 内容显示如下：

```
[Unit]
Description=COPT Cluster Server

[Service]
WorkingDirectory=/home/eleven/copt_remote70/cluster
ExecStart=/home/eleven/copt_remote70/cluster/copt_cluster
Restart=always
RestartSec=1

[Install]
WantedBy=multi-user.target
```

将 copt_cluster.service 文件复制到 /lib/systemd/system/ 路径下，命令如下：

```
sudo cp copt_cluster.service /lib/systemd/system/
```

初次启动系统服务，需要更新系统配置（之后再启动时无需更新），命令如下：

```
sudo systemctl daemon-reload
```

启动系统服务，命令如下：

```
sudo systemctl start copt_cluster.service
```

输入如下命令来验证下系统服务是否在运行：

```
sudo systemctl status copt_cluster.service
```

输入之后显示如下，表示集群服务器系统服务已启动：

```
copt_cluster.service - COPT Cluster Server
Loaded: loaded (/lib/systemd/system/copt_cluster.service; enabled; vendor preset:␣
↪enabled)
Active: active (running) since Sat 2021-08-28 11:46:10 CST; 3s ago
Main PID: 3054 (copt_cluster)
    Tasks: 6 (limit: 4915)
CGroup: /system.slice/copt_cluster.service
           3054 /home/eleven/copt_remote70/cluster/copt_cluster

eleven-ubuntu systemd[1]: Started COPT Cluster Server.
eleven-ubuntu COPTCLS[3054]: LWS: 4.1.4-b2011a00, loglevel 1039
eleven-ubuntu COPTCLS[3054]: NET CLI SRV H1 H2 WS IPv6-absent
eleven-ubuntu COPTCLS[3054]: server started at port 7878
eleven-ubuntu COPTCLS[3054]: LWS: 4.1.4-b2011a00, loglevel 1039
eleven-ubuntu COPTCLS[3054]: NET CLI SRV H1 H2 WS IPv6-absent
eleven-ubuntu COPTCLS[3054]: [NODE] node has been initialized
```

若要停止集群服务器系统服务，则输入以下命令：

```
sudo systemctl stop copt_cluster.service
```

5.4.2 MacOS 系统

下面以 MacOS 系统为例，演示通过系统服务方式启动集群服务器。打开终端，输入以下命令进入集群服务所在路径（以 COPT 远程服务安装在 "/Applications" 路径为例）。

```
cd /Applications/copt_remote70/cluster
```

通过任意文本文件打开系统服务的配置文件 copt_cluster.plist ，内容如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
    <dict>
        <key>Label</key>
        <string>shanshu.copt.cluster</string>
        <key>Program</key>
        <string>/path/to/service/copt_cluster</string>
        <key>RunAtLoad</key>
        <true/>
        <key>KeepAlive</key>
        <true/>
    </dict>
</plist>
```

将路径修改为 copt_cluster 所在的实际路径，修改之后 copt_cluster.plist 内容显示如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
    <dict>
        <key>Label</key>
        <string>shanshu.copt.cluster</string>
        <key>Program</key>
        <string>/Applications/copt_remote70/cluster/copt_cluster</string>
        <key>RunAtLoad</key>
        <true/>
        <key>KeepAlive</key>
        <true/>
    </dict>
</plist>
```

将 copt_cluster.plist 文件复制到 /Library/LaunchAgents 路径下，命令如下：

```
sudo cp copt_cluster.plist /Library/LaunchAgents
```

启动系统服务，命令如下：

```
sudo launchctl load -w /Library/LaunchAgents/copt_cluster.plist
```

输入如下命令来验证下系统服务是否在运行：

```
sudo launchctl list shanshu.copt.cluster
```

输入之后显示如下，表示集群服务器系统服务已启动：

```
{
    "LimitLoadToSessionType" = "System";
    "Label" = "shanshu.copt.cluster";
    "OnDemand" = false;
    "LastExitStatus" = 0;
    "PID" = 16406;
    "Program" = "/Applications/copt_remote70/cluster/copt_cluster";
};
```

若要停止集群服务器系统服务，则输入以下命令：

```
sudo launchctl unload -w /Library/LaunchAgents/copt_cluster.plist
```

若是需要指定此设备下的某一用户才有权限启动集群服务器系统服务，则需要将 UserName 添加到 copt_cluster.plist 中，以 UserName 为 eleven 为例，添加之后 copt_cluster.plist 内容显示如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
    <dict>
        <key>Label</key>
```

```
        <string>shanshu.copt.cluster</string>
        <key>Program</key>
        <string>/Applications/copt_remote70/cluster/copt_cluster</string>
        <key>UserName</key>
        <string>eleven</string>
        <key>RunAtLoad</key>
        <true/>
        <key>KeepAlive</key>
        <true/>
    </dict>
</plist>
```

将添加 UserName 后的 copt_cluster.plist 文件复制到 /Library/LaunchDaemons 路径下，命令如下：

```
sudo cp copt_cluster.plist /Library/LaunchDaemons
```

启动系统服务，命令如下：

```
sudo launchctl load -w /Library/LaunchDaemons/copt_cluster.plist
```

若要停止集群服务器系统服务，则输入以下命令：

```
sudo launchctl unload -w /Library/LaunchDaemons/copt_cluster.plist
```

第 6 章　COPT 在线授权服务

COPT 的在线许可授权服务（Web Licensing Service）为用户提供远程授权服务，无论客户端是位于云端还是容器内，只要能通过 HTTPS 协议访问 Internet，就可以从 COPT 的 Web License 授权服务端获取 Token 来运行 COPT，无需绑定任何硬件信息。

因此，相比于传统的授权方式，在线许可不受限于固定硬件环境，灵活支持多个用户使用，适用于容器化部署（如 Docker）和云服务器部署（需联网环境），总结其特点如下：

1. 不绑定硬件信息，灵活支持云端和容器化部署等（硬件信息不固定）场景；
2. 无版本限制，支持跨版本使用；
3. 至少有一台机器需要连接到互联网（和 web server 远程通信）；
4. 提供 Web License 网页端 供用户自行操作获取和管理 license，使用友好方便。

同时，对应于传统的授权方式，在线许可也包括：在线服务器许可（Web License-Server）、在线浮动许可（Web License-Floating）和在线集群许可（Web License-Cluster）这三个子类别。

1. 在线服务器许可

支持在云开发环境中部署运行 COPT 的服务器（无需绑定任何机器的硬件信息），支持在服务器上同时进行多个建模和求解任务。

2. 在线浮动许可

支持在云开发环境中部署浮动令牌服务器（该台服务器需要能够联网，通过在线许可获取远程授权），通过浮动令牌服务器（服务端）为局域网内的其他机器（客户端）进行授权运行 COPT。

3. 在线集群许可

支持在云开发环境中设置一台或多台计算集群服务器，可以在局域网内本地机器（客户端）上进行建模，在远程集群服务器（服务端）求解，以高效利用服务器强大的计算资源。

用户需要在 杉数 Web License 网页端 进行注册，登录后即可 **在网页端直接申请**获取上述三种类型的在线许可，新建 API Keys 获取许可文件，并对 token 占用情况等进行管理。

关于在线许可的更多介绍和安装使用教程，请进一步参考 Web License 网页端帮助文档 。如在使用过程中遇到问题，请联系我们获取进一步的帮助：

表 6.1: 联系信息

联系邮箱	描述
coptsales@shanshu.ai	商务支持
coptsupport@shanshu.ai	技术支持

第 7 章　COPT 快速入门

7.1 C 接口

本章通过一个简单的示例演示如何使用杉数求解器的 C 接口，待求解的问题数学形式如公式 7.1 所示：

$$
\begin{aligned}
&\text{最大化：} \\
&\qquad 1.2x + 1.8y + 2.1z \\
&\text{约束：} \\
&\qquad 1.5x + 1.2y + 1.8z \leq 2.6 \\
&\qquad 0.8x + 0.6y + 0.9z \geq 1.2 \\
&\text{变量范围：} \\
&\qquad 0.1 \leq x \leq 0.6 \\
&\qquad 0.2 \leq y \leq 1.5 \\
&\qquad 0.3 \leq z \leq 2.8
\end{aligned}
\tag{7.1}
$$

7.1.1 示例解析

使用杉数求解器的 C 接口求解与分析上述问题的代码见 代码 7.1 ：

代码 7.1: `lp_ex1.c`

```
/*
 * This file is part of the Cardinal Optimizer, all rights reserved.
 */

/*
 * The problem to solve:
 *
 *  Maximize:
 *    1.2 x + 1.8 y + 2.1 z
 *
 *  Subject to:
 *    1.5 x + 1.2 y + 1.8 z <= 2.6
 *    0.8 x + 0.6 y + 0.9 z >= 1.2
 *
 *  where:
```

(续下页)

(接上页)

```
 *    0.1 <= x <= 0.6
 *    0.2 <= y <= 1.5
 *    0.3 <= z <= 2.8
 */

#include "copt.h"

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
  int errcode = 0;

  copt_env* env = NULL;
  copt_prob* prob = NULL;

  // Create COPT environment
  errcode = COPT_CreateEnv(&env);
  if (errcode)
    goto COPT_EXIT;

  // Create COPT problem
  errcode = COPT_CreateProb(env, &prob);
  if (errcode)
    goto COPT_EXIT;

  /*
   * Add variables
   *
   *   obj: 1.2 C0 + 1.8 C1 + 2.1 C2
   *
   *   var:
   *     0.1 <= C0 <= 0.6
   *     0.2 <= C1 <= 1.5
   *     0.3 <= C2 <= 2.8
   *
   */
  int ncol = 3;
  double colcost[] = {1.2, 1.8, 2.1};
  double collb[] = {0.1, 0.2, 0.3};
  double colub[] = {0.6, 1.5, 1.8};

  errcode = COPT_AddCols(prob, ncol, colcost, NULL, NULL, NULL, NULL, NULL, collb, colub,␣
↪NULL);
  if (errcode)
    goto COPT_EXIT;
```

(续下页)

(接上页)

```
62
63    /*
64     * Add constraints
65     *
66     *   r0: 1.5 C0 + 1.2 C1 + 1.8 C2 <= 2.6
67     *   r1: 0.8 C0 + 0.6 C1 + 0.9 C2 >= 1.2
68     */
69    int nrow = 2;
70    int rowbeg[] = {0, 3};
71    int rowcnt[] = {3, 3};
72    int rowind[] = {0, 1, 2, 0, 1, 2};
73    double rowelem[] = {1.5, 1.2, 1.8, 0.8, 0.6, 0.9};
74    char rowsen[] = {COPT_LESS_EQUAL, COPT_GREATER_EQUAL};
75    double rowrhs[] = {2.6, 1.2};
76
77    errcode = COPT_AddRows(prob, nrow, rowbeg, rowcnt, rowind, rowelem, rowsen, rowrhs, NULL,␣
   ↪NULL);
78    if (errcode)
79      goto COPT_EXIT;
80
81    // Set parameters and attributes
82    errcode = COPT_SetDblParam(prob, COPT_DBLPARAM_TIMELIMIT, 10);
83    if (errcode)
84      goto COPT_EXIT;
85    errcode = COPT_SetObjSense(prob, COPT_MAXIMIZE);
86    if (errcode)
87      goto COPT_EXIT;
88
89    // Solve problem
90    errcode = COPT_SolveLp(prob);
91    if (errcode)
92      goto COPT_EXIT;
93
94    // Analyze solution
95    int lpstat = COPT_LPSTATUS_UNSTARTED;
96    double lpobjval;
97    double* lpsol = NULL;
98    int* colstat = NULL;
99
100   errcode = COPT_GetIntAttr(prob, COPT_INTATTR_LPSTATUS, &lpstat);
101   if (errcode)
102     goto COPT_EXIT;
103
104   if (lpstat == COPT_LPSTATUS_OPTIMAL)
105   {
106     lpsol = (double*)malloc(ncol * sizeof(double));
107     colstat = (int*)malloc(ncol * sizeof(int));
```

(续下页)

(接上页)

```
    errcode = COPT_GetLpSolution(prob, lpsol, NULL, NULL, NULL);
    if (errcode)
      goto COPT_EXIT;

    errcode = COPT_GetBasis(prob, colstat, NULL);
    if (errcode)
      goto COPT_EXIT;

    errcode = COPT_GetDblAttr(prob, COPT_DBLATTR_LPOBJVAL, &lpobjval);
    if (errcode)
      goto COPT_EXIT;

    printf("\nObjective value: %.6f\n", lpobjval);

    printf("Variable solution: \n");
    for (int i = 0; i < ncol; ++i)
      printf("  x[%d] = %.6f\n", i, lpsol[i]);

    printf("Variable basis status: \n");
    for (int i = 0; i < ncol; ++i)
      printf("  x[%d]: %d\n", i, colstat[i]);

    free(lpsol);
    free(colstat);
  }

  // Write problem, solution and modified parameters to files
  errcode = COPT_WriteMps(prob, "lp_ex1.mps");
  if (errcode)
    goto COPT_EXIT;
  errcode = COPT_WriteBasis(prob, "lp_ex1.bas");
  if (errcode)
    goto COPT_EXIT;
  errcode = COPT_WriteSol(prob, "lp_ex1.sol");
  if (errcode)
    goto COPT_EXIT;
  errcode = COPT_WriteParam(prob, "lp_ex1.par");
  if (errcode)
    goto COPT_EXIT;

  // Error handling
COPT_EXIT:
  if (errcode)
  {
    char errmsg[COPT_BUFFSIZE];

```

(续下页)

(接上页)

```
    COPT_GetRetcodeMsg(errcode, errmsg, COPT_BUFFSIZE);
    printf("ERROR %d: %s\n", errcode, errmsg);

    return 0;
  }

  // Delete problem and environment
  COPT_DeleteProb(&prob);

  COPT_DeleteEnv(&env);

  return 0;
}
```

接下来我们将基于上述代码分步骤讲解求解与分析过程，详细的 C 接口使用说明请用户查阅*C API 参考手册* 。

创建环境

对于任意求解任务，杉数求解器要求首先创建求解环境，并通过检查返回值判断环境是否创建成功。

```
// Create COPT environment
errcode = COPT_CreateEnv(&env);
if (errcode)
  goto COPT_EXIT;
```

若返回值非 0，则跳转至错误处理步骤并反馈相应的错误代号和错误信息，然后退出程序。

创建问题

创建求解环境成功后，用户需要创建问题，问题中将包括待求解的变量、约束等信息，通过检查返回值判断问题是否创建成功。

```
// Create COPT problem
errcode = COPT_CreateProb(env, &prob);
if (errcode)
  goto COPT_EXIT;
```

若返回值非 0，则跳转至错误处理步骤并反馈相应的错误代号和错误信息，然后退出程序。

添加变量

对于线性模型，在创建变量时允许同时指定变量在目标函数中的系数、变量上下界等信息。对于本章节的示例问题，通过下述代码创建待求解变量：

```
/*
 * Add variables
 *
 *   obj: 1.2 C0 + 1.8 C1 + 2.1 C2
 *
 *   var:
 *     0.1 <= C0 <= 0.6
 *     0.2 <= C1 <= 1.5
 *     0.3 <= C2 <= 2.8
 *
 */
int ncol = 3;
double colcost[] = {1.2, 1.8, 2.1};
double collb[] = {0.1, 0.2, 0.3};
double colub[] = {0.6, 1.5, 1.8};

errcode = COPT_AddCols(prob, ncol, colcost, NULL, NULL, NULL, NULL, NULL, collb, colub,␣
↪NULL);
if (errcode)
  goto COPT_EXIT;
```

参数 ncol 指定了待创建变量的数目为 3，参数 colcost、collb 和 colub 分别指定了待添加变量在目标函数中的系数、上下界信息。对于函数 COPT_AddCols 的其它参数如：变量类型、变量名称均传入 NULL 表示默认为连续变量且变量名称由求解器自动生成。对于约束矩阵系数相关参数，传入 NULL 表示留待后续其它函数调用进行添加。

类似地，若返回值非 0，则跳转至错误处理步骤并反馈相应的错误代码和错误信息，然后退出程序。

添加约束

添加变量成功后，进一步添加作用于变量的约束条件。对于本章节的示例问题，其代码实现如下：

```
/*
 * Add constraints
 *
 *   r0: 1.5 C0 + 1.2 C1 + 1.8 C2 <= 2.6
 *   r1: 0.8 C0 + 0.6 C1 + 0.9 C2 >= 1.2
 */
int nrow = 2;
int rowbeg[] = {0, 3};
int rowcnt[] = {3, 3};
int rowind[] = {0, 1, 2, 0, 1, 2};
double rowelem[] = {1.5, 1.2, 1.8, 0.8, 0.6, 0.9};
```

(续下页)

(接上页)

```
  char rowsen[] = {COPT_LESS_EQUAL, COPT_GREATER_EQUAL};
  double rowrhs[] = {2.6, 1.2};

  errcode = COPT_AddRows(prob, nrow, rowbeg, rowcnt, rowind, rowelem, rowsen, rowrhs, NULL,␣
↪NULL);
  if (errcode)
    goto COPT_EXIT;
```

参数 nrow 指定了待创建的约束数目为 2，参数 rowbeg、rowcnt、rowind 和 rowelem 分别表示以 CSR 格式表示的约束系数对应的稀疏矩阵。参数 rowsen 表示约束的方向，参数 rowrhs 表示约束的右端项。对于函数 COPT_AddRows 中的其它参数均传入 NULL。

同样，若返回值为 0，则跳转至错误处理步骤并进行后续处理。

设置参数和属性

用户可以在求解问题前设置求解参数和问题相关属性，如设置求解时间限制为 10 秒，设置问题求解方向为最大化，代码如下：

```
  // Set parameters and attributes
  errcode = COPT_SetDblParam(prob, COPT_DBLPARAM_TIMELIMIT, 10);
  if (errcode)
    goto COPT_EXIT;
  errcode = COPT_SetObjSense(prob, COPT_MAXIMIZE);
  if (errcode)
    goto COPT_EXIT;
```

若设置失败，则返回非 0 值，并跳转至错误处理步骤进行后续处理。

求解问题

当前文所述的步骤都完成后，调用下述函数求解问题：

```
  // Solve problem
  errcode = COPT_SolveLp(prob);
  if (errcode)
    goto COPT_EXIT;
```

若返回值非 0，则跳转至错误处理步骤进行后续处理。

分析结果

求解完成后，首先获取问题求解状态，若状态为找到了最优解，则进一步获取目标函数值、各变量的取值及其基状态信息，代码实现如下：

```
// Analyze solution
int lpstat = COPT_LPSTATUS_UNSTARTED;
double lpobjval;
double* lpsol = NULL;
int* colstat = NULL;

errcode = COPT_GetIntAttr(prob, COPT_INTATTR_LPSTATUS, &lpstat);
if (errcode)
  goto COPT_EXIT;

if (lpstat == COPT_LPSTATUS_OPTIMAL)
{
  lpsol = (double*)malloc(ncol * sizeof(double));
  colstat = (int*)malloc(ncol * sizeof(int));

  errcode = COPT_GetLpSolution(prob, lpsol, NULL, NULL, NULL);
  if (errcode)
    goto COPT_EXIT;

  errcode = COPT_GetBasis(prob, colstat, NULL);
  if (errcode)
    goto COPT_EXIT;

  errcode = COPT_GetDblAttr(prob, COPT_DBLATTR_LPOBJVAL, &lpobjval);
  if (errcode)
    goto COPT_EXIT;

  printf("\nObjective value: %.6f\n", lpobjval);

  printf("Variable solution: \n");
  for (int i = 0; i < ncol; ++i)
    printf("  x[%d] = %.6f\n", i, lpsol[i]);

  printf("Variable basis status: \n");
  for (int i = 0; i < ncol; ++i)
    printf("  x[%d]: %d\n", i, colstat[i]);

  free(lpsol);
  free(colstat);
}
```

问题与结果文件输出

用户可以将当前求解的问题保存为标准的 MPS 模型文件，以及输出变量结果文件、基状态信息文件和修改过的参数文件，代码实现如下：

```
  // Write problem, solution and modified parameters to files
  errcode = COPT_WriteMps(prob, "lp_ex1.mps");
  if (errcode)
    goto COPT_EXIT;
  errcode = COPT_WriteBasis(prob, "lp_ex1.bas");
  if (errcode)
    goto COPT_EXIT;
  errcode = COPT_WriteSol(prob, "lp_ex1.sol");
  if (errcode)
    goto COPT_EXIT;
  errcode = COPT_WriteParam(prob, "lp_ex1.par");
  if (errcode)
    goto COPT_EXIT;
```

错误处理

若在调用 C 接口的过程中返回值非 0，则跳转至该步骤，输出错误代号与错误信息：

```
    goto COPT_EXIT;
  errcode = COPT_WriteParam(prob, "lp_ex1.par");
  if (errcode)
    goto COPT_EXIT;

  // Error handling
COPT_EXIT:
  if (errcode)
  {
    char errmsg[COPT_BUFFSIZE];
```

删除环境与问题

求解完成后，分别删除问题与环境：

```
  // Delete problem and environment
  COPT_DeleteProb(&prob);

  COPT_DeleteEnv(&env);
```

7.1.2 编译与运行

为了便于用户在不同的操作系统下编译与运行示例程序，我们在 Windows、Linux 和 MacOS 平台分别提供了 Visual Studio 工程和 Makefile 文件，具体说明如下。

Windows

对于 Windows 平台的用户，我们提供了 Visual Studio 工程文件，请用户确保计算机中已下载与安装 Visual Studio 2017。假定杉数求解器的安装路径为：`'<instdir>'` ，对于下载可执行安装程序进行安装的用户，则可以直接打开路径 `'<instdir>\examples\c\vsprojects'` 下的 Visual Studio 工程文件 `lp_ex1.vcxproj` 生成解决方案即可。对于下载 ZIP 包安装的用户，请确保已正确配置相关环境变量，详情见*杉数求解器安装说明* 。

Linux 和 MacOS

对于 Linux 和 Mac 平台的用户，我们提供了 Makefile 文件，请用户确保计算机中已分别安装 GCC 和 Clang 编译器工具链，以及 make 工具，并正确配置杉数求解器的必要的环境变量，详情见*杉数求解器安装说明* 。假定杉数求解器的安装路径为：`<instdir>` ，则请用户切换到路径 `'<instdir>\examples\c'` 下，并在终端输入命令 `make` 即可编译成功。

7.2 C++ 接口

本章通过一个简单的示例演示如何使用杉数求解器的 C++ 接口。简单来说，本例演示如何创建环境和建立模型，添加变量和约束，然后求解的过程。最后求解后如何获得最优解的步骤也包括在内。

待求解的线性问题数学公式如下所示：

$$
\begin{aligned}
&\text{最大化：}\\
&\qquad 1.2x + 1.8y + 2.1z\\
&\text{约束：}\\
&\qquad 1.5x + 1.2y + 1.8z \leq 2.6\\
&\qquad 0.8x + 0.6y + 0.9z \geq 1.2\\
&\text{变量范围：}\\
&\qquad 0.1 \leq x \leq 0.6\\
&\qquad 0.2 \leq y \leq 1.5\\
&\qquad 0.3 \leq z \leq 2.8
\end{aligned}
\tag{7.2}
$$

注意到上述问题其实和*C 接口* 里通过建模和求解来演示的问题是同一个。

7.2.1 示例解析

使用杉数求解器的 C++ 接口求解与分析上述问题的代码如下。

代码 7.2: lp_ex1.cpp

```cpp
/*
 * This file is part of the Cardinal Optimizer, all rights reserved.
 */
#include "coptcpp_pch.h"

using namespace std;

/*
 * This example solves the following LP model:
 *
 *  Maximize:
 *    1.2 x + 1.8 y + 2.1 z
 *
 *  Subject to:
 *    R0: 1.5 x + 1.2 y + 1.8 z <= 2.6
 *    R1: 0.8 x + 0.6 y + 0.9 z >= 1.2
 *
 *  Where:
 *    0.1 <= x <= 0.6
 *    0.2 <= y <= 1.5
 *    0.3 <= z <= 2.8
 */
int main(int argc, char* argv[])
{
  try
  {
    Envr env;
    Model model = env.CreateModel("lp_ex1");

    // Add variables
    Var x = model.AddVar(0.1, 0.6, 0.0, COPT_CONTINUOUS, "x");
    Var y = model.AddVar(0.2, 1.5, 0.0, COPT_CONTINUOUS, "y");
    Var z = model.AddVar(0.3, 2.8, 0.0, COPT_CONTINUOUS, "z");

    // Set objective
    model.SetObjective(1.2 * x + 1.8 * y + 2.1 * z, COPT_MAXIMIZE);

    // Add linear constraints using linear expression
    model.AddConstr(1.5 * x + 1.2 * y + 1.8 * z <= 2.6, "R0");

    Expr expr(x, 0.8);
    expr.AddTerm(y, 0.6);
```

(续下页)

(接上页)

```
    expr += 0.9 * z;
    model.AddConstr(expr >= 1.2, "R1");

    // Set parameters
    model.SetDblParam(COPT_DBLPARAM_TIMELIMIT, 10);

    // Solve problem
    model.Solve();

    // Output solution
    if (model.GetIntAttr(COPT_INTATTR_HASLPSOL) != 0)
    {
      cout << "\nFound optimal solution:" << endl;
      VarArray vars = model.GetVars();
      for (int i = 0; i < vars.Size(); i++)
      {
        Var var = vars.GetVar(i);
        cout << "  " << var.GetName() << " = " << var.Get(COPT_DBLINFO_VALUE) << endl;
      }
      cout << "Obj = " << model.GetDblAttr(COPT_DBLATTR_LPOBJVAL) << endl;
    }
  }
  catch (CoptException e)
  {
    cout << "Error Code = " << e.GetCode() << endl;
    cout << e.what() << endl;
  }
  catch (...)
  {
    cout << "Unknown exception occurs!";
  }
}
```

接下来我们将基于上述代码分步骤讲解求解与分析过程。

导入 COPT 的 C++ 头文件

使用杉数求解器的 C++ 接口，需要首先包括 COPT 的 C++ 头文件 `coptcpp_pch.h` 。

```
#include "coptcpp_pch.h"
```

创建求解环境和建模

对于任意使用杉数求解器 C++ 接口的应用程序，求解步骤的第一步是创建求解环境。从求解环境出发，用户可以创建一个或者多个模型。注意到每个模型都对应了一个实际的求解问题以及相关数据，比如参数设置等。

如果用户需要求解多个问题，用户即可以在同一个模型里对多个问题逐个依次求解，也可以在同一求解环境里创建多个模型来求解。

```
Envr env;
Model model = env.CreateModel("lp_ex1");
```

上述代码对 C++ 求解环境实例化，并创建了一个名为“lp_ex1”的模型对象。

添加变量

接下来，我们演示在模型里添加多个变量。杉数求解器提供包括 AddVar() 和 AddVars() 在内的多种途径来添加变量。注意到变量不是独立存在，总是和某个模型关联的。

```
// Add variables
Var x = model.AddVar(0.1, 0.6, 0.0, COPT_CONTINUOUS, "x");
Var y = model.AddVar(0.2, 1.5, 0.0, COPT_CONTINUOUS, "y");
Var z = model.AddVar(0.3, 2.8, 0.0, COPT_CONTINUOUS, "z");
```

上述代码中 AddVar() 的第一和第二个参数分别是指变量的下界和上界；第三个参数是指目标函数中的系数（这里暂设为 0，我们会在 SetObjective() 真正设值）；第四个参数是指变量的类型。例子里所有变量的类型都是连续形的。最后一个参数是指变量名。

为了适应不同的调用需求，杉数求解器实现了 AddVar() 的不同形参的几种重载。细节请参看*C++ API 参考手册* 。

目标函数的创建是调用了 SetObjective() ，并使用了运算符重载。因为 C++ 允许重载算术运算符，这样建立线性表达式的方式比较直观。这里第二个参数是设置优化目标为最大化。

添加约束

接下来，我们演示在模型里添加多个约束。和变量一样，约束也总是和某个模型关联的。杉数求解器提供包括 AddConstr() 和 AddConstrs() 在内的多种途径来添加约束。

```
// Add linear constraints using linear expression
model.AddConstr(1.5 * x + 1.2 * y + 1.8 * z <= 2.6, "R0");

Expr expr(x, 0.8);
expr.AddTerm(y, 0.6);
expr += 0.9 * z;
model.AddConstr(expr >= 1.2, "R1");
```

第一个约束的创建比较直观，因为其线性表达式是通过重载算术运算符和比较运算符的方式生成的。

第二个约束的创建则是通过先建立相关线性表达式的对象，然后再通过 AddTerm() 添加变量和对应系数的方式来逐步建立最终的线性表达式。最后约束的生成同样使用了重载的比较运算符。

设置参数和属性

在求解问题之前，还可以通过 SetDblParam() 等方法设置模型的参数和属性。

```
// Set parameters
model.SetDblParam(COPT_DBLPARAM_TIMELIMIT, 10);
```

这里，调用 SetDblParam() 来设置 COPT_DBLPARAM_TIMELIMIT 参数值，使得求解器在最多执行 10 秒后超时退出。

求解问题

到此为止，我们已经建好了模型。接下来，可以求解问题。

```
// Solve problem
model.Solve();
```

这一步之后，问题已经被求解，并在内部保存了结果，包括求解状态，最优解等属性值。

输出结果

在问题被求解之后，可以通过查询不同的属性值来实现不同的目的。

```
// Output solution
if (model.GetIntAttr(COPT_INTATTR_HASLPSOL) != 0)
{
  cout << "\nFound optimal solution:" << endl;
  VarArray vars = model.GetVars();
  for (int i = 0; i < vars.Size(); i++)
  {
    Var var = vars.GetVar(i);
    cout << "  " << var.GetName() << " = " << var.Get(COPT_DBLINFO_VALUE) << endl;
  }
  cout << "Obj = " << model.GetDblAttr(COPT_DBLATTR_LPOBJVAL) << endl;
}
```

上述代码中，我们首先查询模型的属性值 COPT_INTATTR_HASLPSOL 来知道是否生成了 LP 最优解；再查询变量的属性值 COPT_DBLINFO_VALUE 来获得这个变量的值；然后查询模型的属性值 COPT_DBLATTR_LPOBJVAL 来获得目标函数的最优值。

关于模型、变量、约束的属性名和类型，请参看 C API 参考手册里的章节属性 。

错误处理

杉数求解器 C++ 接口的错误处理使用 C++ 自身的异常处理机制。例子中，整个求解过程都嵌入在 try 块里，中间的任意求解错误都会被 catch 块捕获并显示。

```
catch (CoptException e)
{
  cout << "Error Code = " << e.GetCode() << endl;
  cout << e.what() << endl;
}
catch (...)
{
  cout << "Unknown exception occurs!";
}
```

7.2.2 编译与运行

本章节演示的例子，我们已经放入在杉数求解器安装包里，方便用户尝试编译并运行。具体请参看安装路径下的文件夹 `$COPT_HOME/examples/cpp` 。这个文件夹包括了 Windows 平台下的 Visual Studio 项目文件，或者 Linux/Mac 平台下的 Makefile 文件。

Windows Visual Studio 项目

Windows 平台下的 Visual Studio 项目文件位于 `$COPT_HOME/examples/cpp/vsprojects` 。如果平台已经安装了 Visual Studio 2017 或者更高版本，双击这个项目文件会启动 Visual Studio 来编译和运行。

具体来说，编译例子依赖于杉数求解器 C++ 动态库 `copt_cpp.dll` ，以及一些相关的头文件 `copt.h`, `coptcpp.h` and `coptcpp.idl.h` 。这些头文件声明了从 C++ 动态库 `copt_cpp.dll` 导出的函数、接口、参数和属性的常量。进一步，为了方便使用，杉数求解器 C++ 接口还提供了每个导出类的重包装头文件并重载了相关运算符，位于 `$COPT_HOME/include/coptcpp_inc` 。

简单来说，用户只需要像例子中所示一样包括头文件 `coptcpp_pch.h`、配置附加依赖项为 `copt_cpp.lib` 、设置附加链接库的目录为 `$(COPT_HOME)/lib` ，并确保动态库 `copt_cpp.dll` 已经安装在合适的路径下，可以在运行中被加载即可。

特别需要注意的是 Windows 系统下的杉数求解器 C++ 动态库和 GCC 编译器并不兼容。也就是说，如果你用 GCC 编译这个例子，生成的可执行文件可能不能正常工作。这是因为 GCC 和 Windows SDK 不兼容。除此之外，Windows 系统下 Clang 和 Intel 编译器，和上述的 MSVC 编译器一样，都可以用来编译这个例子。

Makefile 项目

在 Linux 和 Mac 平台，我们同样提供了 Makefile 文件方便用户编译和运行 C++ 例子。假设用户已经在所在平台安装了开发工具 `gcc` 和 `make` 。编译 C++ 例子最简单的方式是先在终端下进入安装路径下的 C++ 例子文件夹 `$COPT_HOME/examples/cpp` ，然后执行命令 `'make'` 。

类似地，编译例子依赖于杉数求解器的 C++ 动态库，分别是 Linux 平台下的 `libcopt_cpp.so` ，Mac 平台下的 `libcopt_cpp.dylib` ，还需要包括头文件 `$COPT_HOME/include/coptcpp_inc/coptcpp_pch.h` 。

最后，用户只要正确安装了杉数求解器安装包（设置好了动态库路径），并配置了有效的授权文档，就可以在各大平台下正确执行这个 C++ 例子。安装和授权细节请参考*安装说明* 。

7.3 C# 接口

本章通过一个简单的示例演示如何使用杉数求解器的 C# 接口。简单来说，本例演示如何创建环境和建立模型，添加变量和约束，然后求解的过程。最后求解后如何获得最优解的步骤也包括在内。

待求解的线性问题数学公式如下所示：

$$
\begin{aligned}
&\text{最大化：} \\
&\qquad 1.2x + 1.8y + 2.1z \\
&\text{约束：} \\
&\qquad 1.5x + 1.2y + 1.8z \leq 2.6 \\
&\qquad 0.8x + 0.6y + 0.9z \geq 1.2 \\
&\text{变量范围：} \\
&\qquad 0.1 \leq x \leq 0.6 \\
&\qquad 0.2 \leq y \leq 1.5 \\
&\qquad 0.3 \leq z \leq 2.8
\end{aligned}
\tag{7.3}
$$

注意到上述问题其实和*C 接口* 里通过建模和求解来演示的问题是同一个。

7.3.1 示例解析

使用杉数求解器的 C# 接口求解与分析上述问题的代码如下。

代码 7.3: `lp_ex1.cs`

```
/*
 * This file is part of the Cardinal Optimizer, all rights reserved.
 */
using Copt;
using System;

/*
 * This C# example solves the following LP model:
```

(续下页)

(接上页)

```
 *
 *
 * Maximize:
 *  1.2 x + 1.8 y + 2.1 z
 *
 * Subject to:
 *  1.5 x + 1.2 y + 1.8 z <= 2.6
 *  0.8 x + 0.6 y + 0.9 z >= 1.2

 * where:
 *  0.1 <= x <= 0.6
 *  0.2 <= y <= 1.5
 *  0.3 <= z <= 2.8
 */
public class lp_ex1
{
  public static void Main()
  {
    try
    {
      Envr env = new Envr();
      Model model = env.CreateModel("lp_ex1");

      /*
       * Add variables x, y, z
       *
       * obj: 1.2 x + 1.8 y + 2.1 z
       *
       * var:
       *  0.1 <= x <= 0.6
       *  0.2 <= y <= 1.5
       *  0.3 <= z <= 2.8
       */
      Var x = model.AddVar(0.1, 0.6, 0.0, Copt.Consts.CONTINUOUS, "x");
      Var y = model.AddVar(0.2, 1.5, 0.0, Copt.Consts.CONTINUOUS, "y");
      Var z = model.AddVar(0.3, 2.8, 0.0, Copt.Consts.CONTINUOUS, "z");

      model.SetObjective(1.2 * x + 1.8 * y + 2.1 * z, Copt.Consts.MAXIMIZE);

      /*
       * Add two constraints using linear expression
       *
       * r0: 1.5 x + 1.2 y + 1.8 z <= 2.6
       * r1: 0.8 x + 0.6 y + 0.9 z >= 1.2
       */
      model.AddConstr(1.5 * x + 1.2 * y + 1.8 * z <= 2.6, "r0");

```

(续下页)

(接上页)

```
      Expr expr = new Expr(x, 0.8);
      expr.AddTerm(y, 0.6);
      expr += 0.9 * z;
      model.AddConstr(expr >= 1.2, "r1");

      // Set parameters
      model.SetDblParam(Copt.DblParam.TimeLimit, 10);

      // Solve problem
      model.Solve();

      // Output solution
      if (model.GetIntAttr(Copt.IntAttr.LpStatus) == Copt.Status.OPTIMAL)
      {
        Console.WriteLine("\nFound optimal solution:");
        VarArray vars = model.GetVars();
        for (int i = 0; i < vars.Size(); i++)
        {
          Var x = vars.GetVar(i);
          Console.WriteLine("  {0} = {1}", x.GetName(), x.Get(Copt.DblInfo.Value));
        }
        Console.WriteLine("Obj = {0}", model.GetDblAttr(Copt.DblAttr.LpObjVal));
      }

      Console.WriteLine("\nDone");
    }
    catch (CoptException e)
    {
      Console.WriteLine("Error Code = {0}", e.GetCode());
      Console.WriteLine(e.Message);
    }
  }
}
```

接下来我们将基于上述代码分步骤讲解求解与分析过程。

引用 COPT 的 C# 命名空间

为代码简洁起见，首先引用 COPT 的 C# 命名空间 Copt 。

```
using Copt;
```

创建求解环境和建模

对于任意使用杉数求解器 C# 接口的应用程序，求解步骤的第一步是创建求解环境。从求解环境出发，用户可以创建一个或者多个模型。注意到每个模型都对应了一个实际的求解问题以及相关数据，比如参数设置等。

如果用户需要求解多个问题，用户即可以在同一个模型里对多个问题逐个依次求解，也可以在同一求解环境里创建多个模型来求解。

```
Envr env = new Envr();
Model model = env.CreateModel("lp_ex1");
```

上述代码对 C# 求解环境实例化，并创建了一个名为“lp_ex1”的模型对象。

添加变量

接下来，我们演示在模型里添加多个变量。杉数求解器提供包括 AddVar() 和 AddVars() 在内的多种途径来添加变量。注意到变量不是独立存在，总是和某个模型关联的。

```
/*
 * Add variables x, y, z
 *
 * obj: 1.2 x + 1.8 y + 2.1 z
 *
 * var:
 *  0.1 <= x <= 0.6
 *  0.2 <= y <= 1.5
 *  0.3 <= z <= 2.8
 */
Var x = model.AddVar(0.1, 0.6, 0.0, Copt.Consts.CONTINUOUS, "x");
Var y = model.AddVar(0.2, 1.5, 0.0, Copt.Consts.CONTINUOUS, "y");
Var z = model.AddVar(0.3, 2.8, 0.0, Copt.Consts.CONTINUOUS, "z");

model.SetObjective(1.2 * x + 1.8 * y + 2.1 * z, Copt.Consts.MAXIMIZE);
```

上述代码中 AddVar() 的第一和第二个参数分别是指变量的下界和上界；第三个参数是指目标函数中的系数（这里暂设为 0，我们会在 SetObjective() 真正设值）；第四个参数是指变量的类型。例子里所有变量的类型都是连续形的。最后一个参数是指变量名。

为了适应不同的调用需求，杉数求解器实现了 AddVar() 的不同形参的几种重载。请参看 C# API 参考手册里的章节*C# 优化建模类* 。

目标函数的创建是调用了 SetObjective() ，并使用了运算符重载。因为 C# 允许重载算术运算符，这样建立线性表达式的方式比较直观。这里第二个参数是设置优化目标为最大化。

添加约束

接下来，我们演示在模型里添加多个约束。和变量一样，约束也总是和某个模型关联的。杉数求解器提供包括 AddConstr() 和 AddConstrs() 在内的多种途径来添加约束。

```
/*
 * Add two constraints using linear expression
 *
 * r0: 1.5 x + 1.2 y + 1.8 z <= 2.6
 * r1: 0.8 x + 0.6 y + 0.9 z >= 1.2
 */
model.AddConstr(1.5 * x + 1.2 * y + 1.8 * z <= 2.6, "r0");

Expr expr = new Expr(x, 0.8);
expr.AddTerm(y, 0.6);
expr += 0.9 * z;
model.AddConstr(expr >= 1.2, "r1");
```

第一个约束的创建比较直观，因为其线性表达式是通过重载算术运算符和比较运算符的方式生成的。

第二个约束的创建则是通过先建立相关线性表达式的对象，然后再通过 AddTerm() 添加变量和对应系数的方式来逐步建立最终的线性表达式。最后约束的生成同样使用了重载的比较运算符。

设置参数和属性

在求解问题之前，还可以通过 SetDblParam() 等方法设置模型的参数和属性。

```
// Set parameters
model.SetDblParam(Copt.DblParam.TimeLimit, 10);
```

这里，调用 SetDblParam() 来设置 Copt.DblParam.TimeLimit 参数值，使得求解器在最多执行 10 秒后超时退出。

求解问题

到此为止，我们已经建好了模型。接下来，可以求解问题。

```
// Solve problem
model.Solve();
```

这一步之后，问题已经被求解，并在内部保存了结果，包括求解状态，最优解等属性值。

输出结果

在问题被求解之后，可以通过查询不同的属性值来实现不同的目的。

```
    // Output solution
    if (model.GetIntAttr(Copt.IntAttr.LpStatus) == Copt.Status.OPTIMAL)
    {
      Console.WriteLine("\nFound optimal solution:");
      VarArray vars = model.GetVars();
      for (int i = 0; i < vars.Size(); i++)
      {
        Var x = vars.GetVar(i);
        Console.WriteLine("  {0} = {1}", x.GetName(), x.Get(Copt.DblInfo.Value));
      }
      Console.WriteLine("Obj = {0}", model.GetDblAttr(Copt.DblAttr.LpObjVal));
    }

    Console.WriteLine("\nDone");
```

上述代码中，我们首先查询模型的属性值 Copt.IntAttr.LpStatus 以判断求解是否达到最优；再查询变量的属性值 Copt.DblInfo.Value 来获得这个变量的值；然后查询模型的属性值 Copt.DblAttr.LpObjVal 来获得目标函数的最优值。

关于模型、变量、约束的属性名和类型，请参看 C# API 参考手册里的章节*C# 常量类* 。

错误处理

杉数求解器 C# 接口的错误处理使用 C# 自身的异常处理机制。例子中，整个求解过程都嵌入在 try 块里，中间的任意求解错误都会被 catch 块捕获并显示。

```
  catch (CoptException e)
  {
    Console.WriteLine("Error Code = {0}", e.GetCode());
    Console.WriteLine(e.Message);
  }
```

7.3.2 编译与运行

本章节演示的例子，我们已经放入在 COPT 安装包里，方便用户尝试编译并运行。具体请参看安装路径下的文件夹 `$COPT_HOME/examples/csharp` 。这个文件夹包括了例子的 csharp 代码，以及 dotnet core 2.0 的项目文件。

这个例子可以在 Windows，Linux 和 Mac 平台下直接编译并运行。首先需要在使用平台上下载并安装 dotnet core 2.0 。感兴趣的用户可以查看 使用教程 了解更多信息。

Dotnet core 2.0 项目

基于 dotnet core 2.0 的 C# 项目文件 `example.csproj` 位于 `$COPT_HOME/examples/csharp/dotnetprojects` 。将示例文件 `lp_ex1.cs` 复制到上述目录，并在终端窗口下进入路径 `$COPT_HOME/examples/csharp/dotnetprojects` ，然后执行命令 `'dotnet run --framework netcoreapp2.0'` 。对于使用 dotnet core 3.0 的用户，用户只需要在上述目录执行命令 `'dotnet run --framework netcoreapp3.0'` 即可运行。

具体来说，编译例子依赖于杉数求解器的基于 dotnet core 2.0 的动态库 `copt_dotnet20.dll` 。它不仅定义了杉数求解器的所有 C# 类，还间接加载了两个相关的杉数求解器动态库，分别是 Windows 下的 `coptcswrap.dll` 和 `copt_cpp.dll` ，Linux 下的 `libcoptcswrap.so` 和 `libcopt_cpp.so` ，Mac 下的 `libcoptcswrap.dylib` 和 `libcopt_cpp.dylib` 。动态库 `coptcswrap` 用来衔接杉数求解器的 dotnet 2.0 动态库和杉数求解器的算法核心库 `copt_cpp`。核心库真正定义了杉数求解器的所有类，接口以及属性和参数常量。用户需要确保上述动态库已经安装在合适的路径下，可以在运行中被加载。

总之，用户只要正确安装了杉数求解器安装包（设置好了动态库路径），并配置了有效的授权文档，就可以在各大平台下正常执行这个 C# 例子。安装和授权细节请参考*安装说明* 。

7.4 Java 接口

本章通过一个简单的示例演示如何使用杉数求解器的 Java 接口。简单来说，本例演示如何创建环境和建立模型，添加变量和约束，然后求解的过程。最后求解后分析的步骤也包括在内。

待求解的线性问题数学公式如下所示：

$$
\begin{aligned}
&\text{最大化：}\\
&\qquad 1.2x + 1.8y + 2.1z\\
&\text{约束：}\\
&\qquad 1.5x + 1.2y + 1.8z \le 2.6\\
&\qquad 0.8x + 0.6y + 0.9z \ge 1.2\\
&\text{变量范围：}\\
&\qquad 0.1 \le x \le 0.6\\
&\qquad 0.2 \le y \le 1.5\\
&\qquad 0.3 \le z \le 2.8
\end{aligned}
\tag{7.4}
$$

注意到上述问题其实和*C 接口* 里通过建模和求解来演示的问题是同一个。

7.4.1 示例解析

使用杉数求解器的 Java 接口求解与分析上述问题的代码如下。

代码 7.4: `Lp_ex1.java`

```
1 /*
2  * This file is part of the Cardinal Optimizer, all rights reserved.
```

(续下页)

(接上页)

```
 */
import copt.*;

/*
 * This Java example solves the following LP model:
 *
 * Maximize:
 *  1.2 x + 1.8 y + 2.1 z
 *
 * Subject to:
 *  1.5 x + 1.2 y + 1.8 z <= 2.6
 *  0.8 x + 0.6 y + 0.9 z >= 1.2
 *
 * where:
 *  0.1 <= x <= 0.6
 *  0.2 <= y <= 1.5
 *  0.3 <= z <= 2.8
 */
public class Lp_ex1 {
  public static void main(final String argv[]) {
    try {
      Envr env = new Envr();
      Model model = env.createModel("lp_ex1");

      /*
       * Add variables x, y, z
       *
       * obj: 1.2 x + 1.8 y + 2.1 z
       *
       * var:
       *  0.1 <= x <= 0.6
       *  0.2 <= y <= 1.5
       *  0.3 <= z <= 2.8
       */
      Var x = model.addVar(0.1, 0.6, 1.2, copt.Consts.CONTINUOUS, "x");
      Var y = model.addVar(0.2, 1.5, 1.8, copt.Consts.CONTINUOUS, "y");
      Var z = model.addVar(0.3, 2.8, 2.1, copt.Consts.CONTINUOUS, "z");

      /*
       * Add two constraints using linear expression
       *
       * r0: 1.5 x + 1.2 y + 1.8 z <= 2.6
       * r1: 0.8 x + 0.6 y + 0.9 z >= 1.2
       */
      Expr e0 = new Expr(x, 1.5);
      e0.addTerm(y, 1.2);
      e0.addTerm(z, 1.8);
```

(续下页)

(接上页)

```
      model.addConstr(e0, copt.Consts.LESS_EQUAL, 2.6, "r0");

      Expr e1 = new Expr(x, 0.8);
      e1.addTerm(y, 0.6);
      e1.addTerm(z, 0.9);
      model.addConstr(e1, copt.Consts.GREATER_EQUAL, 1.2, "r1");

      // Set parameters and attributes
      model.setDblParam(copt.DblParam.TimeLimit, 10);
      model.setObjSense(copt.Consts.MAXIMIZE);

      // Solve problem
      model.solve();

      // Output solution
      if (model.getIntAttr(copt.IntAttr.HasLpSol) != 0) {
        System.out.println("\nFound optimal solution:");
        VarArray vars = model.getVars();
        for (int i = 0; i < vars.size(); i++) {
          Var x = vars.getVar(i);
          System.out.println("  " + x.getName() + " = " + x.get(copt.DblInfo.Value));
        }
        System.out.println("Obj = " + model.getDblAttr(copt.DblAttr.LpObjVal));
      }

      System.out.println("\nDone");
    } catch (CoptException e) {
      System.out.println("Error Code = " + e.getCode());
      System.out.println(e.getMessage());
    }
  }
}
```

接下来我们将基于上述代码分步骤讲解求解与分析过程。

导入 COPT 的 Java 类

使用杉数求解器的 Java 接口，需要首先导入 COPT 的 Java 类

```
import copt.*;
```

创建求解环境和建模

对于任意使用杉数求解器 Java 接口的应用程序，求解步骤的第一步是创建求解环境。从求解环境出发，用户可以创建一个或者多个模型。注意到每个模型都对应了一个实际的求解问题以及相关数据，比如参数设置等。

如果用户需要求解多个问题，用户即可以在同一个模型里对多个问题逐个依次求解，也可以在同一求解环境里创建多个模型来求解。

```
Envr env = new Envr();
Model model = env.createModel("lp_ex1");
```

上述代码对 Java 求解环境实例化，并创建了一个名为“lp_ex1”的模型对象。

添加变量

接下来，我们演示在模型里添加多个变量。杉数求解器提供包括 addVar() 和 addVars() 在内的多种途径来添加变量。注意到变量不是独立存在，总是和某个模型关联的。

```
/*
 * Add variables x, y, z
 *
 * obj: 1.2 x + 1.8 y + 2.1 z
 *
 * var:
 *  0.1 <= x <= 0.6
 *  0.2 <= y <= 1.5
 *  0.3 <= z <= 2.8
 */
Var x = model.addVar(0.1, 0.6, 1.2, copt.Consts.CONTINUOUS, "x");
Var y = model.addVar(0.2, 1.5, 1.8, copt.Consts.CONTINUOUS, "y");
Var z = model.addVar(0.3, 2.8, 2.1, copt.Consts.CONTINUOUS, "z");
```

上述代码中 addVar() 的第一和第二个参数分别是指变量的下界和上界；第三个参数是指目标函数中的系数；第四个参数是指变量的类型。例子里所有变量的类型都是连续形的。最后一个参数是指变量名。

为了适应不同的调用需求，杉数求解器实现了 addVar() 的不同形参的几种重载。请参看 Java API 参考手册里的章节*Java 建模类* 。

添加约束

接下来，我们演示在模型里添加多个约束。和变量一样，约束也总是和某个模型关联的。杉数求解器提供包括 addConstr() 和 addConstrs() 在内的多种途径来添加约束。

```
/*
 * Add two constraints using linear expression
 *
 * r0: 1.5 x + 1.2 y + 1.8 z <= 2.6
```

(续下页)

(接上页)

```
     * r1: 0.8 x + 0.6 y + 0.9 z >= 1.2
     */
    Expr e0 = new Expr(x, 1.5);
    e0.addTerm(y, 1.2);
    e0.addTerm(z, 1.8);
    model.addConstr(e0, copt.Consts.LESS_EQUAL, 2.6, "r0");

    Expr e1 = new Expr(x, 0.8);
    e1.addTerm(y, 0.6);
    e1.addTerm(z, 0.9);
    model.addConstr(e1, copt.Consts.GREATER_EQUAL, 1.2, "r1");
```

上述代码的约束都是通过先建立线性表达式的对象，然后再通过 addTerm() 添加变量和对应系数的方式来逐步建立最终的线性表达式。

设置参数和属性

在求解问题之前，还可以通过 setDblParam() 等方法设置模型的参数和属性。

```
    // Set parameters and attributes
    model.setDblParam(copt.DblParam.TimeLimit, 10);
    model.setObjSense(copt.Consts.MAXIMIZE);
```

这里，调用 setDblParam() 来设置 copt.DblParam.TimeLimit 参数值，使得求解器在最多执行 10 秒后超时退出；调用 setObjSense() 并使用 copt.Consts.MAXIMIZE 参数，设置求解目标为最大化。

求解问题

到此为止，我们已经建好了模型。接下来，可以求解问题。

```
    // Solve problem
    model.solve();
```

这一步之后，问题已经被求解，并在内部保存了结果，包括求解状态，最优解等属性值。

输出结果

在问题被求解之后，可以通过查询不同的属性值来实现不同的目的。

```
    // Output solution
    if (model.getIntAttr(copt.IntAttr.HasLpSol) != 0) {
      System.out.println("\nFound optimal solution:");
      VarArray vars = model.getVars();
      for (int i = 0; i < vars.size(); i++) {
        Var x = vars.getVar(i);
        System.out.println("  " + x.getName() + " = " + x.get(copt.DblInfo.Value));
```

(续下页)

(接上页)

```
        }
        System.out.println("Obj = " + model.getDblAttr(copt.DblAttr.LpObjVal));
      }

      System.out.println("\nDone");
```

上述代码中，我们首先查询模型的属性值 copt.IntAttr.HasLpSol 来知道是否生成了 LP 最优解；再查询变量的属性值 copt.DblInfo.Value 来获得这个变量的值；然后查询模型的属性值 copt.DblAttr.LpObjVal 来获得目标函数的最优值。

关于模型、变量、约束的属性名和类型，请参看 Java API 参考手册里的章节*Java 常量类* 。

错误处理

杉数求解器 Java 接口的错误处理使用 Java 自身的异常处理机制。例子中，整个求解过程都嵌入在 try 块里，中间的任意求解错误都会被 catch 块捕获并显示。

```
    } catch (CoptException e) {
      System.out.println("Error Code = " + e.getCode());
      System.out.println(e.getMessage());
    }
```

7.4.2 编译与运行

本章节演示的例子，我们已经放入在 COPT 安装包里，方便用户尝试编译并运行。具体请参看安装路径下的文件夹 `$COPT_HOME/examples/java` 。这个文件夹包括了例子的 Java 代码，以及一个脚本文件，可以用来直接编译并运行。

本例子可以在任意支持 Java 的平台上运行。首先需要在使用平台上安装 Java 8 或者更高版本。

Java 例子细节

运行 Java 例子最简单的方式是先在 console 或者 terminal 窗口下进入安装路径的 Java 例子文件夹 `$COPT_HOME/examples/java` ，然后执行命令 `'sh run.sh'` 。

具体来说，编译例子依赖于 COPT 的 Java 包 `copt_java.jar` 。它不仅定义了杉数求解器的所有 Java 类，还间接加载了两个相关的 COPT 动态库，分别是 Windows 下的 `coptjniwrap.dll` 和 `copt_cpp.dll` ，Linux 下的 `libcoptjniwrap.so` 和 `libcopt_cpp.so` ，Mac 下的 `libcoptjniwrap.dylib` 和 `libcopt_cpp.dylib` 。这里，动态库 `coptjniwrap` 是使用 swig 对 JNI 进行包装，用来衔接 COPT 的 Java 包和 COPT 的 C++ 核心库 `copt_cpp`。这个核心库真正定义了杉数求解器的所有类，接口以及属性和参数常量。所以用户需要确保上述动态库已经安装在合适的路径下，可以在运行中被加载。

一般来说，用户只要正确安装了 COPT 安装包（设置好了动态库路径），并配置了有效的授权文档，就可以正常执行这个 Java 例子。安装和授权细节请参考*安装说明* 。

7.5 Python 接口

7.5.1 安装说明

目前，杉数求解器的 Python 接口支持 Python 2.7, 3.6-3.11 版本。其中，对于 Python 3.8-3.11 版本，COPT 的 MacOS-Universal 可提供支持；对于 Python 2.7, 3.6-3.7 版本，则只有 MacOS-X86。

使用 Python 接口前，请用户确保已正确安装和配置杉数求解器，详情可参考如何安装杉数求解器 。用户可以从 Anaconda 发行版 或者 Python 官方发行版 下载并安装 Python。我们推荐用户安装 Anaconda 发行版，因为它对 Python 新手使用更加友好与方便（对于 Windows 系统，请勿使用通过 Microsoft Store 安装的 Python）。若使用官方发行版本或系统自带版本，则应确保已安装 `pip` 和 `setuptools` Python 工具包。

注意

我们推荐使用 3.8-3.11 版本，因为 COPT-Python 接口的矩阵建模功能最低版本要求是 3.8。2.7, 3.6-3.7 版本除非必要，不推荐首选。

Windows

方式一：通过 pip install 的方式（推荐）

打开 cmd 命令行窗口（若 Python 是 Anaconda 发行版，则打开 Anaconda 命令行窗口），输入如下命令：

```
pip install coptpy
```

若已安装了旧版本的 `coptpy` 包，请在 cmd 命令行窗口（若 Python 是 Anaconda 发行版，则打开 Anaconda 命令行窗口），输入如下命令，以升级到最新版本的 `coptpy` 包：

```
pip install --upgrade coptpy
```

方式二：通过 COPT 安装包

对于 Windows 系统，假定杉数求解器安装路径为：C:\Program Files\copt70，则切换目录到 C:\Program Files\copt70\lib\python ，并在命令行窗口上执行如下命令：

```
python setup.py install
```

注意：若杉数求解器安装在系统盘（一般为 C 盘），则需要 **以管理员权限执行**打开命令行窗口。为测试 Python 接口是否正确安装，用户可切换目录至 C:\Program Files\copt70\examples\python ，并在命令行窗口上执行如下命令：

```
python lp_ex1.py
```

若模型正确执行，则表示已正确安装杉数求解器的 Python 接口。

注意：若使用 Python 官方发布的 Python 3.8 版本，假定安装路径为："C:\Program Files\Python38"，则需要将杉数求解器安装路径的 "bin" 子目录下的 copt_cpp.dll 文件复制到杉数求解器的 Python 接口安装路径："C:\Program Files\Python38\Lib\site-packages\coptpy" 以解决动态库依赖问题。

目前，coptpy 已经支持 type hints，用户可以在命令行窗口（若 Python 为 Anaconda 发行版，请打开 Anaconda Prompt），执行如下命令：

```
pip install coptpy-stubs
```

安装成功后，在 Python IDE 中编写代码时，会提示变量名补全及函数参数可取值。

Linux

方式一：通过 pip install 的方式（推荐）

打开终端，输入如下命令，安装 COPT Python 接口

```
pip install coptpy
```

若已安装了旧版本的 coptpy 包，请打开终端，输入如下命令，以升级到最新版本的 coptpy ：

```
pip install --upgrade coptpy
```

方式二：通过 COPT 安装包

对于 Linux 系统，假定杉数求解器安装路径为：/opt/copt70 ，则切换目录到 /opt/copt70/lib/python ，并在终端上执行：

```
sudo python setup.py install
```

对于使用 Anaconda 发行版的用户，若执行上述命令失败，则需要使用 Anaconda 发行版 Python 可执行文件的完整路径执行安装命令。假定 Anaconda 安装路径为："/opt/anaconda3"，则在终端上执行下述命令安装杉数求解器的 Python 接口：

```
sudo /opt/anaconda3/bin/python setup.py install
```

为测试 Python 接口是否正确安装，用户可切换目录至 /opt/copt70/examples/python ，并在命令行窗口上执行如下命令：

```
python lp_ex1.py
```

若模型正确执行，则表示已正确安装杉数求解器的 Python 接口。

目前，coptpy 已经支持 type hints，用户可以在终端执行如下命令：

```
pip install coptpy-stubs
```

安装成功后，在 Python IDE 中编写代码时，会提示变量名补全及函数参数可取值。

MacOS

方式一：通过 pip install 的方式（推荐）

打开终端 terminal，输入如下命令，安装 COPT Python 接口：

```
pip install coptpy
```

若已安装了旧版本的 `coptpy` 包，请打开终端，输入如下命令，以升级到最新版本的 `coptpy` ：

```
pip install --upgrade coptpy
```

方式二：通过 COPT 安装包

对于 MacOS 系统，假定杉数求解器的安装路径为：/Applications/copt70 ，则切换目录到 /Applications/copt70/lib/python ，并在终端上执行：

```
sudo python setup.py install
```

为测试 Python 接口是否正确安装，用户可切换目录至 /Applications/copt70/examples/python ，并在终端执行如下命令：

```
python lp_ex1.py
```

若模型正确执行，则表示已正确安装杉数求解器的 Python 接口。

目前，`coptpy` 已经支持 type hints，用户可以在终端执行如下命令：

```
pip install coptpy-stubs
```

安装成功后，在 Python IDE 中编写代码时，会提示变量名补全及函数参数可取值。

7.5.2 示例解析

本章通过一个简单的示例演示如何使用杉数求解器的 Python 接口，待求解的问题数学形式如公式 7.5 所示：

$$
\begin{aligned}
&\text{最大化：} \\
&\qquad 1.2x + 1.8y + 2.1z \\
&\text{约束：} \\
&\qquad 1.5x + 1.2y + 1.8z \leq 2.6 \\
&\qquad 0.8x + 0.6y + 0.9z \geq 1.2 \\
&\text{变量范围：} \\
&\qquad 0.1 \leq x \leq 0.6 \\
&\qquad 0.2 \leq y \leq 1.5 \\
&\qquad 0.3 \leq z \leq 2.8
\end{aligned}
\tag{7.5}
$$

使用杉数求解器的 Python 接口求解与分析上述问题的代码见 代码 7.5 ：

代码 7.5: lp_ex1.py

```python
#
# This file is part of the Cardinal Optimizer, all rights reserved.
#

"""
The problem to solve:

  Maximize:
    1.2 x + 1.8 y + 2.1 z

  Subject to:
    1.5 x + 1.2 y + 1.8 z <= 2.6
    0.8 x + 0.6 y + 0.9 z >= 1.2

  where:
    0.1 <= x <= 0.6
    0.2 <= y <= 1.5
    0.3 <= z <= 2.8
"""

import coptpy as cp
from coptpy import COPT

# Create COPT environment
env = cp.Envr()

# Create COPT model
model = env.createModel("lp_ex1")

# Add variables: x, y, z
x = model.addVar(lb=0.1, ub=0.6, name="x")
y = model.addVar(lb=0.2, ub=1.5, name="y")
z = model.addVar(lb=0.3, ub=2.8, name="z")

# Add constraints
model.addConstr(1.5*x + 1.2*y + 1.8*z <= 2.6)
model.addConstr(0.8*x + 0.6*y + 0.9*z >= 1.2)

# Set objective function
model.setObjective(1.2*x + 1.8*y + 2.1*z, sense=COPT.MAXIMIZE)

# Set parameter
model.setParam(COPT.Param.TimeLimit, 10.0)

# Solve the model
```

(续下页)

(接上页)

```
model.solve()

# Analyze solution
if model.status == COPT.OPTIMAL:
  print("Objective value: {}".format(model.objval))
  allvars = model.getVars()

  print("Variable solution:")
  for var in allvars:
    print(" x[{0}]: {1}".format(var.index, var.x))

  print("Variable basis status:")
  for var in allvars:
    print(" x[{0}]: {1}".format(var.index, var.basis))

  # Write model, solution and modified parameters to file
  model.write("lp_ex1.mps")
  model.write("lp_ex1.bas")
  model.write("lp_ex1.sol")
  model.write("lp_ex1.par")
```

接下来我们将基于上述代码分步骤讲解求解与分析过程，详细的 Python 接口使用说明请用户查阅*Python API 参考手册* 。

导入 Python 接口

使用杉数求解器的 Python 接口，需要首先导入 Python 接口库。

```
import coptpy as cp
from coptpy import COPT
```

创建环境

对于任意求解任务，杉数求解器要求首先创建求解环境。

```
# Create COPT environment
env = cp.Envr()
```

创建问题

创建求解环境成功后，用户需要创建模型，模型中将包括待求解的变量、约束等信息。

```
# Create COPT model
model = env.createModel("lp_ex1")
```

添加变量

创建变量时允许同时指定变量在目标函数中的系数、变量上下界等信息。本示例中创建变量时仅指定上下界和名称信息，其它为默认值。

```
import coptpy as cp
from coptpy import COPT

# Create COPT environment
env = cp.Envr()

# Create COPT model
model = env.createModel("lp_ex1")

# Add variables: x, y, z
x = model.addVar(lb=0.1, ub=0.6, name="x")
y = model.addVar(lb=0.2, ub=1.5, name="y")
z = model.addVar(lb=0.3, ub=2.8, name="z")
```

添加约束

添加变量成功后，进一步添加作用于变量的约束条件。

```
# Add constraints
model.addConstr(1.5*x + 1.2*y + 1.8*z <= 2.6)
model.addConstr(0.8*x + 0.6*y + 0.9*z >= 1.2)
```

设置目标函数

添加完变量和约束后，进一步指定模型的目标函数。

```
# Set objective function
model.setObjective(1.2*x + 1.8*y + 2.1*z, sense=COPT.MAXIMIZE)
```

设置求解参数

用户可以在求解模型前设置求解参数，如设置求解时间限制为 10 秒。

```
# Set parameter
model.setParam(COPT.Param.TimeLimit, 10.0)
```

求解模型

调用下述方法求解模型。

```
# Solve the model
model.solve()
```

分析结果

求解完成后，首先获取模型求解状态，若状态为找到了最优解，则进一步获取目标函数值、各变量的取值及其基状态信息。

```
# Analyze solution
if model.status == COPT.OPTIMAL:
  print("Objective value: {}".format(model.objval))
  allvars = model.getVars()

  print("Variable solution:")
  for var in allvars:
    print(" x[{0}]: {1}".format(var.index, var.x))

  print("Variable basis status:")
  for var in allvars:
    print(" x[{0}]: {1}".format(var.index, var.basis))
```

文件输出

用户可以将当前求解的模型保存为标准的 MPS 模型文件，以及输出变量结果文件、基状态信息文件和修改过的参数文件。

```
  # Write model, solution and modified parameters to file
  model.write("lp_ex1.mps")
  model.write("lp_ex1.bas")
  model.write("lp_ex1.sol")
  model.write("lp_ex1.par")
```

7.5.3 最佳实践

升级新版本

对于已经安装了 COPT python 包的用户，如果想升级到最新版本，建议先卸载旧版本的 coptpy，然后再安装新版本。旧版本的 COPT python 包可以从 Python 的 `site-package` 的子目录 `coptpy` 中找到。

多线程编程

COPT 的建模 API 并不保证可重入性，在多线程编程时如果共享模型对象可能造成数据错误。一般来说，在多个线程下共享 COPT 环境类 Envr 是可以的。但是，注意不要共享模型类，除非用户明确知道自己的行为后果。举例来说，如果在两个线程里共享同一个模型类，确保建模和求解只在其中一个线性进行；在另外一个线程里，可以对求解过程监督，然后在合适的时候通过 `interrupt()` 中断任务执行，比如任务超时了。

Python 的字典顺序

Python 默认的字典是无序的，也就是说，遍历次序和数据加入的次序可能不同。例如，输入数据是 `{'fruits': ['apple', 'orange'], 'veggies': ['carrot', 'pea']}`，但是输出的是 `{'veggies': ['carrot', 'pea'], 'fruits': ['apple', 'orange']}`。

从 Python 3.6 开始，Python 字典的内部实现变成有序了。特别是 Python 3.7 及以后，保持字段顺序成为其自带要求。

如果用户有保持字典次序的需求，请使用基于 Python 3.7 或更新版本的 COPT python 包。例如，下面的程序如果是在 Python 2.7 里运行：

```
m = Envr().createModel("customized model")
vx = m.addVars(['hello', 'world'], [0, 1, 2], nameprefix = "X")
# add a constraint for each var in tupledict 'vx'
m.addConstrs(vx[key] >= 1.0 for key in vx)
```

那么，模型里的约束的次序可能是 `{R(hello,1), R(hello,0), R(world,1), R(world,0), R(hello,2), R(world,2)}`。

尽可能使用 quicksum 和 psdquicksum

COPT python 包支持直观地创建线性、二次和半定表达式。对于线性和二次表达式，建议使用 `quicksum()` 来创建表达式对象；对于线性和半定表达式，建议使用 `psdquicksum()` 来创建表达式对象。他们都优化了表达式项的求和，比直接使用加号运算符在性能上会好很多。

7.6 AMPL 接口

AMPL 是一种便捷描述大规模复杂优化问题的代数式建模语言，支持多种商业与开源求解器，提供了丰富的数据接口与扩展功能，有着广泛的商业及学术用户群体，详见 谁在使用 AMPL? 。为了方便用户在 AMPL 建模环境中使用杉数求解器，我们提供了 AMPL 接口 `coptampl` 求解工具，目前支持求解线性规划问题、二次规划问题、二次约束规划问题和混合整数规划问题。一般来说，可以通过如下方式调用 `coptampl` 进行求解：

```
coptampl stub -AMPL
```

其中 `stub.nl` 是 AMPL 定义的 `.nl` 格式通用模型表示文件，可以通过命令 `'ampl -obstub'` 或者 `'ampl -ogstub'` 生成。问题求解完成后，`coptampl` 将结果写入到 AMPL 定义的 `.sol` 格式的 `stub.sol` 文件中，并用于 AMPL 的后续求解与结果分析。上述过程由 AMPL 自动实现，相关的调用命令如下：

```
ampl: option solver coptampl;
ampl: solve;
```

7.6.1 安装说明

用户在 AMPL 环境下使用 `coptampl` 之前，需要正确安装与配置 AMPL 和杉数求解器，详情可参考如何安装杉数求解器 。请进行下文所述不同操作系统下的检查以确保已正确配置 AMPL 和杉数求解器。

Windows

对于 Windows 系统，用户需要将求解工具 `coptampl.exe` 和动态链接库 `copt.dll` 所在的文件目录添加至用户或者系统环境变量 `PATH` 中，或者位于当前求解调用的相同路径下。

在命令行窗口中输入如下命令检测当前设置是否符合上述要求：

```
coptampl -v
```

若设置正确，则将在屏幕输出类似如下信息：

```
AMPL/x-COPT Optimizer [7.0.2] (windows-x86), driver(20220526), MP(20220526)
```

若命令调用失败，请仔细检查您的设置。

Linux

对于 Linux 系统，用户需要将求解工具 `coptampl` 所在的文件目录添加至环境变量 `$PATH` 中，将动态链接库 `libcopt.so` 所在的路径添加至环境变量 `$LD_LIBRARY_PATH` 中。

类似地，在 Linux 终端中输入下述命令检测当前设置是否符合上述要求：

```
coptampl -v
```

若设置正确，则将在屏幕输出类似如下信息：

```
AMPL/x-COPT Optimizer [7.0.2] (linux-x86), driver(20220526), MP(20220526)
```

若命令调用失败，请仔细检查您的设置。

MacOS

对于 MacOS 操作系统，用户需要将求解工具 `coptampl` 所在的文件目录添加至环境变量 `$PATH` 中，将动态链接库 `libcopt.dylib` 所在的路径添加至环境变量 `$DYLD_LIBRARY_PATH` 中。

同样地，在 MacOS 终端中输入下述命令检查当前设置是否符合上述要求：

```
coptampl -v
```

若设置正确，则将在屏幕输出类似如下信息：

```
AMPL/x-COPT Optimizer [7.0.2] (macos-x86), driver(20220526), MP(20220526)
```

若命令调用失败，请仔细检查您的设置。

7.6.2 求解参数与返回值

`coptampl` 提供了一些求解参数允许用户自定义求解行为。用户可以通过设置 `copt_options` 环境变量或者在 AMPL 中通过 `option` 命令来改变 `coptampl` 的求解参数，并可以通过下述命令查看目前支持的所有参数：

```
coptampl -=
```

对于当前版本的 `coptampl` 求解工具，支持的求解参数及其含义如 表 7.1：

表 7.1: `coptampl` 求解参数

参数名称	参数含义
barhomogeneous	是否使用齐次自对偶方法
bariterlimit	内点法迭代数限制
barthreads	内点法使用的线程数
basis	是否读取或输出基状态
bestbound	是否返回最优下界后缀信息
conflictanalysis	是否使用冲突分析
crossoverthreads	crossover 使用的线程数
cutlevel	割平面生成强度
divingheurlevel	diving 启发式算法的强度
dualize	是否构建并求解对偶模型
dualperturb	是否允许对目标函数进行扰动
dualprice	指定对偶单纯形法的 Pricing 算法
dualtol	对偶解的可行性容差
feastol	变量、约束取值的可行性容差
heurlevel	启发式算法强度

续下页

表 7.1 – 接上页

iisfind	是否计算不可行模型的 IIS 并返回结果
iismethod	指定计算 IIS 的方法
inttol	变量的整数解容差
logging	是否显示求解日志
logfile	日志文件
exportfile	导出模型文件
lpmethod	求解线性规划问题的算法
matrixtol	输入矩阵的系数容差
mipstart	是否使用整数规划初始解
miptasks	MIP 求解使用的任务数
nodecutrounds	搜索树节点生成割平面的次数
nodelimit	整数规划求解的节点数限制
objno	目标函数的序号
count	是否输出的解池结果文件的数目
stub	解池结果文件的前缀
presolve	预求解的强度
relgap	整数规划的最优相对容差
absgap	整数规划的最优绝对容差
return_mipgap	是否返回整数规划最优绝对或相对容差
rootcutlevel	根节点生成割平面的强度
rootcutrounds	根节点生成割平面的次数
roundingheurlevel	rounding 启发式算法的强度
scaling	是否在求解前调整模型系数矩阵的系数
simplexthreads	对偶单纯形法使用的线程数
sos	是否识别 `'.sosno'` 和 `'.ref'` 后缀
sos2	是否将非凸分段线性项使用 SOS2 约束表示
strongbranching	strong branching 的强度
submipheurlevel	基于子 MIP 的启发式算法的强度
threads	问题求解使用的线程数
timelimit	模型求解的时间限制
treecutlevel	搜索树生成割平面的强度
wantsol	是否生成 `'.sol'` 文件

请用户参考*COPT 参数设置* 章节查看各参数的详细使用说明。

AMPL 中利用后缀存储或传递模型信息与解信息，并利用后缀实现一些扩展功能，如 SOS 约束的支持。目前，`coptampl` 支持的后缀信息见 表 7.2：

表 7.2: coptampl 后缀

后缀名	含义
absmipgap	整数规划最优绝对容差
bestbound	整数规划求解结束时最好的下界
iis	存储变量或约束的 IIS 状态
nsol	输出解池的解的数目
ref	指定 SOS 约束中变量的权重
relmipgap	整数规划最优相对容差
sos	存储 AMPL 生成的 SOS 约束类型
sosno	指定 SOS 约束类型
sosref	存储 AMPL 生成的 SOS 约束中变量的权重
sstatus	存储线性规划求解后的基状态信息

关于如何在 AMPL 中开启 SOS 约束的识别扩展功能，详见 AMPL 官网的资料：如何在 AMPL 中使用 SOS 约束 。

求解完成后，coptampl 在屏幕输出求解状态信息并传递返回值。通过下述命令显示返回值：

```
ampl: display solve_result_num;
```

如果未得到结果或者发生错误，coptampl 将依据 表 7.3 返回非零值给 AMPL，如下所示：

表 7.3: coptampl 返回值

返回值	含义
0	最优解
200	模型不可行
300	模型无界
301	模型无解或无界
600	用户中止

7.6.3 使用示例

本节将通过一个著名的案例 “Diet 问题” 来演示 AMPL 的用法。该问题目的是找到给定食物的搭配以满足不同种类的营养元素需求，详见 AMPL 官方手册。

假定已知种类的食物及其定价信息如 表 7.4 所示：

表 7.4: 食物价格

食物	价格
BEEF	3.19
CHK	2.59
FISH	2.29
HAM	2.89
MCH	1.89
MTL	1.99
SPG	1.99
TUR	2.49

每种食物及其单位营养元素含量占每天所需要的最少营养含量需求如 `coptTab_ampldiet_2` 所示：

该问题的求解目标是找出满足至少 7 倍于日常营养需要的食物搭配，且价格花费最少。

综上所述，该问题的数学形式如公式 7.6 所示：

$$
\begin{aligned}
&\text{最大化：} \\
&\quad \sum_{j \in J} cost_j \cdot buy_j \\
&\text{约束：} \\
&\quad n_min_i \le \sum_{j \in J} amt_{i,j} \cdot buy_j \le n_max_i \ \ \forall i \in I \\
&\quad f_min_j \le buy_j \le f_max_j \ \ \forall j \in J
\end{aligned}
\tag{7.6}
$$

该问题的 AMPL 模型 `diet.mod` 见 代码 7.6 ：

代码 7.6: `diet.mod`

```
# The code is adopted from:
#
# https://github.com/Pyomo/pyomo/blob/master/examples/pyomo/amplbook2/diet.mod
#
# with some modification by developer of the Cardinal Optimizer

set NUTR;
set FOOD;

param cost {FOOD} > 0;
param f_min {FOOD} >= 0;
param f_max {j in FOOD} >= f_min[j];

param n_min {NUTR} >= 0;
param n_max {i in NUTR} >= n_min[i];

param amt {NUTR, FOOD} >= 0;

```

(续下页)

(接上页)

```
var Buy {j in FOOD} >= f_min[j], <= f_max[j];

minimize Total_Cost:
  sum {j in FOOD} cost[j] * Buy[j];

subject to Diet {i in NUTR}:
  n_min[i] <= sum {j in FOOD} amt[i, j] * Buy[j] <= n_max[i];
```

求解所需的数据 diet.dat 见 代码 7.7 ：

代码 7.7: diet.dat

```
# The data is adopted from:
#
# https://github.com/Pyomo/pyomo/blob/master/examples/pyomo/amplbook2/diet.dat
#
# with some modification by developer of the Cardinal Optimizer

data;

set NUTR := A B1 B2 C ;
set FOOD := BEEF CHK FISH HAM MCH MTL SPG TUR ;

param:   cost  f_min  f_max :=
  BEEF   3.19    0     100
  CHK    2.59    0     100
  FISH   2.29    0     100
  HAM    2.89    0     100
  MCH    1.89    0     100
  MTL    1.99    0     100
  SPG    1.99    0     100
  TUR    2.49    0     100 ;

param:   n_min  n_max :=
   A      700   10000
   C      700   10000
   B1     700   10000
   B2     700   10000 ;

param amt (tr):
           A    C   B1   B2 :=
   BEEF   60   20   10   15
   CHK     8    0   20   20
   FISH    8   10   15   10
   HAM    40   40   35   10
   MCH    15   35   15   15
   MTL    70   30   15   15
```

(续下页)

(接上页)

```
   SPG    25   50   25   15
   TUR    60   20   15   10 ;
```

若在 AMPL 中调用 coptampl 进行求解，则在进入 AMPL 交互界面后输入下述命令：

```
ampl: model diet.mod;
ampl: data diet.dat;
ampl: option solver coptampl;
ampl: option copt_options 'logging 1';
ampl: solve;
```

coptampl 快速求解完成，并在屏幕输出求解日志及结果状态信息：

```
x-COPT 5.0.1: optimal solution; objective 88.2
1 simplex iterations
```

分析可知，coptampl 找到了问题的最优解，最优解为 88.2 个单位。用户可以进一步显示问题中各变量的取值信息：

```
ampl: display Buy;
```

屏幕中输出如下信息：

```
Buy [*] :=
BEEF   0
 CHK   0
FISH   0
 HAM   0
 MCH  46.6667
 MTL   0
 SPG   0
 TUR   0
;
```

分析可知，当购买约 46.667 个单位的食物 MCH 时，花费最少，为 88.2 个单位。

7.7 Pyomo 接口

Pyomo 是基于 Python 编程语言的开源优化建模语言，它提供了丰富的优化相关功能，并已被许多研究人员用于解决复杂的实际应用问题，感兴趣的用户可以查看 谁在使用 Pyomo? 了解更多信息。本章介绍了如何在 Pyomo 环境下使用杉数求解器。

7.7.1 安装说明

在 Pyomo 环境下调用杉数求解器进行求解之前，用户需要正确安装与配置 Pyomo 和杉数求解器。Pyomo 目前支持 Python 2.7 和 3.6-3.10 版本，用户可以从 Anaconda 发行版 或者 Python 官方发行版 下载并安装 Python。我们推荐用户安装 Anaconda 发行版，因为它对 Python 新手使用更加友好与方便。

使用 conda 安装

我们推荐安装了 Anaconda 发行版 Python 的用户使用它自带的 `conda` 工具安装 Pyomo，在 Windows 的命令行或者 Linux 和 MacOS 上的终端中执行下述命令即可：

```
conda install -c conda-forge pyomo
```

Pyomo 也集成了一些可选的第三方 Python 包扩展其优化相关功能，可通过如下命令进行安装：

```
conda install -c conda-forge pyomo.extras
```

使用 pip 安装

用户也可以通过标准的 `pip` 工具安装 Pyomo，在 Windows 的命令行或者 Linux 和 MacOS 的终端中执行下述命令即可：

```
pip install pyomo
```

如果用户在安装 Pyomo 过程中遇到了任何问题，可以查阅 如何安装 Pyomo 以了解更多信息。关于安装与配置杉数求解器，请用户查看文档中的*如何安装杉数求解器* 章节了解详细步骤。

7.7.2 使用示例

我们将通过求解*AMPL 接口-使用示例* 章节中描述的例子来介绍如何在 Pyomo 中调用杉数求解器进行优化求解。如果用户想了解更详细的 Pyomo 使用说明，可以参考 Pyomo 官方文档 进行学习。

抽象模型

Pyomo 主要提供了两种方式对其支持的各种问题类型进行建模，本部分将介绍使用抽象模型求解上述问题的方式。

使用 Pyomo 对上述问题进行建模与求解的源代码 `pydiet_abstract.py` 如下，详见 代码 7.8：

代码 7.8: `pydiet_abstract.py`

```
# The code is adopted from:
#
# https://github.com/Pyomo/pyomo/blob/master/examples/pyomo/amplbook2/diet.py
#
# with some modification by developer of the Cardinal Optimizer
```

(续下页)

(接上页)

```
6
7  from pyomo.core import *
8
9  model = AbstractModel()
10
11 model.NUTR = Set()
12 model.FOOD = Set()
13
14 model.cost  = Param(model.FOOD, within=NonNegativeReals)
15 model.f_min = Param(model.FOOD, within=NonNegativeReals)
16
17 model.f_max = Param(model.FOOD)
18 model.n_min = Param(model.NUTR, within=NonNegativeReals)
19 model.n_max = Param(model.NUTR)
20 model.amt   = Param(model.NUTR, model.FOOD, within=NonNegativeReals)
21
22 def Buy_bounds(model, i):
23   return (model.f_min[i], model.f_max[i])
24 model.Buy = Var(model.FOOD, bounds=Buy_bounds)
25
26 def Objective_rule(model):
27   return sum_product(model.cost, model.Buy)
28 model.totalcost = Objective(rule=Objective_rule, sense=minimize)
29
30 def Diet_rule(model, i):
31   expr = 0
32
33   for j in model.FOOD:
34     expr = expr + model.amt[i, j] * model.Buy[j]
35
36   return (model.n_min[i], expr, model.n_max[i])
37 model.Diet = Constraint(model.NUTR, rule=Diet_rule)
```

该问题的数据文件 pydiet_abstract.dat 见 代码 7.9 :

代码 7.9: pydiet_abstract.dat

```
1  # The data is adopted from:
2  #
3  # https://github.com/Pyomo/pyomo/blob/master/examples/pyomo/amplbook2/diet.dat
4  #
5  # with some modification by developer of the Cardinal Optimizer
6
7  data;
8
9  set NUTR := A B1 B2 C ;
10 set FOOD := BEEF CHK FISH HAM MCH MTL SPG TUR ;
```

(续下页)

(接上页)

```
11
12 param:   cost  f_min  f_max :=
13   BEEF   3.19    0     100
14   CHK    2.59    0     100
15   FISH   2.29    0     100
16   HAM    2.89    0     100
17   MCH    1.89    0     100
18   MTL    1.99    0     100
19   SPG    1.99    0     100
20   TUR    2.49    0     100 ;
21
22 param:   n_min  n_max :=
23    A      700   10000
24    C      700   10000
25    B1     700   10000
26    B2     700   10000 ;
27
28 param amt (tr):
29            A    C   B1   B2 :=
30    BEEF   60   20   10   15
31    CHK     8    0   20   20
32    FISH    8   10   15   10
33    HAM    40   40   35   10
34    MCH    15   35   15   15
35    MTL    70   30   15   15
36    SPG    25   50   25   15
37    TUR    60   20   15   10 ;
```

通过在 Windows 的命令行或者 Linux 和 Mac 的终端中输入下述命令即可实现在 Pyomo 中调用杉数求解器进行求解：

```
pyomo solve --solver=coptampl pydiet_abstract.py pydiet_abstract.dat
```

求解过程中，Pyomo 在屏幕中输出如下信息：

```
[    0.00] Setting up Pyomo environment
[    0.00] Applying Pyomo preprocessing actions
[    0.00] Creating model
[    0.01] Applying solver
[    0.05] Processing results
    Number of solutions: 1
    Solution Information
      Gap: None
      Status: optimal
      Function Value: 88.19999999999999
    Solver results file: results.yml
[    0.05] Applying Pyomo postprocessing actions
```

(续下页)

(接上页)

```
[    0.05] Pyomo Finished
```

求解完成后，Pyomo 将求解结果输出到如下 `results.yml` 中：

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Lower bound: -inf
  Upper bound: inf
  Number of objectives: 1
  Number of constraints: 4
  Number of variables: 8
  Sense: unknown
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Message: COPT-AMPL\x3a optimal solution; objective 88.2, iterations 1
  Termination condition: optimal
  Id: 0
  Error rc: 0
  Time: 0.03171110153198242
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 1
  number of solutions displayed: 1
- Gap: None
  Status: optimal
  Message: COPT-AMPL\x3a optimal solution; objective 88.2, iterations 1
  Objective:
    totalcost:
      Value: 88.19999999999999
  Variable:
    Buy[MCH]:
      Value: 46.666666666666664
  Constraint: No values
```

分析结果知，杉数求解器找到了最优解约为 88.2 个单位，此时应当购买约 46.67 个单位的食物 MCH。

具象模型

除了使用抽象模型进行建模，Pyomo 还支持具象模型建模方式，本部分将介绍使用具象模型对上述问题进行建模并求解。

具象模型可以使用 "Direct" 和 "Persistent" 接口方式进行求解。该方式依赖杉数求解器的 Pyomo 插件文件 copt_pyomo.py，文件位于安装包的 "lib/pyomo" 子文件夹。使用该插件需要将 copt_pyomo.py 文件复制到待求解 Python 程序的相同目录下，且已正确安装了相应 Python 版本的杉数求解器 Python 接口。

使用 Pyomo 建模求解的源代码 pydiet_concrete.py 见 代码 7.10:

代码 7.10: pydiet_concrete.py

```
# The code is adopted from:
#
# https://github.com/Pyomo/pyomo/blob/master/examples/pyomo/amplbook2/diet.py
#
# with some modification by developer of the Cardinal Optimizer

from __future__ import print_function, division

import pyomo.environ as pyo
import pyomo.opt as pyopt

from copt_pyomo import *

# Nutrition set
NUTR = ["A", "C", "B1", "B2"]
# Food set
FOOD = ["BEEF", "CHK", "FISH", "HAM", "MCH", "MTL", "SPG", "TUR"]

# Price of foods
cost = {"BEEF": 3.19, "CHK": 2.59, "FISH": 2.29, "HAM": 2.89, "MCH": 1.89,
        "MTL":  1.99, "SPG": 1.99, "TUR":  2.49}
# Nutrition of foods
amt = {"BEEF": {"A": 60, "C": 20, "B1": 10, "B2": 15},
       "CHK":  {"A": 8,  "C": 0,  "B1": 20, "B2": 20},
       "FISH": {"A": 8,  "C": 10, "B1": 15, "B2": 10},
       "HAM":  {"A": 40, "C": 40, "B1": 35, "B2": 10},
       "MCH":  {"A": 15, "C": 35, "B1": 15, "B2": 15},
       "MTL":  {"A": 70, "C": 30, "B1": 15, "B2": 15},
       "SPG":  {"A": 25, "C": 50, "B1": 25, "B2": 15},
       "TUR":  {"A": 60, "C": 20, "B1": 15, "B2": 10}}

# The "diet problem" using ConcreteModel
model = pyo.ConcreteModel()

model.NUTR = pyo.Set(initialize=NUTR)
```

(续下页)

(接上页)

```
model.FOOD = pyo.Set(initialize=FOOD)

model.cost = pyo.Param(model.FOOD, initialize=cost)

def amt_rule(model, i, j):
  return amt[i][j]
model.amt  = pyo.Param(model.FOOD, model.NUTR, initialize=amt_rule)

model.f_min = pyo.Param(model.FOOD, default=0)
model.f_max = pyo.Param(model.FOOD, default=100)

model.n_min = pyo.Param(model.NUTR, default=700)
model.n_max = pyo.Param(model.NUTR, default=10000)

def Buy_bounds(model, i):
  return (model.f_min[i], model.f_max[i])
model.buy = pyo.Var(model.FOOD, bounds=Buy_bounds)

def Objective_rule(model):
  return pyo.sum_product(model.cost, model.buy)
model.totalcost = pyo.Objective(rule=Objective_rule, sense=pyo.minimize)

def Diet_rule(model, j):
  expr = 0

  for i in model.FOOD:
    expr = expr + model.amt[i, j] * model.buy[i]

  return (model.n_min[j], expr, model.n_max[j])
model.Diet = pyo.Constraint(model.NUTR, rule=Diet_rule)

# Reduced costs of variables
model.rc = pyo.Suffix(direction=pyo.Suffix.IMPORT)

# Activities and duals of constraints
model.slack = pyo.Suffix(direction=pyo.Suffix.IMPORT)
model.dual = pyo.Suffix(direction=pyo.Suffix.IMPORT)

# Use 'copt_direct' solver to solve the problem
solver = pyopt.SolverFactory('copt_direct')

# Use 'copt_persistent' solver to solve the problem
# solver = pyopt.SolverFactory('copt_persistent')
# solver.set_instance(model)

results = solver.solve(model, tee=True)

```

(续下页)

(接上页)

```
# Check result
print("")
if results.solver.status == pyopt.SolverStatus.ok and \
   results.solver.termination_condition == pyopt.TerminationCondition.optimal:
  print("Optimal solution found")
else:
  print("Something unexpected happened: ", str(results.solver))

print("")
print("Optimal objective value:")
print("  totalcost: {0:6f}".format(pyo.value(model.totalcost)))

print("")
print("Variables solution:")
for i in FOOD:
  print("  buy[{0:4s}] = {1:9.6f} (rc: {2:9.6f})".format(i, \
                                                 pyo.value(model.buy[i]), \
                                                 model.rc[model.buy[i]]))

print("")
print("Constraint solution:")
for i in NUTR:
  print("  diet[{0:2s}] = {1:12.6f} (dual: {2:9.6f})".format(i, \
                                                     model.slack[model.Diet[i]], \
                                                     model.dual[model.Diet[i]]))
```

然后，在 Windows 的命令行或者 Linux 和 MacOS 的终端中执行下述命令进行求解：

```
python pydiet_concrete.py
```

求解完成后，屏幕输出如下内容：

```
Optimal solution found
Objective:
  totalcost: 88.200000
Variables:
  buy[BEEF] = 0.000000
  buy[CHK ] = 0.000000
  buy[FISH] = 0.000000
  buy[HAM ] = 0.000000
  buy[MCH ] = 46.666667
  buy[MTL ] = 0.000000
  buy[SPG ] = 0.000000
  buy[TUR ] = 0.000000
```

结果显示，杉数求解器找到了最优解约 88.2 个单位，此时应当购买约 46.67 个单位的食物 MCH。

7.8 PuLP 接口

PuLP 是基于 Python 编程语言的开源优化建模语言，主要用于对整数线性规划问题进行建模求解。本章介绍了如何在 PuLP 中使用杉数求解器。

7.8.1 安装说明

在 PuLP 中调用杉数求解器进行求解之前，用户需要正确安装与配置 PuLP 和杉数求解器。PuLP 目前支持 Python 2.7 及之后的 Python 版本。用户可以从 Anaconda 发行版 或者 Python 官方发行版 下载并安装 Python。我们推荐用户安装 Anaconda 发行版，因为它对 Python 新手使用更加友好与方便。

使用 conda 安装

我们推荐安装了 Anaconda 发行版 Python 的用户使用它自带的 `conda` 工具安装 PuLP，在 Windows 的命令行或者 Linux 和 MacOS 上的终端中执行下述命令即可：

```
conda install -c conda-forge pulp
```

使用 pip 安装

用户也可以通过标准的 `pip` 工具安装 PuLP，在 Windows 的命令行或者 Linux 和 MacOS 的终端中执行下述命令即可：

```
pip install pulp
```

7.8.2 配置 PuLP 接口

杉数求解器的 PuLP 接口实现代码在杉数求解器安装包中的 Python 文件 `copt_pulp.py` 中。在安装与配置好杉数求解器后，用户只需要将该接口文件放在项目代码工程目录中，并导入该插件：

```
from copt_pulp import *
```

在 PuLP 的求解函数 `solve` 中，指定使用杉数求解器进行求解：

```
prob.solve(COPT_DLL())
```

或者：

```
prob.solve(COPT_CMD())
```

7.8.3 功能介绍

杉数求解器的 PuLP 接口提供了命令行调用和动态库调用两种方式，分别介绍如下：

命令行调用方式

PuLP 接口的命令行调用方式实际上调用了杉数求解器的交互式命令行工具 `copt_cmd` 进行求解，该方式下 PuLP 生成模型对应的 MPS 格式文件，结合用户传入的参数设置，生成命令行调用命令。求解完成后，输出结果文件并读取结果，赋值给相应的变量并传回 PuLP。

命令行调用方式相关功能封装为 Python 类 `COPT_CMD`，用户可在创建类的对象时设置参数控制求解过程，目前主要提供了以下参数：

- `keepFiles`

  该选项控制是否保留生成的临时文件。默认值为 `0`，即不保留临时文件。

- `mip`

  该选项控制是否支持求解整数规划模型。默认值为 `True`，即支持求解整数规划。

- `msg`

  该选项控制是否在屏幕打印日志信息。默认值为 `True`，即打印日志信息。

- `mip_start`

  该选项控制是否使用整数规划初始解信息。默认值为 `False`，即不使用初始解信息。

- `logfile`

  该选项指定求解器日志。默认值为 `None`，即不生成求解器日志。

- `params`

  该选项以 `key=value` 形式设置优化参数。请用户查看参数 章节查看目前支持的优化参数。

动态库调用方式

PuLP 接口的动态库调用方式直接调用杉数求解器算法动态库进行求解，该方式下将获取 PuLP 生成的模型信息并调用相应的接口函数加载模型到杉数求解器，结合用户传入的参数设置，进行优化求解。求解完成后，通过调用接口函数获取求解结果，赋值给相应的变量和约束并传回 PuLP。

动态库调用方式相关功能封装为 Python 类 `COPT_DLL`，用户可在创建类的对象时设置参数控制求解过程，目前主要提供了以下参数：

- `mip`

  该选项控制是否支持求解整数规划模型。默认值为 `True`，即支持求解整数规划。

- `msg`

  该选项控制是否在屏幕打印日志信息。默认值为 `True`，即打印日志信息。

- `mip_start`

  该选项控制是否使用整数规划初始解信息。默认值为 `False`，即不使用初始解信息。

- `logfile`

  该选项指定求解器日志。默认值为 `None`，即不生成求解器日志。

- `params`

  该选项以 `key=value` 形式设置优化参数。请用户查看参数 章节查看目前支持的优化参数。

此外，还提供了以下类方法：

- `setParam(self, name, val)`

  设置优化求解参数。

- `getParam(self, name)`

  获取优化求解参数。

- `getAttr(self, name)`

  获取模型的属性信息。

- `write(self, filename)`

  输出 MPS/LP 格式模型文件、COPT 二进制格式文件、结果文件、基解文件、初始解文件和参数设置文件。

7.9 CVXPY 接口

CVXPY 是一种基于 Python 编程语言的开源凸优化问题建模工具。它允许用户以自然的数学方式表达待求解的问题，十分方便且高效。本章介绍了如何在 CVXPY 中使用杉数求解器。

7.9.1 安装说明

在 CVXPY 中调用杉数求解器进行求解之前，用户需要正确安装与配置 CVXPY 和杉数求解器。CVXPY 目前支持 Python 3.7 及之后的 Python 版本。用户可以从 Anaconda 发行版 或者 Python 官方发行版 下载并安装 Python。我们推荐用户安装 Anaconda 发行版，因为它对 Python 新手使用更加友好与方便。

使用 conda 安装

我们推荐安装了 Anaconda 发行版 Python 的用户使用它自带的 `conda` 工具安装 CVXPY，在 Windows 的命令行或者 Linux 和 MacOS 上的终端中执行下述命令即可：

```
conda install -c conda-forge cvxpy
```

使用 pip 安装

用户也可以通过标准的 pip 工具安装 CVXPY，在 Windows 的命令行或者 Linux 和 MacOS 的终端中执行下述命令即可:

```
pip install cvxpy
```

7.9.2 配置 CVXPY 接口

CVXPY V1.3 及以上版本支持直接调用 COPT。用户需提前安装并配置好杉数求解器后，接着:

```
import cvxpy as cp
```

在 CVXPY 的求解函数 solve 中，指定参数 solver="COPT" 使用杉数求解器进行求解:

```
prob.solve(solver="COPT")
```

7.9.3 功能介绍

杉数求解器的 CVXPY 接口支持求解线性规划问题（LP）、混合整数规划问题（MIP）、半定规划问题（SDP）、凸二次规划问题（convex QP）、二阶锥规划问题（SOCP）、混合整数凸二次规划问题（convex MIQP）和混合整数二阶锥规划问题（MISOCP），常用的参数有:

- verbose

  CVXPY 参数，控制是否显示详细的求解日志，默认为 False，即不显示求解日志。

- params

  该选项以 key=value 形式设置优化参数。请用户查看参数 章节查看目前支持的优化参数。

第 8 章　一般常数

COPT 有三种类型的常量。

1. 用来构建模型的，比如优化方向，约束类型等；
2. 用来查询求解结果的，比如 API 函数返回值，基、解值等；
3. 用来监视求解进程的，比如回调函数的触发点。

8.1 版本信息

- `VERSION_MAJOR`

  软件大版本号

- `VERSION_MINOR`

  软件小版本号

- `VERSION_TECHNICAL`

  软件修复版本编号

8.2 优化方向

在不同的优化场景中，需要最大化或最小化目标函数。为此，我们提供了两种优化方向的常数：

- `MAXIMIZE`

  最大化目标函数

- `MINIMIZE`

  最小化目标函数

当读取一个文件时，优化方向会自动设置。此外，COPT 提供相应函数，通过指定参数 `sense` 的值，让用户手动设置优化方向。在不同的编程接口中的函数如 表 8.1 所示：

表 8.1: 设置目标函数优化方向的函数

编程接口	函数
C	`COPT_SetObjSense`
C++	`Model::SetObjSense()`
C#	`Model.SetObjSense()`
Java	`Model.setObjSense()`
Python	`Model.setObjSense()`

注意：在不同编程接口中的函数名称、调用方式以及参数名称略有不同，详细请参考各编程语言 API 参考手册。

8.3 无边界和未定义

无边界

在 COPT 中，我们使用一个很大的量表达无边界的情况。这个量以常数提供：

- `INFINITY`

  代表无边界的量的默认值 (`1e30`)

未定义

在 COPT 中，我们使用另一个很大的量表达数据未定义的情况。例如，初始解文件（`".mst"` ）中省略的变量的赋值。这个量以常数提供：

- `UNDEFINED`

  代表数据未定义的默认值 (`1e40`)

8.4 约束类型

在优化领域最初兴起的时候，人们往往用约束类型（ `sense` ）来定义一个约束。常见的类型有：

- `LESS_EQUAL`

  形如 $g(x) \leq b$ 的约束

- `GREATER_EQUAL`

  形如 $g(x) \geq b$ 的约束

- `EQUAL`

  形如 $g(x) = b$ 的约束

- `FREE`

  无边界约束的表达式

- `RANGE`

同时有上下边界的，形如 $l \leq g(x) \leq u$ 的约束

注意： COPT 支持使用约束类型来定义约束，但不推荐，我们推荐直接使用上下边界来定义约束。

8.5 变量类型

变量类型指的是一个变量是连续变量、二进制变量或者整数变量。COPT 支持将决策变量设置为以下三种类型：

- `CONTINUOUS`

  连续变量

- `BINARY`

  二进制变量（0-1 变量）

- `INTEGER`

  整数变量

8.6 SOS 约束类型

SOS 约束（Special Ordered Set）是一类限制一组变量取值的特殊约束。目前，COPT 支持两种 SOS 约束：

- `SOS_TYPE1`

  SOS1 约束

  该类型约束中指定的一组变量至多有一个变量可取非零值

- `SOS_TYPE2`

  SOS2 约束

  该类型约束中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

注意： SOS 约束中的变量类型可取连续变量、二进制变量和整数变量。

8.7 二阶锥约束类型

二阶锥约束是一类特殊的二次约束，目前，COPT 支持两种二阶锥约束：

- `CONE_QUAD`：标准二阶锥

$$Q^n = \left\{ x \in \mathbb{R}^n \;\middle|\; x_0 \geq \sqrt{\sum_{i=1}^{n-1} x_i^2}, x_0 \geq 0 \right\} \tag{8.1}$$

- `CONE_RQUAD`：旋转二阶锥

$$Q_r^n = \left\{ x \in \mathbb{R}^n \;\middle|\; 2x_0 x_1 \geq \sum_{i=2}^{n-1} x_i^2, x_0 \geq 0, x_1 \geq 0 \right\} \tag{8.2}$$

8.8 基状态

对于一个有 n 个变量、m 个约束的优化模型，它在内部计算时，m 个约束会被当做 m 个松弛变量看待。这样一共就有 $n+m$ 个变量了。

当使用单纯形法求解模型时，它会把 n 个变量固定在其有限边界上，并计算另外 m 个变量的取值。这 m 个计算取值的变量叫做 **基**变量，而另外那 n 个变量叫做 **非基**变量。单纯形法的求解过程，以及最终的解，都可以用变量的基状态来表达。COPT 中变量/约束基状态取值及其含义如 表 8.2 所示：

表 8.2: 基状态码对应取值及其含义

基状态码	取值	含义
BASIS_LOWER	0	非基变量，取值下边界
BASIS_BASIC	1	基变量
BASIS_UPPER	2	非基变量，取值上边界
BASIS_SUPERBASIC	3	非基变量，但取值非上下边界
BASIS_FIXED	4	非基变量，固定在它唯一的边界（上下边界相等）

8.9 解状态

优化问题的求解状态即为解状态。模型解状态可能取值如 表 8.3 ：

表 8.3: 解状态码及其含义

解状态码	含义
UNSTARTED	尚未开始求解
OPTIMAL	找到了最优解
INFEASIBLE	模型是无解的
UNBOUNDED	目标函数在优化方向没有边界
INF_OR_UNB	模型无解或目标函数在优化方向没有边界
NUMERICAL	求解遇到数值问题
NODELIMIT	在节点限制到达前未能完成求解
TIMEOUT	在时间限制到达前未能完成求解
UNFINISHED	求解终止。但是由于数值问题，求解器无法给出结果
IMPRECISE	求解结果不准确
INTERRUPTED	用户中止

注意

- 在 Java API 和 C# API 的常量类中，关于解状态的常数定义在 `Status` 类里；
- 在 Python API 中，解状态定义在 COPT 的一般常数类中，通过 `"COPT."` 前缀或 `Model.status` 访问；
- 通过属性 `"LpStatus"` 可以获取线性规划解状态，通过属性 `"MipStatus"` 可以获取整数规划解状态。

- 通过 Callback 信息 "NodeStatus" 可以获取当前节点 LP 松弛问题的求解状态。除去 NODELIMIT, UNSTARTED, INF_OR_UNB ，其他均为其可能取值。

8.10 回调函数的触发点

- CBCONTEXT_MIPRELAX

  当发现 MIP 线性松弛解时，触发回调函数。

- CBCONTEXT_MIPSOL

  当发现 MIP 可行解时，触发回调函数。

- CBCONTEXT_MIPNODE

  当处理完成 MIP 节点并求解线性松弛问题完成时，触发回调函数。

8.11 客户端配置参数

对于浮动和集群服务器的客户端，用户可以通过调用接口函数设置客户端配置参数，目前提供的配置参数有：

- CLIENT_CLUSTER

  远程服务器的 IP 地址。

- CLIENT_FLOATING

  令牌服务器的 IP 地址。

- CLIENT_PASSWORD

  远程服务器的密码。

- CLIENT_PORT

  令牌服务器的通信端口。

- CLIENT_WAITTIME

  客户端连接等待时间。

8.12 访问常数相关操作

在不同的编程接口中，访问常数的方式略有差别，在 C 语言接口中常数名称多加前缀 "COPT_" （形如 COPT_MAXIMIZE ）。具体请参考各编程语言 API 参考手册的对应章节部分：

- C 接口：*C API 参考手册：常量章节*
- C++ 接口：*C++ API 参考手册：常量章节*
- C# 接口：*C# API 参考手册：一般常数章节*

- Java 接口：*Java API 参考手册：一般常数章节*
- Python 接口：*Python API 参考手册：一般常数章节*

第 9 章　属性

属性类常数包括优化模型相关和求解结果相关两类属性。本章节将介绍杉数求解器 COPT 提供的属性常数及其含义。章节内容构成如下：

- 优化模型相关属性
- 求解结果相关属性
- 属性获取方式

9.1 优化模型相关属性

优化模型相关属性提供模型构成和描述的相关信息。

表 9.1: 优化模型相关属性总览

属性名	类型	属性含义
Cols	整数属性	变量（系数矩阵列）个数
PSDCols	整数属性	半定变量的个数
Rows	整数属性	约束（系数矩阵行）的个数
Elems	整数属性	系数矩阵中非零元素个数
QElems	整数属性	二次目标函数中非零二次项个数
PSDElems	整数属性	目标函数中半定项个数
SymMats	整数属性	模型中对称矩阵的个数
Bins	整数属性	二进制变量（列）的个数
Ints	整数属性	整数变量（列）的个数
Soss	整数属性	SOS 约束的个数
Cones	整数属性	二阶锥约束的个数
QConstrs	整数属性	二次约束的个数
PSDConstrs	整数属性	半定约束的个数
LMIConstrs	整数属性	LMI 约束的个数
Indicators	整数属性	Indicator 约束的个数
ObjSense	整数属性	优化方向
ObjConst	浮点数属性	目标函数的常数部分
HasQObj	整数属性	模型是否包含二次项目标函数
HasPSDObj	整数属性	模型的目标函数是否包含半定项
IsMIP	整数属性	模型是否为整数规划模型

注意： Double 这个词的准确翻译应是双精度浮点数。表格中和下文为了简便，称之为浮点数。

- `Cols`

  整数属性。

  变量（系数矩阵列）的个数。

- `PSDCols`

  整数属性。

  半定变量的个数。

- `Rows`

  整数属性。

  约束（系数矩阵行）的个数。

- `Elems`

  整数属性。

  系数矩阵的非零元素个数。

- `QElems`

  整数属性。

  二次目标函数中非零二次项个数。

- `PSDElems`

  整数属性。

  目标函数中半定项个数。

- `SymMats`

  整数属性。

  模型中对称矩阵的个数。

- `Bins`

  整数属性。

  二进制变量（列）的个数。

- `Ints`

  整数属性。

  整数变量（列）的个数。

- `Soss`

  整数属性。

  SOS 约束的个数。

- `Cones`

整数属性。

二阶锥约束的个数。

- `QConstrs`

  整数属性。

  二次约束的个数。

- `PSDConstrs`

  整数属性。

  半定约束的个数。

- `LMIConstrs`

  整数属性。

  LMI （Linear Matrix Inequalities，线性矩阵不等式）约束的个数。

- `Indicators`

  整数属性。

  Indicator 约束的个数。

- `ObjSense`

  整数属性。

  优化方向。

- `ObjConst`

  浮点数属性。

  目标函数的常数部分。

- `HasQObj`

  整数属性。

  模型是否包含二次项目标函数。

- `HasPSDObj`

  整数属性。

  模型的目标函数是否包含半定项。

- `IsMIP`

  整数属性。

  模型是否为整数规划模型。

9.2 求解结果相关属性

求解结果相关属性提供模型求解结果构成和描述的相关信息。

表 9.2: 求解结果相关属性总览

属性名	类型	属性含义
LpStatus	整数属性	线性规划求解状态。
MipStatus	整数属性	整数规划求解状态。
SimplexIter	整数属性	单纯形法迭代循环数。
BarrierIter	整数属性	内点法迭代循环数。
NodeCnt	整数属性	分支定界搜索的节点数。
PoolSols	整数属性	解池中的解的数目。
TuneResults	整数属性	参数调优结果的数目。
HasLpSol	整数属性	是否可以提供线性规划的解值。
HasBasis	整数属性	是否可以提供线性规划的基。
HasDualFarkas	整数属性	当线性规划问题无可行解时，是否返回对偶 Farkas（也叫做对偶极射线）。
HasPrimalRay	整数属性	当线性规划问题无界时，是否返回主元射线（也叫做极射线）。
HasMipSol	整数属性	是否存在整数解。
IISCols	整数属性	组成 IIS 的变量边界的数目。
IISRows	整数属性	组成 IIS 的约束的数目。
IISSOSs	整数属性	组成 IIS 的 SOS 约束的数目。
IISIndicators	整数属性	组成 IIS 的 Indicator 约束的数目。
HasIIS	整数属性	是否存在 IIS。
HasFeasRelaxSol	整数属性	是否存在可行化松弛结果。
IsMinIIS	整数属性	计算出的 IIS 是否为极小。
LpObjval	浮点数属性	线性规划目标函数值。
BestObj	浮点数属性	整数规划求解结束时最好的目标函数值。
BestBnd	浮点数属性	整数规划求解结束时最好的下界。
BestGap	浮点数属性	整数规划求解结束时最好的相对容差。
FeasRelaxObj	浮点数属性	可行化松弛值。
SolvingTime	浮点数属性	求解所使用的时间（秒）。

- `LpStatus`

  整数属性。

  线性规划求解状态。

  可取值： 请参考一般常数章节：解状态

- `MipStatus`

  整数属性。

  整数规划求解状态。

  可取值： 请参考一般常数章节：解状态

- SimplexIter

  整数属性。

  单纯形法迭代循环数。

- BarrierIter

  整数属性。

  内点法迭代循环数。

- NodeCnt

  整数属性。

  分支定界搜索的节点数。

- PoolSols

  整数属性。

  解池中的解的数目。

- TuneResults

  整数属性。

  参数调优结果的数目。

- HasLpSol

  整数属性。

  是否可以提供线性规划的解值。

- HasBasis

  整数属性。

  是否可以提供线性规划的基。

- HasDualFarkas

  整数属性。

  当线性规划问题无可行解时，是否返回对偶 Farkas（也叫做对偶极射线）。

- HasPrimalRay

  整数属性。

  当线性规划问题无界时，是否返回主元射线（也叫做极射线）。

- HasMipSol

  整数属性。

  是否存在整数解。

- IISCols

整数属性。

组成 IIS 的变量边界的数目。

- `IISRows`

  整数属性。

  组成 IIS 的约束的数目。

- `IISSOSs`

  整数属性。

  组成 IIS 的 SOS 约束的数目。

- `IISIndicators`

  整数属性。

  组成 IIS 的 Indicator 约束的数目。

- `HasIIS`

  整数属性。

  是否存在 IIS。

- `HasFeasRelaxSol`

  整数属性。

  是否存在可行化松弛结果。

- `IsMinIIS`

  整数属性。

  计算出的 IIS 是否为极小。

- `LpObjval`

  浮点数属性。

  线性规划目标函数值。

- `BestObj`

  浮点数属性。

  整数规划求解结束时最好的目标函数值。

- `BestBnd`

  浮点数属性。

  整数规划求解结束时最好的下界。

- `BestGap`

  浮点数属性。

  整数规划求解结束时最好的相对容差。

- `FeasRelaxObj`

  浮点数属性。

  可行化松弛值。

- `SolvingTime`

  浮点数属性。

  求解所使用的时间（秒）。

9.3 属性获取方式

在不同的编程接口中，获取属性的方式略有差别，具体请参考：

- C 接口：*C API 函数：获取属性章节*
- C++ 接口：*C++ API 参考手册：属性章节*
- C# 接口：*C# API 参考手册：属性章节*
- Java 接口：*Java API 参考手册：属性章节*
- Python 接口：*Python API 参考手册：属性章节*

第 10 章　信息

信息常数描述了模型构成要素（目标函数、约束条件和决策变量）、求解结果以及可行化松弛计算结果等的相关信息。本章节将介绍杉数求解器 COPT 提供的信息常数及其含义。章节内容构成如下：

- 优化模型相关信息
- 求解结果相关信息
- 对偶 *Farkas* 和主元射线
- 可行化松弛结果相关信息
- *Callback* 相关信息
- 信息获取方式

表 10.1: COPT 信息常数总览

信息名	类型	信息含义
Obj	浮点数信息	变量（列）的目标函数系数
LB	浮点数信息	变量（列）或者约束（行）的下界
UB	浮点数信息	变量（列）或者约束（行）的上界
Value	浮点数信息	变量（列）的取值
Slack	浮点数信息	松弛变量的取值，也叫做约束的活跃程度（activities），仅适用于线性规划模型
Dual	浮点数信息	对偶变量的取值，仅适用于线性规划模型
RedCost	浮点数信息	变量的 Reduced cost，仅适用于线性规划模型
DualFarkas	浮点数信息	对偶 Farkas（也叫做对偶极射线），适用于线性规划模型，无可行解或无界情况
PrimalRay	浮点数信息	主元射线（也叫做极射线），适用于线性规划模型，无可行解或无界情况
BestObj	浮点数信息	当前最优目标函数值
BestBnd	浮点数信息	当前最优目标下界
HasIncumbent	整数信息	当前是否有最优可行解
MipCandObj	浮点数信息	当前可行解对应的目标函数值
RelaxSolObj	浮点数信息	当前 LP 松弛问题的目标函数值
RelaxLB	浮点数信息	变量（列）或者约束（行）下界的可行化松弛量
RelaxUB	浮点数信息	变量（列）或者约束（行）上界的可行化松弛量
RelaxValue	浮点数信息	可行化松弛模型中原始模型变量（列）的解
NodeStatus	整数信息	当前节点 LP 松弛问题的求解状态

注意： Double 这个词的准确翻译应是双精度浮点数。表格中和下文为了简便，称之为浮点数。

10.1 模型相关信息

- `Obj`

  浮点数信息。

  变量（列）的目标函数系数。

- `LB`

  浮点数信息。

  变量（列）或者约束（行）的下界。

- `UB`

  浮点数信息。

  变量（列）或者约束（行）的上界。

10.2 求解结果相关信息

- `Value`

  浮点数信息。

  变量（列）的取值。

- `Slack`

  浮点数信息。

  松弛变量的取值，也叫做约束的活跃程度（activities）。仅适用于线性规划模型。

- `Dual`

  浮点数信息。

  对偶变量的取值。仅适用于线性规划模型。

- `RedCost`

  浮点数信息。

  变量的 Reduced cost。仅适用于线性规划模型。

10.3 对偶 Farkas 和主元射线

进阶话题。

当线性规划问题无可行解或者无界时，求解器可以返回对偶 Farkas（也叫做对偶极射线）或者主元射线（也叫做极射线）作为证明。

- `DualFarkas`

  浮点数信息。

  线性规划问题无可行解时，线性约束的对偶 Farkas（也叫做对偶极射线）。请设置 `"ReqFarkasRay"` 这一参数，以确保求解器可以返回对偶 Farkas。

  对偶 Farkas 的作用可以用形如 $Ax = 0$ and $l \leq x \leq u$ 的线性规划约束解释。当该线性规划无可行解时，使用对偶 Farkas 向量 y 可以证明线性约束系统存在冲突：$\max y^T Ax < y^T b = 0$。如何计算 $\max y^T Ax$：使用向量 $\hat{a} = y^T A$，当 $\hat{a}_i < 0$ 时选择 $x_i = l_i$ 或者 $\hat{a}_i > 0$ 时选择 $x_i = u_i$，可以计算出表达式 $y^T Ax$ 的最大可能取值。

  有些应用依赖于另一种等价的线性系统冲突证明：$\min \bar{y}^T Ax > \bar{y}^T b = 0$。对于此种情况，可以对求解器返回的对偶 Farkas 取负值实现，即 $\bar{y} = -y$。

  在极端情况下，求解器可能无法返回有效的对偶 Farkas。例如当线性规划问题的不可行性微乎其微时。此时，我们建议用 FeasRelax 功能研究或者修复线性规划的不可行性。

- `PrimalRay`

  浮点数信息。

  线性规划问题无界时，变量的主元射线（也叫做极射线）。请设置 `"ReqFarkasRay"` 这一参数，以确保求解器可以返回主元射线。

  对于一个求解最小值的线性规划问题 $\min c^T x, Ax = b$ and $x \geq 0$，主元射线向量 r 满足以下条件：$r \geq 0, Ar = 0$ 以及 $c^T r < 0$。

10.4 可行化松弛结果相关信息

- `RelaxLB`

  浮点数信息。

  变量（列）或者约束（行）下界的可行化松弛量。

- `RelaxUB`

  浮点数信息。

  变量（列）或者约束（行）上界的可行化松弛量。

- `RelaxValue`

  浮点数信息。

  可行化松弛模型中原始模型变量（列）的解。

10.5 Callback 相关信息

Callback 相关信息包括可以在回调函数中获取到的信息。

- `BestObj`

  浮点数信息。

  当前最优目标函数值。

- `BestBnd`

  浮点数信息。

  当前最优目标下界。

- `HasIncumbent`

  整数信息。

  当前是否有最优可行解。

- `MipCandObj`

  浮点数信息。

  当前可行解对应的目标函数值。

- `RelaxSolObj`

  浮点数信息。

  当前 LP 松弛问题的目标函数值。

- `NodeStatus`

  整数信息。

  当前节点 LP 松弛问题的求解状态。可取值请参考：*一般常数章节：解状态（部分）*，除去 `NODELIMIT`, `UNSTARTED`, `INF_OR_UNB`，其他均为其可能取值。

10.6 信息获取方式

在不同的编程接口中，获取信息的方式略有差别，具体请参考：

- C 接口：*C API 函数：获取模型信息章节*
- C# 接口：*C# API 参考手册：信息章节*
- Java 接口：*Java API 参考手册：信息章节*
- Python 接口：*Python API 参考手册：信息章节*

第 11 章　参数

优化参数控制 COPT 求解器优化算法的行为。每个参数都有各自的默认值和取值范围，在开始求解前，用户可以通过设置参数为不同取值，对求解算法、求解过程等提出具体要求，当然也可以保持为默认设置。

按照求解器 COPT 执行的任务、求解的优化问题，可以分为不同类型的参数。本章节将介绍杉数求解器 COPT 提供的不同类型参数及其含义。章节内容构成如下：

- 限制和容差 ，预求解相关
- 线性规划相关 ，整数规划相关，半定规划相关
- *IIS* 计算相关 ，可行化松弛计算相关
- 并行计算相关，参数调优相关，*Callback* 相关，其他参数
- 参数相关操作

11.1 限制和容差

限制类参数用来控制求解过程的终止，如果求解算法超过了限制类参数所设定的值（如求解时间、节点数、迭代数等），COPT 会终止求解并返回一个非最优的求解状态（关于求解状态，请参考常数章节：解状态）

表 11.1: 限制类参数总览

参数名	类型	参数含义
TimeLimit	浮点数参数	优化求解的时间限制（秒）
SolTimeLimit	浮点数参数	找到原始可行解的时间限制（秒）
NodeLimit	整数参数	整数规划求解的节点数限制
BarIterLimit	整数参数	内点法求解的迭代数限制

容差类参数对求解器所允许的可行性和最优性的冲突值（violations）提出要求。

注意

Tolerance 有时会翻译为公差或者容差。本文档采用容差这个译法。

Double 这个词的准确翻译应该是双精度浮点数。表格中和下文为了简便，称之为浮点数。

表 11.2: 容差类参数总览

参数名	类型	参数含义
MatrixTol	浮点数参数	输入矩阵的系数容差
FeasTol	浮点数参数	变量、约束取值的可行性容差
DualTol	浮点数参数	对偶解的可行性容差
IntTol	浮点数参数	变量的整数解容差
RelGap	浮点数参数	整数规划的最优相对容差
AbsGap	浮点数参数	整数规划的最优绝对容差

- `TimeLimit`

  浮点数参数。

  优化求解的时间限制（秒）。

  默认值 1e20

  最小值 0

  最大值 1e20

- `SolTimeLimit`

  浮点数参数。

  找到原始可行解的时间限制（秒）。

  默认值 1e20

  最小值 0

  最大值 1e20

- `NodeLimit`

  整数参数。

  整数规划求解的节点数限制。

  默认值 -1（自动选择）

  最小值 -1（自动选择）

  最大值 `INT_MAX`

- `BarIterLimit`

  整数参数。

  内点法求解时的迭代数限制。

  默认值 500

  最小值 0

  最大值 `INT_MAX`

- `MatrixTol`

浮点数参数。

输入矩阵的系数容差。

默认值 1e-10

最小值 0

最大值 1e-7

- `FeasTol`

  浮点数参数。

  变量、约束取值的可行性容差。

  默认值 1e-6

  最小值 1e-9

  最大值 1e-4

- `DualTol`

  浮点数参数。

  对偶解的可行性容差。

  默认值 1e-6

  最小值 1e-9

  最大值 1e-4

- `IntTol`

  浮点参数。

  变量的整数解容差。

  默认值 1e-6

  最小值 1e-9

  最大值 1e-1

- `RelGap`

  浮点参数。

  整数规划的最优相对容差。

  默认值 1e-4

  最小值 0

  最大值 `DBL_MAX`

- `AbsGap`

  浮点参数。

  整数规划的最优绝对容差。

默认值 1e-6

最小值 0

最大值 DBL_MAX

11.2 预求解相关

预求解相关参数控制求解器的预求解算法

表 11.3: 预求解相关参数总览

参数名	类型	参数含义
Presolve	整数参数	预求解的强度
Scaling	整数参数	是否在求解一个模型前，调整系数矩阵的数值（Scaling）
Dualize	整数参数	是否构建并求解对偶模型

- `Presolve`

  整数参数。

  预求解的强度。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 关闭。

  1: 少量快速。

  2: 正常。

  3: 多多益善。

- `Scaling`

  整数参数。

  是否在求解一个模型前，调整系数矩阵的数值（Scaling）。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 不调整。

  1: 调整系数矩阵的数值。

- `Dualize`

  整数参数。

  是否构建并求解对偶模型。

默认值 -1

可选值

-1: 自动选择。

0: 不构建对偶模型。

1: 构建对偶模型。

11.3 线性规划相关

线性规划相关参数控制单纯形法和内点法的算法工作流程。

表 11.4: 线性规划规划相关参数总览

参数名	类型	参数含义
LpMethod	整数参数	求解线性规划问题的算法
DualPrice	整数参数	选定对偶单纯形法的 Pricing 算法
DualPerturb	整数参数	是否允许对偶单纯性算法使用目标函数摄动
BarHomogeneous	整数参数	是否使用齐次自对偶方法
BarOrder	整数参数	内点法的矩阵排列算法
BarStart	整数参数	内点法寻找初始点的算法
Crossover	整数参数	是否使用 Crossover
ReqFarkasRay	整数参数	当线性规划问题无可行解或者无界时，是否计算对偶 Farkas（也称对偶极射线）或主元射线（也称极射线）

- `LpMethod`

  整数参数。

  求解线性规划问题的算法。

  默认值 -1

  可选值

  -1: 自动选择。

  对于线性规划问题，选择对偶单纯形法；

  对于混合整数线性规划问题，选择对偶单纯形法或内点法之一。

  1: 对偶单纯形法。

  2: 内点法。

  3: 直接 Crossover。

  4: 并发求解（同时启动单纯形法与内点法求解）。

  5: 基于稀疏和数值范围等特征自动选择单纯形法或者内点法。

- `DualPrice`

整数参数。

选定对偶单纯形法的 Pricing 算法。

默认值 -1

可选值

-1: 自动选择。

0: 使用 Devex 算法。

1: 使用对偶最陡边算法。

- `DualPerturb`

  整数参数。

  是否允许对偶单纯性算法使用目标函数摄动。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 无摄动。

  1: 允许目标函数摄动。

- `BarHomogeneous`

  整数参数。

  是否使用齐次自对偶方法。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 不使用。

  1: 使用。

- `BarOrder`

  整数参数。

  内点法的矩阵排列算法。

  默认值 -1

  可选值

  -1: 自动选择。

  0: Approximate Minimum Degree（AMD）算法。

  1: Nested Dissection（ND）算法。

- `BarStart`

整数参数。

内点法寻找初始点的算法。

默认值 -1

可选值

-1: 自动选择。

0: Simple 算法。

1: Mehrotra 算法。

2: Modified Mehrotra 算法。

- `Crossover`

  整数参数。

  是否使用 Crossover。

  默认值 1

  可选值

  -1: 自动选择。仅在当线性规划的解不满足容差时使用。

  0: 不使用。

  1: 使用。

- `ReqFarkasRay`

  整数参数。

  进阶话题。当线性规划问题无可行解或者无界时，是否计算对偶 Farkas（也叫做对偶极射线）或者主元射线（也叫做极射线）。

  默认值 0

  可选值

  0: 不计算。

  1: 计算。

11.4 整数规划相关

整数规划相关参数控制整数规划求解算法的工作流程。

表 11.5: 整数规划相关参数总览

参数名	类型	参数含义
CutLevel	整数参数	生成割平面的强度
RootCutLevel	整数参数	根节点生成割平面的强度
TreeCutLevel	整数参数	搜索树生成割平面的强度
RootCutRounds	整数参数	根节点生成割平面的次数
NodeCutRounds	整数参数	搜索树节点生成割平面的次数
HeurLevel	整数参数	启发式算法的强度
RoundingHeurLevel	整数参数	Rounding 启发式算法的强度
DivingHeurLevel	整数参数	Diving 启发式算法的强度
SubMipHeurLevel	整数参数	基于子 MIP 的启发式算法的强度
FAPHeurLevel	整数参数	Fix-and-propagate 启发式算法的强度
StrongBranching	整数参数	Strong Branching 的强度
ConflictAnalysis	整数参数	是否使用冲突分析
MipStartMode	整数参数	处理初始解的方式
MipStartNodeLimit	整数参数	补全不完整的初始解时，求解的子 MIP 的节点数限制

- `CutLevel`

  整数参数。

  生成割平面的强度。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 关闭。

  1: 少量快速。

  2: 正常。

  3: 多多益善。

- `RootCutLevel`

  整数参数。

  根节点生成割平面的强度。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 关闭。

  1: 少量快速。

  2: 正常。

3: 多多益善。

- TreeCutLevel

  整数参数。

  搜索树生成割平面的强度。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 关闭。

  1: 少量快速。

  2: 正常。

  3: 多多益善。

- RootCutRounds

  整数参数。

  根节点生成割平面的次数。

  默认值 -1（自动选择）

  最小值 -1（自动选择）

  最大值 INT_MAX

- NodeCutRounds

  整数参数。

  搜索树节点生成割平面的次数。

  默认值 -1（自动选择）

  最小值 -1（自动选择）

  最大值 INT_MAX

- HeurLevel

  整数参数。

  启发式算法的强度。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 关闭。

  1: 少量快速。

  2: 正常。

3: 多多益善。

- `RoundingHeurLevel`

  整数参数。

  Rounding 启发式算法的强度。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 关闭。

  1: 少量快速。

  2: 正常。

  3: 多多益善。

- `DivingHeurLevel`

  整数参数。

  Diving 启发式算法的强度。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 关闭。

  1: 少量快速。

  2: 正常。

  3: 多多益善。

- `SubMipHeurLevel`

  整数参数。

  基于子 MIP 的启发式算法的强度。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 关闭。

  1: 少量快速。

  2: 正常。

  3: 多多益善。

- `FAPHeurLevel`

整数参数。

Fix-and-propagate 启发式算法的强度。

默认值 -1

可选值

-1: 自动选择。

0: 关闭。

1: 少量快速。

2: 正常。

3: 多多益善。

- `StrongBranching`

  整数参数。

  Strong Branching 的强度。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 关闭。

  1: 少量快速。

  2: 正常。

  3: 多多益善。

- `ConflictAnalysis`

  整数参数。

  是否使用冲突分析。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 不使用。

  1: 使用。

- `MipStartMode`

  整数参数。

  处理初始解的方式。

  默认值 -1

  可选值

-1: 自动选择。

0: 不使用任何初始解。

1: 仅使用完整且可行的初始解。

2: 仅使用可行的初始解（若初始解不完整，通过求解子 MIP 来补全）。

- `MipStartNodeLimit`

  整数参数。

  补全不完整的初始解时，求解的子 MIP 的节点数限制。

  默认值 -1（自动选择）

  最小值 -1（自动选择）

  最大值 `INT_MAX`

11.5 半定规划相关

- `SDPMethod`

  整数参数。

  求解半定规划问题的算法。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 原始-对偶内点法。

  1: 交替方向乘子法。

  2: 对偶内点法。

11.6 并行计算相关

并行计算相关参数用来控制求解算法并行计算的线程数和任务数。

表 11.6: 并行计算相关参数总览

参数名	类型	参数含义
Threads	整数参数	问题求解使用的线程数
BarThreads	整数参数	内点法使用的线程数。若值为-1，则线程数由参数 `Threads` 决定
SimplexThreads	整数参数	对偶单纯形法使用的线程数。若值为-1，则线程数由参数 `Threads` 决定
CrossoverThreads	整数参数	Crossover 使用的线程数。若值为-1，则线程数由参数 `Threads` 决定
MipTasks	整数参数	MIP 求解使用的任务数

- `Threads`

  整数参数。

  问题求解使用的线程数。

  默认值 -1（自动选择）

  最小值 -1（自动选择）

  最大值 128

- `BarThreads`

  整数参数。

  内点法使用的线程数。若值为-1，则线程数由参数 `Threads` 决定。

  默认值 -1

  最小值 -1

  最大值 128

- `SimplexThreads`

  整数参数。

  对偶单纯形法使用的线程数。若值为-1，则线程数由参数 `Threads` 决定。

  默认值 -1

  最小值 -1

  最大值 128

- `CrossoverThreads`

  整数参数。

  Crossover 使用的线程数。若值为-1，则线程数由参数 `Threads` 决定。

  默认值 -1

  最小值 -1

  最大值 128

- `MipTasks`

  整数参数。

  MIP 求解使用的任务数。

  默认值 -1（自动选择）

  最小值 -1（自动选择）

  最大值 256

11.7 IIS 计算相关

- `IISMethod`

  整数参数。

  计算 IIS 的方法。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 计算结果质量优先。

  1: 计算效率优先。

11.8 可行化松弛计算相关

- `FeasRelaxMode`

  整数参数。

  计算可行化松弛的方法。

  默认值 0

  可选值

  0: 最小化加权冲突值。

  1: 计算最小化加权冲突下的原始模型最优可行化松弛。

  2: 最小化冲突数目。

  3: 计算最小化冲突数目下的原始模型最优可行化松弛。

  4: 最小化加权平方冲突值。

  5: 计算最小化加权平方冲突下的原始模型最优可行化松弛。

11.9 参数调优相关

表 11.7: 参数调优相关参数总览

参数名	类型	参数含义
TuneTimeLimit	浮点数参数	参数调优的时间限制
TuneTargetTime	浮点数参数	参数调优的时间目标
TuneTargetRelGap	整数参数	参数调优的最优相对容差目标
TuneMethod	整数参数	参数调优的方法
TuneMode	整数参数	参数调优的模式
TuneMeasure	整数参数	参数调优结果计算方式
TunePermutes	整数参数	参数调优每组参数模型计算次数
TuneOutputLevel	整数参数	参数调优日志输出强度

- `TuneTimeLimit`

  浮点数参数。

  参数调优的时间限制。若值为 0 ，则表示求解器自动设置。

  默认值 0

  最小值 0

  最大值 1e20

- `TuneTargetTime`

  浮点数参数。

  参数调优的时间目标。

  默认值 1e-2

  最小值 0

  最大值 `DBL_MAX`

- `TuneTargetRelGap`

  浮点数参数。

  参数调优的最优相对容差目标。

  默认值 1e-4

  最小值 0

  最大值 `DBL_MAX`

- `TuneMethod`

  整数参数。

  参数调优的方法。

  默认值 -1

可选值

-1: 自动选择。

0: 贪婪搜索策略。

1: 更广泛的搜索策略。

- TuneMode

  整数参数。

  参数调优的模式。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 求解时间。

  1: 最优相对容差。

  2: 目标函数值。

  3: 目标函数值下界。

- TuneMeasure

  整数参数。

  参数调优结果计算方式。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 计算平均值。

  1: 计算最大值。

- TunePermutes

  整数参数。

  参数调优每组参数模型计算次数。若值为 0 ，则表示求解器自动设置。

  默认值 0

  最小值 0

  最大值 INT_MAX

- TuneOutputLevel

  整数参数。

  参数调优日志输出强度。

  默认值 2

可选值

0: 不输出调优日志。

1: 仅输出改进参数的摘要。

2: 输出每次调优尝试的摘要。

3: 输出每次调优尝试的详细日志。

11.10 Callback 相关

- `LazyConstraints`

  整数参数。

  是否将惰性约束加入模型中。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 否。

  1: 是。

注意：

- 该参数仅对 MIP 模型有效。

11.11 其它参数

- `Logging`

  整数参数。

  是否显示求解日志。

  默认值 1

  可选值

  0: 不显示求解日志。

  1: 显示求解日志。

- `LogToConsole`

  整数参数。

  是否显示求解日志到控制台。

  默认值 1

可选值

0: 不显示求解日志到控制台。

1: 显示求解日志到控制台。

11.12 参数相关操作

在不同的编程接口中，获取和设置优化参数的方式略有差别，具体请参考：

- C 接口：*C API 函数：获取和设置参数章节*
- C++ 接口：*C++ API 参考手册：参数章节*
- C# 接口：*C# API 参考手册：参数章节*
- Java 接口：*Java API 参考手册：参数章节*
- Python 接口：*Python API 参考手册：参数章节*

第 12 章　不同类型优化问题建模求解

本章将会介绍如何在杉数求解器 COPT 中，对不同类型的优化问题进行建模和求解。章节内容构成如下：

- 线性规划（*LP*）
- 二阶锥规划（*SOCP*）
- 半定规划（*SDP*）
- 二次规划（*QP*）
- 二次约束规划（*QCP*）
- 混合整数规划（*MIP*）
- 特殊约束

杉数求解器 COPT 目前支持建模和求解的问题类型与相应求解算法如 表 12.1 所示：

表 12.1: COPT 支持建模求解的问题类型与求解算法

问题类型	**求解算法**
线性规划 (LP)	对偶单纯形法、内点法
二阶锥规划 (SOCP)	内点法
凸二次规划 (QP)	内点法
凸二次约束规划 (QCP)	内点法
半定规划 (SDP)	内点法、交替乘子下降法
混合整数线性规划 (MILP)	分支切割法
混合整数二阶锥规划 (MISOCP)	分支切割法
混合整数凸二次规划 (MIQP)	分支切割法
混合整数凸二次约束规划 (MIQCP)	分支切割法

杉数求解器 COPT 支持优化问题的 3 种输入方式：文件输入、编程接口建模、第三方建模工具包；

1. 模型文件输入，请参考*COPT 支持的文件格式* 。
2. COPT 支持的编程语言接口有：
    - C 接口
    - C++ 接口
    - C# 接口
    - Java 接口

- Python 接口
- Fortran 接口

3. COPT 目前支持的第三方建模工具接口有：
   - AMPL
   - GAMS
   - Julia
   - Pyomo
   - PuLP
   - CVXPY

12.1 线性规划（LP）

线性规划（Linear Programming，简称 LP），作为运筹学中最基本和最重要的分支，有着广泛的应用场景。

线性规划问题的目标函数和约束条件都是线性的。

12.1.1 数学模型

数学形式如下：

$$\begin{aligned} \min \quad & \sum_{i=0}^{m-1} c_j x_j + c^f \\ \text{s.t.} \quad & l_i^c \le \sum_{j=0}^{n-1} a_{ij} x_j \le u_i^c, \qquad i = 1, 2, \cdots, m-1 \\ & l_j^v \le x_j \le u_j^v, \qquad j = 1, 2, \cdots, n-1 \end{aligned} \tag{12.1}$$

更简洁地，也可以用向量和矩阵进行表示：

$$\begin{aligned} \min \quad & c^T x + c^f \\ \text{s.t.} \quad & l^c \le Ax \le u^c \\ & l^v \le x \le u^v \end{aligned} \tag{12.2}$$

模型中的变量和参数含义如下：

- 问题规模：m 表示约束条件的个数，n 表示决策变量的个数
- 决策变量：$x = (x_j)_{j=0}^{n-1} \in \mathbb{R}^n$
- 变量取值范围：$l^v, u^v \in \mathbb{R}^n$，其中 l^v 表示变量下界，u^v 表示变量上界
- 约束边界：$l^c, u^c \in \mathbb{R}^m$，其中 u^c 表示约束下界，u^c 表示约束上界
- 线性约束的系数矩阵：$A = (a_{ij})_{m \times n} \in \mathbb{R}^{m \times n}$
- 目标函数中变量系数：$c \in \mathbb{R}^n$ 表示目标函数中变量的系数，c^f 表示目标函数中的常数项

12.1.2 建模

在 COPT 中构建线性规划（LP）模型并求解的基本步骤如下：

1. 创建 COPT 求解环境和求解模型
2. 添加模型参数
3. 构建线性规划模型：
   - 添加决策变量
   - 添加线性约束
   - 设置线性目标函数
4. 设置求解参数并求解
5. 获取求解结果

建模：添加线性约束

杉数求解器 COPT 提供添加线性约束的三种方式：

1. 向模型中添加单条线性约束
2. 批量添加一组线性约束
3. 添加单条带上下界的线性约束

在向模型中添加线性约束时，用户可以指定的参数主要有：

- `expr/builder`：线性约束表达式或线性约束构建器
- `sense`：约束类型，可取值请参考常数章节：约束类型
- `name`：线性约束名称

如果是添加带上下界的线性约束，还需指定：

- `lb`：线性约束下界
- `ub`：线性约束上界

在不同编程接口中实现方式如 表 12.2 所示：

表 12.2: 添加线性约束的函数

编程接口	添加单条约束	批量添加一组约束
C	`COPT_AddRow`	`COPT_AddRows`
C++	`Model::AddConstr()`	`Model::AddConstrs()`
C#	`Model.AddConstr()`	`Model.AddConstrs()`
Java	`Model.addConstr()`	`Model.addConstrs()`
Python	`Model.addConstr()`	`Model.addConstrs()`

注意

- 关于线性约束的相关操作，在不同编程接口中的函数名称、调用方式，以及 **参数名称略有不同**，但是实现的功能和参数含义是一致的；
- COPT 支持使用约束类型来定义约束，但我们 **推荐直接使用上下边界来定义约束**；
- 在 C API 中，以系数矩阵作为参数输入的方式添加线性约束；
- 在 Python API 中，另外提供添加带上下界线性约束的成员方法：`Model.addBoundConstr()` 。

线性规划问题示例

$$
\begin{aligned}
&\text{最大化：} \\
&\qquad 1.2x + 1.8y + 2.1z \\
&\text{约束：} \\
&\qquad 1.5x + 1.2y + 1.8z \leq 2.6 \\
&\qquad 0.8x + 0.6y + 0.9z \geq 1.2 \\
&\text{变量范围：} \\
&\qquad 0.1 \leq x \leq 0.6 \\
&\qquad 0.2 \leq y \leq 1.5 \\
&\qquad 0.3 \leq z \leq 2.8
\end{aligned}
\tag{12.3}
$$

不同编程接口中对应代码实现请参考：*COPT 快速入门章节*

在 COPT 提供的编程接口中，除 C 语言外，其余面向对象的编程接口（C#, C++, Java, Python）均提供线性约束相关的类：

- 线性约束相关操作的封装
  1. `Constraint` 类：杉数求解器 COPT 中线性约束相关操作的封装；
  2. `ConstrArray` 类：方便用户对一组 `Constraint` 类对象进行操作。
- 线性约束构建器的封装
  1. `ConstrBuilder` 类：杉数求解器 COPT 中构建线性约束构建器的封装；
  2. `ConstrBuilderArray` 类：为方便用户对一组 `ConstrBuilder` 类对象进行操作。
- C++ API：*Constraint 类* ， *ConstrArray 类* ， *ConstrBuilder 类* ， *ConstrBuilderArray 类*
- C# API：*Constraint 类* ， *ConstrArray 类* ， *ConstrBuilder 类* ， *ConstrBuilderArray 类*
- Java API：*Constraint 类* ， *ConstrArray 类* ， *ConstrBuilder 类* ， *ConstrBuilderArray 类*
- Python API：*Constraint 类* ， *ConstrArray 类* ， *ConstrBuilder 类* ， *ConstrBuilderArray 类*

12.1.3 求解

对于线性规划问题，杉数求解器 COPT 提供的求解算法有：单纯形法和内点法，通过设置优化参数 "LpMethod" 可以进行指定。通过设置其他线性规划的相关优化参数，可以控制求解算法的具体工作流程，详细请参考参数章节：线性规划相关 。

关于线性规划问题的求解日志请参考求解日志章节：单纯形法 和内点法 。

12.1.4 相关属性和信息

线性规划相关属性

线性规划问题模型描述相关的属性如 表 12.3 所示：

表 12.3: 线性规划模型描述相关属性

属性名	类型	属性含义
Cols	整数属性	变量（系数矩阵列）个数
Rows	整数属性	线性约束（系数矩阵行）数目
Elems	整数属性	系数矩阵中非零元素个数
LpObjval	浮点数属性	内点法求解的迭代数限制
SimplexIter	整数属性	单纯形法迭代循环数
BarrierIter	整数属性	内点法迭代循环数

线性规划问题求解结果相关的属性如 表 12.4 所示：

表 12.4: 线性规划求解结果相关属性

属性名	类型	属性含义
LpStatus	整数属性	线性规划求解状态
HasLpSol	整数属性	是否可以提供线性规划的解值
HasBasis	整数属性	是否可以提供线性规划的基
LpObjval	浮点数属性	内点法求解的迭代数限制
SimplexIter	整数属性	单纯形法迭代循环数
BarrierIter	整数属性	内点法迭代循环数

对于线性规划问题的求解结果，COPT 还提供相关信息常数，以获取对偶问题的相关信息，如 表 12.5 所示：

表 12.5: 线性规划对偶问题相关信息

信息名	类型	信息含义
Slack	浮点数信息	松弛变量的取值，也叫做约束的活跃程度（activities）
Dual	浮点数信息	对偶变量的取值
RedCost	浮点数信息	变量的 Reduced Cost

在不同编程接口中，属性和信息的获取方式，请参考属性、信息 章节。

12.2 二阶锥规划（SOCP）

二阶锥规划（Second-Order Cone Programming，简称 SOCP），目标函数是线性函数，约束条件是二阶锥约束和线性约束。

12.2.1 数学模型

二阶锥 (SOC)：

$$\mathcal{Q}^{n+1} = \{(t, x) \in \mathbb{R} \times \mathbb{R}^n \mid t \geq \|x\|_2\} \tag{12.4}$$

二阶锥约束：

当 $t \in \mathbb{R}$和$x \in \mathbb{R}^n$ 为决策变量时，形如 $t \geq \|x\|_2$ 称为二阶锥约束。

数学形式如下：

$$\begin{aligned} \min \quad & c^T x + c^f \\ \text{s.t.} \quad & l^c \leq Ax \leq u^c \\ & l^v \leq x \leq u^v \\ & Fx + g \in \mathcal{Q} \end{aligned} \tag{12.5}$$

模型中的变量和参数含义如下：

- 决策变量：$x = (x_j)_{j=0}^{n-1} \in \mathbb{R}^n$
- 决策变量取值范围：$l^v, u^v \in \mathbb{R}^n$，其中 l^v 表示变量下界，u^v 表示变量上界
- 约束边界：$l^c, u^c \in \mathbb{R}^m$，其中 u^c 表示线性约束下界，u^c 表示线性约束上界
- 线性约束的系数矩阵：$A = (a_{ij})_{m \times n} \in \mathbb{R}^{m \times n}$
- 目标函数中的系数：$c \in \mathbb{R}^n$ 表示目标函数中变量的系数，c^f 表示目标函数中的常数项
- 问题规模：m 表示线性约束条件的个数，n 表示决策变量的个数，k 表示二阶锥约束的个数

其中，锥约束相关参数含义如下：

- F：$F \in \mathbb{R}^{k \times n}$ 锥约束的系数矩阵
- g：$g \in \mathbb{R}^k$ 锥约束的常数向量
- $\mathcal{Q}$：表示 k 个集合的笛卡尔积，$\mathcal{Q} = \mathcal{Q}_1 \times \mathcal{Q}_2 \times \cdots \mathcal{Q}_p$，其中 p 表示二阶锥约束的个数，$\mathcal{Q}_i, i \in \{1, 2, \cdots, p\}$ 都表示二阶锥。

12.2.2 建模

在 COPT 中构建二阶锥规划（SOCP）模型并求解的基本步骤如下：

1. 创建 COPT 求解环境和求解模型
2. 添加模型参数
3. 构建二阶锥规划模型

- 添加决策变量
- 添加约束条件（二阶锥约束、线性约束）
- 设置目标函数

4. 设置求解参数并求解
5. 获取求解结果

建模：添加二阶锥约束

杉数求解器 COPT 支持对以下两种类型的二阶锥约束进行建模：

标准二阶锥

$$Q^n = \left\{ x \in \mathbb{R}^n \;\middle|\; x_0 \geq \sqrt{\sum_{i=1}^{n-1} x_i^2}, x_0 \geq 0 \right\} \tag{12.6}$$

常数表示：`CONE_QUAD`

旋转二阶锥

$$Q_r^n = \left\{ x \in \mathbb{R}^n \;\middle|\; 2x_0x_1 \geq \sum_{i=2}^{n-1} x_i^2, x_0 \geq 0, x_1 \geq 0 \right\} \tag{12.7}$$

常数表示：`CONE_RQUAD`

在向模型中添加二阶锥约束时，用户可以指定的参数主要有：

- `ctype`：二阶锥约束的类型
  - `CONE_QUAD`：标准二阶锥
  - `CONE_RQUAD`：旋转二阶锥
- `cvars`：参与构成二阶锥约束的变量
- `dim`：二阶锥约束的维度

在不同编程接口中实现方式如 表 12.6 所示：

表 12.6: 添加二阶锥约束的函数

编程接口	函数
C	`COPT_AddCones`
C++	`Model::AddCone()`
C#	`Model.addCone()`
Java	`Model.addCone()`
Python	`Model.addCone()`

注意

- 关于二阶锥约束建模的相关操作，在不同编程接口中的函数名称、调用方式，以及 **参数名称略有不同**，但是实现的功能和参数含义是一致的；
- 在提供参与构成二阶锥约束变量时，请按照变量在约束表达式中的顺序依次输入；

- 在 C API 中，以行压缩存储格式提供二阶锥约束的成员，关于 C API 中稀疏矩阵的压缩存储，请参考具体示例 ；
- 在 Python API 中，对于指定二阶锥维度的方式，另外提供成员方法：`Model.addConeByDim()` 。

二阶锥约束示例

由 x_4, x_1, x_2, x_3 构成的标准二阶锥：

$$x_4 \geq \sqrt{x_1^2 + x_2^2 + x_3^2} \tag{12.8}$$

由 x_3, x_4, x_1, x_2 构成的旋转二阶锥：

$$2x_3x_4 \geq x_1^2 + x_2^2 \tag{12.9}$$

以 x_4, x_1, x_2, x_3 构成的标准二阶锥为例，在不同编程接口中的代码实现如下：

C 接口：

```
int ncone = 1;
int conetype[] = {COPT_CONE_QUAD};
int conebeg[] = {0};
int conecnt[] = {4};
int coneind[] = {3, 0, 1, 2};
COPT_AddCones(prob, ncone, conetype, conebeg, conecnt, coneind)
```

C++ 接口：

```
VarArray cvars;
cvars.PushBack(x4);
cvars.PushBack(x1);
cvars.PushBack(x2);
cvars.PushBack(x3);
model.AddCone(cvars, COPT_CONE_QUAD);
```

C# 接口：

```
VarArray cvars = new VarArray();
cvars.PushBack(x4);
cvars.PushBack(x1);
cvars.PushBack(x2);
cvars.PushBack(x3);
model.AddCone(cvars, Copt.Consts.CONE_QUAD);
```

Java 接口：

```
VarArray cvars = new VarArray();
cvars.PushBack(x4);
cvars.PushBack(x1);
cvars.PushBack(x2);
```

(续下页)

(接上页)

```
cvars.PushBack(x3);
model.addCone(cvars, copt.Consts.CONE_QUAD);
```

Python 接口：

```
model.addCone([x4, x1, x2, x3], COPT.CONE_QUAD)
```

在 COPT 提供的编程接口中，除 C 语言外，其余面向对象的编程接口（C#, C++, Java, Python）均提供二阶锥约束相关的类：

- 二阶锥约束相关操作的封装
  1. `Cone` 类：杉数求解器的二阶锥约束的相关操作的封装；
  2. `ConeArray` 类：方便用户对一组 `Cone` 类对象进行操作。
- 二阶锥约束构建器的封装
  1. `ConeBuilder` 类：杉数求解器中构建二阶锥约束的构建器的封装；
  2. `ConeBuilderArray` 类：为方便用户对一组 `ConeBuilder` 类对象进行操作。
- C++ API：*Cone 类* ， *ConeArray 类* ， *ConeBuilder 类* ， *ConeBuilderArray 类*
- C# API：*Cone 类* ， *ConeArray 类* ， *ConeBuilder 类* ， *ConeBuilderArray 类*
- Java API：*Cone 类* ， *ConeArray 类* ， *ConeBuilder 类* ， *ConeBuilderArray 类*
- Python API：*Cone 类* ， *ConeArray 类* ， *ConeBuilder 类* ， *ConeBuilderArray 类*

二阶锥约束相关属性

- `COPT_INTATTR_CONES` 或 `"Cones"`

  整数属性。

  二阶锥约束的个数。

12.3 半定规划（SDP）

半定规划（Semidefinite Programming，简称 SDP）的目标函数是线性函数，约束条件半定锥约束和线性约束。

12.3.1 数学模型

正半定锥：

令 $\mathcal{S}^n$ 表示 n 维对称矩阵的集合，则正半定锥表示为：

$$\mathcal{S}_+^n = \left\{X \in \mathcal{S}^n \mid u^T X u \geq 0, \forall u \in \mathbb{R}^n\right\} \tag{12.10}$$

正半定 (Positive Semidefinite，简称 PSD) 变量：

在 SDP 模型中，决策变量 $X \in \mathcal{S}_+^n$，也可表示为 $X \succeq 0$。决策变量被称为半定变量；构成模型的线性约束中含有半定变量。

半定锥约束和半定约束：

在 SDP 模型中，$X \succeq 0$ 习惯上被称为 **半定锥约束**，而形如 $A \bullet X = b$ 是含半定变量的线性约束。此处为了表述方便，下文我们将其简称为 **半定约束**，同时与半定锥约束（$X \succeq 0$）区分开来。

半定规划的数学形式如下：

$$\begin{aligned} \min \quad & \sum_{j=0}^{n-1} c_j x_j + \sum_{j=0}^{p-1} C_j \bullet X_j + c^f \\ \text{s.t.} \quad & l_i^c \le \sum_{j=0}^{n-1} a_{ij} x_j + \sum_{j=0}^{p-1} A_j \bullet X_j \le u_i^c, \qquad i = 0, 1, ..., m-1 \\ & l_j^v \le \qquad\qquad x_j \qquad\qquad \le u_j^v, \qquad j = 0, 1, ..., n-1 \\ & \qquad\qquad X_j \succeq 0, \qquad j = 0, 1, ..., p-1 \end{aligned} \tag{12.11}$$

这里运算符 $\bullet$ 表示矩阵内积运算：

给定两个 $m \times n$ 维矩阵：$A = \{a_{ij}\} \in \mathbb{R}^{m \times n}, B = \{b_{ij}\} \in \mathbb{R}^{m \times n}$，矩阵 A 和矩阵 B 的内积定义为

$$A \bullet B = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_{ij} b_{ij} \tag{12.12}$$

模型中的变量和参数含义如下：

- 问题规模：m 表示约束条件的个数，n 表示非半定变量的个数，p 表示半定变量的个数
- 决策变量：非半定变量 $x = (x_j)_{j=0}^{n-1} \in \mathbb{R}^n$，半定变量 $X_j \in \mathcal{S}_+^{r_j}$ （$j = 0, ..., p-1$）
- 非半定变量取值范围：$l^v, u^v \in \mathbb{R}^n$，其中 l^v 表示非半定变量下界，u^v 表示非半定变量上界
- 约束边界：$l^c, u^c \in \mathbb{R}^m$，其中 u^c 表示约束下界，u^c 表示约束上界
- 线性约束的系数矩阵：$A = (a_{ij})_{m \times n} \in \mathbb{R}^{m \times n}$，$A_j \in \mathbb{R}^{r_j \times r_j}$
- 目标函数中变量系数：$c \in \mathbb{R}^n$ 表示目标函数中非半定变量的系数，$C_j \in \mathbb{R}^{r_j \times r_j}$ 表示目标函数中半定变量的系数，c^f 表示目标函数中的常数项

12.3.2 建模

在 COPT 中构建半定规划（SDP）模型并求解的基本步骤如下：

1. 创建 COPT 求解环境和求解模型
2. 添加模型参数
3. 构建半定规划模型：
   - 添加决策变量（半定变量、非半定变量）
   - 添加约束条件（半定约束）
   - 设置目标函数
4. 设置求解参数并求解

5. 获取求解结果

建模：添加半定变量、添加半定约束

COPT 通过指定半定变量维度（dim）的方式添加半定变量：

1. 添加单个半定变量
2. 添加多个半定变量

杉数求解器 COPT 提供添加半定约束的两种方式：

1. 先构建半定约束表达式，分别对半定项（半定变量及其对应系数矩阵）和线性项进行组合，再添加至模型中；
2. 直接给出半定约束表达式（或半定约束构建器）作为参数输入，添加至模型中。

在向模型中添加半定约束时，用户可以指定的参数主要有：

- `expr` / `builder`：半定约束表达式或半定约束构建器
- `rhs`：二次约束右端项
- `sense`：约束类型，可取值请参考*常数章节：约束类型*
- `name`：半定约束名称

在不同编程接口中实现方式如 表 12.7 所示：

表 12.7: 添加半定变量和半定约束的函数

编程接口	添加半定变量	添加半定约束
C	`COPT_AddPSDCol` / `COPT_AddPSDCols`	`COPT_AddPSDConstr`
C++	`Model::AddPsdVar()` / `Model.AddPsdVars()`	`Model::AddPsdConstr()`
C#	`Model.AddPsdVar()` / `Model.AddPsdVars()`	`Model.AddPsdConstr()`
Java	`Model.addPsdVar()` / `Model.addPsdVars()`	`Model.addPsdConstr()`
Python	`Model.addPsdVar()` / `Model.addPsdVars()`	`Model.addPsdConstr()`

注意

- 关于半定变量、半定约束建模的相关操作，在不同编程接口中的函数名称、调用方式，以及 **参数名称略有不同**，但是实现的功能和参数含义是一致的；
- 在 C API 中，添加半定约束时，需提供非零线性项系数列下标、半定变量列下标等参数，具体请参考*C API 函数：构造和修改模型* 的函数 `COPT_AddPSDConstr` ；
- 在 Python API 中，`Model.addConstr()` 可以用于添加一条线性约束、半定约束或 Indicator 约束至模型中，具体请参考*Python API 函数：Model 类* 。

半定变量和半定约束示例

$$\begin{aligned} & A \bullet X + x_1 + x_2 = 0.6 \\ & x_1 \geq 0, x_2 \geq 0, X \succeq 0 \end{aligned} \tag{12.13}$$

其中：

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{12.14}$$

在不同编程接口中的代码实现如下：

C 接口：

```
{
  /* Matrix A */
  int ndim = 3;
  int nelem = 6;
  int rows[] = {0, 1, 2, 1, 2, 2};
  int cols[] = {0, 0, 0, 1, 1, 2};
  double elems[] = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0};
  retcode = COPT_AddSymMat(prob, ndim, nelem, rows, cols, elems);
  if (retcode) goto exit_cleanup;
}
/* Add PSD columns */
int nPSDCol = 1;
int colDims[] = {3};
retcode = COPT_AddPSDCols(prob, nPSDCol, colDims, NULL);
if (retcode) goto exit_cleanup;
/* Add columns */
int nCol = 2;
retcode = COPT_AddCols(prob, nCol, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL);
if (retcode) goto exit_cleanup;
/* Add PSD constraints */
{
  int nRowMatCnt = 2;
  int rowMatIdx[] = {0, 1};
  double rowMatElem[] = {1.0, 1.0};
  int nColCnt = 1;
  int psdColIdx[] = {0};
  int symMatIdx[] = {2};
  char cRowSense = COPT_EQUAL;
  double dRowBound = 0.6;
  retcode = COPT_AddPSDConstr(prob, nRowMatCnt, rowMatIdx, rowMatElem, nColCnt,
    psdColIdx, symMatIdx, cRowSense, dRowBound, 0.0, NULL);
  if (retcode) goto exit_cleanup;
}
```

C++ 接口：

```
SymMatrix A = model.AddOnesMat(3);
Var x1 = model.AddVar(0.0, COPT_INFINITY, 0, COPT_CONTINUOUS, "x1");
Var x2 = model.AddVar(0.0, COPT_INFINITY, 0, COPT_CONTINUOUS, "x2");
```

(续下页)

(接上页)

```
PsdVar barX = model.AddPsdVar(3, "X");
model.AddPsdConstr(A * barX + x1 + x2 == 0.6, "PSD_R");
```

C# 接口：

```
SymMatrix A = model.AddOnesMat(3);
Var x1 = model.AddVar(0.0, Copt.Consts.INFINITY, 0, Copt.Consts.CONTINUOUS, "x1");
Var x2 = model.AddVar(0.0, Copt.Consts.INFINITY, 0, Copt.Consts.CONTINUOUS, "x2");
PsdVar barX = model.AddPsdVar(3, "X");
model.AddPsdConstr(A * barX + x1 + x2 == 0.6, "PSD_R");
```

Java 接口：

```
SymMatrix A = model.addOnesMat(3);
Var x1 = model.addVar(0.0, copt.Consts.INFINITY, 0, copt.Consts.CONTINUOUS, "x1");
Var x2 = model.addVar(0.0, copt.Consts.INFINITY, 0, copt.Consts.CONTINUOUS, "x2");
PsdVar barX = model.addPsdVar(3, "X");
PsdExpr pexpr = new PsdExpr(x1, 1.0);
pexpr.addTerm(x2, 1.0);
pexpr.addTerm(barX, A);
model.addPsdConstr(pexpr, copt.Consts.EQUAL, 0.6, "PSD_R");
```

Python 接口：

```
A = m.addOnesMat(3)
x1 = m.addVar(lb=0.0, ub=COPT.INFINITY, name="x1")
x2 = m.addVar(lb=0.0, ub=COPT.INFINITY, name="x2")
X = m.addPsdVars(3, "BAR_X")
psdc = model.addConstr(A * X + x1 + x2 == 0.6, name="PSD_C")
```

在 COPT 提供的编程接口中，除 C 语言外，其余面向对象的编程接口（C#, C++, Java, Python）均提供半定约束相关的类：

- 构建半定表达式相关操作的封装
  1. `PsdExpr` 类：杉数求解器 COPT 中用于构建半定表达式时对半定变量的相关组合操作的封装。
- 半定约束相关操作的封装
  1. `PsdConstraint` 类：杉数求解器半定约束的相关操作的封装；
  2. `PsdConstrArray` 类：方便用户对一组 `PsdConstraint` 类对象进行操作。
- 半定约束构建器的封装
  1. `PsdConstrBuilder` 类：杉数求解器中构建半定约束构建器的封装；
  2. `PsdConstrBuilderArray` 类：为方便用户对一组 `PsdConstrBuilder` 类对象进行操作。
- C++ API：*PsdExpr 类* ，*PsdConstrArray 类* ，*PsdConstrBuilder 类* ，*PsdConstrBuilderArray 类*
- C# API：*PsdExpr 类* ，*PsdConstraint 类* ，*PsdConstrArray 类* ，*PsdConstrBuilder 类* ，*PsdConstrBuilderArray 类*

- Java API：*PsdExpr 类* ，*PsdConstraint 类* ，*PsdConstrArray 类* ，*PsdConstrBuilder 类* ，*PsdConstrBuilderArray 类*
- Python API：*PsdExpr 类* ，*PsdConstraint 类* ，*PsdConstrArray 类* ，*PsdConstrBuilder 类* ，*PsdConstrBuilderArray 类*

12.3.3 求解

对于半定规划问题，杉数求解器 COPT 提供的求解算法有：内点法和交替乘子下降法，通过设置优化参数 `SDPMethod` 可以进行指定。

详细请参考参数章节：半定规划相关 。

12.3.4 相关属性

半定规划模型描述相关属性如 表 12.8 所示：

表 12.8: 半定变量和半定约束相关属性

属性名	类型	属性含义
PSDCols	整数属性	半定变量的个数
PSDElems	整数属性	目标函数中半定项个数
SymMats	整数属性	模型中对称矩阵的个数
PSDConstrs	整数属性	半定约束的个数
HasPSDObj	整数属性	模型的目标函数是否包含半定项

12.4 二次规划（QP）

12.4.1 数学模型

凸二次规划（Convex Quadratic Programming，简称 QP）的目标函数是凸二次函数，约束条件是线性约束。

数学形式如下：

$$\begin{aligned} \min \quad & \frac{1}{2}x^T Q x + c^T x + c^f \\ \text{s.t.} \quad & l_i^c \le \sum_{j=0}^{n-1} a_{ij} x_j \le u_i^c, \qquad i = 0, \cdots, m-1 \\ & l_j^v \le x_j \le u_j^v, \qquad j = 0, 1, \cdots, n-1 \end{aligned} \tag{12.15}$$

模型中的变量和参数含义如下：

- 决策变量：$x = (x_j)_{j=0}^{n-1} \in \mathbb{R}^n$；
- 变量取值范围：$l^v, u^v \in \mathbb{R}^n$；其中，$l^v$ 表示变量下界，u^v 表示变量上界；
- 约束边界：$l^c, u^c \in \mathbb{R}^m$；其中，$l^c$ 表示约束下界，u^c 表示约束上界；

- 线性约束系数矩阵：$A = (a_{ij})_{m\times n} \in \mathbb{R}^{m\times n}$
- 目标函数中变量系数：
  - $Q \in \mathbb{R}^{n\times n}$ 表示目标函数中二次项系数
  - $c \in \mathbb{R}^n$ 表示目标函数中线性项系数，c^f 表示目标函数中的常数项
- 问题规模：m 表示约束条件数目，n 表示决策变量个数。

12.5 二次约束规划（QCP）

凸二次约束规划（Convex Quadratically Constrained Programming，简称 QCP）的目标函数是凸二次函数，约束条件是凸二次约束和线性约束。

12.5.1 数学模型

数学形式如下：

$$\begin{aligned} \min \quad & \frac{1}{2}x^T Q x + c^T x + c^f \\ \text{s.t.} \quad & \frac{1}{2}x^T P_i x + a_i^T x_j \le u_i^c, \qquad i = 0, \cdots, m-1 \\ & l^v \le x_j \le u^v, \qquad j = 0, 1, \cdots, n-1 \end{aligned} \tag{12.16}$$

模型中的变量和参数含义如下：

- 决策变量：$x = (x_j)_{j=0}^{n-1} \in \mathbb{R}^n$ ；
- 变量取值范围：$l^v, u^v \in \mathbb{R}^n$，其中 l^v 表示变量下界，u^v 表示变量上界；
- 约束边界：$u^c \in \mathbb{R}^m$，其中 u^c 表示约束上界；
- 二次约束中的系数：
  - $P_i \in \mathbb{R}^{n\times n} \quad (i = 0, ..., m-1)$，
  - $a_i = (a_{ij})_n \in \mathbb{R}^n$
- 目标函数中变量系数：
  - $Q \in \mathbb{R}^{n\times n}$ 表示目标函数中二次项系数
  - $c \in \mathbb{R}^n$ 表示目标函数中线性项系数，c^f 表示目标函数中的常数项
- 问题规模：m 表示约束条件数目，n 表示决策变量个数。

注意：

- 杉数求解器 COPT 目前支持求解凸二次规划和凸二次约束规划问题，此处的 Q 和 P_i $(i = 0, 1, \cdots, n)$ 需要为 **正半定矩阵**；
- 当约束类型为 >= 时（形如：$\frac{1}{2}x^T P_i x + \sum_{j=0}^{n-1} a_{ij}x_j \ge l_i^c$），此处的 Q 则为 **负半定矩阵**；
- 从以上数学形式看出，二次约束表达式中包含二次项、线性项和常数项。

12.5.2 建模

在 COPT 中构建二次约束规划（QCP）模型并求解的基本步骤如下：

1. 创建 COPT 求解环境和求解模型
2. 添加模型参数
3. 构建二次约束规划模型：
   - 添加变量
   - 添加二次约束
   - 设置二次目标函数
4. 设置求解参数并求解
5. 获取求解结果

建模：添加二次约束

杉数求解器 COPT 提供添加二次约束的两种方式：

1. 先对构建二次约束表达式，分别对二次项和线性项变量进行组合，再添加至模型中；
2. 直接给出二次约束表达式（或二次约束构建器）作为参数输入，添加至模型中。

在向模型中添加二次约束时，用户可以指定的参数主要有：

- `expr` / `builder`：二次约束表达式或二次约束构建器
- `rhs`：二次约束右端项
- `sense`：约束类型，可取值有：`COPT_LESS_EQUAL` 和 `COPT_GREATER_EQUAL`
- `name`：二次约束名称

在不同编程接口中实现方式如 表 12.9 所示：

表 12.9: 添加二次约束的函数

编程接口	函数
C	`COPT_AddQConstr`
C++	`Model::AddQConstr()`
C#	`Model.AddQConstr()`
Java	`Model.addQConstr()`
Python	`Model.addQConstr()`

注意：

- 关于二次约束建模相关操作，在不同编程接口中的函数名称、调用方式，以及 **参数名称略有不同**，但是实现的功能和参数含义是一致的；
- 在 C API 中，添加二次约束时，需提供非零线性项系数列下标，二次项下标等参数，具体请参考*C API 函数：构造和修改模型* 的函数 `COPT_AddQConstr` ；

- 在 Python API 中，`Model.addQConstr()` 可以用于添加一条线性约束二次约束至模型中，具体请参考*Python API 函数：Model 类* 。具体请参考*Python API 函数：Model 类* 。

二次约束示例

$$x_1^2 + x_2^2 + x_3^2 + x_1 + 2x_2 <= 0 \tag{12.17}$$

在不同编程接口中的代码实现如下：

C 接口：

```
int nRowMatCnt = 2;
int rowMatIdx[] = {0, 1};
double rowMatElem[] = {1.0, 2.0};

int nQMatCnt = 2;
int qMatRow[] = {0, 1};
int qMatCol[] = {0, 1};
double qMatElem[] = {1.0, 1.0};
char cRowSense = COPT_LESS_EQUAL;
double dRowBound = 0.0;
char *name = "q1";
errcode = COPT_AddQConstr(prob, nRowMatCnt, rowMatIdx, rowMatElem,
                          nQMatCnt, qMatRow, qMatCol, qMatElem,
                          cRowSense, dRowBound, name);
```

C++ 接口：

```
model.AddQConstr(x1*x1 + x2*x2 + x1 + 2*x2 <= 0, "q1");
```

C# 接口：

```
model.AddQConstr(x1*x1 + x2*x2 + x1 + 2*x2 <= 0, "q1");
```

Java 接口：

```
QuadExpr q1 = new QuadExpr(0.0);
q1.addTerm(x1, x1, 1);
q1.addTerm(x2, x2, 1);
q1.addTerm(x1, 1);
q1.addTerm(x2, 2);
model.addQConstr(q1, copt.Consts.LESS_EQUAL, 0, "q1");
```

Python 接口：

```
model.addConstr(x1*x1 + x2*x2 + x1 + 2*x2 <= 0, name="q1")
```

在 COPT 提供的编程接口中，除 C 语言外，其余面向对象的编程接口（C#, C++, Java, Python）均提供二阶锥约束相关的类：

- 构建二次表达式相关操作的封装
  1. `QuadExpr` 类：杉数求解器 COPT 中用于构建二次表达式时对变量的相关组合操作的封装。
- 二次约束相关操作的封装
  1. `QConstraint` 类：杉数求解器二次约束的相关操作的封装；
  2. `QConstrArray` 类：为方便用户对一组 `QConstraint` 类对象进行操作。
- 二次约束构建器的封装
  1. `QConstrBuilder` 类：杉数求解器中构建二次约束时的构建器的封装；
  2. `QConstrBuilderArray` 类：为方便用户对一组 `QConstrBuilder` 类对象进行操作。
- C++ API：*QuadExpr 类* ，*QConstraint 类* ，*QConstrArray 类* ，*QConstrBuilder 类* ，*QConstrBuilderArray 类*
- C# API：*QuadExpr 类* ，*QConstraint 类* ，*QConstrArray 类* ，*QConstrBuilder 类* ，*QConstrBuilderArray 类*
- Java API：*QuadExpr 类* ，*QConstraint 类* ，*QConstrArray 类* ，*QConstrBuilder 类* ，*QConstrBuilderArray 类*
- Python API：*QuadExpr 类* ，*QConstraint 类* ，*QConstrArray 类* ，*QConstrBuilder 类* ，*QConstrBuilderArray 类*

12.5.3 相关属性

二次规划和二次约束规划模型描述相关属性如 表 12.10 所示：

表 12.10: 二次规划和二次约束规划相关属性

属性名	类型	属性含义
`QConstrs`	整数属性	二次约束的个数
`QElems`	整数属性	二次目标函数中非零二次项个数
`HasQObj`	整数属性	模型是否包含二次项目标函数

12.6 混合整数规划（MIP）

12.6.1 建模

混合整数规划问题 (MIP) 指的是模型中部分决策变量的取值为整数型。目前杉数求解器 COPT 支持整数变量和线性规划、二阶锥规划、二次规划、二次约束规划结合。

MIP 问题类型	求解算法
混合整数线性规划 (MILP)	分支切割法
混合整数二阶锥规划 (MISOCP)	分支切割法
混合整数凸二次规划 (MIQP)	分支切割法
混合整数凸二次约束规划 (MIQCP)	分支切割法

在 COPT 中，支持的整数型变量及对应常数表示如下：

- `BINARY`

  二进制变量 (0-1 变量)

- `INTEGER`

  整数变量

在向模型中添加决策变量时，

- 指定参数 `vtype` 为 `BINARY` 添加 0-1 变量；
- 指定参数 `vtype` 为 `INTEGER` 添加整数变量。

12.6.2 求解

对于整数规划问题，杉数求解器 COPT 提供的求解算法有：分支切割法，通过设置优化参数 `"MipMethod"` 可以进行指定。通过设置其他整数规划的相关优化参数，可以控制分支切割求解算法的具体工作流程，详细请参考参数章节：整数规划相关 。

关于线性规划问题的求解日志请参考求解日志章节：分支切割法 。

12.6.3 相关属性

表 12.11: MIP 相关属性总览

属性名	类型	属性含义
`Bins`	整数属性	二进制变量（列）的个数
`Ints`	整数属性	整数变量（列）的个数
`Indicators`	整数属性	Indicator 约束的个数
`IsMIP`	整数属性	模型是否为整数规划模型
`NodeCnt`	整数属性	分支定界搜索的节点数。
`HasMipSol`	整数属性	是否存在整数解。
`BestObj`	浮点数属性	整数规划求解结束时最好的目标函数值。
`BestBnd`	浮点数属性	整数规划求解结束时最好的下界。
`BestGap`	浮点数属性	整数规划求解结束时最好的相对容差。

12.7 特殊约束

COPT 支持构建两种特殊的约束：SOS 约束和 Indicator 约束。

12.7.1 SOS 约束

SOS 约束（Special Ordered Set）是一类限制一组变量取值的特殊约束。目前，COPT 支持两种类型的 SOS 约束：

1. **SOS1 约束**：该类型约束中指定的一组变量至多有一个变量可取非零值；
2. **SOS2 约束**：该类型约束中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。

以上两种 SOS 约束在 COPT 中分别对应常数如下，在向模型添加 SOS 约束时可以进行指定：

- `SOS_TYPE1`

  SOS1 约束

- `SOS_TYPE2`

  SOS2 约束

在向模型中添加 SOS 约束时，用户可以指定以下参数：

- `sostype`：设置 SOS 约束的类型
- `vars`：参与 SOS 约束的变量
- `weights`：参与 SOS 约束变量的权重，可选参数，默认为 `None`

注意

- 参与构成 SOS 约束中的变量类型可以为连续变量、二进制变量或整数变量；
- 若模型中包含 SOS 约束，则模型为整数规划模型；
- 关于 SOS 约束的具体操作和用法，在不同编程接口中的函数名称、调用方式，以及 **参数名称略有不同**，但是实现的功能和参数含义是一样的。

杉数求解器 COPT 提供相关函数，支持对 SOS 约束进行一些操作。下面罗列不同编程接口中，添加和获取 SOS 约束对应的函数：

表 12.12: 添加和获取 SOS 约束

编程接口	添加 SOS 约束	获取模型中所有 SOS 约束
C	`COPT_AddSOSs`	`COPT_GetSOSs`
C++	`Model::AddSos()`	`Model::GetSoss()`
C#	`Model.AddSos()`	`Model.GetSoss()`
Java	`Model.addSos()`	`Model.getSoss()`
Python	`Model.addSOS()`	`Model.getSOSs()`

关于 SOS 约束的相关操作，在不同编程接口中，函数的名称、调用方式，以及参数名称略有不同，但是实现的功能和参数含义是一致的。请进一步参考各编程语言 API 参考手册的对应章节，以获取具体介绍：

- C API：构造和修改模型

- C++ API：*Model 类*
- C# API：*Model 类*
- Java API：*Model 类*
- Python API：*Model 类*

在 COPT 提供的编程接口中，除 C 语言外，其余面向对象的编程接口（C#, C++, Java, Python）均提供 SOS 约束相关的类：

- SOS 约束相关操作的封装：
  1. `Sos` 类：杉数求解器 COPT 中 SOS 约束相关操作的封装
  2. `SosArray` 类：方便用户对一组 `Sos` 类对象进行操作，
- SOS 约束构建器的封装：
  1. `SosBuilder` 类：是杉数求解器 COPT 中构建 SOS 约束的构建器的封装，提供了以下成员方法：
  2. `SosBuilderArray` 类：为方便用户对一组 `SosBuilder` 类对象进行操作，

以上 SOS 约束相关的类中成员方法和具体介绍，请进一步参考各编程语言 API 参考手册中的对应内容：

- C++ API：*Sos 类* ，*SosArray 类* ，*SosBuilder 类* ，*SosBuilderArray 类*
- C# API：*Sos 类* ，*SosArray 类* ，*SosBuilder 类* ，*SosBuilderArray 类*
- Java API：*Sos 类* ，*SosArray 类* ，*SosBuilder 类* ，*SosBuilderArray 类*
- Python API：*SOS 类* ，*SOSArray 类* ，*SOSBuilder 类* ，*SOSBuilderArray 类*

12.7.2 Indicator 约束

Indicator 约束是一类逻辑关系约束，它以一个二进制类型变量 y 作为 Indicator 变量，根据变量 y 的取值决定线性约束 $a^T x \leq b$ 是否成立。Indicator 约束的一般表达式：

$$\begin{aligned} & y = f \rightarrow a^T x \leq b \\ & f \in \{0, 1\} \end{aligned} \tag{12.18}$$

- 当 $y = f$ 时，线性约束成立；
- 当 $y \neq f$ 时，线性约束无效（可以被违反）。

构建 Indicator 约束时，用户需要指定以下参数：

- `binVar`： 二进制的 Indicator 变量
- `binval`： 要求线性约束必须满足的 Indicator 变量的取值（0 或 1）
- `builder`： 线性约束构建器

注意

- 以上给出的线性约束的一般表达式 $a^T x \leq b$，实际上，线性约束的方向可取 $\leq$ 、$\geq$ 和 $=$ 三种情形；
- 若模型包含 Indicator 约束，则模型为整数规划模型；

- 关于 SOS 约束的具体操作和用法，在不同编程接口中的函数名称、调用方式，以及 **参数名称略有不同**，但是实现的功能和参数含义是一样的。

杉数求解器 COPT 提供相关函数，支持对添加 Indicator 约束、获取指定 Indicator 约束对应的 Indicator 约束构建器，罗列如下：

表 12.13: 不同编程接口添加和获取 Indicator 约束

编程接口	添加 Indicator 约束	获取指定 Indicator 约束构建器
C	`COPT_AddIndicator`	`COPT_GetIndicator`
C++	`Model::AddGenConstrIndicator()`	`Model::GetGenConstrIndicator()`
C#	`Model.AddSos()`	`Model.GetGenConstrIndicator()`
Java	`Model.addSos()`	`Model.getGenConstrIndicator()`
Python	`Model.addGenConstrIndicator()`	`Model.getGenConstrIndicator()`

关于 Indicator 约束的构建、添加等操作，在不同编程接口中，函数的名称、调用方式，以及参数名称略有不同，但是实现的功能和参数含义是一致的。请进一步参考各编程语言 API 参考手册的对应章节，以获取具体介绍：

- C API：*构造和修改模型*
- C++ API：*Model 类*
- C# API：*Model 类*
- Java API：*Model 类*
- Python API：*Model 类*

在 COPT 支持的编程接口中，除 C 语言外，其余面向对象的编程接口（C#, C++, Java, Python）均提供 Indicator 约束相关的类：

- Indicator 约束相关操作的封装：
  1. `GenConstr` 类：杉数求解器 COPT 的 `Indicator` 约束相关操作的封装；
  2. `GenConstrArray` 类：方便用户对一组 `GenConstr` 类对象进行操作。
- Indicator 约束构建器的封装：
  1. `GenConstrBuilder` 类：杉数求解器 COPT 中构建 `Indicator` 约束时构建器的封装；
  2. `GenConstrBuilderArray` 类：方便用户对一组 `GenConstrBuilder` 类对象进行操作。

以上 Indicator 约束相关的类中成员方法和具体介绍，请进一步参考各编程语言 API 参考手册中的对应内容：

- C++ API：*GenConstr 类*，*GenConstrArray 类*，*GenConstrBuilder 类*，*GenConstrBuilderArray 类*
- C# API：*GenConstr 类*，*GenConstrArray 类*，*GenConstrBuilder 类*，*GenConstrBuilderArray 类*
- Java API：*GenConstr 类*，*GenConstrArray 类*，*GenConstrBuilder 类*，*GenConstrBuilderArray 类*

- Python API：*GenConstr 类*，*GenConstrArray 类*，*GenConstrBuilder 类*，*GenConstrBuilderArray 类*

12.7.3 特殊约束相关属性

COPT 提供以下属性，用来描述模型中特殊约束的数目，如 表 12.14 所示。

不同编程接口中的属性获取方法，请参考：*属性章节*。

表 12.14: 特殊约束相关属性总览

属性名	类型	属性含义
Soss	整数属性	SOS 约束的个数
Indicators	整数属性	Indicator 约束的个数
IISSOSs	整数属性	组成 IIS 的 SOS 约束的数目
IISIndicators	整数属性	组成 IIS 的 Indicator 约束的数目

12.7.4 特殊约束的 IIS 状态

关于不可行模型的 IIS 计算结果，COPT 提供相关函数，以获取 SOS 约束的 IIS 状态，请参考不可行模型的处理章节：*获取特殊约束 IIS 状态* 。

第 13 章　不可行模型的处理

本章将介绍杉数求解器 COPT 对不可行问题的两种处理方式，章节内容构成如下：

- 不可行模型的 *IIS*
- 不可行模型的可行化松弛

在对现实问题进行建模时，我们通常会遇到模型不可行的情况，对应的解状态码为 `COPT.INFEASIBLE` 。导致模型不可行的原因通常有以下两种：

1. 描述模型的参数数据输入出错（例如某条约束左端项为空）；
2. 问题本身就不可行，某些约束条件或变量范围之间冲突。

COPT 提供两种不可行模型的分析和处理方式：

1. **计算不可行模型的 IIS**：定位导致模型不可行的关键约束和变量范围；
2. **可行化松弛（FeasRelax）**：定量给出导致模型不可行的约束或变量范围的冲突值（Violations）。

13.1 不可行模型的 IIS

IIS（不可约不一致子系统，Irreducible Inconsistent Subsystem）是导致模型不可行的最小冲突集，具有以下特点：

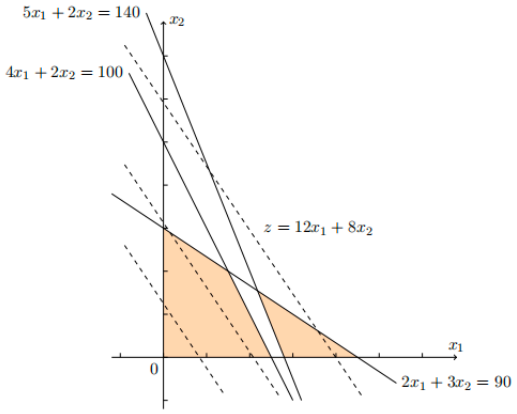
1. 仍然是不可行的；
2. 剔除 IIS 中任意一条约束或变量范围，这个子系统会变得可行；

注意： COPT 计算得到的 IIS 不一定是最小的，也不是唯一的。因此有时需要多次调整约束条件后，重新计算 IIS，才能使模型最终变得可行。

下面是一个不可行线性规划模型的例子：

$$\begin{aligned} \max \quad & z = 12x_1 + 8x_2 \\ \text{s.t.} \quad & 5x_1 + 2x_2 \geq 140 \\ & 2x_1 + 3x_2 \leq 90 \\ & 4x_1 + 2x_2 \leq 100 \\ & x_1, x_2 \geq 0 \end{aligned} \tag{13.1}$$

模型的可行域如下图所示：



从图示中我们可以清楚看出，相互冲突的两条约束为：

$$\begin{aligned} c1 &: 5x_1 + 2x_2 \geq 140 \\ c3 &: 4x_1 + 2x_2 \leq 100 \end{aligned} \tag{13.2}$$

计算上述模型的 IIS 结果并写入文件（`.iis` 格式），文件内容如下，和图示结果一致：

```
\Generated by Cardinal Operations

Maximize
   12 x[1] + 8 x[2]
Subject To
 c1: 5 x[1] + 2 x[2] >= 140
 c3: 4 x[1] + 2 x[2] <= 100
END
```

13.1.1 计算 IIS

COPT 支持的编程接口提供计算不可行模型 IIS 的函数，得到一个包含模型中相互冲突约束和变量的集合。用户也可以将 IIS 计算结果写入文件，指定文件名后缀为 `.iis`（如：`example.iis`），相关函数名称如 表 13.1 所示：

表 13.1: 不可行模型 IIS 计算功能的函数

编程接口	计算 IIS	将 IIS 计算结果写入文件
C	`COPT_ComputeIIS`	`COPT_WriteIIS`
C++	`Model::ComputeIIS()`	`Model::WriteIIS()`
C#	`Model.ComputeIIS()`	`Model.WriteIIS()`
Java	`Model.computeIIS()`	`Model.writeIIS()`
Python	`Model.computeIIS()`	`Model.writeIIS()`

13.1.2 获取变量和约束的 IIS 状态

获取变量 IIS 状态相关函数

计算不可行模型 IIS 完成后，可以通过获取变量（下/上界）的 IIS 状态，状态输出值表明变量边界是否在 IIS 中：

- 1：指定变量（下界/上界）在 IIS 中
- 0：指定变量（下界/上界）不在 IIS 中

不同编程语言接口提供函数如 表 13.2 所示：

表 13.2: 获取变量 IIS 状态相关函数

编程接口	变量下界 IIS 状态	变量上界 IIS 状态
C	`COPT_GetColLowerIIS`	`COPT_GetColUpperIIS`
C++	`Model::GetVarLowerIIS()`	`Model::GetVarUpperIIS()`
C#	`Model.GetVarLowerIIS()`	`Model.GetVarUpperIIS()`
Java	`Model.getVarLowerIIS()`	`Model.getVarUpperIIS()`
Python	`Model.getVarLowerIIS()`	`Model.getVarUpperIIS()`

注意

- 以上函数的输入值均可以为单个变量（`Var` 类对象）或一组变量（`VarArray` 类或 `tupledict` 类对象）；
- 除 C API 外，其余编程语言接口的 `Var` 类还提供获取单个变量 IIS 状态的函数。

  以 Python 为例，`Var.getLowerIIS()` 和 `Var.getUpperIIS()` ；
- 在 Python API 中，还可以直接访问 `Var` 类对象的成员属性获取单个变量的 IIS 状态：

  `Var.iislb` 获取变量下界 IIS 状态，`Var.iisub` 获取变量上界 IIS 状态。

获取约束 IIS 状态相关函数

计算不可行模型 IIS 完成后，COPT 支持获取约束（下/上边界）的 IIS 状态，状态输出值表明约束边界是否在 IIS 中：

- 1：指定约束（下边界/上边界）在 IIS 中
- 0：指定约束（下边界/上边界）不在 IIS 中

不同编程语言接口提供函数如 表 13.3 所示：

表 13.3: 获取约束边界 IIS 状态相关函数

编程接口	约束下边界 IIS 状态	约束上边界 IIS 状态
C	`COPT_GetRowLowerIIS`	`COPT_GetRowUpperIIS`
C++	`Model::GetConstrLowerIIS()`	`Model::GetConstrUpperIIS()`
C#	`Model.GetConstrLowerIIS()`	`Model.GetConstrUpperIIS()`
Java	`Model.getConstrLowerIIS()`	`Model.getConstrUpperIIS()`
Python	`Model.getConstrLowerIIS()`	`Model.getConstrUpperIIS()`

- 以上函数的输入值均可以为单条约束（`Constraint` 类对象）或一组约束（`Constraint` 类或 `ConstrArray` 类对象）；
- 除 C API 外，其余编程语言接口的 `Constraint` 类还提供获取单条约束 IIS 状态的函数。

  以 Python 为例，`Constraint.getLowerIIS()` 和 `Constraint.getUpperIIS()` ；
- 在 Python API 中，还可以直接访问 `Constraint` 类对象的成员属性获取单条约束的 IIS 状态：

  `Constraint.iislb` 获取约束下边界 IIS 状态，`Constraint.iisub` 获取约束上边界 IIS 状态。

获取特殊约束 IIS 状态

此外，COPT 提供如 表 13.4 所示函数，分别获取 SOS 约束和 Indicator 约束的 IIS 状态。

表 13.4: 获取特殊约束 IIS 状态相关函数

编程接口	SOS 约束	Indicator 约束
C	`COPT_GetSOSIIS`	`COPT_GetIndicatorIIS`
C++	`Model::GetSOSIIS()`	`Model::GetIndicatorIIS()`
C#	`Model.GetSOSIIS()`	`Model.GetIndicatorIIS()`
Java	`Model.getSOSIIS()`	`Model.getIndicatorIIS()`
Python	`Model.getSOSIIS()`	`Model.getIndicatorIIS()`

关于计算和获取 IIS 状态的函数具体用法和详细介绍，请进一步参考不同编程接口 API 参考手册的对应章节：

- C API：*不可行模型 IIS 计算功能函数*
- C++ API：*Model 类*，*Var 类*，*Constraint 类*
- C# API：*Model 类*，*Var 类*，*Constraint 类*
- Java API：*Model 类*，*Var 类*，*Constraint 类*
- Python API：*Model 类*，*Var 类*，*Constraint 类*

13.1.3 IIS 相关参数、属性和信息

优化参数

用户可以通过设置参数 "IISMethod" 的不同取值，选择计算 IIS 的方法。不同编程接口的参数设置方法，请参考：参数章节。

- IISMethod

  整数参数。

  计算 IIS 的方法。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 计算结果质量优先。

  1: 计算效率优先。

属性

COPT 提供相关属性，用来描述 IIS 的计算结果，主要包括判断 IIS 是否存在以及统计 IIS 中变量和约束数目两类属性。不同编程接口的属性获取方法，请参考：属性章节。

表 13.5: IIS 计算结果相关属性总览

属性名	类型	属性含义
HasIIS	整数属性	是否存在 IIS。
IsMinIIS	整数属性	计算出的 IIS 是否为极小。
IISCols	整数属性	组成 IIS 的变量边界的数目。
IISRows	整数属性	组成 IIS 的约束的数目。
IISSOSs	整数属性	组成 IIS 的 SOS 约束的数目。
IISIndicators	整数属性	组成 IIS 的 Indicator 约束的数目。

示例代码

用户可以将不可行问题写成模型文件的格式，COPT 支持读取模型文件后，直接计算不可行模型的 IIS。以 Python 为例，具体实现示例如下：

```
from coptpy import *
env = Envr()
model = env.createModel("example")
model.read("example.lp")
model.computeIIS()
model.writeIIS ("example.iis")
```

用户可以在 COPT 安装包的 "examples" 文件夹下找到不同编程语言处理不可行模型 IIS 的示例代码，文件名为："iis_ex1.py" ，以 COPT 7.0 版本和 Python 语言为例，相应的文件路径为："copt70/examples/python/iis_ex1.py" 。

13.2 不可行模型的可行化松弛

可行化松弛（Feasibility Relaxation）的过程就是通过最小化原不可行模型约束和变量的冲突值，求解变量和约束上下界的松弛量。用户可以根据可行化松弛定量计算的结果，相应地调整约束或变量范围，从而使得模型变得可行。

13.2.1 计算可行化松弛

COPT 支持的编程接口提供计算可行化松弛的函数，同时也可以将可行化松弛模型写入文件，指定文件名后缀为 .relax (如：example.relax)，相关函数名称如 表 13.6 所示：

表 13.6: 不可行模型 IIS 计算功能的函数

编程接口	计算可行化松弛	将可行化松弛模型写入文件
C	COPT_FeasRelax	COPT_WriteRelax
C++	Model::FeasRelax()	Model::WriteRelax()
C#	Model.FeasRelax()	Model.WriteRelax()
Java	Model.feasRelax()	Model.writeRelax()
Python	Model.feasRelax()	Model.writeRelax()

COPT 提供了两种方式计算可行化松弛，在除 Python 之外的编程接口中，用户可以通过对函数的不同参数进行赋值来实现：

1. 简化版：对模型中全部变量和约束的可行化松弛，只需提供 2 个参数，以选择全部变量或全部约束
   - ifRelaxVars ： 是否对变量进行松弛，默认值为：True
   - ifRelaxCons ： 是否对约束进行松弛，默认值为：True
2. 完整版：可以提供更多参数输入（指定变量/约束，设置变量/约束边界的惩罚因子）
   - vars：待松弛变量
   - cons：待松弛约束
   - colLowPen：变量下界的惩罚因子
   - colUppPen：变量上界的惩罚因子
   - rowBndPen：约束边界的惩罚因子
   - rowUppPen：若 cons 为双边约束，则表示约束上边界的惩罚因子

   注意：一般情况下，设置 rowUppPen 为 NULL 即可。

可行化松弛计算函数在不同编程接口中的调用方式和参数名称 **略有不同**，但是函数功能和参数含义是一致的，请进一步参考不同编程接口 API 参考手册的对应章节获取具体介绍：

- C API：可行化松弛计算功能函数
- C++ API：*Model* 类
- C# API：*Model* 类
- Java API：*Model* 类
- Python API：*Model* 类

注意

- 针对简化版（Simplified）的可行化松弛计算方式，COPT 的 Python 接口另外提供函数：`Model.feasrelaxS(vrelax, crelax)`，只需输入两个参数（`vrelax` 和 `crelax`）：
  - `vrelax`：是否对变量进行松弛，默认值为：`True`
  - `crelax`：是否对约束进行松弛，默认值为：`True`

13.2.2 可行化松弛的相关参数、属性和信息

优化参数

用户可以通过设置参数 `"FeasRelaxMode"` 的不同取值，选择计算可行化松弛的方法。不同编程接口的参数设置方法，请参考：参数章节。

- `FeasRelaxMode`

  整数参数。

  计算可行化松弛的方法。

  默认值 0

  可选值

  0: 最小化加权冲突值。

  1: 计算最小化加权冲突下的原始模型最优可行化松弛。

  2: 最小化冲突数目。

  3: 计算最小化冲突数目下的原始模型最优可行化松弛。

  4: 最小化加权平方冲突值。

  5: 计算最小化加权平方冲突下的原始模型最优可行化松弛。

属性

COPT 提供相关属性，用来描述可行化松弛的结果，如 表 13.7 所示。不同编程接口的属性获取方法，请参考：属性章节。

表 13.7: 可行化松弛结果相关属性总览

属性名	类型	属性含义
HasFeasRelaxSol	整数属性	是否存在可行化松弛结果。
FeasRelaxObj	浮点数属性	可行化松弛值。

信息

COPT 提供以下信息，分别表示变量（或约束）下界的可行化松弛量，以及变量（或约束）上界的可行化松弛量。不同编程接口的信息获取方法，请参考：信息章节。

- `RelaxLB`

  浮点数信息。

  变量（列）或者约束（行）下界的可行化松弛量。

- `RelaxUB`

  浮点数信息。

  变量（列）或者约束（行）上界的可行化松弛量。

示例代码

用户可以在 COPT 安装包的 `"examples"` 文件夹下找到不同编程语言实现可行化松弛的示例代码，文件名为：`"feasrelax_ex1.py"` ，以 COPT 7.0 版本和 Python 语言为例，相应的文件路径为：`"cop70/examples/python/feasrelax_ex1.py"` 。

第 14 章　整数规划初始解功能

14.1 相关函数

14.1.1 设置并加载初始解

对于整数规划问题，杉数求解器 COPT 提供设置指定单个变量或一组变量初始值的方法，并加载至模型中。可以指定的参数主要有：

- `vars`： 指定变量
- `startvals`： 指定变量取值

不同编程接口中的实现方式如 表 14.1 所示：

表 14.1: 设置初始解的函数

编程接口	函数
C	`COPT_AddMipStart`
C++	`Model::SetMipStart()`
C#	`Model.SetMipStart()`
Java	`Model.setMipStart()`
Python	`Model.setMipStart(vars, startvals)`

注意

- 关于初始解相关操作，在不同编程接口中的函数名称、调用方式，以及 **参数名称略有不同**，但是实现的功能和参数含义是一致的；
- 关于在 C 语言接口设置初始解，具体请参考*C API 函数：整数规划初始解功能函数* 章节的 `COPT_AddMipStart` 函数；
- 可以通过多次调用该方法来输入不同的初始解。请在输入结束后，调用 Model 类的 `loadMipStart()` 函数，将当前设置的初始解加载至模型中，并清空之前指定的初始解信息，用户可以继续指定新的初始解。

14.1.2 初始解的读入/写入

COPT 提供文件读/写的相关函数，可以从初始解文件（".mst"）中读取变量取值，作为整数规划变量的初始解取值；也可以将整数规划求解结果或已有初始解信息，写入至初始解文件（".mst"）中。不同编程接口中初始解文件读写的函数，如 表 14.2 所示：

表 14.2: 初始解读写相关的函数

编程接口	读入初始解文件	写入初始解文件
C	`COPT_ReadMst`	`COPT_WriteMst`
C++	`Model::ReadMst`	`Model::WriteMst`
C#	`Model.ReadMst()`	`Model.WriteMst()`
Java	`Model.readMst()`	`Model.writeMst()`
Python	`Model.readMst()`	`Model.writeMst()`

14.2 相关参数

杉数求解器 COPT 提供以下优化参数，可以控制算法对模型初始解的处理方式。

- `MipStartMode`

  整数参数。

  处理初始解的方式。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 不使用任何初始解。

  1: 仅使用完整且可行的初始解。

  2: 仅使用可行的初始解（若初始解不完整，通过求解子 MIP 来补全）。

注意： 如果提供的初始解不完整，需要设置 `MipStartMode=2` ，否则初始解会被拒绝。

- `MipStartNodeLimit`

  整数参数。

  补全不完整的初始解时，求解的子 MIP 的节点数限制。

  默认值 -1

  最小值 -1

  最大值 `INT_MAX`

14.3 初始解日志输出

14.3.1 初始解被接受的情况

1. 初始解可行且比历史初始解更优

```
Initial MIP solution # 1 with objective value 9.73987 was accepted
```

2. 提供部分初始解，且设置参数 MipStartMode=2 ，通过求解子 MIP 后补全

```
Loading 1 initial MIP solution
Extending partial MIP solution # 1
Extending partial MIP solution # 1 succeed (0.2s)
Initial MIP solution # 1 with objective value 9.66566 was accepted
```

14.3.2 初始解被拒绝的情况

1. 提供的初始解不可行

```
Initial MIP solution # 1 was rejected: Primal Inf 1.00e+00  Int Inf 1.78e-15
```

2. 当前载入的初始解没有比模型中历史初始解更优（ not better ）

```
Initial MIP solution # 2 with objective value 10.3312 was rejected (not better)
```

3. 初始解不完整（ partial ），且未设置 MipStartMode=2

```
Loading 1 initial MIP solution
Initial MIP solution # 1 was rejected: partial
```

4. 从当前不完整的初始解，无法通过求解子 MIP 找到可行解

```
Loading 1 initial MIP solution
Extending partial MIP solution # 1
Extending partial MIP solution # 1 failed (infeasible)
Initial MIP solution # 1 was rejected: partial
```

第 15 章　整数规划解池功能

在使用分支定界法求解整数规划问题过程中，求解器会找到多个可行解，COPT 提供整数规划解池（Solution Pool）功能，可以获取解池中的解和对应目标函数值。支持的问题类型包括：MILP，MISOCP，MIQ(C)P。

COPT 提供相关函数，通过指定以下参数，获取解池中第 `iSol` 个解中指定变量的取值，也可以获取第 `iSol` 个解对应目标函数的值。

- `iSol`：解池中解的索引，即解池中第 `iSol` 个解（从 0 开始）；
- `vars`：指定的变量。

在不同编程接口中实现方式如 表 15.1 所示：

表 15.1: 获取解池中解和目标函数值的函数

编程接口	获取解	获取目标函数值
C	`COPT_GetSolution`	`COPT_GetPoolObjVal`
C++	`Model::GetPoolSolution()`	`Model.GetPoolSolution()`
C#	`Model.GetPoolSolution()`	`Model.GetPoolSolution()`
Java	`Model.getPoolSolution()`	`Model.getPoolObjVal()`
Python	`Model.getPoolSolution()`	`Model.getPoolObjVal()`

注意：关于解池的相关操作，在不同编程接口中的函数名称、调用方式，以及 **参数名称略有不同**，但是实现的功能和参数含义是一致的；

解池相关属性

- `PoolSols`

  整数属性。

  解池中的解的数目。

第 16 章　COPT 调优工具

16.1 调优工具基本介绍

对于支持的优化问题类型，杉数求解器 COPT 调优工具可以进行求解性能的自动调优。对于连续/非连续优化问题，支持的调优模式有所不同：

- 离散优化问题：支持调优求解时间、最优相对容差、目标函数值和目标函数值下界；
- 连续优化问题：支持调优求解时间。

COPT 调优工具的工作流程如下：

1. 首先，进行基准计算，并允许用户自定义基准计算参数；
2. 然后，逐次生成调优参数，通过参数调优计算寻找改进求解性能的参数组合。

16.2 调优工具相关参数

调优工具相关的参数及对应含义，如 表 16.1 所示：

表 16.1: 参数调优相关参数总览

参数名	类型	参数含义
TuneTimeLimit	浮点数参数	参数调优的时间限制
TuneTargetTime	浮点数参数	参数调优的时间目标
TuneTargetRelGap	整数参数	参数调优的最优相容差目标
TuneMethod	整数参数	参数调优的方法
TuneMode	整数参数	参数调优的模式
TuneMeasure	整数参数	参数调优结果计算方式
TunePermutes	整数参数	参数调优每组参数模型计算次数
TuneOutputLevel	整数参数	参数调优日志输出强度

16.3 参数调优功能

COPT 调优工具提供了以下功能：

16.3.1 调优方法

该功能由参数 TuneMethod 控制，提供如下两种搜索方式：

- 贪婪搜索方法：期望以较少次数参数调优计算，寻找较优的参数设置；
- 广泛搜索方法：尝试更多的参数组合，具有更大的搜索空间，更可能寻找到较优的参数设置，但同时也会消耗更多的调优时间。

参数 TuneMethod 的可能取值及对应含义如下，通过设置其为不同取值，以进行搜索方式的选择，默认设置为自动选择。

- -1: 自动选择
- 0: 贪婪搜索策略
- 1: 更广泛的搜索策略

16.3.2 调优模式

该功能由参数 TuneMode 控制，可选项有：求解时间、最优相对容差、目标函数值和目标函数值下界，默认设置为自动选择。不同调优模式选项分别对应该参数的不同取值：

- 0: 求解时间
- 1: 最优相对容差
- 2: 目标函数值
- 3: 目标函数值下界

注意：对于整数规划问题，默认设置下，若基准计算在给定的时间限制内未将模型求至最优，则调优工具将自动将调优模式切换为最优相对容差。

16.3.3 调优扰动次数

该功能由参数 TunePermutes 控制。为了测试模型求解性能的扰动变化，COPT 调优工具允许用户指定每次参数调优计算运行模型的次数，默认设置为自动选择。

16.3.4 调优准则

该功能由参数 `TuneMeasure` 控制，用来选择计算调优结果的计算方式。

当用户每次参数调优计算运行多次模型时，COPT 调优工具将根据设置的调优准则计算合成的调优值。

该参数的可选项有：平均值和最大值，默认设置为自动选择。

- 0: 计算平均值
- 1: 计算最大值

16.3.5 调优目标

该功能由参数 `TuneTargetTime` 和 `TuneTargetRelGap` 控制。

COPT 调优工具允许用户指定求解时间和最优相对容差作为调优目标，当调优工具发现满足指定调优目标的参数后，即自动终止调优。对于求解时间，默认值为 0.01 秒，对于最优相对容差，默认值为 `1e-4` 。

16.3.6 调优日志输出

该功能由参数 `TuneOutputLevel` 控制，默认输出每次调优尝试的摘要。可选项及对应参数取值如下：

- 0: 不输出调优日志
- 1: 仅输出改进参数的摘要
- 2: 输出每次调优尝试的摘要
- 3: 输出每次调优尝试的详细日志

16.3.7 调优时间限制

该功能由参数 `TuneTimeLimit` 控制，它表示参数调优计算的时间限制。默认设置为自动选择。

16.3.8 用户自定义部分

用户可以自定义以下部分：

- 自定义基准参数：

  COPT 调优工具允许用户设置基准计算的参数，在后续的参数调优计算中，这些参数将作为固定参数，COPT 调优工具将不会对固定参数进行调优。

- 自定义整数初始解：

  COPT 调优工具允许用户设置基准计算的初始解，在后续的参数调优计算中，也将使用这些初始解进行计算。

- 自定义调优文件

  COPT 调优工具允许用户从调优文件中读取待选择的参数组合，此时调优工具将以指定的参数组合为依据进行参数调优计算，否则 COPT 调优工具将自动生成参数组合进行调优计算。

注意： COPT 调优文件与 COPT 参数文件格式相似，区别在于调优文件允许单个参数指定多个值。

16.3.9 获取调优结果

参数调优结束后，可以通过属性 `TuneResults` 获取参数调优结果数目，也可以加载指定编号的调优结果到模型或者写入到参数文件中。

COPT 提供输出指定编号的参数调优结果到参数文件（`".par"`）中，需要指定的参数有：

- `idx`： 参数调优结果编号
- `filename`： 文件名称

不同编程接口中对应的函数如下：

表 16.2: 不同接口中写入参数调优结果的函数

编程接口	函数
C	`COPT_WriteTuneParam`
C++	`Model::WriteTuneParam()`
C#	`Model.WriteTuneParam()`
Java	`Model.writeTuneParam()`
Python	`Model.writeTuneParam()`

16.4 参数调优示例

例如，使用 COPT 命令行工具对模型 `"foo.mps"` 进行调优以改进求解时间，相关命令为：

```
copt_cmd -c "read foo.mps; tune; exit"
```

以 Python 接口为例，使用 COPT 调优工具的代码示例如下：

```
env = Envr()
m = env.createModel()
m.read("foo.mps")
m.tune()
```

第 17 章　Callbacks 功能

杉数求解器 COPT 提供 Callbacks（回调）功能，支持用户在混合整数规划分支切割的求解过程中，获取中间信息（如当前最优下界、当前最优目标值等）；以及控制求解进程：如修改模型（如添加惰性约束、添加割平面），终止求解等。目前支持使用 Callbacks 的问题类型有：混合整数规划、混合二阶锥规划、混合整数二次（约束）规划。

本章节的内容构成如下：

- *获取 MIP 求解过程中间信息*
- *控制 MIP 求解进程*
- *不同 API 中使用 Callback 功能*

17.1 获取 MIP 求解过程中间信息

MIP 求解过程中，可获取到的中间信息取决于 Callback Context（Callback 的触发点），支持的 Context 如下：

- `CBCONTEXT_MIPSOL`：当找到 MIP 可行解时，触发回调函数；
- `CBCONTEXT_MIPRELAX`：当找到 MIP 线性松弛解时，触发回调函数；
- `CBCONTEXT_MIPNODE`：当处理完成 MIP 节点并求解 LP 松弛问题完成时，触发回调函数。

COPT 支持获取的 Callback 信息，详细请参考*Callback 相关信息* 部分。

对应关系罗列如下表：

Context	Callback Information
`CBCONTEXT_MIPSOL`	`MipCandObj`, `MipCandidate`
`CBCONTEXT_MIPRELAX`	`RelaxSolution`, `RelaxSolObj`
`CBCONTEXT_MIPNODE`	`NodeStatus`, `RelaxSolution`, `RelaxSolObj`, `MipCandObj`, `MipCandidate`

除此之外的 `BestObj`, `BestBnd`, `HasIncumbent`, `Incumbent` 在任何触发点都可以被获取（即 `CBCONTEXT_MIPSOL`, `CBCONTEXT_MIPRELAX`, `CBCONTEXT_MIPNODE` 均可）。

注意

1. 如果 `HasIncumbent == False`，则无法获取 `Incumbent` 。

2. 信息 NodeStatus 的返回值为常数，表示当前节点 LP 松弛问题的求解状态，可取值请参考：一般常数章节：解状态（部分） 。

2. 对于 **可行解**，**LP 松弛解**，**当前最优可行解**这三项信息，不同接口中的获取方式有所不同：

   - C 语言：通过函数 COPT_GetCallbackInfo ，需获取的中间信息名称作为函数的参数提供；
   - 面向对象的编程语言（C++/C#/Java/Python）中，CallbackBase 类提供专门的函数，以获取相应中间信息。

3. 面向对象的编程语言（C++/C#/Java/Python）中，CallbackBase.getInfo(infoname) 函数，仅支持获取标量类型的信息常数，即：NodeStatus, MipCandObj, RelaxSolObj , BestObj , BestBnd , HasIncumbent 。其他的非标量信息提供专门的函数获取。

   以 Python 接口为例，将对应函数名称罗列如下。其他编程语言接口类似，可参考各 API 的 CallbackBase 类。

   - 获取当前可行解：CallbackBase.getSolution()
   - 获取当前 LP 松弛解：CallbackBase.getRelaxSol()
   - 获取当前最优可行解：CallbackBase.getIncumbent()

17.2 控制 MIP 求解进程

COPT 提供相关函数，让用户在 MIP 分支切割法的求解过程中，交互式地添加惰性约束或割平面，以控制 MIP 求解进程。主要有如下三类操作：

1. 添加惰性约束
2. 添加割平面
3. 设置自定义的可行解

17.2.1 添加惰性约束

COPT 支持用户通过两种方式添加惰性约束，一种是直接向模型中添加，另一种是通过 Callback 在求解过程中，在指定的 Callback Context 中添加。在 C API 中，根据函数名称中是否包含 "Callback" 进行区分；在面向对象的 API 中，分别对应 Model 类和 CallbackBase 的函数。

1. 在求解之前，直接向模型中添加，支持添加单边或双边的惰性约束。以 Python 为例：

   - 添加单个：Model.addLazyConstr()
   - 批量添加多个：Model.addLazyConstrs()

2. 在求解过程中，通过 Callback，根据 Callback Context 动态添加，仅支持添加单边的惰性约束。以 Python 为例：

   - 添加单个：CallbackBase.addLazyConstr()
   - 批量添加多个：CallbackBase.addLazyConstrs()

17.2.2 添加割平面

同样，对于割平面，COPT 也支持上述两种添加方式。

1. 在求解之前，直接向模型中添加，支持添加单边或双边的割平面约束。以 Python 为例：
   - 添加单个：Model.addUserCut()
   - 批量添加多个：Model.addUserCuts()
2. 在求解过程中，通过 Callback，根据 Callback Context 动态添加，仅支持添加单边的割平面约束。以 Python 为例：
   - 添加单个：CallbackBase.addUserCut()
   - 批量添加多个：CallbackBase.addUserCuts()

注意

- 在 Callback 中调用 Model 类添加惰性约束和用户割平面是无效的。

17.2.3 设置自定义的可行解

COPT 的 callback 功能支持在 MIP 求解过程中，添加用户自定义的可行解（可以是用户通过任意方式找到的可行解，例如通过启发式算法）。以 Python 为例：

- 设置自定义的可行解：`CallbackBase.setSolution(vars, val)`
- 将自定义的解加载到模型中：`CallbackBase.loadSolution()`

注意

当前支持设置完整的可行解。

同样，操作修改模型的函数需要在指定的 Callback Context 中才能被调用。

以 python 为例，`CallbackBase` 类中的成员函数，以及对应可调用的 Context 罗列如下：

Context	函数

CBCONTEXT_MIPSOL	`CallbackBase.addLazyConstr`
	`CallbackBase.addLazyConstrs`
	`CallbackBase.getSolution`
	`CallbackBase.setSolution`
	`CallbackBase.getIncumbent`
	`CallbackBase.getInfo`

CBCONTEXT_MIPRELAX	`CallbackBase.addUserCut`
	`CallbackBase.addUserCuts`
	`CallbackBase.getRelaxSol`
	`CallbackBase.getIncumbent`
	`CallbackBase.setSolution`
	`CallbackBase.getInfo`

CBCONTEXT_MIPNODE	`CallbackBase.getRelaxSol`
	`CallbackBase.getIncumbent`
	`CallbackBase.setSolution`
	`CallbackBase.getInfo`

注意

其他编程语言 API 的函数调用形式略有不同，但对应关系是一致的。C 语言章节也罗列了 Callback 相关函数及 Callback Context 的对应关系，详细请参考*C API：回调功能函数* 。

17.3 不同 API 中使用 Callback 功能

以面向对象的编程接口为例，展示在不同的 API 中，调用 Callback 功能的基本步骤：

1. 构建自定义 Callback 类，并继承 `CallbackBase` 类；
2. 实现 `CallbackBase.callback()` 函数，在其中自定义需要进行的操作（获取中间信息或者添加惰性约束/割平面等）；
3. 新建自定义 Callback 实例，并传入用户需要的参数；
4. 通过 Model 类的 `Model.setCallback()` 添加 Callback 实例，并将 Callback Context 作为参数输入。

后续在求解过程中，在指定的 Callback Context 中，则会触发调用 Callback 实例中，用户自定义的 callback 函数。

在各编程语言 API 中的具体实现也类似，以 Python 为例，可以参考安装包中 examples 目录中的的示例代码 `"cb_ex1.py"` 。

回调函数在不同编程接口中的调用方式和函数名称 **略有不同**，但是支持的功能和函数含义是一致的，请进一步参考不同编程接口 API 参考手册的对应章节，以获取具体介绍：

- C API：回调功能函数

- C++ API：*CallbackBase 类*
- C# API：*CallbackBase 类*
- Java API：*CallbackBase 类*
- Python API：*CallbackBase 类*

第 18 章　矩阵建模方式

杉数求解器 COPT 的 Python API 提供矩阵建模方式，支持 NumPy 的多维数组，二维的 NumPy 矩阵，以及 SciPy 列压缩矩阵（ csc_matrix ）和行压缩矩阵（ csr_matrix ）的运算（NumPy 最低版本要求为 1.23，Python 最低版本要求为 3.8），并且可以和常规（标量）变量和常规约束结合，主要提供如下功能：

1. 添加多维变量（MVar 类对象）等相关操作；
2. 构建多维线性表达式（MLinExpr 类对象），添加多维线性约束（MConstr 类对象）等相关操作；
3. 构建多维二次表达式（MQuadExpr 类对象），添加多维凸二次约束（QConstraint 类对象）等相关操作。

18.1 多维变量

1. 添加多维变量 MVar

   MVar 是多维变量相关操作的封装，用户可以调用 Model.addMVar() 向模型中添加任意维度和形状的多维变量 MVar 。除了需要指定参数 shape 形状之外，其余参数与添加普通变量时需指定的一致，包括：lb , ub , vtype , nameprefix 。

   - 添加一维连续型多维变量：x = Model.addMVar(3)
   - 添加二维 3x3 二进制型多维变量：y = Model.addMVar((3,3), vtype=COPT.BINARY)

   此外，还可以对 MVar 多维变量进行切片操作，如：y1 = y[:,0:2]

2. 获取多维变量相关属性

   - 维度数量：MVar.ndim
   - 多维变量的形状：MVar.shape
   - 元素数量：MVar.size

18.2 多维数组运算及表达式

18.2.1 多维线性表达式

多维变量与其系数（可以为 ndarray ）构成多维线性表达式（MLinExpr ），支持的运算主要有：

1. 矩阵乘法运算：A @ x

```
x = model.addMVar(3)
A = np.array([[1, 0, 1],[0, 0, 1]])
expr1 = A @ x
```

2. 向量内积运算

```
x = model.addMVar(3)
c = np.array([1, 2, 3])
expr2 = c @ x
```

18.2.2 多维二次表达式

常见的多维二次表达式及对应的数学形式如下：

- x @ Q @ x：x^TQx
- x @ x：x^Tx
- x @ Q @ x + c @ x + b：$x^TQx + c^Tx + b$

18.2.3 其他多维数组运算

1. 与常规变量、常规线性表达式，以及常数结合：

```
x = model.addMVar(3)
y = model.addVar()
c = np.array([1, 2, 3])
Q = np.full((3, 3), 1)
expr3 = 2 * x @ Q @ x + c @ x + 2 * y + 1
```

2. 自加/自减/自乘运算：

```
mx = m.addMVar((3, 3))
B = np.array([[1, 0, 1], [0, 1, 1]])
expr_add = B @ mx
expr_add += 1
expr_add *= 2
```

注意

- 我们直接打印多维表达式 `print(MLinExpr)/print(MQuadExpr)` 时，同时会输出表达式的形状 `shape` 。当 `shape=()` 时，表示该表达式为标量（单一线性/二次表达式），对应的 `ndim=0` ，`size=1` ，对于多维变量 `MVar` 的 `shape` 也是如此；
- 当进行矩阵乘法运算（A@x）时，需要满足矩阵乘法运算法则，A 的列数和 X 的行数需要相同；
- COPT 支持 `MLinExpr` 类对象和 `LinExpr` 结合使用，但需注意此时的 `MLinExpr` 类对象需要 `shape=()` ，最终返回的表达式为 `MLinExpr` 类对象且 `shape=()` 。

18.3 矩阵约束

18.3.1 矩阵线性约束

COPT 支持添加矩阵线性约束的两种方式，函数参数提供的格式有所不同：

1. 专门添加矩阵线性约束的 `Model.addMConstr()` ，可以指定的参数有：
   - `A` ： 线性约束的系数矩阵
   - `x` ： 决策变量（`MVar` 类对象）
   - `sense` ： 线性约束的类型，可取值有：`'L'` （小于等于）、`'G'` （大于等于）、`'E'` （等于）等
   - `b` ： 线性约束右端项（向量，维度与矩阵 `A` 的行数一致）
   - `name` ： 约束名称

```
x = model.addMVar(shape=3, vtype=COPT.BINARY, nameprefix='x')
A = np.array([[1, 2, 3], [3, 2, 1]])
b = np.array([2, 5])
mconstrs = model.addMConstr(A, x, 'L', b, nameprefix='c')
obj = np.array([1, 2, 1])
model.setObjective(obj @ x, COPT.MINIMIZE)
```

2. 多维线性约束可视为一组线性约束，故 `Model.addConstrs()` 也可以添加多维线性约束：

```
x = model.addMVar(shape=3, vtype=COPT.BINARY, nameprefix='x')
A = np.array([[1, 2, 3], [3, 2, 1]])
b = np.array([2, 5])
mconstrs = model.addConstrs(A @ x <= b, nameprefix='c')
obj = np.array([1, 2, 1])
model.setObjective(obj @ x, COPT.MINIMIZE)
```

18.3.2 二次约束

COPT 支持添加矩阵二次约束的两种方式，函数参数提供的格式有所不同：

1. 专门添加矩阵二次约束的 Model.addMQConstr()，可以指定的参数有：
   - Q：二次项系数矩阵
   - c：线性项系数向量，若无线性项，则为 None
   - sense：二次约束的类型，可取值有：'L'（小于等于）、'G'（大于等于）、'E'（等于）等。
   - rhs：二次约束右端项
   - xQ_L：二次项系数矩阵 Q 左端变量（向量，长度与矩阵 Q 的行数一致）
   - xQ_R：二次项系数矩阵 Q 右端变量（向量，长度与矩阵 Q 的列数一致）
   - xc：线性项的变量，若无线性项，则为 None
   - name：约束名称

```
Q = np.diag([3, 2, 1])
x = model.addMVar(3)
y = model.addMVar(3)
c1 = model.addMQConstr(Q, None, 'E', 1.0, x, y)
```

2. Model.addQConstr()，直接给出多维二次表达式
   - lhs：多维二次表达式
   - sense：约束类型
   - rhs：约束右端项

```
Q = np.diag([3, 2, 1])
x = model.addMVar(3)
y = model.addMVar(3)
c2 = model.addQConstr(lhs=x@Q@y, sense=COPT.EQUAL, rhs=1.0)
```

18.4 由多维变量构成的目标函数

COPT 支持设置线性和二次目标函数，并提供设置由多维变量构成目标函数的两种方式，函数参数的格式有所不同：

1. 专门设置由多维变量构成的目标函数的 Model.setMObjective()，可以指定的参数有：
   - Q：二次项系数矩阵，若目标函数是线性的，则为 None
   - c：线性项系数向量，若无线性项，则为 None
   - constant：目标函数的常数项
   - xQ_L：二次项系数矩阵 Q 左端变量（向量，长度与矩阵 Q 的行数一致），若目标函数是线性的，则为 None

- xQ_R：二次项系数矩阵 Q 右端变量（向量，长度与矩阵 Q 的列数一致），若目标函数是线性的，则为 None
- xc：线性项的变量，若无线性项，则为 None
- sense：优化方向，可取值为：COPT.MINIMIZE 或 COPT.MAXIMIZE

2. 直接给出目标函数表达式的 Model.setObjective()
   - expr：目标函数表达式，可以为线性或者二次表达式
   - sense：优化方向，可取值为：COPT.MINIMIZE 或 COPT.MINIMIZE

```
x = model.addMVar(shape=3, vtype=COPT.BINARY, nameprefix="x")
obj = np.array([1, 2, 1])
model.setObjective(obj @ x, COPT.MINIMIZE)
```

关于矩阵建模方式，COPT Python 接口分别提供多维变量及其（线性和凸二次）表达式、矩阵约束的类，对相关操作进行封装，其中包含的成员方法及具体介绍请参考 Python API 对应部分：

- 多维变量：*MVar 类*
- 多维线性表达式：*MLinExpr 类*
- 二次表达式：*MQuadExpr 类*
- 矩阵约束：*MConstr 类*

第 19 章　求解日志

杉数求解器 COPT 在求解过程中会输出日志，用户可以从其中获取求解结果信息，追踪优化求解的处理过程和迭代步骤。本章将会分别对不同算法的求解日志进行解读，内容构成如下：

- 求解日志相关参数和函数
- 求解日志基础信息部分
- 单纯形法（*Simplex*）求解日志
- 内点法（*Barrier*）求解日志
- 分支切割法（*Branch-and-Cut*）求解日志

19.1 求解日志相关参数和函数

用户可以通过设置求解日志相关参数，来控制是否显示求解日志。

- `Logging`

  整数参数

  是否显示求解日志

  默认值 1

  可选值

  0: 不显示求解日志。

  1: 显示求解日志。

- `LogToConsole`

  整数参数

  是否显示求解日志到控制台

  默认值 1

  可选值

  0: 不显示求解日志到控制台。

  1: 显示求解日志到控制台。

杉数求解器 COPT 提供的和求解日志相关操作有：设置求解日志文件、设置求解日志回调函数。

COPT 提供设置求解日志文件的相关函数，将求解日志写入至指定的文件（文件名后缀为 `.log` ）中，以便用户对日志进行保存。不同编程接口的函数如 表 19.1 所示：

表 19.1: 不同编程接口设置求解日志文件函数一览

编程接口	设置求解日志文件函数
C	`COPT_SetLogFile`
C++	`Model::SetLogFile()`
C#	`Model.SetLogFile()`
Java	`Model.setLogFile()`
Python	`Model.setLogFile()`

注意：在调用该函数时，用户需通过参数 `logfile` 指定保存求解日志的文件名。

19.2 求解日志基础信息部分

杉数求解器 COPT 会根据问题类型运用不同的算法进行求解，在开始求解前均会输出以下基础信息：

```
Model fingerprint: 2c27ab28

Hardware has 64 cores and 128 threads. Using instruction set X86_AVX512_E1 (14)
Minimizing a MIP problem
```

其中，`Model fingerprint` 当前求解模型的唯一编码；

接下来输出的是求解所使用的硬件信息，包括：CPU 核数（`cores` ）和线程数（`threads` ）等；

最后是问题类型和优化方向，如：

`Minimizing an LP problem` 、`Minimizing a MIP problem` 和 `Minimizing an SDP problem` 等。

19.3 单纯形法（Simplex）求解日志

按照求解过程的不同阶段，单纯形法求解日志可以划分为以下 3 个部分：

1. 预求解（Presolve）
2. 单纯形法求解过程
3. 求解结果汇总

这里将会以 afiro.mps 算例的求解日志为例，对单纯形法求解日志中的信息进行解读。

19.3.1 预求解（Presolve）

默认参数设置下，使用单纯形法开始求解前，杉数求解器 COPT 会对模型进行预求解，以改善模型质量，传给求解器 COPT 的是经过预处理后的模型。

日志的预求解（Presolve）部分会输出预求解前后模型规模的变化：

```
The original problem has:
    27 rows, 32 columns and 83 non-zero elements
The presolved problem has:
    7 rows, 10 columns and 28 non-zero elements
```

LP 问题规模描述包括以下信息项：

- 约束（rows ）数目
- 变量（columns ）数目
- 系数矩阵非零元素（non-zero elements ）数目

以上的求解日志为例，经过预求解后，约束和变量数目，以及系数矩阵非零元素数目都有一定缩减。

19.3.2 单纯形法求解过程

此部分日志提供运用单纯形法求解迭代过程的相关信息。

```
Starting the simplex solver using up to 8 threads

Method   Iteration           Objective  Primal.NInf   Dual.NInf        Time
Dual             0   -4.8553789460e+02            3           0       0.00s
Dual             3   -4.6476735494e+02            0           0       0.00s
Postsolving
Dual             3   -4.6475314286e+02            0           0       0.00s
```

其中，首行信息输出显示当前求解算法为单纯形法，使用了 8 线程（threads ）进行计算。

接下来是求解迭代过程，由 6 列信息项构成：

- Method： 使用的求解算法，"Dual" 表示对偶单纯形法（Dual Simplex ）
- Iteration： 迭代次数
- Objective： 目标函数值
- Primal.NInf：原始问题中不可行个数
- Dual.NInf： 对偶问题中不可行个数
- Time： 求解使用时间（单位：秒）

19.3.3 求解结果汇总

此部分日志是在求解完毕后，对模型求解结果和单纯形法迭代过程进行总结。

```
Solving finished
Status: Optimal  Objective: -4.6475314286e+02  Iterations: 3  Time: 0.00s
```

包括的信息项有：

- 求解状态（Status ）：若模型有最优解，则为 Optimal
- 目标函数值（Objective ）：若模型有最优解，Objective 则显示最优目标函数值
- 总迭代数目（Iterations ）
- 总求解时间（Time ）

若模型不可行，对应日志输出如下：

```
Solving finished
Status: Infeasible  Objective: -  Iterations: 2  Time: 0.00s
```

19.4 内点法（Barrier）求解日志

按照求解过程的不同阶段，内点法求解日志可以划分为以下 3 个部分：

1. 预求解（Presolve）
2. 内点法求解过程
3. 求解结果汇总

注意：通过设置优化参数 "LpMethod = 2"，可以选择求解线性规划问题的算法为内点法。

同样，以 afiro.mps 算例的求解日志为例，对内点法求解 LP 问题日志中的信息进行解读。

19.4.1 预求解（Presolve）

默认参数设置下，使用内点法开始求解前，杉数求解器 COPT 会对模型进行预求解，以改善模型质量，传给求解器 COPT 的是经过预处理后的模型。

日志的预求解（Presolve）部分会输出预求解前后模型规模的变化：

```
The original problem has:
    27 rows, 32 columns and 83 non-zero elements
The presolved problem has:
    7 rows, 10 columns and 28 non-zero elements
```

19.4.2 模型信息部分

此部分日志呈现模型的相关数值信息：

```
Starting barrier solver using 64 threads

Problem info:
Dualized in presolve:            No
Range of matrix coefficients:    [4e-01,4e+00]
Range of rhs coefficients:       [8e+01,3e+02]
Range of bound coefficients:     [4e+01,1e+02]
Range of cost coefficients:      [2e-01,2e+00]

Factor info:
Number of free columns:          0
Number of dense columns:         0
Number of matrix entries:        2.800e+01
Number of factor entries:        2.800e+01
Number of factor flops:          1.140e+02
```

其中，首行信息输出显示当前求解算法为内点法，使用了 64 线程（`threads` ），其中：

- `Problem info` 输出信息包括：是否求解对偶化模型、模型系数矩阵范围、右端项范围、约束变量边界范围和目标函数系数范围；
- `Factor info` 输出信息包括：无边界变量数目、致密列数目、线性系统矩阵分解相关信息。

19.4.3 内点法（Barrier）求解过程

这部分日志内容呈现了杉数求解器 COPT 运用内点法求解的迭代过程，包含的信息项有：迭代（`Iter` ）次数、目标函数值、求解时间（`Time` ）等。各信息项的解释如下：

- `Iter`： 迭代次数
- `Primal.Obj`： 原始问题目标函数值
- `Dual.Obj`： 对偶问题目标函数值
- `Compl`：互补性冲突值（Complementarity violation）
- `Primal.Inf`： 原始问题不可行的程度
- `Dual.Inf`： 对偶问题不可行的程度
- `Time`： 求解使用时间（s）

```
Iter       Primal.Obj         Dual.Obj      Compl  Primal.Inf  Dual.Inf    Time
   0  +2.07010046e+01  -4.97632246e+02   5.89e+03    4.50e+02  2.65e+00   0.02s
   1  -1.18912241e+02  -5.91808560e+02   7.58e+02    3.36e+01  1.61e-01   0.02s
   2  -3.98096520e+02  -4.77597371e+02   2.28e+02    9.32e+00  7.45e-02   0.02s
   3  -4.55223227e+02  -4.68222895e+02   1.86e+01    3.60e-01  2.63e-03   0.02s
```

（续下页）

(接上页)

```
   4  -4.64587726e+02  -4.64803786e+02   2.52e-01    7.80e-03  7.93e-06   0.02s
   5  -4.64753143e+02  -4.64753143e+02   3.11e-07    7.80e-09  1.56e-11   0.02s
```

19.4.4 内点法（Barrier）求解总结

主要信息包括：求解状态、原始和对偶问题最优目标函数值等。

```
Barrier status:                 OPTIMAL
Primal objective:               -4.64753143e+02
Dual objective:                 -4.64753143e+02
Duality gap (abs/rel):          2.61e-07 / 5.63e-10
Primal infeasibility (abs/rel): 7.80e-09 / 2.60e-11
Dual infeasibility (abs/rel):   1.56e-11 / 6.99e-12
```

Crossover 过程

```
Starting crossover using up to 8 threads

    1 primal pushes remaining      0.03s
    0 primal pushes remaining      0.03s
    1 dual pushes remaining        0.03s
    0 dual pushes remaining        0.03s

Method   Iteration           Objective  Primal.NInf   Dual.NInf        Time
Dual             1   -4.6475314286e+02            0           0       0.03s
Postsolving
Dual             1   -4.6475314286e+02            0           0       0.03s
```

19.4.5 求解结果汇总

此部分日志是在求解完毕后，对模型求解结果和单纯形法迭代过程进行总结。

```
Solving finished
Status: Optimal  Objective: -4.6475314286e+02  Iterations: 1  Time: 0.03s
```

和单纯形法求解日志一样，此处包括的信息项有：

- 求解状态（Status ）：若模型有最优解，则为 Optimal
- 目标函数值（Objective ）：若模型有最优解，则 Objective 显示最优目标函数值
- 总迭代数目（Iterations ）
- 总求解时间（Time ）

19.5 分支切割法（Branch-and-Cut）求解日志

按照求解过程的不同阶段，分支切割法求解日志的内容可划分为 3 个部分：

1. 预求解（Presolve）
2. 分支切割法搜索求解过程
3. 结果汇总

这里将会以 binkar10_1.mps.gz 算例的求解日志为例，对 MIP 求解日志中的信息进行解读。

19.5.1 预求解（Presolve）

为了简化模型，杉数求解器 COPT 会对 MIP 模型进行预求解，以去除冗余的约束或变量范围。交给 COPT 的是经过预处理后的模型，接下来会基于预处理后的模型，开始使用分支切割法进行求解。

Presolve 日志中输出的是预求解前后模型规模的变化：包括原始问题规模和预求解后问题规模的变化。

```
The original problem has:
    1026 rows, 2298 columns and 4496 non-zero elements
    170 binaries

Presolving the problem

The presolved problem has:
    810 rows, 1417 columns and 3384 non-zero elements
    170 binaries
```

MIP 问题规模描述包括以下信息项：

- 约束（rows ）数目
- 变量（columns ）数目
- 系数矩阵非零元素（non-zero elements ）数目
- 0-1 变量（binaries ）数目
- 整数变量（integers ）数目

注意

- 这里的变量数目（cols ）统计的是模型中全部变量的个数，包括连续性变量、整数型变量和 0-1 变量；
- 在理论上来说，0-1 变量属于特殊的整数型变量，但是此处二者是分开统计的。

以上的求解日志为例，经过预处理后，问题规模（约束和变量数目）得到了缩减。

19.5.2 求解过程

此部分日志输出了分支切割法 (Branch-and-Cut) 搜索求解的过程。

```
Starting the MIP solver with 128 threads and 256 tasks

    Nodes    Active  LPit/n  IntInf     BestBound  BestSolution    Gap   Time
        0         1      --       0 -1.600066e+07            --    Inf  0.12s
H       0         1      --       0 -1.600066e+07  1.010642e+06 106.3%  0.13s
H       0         1      --       0 -1.600066e+07  1.121484e+04 100.1%  0.13s
H       0         1      --       0 -1.600066e+07  1.117324e+04 100.1%  0.13s
        0         1      --      38  6.637188e+03  1.117324e+04  40.6%  0.24s
     3000       366    36.4      67  6.734588e+03  6.742200e+03  0.11%  8.37s
     4000         6    31.0      47  6.740212e+03  6.742200e+03  0.03%  8.84s
     4080         0    30.6      47  6.742200e+03  6.742200e+03  0.00%  8.86s
```

注：由于篇幅限制，此处只呈现求解日志的部分内容，目的是为了日志内容解读的需要。

按照信息项的含义，可将求解过程日志分为以下三部分内容，下面会分别解读各信息项含义：

- 节点（Nodes ）搜索信息（1-4 列）
- 可行解区间信息（5-7 列）
- 求解时间信息（第 8 列）

节点搜索信息（1-4 列）：

Nodes：已搜索过的节点个数

Active：尚未被搜索的叶子节点个数

LPit/n：每个节点单纯形法（Simplex）迭代平均次数

IntInf：当前线性松弛 (LP relaxtion) 的解中尚未取到整数值的整数变量个数

可行解区间信息（5-7 列）：

BestBound：当前最优的目标边界

BestSolution：当前最优的目标函数值

Gap：上下界之间的相对容差，若小于参数 RelGap 的值，将会停止求解

求解时间信息（第 9 列）：

Time：求解所用时间

注意：

- Nodes 第 1 列前的标记 (H/*) 表示找到了一个新的可行解。
  - H：通过启发式（heuristic）方法找到；
  - *：通过分支（branching）求解子问题的方法找到。

- 有时会看到 Nodes (已搜索过的节点个数) 长时间为 0，这说明 COPT 在处理根节点。在根节点做的工作主要有：产生割平面以及尝试多种启发式方法，以获取最优可行解，目的是减小后续搜索的规模。

19.5.3 求解结果汇总

这部分是求解完毕后，对 MIP 问题的最终求解状态和分支切割法的搜索过程进行总结，主要包括：模型求解结果和求解搜索工作量两部分内容。

```
Best solution   : 6742.200024746
Best bound      : 6742.200024746
Best gap        : 0.0000%
Solve time      : 8.87
Solve node      : 4080
MIP status      : solved
Solution status : integer optimal (relative gap limit 0.0001)

Violations    :     absolute     relative
bounds        :        1e-08        1e-08
rows          :  6.25278e-13  6.25278e-13
integrality   :            0
```

求解结果：

- 最优目标函数值（Best solution ）
- 最优下界（Best bound ）
- 最优容差（Best gap ）
- 求解状态（Solution status ）

求解搜索工作量：

- 求解时间（Solve time）
- 搜索节点个数（Solve node）

其中 Violations 部分表示最优解对模型约束和变量范围的满足程度，其中包括：

- 变量（rows ）和约束（bounds ）的冲突值
- 变量整数解（integrality ）冲突值

第 20 章　文件格式

20.1 文件格式一览

目前杉数求解器 COPT 支持的文件格式如 表 20.1 所示。

表 20.1: 支持的文件格式

文件格式	文件后缀
MPS 格式模型文件	`.mps`, `.mps.gz`
LP 格式模型文件	`.lp`, `.lp.gz`
SDPA 格式模型文件	`.dat-s`, `.dat-s.gz`
CBF 格式模型文件	`.cbf`, `.cbf.gz`
COPT 二进制格式文件	`.bin`, `.bin.gz`
IIS 文件	`.iis`
可行化松弛文件	`.relax`
基解文件	`.bas`
结果文件	`.sol`
初始解文件	`.mst`
参数文件	`.par`
调参文件	`.tune`

20.2 文件读写操作

可以通过调用相关函数，用户可以将外部模型文件输入给 COPT 进行读取；同时，也可以将在 COPT 中构建的模型、优化的结果等输出保存下来，进行文件的写出。

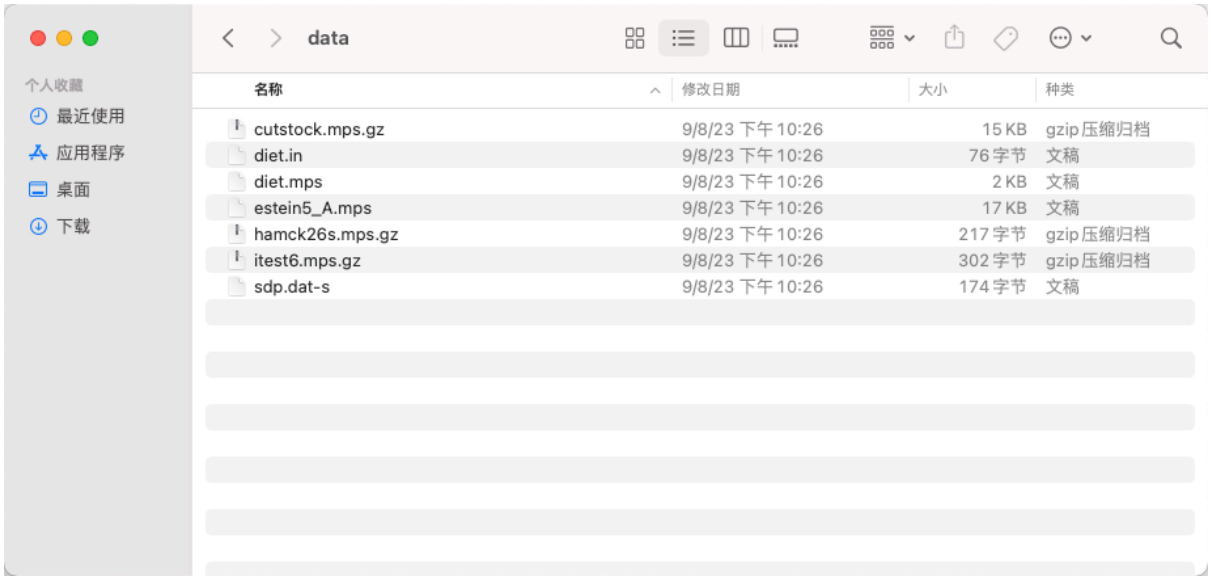
我们以读取或者写出当前目录下的 mps 格式的模型为例（其他文件也是类似操作），在不同接口中实现方式如 表 12.7 所示：

表 20.2: 读写文件的函数

接口	读取	写出
COPT 命令行工具	`read example.mps`	`write example.mps`
C	`COPT_ReadMps`	`COPT_WriteMps`
Python	`Model.read()` / `Model.readMps()`	`Model.write()` / `Model.writeMps()`
C++	`Model::read()` / `Model::readMps()`	`Model::write()` / `Model::writeMps()`
C#	`Model.Read()` / `Model.ReadMps()`	`Model.Write()` / `Model.WriteMps()`
Java	`Model.read()` / `Model.readMps()`	`Model.write()` / `Model.writeMps()`

20.3 模型文件介绍

用户可以在安装包的 `"examples/data"` 目录下找到 COPT 自带的模型文件实例。此处我们介绍常见的两种模型文件格式具体内容：MPS 和 LP。



MPS 格式

MPS 是通用的模型文件标准格式，不同类型的优化问题都可以输出为 mps 格式存储下来，在求解器软件中被广泛使用。

如下是一个 MPS 格式的模型文件示例：

```
NAME          COPTPROB
OBJSENSE
    MAX
ROWS
N  __OBJ___
L  R0000000
```

(续下页)

(接上页)

```
G  R0000001
COLUMNS
    x         __OBJ___  1.2
    x         R0000000  1.5
    x         R0000001  0.80000000000000004
    y         __OBJ___  1.8
    y         R0000000  1.2
    y         R0000001  0.59999999999999998
    z         __OBJ___  2.1000000000000001
    z         R0000000  1.8
    z         R0000001  0.90000000000000002
RHS
    RHS       R0000000  2.6000000000000001
    RHS       R0000001  1.2
BOUNDS
LO BOUND     x         0.10000000000000001
UP BOUND     x         0.59999999999999998
LO BOUND     y         0.20000000000000001
UP BOUND     y         1.5
LO BOUND     z         0.29999999999999999
UP BOUND     z         2.7999999999999998
ENDATA
```

我们可以看到 MPS 中主要包括几个部分：NAME、OBJSENSE、ROWS、COLUMNS、RHS、BOUNDS。

1. NAME: 模型的名称
2. OBJSENSE: 目标函数的优化方向
3. ROWS: 模型中约束及其方向（L 表示 <= 约束；G 表示 >= 约束；N 表示无边界）
4. COLUMNS: 模型中变量及其系数
5. RHS: 约束的右端项取值
6. BOUNDS: 变量的边界（LO 表示下界，UP 表示上界，FR 表示无边界）

注意

1. ROWS 部分中，第一行 `__OBJ___` 表示目标函数；
2. COLS 部分中，形如 `x __OBJ___ 1.2` ，表示变量 x 在目标函数中的系数为 1.2；
3. 在 MPS 格式中，整数型变量会通过如下的字段进行标识：
   - 第一个整数变量：`MARKER 'MARKER' 'INTORG'`
   - 最后一个整数变量：`MARKER 'MARKER' 'INTEND'`

LP 格式

LP 格式更加贴近于代数形式，与 MPS 相比，可读性更高，并且能够容易地对应到其原始的数学模型。

如下是一个 LP 格式的模型文件示例：

```
\Generated by Cardinal Operations

Maximize
1.2 x + 1.8 y + 2.1 z
Subject To
1.5 x + 1.2 y + 1.8 z <= 2.6
0.8 x + 0.6 y + 0.9 z >= 1.2
Bounds
0.1 <= x <= 0.6
0.2 <= y <= 1.5
0.3 <= z <= 2.8
END
```

我们可以看到 LP 中主要包括几个部分：目标函数（Maximize）、约束条件（Subject To）、变量范围（Bounds）。

注意

1. 有时在 LP 格式中，我们会看到形如 `x#1` 的变量名称，这个是标记 x 是第 1 个变量。当用户没有指定变量名称时，这是输出 lp 模型文件时自动生成的名称；
2. 如果变量中有二进制类型的，会通过 Binaries 字段进行标识。

第 21 章　常见问题

21.1 安装和许可配置相关

- **问:** 配置许可时报错 `invalid username` 是什么原因?

  答: 此错误表明在申请时 username 填写错误，使用正确的 username 重新填表申请即可。关于不同操作系统下如何获取 username，请查看 杉数官网求解器 COPT 申请页 的操作指引，请在申请原因中备注 **“用户名填写错误重新申请”**，我们会为您重新发放正确用户名的许可。

- **问:** 下载 COPT 后，计算机安装的某杀毒软件报告存在病毒，自动将其隔离。

  答: 从 COPT 官方下载链接下载的 COPT 软件为正式版本，COPT 软件开发时未使用任何可疑的病毒行为操作，可以确定是杀毒软件误报，请用户暂时关闭杀毒软件后再下载即可。

- **问:** 验证许可（执行 `copt_licgen -v` ）后报错：`Missing Files` 或 `Invalid Signature` 。

  答: 此类错误表明许可文件配置失败，请参考安装说明：配置许可文件 检查是否按照许可文件配置的步骤进行，常见原因如下：

  1. 当前工作目录下的许可文件和 COPT 的版本不适配（如：许可文件是 4.0 版本，而 COPT 是 5.0 版本的），请检查 `"license.dat"` 中的 `VERSION` 和 COPT 的大版本号是否一致，若不一致请前往 杉数官网求解器 COPT 申请页 重新申请，我们会为您发放最新的许可。
  2. Windows 系统中，若 COPT 软件安装在系统盘（一般为 `C` 盘）非用户目录下（如：默认安装路径 `"C:\Program Files\copt70"` ，则需要 **以管理员权限**打开 Windows 命令行窗口后，再执行许可获取命令 `copt_licgen` 。

- **问:** 原先已安装了旧版本的 COPT Python 接口（`coptpy` ），现在如何升级到新版本呢?

  答: 请在终端输入：`pip install --upgrade coptpy` 。

- **问:** 在 Python IDE 中，调试程序时报告变量 `not defined` 警告，或代码中变量的下方出现波浪线，是什么原因呢?

  答: 目前 `coptpy` 已经支持 type hints，请在终端输入 `pip install coptpy-stubs` 下载安装 `coptpy-stubs` 以解决此问题；安装成功后，在 IDE 中编写代码时，会提示变量名补全及函数参数可取值。

21.1.1 MacOS 系统

- **问:** MacOS 系统中调用 coptpy 报错：`ImportError: from .coptpywrap import * symbol not found in flat namespace` 。

  答: 这种错误是因为在 MacOS 系统中，coptpy 和 Anaconda 的架构不匹配。如 coptpy 是 M1 版本的，而 Anaconda 是 x86 版本的，可以安装 M1 版本的 Anaconda 以解决此问题。

- **问:** MacOS 系统中配置许可时，在终端执行 `copt_licgen` 命令，报错：`command not found: copt_licgen` 。

  答: 这种错误是因为没有配置 COPT 的相关环境变量，在 MacOS 系统中，安装 COPT 后还需配置环境变量，请参考*安装说明：MacOS 系统* 章节获取详细的安装指引。

- **问:** 在手动配置环境变量时，从文档中直接复制需添加内容至 `.zshrc` 文件或者 `.bash_profile` 文件中，导致配置失败。

  答: 由于文档编码问题，不能直接复制上述环境变量至相应文件中，需要手动输入环境变量的内容。

21.1.2 Windows 系统

- **问:** 在 Windows 系统中，执行 `copt_licgen` 生成许可文件时，报错无法写入许可文件到硬盘，错误信息如：`error opening file` 。

  答: 若 COPT 软件安装在系统盘（一般为 C 盘）非用户目录下（如：默认安装路径 `"C:\Program Files\copt65"` ，则需要 **以管理员权限**打开 Windows 命令行，再执行许可获取命令 `copt_licgen` ，才能正常写入许可文件到 C 盘。对于用户目录如 `"C:\Users\shanshu"` 下执行许可获取命令，则不需要管理员权限。

- **问:** 在 Windows 系统中，通过 `pip install coptpy` 的方式安装 COPT Python 接口时报错：`could not find a version, no matching distribution` ，是什么原因?

  答: 对请勿使用通过 Microsoft Store 安装的 Python，建议从 Anaconda 发行版 或者 Python 官方发行版 下载 Python

- **问:** 在 Windows 系统中，通过 COPT 安装包的方式（`python setup.py install` ）安装 COPT Python 接口，报错 `could not create build` 。

  答: 若 COPT 安装在系统盘（一般为 C 盘）非用户目录下（如：默认安装路径 `"C:\Program Files\copt70"`，则需要先 **以管理员权限**打开命令行窗口后，再执行命令 `python setup.py install` 。

21.2 建模求解和功能使用相关

- **问：** 如何处理模型不可行的情况呢？

  答： COPT 提供计算 IIS 和可行化松弛的功能以分析模型不可行的原因：计算 IIS 会获得最小不可行约束和变量集合，可行化松弛尝试以最小的改动让模型变得可行。详细介绍和用法请查看不可行模型处理 章节。

- **问：** 使用 COPT 的矩阵建模方式，在添加矩阵变量时，报错 `ValueError: cannot create object arrays from iterator`. 是什么原因？

  答： COPT Python 支持的矩阵建模功能有最低版本要求，`NumPy` 最低版本要求为 1.23，`Python` 最低版本要求为 3.8）。可以通过 `pip install --upgrade numpy` 将 `NumPy` 升级至最新版本。

第 22 章　C API 参考手册

杉数优化求解器提供了适用于高级应用场景的 C 语言 API 库函数。本章节提供在 `copt.h` 中定义的常量、函数、参数、属性的文档。

22.1 常量

COPT 有三种类型的常量。

1. 用来构建模型的，比如优化方向，约束类型等；
2. 用来查询求解结果的，比如 API 函数返回值，基、解值等；
3. 用来监视求解进程的，比如回调函数的触发点。

22.1.1 优化方向

在不同的优化场景中，可能需要最大化或者最小化一个目标函数。为此，我们提供了两种优化方向常数：

- `COPT_MINIMIZE`

  最小化目标函数

- `COPT_MAXIMIZE`

  最大化目标函数

当读取一个文件时，优化方向会自动设置。此外，可以调用 `COPT_SetObjSense` 手动设置优化方向。

22.1.2 无边界

在 COPT 中，我们使用一个很大的量表达无边界的情况。这个量可以通过 `COPT_DBLPARAM_INFBOUND` 参数设置。而这个参数的默认值，以常数提供：

- `COPT_INFINITY`

  代表无边界的量的默认值 (`1e30`)

22.1.3 未定义

在 COPT 中，我们使用另一个很大的量表达数据未定义的情况。例如，初始解文件 (.mst) 中省略的变量的赋值。这个量，以常数提供：

- COPT_UNDEFINED

  代表数据未定义 (1e40)

22.1.4 约束类型

注意： COPT 支持使用约束类型来定义约束，但不推荐。我们推荐直接使用上下边界来定义约束。

在优化领域最初兴起的时候，人们往往用约束类型 (**sense**) 来定义一个约束。常见的类型有：

- COPT_LESS_EQUAL

  形如 $g(x) \leq b$ 的约束

- COPT_GREATER_EQUAL

  形如 $g(x) \geq b$ 的约束

- COPT_EQUAL

  形如 $g(x) = b$ 的约束

此外，还有两种用的较少的类型

- COPT_FREE

  无边界约束的表达式

- COPT_RANGE

  同时有上下边界的，形如 $l \leq g(x) \leq u$ 的约束

  请参考 COPT_LoadProb 函数的文档，以了解如何使用 COPT_RANGE 来定义同时有上下界的约束。

22.1.5 变量类型

变量类型指的是一个变量是连续变量、二进制变量或者整数变量。

- COPT_CONTINUOUS

  连续变量

- COPT_BINARY

  二进制变量

- COPT_INTEGER

  整数变量

22.1.6 SOS 约束类型

SOS 约束（Special Ordered Set）是一类限制一组变量取值的特殊约束。目前，COPT 支持两种 SOS 约束，一是 SOS1 约束，该类型约束中指定的一组变量至多有一个变量可取非零值，二是 SOS2 约束，该类型约束中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。SOS 约束中的变量类型可取连续变量、二进制变量和整数变量。

- `COPT_SOS_TYPE1`

  SOS1 约束

- `COPT_SOS_TYPE2`

  SOS2 约束

22.1.7 Indicator 约束

Indicator 约束是一类逻辑关系约束，它以一个二进制类型变量 y 作为 Indicator 变量，根据变量 y 的取值决定线性约束 $a^T x \leq b$ 是否成立。Indicator 约束的一般表达式：

$$y = f \rightarrow a^T x \leq b \tag{22.1}$$

其中，$f \in \{0, 1\}$ 。当 $y = f$ 时，线性约束成立。当 $y \neq f$ 时，线性约束无效（可以被违反）。线性约束的方向可取 $\leq$ 、$\geq$ 和 $=$ 三种情形。

22.1.8 二阶锥约束

二阶锥约束是一类特殊的二次约束，包括：

- `COPT_CONE_QUAD`

  标准二阶锥。

数学形式为：

$$Q^n = \left\{ x \in \mathbb{R}^n \;\middle|\; x_0 \geq \sqrt{\sum_{i=1}^{n-1} x_i^2}, x_0 \geq 0 \right\} \tag{22.2}$$

- `COPT_CONE_RQUAD`

  旋转二阶锥。

数学形式为：

$$Q_r^n = \left\{ x \in \mathbb{R}^n \;\middle|\; 2x_0 x_1 \geq \sum_{i=2}^{n-1} x_i^2, x_0 \geq 0, x_1 \geq 0 \right\} \tag{22.3}$$

22.1.9 基状态

对于一个有 n 个变量、m 个约束的优化模型，它在内部计算时，m 个约束会被当做 m 个松弛变量看待。这样一共就有 $n+m$ 个变量了。

当使用单纯形法求解模型时，它会把 n 个变量固定在其有限边界上，并计算另外 m 个变量的取值。这 m 个计算取值的变量叫做 **基**变量，而另外那 n 个变量叫做 **非基**变量。单纯形法的求解过程，以及最终的解，都可以用变量的基状态来表达。

COPT 中的变量基状态有：

- `COPT_BASIS_LOWER`

  非基变量，取值下边界。

- `COPT_BASIS_BASIC`

  基变量。

- `COPT_BASIS_UPPER`

  非基变量，取值上边界。

- `COPT_BASIS_SUPERBASIC`

  非基变量，但取值非上下边界。

- `COPT_BASIS_FIXED`

  非基变量，固定在它唯一的边界（上下边界相等）。

22.1.10 LP 的解状态

求解线性规划模型之后，其解的状态可以叫做 LP 的解状态。这个值可以通过 `COPT_INTATTR_LPSTATUS` 属性获取。

LP 的解状态的可能取值为：

- `COPT_LPSTATUS_UNSTARTED`

  尚未开始求解。

- `COPT_LPSTATUS_OPTIMAL`

  找到了最优解。

- `COPT_LPSTATUS_INFEASIBLE`

  模型是无解的。

- `COPT_LPSTATUS_UNBOUNDED`

  目标函数在优化方向没有边界。

- `COPT_LPSTATUS_NUMERICAL`

  求解遇到数值问题。

- `COPT_LPSTATUS_TIMEOUT`

在时间限制到达前未能完成求解。

- COPT_LPSTATUS_UNFINISHED

  求解终止。但是由于数值问题，求解器无法给出结果。

- COPT_LPSTATUS_IMPRECISE

  求解结果不准确。

- COPT_LPSTATUS_INTERRUPTED

  用户中止。

22.1.11 MIP 的解状态

求解整数规划模型之后，其解的状态叫做 MIP 的解状态。这个值可以通过 COPT_INTATTR_MIPSTATUS 属性获取。

MIP 的解状态的可能取值为：

- COPT_MIPSTATUS_UNSTARTED

  尚未开始求解。

- COPT_MIPSTATUS_OPTIMAL

  找到了最优解。

- COPT_MIPSTATUS_INFEASIBLE

  模型是无解的。

- COPT_MIPSTATUS_UNBOUNDED

  目标函数在优化方向没有边界。

- COPT_MIPSTATUS_INF_OR_UNB

  模型无解或目标函数在优化方向没有边界。

- COPT_MIPSTATUS_NODELIMIT

  在节点限制到达前未能完成求解。

- COPT_MIPSTATUS_TIMEOUT

  在时间限制到达前未能完成求解。

- COPT_MIPSTATUS_UNFINISHED

  求解终止。但是由于数值问题，求解器无法给出结果。

- COPT_MIPSTATUS_INTERRUPTED

  用户中止。

22.1.12 回调函数的触发点

- COPT_CBCONTEXT_MIPRELAX

  当发现 MIP 线性松弛解时，触发回调函数。

- COPT_CBCONTEXT_MIPSOL

  当发现 MIP 可行解时，触发回调函数。

- COPT_CBCONTEXT_MIPNODE

  当处理完成 MIP 节点并求解 LP 松弛问题完成时，触发回调函数。

22.1.13 API 函数的返回值

当一个 API 函数的调用完成之时，它会返回一个整数返回值。这个值的数值，代表这次函数调用是否成功或失败，以及为何失败。

可能的函数返回值有：

- COPT_RETCODE_OK

  调用成功。

- COPT_RETCODE_MEMORY

  调用失败：内存分配失败。

- COPT_RETCODE_FILE

  调用失败：文件读写失败。

- COPT_RETCODE_INVALID

  调用失败：非法数据。

- COPT_RETCODE_LICENSE

  调用失败：授权检测失败。在这种情况下，请接着调用 COPT_GetLicenseMsg 以获取具体原因。

- COPT_RETCODE_INTERNAL

  调用失败：内部错误。

- COPT_RETCODE_THREAD

  调用失败：线程操作错误。

- COPT_RETCODE_SERVER

  调用失败：远程服务器错误。

- COPT_RETCODE_NONCONVEX

  调用失败：模型非凸。

22.1.14 客户端配置参数

对于浮动和集群服务器的客户端，用户可以通过调用接口函数设置客户端配置参数，目前提供的配置参数有：

- COPT_CLIENT_CLUSTER

  远程服务器的 IP 地址。

- COPT_CLIENT_FLOATING

  令牌服务器的 IP 地址。

- COPT_CLIENT_PASSWORD

  远程服务器的密码。

- COPT_CLIENT_PORT

  令牌服务器的通信端口。

- COPT_CLIENT_WAITTIME

  客户端连接等待时间。

22.1.15 其他常量

- COPT_BUFFSIZE

  定义了获取文本信息的时候，所用到的数组的推荐长度。可在调用 COPT_GetBanner 、COPT_GetRetcodeMsg 等函数时使用。

22.2 属性

在 C API 中，提供获取相关属性取值的 2 个函数如下，具体请参考*C API 函数：获取属性章节* 。

- COPT_GetIntAttr ： 获取整数属性取值
- COPT_GetDblAttr ： 获取浮点型属性取值

注意：在 C API 中，以属性 Cols 为例，属性名称可以有两种形式：COPT_INTATTR_COLS 或 "Cols"

22.2.1 优化模型相关属性

注意： Double 这个词的准确翻译应是双精度浮点数。下文为了简便，称之为浮点数。

- COPT_INTATTR_COLS 或 "Cols"

  整数属性。

  变量（系数矩阵列）的个数。

- COPT_INTATTR_PSDCOLS 或 "PSDCols"

整数属性。

半定变量的个数。

- COPT_INTATTR_ROWS 或 "Rows"

  整数属性。

  约束（系数矩阵行）的个数。

- COPT_INTATTR_ELEMS 或 "Elems"

  整数属性。

  系数矩阵的非零元素个数。

- COPT_INTATTR_QELEMS 或 "QElems"

  整数属性。

  二次目标函数中非零二次项个数。

- COPT_INTATTR_PSDELEMS 或 "PSDElems"

  整数属性。

  目标函数中半定项个数。

- COPT_INTATTR_SYMMATS 或 "SymMats"

  整数属性。

  模型中对称矩阵的个数。

- COPT_INTATTR_BINS 或 "Bins"

  整数属性。

  二进制变量（列）的个数。

- COPT_INTATTR_INTS 或 "Ints"

  整数属性。

  整数变量（列）的个数。

- COPT_INTATTR_SOSS 或 "Soss"

  整数属性。

  SOS 约束的个数。

- COPT_INTATTR_CONES 或 "Cones"

  整数属性。

  二阶锥约束的个数。

- COPT_INTATTR_QCONSTRS 或 "QConstrs"

  整数属性。

  二次约束的个数。

- COPT_INTATTR_PSDCONSTRS 或 "PSDConstrs"

  整数属性。

  半定约束的个数。

- COPT_INTATTR_LMICONSTRS 或 "LMIConstrs"

  整数属性。

  LMI 约束的个数。

- COPT_INTATTR_INDICATORS 或 "Indicators"

  整数属性。

  Indicator 约束的个数。

- COPT_INTATTR_OBJSENSE 或 "ObjSense"

  整数属性。

  优化方向。

- COPT_DBLATTR_OBJCONST 或 "ObjConst"

  浮点数属性。

  目标函数的常数部分。

- COPT_INTATTR_HASQOBJ 或 "HasQObj"

  整数属性。

  模型是否包含二次项目标函数。

- COPT_INTATTR_HASPSDOBJ 或 "HasPSDObj"

  整数属性。

  模型的目标函数是否包含半定项。

- COPT_INTATTR_ISMIP 或 "IsMIP"

  整数属性。

  模型是否为整数规划模型。

22.2.2 求解结果相关属性

- COPT_INTATTR_LPSTATUS 或 "LpStatus"

  整数属性。

  线性规划求解状态。请参考相关的求解状态常数文档常量：*LP 的解状态*。

- COPT_INTATTR_MIPSTATUS 或 "MipStatus"

  整数属性。

  整数规划求解状态。请参考相关的求解状态常数文档常量：*MIP 的解状态*。

- COPT_INTATTR_SIMPLEXITER 或 "SimplexIter"

  整数属性。

  单纯形法迭代循环数。

- COPT_INTATTR_BARRIERITER 或 "BarrierIter"

  整数属性。

  内点法迭代循环数。

- COPT_INTATTR_NODECNT 或 "NodeCnt"

  整数属性。

  分支定界搜索的节点数。

- COPT_INTATTR_POOLSOLS 或 "PoolSols"

  整数属性。

  解池中的解的数目。

- COPT_INTATTR_TuneResults 或 "TuneResults"

  整数属性。

  参数调优结果的数目。

- COPT_INTATTR_HASLPSOL 或 "HasLpSol"

  整数属性。

  是否可以提供线性规划的解值。

- COPT_INTATTR_HASBASIS 或 "HasBasis"

  整数属性。

  是否可以提供线性规划的基。

- COPT_INTATTR_HASDUALFARKAS 或 "HasDualFarkas"

  整数属性。

  当线性规划问题无可行解时，是否返回对偶 Farkas（也叫做对偶极射线）。

- COPT_INTATTR_HASPRIMALRAY 或 "HasPrimalRay"

  整数属性。

  当线性规划问题无界时，是否返回主元射线（也叫做极射线）。

- COPT_INTATTR_HASMIPSOL 或 "HasMipSol"

  整数属性。

  是否存在整数解。

- COPT_INTATTR_IISCOLS 或 "IISCols"

整数属性。

组成 IIS 的变量边界的数目。

- COPT_INTATTR_IISROWS 或 "IISRows"

  整数属性。

  组成 IIS 的约束的数目。

- COPT_INTATTR_IISSOSS 或 "IISSOSs"

  整数属性。

  组成 IIS 的 SOS 约束的数目。

- COPT_INTATTR_IISINDICATORS 或 "IISIndicators"

  整数属性。

  组成 IIS 的 Indicator 约束的数目。

- COPT_INTATTR_HASIIS 或 "HasIIS"

  整数属性。

  是否存在 IIS。

- COPT_INTATTR_HASFEASRELAXSOL 或 "HasFeasRelaxSol"

  整数属性。

  是否存在可行化松弛结果。

- COPT_INTATTR_ISMINIIS 或 "IsMinIIS"

  整数属性。

  计算出的 IIS 是否为极小。

- COPT_DBLATTR_LPOBJVAL 或 "LpObjval"

  浮点数属性。

  线性规划目标函数值。

- COPT_DBLATTR_BESTOBJ 或 "BestObj"

  浮点数属性。

  整数规划求解结束时最好的目标函数值。

- COPT_DBLATTR_BESTBND 或 "BestBnd"

  浮点数属性。

  整数规划求解结束时最好的下界。

- COPT_DBLATTR_BESTGAP 或 "BestGap"

  浮点数属性。

  整数规划求解结束时最好的相对容差。

- COPT_DBLATTR_FEASRELAXOBJ 或 "FeasRelaxObj"

  浮点数属性。

  可行化松弛值。

- COPT_DBLATTR_SOLVINGTIME 或 "SolvingTime"

  浮点数属性。

  求解所使用的时间（秒）。

22.3 信息

信息常数包括模型信息和求解结果相关信息。

在 C API 中，提供获取变量和约束信息的函数，具体请参考*C API 函数：获取模型信息章节*

注意： Double 这个词的准确翻译应是双精度浮点数。下文为了简便，称之为浮点数。

22.3.1 模型相关信息

- COPT_DBLINFO_OBJ 或 "Obj"

  浮点数信息。

  变量（列）的目标函数系数。

- COPT_DBLINFO_LB 或 "LB"

  浮点数信息。

  变量（列）或者约束（行）的下界。

- COPT_DBLINFO_UB 或 "UB"

  浮点数信息。

  变量（列）或者约束（行）的上界。

22.3.2 求解结果相关信息

- COPT_DBLINFO_VALUE 或 "Value"

  浮点数信息。

  变量（列）的取值。

- COPT_DBLINFO_SLACK 或 "Slack"

  浮点数信息。

  松弛变量的取值，也叫做约束的活跃程度（activities）。仅适用于线性规划模型。

- COPT_DBLINFO_DUAL 或 "Dual"

浮点数信息。

对偶变量的取值。仅适用于线性规划模型。

- `COPT_DBLINFO_REDCOST` 或 `"RedCost"`

  浮点数信息。

  变量的 Reduced cost。仅适用于线性规划模型。

22.3.3 对偶 Farkas 和主元射线

进阶话题。

当线性规划问题无可行解或者无界时，求解器可以返回对偶 Farkas（也叫做对偶极射线）或者主元射线（也叫做极射线）作为证明。

- `COPT_DBLINFO_DUALFARKAS` 或 `"DualFarkas"`

  浮点数信息。

  线性规划问题无可行解时，线性约束的对偶 Farkas（也叫做对偶极射线） 。请设置 `"ReqFarkasRay"` 这一参数，以确保求解器可以返回对偶 Farkas。

  对偶 Farkas 的作用可以用形如 $Ax = 0$ and $l \leq x \leq u$ 的线性规划约束解释。当该线性规划无可行解时，使用对偶 Farkas 向量 y 可以证明线性约束系统存在冲突：$\max y^T Ax < y^T b = 0$。如何计算 $\max y^T Ax$：使用向量 $\hat{a} = y^T A$，当 $\hat{a}_i < 0$ 时选择 $x_i = l_i$ 或者 $\hat{a}_i > 0$ 时选择 $x_i = u_i$ ，可以计算出表达式 $y^T Ax$ 的最大可能取值。

  有些应用依赖于另一种等价的线性系统冲突证明：$\min \bar{y}^T Ax > \bar{y}^T b = 0$。对于此种情况，可以对求解器返回的对偶 Farkas 取负值实现，即 $\bar{y} = -y$。

  在极端情况下，求解器可能无法返回有效的对偶 Farkas。例如当线性规划问题的不可行性微乎其微时。此时，我们建议用 FeasRelax 功能研究或者修复线性规划的不可行性。

- `COPT_DBLINFO_PRIMALRAY` 或 `"PrimalRay"`

  浮点数信息。

  线性规划问题无界时，变量的主元射线（也叫做极射线）。请设置 `"ReqFarkasRay"` 这一参数，以确保求解器可以返回主元射线。

  对于一个求解最小值的线性规划问题 $\min c^T x, Ax = b$ and $x \geq 0$，主元射线向量 r 满足以下条件：$r \geq 0, Ar = 0$ 以及 $c^T r < 0$。

22.3.4 可行化松弛结果相关信息

- `COPT_DBLINFO_RELAXLB` 或 `"RelaxLB"`

  浮点数信息。

  变量（列）或者约束（行）下界的可行化松弛量。

- `COPT_DBLINFO_RELAXUB` 或 `"RelaxUB"`

浮点数信息。

变量（列）或者约束（行）上界的可行化松弛量。

22.4 Callback 相关信息

- COPT_CBINFO_BESTOBJ or "BestObj"

  浮点数信息。

  当前最优目标函数值。

- COPT_CBINFO_BESTBND or "BestBnd"

  浮点数信息。

  当前最优目标下界。

- COPT_CBINFO_HASINCUMBENT or "HasIncumbent"

  整数信息。

  当前是否有最优可行解。

- COPT_CBINFO_INCUMBENT or "Incumbent"

  浮点数信息。

  当前最优可行解。

- COPT_CBINFO_MIPCANDIDATE or "MipCandidate"

  浮点数信息。

  当前可行解。

- COPT_CBINFO_MIPCANDOBJ or "MipCandObj"

  浮点数信息。

  当前可行解对应的目标函数值。

- COPT_CBINFO_RELAXSOLUTION or "RelaxSolution"

  浮点数信息。

  当前 LP 松弛问题的解。

- COPT_CBINFO_RELAXSOLOBJ or "RelaxSolObj"

  浮点数信息。

  当前 LP 松弛问题的目标函数值。

- COPT_CBINFO_NODESTATUS or "NodeStatus"

  整数数信息。

  当前节点 LP 松弛问题的求解状态。可取值请参考：一般常数章节：解状态（部分），除去 NODELIMIT, UNSTARTED, INF_OR_UNB ，其他均为其可能取值。

22.5 参数

22.5.1 限制和容差

注意

Tolerance 有时会翻译为公差或者容差。本文档采用容差这个译法。

Double 这个词的准确翻译应该是双精度浮点数。下文为了简便，称之为浮点数。

- `COPT_DBLPARAM_TIMELIMIT` 或 `"TimeLimit"`

  浮点数参数。

  优化求解的时间限制（秒）。

  默认值 1e20

  最小值 0

  最大值 1e20

- `COPT_DBLPARAM_SOLTIMELIMIT` 或 `"SolTimeLimit"`

  浮点数参数。

  找到原始可行解的时间限制（秒）。

  默认值 1e20

  最小值 0

  最大值 1e20

- `COPT_INTPARAM_NODELIMIT` 或 `"NodeLimit"`

  整数参数。

  整数规划求解的节点数限制。

  默认值 -1（自动选择）

  最小值 -1（自动选择）

  最大值 `INT_MAX`

- `COPT_INTPARAM_BARITERLIMIT` 或 `"BarIterLimit"`

  整数参数。

  内点法求解时的迭代数限制。

  默认值 500

  最小值 0

  最大值 `INT_MAX`

- `COPT_DBLPARAM_MATRIXTOL` 或 `"MatrixTol"`

浮点数参数。

输入矩阵的系数容差。

默认值 1e-10

最小值 0

最大值 1e-7

- COPT_DBLPARAM_FEASTOL 或 "FeasTol"

  浮点数参数。

  变量、约束取值的可行性容差。

  默认值 1e-6

  最小值 1e-9

  最大值 1e-4

- COPT_DBLPARAM_DUALTOL 或 "DualTol"

  浮点数参数。

  对偶解的可行性容差。

  默认值 1e-6

  最小值 1e-9

  最大值 1e-4

- COPT_DBLPARAM_INTTOL 或 "IntTol"

  浮点参数。

  变量的整数解容差。

  默认值 1e-6

  最小值 1e-9

  最大值 1e-1

- COPT_DBLPARAM_RELGAP 或 "RelGap"

  浮点参数。

  整数规划的最优相对容差。

  默认值 1e-4

  最小值 0

  最大值 DBL_MAX

- COPT_DBLPARAM_ABSGAP 或 "AbsGap"

  浮点参数。

  整数规划的最优绝对容差。

默认值 1e-6

最小值 0

最大值 DBL_MAX

22.5.2 预求解相关

- COPT_INTPARAM_PRESOLVE 或 "Presolve"

  整数参数。

  预求解的强度。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 关闭。

  1: 少量快速。

  2: 正常。

  3: 多多益善。

- COPT_INTPARAM_SCALING 或 "Scaling"

  整数参数。

  是否在求解一个模型前，调整系数矩阵的数值（Scaling）。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 不调整。

  1: 调整系数矩阵的数值。

- COPT_INTPARAM_DUALIZE 或 "Dualize"

  整数参数。

  是否构建并求解对偶模型。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 不构建对偶模型。

  1: 构建对偶模型。

22.5.3 线性规划相关

- `COPT_INTPARAM_LPMETHOD` 或 `"LpMethod"`

  整数参数。

  求解线性规划问题的算法。

  默认值 -1

  可选值

  -1: 自动选择。

  对于线性规划问题，选择对偶单纯形法；

  对于混合整数线性规划问题，选择对偶单纯形法或内点法之一。

  1: 对偶单纯形法。

  2: 内点法。

  3: 直接 Crossover。

  4: 并发求解（同时启动单纯形法与内点法求解）。

  5: 基于稀疏和数值范围等特征自动选择单纯形法或者内点法。

- `COPT_INTPARAM_DUALPRICE` 或 `"DualPrice"`

  整数参数。

  选定对偶单纯形法的 Pricing 算法。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 使用 Devex 算法。

  1: 使用对偶最陡边算法。

- `COPT_INTPARAM_DUALPERTURB` 或 `"DualPerturb"`

  整数参数。

  是否允许对偶单纯形算法使用目标函数摄动。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 无摄动。

  1: 允许目标函数摄动。

- `COPT_INTPARAM_BARHOMOGENEOUS` 或 `"BarHomogeneous"`

整数参数。

是否使用齐次自对偶方法。

默认值 -1

可选值

-1: 自动选择。

0: 不使用。

1: 使用。

- `COPT_INTPARAM_BARORDER` 或 `"BarOrder"`

  整数参数。

  内点法矩阵排列算法。

  默认值 -1

  可选值

  -1: 自动选择。

  0: Approximate Minimum Degree（AMD）算法。

  1: Nested Dissection（ND）算法。

- `COPT_INTPARAM_BARSTART` 或 `"BarStart"`

  整数参数。

  内点法寻找初始点的算法。

  默认值 -1

  可选值

  -1: 自动选择。

  0: Simple 算法。

  1: Mehrotra 算法。

  2: Modified Mehrotra 算法。

- `COPT_INTPARAM_CROSSOVER` 或 `"Crossover"`

  整数参数。

  是否使用 Crossover。

  默认值 1

  可选值

  -1: 自动选择。仅在当前线性规划的解不满足容差时使用。

  0: 不使用。

  1: 使用。

- COPT_INTPARAM_REQFARKASRAY 或 "ReqFarkasRay"

  整数参数。

  进阶话题。当线性规划问题无可行解或者无界时，是否计算对偶 Farkas（也叫做对偶极射线）或者主元射线（也叫做极射线）。

  默认值 0

  可选值

  0: 不计算。

  1: 计算。

22.5.4 半定规划相关

- COPT_INTPARAM_SDPMETHOD 或 "SDPMethod"

  整数参数。

  求解半定规划问题的算法。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 原始-对偶内点法。

  1: 交替方向乘子法。

  2: 对偶内点法。

22.5.5 整数规划相关

- COPT_INTPARAM_CUTLEVEL 或 "CutLevel"

  整数参数。

  生成割平面的强度。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 关闭。

  1: 少量快速。

  2: 正常。

  3: 多多益善。

- COPT_INTPARAM_ROOTCUTLEVEL 或 "RootCutLevel"

整数参数。

根节点生成割平面的强度。

默认值 -1

可选值

-1: 自动选择。

0: 关闭。

1: 少量快速。

2: 正常。

3: 多多益善。

- COPT_INTPARAM_TREECUTLEVEL 或 "TreeCutLevel"

  整数参数。

  搜索树生成割平面的强度。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 关闭。

  1: 少量快速。

  2: 正常。

  3: 多多益善。

- COPT_INTPARAM_ROOTCUTROUNDS 或 "RootCutRounds"

  整数参数。

  根节点生成割平面的次数。

  默认值 -1（自动选择）

  最小值 -1（自动选择）

  最大值 INT_MAX

- COPT_INTPARAM_NODECUTROUNDS 或 "NodeCutRounds"

  整数参数。

  搜索树节点生成割平面的次数。

  默认值 -1（自动选择）

  最小值 -1（自动选择）

  最大值 INT_MAX

- COPT_INTPARAM_HEURLEVEL 或 "HeurLevel"

整数参数。

启发式算法的强度。

默认值 -1

可选值

-1: 自动选择。

0: 关闭。

1: 少量快速。

2: 正常。

3: 多多益善。

- `COPT_INTPARAM_ROUNDINGHEURLEVEL` 或 `"RoundingHeurLevel"`

  整数参数。

  Rounding 启发式算法的强度。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 关闭。

  1: 少量快速。

  2: 正常。

  3: 多多益善。

- `COPT_INTPARAM_DIVINGHEURLEVEL` 或 `"DivingHeurLevel"`

  整数参数。

  Diving 启发式算法的强度。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 关闭。

  1: 少量快速。

  2: 正常。

  3: 多多益善。

- `COPT_INTPARAM_FAPHEURLEVEL` 或 `"FAPHeurLevel"`

  整数参数。

  Fix-and-propagate 启发式算法的强度。

默认值 -1

可选值

-1: 自动选择。

0: 关闭。

1: 少量快速。

2: 正常。

3: 多多益善。

- `COPT_INTPARAM_STRONGBRANCHING` 或 `"StrongBranching"`

  整数参数。

  Strong Branching 的强度。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 关闭。

  1: 少量快速。

  2: 正常。

  3: 多多益善。

- `COPT_INTPARAM_CONFLICTANALYSIS` 或 `"ConflictAnalysis"`

  整数参数。

  是否使用冲突分析。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 不使用。

  1: 使用。

- `COPT_INTPARAM_MIPSTARTMODE` 或 `"MipStartMode"`

  整数参数。

  处理初始解的方式。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 不使用任何初始解。

1: 仅使用完整且可行的初始解。

2: 仅使用可行的初始解（若初始解不完整，通过求解子 MIP 来补全）。

- COPT_INTPARAM_MIPSTARTNODELIMIT 或 "MipStartNodeLimit"

  整数参数。

  补全不完整的初始解时，求解的子 MIP 的节点数限制。

  默认值 -1（自动选择）

  最小值 -1（自动选择）

  最大值 INT_MAX

22.5.6 并行计算相关

- COPT_INTPARAM_THREADS 或 "Threads"

  整数参数。

  问题求解使用的线程数。

  默认值 -1（自动选择）

  最小值 -1（自动选择）

  最大值 128

- COPT_INTPARAM_BARTHREADS 或 "BarThreads"

  整数参数。

  内点法使用的线程数。若值为-1，则线程数由参数 Threads 决定。

  默认值 -1

  最小值 -1

  最大值 128

- COPT_INTPARAM_SIMPLEXTHREADS 或 "SimplexThreads"

  整数参数。

  对偶单纯形法使用的线程数。若值为-1，则线程数由参数 Threads 决定。

  默认值 -1

  最小值 -1

  最大值 128

- COPT_INTPARAM_CROSSOVERTHREADS 或 "CrossoverThreads"

  整数参数。

  Crossover 使用的线程数。若值为-1，则线程数由参数 Threads 决定。

  默认值 -1

最小值 -1

最大值 128

- COPT_INTPARAM_MIPTASKS 或 "MipTasks"

  整数参数。

  MIP 求解使用的任务数。

  默认值 -1（自动选择）

  最小值 -1（自动选择）

  最大值 256

22.5.7 IIS 计算相关

- COPT_INTPARAM_IISMETHOD 或 "IISMethod"

  整数参数。

  计算 IIS 的方法。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 计算结果质量优先。

  1: 计算效率优先。

22.5.8 可行化松弛计算相关

- COPT_INTPARAM_FEASRELAXMODE 或 "FeasRelaxMode"

  整数参数。

  计算可行化松弛的方法。

  默认值 0

  可选值

  0: 最小化加权冲突值。

  1: 计算最小化加权冲突下的原始模型最优可行化松弛。

  2: 最小化冲突数目。

  3: 计算最小化冲突数目下的原始模型最优可行化松弛。

  4: 最小化加权平方冲突值。

  5: 计算最小化加权平方冲突下的原始模型最优可行化松弛。

22.5.9 参数调优相关

- COPT_DBLPARAM_TUNETIMELIMIT 或 "TuneTimeLimit"

  浮点数参数。

  参数调优的时间限制。若值为 0 ，则表示求解器自动设置。

  默认值 0

  最小值 0

  最大值 1e20

- COPT_DBLPARAM_TUNETARGETTIME 或 "TuneTargetTime"

  浮点数参数。

  参数调优的时间目标。

  默认值 1e-2

  最小值 0

  最大值 DBL_MAX

- COPT_DBLPARAM_TUNETARGETRELGAP 或 "TuneTargetRelGap"

  浮点数参数。

  参数调优的最优相对容差目标。

  默认值 1e-4

  最小值 0

  最大值 DBL_MAX

- COPT_INTPARAM_TUNEMETHOD 或 "TuneMethod"

  整数参数。

  参数调优的方法。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 贪婪搜索策略。

  1: 更广泛的搜索策略。

- COPT_INTPARAM_TUNEMODE 或 "TuneMode"

  整数参数。

  参数调优的模式。

  默认值 -1

  可选值

-1: 自动选择。

0: 求解时间。

1: 最优相对容差。

2: 目标函数值。

3: 目标函数值下界。

- COPT_INTPARAM_TUNEMEASURE 或 "TuneMeasure"

  整数参数。

  参数调优结果计算方式。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 计算平均值。

  1: 计算最大值。

- COPT_INTPARAM_TUNEPERMUTES 或 "TunePermutes"

  整数参数。

  参数调优每组参数模型计算次数。若值为 0 ，则表示求解器自动设置。

  默认值 0

  最小值 0

  最大值 INT_MAX

- COPT_INTPARAM_TUNEOUTPUTLEVEL 或 "TuneOutputLevel"

  整数参数。

  参数调优日志输出强度。

  默认值 2

  可选值

  0: 不显示。

  1: 仅输出改进参数的摘要。

  2: 输出每次调优尝试的摘要。

  3: 输出每次调优尝试的详细日志。

22.5.10 回调函数相关

- COPT_INTPARAM_LAZYCONSTRAINTS or "LazyConstraints"

  整数参数。

  是否将惰性约束加入模型中。

  默认值 -1

  可选值

  -1: 自动选择。

  0: 否。

  1: 是。

注意：

- 该参数仅对 MIP 模型有效。

22.5.11 其它参数

- COPT_INTPARAM_LOGGING 或 "Logging"

  整数参数。

  是否显示求解日志。

  默认值 1

  可选值

  0: 不显示求解日志。

  1: 显示求解日志。

- COPT_INTPARAM_LOGTOCONSOLE 或 "LogToConsole"

  整数参数。

  是否显示求解日志到控制台。

  默认值 1

  可选值

  0: 不显示求解日志到控制台。

  1: 显示求解日志到控制台。

22.6 API 函数

API 函数的文档按照功能进行分组。

全部 API 函数的返回值都是整数，可能的取值及其含义请参考常量部分的文档。

22.6.1 创建求解环境和模型

COPT_CreateEnvConfig

概要

```
int COPT_CreateEnvConfig(copt_env_config **p_config)
```

描述

创建一个 COPT 客户端配置。

参量

`p_config`

指向 COPT 客户端配置的输出指针。

COPT_DeleteEnvConfig

概要

```
int COPT_DeleteEnvConfig(copt_env_config **p_config)
```

描述

删除 COPT 客户端配置。

参量

`p_config`

指向 COPT 客户端配置的输入指针。

COPT_SetEnvConfig

概要

```
int COPT_SetEnvConfig(copt_env_config *config, const char *name,
const char *value)
```

描述

设置 COPT 客户端配置参数。

参量

`config`

COPT 客户端配置。

`name`

客户端配置参数名。

`value`

客户端配置参数值。

COPT_CreateEnv

概要

```
int COPT_CreateEnv(copt_env **p_env)
```

描述

创建一个 COPT 求解环境。

调用这个函数是使用 COPT 的第一件事情。这个函数会检查授权文件，如果成功，则可以进一步的创建 COPT 模型。如果检查失败，则可以调用 `COPT_GetLicenseMsg` 来获取更多信息，以便找出问题所在。

参量

`p_env`

指向 COPT 求解环境的输出指针。

COPT_CreateEnvWithPath

概要

```
int COPT_CreateEnvWithPath(const char *licDir, copt_env **p_env)
```

描述

创建一个 COPT 求解环境，授权文件路径为参数 `licDir` 指定的路径。

调用这个函数是使用 COPT 的第一件事情。这个函数会检查授权文件，如果成功，则可以进一步的创建 COPT 模型。如果检查失败，则可以调用 `COPT_GetLicenseMsg` 来获取更多信息，以便找出问题所在。

参量

`licDir`

授权文件路径。

`p_env`

指向 COPT 求解环境的输出指针。

COPT_CreateEnvWithConfig

概要

```
int COPT_CreateEnvWithConfig(copt_env_config *config, copt_env
**p_env)
```

描述

创建一个 COPT 求解环境，客户端配置由参数 config 指定。

调用这个函数是使用 COPT 的第一件事情。这个函数会检查客户端参数配置，如果成功，则可以进一步的创建 COPT 模型。如果检查失败，则可以调用 COPT_GetLicenseMsg 来获取更多信息，以便找出问题所在。

参量

config

客户端配置。

p_env

指向 COPT 求解环境的输出指针。

COPT_DeleteEnv

概要

```
int COPT_DeleteEnv(copt_env **p_env)
```

描述

删除 COPT 求解环境。

参量

p_env

指向 COPT 求解环境的输入指针。

COPT_GetLicenseMsg

概要

```
int COPT_GetLicenseMsg(copt_env *env, char *buff, int buffSize)
```

描述

返回一个 C 语言风格的字符串，说明授权文件检查相关信息。

请在调用 COPT_CreateEnv 失败时使用这个函数。

参量

env

COPT 求解环境。

`buff`

用以获取字符串的数组。

`buffSize`

上述数组的大小。

COPT_CreateProb

概要

```
int COPT_CreateProb(copt_env *env, copt_prob **p_prob)
```

描述

创建一个空的 COPT 模型。

参量

`env`

COPT 求解环境。

`p_prob`

指向 COPT 模型的输出指针。

COPT_CreateCopy

概要

```
int COPT_CreateCopy(copt_prob *src_prob, copt_prob **p_dst_prob)
```

描述

创建一个现有 COPT 模型的深拷贝。

注意：模型的求解参数设置也将被拷贝，若用户希望使用不同的参数求解拷贝后的模型，则需要对拷贝后的模型调用 `COPT_ResetParam` 函数，将求解参数重置为默认设置后再进行自定义设置。

参量

`src_prob`

待拷贝 COPT 模型。

`p_dst_prob`

指向新创建 COPT 模型的输出指针。

COPT_DeleteProb

概要

```
int COPT_DeleteProb(copt_prob **p_prob)
```

描述

删除 COPT 模型。

参量

p_prob

指向 COPT 模型的输入指针。

22.6.2 构造和修改模型

COPT_LoadProb

概要

```
int COPT_LoadProb(
    copt_prob *prob,
    int nCol,
    int nRow,
    int iObjSense,
    double dObjConst,
    const double *obj,
    const int *colMatBeg,
    const int *colMatCnt,
    const int *colMatIdx,
    const double *colMatElem,
    const char *colType,
    const double *colLower,
    const double *colUpper,
    const char *rowSense,
    const double *rowBound,
    const double *rowUpper,
    char const *const *colNames,
    char const *const *rowNames)
```

描述

通过多个数组，加载一个 COPT 模型。

参量

`prob`

COPT 模型。

`nCol`

变量数（系数矩阵列数）。

`nRow`

约束数（系数矩阵行数）。

`iObjSense`

优化方向。可能的取值有 `COPT_MAXIMIZE` 或 `COPT_MINIMIZE` 。

`dObjConst`

目标函数的常数项。

`obj`

变量的目标函数系数。

`colMatBeg`, `colMatCnt`, `colMatIdx` 和 `colMatElem`

以列压缩存储格式定义系数矩阵。我们在 **其他信息**中提供了一个使用列压缩存储格式的例子供参考。

如果 `colMatCnt` 为 `NULL` , 则 `colMatBeg` 需要有 `nCol+1` 元素，且指向第 `i` 列的开始和终止指针分别使用 `colMatBeg[i]` 和 `colMatBeg[i+1]` 定义。

如果提供了 `colMatCnt` ，则指向第 `i` 列的开始和终止指针分别使用 `colMatBeg[i]` 和 `colMatBeg[i] + colMatCnt[i]` 定义。

`colType`

变量类型。

如果不提供这个数组，则所有的变量都是连续变量。

`colLower` and `colUpper`

变量的上下边界。

如果不提供下边界 `colLower` ，下边界都将是 0。

如果不提供上边界 `colUpper` ，则变量都无上边界。

`rowSense`

约束类型。

请参考常量部分的文档，了解 COPT 所支持的约束类型。

如果不提供 `rowSense` 这个数组，那么 `rowBound` 和 `rowUppe` 会被当做约束的上下边界。这是我们推荐的定义约束的方式。

如果提供了 rowSense 则 rowBound 和 rowUppe 会被当做 RHS 和 *range* 。在此情况下，rowUpper 数组仅在有 COPT_RANGE 约束类型的时候会用到，且在此时，上下界分别为

下界 rowBound[i] - fabs(rowUpper[i])

上界 rowBound[i]

rowBound

约束的下界或 RHS。

rowUpper

约束的上界或 **range** 。

colNames and rowNames

变量和约束的名字。可以不提供。

其他信息

列压缩存储 (compressed column storage, CCS) 格式，是一种常见的存储稀疏矩阵的格式。我们再次演示如何用此格式来存储这个 4 列 3 行的示例矩阵。

$$A = \begin{bmatrix} 1.1 & 1.2 & & \\ & 2.2 & 2.3 & \\ & & 3.3 & 3.4 \end{bmatrix} \tag{22.4}$$

```
// 仅使用colMatBeg
colMatBeg[5]  = {  0,   1,   3,   5,   6};
colMatIdx[6]  = {  0,   0,   1,   1,   2,   2};
colMatElem[6] = {1.1, 1.2, 2.2, 2.3, 3.3, 3.4};

// 同时用到colMatBeg和colMatCnt
// 星号(*)部分是用不到的数据
colMatBeg[4]  = {  0,   1,   5,   7};
colMatCnt[4]  = {  1,   2,   2,   1};
colMatIdx[6]  = {  0,   0,   1,   1,   2,   *,   *,   2};
colMatElem[6] = {1.1, 1.2, 2.2, 2.3, 3.3,   *,   *, 3.4};
```

COPT_AddCol

概要

```
int COPT_AddCol(
    copt_prob *prob,
    double dColObj,
    int nColMatCnt,
    const int *colMatIdx,
    const double *colMatElem,
```

```
    char cColType,
    double dColLower,
    double dColUpper,
    const char *colName)
```

描述

添加一个变量（列）。

参量

`prob`

COPT 模型。

`dColObj`

变量的目标函数系数。

`nColMatCnt`

此列系数矩阵中的非零元素的数量。

`colMatIdx`

此列系数矩阵中的非零元素的行编号。

`colMatElem`

此列系数矩阵中的非零元素的数值。

`cColType`

变量类型。

`dColLower` and `dColUpper`

变量的上下边界。

`colName`

变量的名字。可以传入 `NULL` 。

COPT_AddPSDCol

概要

```
int COPT_AddPSDCol(copt_prob *prob, int colDim, const char *name)
```

描述

添加一个半定变量。

参量

`prob`

COPT 模型。

`colDim`

新添加的半定变量维度。

name

新添加的半定变量名字。可以传入 NULL 。

COPT_AddRow

概要

```
int COPT_AddRow(
    copt_prob *prob,
    int nRowMatCnt,
    const int *rowMatIdx,
    const double *rowMatElem,
    char cRowSense,
    double dRowBound,
    double dRowUpper,
    const char *rowName)
```

描述

添加一个约束（行）。

参量

prob

COPT 模型。

nRowMatCnt

此行系数矩阵中的非零元素的数量。

rowMatIdx

此行系数矩阵中的非零元素的列编号。

rowMatElem

此行系数矩阵中的非零元素的数值。

cRowSense

约束类型。

请参考常量部分的文档，了解 COPT 所支持的约束类型。

如果 cRowSense 是 0, 则 dRowBound 和 dRowUpper 会被当做约束的上下边界。这是我们推荐的定义约束的方式。

如果 cRowSense 是有意义的量，则 dRowBound 和 dRowUpper 会被当做 RHS 和 **range** 。在此情况下，dRowUpper 仅在约束为 COPT_RANGE 类型时才会用到。且在此时，上下界分别为

下界 dRowBound - dRowUpper

上界 dRowBound

dRowBound

约束的下界或 RHS。

dRowUpper

约束的上界或 **range** 。

rowName

约束的名字。可以传入 NULL 。

COPT_AddCols

概要

```
int COPT_AddCols(
    copt_prob *prob,
    int nAddCol,
    const double *colObj,
    const int *colMatBeg,
    const int *colMatCnt,
    const int *colMatIdx,
    const double *colMatElem,
    const char *colType,
    const double *colLower,
    const double *colUpper,
    char const *const *colNames)
```

描述

添加 nAddCol 个变量（列）。

参量

prob

COPT 模型。

nAddCol

新添加的变量数（系数矩阵列数）。

`colObj`

新变量的目标函数系数。

`colMatBeg`，`colMatCnt`，`colMatIdx` 和 `colMatElem`

以列压缩存储格式提供系数矩阵。有关列压缩存储格式的具体示例，请参见 `COPT_LoadProb` 的 **其他信息**。

`colType`

新变量类型。如果不提供这个数组，则所有的变量都是连续变量。

`colLower` 和 `colUpper`

新变量的上下边界。

如果不提供下边界 `colLower` ，下边界都将是 0。

如果不提供上边界 `colUpper` ，则新变量都无上边界。

`colNames`

新变量的名字。可以传入 `NULL` 。

COPT_AddPSDCols

概要

```
int COPT_AddPSDCols(
    copt_prob *prob,
    int nAddCol,
    const int* colDims,
    char const *const *names)
```

描述

添加 `nAddCol` 个半定变量。

参量

`prob`

COPT 模型。

`nAddCol`

新添加的半定变量个数。

`colDims`

新添加的半定变量维度。

`names`

新添加的半定变量名字。可以传入 `NULL` 。

COPT_AddRows

概要

```
int COPT_AddRows(
    copt_prob *prob,
    int nAddRow,
    const int *rowMatBeg,
    const int *rowMatCnt,
    const int *rowMatIdx,
    const double *rowMatElem,
    const char *rowSense,
    const double *rowBound,
    const double *rowUpper,
    char const *const *rowNames)
```

描述

添加 nAddRow 个新约束（行）。

参量

prob

COPT 模型。

nAddRow

新添加的约束数（系数矩阵行数）。

rowMatBeg，rowMatCnt，rowMatIdx 和 rowMatElem

以行压缩存储格式提供系数矩阵。有关稀疏矩阵的压缩存储格式的具体示例，请参见 COPT_LoadProb 的 **其他信息**。

rowSense

新约束类型。

请参考常量部分的文档，了解 COPT 所支持的约束类型。

如果不提供 rowSense 这个数组，那么 rowBound 和 rowUpper 会被当做约束的上下边界。这是我们推荐的定义约束的方式。

如果提供了 rowSense 则 rowBound 和 rowUpper 会被当做 RHS 和 **range** 。在此情况下，rowUpper 数组仅在有 COPT_RANGE 约束类型的时候会用到，且在此时，上下界分别为

下界 rowBound[i] - fabs(rowUpper[i])

上界 rowBound[i]

`rowBound`

新约束的下界或 RHS。

`rowUpper`

新约束的上界或 **range** 。

`rowNames`

新约束的名字。可以传入 `NULL` 。

COPT_AddSOSs

概要

```
int COPT_AddSOSs(
    copt_prob *prob,
    int nAddSOS,
    const int *sosType,
    const int *sosMatBeg,
    const int *sosMatCnt,
    const int *sosMatIdx,
    const double *sosMatWt)
```

描述

添加 `nAddSOS` 个 SOS 约束。若 `sosMatWt` 为 `NULL` ，则 COPT 在内部自动生成。
注意：若模型包含 SOS 约束，则模型为整数规划模型。

参量

`prob`

COPT 模型。

`nAddSOS`

新添加的 SOS 约束个数。

`sosType`

SOS 约束的类型。

`sosMatBeg`，`sosMatCnt`，`sosMatIdx` 和 `sosMatWt`

以行压缩存储格式提供 SOS 约束的成员。有关稀疏矩阵的压缩存储格式的具体示例，请参见 `COPT_LoadProb` 的 **其他信息**。

`sosMatWt`

SOS 约束中各成员的权重。可以为 `NULL` 。

COPT_AddCones

概要

```
int COPT_AddCones(
    copt_prob *prob,
    int nAddCone,
    const int *coneType,
    const int *coneBeg,
    const int *coneCnt,
    const int *coneIdx)
```

描述

添加 nAddCone 个二阶锥约束。

参量

prob

COPT 模型。

nAddCone

新添加的二阶锥约束个数。

coneType

二阶锥约束的类型。

coneBeg, coneCnt, coneIdx

以行压缩存储格式提供二阶锥约束的成员。有关稀疏矩阵的压缩存储格式的具体示例，请参见 COPT_LoadProb 的 **其他信息**。

COPT_AddQConstr

概要

```
int COPT_AddQConstr(
    copt_prob *prob,
    int nRowMatCnt,
    const int *rowMatIdx,
    const int *rowMatElem,
    int nQMatCnt,
    const int *qMatRow,
    const int *qMatCol,
```

```
    const double *qMatElem,
    char cRowSense,
    double dRowBound,
    const char *name)
```

描述

添加一个二次约束。

注意：目前仅支持求解凸二次约束。

参量

prob

COPT 模型。

nRowMatCnt

二次约束中非零线性项的数目。

rowMatIdx

二次约束中非零线性项系数的列下标。

rowMatElem

二次约束中非零线性项的系数。

nQMatCnt

二次约束中非零二次项的数目。

qMatRow

二次约束中非零二次项的行下标。

qMatCol

二次约束中非零二次项的列下标。

qMatElem

二次约束中非零二次项的系数。

cRowSense

二次约束的类型。可选取值为：COPT_LESS_EQUAL 和 COPT_GREATER_EQUAL 。

dRowBound

二次约束的右端项。

name

二次约束的名字。可以传入 NULL 。

COPT_AddPSDConstr

概要

```
int COPT_AddPSDConstr(
    copt_prob *prob,
    int nRowMatCnt,
    const int *rowMatIdx,
    const int *rowMatElem,
    int nColCnt,
    const int *psdColIdx,
    const int *symMatIdx,
    char cRowSense,
    double dRowBound,
    double dRowUpper,
    const char *name)
```

描述

添加一个半定约束。

参量

prob

COPT 模型。

nRowMatCnt

半定约束中非零线性项的数目。

rowMatIdx

半定约束中非零线性项系数的列下标。

rowMatElem

半定约束中非零线性项的系数。

nColCnt

半定约束中半定项的数目。

psdColIdx

半定约束中半定项的半定变量下标。

symMatIdx

二次约束中半定项的对称矩阵下标。

cRowSense

半定约束的类型。

请参考常量部分的文档，了解 COPT 所支持的约束类型。

如果 cRowSense 是 0, 则 dRowBound 和 dRowUpper 会被当做约束的上下边界。这是我们推荐的定义约束的方式。

如果 cRowSense 是有意义的量，则 dRowBound 和 dRowUpper 会被当做 RHS 和 **range** 。在此情况下，dRowUpper 仅在约束为 COPT_RANGE 类型时才会用到。且在此时，上下界分别为

下界 dRowBound - dRowUpper

上界 dRowBound

dRowBound

半定约束的下界或者右端项。

dRowUpper

半定约束的上界或 **range** 。

name

半定约束的名字。可以传入 NULL 。

COPT_AddLMIConstr

概要

```
int COPT_AddLMIConstr(
    copt_prob *prob,
    int nDim,
    int nLMIMatCnt,
    const int *colIdx,
    const int *symMatIdx,
    int constMatIdx,
    const char *name)
```

描述

添加一个 LMI 约束。

参量

prob

COPT 模型。

nDim

LMI 约束中对称矩阵的维度。

`nLMIMatCnt`

LMI 约束中系数对称矩阵的个数。

`colIdx`

LMI 约束中标量变量的下标。

`symMatIdx`

LMI 约束中系数对称矩阵的下标。

`constMatIdx`

LMI 约束中常数项对称矩阵的下标。

`name`

LMI 约束的名称。可以传入 `NULL` 。

COPT_AddIndicator

概要

```
int COPT_AddIndicator(
    copt_prob *prob,
    int binColIdx,
    int binColVal,
    int nRowMatCnt,
    const int *rowMatIdx,
    const double *rowMatElem,
    char cRowSense, double dRowBound)
```

描述

添加一个 Indicator 约束。

注意：若模型包含 Indicator 约束，则模型为整数规划模型。

参量

`prob`

COPT 模型。

`binColIdx`

二进制类型 Indicator 变量（列）的下标。

`binColVal`

二进制类型 Indicator 变量（列）的取值。

`nRowMatCnt`

线性约束（行）系数矩阵中的非零元素的数量。

`rowMatIdx`

线性约束（行）系数矩阵中的非零元素的列编号。

`rowMatElem`

线性约束（行）系数矩阵中的非零元素的数值。

`cRowSense`

线性约束（行）类型。可选取值为：`COPT_EQUAL` 、`COPT_LESS_EQUAL` 和 `COPT_GREATER_EQUAL` 。

`dRowBound`

线性约束（行）的右端项。

COPT_AddSymMat

概要

```
int COPT_AddSymMat(copt_prob *prob, int ndim, int nelem, int *rows,
int *cols, double *elems)
```

描述

添加一个对称矩阵（传入下三角部分即可）。

参量

`prob`

COPT 模型。

`ndim`

对称矩阵的维度。

`nelem`

对称矩阵的非零元数目。

`rows`

对称矩阵非零元素的行下标。

`cols`

对称矩阵非零元素的列下标。

`elems`

对称矩阵的非零元素。

COPT_DelCols

概要

```
int COPT_DelCols(copt_prob *prob, int num, const int *list)
```

描述

删除 num 个变量（列）。

参量

prob

COPT 模型。

num

要删除的变量个数。

list

要删除的变量下标列表。

COPT_DelPSDCols

概要

```
int COPT_DelPSDCols(copt_prob *prob, int num, const int *list)
```

描述

删除 num 个半定变量。

参量

prob

COPT 模型。

num

要删除的半定变量个数。

list

要删除的半定变量下标列表。

COPT_DelRows

概要

```
int COPT_DelRows(copt_prob *prob, int num, const int *list)
```

描述

删除 num 个约束（行）。

参量

prob

COPT 模型。

num

要删除的约束个数。

list

要删除的约束下标列表。

COPT_DelSOSs

概要

```
int COPT_DelSOSs(copt_prob *prob, int num, const int *list)
```

描述

删除 num 个 SOS 约束。

参量

prob

COPT 模型。

num

要删除的 SOS 约束个数。

list

要删除的 SOS 约束下标列表。

COPT_DelCones

概要

```
int COPT_DelCones(copt_prob *prob, int num, const int *list)
```

描述

删除 num 个二阶锥约束。

参量

prob

COPT 模型。

num

要删除的二阶锥约束个数。

list

要删除的二阶锥约束下标列表。

COPT_DelQConstrs

概要

```
int COPT_DelQConstrs(copt_prob *prob, int num, const int *list)
```

描述

删除 num 个二次约束。

参量

prob

COPT 模型。

num

要删除的二次约束个数。

list

要删除的二次约束下标列表。

COPT_DelPSDConstrs

概要

```
int COPT_DelPSDConstrs(copt_prob *prob, int num, const int *list)
```

描述

删除 num 个半定约束。

参量

prob

COPT 模型。

num

要删除的半定约束个数。

list

要删除的半定约束下标列表。

COPT_DelLMIConstrs

概要

```
int COPT_DelLMIConstrs(copt_prob *prob, int num, const int *list)
```

描述

删除 num 个 LMI 约束。

参量

`prob`

COPT 模型。

`num`

要删除的 LMI 约束个数。

`list`

要删除的 LMI 约束下标列表。

COPT_DelIndicators

概要

```
int COPT_DelIndicators(copt_prob *prob, int num, const int *list)
```

描述

删除 `num` 个 Indicator 约束。

参量

`prob`

COPT 模型。

`num`

要删除的 Indicator 约束个数。

`list`

要删除的 Indicator 约束下标列表。

COPT_DelQuadObj

概要

```
int COPT_DelQuadObj(copt_prob *prob)
```

描述

删除二次目标函数中的二次项。

参量

`prob`

COPT 模型。

COPT_DelPSDObj

概要

```
int COPT_DelPSDObj(copt_prob *prob)
```

描述

删除目标函数中的半定项。

参量

`prob`

COPT 模型。

COPT_SetElem

概要

```
int COPT_SetElem(copt_prob *prob, int iCol, int iRow, double
newElem)
```

描述

设置指定行列对应的系数。

注意：若 `newElem` 的值小于或者等于参数 `MatrixTol` 的值，则将被设置为 0。

参量

`prob`

COPT 模型。

`iCol`

列的下标。

`iRow`

行的下标。

`newElem`

待设置的新系数。

COPT_SetPSDElem

概要

```
int COPT_SetPSDElem(copt_prob *prob, int iCol, int iRow, int newIdx)
```

描述

设置指定半定约束中半定项的对称矩阵下标。

参量

`prob`

COPT 模型。

`iCol`

半定变量的下标。

`iRow`

半定约束的下标。

`newIdx`

待设置的对称矩阵新下标。

COPT_SetLMIElem

概要

```
int COPT_SetLMIElem(copt_prob *prob, int iCol, int iRow, int newIdx)
```

描述

设置指定 LMI 约束中指定变量的系数对称矩阵下标。

参量

`prob`

COPT 模型。

`iCol`

标量变量的下标。

`iRow`

LMI 约束的下标。

`newIdx`

指向新的系数对称矩阵下标的指针。

COPT_SetObjSense

概要

```
int COPT_SetObjSense(copt_prob *prob, int iObjSense)
```

描述

设定（修改）优化方向。

参量

`prob`

COPT 模型。

`iObjSense`

优化方向。可能的取值有 `COPT_MAXIMIZE` 或 `COPT_MINIMIZE` 。

COPT_SetObjConst

概要

```
int COPT_SetObjConst(copt_prob *prob, double dObjConst)
```

描述

设定目标函数的常数项。

参量

prob

COPT 模型。

dObjConst

目标函数的常数项。

COPT_SetColObj/Type/Lower/Upper/Names

概要

```
int COPT_SetColObj(copt_prob *prob, int num, const int *list, const
double *obj)

int COPT_SetColType(copt_prob *prob, int num, const int *list, const
char *type)

int COPT_SetColLower(copt_prob *prob, int num, const int *list,
const double *lower)

int COPT_SetColUpper(copt_prob *prob, int num, const int *list,
const double *upper)

int COPT_SetColNames(copt_prob *prob, int num, const int *list, char
const *const *names)
```

描述

以上五个函数，分别修改 num 个变量（列）的

目标函数参数

变量类型

下边界

上边界

名字

参量

prob

COPT 模型。

num

要修改的变量个数。

`list`

要修改的变量列表。

`obj`

列表中出现的各个变量的新目标函数参数。

`types`

列表中出现的各个变量的新类型。

`lower`

列表中出现的各个变量的新下界。

`upper`

列表中出现的各个变量的新上界。

`names`

列表中出现的各个变量的新名字。

COPT_SetPSDColNames

概要

```
int COPT_SetPSDColNames(copt_prob *prob, int num, const int *list,
char const *const *names)
```

描述

修改 `num` 个半定变量的名字。

参量

`prob`

COPT 模型。

`num`

要修改的半定变量个数。

`list`

要修改的半定变量下标列表。

`names`

列表中出现的各个半定变量的新名字。

COPT_SetRowLower/Upper/Names

概要

```
int COPT_SetRowLower(copt_prob *prob, int num, const int *list,
const double *lower)
```

```
int COPT_SetRowUpper(copt_prob *prob, int num, const int *list,
const double *upper)
```

```
int COPT_SetRowNames(copt_prob *prob, int num, const int *list, char
const *const *names)
```

描述

以上三个函数，分别修改 num 个约束（行）的

下边界

上边界

名字

参量

prob

COPT 模型。

num

要修改的约束个数。

list

要修改的约束列表。

lower

列表中出现的各个约束的新下界。

upper

列表中出现的各个约束的新上界。

names

列表中出现的各个约束的新名字。

COPT_SetQConstrSense/Rhs/Names

概要

```
int COPT_SetQConstrSense(copt_prob *prob, int num, const int *list,
const char *sense)
```

```
int COPT_SetQConstrRhs(copt_prob *prob, int num, const int *list,
const double *rhs)
```

```
int COPT_SetQConstrNames(copt_prob *prob, int num, const int *list,
char const *const *names)
```

描述

以上三个函数，分别修改 num 个二次约束的

类型

右端项

名字

参量

prob

COPT 模型。

num

要修改的二次约束个数。

list

要修改的二次约束列表。

sense

列表中出现的各个二次约束的新类型。

rhs

列表中出现的各个二次约束的新右端项。

names

列表中出现的各个二次约束的新名字。

COPT_SetPSDConstrLower/Upper/Names

概要

```
int COPT_SetPSDConstrLower(copt_prob *prob, int num, const int
*list, const double *lower)

int COPT_SetPSDConstrUpper(copt_prob *prob, int num, const int
*list, const double *upper)

int COPT_SetPSDConstrNames(copt_prob *prob, int num, const int
*list, char const *const *names)
```

描述

以上三个函数，分别修改 num 个半定约束的

下边界

上边界

名字

参量

`prob`

COPT 模型。

`num`

要修改的半定约束个数。

`list`

要修改的半定约束下标列表。

`lower`

列表中出现的各个半定约束的新下界。

`upper`

列表中出现的各个半定约束的新上界。

`names`

列表中出现的各个半定约束的新名字。

COPT_SetLMIConstrRhs

概要

```
int COPT_SetLMIConstrRhs(copt_prob *prob, int num, const int *list,
const int *newIdx)
```

描述

修改 `num` 个 LMI 约束的常数项对称矩阵。

参量

`prob`

COPT 模型。

`num`

要修改的 LMI 约束个数。

`list`

要修改的 LMI 约束下标列表。

`newIdx`

待设置的常数项对称矩阵新下标。

COPT_SetLMIConstrNames

概要

```
int COPT_SetLMIConstrNames(copt_prob *prob, int num, const int
*list, char const *const *names)
```

描述

修改 num 个 LMI 约束的名称。

参量

prob

COPT 模型。

num

要修改的 LMI 约束个数。

list

要修改的 LMI 约束下标列表。

names

列表中出现的各个 LMI 约束的新名称。

COPT_ReplaceColObj

概要

```
int COPT_ReplaceColObj(copt_prob *prob, int num, const int *list,
const double *obj)
```

描述

使用指定的参数构成的目标函数替换之前的目标函数。

参量

prob

COPT 模型。

num

要修改的变量个数。

list

要修改的变量列表。

obj

列表中出现的各个变量的新目标函数参数。

COPT_ReplacePSDObj

概要

```
int COPT_ReplacePSDObj(copt_prob *prob, int num, const int *list,
const int *idx)
```

描述

使用指定的半定项替换目标函数中的半定项。

参量

`prob`

COPT 模型。

`num`

要修改的半定项数目。

`list`

要修改的半定变量列表。

`idx`

列表中出现的各个半定变量的对称矩阵新下标。

COPT_SetQuadObj

概要

```
int COPT_SetQuadObj(copt_prob *prob, int num, int *qRow, int *qCol,
double *qElem)
```

描述

设置二次目标函数中的二次项。

参量

`prob`

COPT 模型。

`num`

二次目标函数中非零二次项数目。

`qRow`

二次目标函数中非零二次项行下标。

`qCol`

二次目标函数中非零二次项列下标。

`qElem`

二次目标函数中非零二次项系数。

COPT_SetPSDObj

概要

```
int COPT_SetPSDObj(copt_prob *prob, int iCol, int newIdx)
```

描述

设置目标函数中的半定项。

参量

`prob`

COPT 模型。

`iCol`

目标函数中半定项的半定变量下标。

`newIdx`

目标函数中半定项的对称矩阵下标。

22.6.3 读入与输出模型

COPT_ReadMps

概要

```
int COPT_ReadMps(copt_prob *prob, const char *mpsfilename)
```

描述

从 MPS 文件中读取模型。

参量

`prob`

COPT 模型。

`mpsfilename`

MPS 文件路径。

COPT_ReadLp

概要

```
int COPT_ReadLp(copt_prob *prob, const char *lpfilename)
```

描述

从 LP 文件中读取模型。

参量

`prob`

COPT 模型。

`lpfilename`

LP 文件路径。

COPT_ReadSDPA

概要

```
int COPT_ReadSDPA(copt_prob *prob, const char *sdpafilename)
```

描述

从 SDPA 文件中读取模型。

参量

`prob`

COPT 模型。

`sdpafilename`

SDPA 文件路径。

COPT_ReadCbf

概要

```
int COPT_ReadCbf(copt_prob *prob, const char *cbffilename)
```

描述

从 CBF 文件中读取模型。

参量

`prob`

COPT 模型。

`cbffilename`

CBF 文件路径。

COPT_ReadBin

概要

```
int COPT_ReadBin(copt_prob *prob, const char *binfilename)
```

描述

从 COPT 二进制文件中读取模型。

参量

`prob`

COPT 模型。

`binfilename`

COPT 二进制文件路径。

COPT_ReadBlob

概要

```
int COPT_ReadBlob(copt_prob *prob, void *blob, COPT_INT64 len)
```

描述

从 COPT 序列化数据中读取模型。

参量

`prob`

COPT 模型。

`blob`

序列化数据。

`len`

序列化数据长度。

COPT_WriteMps

概要

```
int COPT_WriteMps(copt_prob *prob, const char *mpsfilename)
```

描述

把内部的 COPT 模型写出到 MPS 格式文件中。

参量

`prob`

COPT 模型。

`mpsfilename`

MPS 格式文件路径。

COPT_WriteMpsStr

概要

```
int COPT_WriteMpsStr(copt_prob *prob, char *str, int nStrSize, int
*pReqSize)
```

描述

把内部的 COPT 模型以 MPS 格式写出到字符流中。

参量

`prob`

COPT 模型。

`str`

MPS 格式模型字符流。

`nStrSize`

字符流缓冲区的大小。

`pReqSize`

保存模型的字符流最小空间大小。

COPT_WriteLp

概要

```
int COPT_WriteLp(copt_prob *prob, const char *lpfilename)
```

描述

把内部的 COPT 模型写出到 LP 格式文件中。

参量

`prob`

COPT 模型。

`lpfilename`

LP 格式文件路径。

COPT_WriteCbf

概要

```
int COPT_WriteCbf(copt_prob *prob, const char *cbffilename)
```

描述

把内部的 COPT 模型写出到 CBF 格式文件中。

参量

`prob`

COPT 模型。

`cbffilename`

CBF 格式文件路径。

COPT_WriteBin

概要

```
int COPT_WriteBin(copt_prob *prob, const char *binfilename)
```

描述

把内部的 COPT 模型写出到 COPT 二进制格式文件中。

参量

`prob`

COPT 模型。

`binfilename`

COPT 二进制格式文件路径。

COPT_WriteBlob

概要

```
int COPT_WriteBlob(copt_prob *prob, int tryCompress, void **p_blob,
COPT_INT64 *pLen)
```

描述

把内部的 COPT 模型写出到 COPT 序列化数据中。

参量

`prob`

COPT 模型。

`tryCompress`

是否尝试压缩数据。

`p_blob`

指向序列化数据的输出指针。

`pLen`

指向序列化数据的长度的指针。

22.6.4 求解和获取解

COPT_SolveLp

概要

```
int COPT_SolveLp(copt_prob *prob)
```

描述

求解线性规划模型、二阶锥规划模型、二次规划模型、二次约束规划模型或半定规划模型。若模型为整数规划模型，则忽略变量的整数限制，以及 SOS 约束和 Indicator 约束，将其作为连续模型求解。

参量

`prob`

COPT 模型。

COPT_Solve

概要

```
int COPT_Solve(copt_prob *prob)
```

描述

求解线性规划、二阶锥规划、二次规划、二次约束规划、半定规划或者整数规划模型。

参量

`prob`

COPT 模型。

COPT_GetSolution

概要

```
int COPT_GetSolution(copt_prob *prob, double *colVal)
```

描述

获取整数规划模型的解。

参量

prob

COPT 模型。

colVal

变量的取值。

COPT_GetPoolObjVal

概要

```
int COPT_GetPoolObjVal(copt_prob *prob, int iSol, double *p_objVal)
```

描述

获取解池中第 iSol 个解的目标函数值。

参量

prob

COPT 模型。

iSol

解的索引。

p_objVal

指向目标函数值的指针。

COPT_GetPoolSolution

概要

```
int COPT_GetPoolSolution(copt_prob *prob, int iSol, int num, const
int *list, double *colVal)
```

描述

获取解池中第 iSol 个解。

参量

prob

COPT 模型。

iSol

解的索引。

num

获取解的变量（列）的个数。

list

获取解的变量（列）的下标列表。可以为 NULL 。

`colVal`

返回获取的解的数组。

COPT_GetLpSolution

概要

```
int COPT_GetLpSolution(copt_prob *prob, double *value, double
*slack, double *rowDual, double *redCost)
```

描述

获取线性规划、二阶锥规划、二次规划、二次约束规划和半定规划中变量的解。

注意：对于半定规划，获取半定变量的解请使用 `COPT_GetPSDColInfo` 函数。

参量

`prob`

COPT 模型。

`value`

变量的取值。可以为 `NULL` 。

`slack`

松弛变量的取值。也叫做约束的活跃程度（activities）。可以为 `NULL` 。

`rowDual`

对偶变量的取值。可以为 `NULL` 。

`redCost`

变量的 Reduced cost。可以为 `NULL` 。

COPT_SetLpSolution

概要

```
int COPT_SetLpSolution(copt_prob *prob, double *value, double
*slack, double *rowDual, double *redCost)
```

描述

设置线性规划的解。

参量

`prob`

COPT 模型。

`value`

变量的取值。

`slack`

松弛变量的取值。

`rowDual`

对偶变量的取值。

`redCost`

变量的 Reduced cost。

COPT_GetBasis

概要

```
int COPT_GetBasis(copt_prob *prob, int *colBasis, int *rowBasis)
```

描述

获取线性规划的基状态。

参量

`prob`

COPT 模型。

`colBasis` 和 `rowBasis`

分别是变量和约束的基状态。请参考常量部分关于基状态的种类及其含义。

COPT_SetBasis

概要

```
int COPT_SetBasis(copt_prob *prob, const int *colBasis, const int
*rowBasis)
```

描述

设置线性规划的基。使用这个函数可以热启动线性优化。

参量

`prob`

COPT 模型。

`colBasis` 和 `rowBasis`

分别是变量和约束的基状态。请参考常量部分关于基状态的种类及其含义。

COPT_SetSlackBasis

概要

```
int COPT_SetSlackBasis(copt_prob *prob)
```

描述

设置线性规划的基为松弛基。线性规划默认的起始点就是松弛基。使用这个函数，可以使线性规划重置为其默认起点。

参量

`prob`

COPT 模型。

COPT_Reset

概要

```
int COPT_Reset(copt_prob *prob, int iClearAll)
```

描述

重置模型中存储的求解结果，使下次求解重新开始。当 `iClearAll` 为 1 时，清除初始解等其它信息。

参量

`prob`

COPT 模型。

`iClearAll`

是否清除其它信息。

COPT_ReadSol

概要

```
int COPT_ReadSol(copt_prob *prob, const char *solfilename)
```

描述

从结果文件中读取变量的取值。对于整数规划，可以作为整数规划的初始解。

注意：调用该函数时，变量的默认取值为 0，即任何不完整的解会被自动用零补全。

参量

`prob`

COPT 模型。

`solfilename`

求解结果文件路径。

COPT_WriteSol

概要

```
int COPT_WriteSol(copt_prob *prob, const char *solfilename)
```

描述

将模型的求解结果写到文件中。

参量

`prob`

COPT 模型。

`solfilename`

求解结果文件路径。

COPT_WritePoolSol

概要

```
int COPT_WritePoolSol(copt_prob *prob, int iSol, const char
*solfilename)
```

描述

将指定的解池中的解写到文件中。仅适用于整数规划模型。

参量

`prob`

COPT 模型。

`iSol`

解池中解的索引。

`solfilename`

求解结果文件路径。

COPT_WriteBasis

概要

```
int COPT_WriteBasis(copt_prob *prob, const char *basfilename)
```

描述

把 COPT 内部的线性规划的基状态写出到文件中。

参量

`prob`

COPT 模型。

`basfilename`

基状态文件路径。

COPT_ReadBasis

概要

```
int COPT_ReadBasis(copt_prob *prob, const char *basfilename)
```

描述

从文件中读取线性规划的基状态。使用这个函数可以热启动线性优化。

参量

`prob`

COPT 模型。

`basfilename`

基状态文件路径。

22.6.5 获取模型信息

COPT_GetCols

概要

```
int COPT_GetCols(
    copt_prob *prob,
    int nCol,
    const int *list,
    int *colMatBeg,
    int *colMatCnt,
    int *colMatIdx,
    double *colMatElem,
    int nElemSize,
    int *pReqSize)
```

描述

按列的方式获取模型的系数矩阵。

一般来说，用户需要连续调用两次该函数完成系数矩阵抽取操作。首先，将 `colMatBeg`，`colMatCnt`，`colMatIdx` 和 `colMatElem` 传入 `NULL`，通过 `pReqSize` 返回由 `nCol` 和 `list` 指定的系数矩阵非零元个数，然后对列压缩格式矩阵参量分配合理的空间，并再次调用该函数获取指定的系数矩阵。若传入的系数矩阵长度不

够，则返回前 nElemSize 个长度的非零系数，并通过 pReqSize 返回需要的最小数组长度。若 list 为 NULL ，则返回前 nCol 个列对应的系数矩阵。

参量

prob

COPT 模型。

nCol

待获取的列的个数。

list

待获取的列的下标列表。可以为 NULL 。

colMatBeg，colMatCnt，colMatIdx 和 colMatElem

以列压缩存储格式提供系数矩阵。有关列压缩存储格式的具体示例，请参见 COPT_LoadProb 的 **其他信息**。

nElemSize

传入非零系数数组的长度。

pReqSize

指向待获取的列包含的非零系数的总数目的指针。可以为 NULL 。

COPT_GetPSDCols

概要

```
int COPT_GetPSDCols(copt_prob *prob, int nCol, int *list, int*
colDims, int *colLens)
```

描述

获取指定半定变量的维度与展开长度。

参量

prob

COPT 模型。

nCol

指定的半定变量个数。

list

指定的半定变量下标列表。

colDims

指定的半定变量维度。

colLens

指定的半定变量展开长度。

COPT_GetRows

概要

```
int COPT_GetRows(
    copt_prob *prob,
    int nRow,
    const int *list,
    int *rowMatBeg,
    int *rowMatCnt,
    int *rowMatIdx,
    double *rowMatElem,
    int nElemSize,
    int *pReqSize)
```

描述

按行的方式获取模型的系数矩阵。

一般来说，用户需要连续调用两次该函数完成系数矩阵抽取操作。首先，将 rowMatBeg，rowMatCnt，rowMatIdx 和 rowMatElem 传入 NULL ，通过 pReqSize 返回由 nRow 和 list 指定的系数矩阵非零元个数，然后对行压缩格式矩阵参量分配合理的空间，并再次调用该函数获取指定的系数矩阵。若传入的系数矩阵长度不够，则返回前 nElemSize 个长度的非零系数，并通过 pReqSize 返回需要的最小数组长度。若 list 为 NULL ，则返回前 nRow 个行对应的系数矩阵。

参量

prob

COPT 模型。

nRow

待获取的行的个数。

list

待获取的行的下标列表。可以为 NULL 。

rowMatBeg，rowMatCnt，rowMatIdx 和 rowMatElem

以行压缩存储格式提供系数矩阵。有关行压缩存储格式的具体示例，请参见 COPT_LoadProb 的 **其他信息**。

nElemSize

传入非零系数数组的长度。

pReqSize

指向待获取的行包含的非零系数的总数目的指针。可以为 NULL 。

COPT_GetElem

概要

```
int COPT_GetElem(copt_prob *prob, int iCol, int iRow, double
*p_elem)
```

描述

获取指定行列对应的系数。

参量

prob

COPT 模型。

iCol

列的下标。

iRow

行的下标。

p_elem

指向待获取系数的指针。

COPT_GetPSDElem

概要

```
int COPT_GetPSDElem(copt_prob *prob, int iCol, int iRow, int *p_idx)
```

描述

获取指定半定约束中指定半定项的对称矩阵下标。

参量

prob

COPT 模型。

iCol

半定变量的下标。

iRow

半定约束的下标。

p_idx

指向待获取对称矩阵下标的指针。

COPT_GetLMIElem

概要

```
int COPT_GetLMIElem(copt_prob *prob, int iCol, int iRow, int *p_idx)
```

描述

获取指定 LMI 约束中指定变量的系数对称矩阵下标。

参量

`prob`

COPT 模型。

`iCol`

标量变量的下标。

`iRow`

LMI 约束的下标。

`p_idx`

指向待获取系数对称矩阵下标的指针。

COPT_GetSymMat

概要

```
int COPT_GetSymMat(
    copt_prob *prob,
    int iMat,
    int *p_nDim,
    int *p_nElem,
    int *rows,
    int *cols,
    double *elems)
```

描述

获取指定的对称矩阵。

一般来说，用户需要连续调用两次该函数完成对称矩阵的抽取操作。首先，将 `rows`，`cols` 和 `elems` 传入 `NULL`，通过 `p_nDim` 和 `p_nElem` 返回对称矩阵的维度与非零元个数，然后为对称矩阵参量 `rows`，`cols` 和 `elems` 分配合理的空间，并再次调用该函数获取指定的对称矩阵。

参量

`prob`

COPT 模型。

`iMat`

对称矩阵的下标。

`p_nDim`

指向对称矩阵维度的指针。

`p_nElem`

指向对称矩阵非零元数目的指针。

`rows`

对称矩阵非零元素行索引下标。

`cols`

对称矩阵非零元素列索引下标。

`elems`

对称矩阵的非零元素。

COPT_GetQuadObj

概要

```
int COPT_GetQuadObj(copt_prob* prob, int* p_nQElem, int* qRow, int*
qCol, double* qElem)
```

描述

获取目标函数中的非零二次项。

参量

`prob`

COPT 模型。

`p_nQElem`

指向非零二次项数目的指针。

`qRow`

非零二次项的行下标。

`qCol`

非零二次项的列下标。

`qElem`

非零二次项系数。

COPT_GetPSDObj

概要

```
int COPT_GetPSDObj(copt_prob *prob, int iCol, int *p_idx)
```

描述

获取目标函数中的指定半定项。

参量

`prob`

COPT 模型。

`iCol`

半定变量的下标。

`p_idx`

指向对称矩阵下标的指针。

COPT_GetSOSs

概要

```
int COPT_GetSOSs(
    copt_prob *prob,
    int nSos,
    const int *list,
    int *sosMatBeg,
    int *sosMatCnt,
    int *sosMatIdx,
    double *sosMatElem,
    int nElemSize,
    int *pReqSize)
```

描述

获取模型的 SOS 约束矩阵。

一般来说，用户需要连续调用两次该函数完成 SOS 约束矩阵抽取操作。首先，将 `sosMatBeg`，`sosMatCnt`，`sosMatIdx` 和 `sosMatElem` 传入 `NULL`，通过 `pReqSize` 返回由 `nSos` 和 `list` 指定的 SOS 约束矩阵非零元个数，然后对行压缩格式矩阵参量分配合理的空间，并再次调用该函数获取指定的 SOS 约束矩阵。若传入的 SOS 约束矩阵长度不够，则返回前 `nElemSize` 个长度的非零系数，并通过 `pReqSize` 返回需要的最小数组长度。若 `list` 为 `NULL`，则返回前 `nSos` 个行对应的 SOS 约束矩阵。

参量

prob

COPT 模型。

nSos

待获取的 SOS 约束的个数。

list

待获取的 SOS 约束的下标列表。可以为 NULL 。

sosMatBeg, sosMatCnt, sosMatIdx 和 sosMatElem

以行压缩存储格式提供 SOS 约束矩阵。有关行压缩存储格式的具体示例，请参见 COPT_LoadProb 的 **其他信息**。

nElemSize

传入非零系数数组的长度。

pReqSize

指向待获取的行包含的非零系数的总数目的指针。可以为 NULL 。

COPT_GetCones

概要

```
int COPT_GetCones(
    copt_prob *prob,
    int nCone,
    const int *list,
    int *coneBeg,
    int *coneCnt,
    int *coneIdx,
    int nElemSize,
    int *pReqSize)
```

描述

获取模型的二阶锥约束矩阵。

一般来说，用户需要连续调用两次该函数完成二阶锥约束矩阵抽取操作。首先，将 coneBeg, coneCnt 和 coneIdx 传入 NULL , 通过 pReqSize 返回由 nCone 和 list 指定的二阶锥约束矩阵非零元个数，然后对行压缩格式矩阵参量分配合理的空间，并再次调用该函数获取指定的二阶锥约束矩阵。若传入的二阶锥约束矩阵长度不够，则返回前 nElemSize 个长度的非零系数，并通过 pReqSize 返回需要的最小数组长度。若 list 为 NULL ， 则返回前 nCone 个行对应的二阶锥约束矩阵。

参量

`prob`

COPT 模型。

`nCone`

待获取的二阶锥约束的个数。

`list`

待获取的二阶锥约束的下标列表。可以为 `NULL` 。

`coneBeg, coneCnt, coneIdx`

以行压缩存储格式提供二阶锥约束矩阵。有关行压缩存储格式的具体示例，请参见 `COPT_LoadProb` 的 **其他信息**。

`nElemSize`

传入非零系数数组的长度。

`pReqSize`

指向待获取的行包含的非零系数的总数目的指针。可以为 `NULL` 。

COPT_GetQConstr

概要

```
int COPT_GetQConstr(
    copt_prob *prob,
    int qConstrIdx,
    int *qMatRow,
    int *qMatCol,
    double *qMatElem,
    int nQElemSize,
    int *pQReqSize,
    int *rowMatIdx,
    double *rowMatElem,
    char *cRowSense,
    double *dRowBound,
    int nElemSize,
    int *pReqSize)
```

描述

获取模型的二次约束。

一般来说，用户需要连续调用两次该函数完成二次约束的抽取操作。首先，将 qMatRow，qMatCol 和 qMatElem 传入 NULL，通过 pQReqSize 返回由 qConstrIdx 指定的二次约束的非零二次项个数，将 rowMatIdx 和 rowMatElem 传入 NULL，通过 pReqSize 返回由 qConstrIdx 指定的二次约束的非零线性项个数，然后对二次项和线性项系数参量分配合理的空间，并再次调用该函数获取指定的二次约束。若传入的二次项系数向量长度不够，则返回前 nQElemSize 个长度的非零二次项系数，并通过 pQReqSize 返回需要的最小数组长度；若传入的线性项系数向量长度不够，则返回前 nElemSize 个长度的非零线性项系数，并通过 pReqSize 返回需要的最小数组长度。

参量

prob

COPT 模型。

qConstrIdx

二次约束的下标。

qMatRow

二次约束中非零二次项的行下标。

qMatCol

二次约束中非零二次项的列下标。

qMatElem

二次约束中非零二次项系数。

nQElemSize

传入的非零二次项系数数组长度。

pQReqSize

指向待获取的二次约束包含的非零二次项的总数目的指针。可以为 NULL。

rowMatIdx

二次约束中非零线性项系数的下标。

rowMatElem

二次约束中的非零线性项系数。

cRowSense

二次约束的类型。

dRowBound

二次约束的右端项。

nElemSize

传入非零线性项系数数组的长度。

pReqSize

指向待获取的二次约束包含的非零线性项系数的总数目的指针。可以为 NULL 。

COPT_GetPSDConstr

概要

```
int COPT_GetPSDConstr(
    copt_prob *prob,
    int psdConstrIdx,
    int *psdColIdx,
    int *symMatIdx,
    int nColSize,
    int *pColReqSize,
    int *rowMatIdx,
    double *rowMatElem,
    double *dRowLower,
    double *dRowUpper,
    int nElemSize,
    int *pReqSize)
```

描述

获取模型的半定约束。

一般来说，用户需要连续调用两次该函数完成半定约束的抽取操作。首先，将 psdColIdx 和 symMatIdx 传入 NULL ，通过 pColReqSize 返回由 psdConstrIdx 指定的半定约束的半定项个数，将 rowMatIdx 和 rowMatElem 传入 NULL ，通过 pReqSize 返回由 qConstrIdx 指定的半定约束的非零线性项个数，然后对半定项和线性项系数参量分配合理的空间，并再次调用该函数获取指定的半定约束。若传入的半定项向量长度不够，则返回前 nColSize 个长度的半定项系数，并通过 pColReqSize 返回需要的最小数组长度；若传入的线性项系数向量长度不够，则返回前 nElemSize 个长度的非零线性项系数，并通过 pReqSize 返回需要的最小数组长度。

参量

prob

COPT 模型。

psdConstrIdx

半定约束的下标。

psdColIdx

半定变量的下标。

symMatIdx

对称矩阵的下标。

nColSize

传入的半定项数组长度。

pColReqSize

指向待获取的半定约束中半定项的总数目的指针。可以为 NULL 。

rowMatIdx

半定约束中非零线性项系数的下标。

rowMatElem

半定约束中的非零线性项系数。

dRowLower

指向半定约束下界的指针。

dRowUpper

指向半定约束上界的指针。

nElemSize

传入非零线性项系数数组的长度。

pReqSize

指向待获取的半定约束包含的非零线性项系数的总数目的指针。可以为 NULL 。

COPT_GetLMIConstr

概要

```
int COPT_GetLMIConstr(
    copt_prob *prob,
    int lmiConstrIdx,
    int *nDim,
    int *nLMILen,
    int *colIdx,
    int *symMatIdx,
    int *constMatIdx,
    int nElemSize,
```

```
    int *pReqSize)
```

描述

获取模型中指定下标的 LMI 约束。

参量

`prob`

COPT 模型。

`lmiConstrIdx`

LMI 约束的下标。

`nDim`

指向 LMI 约束中对称矩阵维度的指针。

`nLMILen`

指向展开后的 LMI 约束长度的指针。

`colIdx`

LMI 约束中标量变量的下标。

`symMatIdx`

LMI 约束中系数对称矩阵的下标。

`constMatIdx`

指向 LMI 约束中常数项对称矩阵下标的指针。

`nElemSize`

传入非零系数数组的长度。

`pReqSize`

指向待获取的 LMI 约束包含的非零线性项系数的总数目的指针。可以为 `NULL` 。

COPT_GetIndicator

概要

```
int COPT_GetIndicator(
    copt_prob *prob,
    int rowIdx,
    int *binColIdx,
    int *binColVal,
    int *nRowMatCnt,
    int *rowMatIdx,
```

```
    double *rowMatElem,
    char *cRowSense,
    double *dRowBound,
    int nElemSize,
    int *pReqSize)
```

描述

获取模型的 Indicator 约束。

一般来说，用户需要连续调用两次该函数完成 Indicator 约束的抽取操作。首先，将 `nRowMatCnt`，`rowMatIdx` 和 `rowMatElem` 传入 `NULL`，通过 `pReqSize` 返回由 `rowIdx` 指定的 Indicator 约束的非零元个数，然后对约束系数参量分配合理的空间，并再次调用该函数获取指定的 Indicator 约束。若传入的 Indicator 约束系数向量长度不够，则返回前 `nElemSize` 个长度的非零系数，并通过 `pReqSize` 返回需要的最小数组长度。

参量

`prob`

COPT 模型。

`rowIdx`

Indicator 约束（行）的下标。

`binColIdx`

二进制类型 Indicator 变量（列）的下标。

`binColVal`

二进制类型 Indicator 变量（列）的取值。

`nRowMatCnt`

线性约束（行）系数向量中非零元素的数量。

`rowMatIdx`

线性约束（行）系数向量中非零元素的下标。

`rowMatElem`

线性约束（行）系数向量中的非零元素。

`cRowSense`

线性约束（行）的类型。

`dRowBound`

线性约束（行）的右端项。

`nElemSize`

传入非零系数数组的长度。

`pReqSize`

指向待获取的行包含的非零系数的总数目的指针。可以为 `NULL` 。

COPT_GetColIdx

概要

```
int COPT_GetColIdx(copt_prob *prob, const char *colName, int
*p_iCol)
```

描述

根据变量（列）的名字获取其在 COPT 内部的下标。

参量

`prob`

COPT 模型。

`colName`

变量（列）的名字。

`p_iCol`

指向变量（列）在 COPT 内部下标的指针。

COPT_GetPSDColIdx

概要

```
int COPT_GetPSDColIdx(copt_prob *prob, const char *psdColName, int
*p_iPSDCol)
```

描述

根据半定变量的名字获取其在 COPT 内部的下标。

参量

`prob`

COPT 模型。

`psdColName`

半定变量的名字。

`p_iPSDCol`

指向半定变量在 COPT 内部下标的指针。

COPT_GetRowIdx

概要

```
int COPT_GetRowIdx(copt_prob *prob, const char *rowName, int
*p_iRow)
```

描述

根据约束（行）的名字获取其在 COPT 内部的下标。

参量

`prob`

COPT 模型。

`rowName`

约束（行）的名字。

`p_iRow`

指向约束（行）在 COPT 内部下标的指针。

COPT_GetQConstrIdx

概要

```
int COPT_GetQConstrIdx(copt_prob *prob, const char *qConstrName, int
*p_iQConstr)
```

描述

根据二次约束的名字获取其在 COPT 内部的下标。

参量

`prob`

COPT 模型。

`qConstrName`

二次约束的名字。

`p_iQConstr`

指向二次约束在 COPT 内部下标的指针。

COPT_GetPSDConstrIdx

概要

```
int COPT_GetPSDConstrIdx(copt_prob *prob, const char *psdConstrName,
int *p_iPSDConstr)
```

描述

根据半定约束的名字获取其在 COPT 内部的下标。

参量

`prob`

COPT 模型。

`psdConstrName`

半定约束的名字。

`p_iPSDConstr`

指向半定约束在 COPT 内部下标的指针。

COPT_GetLMIConstrIdx

概要

```
int COPT_GetLMIConstrIdx(copt_prob *prob, const char *lmiConstrName,
int *p_iLMIConstr)
```

描述

根据 LMI 约束名称获取其在 COPT 内部的下标。

参量

`prob`

COPT 模型。

`lmiConstrName`

LMI 约束的名称。

`p_iLMIConstr`

指向 LMI 约束在 COPT 内部下标的指针。

COPT_GetColInfo

概要

```
int COPT_GetColInfo(copt_prob *prob, const char *infoName, int num,
const int *list, double *info)
```

描述

获取变量（列）的信息。若 `list` 为 `NULL` ，则返回前 `num` 个变量（列）的信息。

参量

`prob`

COPT 模型。

`infoName`

信息的名字。请查看信息 章节了解目前支持的信息类型。

`num`

获取信息的变量（列）个数。

`list`

获取信息的变量（列）的下标列表。可以为 `NULL` 。

`info`

返回获取的信息的数组。

COPT_GetPSDColInfo

概要

```
int COPT_GetPSDColInfo(copt_prob *prob, const char *infoName, int
iCol, double *info)
```

描述

获取半定变量的信息。

参量

`prob`

COPT 模型。

`infoName`

信息的名字。请查看信息 章节了解目前支持的信息类型。

`iCol`

获取信息的半定变量下标。

`info`

返回获取的信息的数组。

COPT_GetRowInfo

概要

```
int COPT_GetRowInfo(copt_prob *prob, const char *infoName, int num,
const int *list, double *info)
```

描述

获取约束（行）的信息。若 `list` 为 `NULL` ，则返回前 `num` 个约束（行）的信息。

参量

`prob`

COPT 模型。

`infoName`

信息的名字。请查看信息 章节了解目前支持的信息类型。

`num`

获取信息的约束（行）个数。

`list`

获取信息的约束（行）的下标列表。可以为 `NULL` 。

`info`

返回获取的信息的数组。

COPT_GetQConstrInfo

概要

```
int COPT_GetQConstrInfo(copt_prob *prob, const char *infoName, int
num, const int *list, double *info)
```

描述

获取二次约束的信息。若 `list` 为 `NULL` ，则返回前 `num` 个二次约束的信息。

参量

`prob`

COPT 模型。

`infoName`

信息的名字。请查看信息 章节了解目前支持的信息类型。

`num`

获取信息的二次约束个数。

`list`

获取信息的二次约束的下标列表。可以为 `NULL` 。

`info`

返回获取的信息的数组。

COPT_GetPSDConstrInfo

概要

```
int COPT_GetPSDConstrInfo(copt_prob *prob, const char *infoName, int
num, const int* list, double *info)
```

描述

获取半定约束的信息。若 `list` 为 `NULL` ，则返回前 `num` 个半定约束的信息。

参量

`prob`

COPT 模型。

`infoName`

信息的名字。请查看信息 章节了解目前支持的信息类型。

`num`

获取信息的半定约束个数。

`list`

获取信息的半定约束的下标列表。可以为 `NULL` 。

`info`

返回获取的信息的数组。

COPT_GetLMIConstrInfo

概要

```
int COPT_GetLMIConstrInfo(copt_prob *prob, const char *infoName, int
iLMI, double *info)
```

描述

获取 LMI 约束的一组信息。

参量

`prob`

COPT 模型。

`infoName`

信息的名称。可取值为：`COPT_DBLINFO_SLACK` 和 `COPT_DBLINFO_DUAL` 。

`iLMI`

待获取信息的 LMI 约束的下标。

`info`

返回获取的信息的数组。

COPT_GetColType

概要

```
int COPT_GetColType(copt_prob *prob, int num, const int *list, char
*type)
```

描述

获取变量（列）的类型。若 `list` 为 `NULL` ，则返回前 `num` 个变量（列）的类型。

参量

`prob`

COPT 模型。

`num`

变量（列）的个数。

`list`

变量（列）的下标列表。可以为 `NULL` 。

`type`

返回的变量（列）的类型数组。

COPT_GetColBasis

概要

```
int COPT_GetColBasis(copt_prob *prob, int num, const int *list, int
*colBasis)
```

描述

获取变量（列）的基状态。若 `list` 为 `NULL` ，则返回前 `num` 个变量（列）的基状态。

参量

`prob`

COPT 模型。

`num`

变量（列）的个数。

`list`

变量（列）的下标列表。可以为 `NULL` 。

colBasis

返回的变量（列）的基状态数组。

COPT_GetRowBasis

概要

```
int COPT_GetRowBasis(copt_prob *prob, int num, const int *list, int
*rowBasis)
```

描述

获取约束（行）的基状态。若 list 为 NULL ，则返回前 num 个约束（行）的基状态。

参量

prob

COPT 模型。

num

约束（行）的个数。

list

约束（行）的下标列表。可以为 NULL 。

rowBasis

返回的约束（行）的基状态数组。

COPT_GetQConstrSense

概要

```
int COPT_GetQConstrSense(copt_prob *prob, int num, const int *list,
char *sense)
```

描述

获取二次约束的类型。若 list 为 NULL ，则返回前 num 个二次约束的类型。

参量

prob

COPT 模型。

num

二次约束的个数。

list

二次约束的下标列表。可以为 NULL 。

sense

返回的二次约束类型数组。

COPT_GetQConstrRhs

概要

```
int COPT_GetQConstrRhs(copt_prob *prob, int num, const int *list,
double *rhs)
```

描述

获取二次约束的右端项。若 `list` 为 `NULL` ，则返回前 `num` 个二次约束的右端项。

参量

`prob`

COPT 模型。

`num`

二次约束的个数。

`list`

二次约束的下标列表。可以为 `NULL` 。

`rhs`

返回的二次约束右端项。

COPT_GetColName

概要

```
int COPT_GetColName(copt_prob *prob, int iCol, char *buff, int
buffSize, int *pReqSize)
```

描述

根据变量（列）的下标获取其名字。若 `buff` 参量有效长度不够，且将待获取名字的前 `buffSize` 长度的子串返回，且通过 `pReqSize` 返回完全存储待获取名字需要的字符串长度。若 `buff` 参量为 `NULL`，则可以通过 `pReqSize` 返回待获取名字的字符串长度。

参量

`prob`

COPT 模型。

`iCol`

变量（列）的下标。

`buff`

用以获取字符串的数组。

buffSize

上述数组的大小。

pReqSize

指向存储待获取名字的最小字符串长度的指针。可以为 NULL 。

COPT_GetPSDColName

概要

```
int COPT_GetPSDColName(copt_prob *prob, int iPSDCol, char *buff, int
buffSize, int *pReqSize)
```

描述

根据半定变量的下标获取其名字。若 buff 参量有效长度不够，且将待获取名字的前 buffSize 长度的子串返回，且通过 pReqSize 返回完全存储待获取名字需要的字符串长度。若 buff 参量为 NULL，则可以通过 pReqSize 返回待获取名字的字符串长度。

参量

prob

COPT 模型。

iPSDCol

半定变量的下标。

buff

用以获取字符串的数组。

buffSize

上述数组的大小。

pReqSize

指向存储待获取名字的最小字符串长度的指针。可以为 NULL 。

COPT_GetRowName

概要

```
int COPT_GetRowName(copt_prob *prob, int iRow, char *buff, int
buffSize, int *pReqSize)
```

描述

根据约束（行）的下标获取其名字。若 buff 参量有效长度不够，且将待获取名字的前 buffSize 长度的子串返回，且通过 pReqSize 返回完全存储待获取名字需要的字符串长度。若 buff 参量为 NULL，则可以通过 pReqSize 返回待获取名字的字符串长度。

参量

prob

COPT 模型。

iRow

约束（行）的下标。

buff

用以获取字符串的数组。

buffSize

上述数组的大小。

pReqSize

指向存储待获取名字的最小字符串长度的指针。可以为 NULL 。

COPT_GetQConstrName

概要

```
int COPT_GetQConstrName(copt_prob *prob, int iQConstr, char *buff,
int buffSize, int *pReqSize)
```

描述

根据二次约束的下标获取其名字。若 buff 参量有效长度不够，且将待获取名字的前 buffSize 长度的子串返回，且通过 pReqSize 返回完全存储待获取名字需要的字符串长度。若 buff 参量为 NULL，则可以通过 pReqSize 返回待获取名字的字符串长度。

参量

prob

COPT 模型。

iQConstr`

二次约束的下标。

buff

用以获取字符串的数组。

buffSize

上述数组的大小。

pReqSize

指向存储待获取名字的最小字符串长度的指针。可以为 NULL 。

COPT_GetPSDConstrName

概要

```
int COPT_GetPSDConstrName(copt_prob *prob, int iPSDConstr, char
*buff, int buffSize, int *pReqSize)
```

描述

根据半定约束的下标获取其名字。若 buff 参量有效长度不够，且将待获取名字的前 buffSize 长度的子串返回，且通过 pReqSize 返回完全存储待获取名字需要的字符串长度。若 buff 参量为 NULL，则可以通过 pReqSize 返回待获取名字的字符串长度。

参量

prob

COPT 模型。

iPSDConstr`

半定约束的下标。

buff

用以获取字符串的数组。

buffSize

上述数组的大小。

pReqSize

指向存储待获取名字的最小字符串长度的指针。可以为 NULL 。

COPT_GetLMIConstrName

概要

```
int COPT_CALL COPT_GetLMIConstrName(copt_prob *prob, int iLMIConstr,
char *buff, int buffSize, int *pReqSize)
```

描述

根据 LMI 约束的下标获取其名字。若 buff 参量有效长度不够，且将待获取名字的前 buffSize 长度的子串返回，且通过 pReqSize 返回完全存储待获取名字需要的字符串长度。若 buff 参量为 NULL，则可以通过 pReqSize 返回待获取名字的字符串长度。

参量

prob

COPT 模型。

iLMIConstr

LMI 约束的下标。

`buff`

用以获取字符串的数组。

`buffSize`

上述数组的大小。

`pReqSize`

指向存储待获取名字的最小字符串长度的指针。可以为 `NULL` 。

COPT_GetLMIConstrRhs

概要

```
int COPT_GetLMIConstrRhs(copt_prob *prob, int num, const int *list,
int *constMatIdx)
```

描述

获取 `num` 个 LMI 约束的常数项对称矩阵。

参量

`prob`

COPT 模型。

`num`

待获取的 LMI 约束的个数。

`list`

待获取的 LMI 约束下标数组。

`constMatIdx`

待获取的 LMI 约束中常数项对称矩阵下标数组。

22.6.6 设置和获取参数

COPT_SetIntParam

概要

```
int COPT_SetIntParam(copt_prob *prob, const char *paramName, int
intParam)
```

描述

设置一个整数参数。

参量

`prob`

COPT 模型。

paramName

整数参数的名字。

intParam

整数参数的取值。

COPT_GetIntParam, COPT_GetIntParamDef/Min/Max

概要

```
int COPT_GetIntParam(copt_prob *prob, const char *paramName, int
*p_intParam)
```

```
int COPT_GetIntParamDef(copt_prob *prob, const char *paramName, int
*p_intParam)
```

```
int COPT_GetIntParamMin(copt_prob *prob, const char *paramName, int
*p_intParam)
```

```
int COPT_GetIntParamMax(copt_prob *prob, const char *paramName, int
*p_intParam)
```

描述

以上四个函数，分别可以获取一个整数参数的

当前值

默认值

最小值

最大值

参量

prob

COPT 模型。

paramName

整数参数的名字。

p_intParam

指向整数参数的值的指针。

COPT_SetDblParam

概要

```
int COPT_SetDblParam(copt_prob *prob, const char *paramName, double
dblParam)
```

描述

设置一个（双精度）浮点数参数。

参量

`prob`

COPT 模型。

`paramName`

浮点数参数的名字。

`dblParam`

浮点数参数的取值。

COPT_GetDblParam, COPT_GetDblParamDef/Min/Max

概要

```
int COPT_GetDblParam(copt_prob *prob, const char *paramName, double
*p_dblParam)
```

```
int COPT_GetDblParamDef(copt_prob *prob, const char *paramName,
double *p_dblParam)
```

```
int COPT_GetDblParamMin(copt_prob *prob, const char *paramName,
double *p_dblParam)
```

```
int COPT_GetDblParamMax(copt_prob *prob, const char *paramName,
double *p_dblParam)
```

描述

以上四个函数，分别可以获取一个（双精度）浮点数参数的

当前值

默认值

最小值

最大值

参量

`prob`

COPT 模型。

`paramName`

浮点数参数的名字。

p_dblParam

指向浮点数参数的值的指针。

COPT_ResetParam

概要

```
int COPT_ResetParam(copt_prob *prob)
```

描述

将模型的参数设置重置为默认设置。

参量

prob

COPT 模型。

COPT_WriteParam

概要

```
int COPT_WriteParam(copt_prob *prob, const char *parfilename)
```

描述

把用户设置过的所有参数写出到文件中。这个函数会把所有当前值和默认值不同的参数写到文件中。

参量

prob

COPT 模型。

parfilename

参数文件路径。

COPT_WriteParamStr

概要

```
int COPT_WriteParamStr(copt_prob *prob, char *str, int nStrSize, int
*pReqSize)
```

描述

把用户设置过的所有参数写出到字符流中。

参量

prob

COPT 模型。

`str`

参数设置字符流。

`nStrSize`

字符流缓冲区的大小。

`pReqSize`

保存参数设置的字符流最小空间大小。

COPT_ReadParam

概要

```
int COPT_ReadParam(copt_prob *prob, const char *parfilename)
```

描述

从文件中读取参数，并在 COPT 中设置这些参数。

参量

`prob`

COPT 模型。

`parfilename`

参数文件路径。

COPT_ReadParamStr

概要

```
int COPT_ReadParamStr(copt_prob *prob, const char *strParam)
```

描述

从字符流中读取参数设置，并在 COPT 中设置这些参数。

参量

`prob`

COPT 模型。

`strParam`

参数设置字符流。

22.6.7 获取属性

COPT_GetIntAttr

概要

```
int COPT_GetIntAttr(copt_prob *prob, const char *attrName, int
*p_intAttr)
```

描述

获取一个整数属性的值。

参量

`prob`

COPT 模型。

`attrName`

整数属性的名字。

`p_intAttr`

指向整数属性的取值的指针。

COPT_GetDblAttr

概要

```
int COPT_GetDblAttr(copt_prob *prob, const char *attrName, int
*p_dblAttr)
```

描述

获取一个（双精度）浮点数属性的值。

参量

`prob`

COPT 模型。

`attrName`

浮点数属性的名字。

`p_dblAttr`

指向浮点数属性的取值的指针。

22.6.8 日志函数

COPT_SetLogFile

概要

```
int COPT_SetLogFile(copt_prob *prob, char *logfilename)
```

描述

设置求解日志文件。

参量

`prob`

COPT 模型。

`logfilename`

COPT 求解日志文件。

COPT_SetLogCallback

概要

```
int COPT_SetLogCallback(copt_prob *prob, void (*logcb)(char *msg,
void *userdata), void *userdata)
```

描述

设置求解日志 Callback 函数。

参量

`prob`

COPT 模型。

`logcb`

求解日志 Callback 函数。

`userdata`

用户自定义数据。该数据将无修改地传递给求解日志 Callback 函数。

22.6.9 整数规划初始解功能函数

COPT_AddMipStart

概要

```
int COPT_AddMipStart(copt_prob *prob, int num, const int *list,
double *colVal)
```

描述

添加整数规划初始解信息。若 `list` 为 `NULL` ，则添加前 `num` 个变量（列）的初始解信息。每调用一次该函数，即添加一组初始解信息。

参量

`prob`

COPT 模型。

`num`

待添加初始解信息的变量（列）个数。

`list`

待添加初始解信息的变量（列）的下标列表。可以为 `NULL` 。

`colVal`

初始解信息数组。

COPT_ReadMst

概要

```
int COPT_ReadMst(copt_prob *prob, const char *mstfilename)
```

描述

从初始解文件中读取变量的取值，作为整数规划的初始解。

参量

`prob`

COPT 模型。

`mstfilename`

初始解文件路径。

COPT_WriteMst

概要

```
int COPT_WriteMst(copt_prob *prob, const char *mstfilename)
```

描述

将整数规划模型的求解结果或者已有的初始解信息写到初始解文件中。

参量

`prob`

COPT 模型。

`mstfilename`

初始解文件路径。

22.6.10 不可行模型 IIS 计算功能函数

COPT_ComputeIIS

概要

```
int COPT_ComputeIIS(copt_prob *prob)
```

描述

计算不可行模型的 IIS (Irreducible Inconsistent Subsystem) 。

参量

`prob`

COPT 模型。

COPT_WriteIIS

概要

```
int COPT_WriteIIS(copt_prob *prob, const char *iisfilename)
```

描述

把计算得到的 IIS 模型写出到文件中。

参量

`prob`

COPT 模型。

`iisfilename`

IIS 模型文件路径。

COPT_GetColLowerIIS

概要

```
int COPT_GetColLowerIIS(copt_prob *prob, int num, const int *list,
int *colLowerIIS)
```

描述

获取变量下边界的 IIS 状态。若 `list` 为 `NULL` ，则返回前 `num` 个变量下边界的 IIS 状态。

参量

`prob`

COPT 模型。

`num`

变量的个数。

`list`

变量的下标列表。可以为 `NULL` 。

`colLowerIIS`

返回的变量下边界的 IIS 状态数组。

COPT_GetColUpperIIS

概要

```
int COPT_GetColUpperIIS(copt_prob *prob, int num, const int *list,
int *colUpperIIS)
```

描述

获取变量上边界的 IIS 状态。若 `list` 为 `NULL` ，则返回前 `num` 个变量上边界的 IIS 状态。

参量

`prob`

COPT 模型。

`num`

变量的个数。

`list`

变量的下标列表。可以为 `NULL` 。

`colUpperIIS`

返回的变量上边界的 IIS 状态数组。

COPT_GetRowLowerIIS

概要

```
int COPT_GetRowLowerIIS(copt_prob *prob, int num, const int *list,
int *rowLowerIIS)
```

描述

获取约束下边界的 IIS 状态。若 `list` 为 `NULL` ，则返回前 `num` 个约束下边界的 IIS 状态。

参量

`prob`

COPT 模型。

`num`

约束的个数。

`list`

约束的下标列表。可以为 `NULL` 。

`rowLowerIIS`

返回的约束下边界的 IIS 状态数组。

COPT_GetRowUpperIIS

概要

```
int COPT_GetRowUpperIIS(copt_prob *prob, int num, const int *list,
int *rowUpperIIS)
```

描述

获取约束上边界的 IIS 状态。若 `list` 为 `NULL` ，则返回前 `num` 个约束上边界的 IIS 状态。

参量

`prob`

COPT 模型。

`num`

约束的个数。

`list`

约束的下标列表。可以为 `NULL` 。

`rowUpperIIS`

返回的约束上边界的 IIS 状态数组。

COPT_GetSOSIIS

概要

```
int COPT_GetSOSIIS(copt_prob *prob, int num, const int *list, int
*sosIIS)
```

描述

获取 SOS 约束的 IIS 状态。若 `list` 为 `NULL` ，则返回前 `num` 个 SOS 约束的 IIS 状态。

参量

`prob`

COPT 模型。

`num`

SOS 约束的个数。

`list`

SOS 约束的下标列表。可以为 `NULL` 。

`sosIIS`

返回的 SOS 约束的 IIS 状态数组。

COPT_GetIndicatorIIS

概要

```
int COPT_GetIndicatorIIS(copt_prob *prob, int num, const int *list,
int *indicatorIIS)
```

描述

获取 Indicator 约束的 IIS 状态。若 `list` 为 `NULL` ，则返回前 `num` 个 Indicator 约束的 IIS 状态。

参量

`prob`

COPT 模型。

`num`

Indicator 约束的个数。

`list`

Indicator 约束的下标列表。可以为 `NULL` 。

`indicatorIIS`

返回的 Indicator 约束的 IIS 状态数组。

22.6.11 可行化松弛计算功能函数

COPT_FeasRelax

概要

```
int COPT_FeasRelax(copt_prob *prob, double *colLowPen, double
*colUppPen, double *rowBndPen, double *rowUppPen)
```

描述

计算不可行模型的可行松弛。

参量

`prob`

COPT 模型。

colLowPen

变量下界的惩罚因子。若为 NULL，则表示不松弛变量下界；若 colLowPen 中惩罚因子为 COPT_INFINITY，则表示不松弛相应变量的下界。

colUppPen

变量上界的惩罚因子。若为 NULL，则表示不松弛变量上界；若 colUppPen 中惩罚因子为 COPT_INFINITY，则表示不松弛相应变量的下界。

rowBndPen

约束边界的惩罚因子。若为 NULL，则表示不松弛约束边界；若 rowBndPen 中惩罚因子为 COPT_INFINITY，则表示不松弛相应约束的边界。

rowUppPen

约束上边界的惩罚因子。若模型中存在双边约束，且 rowUppPen 不为 NULL，则表示约束上边界的惩罚因子；若 rowUppPen 中惩罚因子为 COPT_INFINITY，则表示不松弛相应约束的上边界。

注意：一般情况下，设置 rowUppPen 为 NULL 即可。

COPT_WriteRelax

概要

```
int COPT_WriteRelax(copt_prob *prob, const char *relaxfilename)
```

描述

把计算得到的可行化松弛模型写出到文件中。

参量

prob

COPT 模型。

relaxfilename

可行化松弛模型文件路径。

22.6.12 参数调优功能函数

COPT_Tune

概要

```
int COPT_Tune(copt_prob *prob)
```

描述

对模型进行参数调优。

参量

prob

COPT 模型。

COPT_LoadTuneParam

概要

```
int COPT_LoadTuneParam(copt_prob *prob, int idx)
```

描述

加载指定编号的参数调优结果到模型。

参量

prob

COPT 模型。

idx

参数调优结果的编号。

COPT_ReadTune

概要

```
int COPT_ReadTune(copt_prob *prob, const char *tunefilename)
```

描述

从调优文件中读取待调优参数组合到模型。

参量

prob

COPT 模型。

tunefilename

调优文件名。

COPT_WriteTuneParam

概要

```
int COPT_WriteTuneParam(copt_prob *prob, int idx, const char
*parfilename)
```

描述

输出指定编号的参数调优结果到参数文件中。

参量

prob

COPT 模型。

`idx`

参数调优结果的编号。

`parfilename`

参数文件名。

22.6.13 回调功能函数

Callback 功能函数只要在指定的 Callback Context 中才能被调用，对应关系罗列如下：

表 22.1: 回调功能函数

Context	Methods
COPT_CBCONTEXT_MIPSOL	*COPT_AddCallbackLazyConstr* *COPT_AddCallbackLazyConstrs* *COPT_AddCallbackSolution* *COPT_GetCallbackInfo*
COPT_CBCONTEXT_MIPRELAX	*COPT_AddCallbackUserCut* *COPT_AddCallbackUserCuts* *COPT_AddCallbackSolution* *COPT_GetCallbackInfo*
COPT_CBCONTEXT_MIPNODE	*COPT_AddCallbackSolution* *COPT_GetCallbackInfo*

COPT_AddLazyConstr

概要

```
int COPT_AddLazyConstr(
    copt_prob *prob,
    int nRowMatCnt,
    const int *rowMatIdx,
    const double *rowMatElem,
    char cRowSense,
    double dRowBound,
    double dRowUpper,
    const char *rowName)
```

描述

向模型中添加一个惰性约束。

参量

prob

COPT 模型。

nRowMatCnt

惰性约束系数矩阵中的非零元素的数量。

rowMatIdx

惰性约束系数矩阵中的非零元素的列编号。

rowMatElem

惰性约束系数矩阵中的非零元素的数值。

cRowSense

惰性约束类型。

可取值详见常量：约束类型

如果 cRowSense 是 0，则 dRowBound 和 dRowUpper 会被当做约束的上下边界。这是我们推荐的定义约束的方式。

如果 cRowSense 是有意义的量，则 dRowBound 和 dRowUpper 会被当做 RHS 和 **range** 。在此情况下，dRowUpper 仅在约束为 COPT_RANGE 类型时才会用到。且在此时，上下界分别为：

下界 dRowBound - dRowUpper

上界 dRowBound

dRowBound

惰性约束的下界或 RHS。

dRowUpper

惰性约束的上界或 **range** 。

rowName

惰性约束的名称。可以传入 NULL 。

COPT_AddLazyConstrs

概要

```
int COPT_AddLazyConstrs(
    copt_prob *prob,
    int nAddRow,
    const int *rowMatBeg,
    const int *rowMatCnt,
    const int *rowMatIdx,
```

```
    const double *rowMatElem,

    const char *rowSense,

    const double *rowBound,

    const double *rowUpper,

    char const *const *rowNames)
```

描述

向模型中批量添加多个惰性约束。

参量

prob

COPT 模型。

nAddRow

新添加的惰性约束个数（系数矩阵行数）。

rowMatBeg，rowMatCnt，rowMatIdx 和 rowMatElem

以行压缩存储格式提供惰性约束的系数矩阵。有关稀疏矩阵的压缩存储格式的具体示例，请参见 COPT_LoadProb 的 **其他信息**。

rowSense

惰性约束类型。

可取值详见常量：约束类型

如果 cRowSense 是 0，则 dRowBound 和 dRowUpper 会被当做约束的上下边界。这是我们推荐的定义约束的方式。

如果 cRowSense 是有意义的量，则 dRowBound 和 dRowUpper 会被当做 RHS 和 **range** 。在此情况下，dRowUpper 仅在约束为 COPT_RANGE 类型时才会用到。且在此时，上下界分别为：

下界 dRowBound - dRowUpper

上界 dRowBound

rowBound

惰性约束的下界或 RHS。

rowUpper

惰性约束的上界或 **range** 。

rowNames

惰性约束的名称。可以传入 NULL 。

COPT_AddUserCut

概要

```
int COPT_AddUserCut(
    copt_prob* prob,
    int nRowMatCnt,
    const int* rowMatIdx,
    const double* rowMmatElem,
    char cRowSense,
    double dRowBound,
    double dRowUpper,
    const char* rowName)
```

描述

向模型中添加一个割平面。

参量

`prob`

COPT 模型。

`nRowMatCnt`

割平面系数矩阵中的非零元素的数量。

`rowMatIdx`

割平面系数矩阵中的非零元素的列编号。

`rowMmatElem`

割平面系数矩阵中的非零元素的数值。

`cRowSense`

割平面约束类型。

可取值详见常量：约束类型

如果 `cRowSense` 是 0，则 `dRowBound` 和 `dRowUpper` 会被当做约束的上下边界。这是我们推荐的定义约束的方式。

如果 `cRowSense` 是有意义的量，则 `dRowBound` 和 `dRowUpper` 会被当做 RHS 和 **range** 。在此情况下，`dRowUpper` 仅在约束为 `COPT_RANGE` 类型时才会用到。且在此时，上下界分别为

下界 `dRowBound - dRowUpper`

上界 `dRowBound`

`dRowBound`

割平面约束的下界或 RHS。

`dRowUpper`

割平面约束的上界或 **range** 。

`rowName`

割平面的名称。可以传入 `NULL` 。

COPT_AddUserCuts

概要

```
int COPT_CALL COPT_AddUserCuts(
    copt_prob *prob,
    int nAddRow,
    const int *rowMatBeg,
    const int *rowMatCnt,
    const int *rowMatIdx,
    const double *rowMatElem,
    const char *rowSense,
    const double *rowBound,
    const double *rowUpper,
    char const *const *rowNames)
```

描述

向模型中批量添加多个割平面。

参量

`prob`

COPT 模型。

`nAddRow`

新添加的割平面个数（系数矩阵行数）。

`rowMatBeg`，`rowMatCnt`，`rowMatIdx` 和 `rowMatElem`

以行压缩存储格式提供割平面的系数矩阵。有关稀疏矩阵的压缩存储格式的具体示例，请参见 `COPT_LoadProb` 的 **其他信息**。

`rowSense`

割平面约束类型。

可取值详见常量：约束类型

如果 cRowSense 是 0，则 dRowBound 和 dRowUpper 会被当做约束的上下边界。这是我们推荐的定义约束的方式。

如果 cRowSense 是有意义的量，则 dRowBound 和 dRowUpper 会被当做 RHS 和 **range** 。在此情况下，dRowUpper 仅在约束为 COPT_RANGE 类型时才会用到。且在此时，上下界分别为：

下界 dRowBound - dRowUpper

上界 dRowBound

rowBound

割平面约束的下界或 RHS。

rowUpper

割平面约束的上界或 **range** 。

rowNames

割平面的名称。可以传入 NULL 。

COPT_SetCallback

概要

```
int COPT_SetCallback(
    copt_prob *prob,
    int (COPT_CALL *cb)(copt_prob *prob, void *cbdata, int cbctx,
    void *userdata),
    int cbctx,
    void *userdata)
```

描述

在 COPT 模型中，设置用户自定义的回调函数。

参量

prob

COPT 模型。

cb

回调函数。

cbctx

回调函数的触发点。可取值详见*常量：回调函数的触发点* 。

userdata

用户定义的数据。数据将不加修改地传递给 COPT。

COPT_GetCallbackInfo

概要

```
int COPT_GetCallbackInfo(void *cbdata, const char* cbinfo, void
*p_val)
```

描述

获取指定的回调信息值。

参量

`cbdata`

COPT 传递给回调的 `cbdata` 参数。

此参数必须未经修改地从用户定义的回调函数传递到 `COPT_GetCallbackInfo()` 。

`cbinfo`

待获取的回调信息名称。可取值详见信息：*Callback* 相关信息 。

`p_val`

指向回调信息值的指针。

COPT_AddCallbackSolution

概要

```
int COPT_AddCallbackSolution(void *cbdata, const double *sol,
double* p_objval)
```

描述

为指定变量设置自定义的可行解。

参量

`cbdata`

COPT 传递给回调的 `cbdata` 参数。

此参数必须未经修改地从用户定义的回调函数传递到 `COPT_AddCallbackSolution()` 。

`sol`

可行解的向量。

`p_objval`

指向可行解目标函数值的指针。

COPT_AddCallbackUserCut

概要

```
int COPT_AddCallbackUserCut(
    void *cbdata,
    int nRowMatCnt,
    const int *rowMatIdx,
    const double *rowMatElem,
    char cRowSense,
    double dRowRhs)
```

描述

向模型中添加一个割平面。

参量

cbdata

COPT 传递给回调的 cbdata 参数。

此参数必须未经修改地从用户定义的回调函数传递到 COPT_AddCallbackUserCut() 。

nRowMatCnt

割平面系数矩阵中的非零元素的数量。

rowMatIdx

割平面系数矩阵中的非零元素的列编号。

rowMatElem

割平面系数矩阵中的非零元素的数值。

cRowSense

割平面约束类型。支持 COPT_LESS_EQUAL ， COPT_GREATER_EQUAL ， COPT_EQUAL 和 COPT_FREE 。

通过回调函数添加的割平面仅支持单边。

dRowRhs

割平面约束的 RHS。

COPT_AddCallbackUserCuts

概要

```
int COPT_AddCallbackUserCuts(
    void *cbdata,
    int nAddRow,
    const int *rowMatBeg,
    const int *rowMatCnt,
    const int *rowMatIdx,
    const double *rowMatElem,
    const char *rowSense,
    const double *rowRhs)
```

描述

向模型中批量添加多个割平面。

参量

cbdata

COPT 传递给回调的 cbdata 参数。

此参数必须未经修改地从用户定义的回调函数传递到 COPT_AddCallbackUserCuts() 。

nAddRow

新添加的割平面个数（系数矩阵行数）。

rowMatBeg，rowMatCnt，rowMatIdx 和 rowMatElem

以行压缩存储格式提供割平面约束的系数矩阵。有关稀疏矩阵的压缩存储格式的具体示例，请参见 COPT_LoadProb 的 **其他信息**。

rowSense

割平面约束类型。支持 COPT_LESS_EQUAL ， COPT_GREATER_EQUAL ， COPT_EQUAL 和 COPT_FREE 。

通过回调函数添加的割平面仅支持单边。

rowRhs

割平面约束的 RHS。

COPT_AddCallbackLazyConstr

概要

```
int COPT_AddCallbackLazyConstr(
    void *cbdata,
    int nRowMatCnt,
    const int *rowMatIdx,
    const double *rowMatElem,
    char cRowSense,
    double dRowRhs)
```

描述

向模型中添加一个惰性约束。

参量

cbdata

COPT 传递给回调的 cbdata 参数。

此参数必须未经修改地从用户定义的回调函数传递到 COPT_AddCallbackLazyConstr() 。

nRowMatCnt

惰性约束系数矩阵中的非零元素的数量。

当 nRowMatCnt<=0 时，则表示 MIP 可行解会被直接拒绝，此时无需添加惰性约束。

rowMatIdx

惰性约束系数矩阵中的非零元素的列编号。

rowMatElem

惰性约束系数矩阵中的非零元素的数值。

cRowSense

惰性约束类型。支持 COPT_LESS_EQUAL ， COPT_GREATER_EQUAL ， COPT_EQUAL 和 COPT_FREE 。

通过回调函数添加的惰性约束仅支持单边。

dRowRhs

惰性约束的 RHS。

COPT_AddCallbackLazyConstrs

概要

```
int COPT_AddCallbackLazyConstrs(
    void *cbdata,
    int nAddRow,
    const int *rowMatBeg,
    const int *rowMatCnt,
    const int *rowMatIdx,
    const double *rowMatElem,
    const char *rowSense,
    const double *rowRhs)
```

描述

向模型中批量添加多个惰性约束。

参量

cbdata

COPT 传递给回调的 cbdata 参数。

此参数必须未经修改地从用户定义的回调函数传递到 COPT_AddCallbackLazyConstrs() 。

nAddRow

新添加的惰性约束个数（系数矩阵行数）。

当 nAddRow<=0 时，则表示 MIP 可行解会被直接拒绝，此时无需添加惰性约束。

rowMatBeg，rowMatCnt，rowMatIdx 和 rowMatElem

以行压缩存储格式提供惰性约束的系数矩阵。有关稀疏矩阵的压缩存储格式的具体示例，请参见 COPT_LoadProb 的 **其他信息**。

rowSense

惰性约束类型。支持 COPT_LESS_EQUAL ， COPT_GREATER_EQUAL ， COPT_EQUAL 和 COPT_FREE 。

通过回调函数添加的惰性约束仅支持单边。

rowRhs

惰性约束的 RHS。

22.6.14 其他 API 函数

COPT_GetBanner

概要

```
int COPT_GetBanner(char *buff, int buffSize)
```

描述

获取包含 COPT 版本等求解器基本信息的字符串。

参量

`buff`

用以获取字符串的数组。

`buffSize`

上述数组的大小。

COPT_GetRetcodeMsg

概要

```
int COPT_GetRetcodeMsg(int code, char *buff, int buffSize)
```

描述

获取解释 COPT 返回值的字符串。

参量

`code`

函数返回值。

`buff`

用以获取字符串的数组。

`buffSize`

上述数组的大小。

COPT_Interrupt

概要

```
int COPT_Interrupt(copt_prob *prob)
```

描述

中断当前求解的模型。

参量

`prob`

COPT 模型。

第 23 章　Python API 参考手册

为了方便用户对复杂应用场景进行建模并调用求解器进行求解，杉数求解器设计并提供了 Python 接口，本章节将对 Python 接口的功能进行阐述。

23.1 COPT 常数类

常数类定义了使用 Python 接口建模时必要的常数，包括一般常数、信息、参数和属性四大类常数。本节将依次阐述上述四类常数的内容与使用方法。

23.1.1 一般常数

一般常数主要定义了建模中常用的常数，包括优化方向、变量类型和求解状态等。用户通过 COPT 前缀访问一般常数，如 COPT.VERSION_MAJOR 表示软件大版本号信息。

一般常数类中的内容，详见一般常数章节。

23.1.2 属性

属性类常数包括优化模型和求解结果相关属性。

在 Python API 中，属性类常数中的内容，详见属性章节。

在 Python API 中，用户可以通过 COPT.Attr 前缀访问属性常数，如 COPT.Attr.Cols 表示模型中变量个数。

在 Python API 中，通过指定属性名称获取属性取值，具体请参考*Python Model 类* 。

- Model.getAttr() ，如：Model.getAttr("Cols") 获取模型中变量个数；

23.1.3 信息

信息类常数包括模型信息和求解结果相关信息。

Python API 信息类常数中的内容，详见信息章节。

在 Python API 中，用户可以通过 COPT.Info 前缀访问信息常数，如 COPT.Info.Obj 表示变量在目标函数中系数。可以通过指定信息名称获取或设置对象的信息取值：

- 获取变量或约束信息取值：Model.getInfo() / Var.getInfo() / Constraint.getInfo()

- 设置变量或约束信息取值：`Model.setInfo()` / `Var.setInfo()` / `Constraint.setInfo()`

23.1.4 Callback 信息

Python API Callback 信息常数中的内容，详见信息章节：*Callback* 相关信息。

在 Python API 中，Callback 相关的信息常数定义在 `CbInfo` 类里，用户可以通过前缀 `COPT.CbInfo.` 访问 Callback 信息常数。

例如，`COPT.CbInfo.BestObj` 是当前最优目标函数值。

在 Python API 中，用户可以通过 CallbackBase 类的函数，指定信息名称来获取回调信息的值。

例如，`CallbackBase.getInfo(COPT.CbInfo.BestObj)`：获取当前最优目标函数值。

23.1.5 参数

参数类常数表示杉数求解器的优化参数，优化参数控制 COPT 求解器优化算法的行为。

Python API 参数类常数中的内容，详见参数章节。

在 Python API 中，用户可通过 `COPT.Param` 前缀访问参数常数，如 `COPT.Param.TimeLimit` 表示模型的求解时间限制。

在 Python API 中，通过指定参数名称获取和设置参数取值。提供的函数如下，具体请参考*Python Model 类* 。

- 获取指定参数的详细信息（当前值/最大值/最小值）：`Model.getParamInfo()`
- 获取指定参数当前值：`Model.getParam()`
- 设置指定参数取值：`Model.setParam()`

23.2 优化建模类

优化建模类是杉数求解器的 Python 接口的主要功能类，它提供了丰富且易用的方法，便于用户快速构建复杂的实际场景下的优化模型。本节内容中将详细阐述各方法的功能与使用方式。

23.2.1 EnvrConfig 类

EnvrConfig 类是杉数求解器的客户端配置相关操作的封装，提供了以下成员方法：

EnvrConfig()

摘要

EnvrConfig()

描述

EnvrConfig 类的构造函数，该方法将创建并返回一个*EnvrConfig 类* 对象。

示例

```
# 创建杉数求解器客户端配置
envconfig = EnvrConfig()
```

EnvrConfig.set()

摘要

set(name, value)

描述

设置客户端配置参数。

参量

name

客户端配置参数名。

value

客户端配置参数值。

示例

```
# 设置客户端配置参数
envconfig.set(COPT.CLIENT_WAITTIME, 600)
```

23.2.2 Envr 类

Envr 类是杉数求解器的求解环境相关操作的封装，提供了以下成员方法：

Envr()

摘要

Envr(arg=None)

描述

Envr 类的构造函数，该方法将创建并返回一个*Envr 类* 对象。

参量

arg

授权文件的路径或客户端配置。可选参数，默认为 None 。

示例

```
# 创建杉数求解器求解环境
env = Envr()
```

Envr.createModel()

摘要

createModel(name="")

描述

创建优化模型，返回一个*Model 类* 对象。

参量

name

待创建优化模型的名称。可选参数，默认为 "" 。

示例

```
# 创建杉数求解器求解模型
model = env.createModel("coptprob")
```

Envr.close()

摘要

close()

描述

关闭与远程服务器的连接。

示例

```
# 关闭与远程服务器的连接
env.close()
```

23.2.3 Model 类

为了方便用户访问模型的相关属性和优化参数值，Model 类提供了形如 `Model.Rows` 的访问方式。目前支持的属性详见属性章节 部分，对于每个属性名，其大小写无关。

需要指出的是，对于线性或者整数规划模型，均通过 `Model.objval` 和 `Model.status` 获取模型的目标函数值和求解状态。

对于优化求解参数，用户还可以通过形如 `"Model.Param.TimeLimit = 10"` 的方式设置优化参数。目前支持的参数名称详见参数 部分，

Model 类是杉数求解器模型相关操作的封装，提供了以下成员方法：

Model.addVar()

摘要

```
addVar(lb=0.0, ub=COPT.INFINITY, obj=0.0, vtype=COPT.CONTINUOUS,
name="", column=None)
```

描述

添加一个变量到模型中，并返回创建的一个*Var 类* 对象。

参量

`lb`

变量的下界。可选参量，默认为 0.0。

`ub`

变量的上界。可选参量，默认为 `COPT.INFINITY` 。

`obj`

变量的目标函数系数。可选参量，默认为 0.0。

`vtype`

变量类型。可选参量，默认为 `COPT.CONTINUOUS` ，可取值详见变量类型 。

`name`

变量的名字。可选参量，默认为 `""` ，由求解器内部自动生成。

`column`

变量对应的列。可选参量，默认为 `None` 。

示例

```
# 添加一个连续变量
x = m.addVar()
# 添加一个二进制变量
y = m.addVar(vtype=COPT.BINARY)
```

(续下页)

(接上页)

```
# 添加一个整数变量，下界为-1.0，上界为1.0，目标函数系数为1.0，变量名为"z"
z = m.addVar(-1.0, 1.0, 1.0, COPT.INTEGER, "z")
```

Model.addVars()

摘要

addVars(*indices, lb=0.0, ub=COPT.INFINITY, obj=0.0, vtype=COPT.CONTINUOUS, nameprefix="C")

描述

添加一组变量到模型中，并返回一个*tupledict 类* 对象，其键为变量的下标，值为相应的*Var 类* 对象。

参量

*indices

变量的下标。

lb

变量的下界。可选参量，默认为 0.0。

ub

变量的上界。可选参量，默认为 COPT.INFINITY 。

obj

变量的目标函数系数。可选参量，默认为 0.0。

vtype

变量的类型。可选参量，默认为 COPT.CONTINUOUS ，可取值详见变量类型 。

nameprefix

变量的名称前缀。可选参量，默认为 "C"，其实际名称结合变量的下标自动生成。

示例

```
# 添加三维整数变量x，共计6个变量
x = m.addVars(1, 2, 3, vtype=COPT.INTEGER)
# 添加2个连续变量y，其下标由tl中的元素指定，变量名前缀为"tl"
tl = tuplelist([(0, 1), (1, 2)])
y  = m.addVars(tl, nameprefix="tl")
```

Model.addMVar()

摘要

addMVars(shape, lb=0.0, ub=COPT.INFINITY, obj=0.0, vtype=COPT.CONTINUOUS, nameprefix="")

描述

添加*MVar 类* 对象到模型中。它在矩阵建模中使用，可以像 NumPy 里多维数组一样运算，其形状和维度都有类似定义。

参量

shape

取值为整数，或者整数元组。表示*MVar 类* 对象的形状。

lb

变量的下界。可选参量，默认为 0.0。

ub

变量的上界。可选参量，默认为 COPT.INFINITY 。

obj

变量的目标函数系数。可选参量，默认为 0.0。

vtype

变量的类型。可选参量，默认为 COPT.CONTINUOUS ，可取值详见变量类型 。

nameprefix

变量的名称前缀。可选参量，默认为 ""，其实际名称结合变量的下标自动生成。

返回值

返回一个*MVar 类* 对象

示例

```
model.addMVar((2, 3), lb=0.0, nameprefix="mx")
```

Model.addConstr()

摘要

addConstr(lhs, sense=None, rhs=None, name="")

描述

添加一个线性约束到模型中，返回*Constraint 类* 对象；

添加一个半定约束到模型中，返回*PsdConstraint 类* 对象；

添加一个 Indicator 约束到模型中，返回*GenConstr 类* 对象；

添加 LMI 约束到模型中，返回*LmiConstraint 类* 对象。

若添加线性约束，则参数 `lhs` 可取值为*Var 类* 对象、*LinExpr 类* 对象或*ConstrBuilder 类* 对象；若添加半定约束，则参数 `lhs` 可取值为*PsdExpr 类* 对象或*PsdConstrBuilder 类* 对象；若添加 Indicator 约束，则参数 `lhs` 为*GenConstrBuilder 类* 对象，并忽略其它参数；若添加 LMI 约束，则参数 `lhs` 为*LmiExpr 类* 对象。

参量

`lhs`

线性约束的左端项或约束构建器。

`sense`

线性约束的类型。可选参量，默认为 `None` 。可取值详见*约束类型* 。

`rhs`

线性约束的右端项。可选参量，默认为 `None` 。可取值为常数、*Var 类* 对象或*LinExpr 类* 对象。

`name`

线性约束的名称。可选参量，默认为 `""`，由求解器内部自动生成。

示例

```
# 添加一个线性等式约束：x + y == 2
m.addConstr(x + y, COPT.EQUAL, 2)
# 添加一个线性大于等于约束：x + 2*y >= 3
m.addConstr(x + 2*y >= 3.0)
# 添加一个Indicator约束
m.addConstr((x == 1) >> (2*y + 3*z <= 4))
```

Model.addBoundConstr()

摘要

`addBoundConstr(expr, lb=-COPT.INFINITY, ub=COPT.INFINITY, name="")`

描述

添加一个带上下界的线性约束到模型中，并返回添加的*Constraint 类* 对象。

参量

`expr`

线性约束的表达式。可取值为*Var 类* 对象或*LinExpr 类* 对象。

`lb`

线性约束的下界。可选参量，默认为 `-COPT.INFINITY` 。

ub

线性约束的上界。可选参量，默认为 COPT.INFINITY 。

name

线性约束的名称。可选参量，默认为 ""，由求解器内部自动生成。

示例

```
# 添加线性双边约束：-1 <= x + y <= 1
m.addBoundConstr(x + y, -1.0, 1.0)
```

Model.addConstrs()

摘要

addConstrs(generator, nameprefix="R")

描述

添加一组线性约束到模型中。

若参数 generator 为整数，则返回一个*ConstrArray 类* 对象，其元素为 generator 个空*Constraint 类* 对象，用户需要进一步指定这些约束的具体信息；

若参数 generator 为表达式生成器，则返回一个*tupledict 类* 对象，其键为线性约束的下标，值为相应的*Constraint 类* 对象，每个迭代生成一个*Constraint 类* 对象；

若参数 generator 为矩阵表达式生成器，则返回一个*MConstr 类* 对象。

参量

generator

整数或（矩阵）表达式生成器。

nameprefix

线性约束的名称前缀。可选参量，默认为 "C"，其实际名称结合线性约束的下标自动生成。

示例

```
# 添加10个线性约束，每个约束形如：x[0] + y[0] >= 2.0
m.addConstrs(x[i] + y[i] >= 2.0 for i in range(10))
```

Model.addMConstr()

摘要

addMConstr(A, x, sense, b, nameprefix="")

描述

通过矩阵建模的方式，添加一组线性约束到模型中。如果这里 sense 取值为 COPT.LESS_EQUAL ，添加的约束为 $Ax <= b$ 。

更方便的是通过矩阵乘法生成*MLinExpr 类* 对象，再使用重载的比较运算符生成*MConstrBuilder 类* 对象，可以作为 Model.addConstrs() 的输入生成一组线性约束。

参量

A

参数 A 是一个二维的 NumPy 矩阵，或者 SciPy 列压缩矩阵（ csc_matrix ）或行压缩矩阵（ csr_matrix ）。

x

线性项对应的变量，可以是*MVar 类* 对象，*VarArray 类* 对象、列表、字典或*tupledict 类* 对象；如果为空，但参数 c 不空，则取模型中所有的变量。

sense

约束的类型。可取值参考*约束类型* 。

b

约束右边值，通常是浮点数，也可以是一组数，或者 NumPy 的一维数组。

nameprefix

约束名前缀。

返回值

返回一个*MConstr 类* 对象

示例

```
A = np.full((2, 3), 1)
mx = model.addMVar(3, nameprefix="mx")
mc = model.addMConstr(A, mx, 'L', 1.0, nameprefix="mc")
```

Model.addSOS()

摘要

addSOS(sostype, vars, weights=None)

描述

添加一个 SOS 约束到模型中，并返回添加的*SOS 类* 对象。

若参数 sostype 为*SOSBuilder 类* 对象，则参数 vars 和参数 weights 的取值将被忽略；若参数 sostype 为 SOS 约束类型，可取值为*SOS 约束类型*，则参数 vars 表示 SOS 约束的变量，可取值为*VarArray 类* 对象、列表、字典或*tupledict 类* 对象；若参数 weights 为 None ，则 SOS 约束的变量权重由求解器自动生成，否则采用用户传入的数据作为权重，可取值为列表、字典或*tupledict 类* 对象。

参量

sostype

SOS 约束类型或 SOS 约束构建器。

vars

SOS 约束的变量。

weights

SOS 约束的变量的权重。可选取值，默认为 None。

示例

```
# 添加一个SOS1约束，包括变量x和y，权重分别为1和2
m.addSOS(COPT.SOS_TYPE1, [x, y], [1, 2])
```

Model.addGenConstrIndicator()

摘要

addGenConstrIndicator(binvar, binval, lhs, sense=None, rhs=None)

描述

添加一个 Indicator 约束到模型中，并返回添加的*GenConstr 类* 对象。

若参数 lhs 为*ConstrBuilder 类* 对象，则参数 sense 和参数 rhs 的取值将被忽略；若参数 lhs 表示线性约束左端项，可取值为*Var 类* 对象或*LinExpr 类* 对象。

参量

binvar

Indicator 变量。

binval

Indicator 变量的取值，可取值为 True 或 False 。

lhs

Indicator 约束中线性约束的左端项或线性约束构建器。

sense

Indicator 约束中线性约束的类型。可选参量，默认为 None 。可取值详见约束类型 。

rhs

Indicator 约束中线性约束的右端项。可选参量，默认为 None 。可取值为常数。

示例

```
# 添加一个Indicator约束，当x为真时，线性约束 y + 2*z >= 3成立
m.addGenConstrIndicator(x, True, y + 2*z >= 3)
```

Model.addConeByDim()

摘要

addConeByDim(dim, ctype, vtype, nameprefix="ConeV")

描述

添加一个指定维度的二阶锥约束到模型中，并返回添加的*Cone* 类 对象。

参量

dim

二阶锥约束的维度。

ctype

二阶锥约束的类型。

vtype

二阶锥约束中变量的类型。

nameprefix

二阶锥约束中变量名称的前缀。可选取值，默认为 "ConeV"。

示例

```
# 添加一个5维的旋转二阶锥
m.addConeByDim(5, COPT.CONE_RQUAD, None)
```

Model.addCone()

摘要

addCone(vars, ctype)

描述

添加一个指定变量构成的二阶锥约束。

若参数 vars 为*ConeBuilder 类* 对象，则参数 ctype 的值将被忽略；若参数 vars 为变量，可取值为*VarArray 类* 对象、列表、字典或*tupledict 类* 对象，参数 ctype 表示二阶锥约束的类型。

参量

vars

构成二阶锥约束的变量。

ctype

二阶锥约束的类型。

示例

```
# 添加由[z, x, y]构成的标准二阶锥约束
m.addCone([z, x, y], COPT.CONE_QUAD)
```

Model.addQConstr()

摘要

addQConstr(lhs, sense=None, rhs=None, name="")

描述

添加一个线性约束或二次约束到模型中，并返回添加的*Constraint 类* 对象或*QConstraint 类* 对象。

若添加线性约束，则参数 lhs 可取值为*Var 类* 对象、*LinExpr 类* 对象或*ConstrBuilder 类* 对象；若添加二次约束，则参数 lhs 为*QConstrBuilder 类* 对象，或者*MQConstrBuilder 类* 对象，并忽略其它参数。

参量

lhs

约束的左端项或约束构建器。

sense

约束的类型。可选参量，默认为 None 。可取值详见*约束类型* 。

rhs

约束的右端项。可选参量，默认为 None 。可取值为常数、*Var 类* 对象、*LinExpr 类* 对象或*QuadExpr 类* 对象。

name

约束的名称。可选参量，默认为 ""，由求解器内部自动生成。

示例

```
# 添加一个线性等式约束：x + y == 2
m.addQConstr(x + y, COPT.EQUAL, 2)
# 添加一个二次约束：x*x + y*y <= 3
m.addQConstr(x*x + y*y <= 3.0)
```

Model.addMQConstr()

摘要

addMQConstr(Q, c, sense, rhs, xQ_L=None, xQ_R=None, xc=None, name="")

描述

通过矩阵建模的方式，添加一个二次约束到模型中。如果这里 sense 取值为 COPT.LESS_EQUAL，添加的约束为 $x_{Q_L}Qx_{Q_R} + cx_c <= rhs$ 。

更方便的是通过矩阵乘法生成*MQuadExpr 类* 对象，再使用重载的比较运算符生成*MQConstrBuilder 类* 对象，可以作为 Model.addQConstr() 的输入生成二次约束。

参量

Q

如果二次项非空，需要提供参数 Q，即一个二维的 NumPy 矩阵，或者 SciPy 列压缩矩阵（ csc_matrix ）或行压缩矩阵（ csr_matrix ）。

c

如果一次项非空，需要提供参数 c，即一个一维的 NumPy 数组，或者 Python 列表。

sense

约束的类型。可取值参考*约束类型* 。

rhs

约束右边值，通常是浮点数。

xQ_L

二次项左侧的变量，可以是*MVar 类* 对象，*VarArray 类* 对象、列表、字典或*tupledict 类* 对象；

如果为空，则取模型中所有的变量。

xQ_R

二次项右侧的变量，可以是*MVar 类* 对象，*VarArray 类* 对象、列表、字典或*tupledict 类* 对象；

如果为空，则取模型中所有的变量。

xc

线性项对应的变量，可以是*MVar 类* 对象，*VarArray 类* 对象、列表、字典或*tupledict 类* 对象；如果为空，但参数 c 不空，则取模型中所有的变量。

name

约束名。

返回值

返回一个*QConstraint 类* 对象

示例

```
Q = np.full((3, 3), 1)
mx = model.addMVar(3, nameprefix="mx")
mqc = model.addMQConstr(Q, None, 'L', 1.0, mx, mx, None, name="mqc")
```

Model.addPsdVar()

摘要

addPsdVar(dim, name="")

描述

添加一个半定变量。

参量

dim

半定变量的维度。

name

半定变量的名字。

示例

```
# 添加维度为3，名称为 "X" 的半定变量
m.addPsdVar(3, "X")
```

Model.addPsdVars()

摘要

addPsdVars(dims, nameprefix="PSDV")

描述

添加一组半定变量。

参量

dim

半定变量的维度。

nameprefix

半定变量名字的前缀。

示例

```
# 添加2个维度均为3的半定变量
m.addPsdVars([3, 3])
```

Model.addUserCut()

摘要

addUserCut(lhs, sense = None, rhs = None, name="")

描述

向模型中添加一个割平面。

参量

lhs

割平面约束的左端项。

可取值为*Var 类* 对象，*LinExpr 类* 对象，或者约束构建器（*ConstrBuilder 类* 对象）。

sense

割平面的约束类型。

可选参量，默认为 None 。可取值详见*约束类型* 。

rhs

割平面约束的右端项。

可选参量，默认为 None 。可取值为常数、*Var 类* 对象，或者*LinExpr 类* 对象。

name

割平面的名称。

可选参量。默认为""，由求解器内部自动生成。

示例

```
model.addUserCut(x+y <= 1)
model.addUserCut(x+y == [0,1])
```

Model.addUserCuts()

摘要

addUserCuts(generator, nameprefix="U")

描述

向模型中批量添加多个割平面。

参量

generator

一组割平面生成器。

可取值为一组线性约束构建器*ConstrBuilderArray 类* 对象，或多维线性约束生成器*MConstrBuilder 类* 对象。

nameprefix

割平面的名称前缀。

可选参量，默认为 "U" ，其实际名称结合割平面的下标自动生成。

示例

```
model.addUserCuts(x[i]+y[i] <= 1 for i in range(10))
```

Model.addLazyConstr()

摘要

addLazyConstr(lhs, sense = None, rhs = None, name="")

描述

向模型中添加一个惰性约束。

参量

lhs

惰性约束的左端项。

可取值为*Var 类* 对象，*LinExpr 类* 对象，或者约束构建器（*ConstrBuilder 类* 对象）。

sense

惰性约束的约束类型。

可选参量，默认为 None 。可取值详见*约束类型* 。

rhs

惰性约束的右端项。

可选参量，默认为 None 。可取值为常数、*Var 类* 对象，或者*LinExpr 类* 对象。

name

惰性约束的名称。

可选参量。默认为 "" ，由求解器内部自动生成。

示例

```
model.addLazyConstr(x+y <= 1)
model.addLazyConstr(x+y == [0,1])
```

Model.addLazyConstrs()

摘要

addLazyConstrs(generator, nameprefix="L")

描述

向模型中批量添加多个惰性约束。

参量

generator

一组惰性约束生成器。

可取值为一组线性约束构建器*ConstrBuilderArray 类* 对象，或多维线性约束生成器*MConstrBuilder 类* 对象。

nameprefix

惰性约束的名称前缀。

可选参量，默认为 "L" ，其实际名称结合惰性约束的下标自动生成。

示例

```
model.addLazyConstrs(x[i]+y[i] <= 1 for i in range(10))
```

Model.addGenConstrMin()

摘要

```
addGenConstrMin(resvar, vars, constant=None, name="")
```

描述

添加一条形如 $y = \min\{x_1, x_2, \cdots, x_n, c\}$ 的约束到模型中。

参量

`resvar`

等式左端项 `y` ，可取值为 `Var` 类对象。

`vars`

等式右端 min{} 函数的变量，可取值为 `list` 类对象。

`constant`

等式右端 min{} 函数中的常数项，可选参数，可取值为浮点数，默认值为 `None` 。

`name`

约束名称，可选参数，默认值为 `""` 。

返回值

返回一个 `GenConstrX` 类对象。

Model.addGenConstrMax()

摘要

```
addGenConstrMax(resvar, vars, constant=None, name="")
```

描述

添加一条形如 $y = \max\{x_1, x_2, \cdots, x_n, c\}$ 的约束到模型中。

参量

`resvar`

等式左端项 `y` ，可取值为 `Var` 类对象。

`vars`

等式右端 max{} 函数的变量，可取值为 `list` 类对象。

`constant`

等式右端 max{} 函数中的常数项，可选参量，可取值为浮点数，默认值为 `None` 。

`name`

约束名称，可选参量，默认值为 `""` 。

返回值

返回一个 GenConstrX 类对象。

Model.addGenConstrAbs()

摘要

```
addGenConstrAbs(resvar, argvar, name="")
```

描述

添加一条形如 $y = |x|$ 的约束到模型中。

参量

resvar

y ，可取值为 Var 类或 LinExpr 类对象。

argvar

x ，可取值为 Var 类对象。

name

约束名称，可选参数，默认值为 "" 。

返回值

返回一个 GenConstrX 类对象。

Model.addGenConstrAnd()

摘要

```
addGenConstrAnd(resvar, vars, name="")
```

描述

添加一条逻辑 and 约束，$y = x_1 \text{ and } x_2 \cdots \text{ and } x_n$ 至模型中。

参量

resvar

等式左端项 y ，可取值为二进制型 Var 类对象。

vars

逻辑运算符 and 连接的元素 $x_i, \text{for } i \in \{1, 2, \cdots, n\}$ ，可取值为 List 类（其中元素为二进制型 Var 类对象）。

name

约束名称，可选参数，默认值为 "" 。

返回值

返回一个 GenConstrX 类对象。

Model.addGenConstrOr()

摘要

addGenConstrOr(resvar, vars, name="")

描述

添加一条逻辑 or 约束，$y = x_1$ or $x_2 \cdots$ or x_n 至模型中。

参量

resvar

等式左端项 y ，可取值为二进制型 Var 类对象。

argvar

逻辑运算符 or 连接的元素 $x_i, \text{for } i \in \{1, 2, \cdots, n\}$ ，可取值为 List 类（其中元素为二进制型 Var 类对象）。

name

约束名称，可选参数，默认值为 "" 。

返回值

返回一个 GenConstrX 类对象。

Model.addGenConstrPWL()

摘要

addGenConstrPWL(xvar, yvar, xpts, ypts, name="")

描述

添加形如 $y = f(x)$ 的约束，其中分段线性函数定义为：

$$f(v) = \begin{cases} \tilde{y}_1 + \frac{\tilde{y}_2 - \tilde{y}_1}{\tilde{x}_2 - \tilde{x}_1}(v - \tilde{x}_1), & \text{if } v \le x_1 \\ \tilde{y}_i + \frac{\tilde{y}_{i+1} - \tilde{y}_i}{\tilde{x}_{i+1} - \tilde{x}_i}(v - \tilde{x}_i), & \text{if } \tilde{x}_i \le v \le \tilde{x}_{i+1} \\ \tilde{y}_n + \frac{\tilde{y}_n - \tilde{y}_{n-1}}{\tilde{x}_n - \tilde{x}_{n-1}}(v - \tilde{x}_n), & \text{if } v \ge \tilde{x}_n \end{cases}$$

参量

xvar

x ，可取值为 Var 类对象。

yvar

等式左端项 y ，可取值为 Var 类或 LinExpr 类对象。

xpts

$\tilde{\boldsymbol{x}}$ ，分段点的横坐标，需按照取值从小到大的顺序排列，可取值为 List 类。

ypts

$\tilde{y}$ ，分段点的纵坐标，可取值为 List 类。

name

约束名称，可选参数，默认值为 "" 。

返回值

返回一个 GenConstrX 类对象。

Model.addSparseMat()

摘要

addSparseMat(dim, rows, cols=None, vals=None)

描述

添加稀疏矩阵表示的对称矩阵。

参量

dim

对称矩阵的维度。

rows

对称矩阵非零元素的行索引。

cols

对称矩阵非零元素的列索引。

vals

对称矩阵非零元素值。

示例

```
# 添加维度为3的对称矩阵
m.addSparseMat(3, [0, 1, 2], [0, 1, 2], [2.0, 5.0, 8.0])
# 添加维度为2的对称矩阵
m.addSparseMat(2, [(0, 0, 3.0), (1, 0, 1.0)])
```

Model.addDenseMat()

摘要

addDenseMat(dim, vals)

描述

添加致密矩阵表示的对称矩阵。

参量

dim

对称矩阵的维度。

`vals`

对称矩阵非零元素值。可取值为：常数、列表。

示例

```
# 添加维度为3的全1对称矩阵
m.addDenseMat(3, 1.0)
```

Model.addDiagMat()

摘要

`addDiagMat(dim, vals, offset=None)`

描述

添加对角对称矩阵。

参量

`dim`

对称矩阵的维度。

`vals`

对称矩阵非零元素值。可取值为：常数、列表。

`offset`

对角元素位置偏移量，若为正值，则对角线向上偏移；若为负值，则对角线向下偏移。

示例

```
# 添加维度为3的单位矩阵
m.addDiagMat(3, 1.0)
```

Model.addOnesMat()

摘要

`addOnesMat(dim)`

描述

添加全 1 矩阵。

参量

`dim`

全 1 矩阵的维度。

示例

```
# 添加维度为3的全1矩阵
m.addOnesMat(3)
```

Model.addEyeMat()

摘要

addEyeMat(dim)

描述

添加单位矩阵。

参量

dim

单位矩阵的维度。

示例

```
# 添加维度为3的单位矩阵
m.addEyeMat(3)
```

Model.setObjective()

摘要

setObjective(expr, sense=None)

描述

设置模型的目标函数。

参量

expr

目标函数的表达式。可取值为常数、*Var 类* 对象、*LinExpr 类* 对象、*QuadExpr 类* 对象，以及*MLinExpr 类* 对象、*MQuadExpr 类* 对象。

sense

目标函数的优化方向。可选参量，默认为 None ，表示不改动模型的优化方向。模型的当前优化方向通过属性 ObjSense 查看。可取值详见*优化方向* 。

示例

```
# 设置目标函数为 x + y，优化方向为最大化
m.setObjective(x + y, COPT.MAXIMIZE)
```

Model.setMObjective()

摘要

setMObjective(Q, c, constant, xQ_L=None, xQ_R=None, xc=None, sense=None)

描述

通过矩阵建模的方式，设置模型的二次目标。可添加形如 $x_{Q_L}Qx_{Q_R}+cx_c+constant$ 的目标函数。

更方便的是通过矩阵乘法生成*MQuadExpr 类* 对象，可以作为 setObjective() 的输入设置目标函数。

参量

Q

如果二次项非空，需要提供参数 Q，即一个二维的 NumPy 矩阵，或者 SciPy 列压缩矩阵（ csc_matrix ）或行压缩矩阵（ csr_matrix ）。

c

如果一次项非空，需要提供参数 c，即一个一维的 NumPy 数组，或者 Python 列表。

constant

常数项，通常是浮点数。

xQ_L

二次项左侧的变量，可以是*MVar 类* 对象，*VarArray 类* 对象、列表、字典或*tupledict 类* 对象。如果为空，则取模型中所有的变量。

xQ_R

二次项右侧的变量，可以是*MVar 类* 对象，*VarArray 类* 对象、列表、字典或*tupledict 类* 对象。如果为空，则取模型中所有的变量。

xc

线性项对应的变量，可以是*MVar 类* 对象，*VarArray 类* 对象、列表、字典或*tupledict 类* 对象。如果为空，但参数 c 不空，则取模型中所有的变量。

sense

目标函数的优化方向。可选参量，默认为 None ，表示不改动模型的优化方向。模型的当前优化方向通过属性 ObjSense 查看。可取值详见优化方向 。

示例

```
Q = np.full((3, 3), 1)
mx = model.addMVar(3, nameprefix="mx")
my = model.addVars(3, nameprefix="my")
mqc = model.setMObjective(Q, None, 0.0, mx, my, None, sense=COPT.MINIMIZE)
```

Model.setObjSense()

摘要

setObjSense(sense)

描述

设置目标函数的优化方向。

参量

sense

目标函数的优化方向。可取值详见优化方向 。

示例

```
# 设置优化方向为最大化
m.setObjSense(COPT.MAXIMIZE)
```

Model.setObjConst()

摘要

setObjConst(const)

描述

设置目标函数的常数偏移量。

参量

const

目标函数常数偏移量。

示例

```
# 设置目标函数常数偏移为1.0
m.setObjConst(1.0)
```

Model.getObjective()

摘要

getObjective()

描述

获取模型的目标函数，返回一个*LinExpr 类* 对象。

示例

```
# 获取模型的目标函数表达式
obj = m.getObjective()
```

Model.delQuadObj()

摘要

delQuadObj()

描述

删除二次目标函数中的二次项。

示例

```
# 删除目标函数中的二次项
m.delQuadObj()
```

Model.delPsdObj()

摘要

delPsdObj()

描述

删除目标函数中的半定项。

示例

```
# 删除目标函数中的半定项
m.delPsdObj()
```

Model.getCol()

摘要

getCol(var)

描述

获取指定变量对应的列，返回一个*Column 类* 对象。

示例

```
# 获取变量x相应的列
col = m.getCol(x)
```

Model.getRow()

摘要

getRow(constr)

描述

获取指定线性约束对应的行，返回一个*LinExpr 类* 对象。

示例

```
# 获取线性约束conx相应的行
linexpr = m.getRow(conx)
```

Model.getQuadRow()

摘要

getQuadRow(qconstr)

描述

获取指定二次约束对应的行，返回一个*QuadExpr 类* 对象。

示例

```
# 获取二次约束qconx相应的行
quadexpr = m.getQuadRow(qconx)
```

Model.getPsdRow()

摘要

getPsdRow(constr)

描述

获取指定半定约束对应的行，返回一个*PsdExpr 类* 对象。

示例

```
# 获取半定约束psdcon相应的行
psdexpr = m.getPsdRow(psdcon)
```

Model.getVar()

摘要

getVar(idx)

描述

根据变量在模型中的下标获取相应的变量，返回一个*Var 类* 对象。

参量

idx

变量在系数矩阵中的下标。起始为 0。

示例

```
# 获取下标为1的变量
x = m.getVar(1)
```

Model.getVarByName()

摘要

getVarByName(name)

描述

根据变量的名称获取相应的变量，返回一个*Var 类* 对象。

参量

name

变量的名称。

示例

```
# 获取名称为"x"的变量
x = m.getVarByName("x")
```

Model.getVars()

摘要

getVars()

描述

获取模型中的全部变量，返回一个*VarArray 类* 对象。

示例

```
# 获取模型中的全部变量
vars = m.getVars()
```

Model.getConstr()

摘要

getConstr(idx)

描述

根据线性约束在模型中的下标获取相应的线性约束，返回一个*Constraint 类* 对象。

参量

idx

线性约束在系数矩阵中的下标。起始为 0。

示例

```
# 获取下标为1的线性约束
r = m.getConstr(1)
```

Model.getConstrByName()

摘要

getConstrByName(name)

描述

根据线性约束的名称获取相应的线性约束，返回一个*Constraint 类* 对象。

参量

name

线性约束的名称。

示例

```
# 获取名称为"r"的线性约束
r = m.getConstrByName("r")
```

Model.getConstrs()

摘要

getConstrs()

描述

获取模型中的全部线性约束，返回一个*ConstrArray 类* 对象。

示例

```
# 获取模型中的全部线性约束
cons = m.getConstrs()
```

Model.getConstrBuilders()

摘要

getConstrBuilders(constrs=None)

描述

获取当前模型中的线性约束相应的构建器。

若参数 constrs 为 None，则返回全部线性约束相应构建器组成的一个*ConstrBuilderArray 类*对象；若参数 constrs 为*Constraint 类*对象，则返回指定约束相应的*ConstrBuilder 类*对象；若参数 constrs 为列表或*ConstrArray 类*对象，则返回指定约束相应构建器组成的一个*ConstrBuilderArray 类*对象；若参数 constrs 为字典或*tupledict 类*对象，则返回键为指定约束的下标，值为指定约束相应的构建器组成的一个*tupledict 类*对象。

参量

constrs

指定的线性约束。可选参量，默认为 None 。

示例

```
# 获取所有的线性约束构建器
conbuilders = m.getConstrBuilders()
# 获取线性约束x相应的构建器
conbuilders = m.getConstrBuilders(x)
# 获取线性约束x和y相应的构建器
conbuilders = m.getConstrBuilders([x, y])
# 获取tupledict对象xx中的线性约束相应的构建器
conbuilders = m.getConstrBuilders(xx)
```

Model.getSOS(sos)

摘要

getSOS(sos)

描述

根据指定的 SOS 约束获取相应的 SOS 约束构建器，返回一个*SOSBuilder 类*对象。

参量

sos

指定的 SOS 约束。

示例

```
# 获取SOS约束sosx相应的构建器
sosbuilder = m.getSOS(sosx)
```

Model.getSOSs()

摘要

getSOSs()

描述

获取模型中的全部 SOS 约束，返回一个*SOSArray 类* 对象。

示例

```
# 获取模型中的全部SOS约束
soss = m.getSOSs()
```

Model.getSOSBuilders()

摘要

getSOSBuilders(soss=None)

描述

获取指定 SOS 约束相应的 SOS 约束构建器。

若参数 soss 为 None，则返回全部 SOS 约束相应构建器组成的一个*SOSBuilderArray 类* 对象；若参数 soss 为*SOS 类* 对象，则返回指定 SOS 约束相应的*SOSBuilder 类* 对象；若参数 soss 为列表或*SOSArray 类* 对象，则返回指定 SOS 约束相应构建器组成的一个*SOSBuilderArray 类* 对象。

参量

soss

指定的 SOS 约束。可选参量，默认为 None 。

示例

```
# 获取模型中所有SOS约束相应的构建器
soss = m.getSOSBuilders()
```

Model.getGenConstrIndicator()

摘要

getGenConstrIndicator(genconstr)

描述

获取指定 Indicator 约束相应的 Indicator 约束构建器，返回一个*GenConstrBuilder 类* 对象。

参量

genconstr

指定的 Indicator 约束。

示例

```
# 获取Indicator约束genx相应的构建器
indic = m.getGenConstrIndicator(genx)
```

Model.getCones()

摘要

getCones()

描述

获取模型中的全部二阶锥约束，返回一个*ConeArray 类* 对象。

示例

```
# 获取模型中的全部二阶锥约束
cones = m.getCones()
```

Model.getConeBuilders()

摘要

getConeBuilders(cones=None)

描述

获取指定二阶锥约束相应的二阶锥约束构建器。

若参数 cones 为 None，则返回全部二阶锥约束相应构建器组成的一个*ConeBuilderArray 类* 对象；若参数 cones 为*Cone 类* 对象，则返回指定二阶锥约束相应的*ConeBuilder 类* 对象；若参数 cones 为列表或*ConeArray 类* 对象，则返回指定二阶锥约束相应构建器组成的一个*ConeBuilderArray 类* 对象。

参量

cones

指定的二阶锥约束。可选参量，默认为 None 。

示例

```
# 获取模型中所有二阶锥约束相应的构建器
cones = m.getConeBuilders()
```

Model.getQConstr()

摘要

getQConstr(idx)

描述

根据二次约束在模型中的下标获取相应的二次约束，返回一个*QConstraint 类* 对象。

参量

idx

二次约束在模型中的下标。起始为 0。

示例

```
# 获取下标为1的二次约束
qr = m.getQConstr(1)
```

Model.getQConstrByName()

摘要

getQConstrByName(name)

描述

根据二次约束的名称获取相应的二次约束，返回一个*QConstraint 类* 对象。

参量

name

二次约束的名称。

示例

```
# 获取名称为"qr"的二次约束
qr = m.getQConstrByName("qr")
```

Model.getQConstrs()

摘要

getQConstrs()

描述

获取模型中的全部二次约束，返回一个*QConstrArray 类* 对象。

示例

```
# 获取模型中的全部二次约束
qcons = m.getQConstrs()
```

Model.getQConstrBuilders()

摘要

getQConstrBuilders(qconstrs=None)

描述

获取当前模型中的二次约束相应的构建器。

若参数 qconstrs 为 None，则返回全部二次约束相应构建器组成的一个*QConstrBuilderArray 类*对象；若参数 qconstrs 为*QConstraint 类*对象，则返回指定约束相应的*QConstrBuilder 类*对象；若参数 qconstrs 为列表或*QConstrArray 类*对象，则返回指定约束相应构建器组成的一个*QConstrBuilderArray 类*对象；若参数 qconstrs 为字典或*tupledict 类*对象，则返回键为指定约束的下标，值为指定约束相应的构建器组成的一个*tupledict 类*对象。

参量

qconstrs

指定的二次约束。可选参量，默认为 None 。

示例

```
# 获取所有的二次约束构建器
qconbuilders = m.getQConstrBuilders()
# 获取二次约束qx相应的构建器
qconbuilders = m.getQConstrBuilders(qx)
# 获取二次约束x和y相应的构建器
qconbuilders = m.getQConstrBuilders([qx, qy])
# 获取tupledict对象qxx中的二次约束相应的构建器
qconbuilders = m.getQConstrBuilders(qxx)
```

Model.getPsdVar()

摘要

getPsdVar(idx)

描述

根据半定变量在模型中的下标获取相应的半定变量，返回一个*PsdVar 类*对象。

参量

idx

半定变量在模型中的下标。起始为 0。

示例

```
# 获取下标为1的半定变量
x = m.getPsdVar(1)
```

Model.getPsdVarByName()

摘要

getPsdVarByName(name)

描述

根据半定变量的名称获取相应的半定变量，返回一个*PsdVar 类* 对象。

参量

name

半定变量的名称。

示例

```
# 获取名称为"x"的半定变量
x = m.getPsdVarByName("x")
```

Model.getPsdVars()

摘要

getPsdVars()

描述

获取模型中的全部半定变量，返回一个*PsdVarArray 类* 对象。

示例

```
# 获取模型中的全部半定变量
vars = m.getPsdVars()
```

Model.getPsdConstr()

摘要

getPsdConstr(idx)

描述

根据半定约束在模型中的下标获取相应的半定约束，返回一个*PsdConstraint 类* 对象。

参量

idx

半定约束在模型中的下标。起始为 0。

示例

```
# 获取下标为1的半定约束
r = m.getPsdConstr(1)
```

Model.getPsdConstrByName()

摘要

getPsdConstrByName(name)

描述

根据半定约束的名称获取相应的半定约束，返回一个*PsdConstraint 类* 对象。

参量

name

半定约束的名称。

示例

```
# 获取名称为"r"的半定约束
r = m.getPsdConstrByName("r")
```

Model.getPsdConstrs()

摘要

getPsdConstrs()

描述

获取模型中的全部半定约束，返回一个*PsdConstrArray 类* 对象。

示例

```
# 获取模型中的全部半定约束
cons = m.getPsdConstrs()
```

Model.getPsdConstrBuilders()

摘要

getPsdConstrBuilders(constrs=None)

描述

获取当前模型中的半定约束相应的构建器。

若参数 constrs 为 None，则返回全部半定约束相应构建器组成的一个*PsdConstrBuilderArray 类* 对象；若参数 constrs 为*PsdConstraint 类* 对象，则返回指定半定约束相应的*PsdConstrBuilder 类* 对象；若参数 constrs 为列表或*PsdConstrArray 类* 对象，则返回指定半定约束相应构建器组成的一

个*PsdConstrBuilderArray 类* 对象；若参数 constrs 为字典或*tupledict 类* 对象，则返回键为指定半定约束的下标，值为指定半定约束相应的构建器组成的一个*tupledict 类* 对象。

参量

constrs

指定的半定约束。可选参量，默认为 None 。

示例

```
# 获取所有的半定约束构建器
conbuilders = m.getPsdConstrBuilders()
# 获取半定约束x相应的构建器
conbuilders = m.getPsdConstrBuilders(x)
# 获取半定约束x和y相应的构建器
conbuilders = m.getPsdConstrBuilders([x, y])
# 获取tupledict对象xx中的半定约束相应的构建器
conbuilders = m.getPsdConstrBuilders(xx)
```

Model.getLmiRow()

摘要

getLmiRow(constr)

描述

获取参与指定 LMI 约束的 LMI 表达式，包括变量和对应的系数矩阵。

参量

constr

指定的约束。

示例

```
# 获取LMI约束c中的表达式
expr = m.getLmiRow(c)
```

Model.getLmiConstr()

摘要

getLmiConstr(idx)

描述

获取模型中指定索引对应的 LMI 约束。

参量

idx

LMI 约束在模型中的下标。起始为 0。

示例

```
# 获取模型中第1个LMI约束
coeff = m.getLmiConstr(1)
```

Model.getLmiConstrByName()

摘要

getLmiConstrByName(name)

描述

获取模型中指定名称的 LMI 约束。

参量

name

指定的 LMI 约束名称。

示例

```
# 获取模型中名称为r1的LMI约束
name = m.getLmiConstrByName("r1")
```

Model.getLmiConstrs()

摘要

getLmiConstrs()

描述

获取模型中所有的 LMI 约束。返回 LMI 约束构成的*LmiConstrArray 类* 对象。

Model.getLmiRhs()

摘要

getLmiRhs(constr)

描述

获取指定 LMI 约束的常数项。返回*SymMatrix 类* 对象。

参量

constr

指定的 LMI 约束。

Model.setLmiRhs()

摘要

setLmiRhs(constr, mat)

描述

设置指定 LMI 约束的常数项。

参量

constr

指定的 LMI 约束。

mat

新的常数项矩阵。

示例

```
# 设置LMI约束con的常数项对称矩阵为D
m.setLmiRhs(con, D)
```

Model.getLmiSolution()

摘要

getLmiSolution()

描述

获取 LMI 约束的取值和对偶值。

Model.getLmiSlacks()

摘要

getLmiSlacks()

描述

获取 LMI 约束全部松弛变量的取值，返回一个列表对象。

Model.getLmiDuals()

摘要

getLmiDuals()

描述

获取 LMI 约束全部对偶变量的取值，返回一个列表对象。

Model.getCoeff()

摘要

getCoeff(constr, var)

描述

获取变量在线性约束、半定约束或 LMI 约束中的系数。

参量

constr

指定的线性约束、半定约束或 LMI 约束。

var

指定的变量或半定变量。

示例

```
# 获取变量x在约束c1中的系数
coeff1 = m.getCoeff(c1, x)
# 获取半定变量X在约束c2中的系数
coeff2 = m.getCoeff(c2, X)
```

Model.setCoeff()

摘要

setCoeff(constr, var, newval)

描述

设置变量在线性约束、半定约束或 LMI 约束中的系数。

参量

constr

指定的线性约束、半定约束或 LMI 约束。

var

指定的变量或半定变量。

newval

待设置的新系数或系数对称矩阵。

示例

```
# 设置变量x在约束c中的系数为1.0
m.setCoeff(c, x, 1.0)
```

Model.getA()

摘要

getA()

描述

获取模型的系数矩阵，返回一个 scipy.sparse.csc_matrix 对象。该方法依赖 scipy 工具包。

示例

```
# 获取模型的系数矩阵
A = model.getA()
```

Model.loadMatrix()

摘要

loadMatrix(c, A, lhs, rhs, lb, ub, vtype=None)

描述

加载矩阵和向量信息构建模型。该方法依赖 scipy 工具包。

参量

c

目标函数系数。若为 None，则表示目标函数系数全为 0。

A

系数矩阵。类型要求为 scipy.sparse.csc_matrix。

lhs

约束的下边界。

rhs

约束的上边界。

lb

变量的下边界。若为 None，则表示下边界全为 0。

ub

变量的上边界。若为 None，则表示上边界全为 COPT.INFINITY。

vtype

变量类型。默认为 None，表示全部为连续变量。

示例

```
# 矩阵数据构建模型
m.loadMatrix(c, A, lhs, rhs, lb, ub)
```

Model.getLpSolution()

摘要

getLpSolution()

描述

获取线性规划模型的变量取值、松弛变量取值、对偶变量取值和变量的 Reduced cost，返回一个四元元组对象，元组中每个元素为一个列表对象。

示例

```
# 获取线性规划模型的解
values, slacks, duals, redcosts = m.getLpSolution()
```

Model.setLpSolution()

摘要

setLpSolution(values, slack, duals, redcost)

描述

设置线性规划模型的变量取值、松弛变量取值、对偶变量取值和变量的 Reduced cost。

参量

values

变量取值。

slack

松弛变量取值。

duals

对偶变量取值。

redcost

变量的 Reduced cost。

示例

```
# 设置线性规划模型的解
m.setLpSolution(values, slack, duals, redcost)
```

Model.getValues()

摘要

getValues()

描述

获取线性或整数规划模型的全部变量取值，返回一个列表对象。

示例

```
# 获取模型中所有变量的取值
values = m.getValues()
```

Model.getRedcosts()

摘要

getRedcosts()

描述

获取线性规划模型全部变量的 Reduced cost，返回一个列表对象。

示例

```
# 获取模型中所有变量的Reduced cost
redcosts = m.getRedcosts()
```

Model.getSlacks()

摘要

getSlacks()

描述

获取线性规划全部松弛变量的取值，返回一个列表对象。

示例

```
# 获取模型中所有松弛变量的取值
slacks = m.getSlacks()
```

Model.getDuals()

摘要

getDuals()

描述

获取线性规划全部对偶变量的取值，返回一个列表对象。

示例

```
# 获取模型中所有对偶变量的取值
duals = m.getDuals()
```

Model.getVarBasis()

摘要

getVarBasis(vars=None)

描述

获取指定变量的基状态。

若参数 vars 为 None，则返回全部变量的基状态组成的一个列表对象；若参数 vars 为*Var 类* 对象，则返回指定变量的基状态；若参数 vars 为列表或*VarArray 类* 对象，则返回指定变量的基状态组成的一个列表对象；若参数 vars 为字典或*tupledict 类* 对象，则返回键为指定变量的下标，值为指定变量的基状态组成的一个*tupledict 类* 对象。

参量

vars

指定的变量。可选参量，默认为 None 。

示例

```
# 获取模型中全部变量的基状态
varbasis = m.getVarBasis()
# 获取变量x和y的基状态
varbasis = m.getVarBasis([x, y])
# 获取tupledict对象xx中的变量相应的基状态
varbasis = m.getVarBasis(xx)
```

Model.getConstrBasis()

摘要

getConstrBasis(constrs=None)

描述

获取线性规划中线性约束的基状态。

若参数 constrs 为 None，则返回全部线性约束的基状态组成的一个列表对象；若参数 constrs 为*Constraint 类* 对象，则返回指定线性约束的基状态；若参数 constrs 为列表或*ConstrArray 类* 对象，则返回指定线性约束的基状态组成的一个列表对象；若参数 constrs 为字典或*tupledict 类* 对象，则返回键为指定线性约束的下标，值为指定约束的基状态组成的一个*tupledict 类* 对象。

参量

constrs

指定的线性约束。可选参量，默认为 None 。

示例

```
# 获取模型中全部线性约束的基状态
conbasis = m.getConstrBasis()
# 获取模型中线性约束r0和r1相应的基状态
conbasis = m.getConstrBasis([r0, r1])
# 获取tupledict对象rr中的线性约束相应的基状态
conbasis = m.getConstrBasis(rr)
```

Model.getPoolObjVal()

摘要

getPoolObjVal(isol)

描述

获取解池中第 isol 个解的目标函数值，返回一个常数。

参量

isol

解池中解的索引。

示例

```
# 获取第2个解的目标函数值
objval = m.getPoolObjVal(2)
```

Model.getPoolSolution()

摘要

getPoolSolution(isol, vars)

描述

获取解池中第 isol 个解中指定变量的取值。

若参数 vars 为*Var 类* 对象，则返回指定变量的取值；若参数 vars 为列表或*VarArray 类* 对象，则返回指定变量的取值组成的一个列表对象；若参数 vars 为字典或*tupledict 类* 对象，则返回键为指定变量的下标，值为指定变量的取值组成的一个*tupledict 类* 对象。

参量

isol

解池中解的索引。

vars

指定的变量。

示例

```
# 获取第2个解中变量x的值
xval = m.getPoolSolution(2, x)
```

Model.getVarLowerIIS()

摘要

getVarLowerIIS(vars)

描述

获取指定变量下边界的 IIS 状态。

若参数 vars 为*Var 类* 对象，则返回指定变量下边界的 IIS 状态；若参数 vars 为列表或*VarArray 类* 对象，则返回指定变量下边界的 IIS 状态组成的一个列表对象；若参数 vars 为字典或*tupledict 类* 对象，则返回键为指定变量的下标，值为指定变量下边界的 IIS 状态组成的一个*tupledict 类* 对象。

参量

vars

指定的变量。

示例

```
# 获取变量x和y的下边界的IIS状态
lowerIIS = m.getVarLowerIIS([x, y])
```

(续下页)

(接上页)

```
# 获取tupledict对象xx中的变量下边界相应的IIS状态
lowerIIS = m.getVarLowerIIS(xx)
```

Model.getVarUpperIIS()

摘要

getVarUpperIIS(vars)

描述

获取指定变量上边界的 IIS 状态。

若参数 vars 为*Var 类* 对象，则返回指定变量上边界的 IIS 状态；若参数 vars 为列表或*VarArray 类* 对象，则返回指定变量上边界的 IIS 状态组成的一个列表对象；若参数 vars 为字典或*tupledict 类* 对象，则返回键为指定变量的下标，值为指定变量上边界的 IIS 状态组成的一个*tupledict 类* 对象。

参量

vars

指定的变量。

示例

```
# 获取变量x和y的上边界的IIS状态
upperIIS = m.getVarUpperIIS([x, y])
# 获取tupledict对象xx中的变量上边界相应的IIS状态
upperIIS = m.getVarUpperIIS(xx)
```

Model.getConstrLowerIIS()

摘要

getConstrLowerIIS(constrs)

描述

获取指定约束下边界的 IIS 状态。

若参数 constrs 为*Constraint 类* 对象，则返回指定约束下边界的 IIS 状态；若参数 constrs 为列表或*ConstrArray 类* 对象，则返回指定约束下边界的 IIS 状态组成的一个列表对象；若参数 constrs 为字典或*tupledict 类* 对象，则返回键为指定约束的下标，值为指定约束下边界的 IIS 状态组成的一个*tupledict 类* 对象。

参量

constrs

指定的约束。

示例

```
# 获取模型中约束r0和r1相应下边界的IIS状态
lowerIIS = m.getConstrLowerIIS([r0, r1])
# 获取tupledict对象rr中的约束下边界相应的IIS状态
lowerIIS = m.getConstrLowerIIS(rr)
```

Model.getConstrUpperIIS()

摘要

getConstrUpperIIS(constrs)

描述

获取指定约束上边界的 IIS 状态。

若参数 constrs 为*Constraint 类* 对象，则返回指定约束上边界的 IIS 状态；若参数 constrs 为列表或*ConstrArray 类* 对象，则返回指定约束上边界的 IIS 状态组成的一个列表对象；若参数 constrs 为字典或*tupledict 类* 对象，则返回键为指定约束的下标，值为指定约束上边界的 IIS 状态组成的一个*tupledict 类* 对象。

参量

constrs

指定的约束。

示例

```
# 获取模型中约束r0和r1相应上边界的IIS状态
upperIIS = m.getConstrUpperIIS([r0, r1])
# 获取tupledict对象rr中的约束上边界相应的IIS状态
upperIIS = m.getConstrUpperIIS(rr)
```

Model.getSOSIIS()

摘要

getSOSIIS(soss)

描述

获取指定 SOS 约束的 IIS 状态。

若参数 soss 为*SOS 类* 对象，则返回指定 SOS 约束的 IIS 状态；若参数 soss 为列表或*SOSArray 类* 对象，则返回指定 SOS 约束的 IIS 状态组成的一个列表对象；若参数 soss 为字典或*tupledict 类* 对象，则返回键为指定 SOS 约束的下标，值为指定 SOS 约束的 IIS 状态组成的一个*tupledict 类* 对象。

参量

soss

指定的 SOS 约束。

示例

```
# 获取模型中SOS约束r0和r1相应的IIS状态
sosIIS = m.getSOSIIS([r0, r1])
# 获取tupledict对象rr中的SOS约束相应的IIS状态
sosIIS = m.getSOSIIS(rr)
```

Model.getIndicatorIIS()

摘要

getIndicatorIIS(genconstrs)

描述

获取指定 Indicator 约束的 IIS 状态。

若参数 genconstrs 为*GenConstr 类* 对象，则返回指定 Indicator 约束的 IIS 状态；若参数 genconstrs 为列表或*GenConstrArray 类* 对象，则返回指定 Indicator 约束的 IIS 状态组成的一个列表对象；若参数 genconstrs 为字典或*tupledict 类* 对象，则返回键为指定 Indicator 约束的下标，值为指定 Indicator 约束的 IIS 状态组成的一个*tupledict 类* 对象。

参量

genconstrs

指定的 Indicator 约束。

示例

```
# 获取模型中Indicator约束r0和r1相应的IIS状态
indicatorIIS = m.getIndicatorIIS([r0, r1])
# 获取tupledict对象rr中的Indicator约束相应的IIS状态
indicatorIIS = m.getIndicatorIIS(rr)
```

Model.getAttr()

摘要

getAttr(attrname)

描述

获取指定的模型属性值，返回一个常数。

参量

attrname

指定的属性名。可取值详见属性章节 部分。

示例

```
# 获取目标函数的常数项
objconst = m.getAttr(COPT.Attr.ObjConst)
```

Model.getInfo()

摘要

getInfo(infoname, args)

描述

获取指定的信息值。

若参数 args 为*Var 类* 对象或*Constraint 类* 对象，则返回指定变量或约束的信息值常数；若参数 args 为列表、*VarArray 类* 对象或*ConstrArray 类* 对象，则返回指定变量或约束的信息值组成的一个列表对象；若参数 args 为字典或*tupledict 类* 对象，则返回键为指定变量或约束的下标，值为指定变量或约束的信息值组成的一个*tupledict 类* 对象；若参数 args 为*MVar 类* 对象或*MConstr 类* 对象，则返回指定变量或约束的信息值组成的一个 numpy.ndarray 对象。

参量

infoname

待获取信息名。可取值详见*信息* 部分。

args

待获取信息的变量或者约束。

示例

```
# 获取模型中全部线性约束的下界信息
lb = m.getInfo(COPT.Info.LB, m.getConstrs())
# 获取变量x和y的取值信息
sol = m.getInfo(COPT.Info.Value, [x, y])
# 获取tupledict对象shipconstr中的线性约束相应的对偶变量取值信息
dual = m.getInfo(COPT.Info.Dual, shipconstr)
```

Model.getVarType()

摘要

getVarType(vars)

描述

获取指定变量的类型。

若参数 vars 为*Var 类* 对象，则返回指定变量的类型；若参数 vars 为列表或*VarArray 类* 对象，则返回指定变量的类型组成的一个列表对象；若参数 vars 为字典或*tupledict 类* 对象，则返回键为指定变量的下标，值为指定变量的类型组成的一个*tupledict 类* 对象。

参量

vars

指定的变量。

示例

```
# 获取变量x的类型
xtype = m.getVarType(x)
# 获取变量x、y和z的类型
xtype = m.getVarType([x, y, z])
# 获取tupledict对象xdict中的变量的类型
xtype = m.getVarType(xdict)
```

Model.getParam()

摘要

getParam(paramname)

描述

获取指定的优化参数的当前值，返回一个常数。

参量

paramname

指定的优化参数名。可取值详见参数 部分。

示例

```
# 获取优化求解时间限制的当前值
timelimit = m.getParam(COPT.Param.TimeLimit)
```

Model.getParamInfo()

摘要

getParamInfo(paramname)

描述

获取指定的优化参数的信息，返回一个元组对象，其元素分别为：参数名、当前值、默认值、最小值和最大值。

参量

paramname

指定的优化参数名。可取值详见参数 部分。

示例

```
# 获取优化求解时间限制参数的信息
pname, pcur, pdef, pmin, pmax = m.getParamInfo(COPT.Param.TimeLimit)
```

Model.setBasis()

摘要

setBasis(varbasis, constrbasis)

描述

设置线性规划中全部变量和线性约束的基状态。参数 varbasis 和 constrbasis 为列表对象，其元素数目分别为模型中的变量总数和线性约束总数。

参量

varbasis

变量的基状态。

constrbasis

线性约束的基状态。

示例

```
# 设置模型中全部变量和线性约束的基状态
m.setBasis(varbasis, constrbasis)
```

Model.setSlackBasis()

摘要

setSlackBasis()

描述

设置线性规划的基为松弛基。

示例

```
# 设置基状态为松弛基
m.setSlackBasis()
```

Model.setVarType()

摘要

setVarType(vars, vartypes)

描述

设置指定变量的类型。

若参数 vars 为*Var 类*对象，则参数 vartypes 为变量类型常量；若参数 vars 为字典或*tupledict 类*对象，则参数 vartypes 可为变量类型常量、字典或*tupledict 类*对象；若参数 vars 为列表或*VarArray 类*对象，则参数 vartypes 可为变量类型常量或列表对象。

参量

vars

指定的变量。

vartypes

指定的变量类型。

示例

```
# 设置变量x为整数变量
m.setVarType(x, COPT.INTEGER)
# 设置变量x和y为二进制变量
m.setVarType([x, y], COPT.BINARY)
# 设置tupledict对象xdict中的变量为连续变量
m.setVarType(xdict, COPT.CONTINUOUS)
```

Model.setMipStart()

摘要

setMipStart(vars, startvals)

描述

设置指定变量的初始值，仅对整数规划模型有效。

若参数 vars 为*Var 类*对象，则参数 startvals 为常量；若参数 vars 为字典或*tupledict 类*对象，则参数 startvals 可为常量、字典或*tupledict 类*对象；若参数 vars 为列表或*VarArray 类*对象，则参数 startvals 可为常量或列表对象。

注意：可以通过多次调用该方法来输入不同的初始解。请务必在输入结束后，调用 loadMipStart()。

参量

vars

指定的变量。

startvals

指定的变量初始值。

示例

```
# 设置变量x的初始解为1
m.setMipStart(x, 1)
# 设置变量x和y的初始解分别为2和3
m.setMipStart([x, y], [2, 3])
# 设置tupledict对象xdict中的变量的初始解均为1
m.setMipStart(xdict, 1)

# 加载初始解信息到模型
m.loadMipStart()
```

Model.loadMipStart()

摘要

loadMipStart()

描述

将当前已指定的初始值作为一组初始值设置加载到模型中。

注意：调用该方法后，将清空之前指定的初始值信息，用户可以继续指定新的初始解。

Model.setInfo()

摘要

setInfo(infoname, args, newvals)

描述

设置指定变量或线性约束的信息值。

若参数 args 为*Var 类*对象或*Constraint 类*对象，则参数 newvals 为常量；若参数 args 为字典或*tupledict 类*对象，则参数 newvals 可为常量、字典或*tupledict 类*对象；若参数 args 为列表*VarArray 类*对象或*ConstrArray 类*对象，则参数 newvals 可为常量或列表对象；若参数 args 为列表*MVar 类*对象或*MConstr 类*对象，则参数 newvals 可为常量或 numpy.ndarray 对象。

参量

infoname

指定信息名。可取值详见*信息*部分。

args

指定的变量或线性约束。

newvals

指定的新信息值。

示例

```
# 设置变量x的上界为1.0
m.setInfo(COPT.Info.UB, x, 1.0)
# 设置变量x和y的下界分别为1.0和2.0
m.setInfo(COPT.Info.LB, [x, y], [1.0, 2.0])
# 设置tupledict对象xdict中的变量的目标函数系数均为0
m.setInfo(COPT.Info.OBJ, xdict, 0.0)
```

Model.setParam()

摘要

setParam(paramname, newval)

描述

设置优化参数为指定值。

参量

paramname

指定的优化参数。可取值详见参数 部分。

newval

指定的优化参数新值。

示例

```
# 设置优化参数求解时间限制为1小时
m.setParam(COPT.Param.TimeLimit, 3600)
```

Model.resetParam()

摘要

resetParam()

描述

将模型所有优化参数重置为默认值。

示例

```
# 重置模型所有优化参数为默认值
m.resetParam()
```

Model.read()

摘要

read(filename)

描述

根据文件名后缀判断文件类型并读入到模型中。

目前支持 MPS 格式模型文件（后缀为 '.mps' 或 '.mps.gz'）、LP 格式模型文件（后缀为 '.lp' 或 '.lp.gz'）、SDPA 格式模型文件（后缀为 '.dat-s' 或 '.dat-s.gz'）、CBF 格式模型文件（后缀为 '.cbf' 或 '.cbf.gz'）、COPT 二进制格式文件（后缀为 '.bin'）、基解文件（后缀为 '.bas'）、结果文件（后缀为 '.sol'）、初始解文件（后缀为 '.mst'）和参数文件（后缀为 '.par'）。

参量

filename

待读取文件的名称。

示例

```
# 读取MPS格式模型文件
m.read('test.mps.gz')
# 读取LP格式模型文件
m.read('test.lp.gz')
# 读取COPT二进制格式模型文件
m.read('test.bin')
# 读取基解文件
m.read('testlp.bas')
# 读取结果文件
m.read('testmip.sol')
# 读取初始解文件
m.read('testmip.mst')
# 读取参数设置文件
m.read('test.par')
```

Model.readMps()

摘要

readMps(filename)

描述

按照 MPS 文件格式读取指定的文件到模型中。

参量

filename

待读取文件名。

示例

```
# 按照MPS文件格式读取文件"test.mps.gz"
m.readMps('test.mps.gz')
# 按照MPS文件格式读取文件"test.lp.gz"
m.readMps('test.lp.gz')
```

Model.readLp()

摘要

readLp(filename)

描述

按照 LP 文件格式读取指定的文件到模型中。

参量

filename

待读取文件名。

示例

```
# 按照LP文件格式读取文件"test.mps.gz"
m.readLp('test.mps.gz')
# 按照LP文件格式读取文件"test.lp.gz"
m.readLp('test.lp.gz')
```

Model.readSdpa()

摘要

readSdpa(filename)

描述

按照 SDPA 文件格式读取指定的文件到模型中。

参量

filename

待读取文件名。

示例

```
# 按照SDPA文件格式读取文件"test.dat-s"
m.readSdpa('test.dat-s')
```

Model.readCbf()

摘要

readCbf(filename)

描述

按照 CBF 文件格式读取指定的文件到模型中。

参量

filename

待读取文件名。

示例

```
# 按照CBF文件格式读取文件"test.cbf"
m.readCbf('test.cbf')
```

Model.readBin()

摘要

readBin(filename)

描述

按照 COPT 二进制文件格式读取指定的文件到模型中。

参量

filename

待读取文件名。

示例

```
# 按照COPT二进制文件格式读取文件"test.bin"
m.readBin('test.bin')
```

Model.readSol()

摘要

readSol(filename)

描述

按照结果文件格式读取文件到模型中。

注意：若读取成功，则读取的值将作为整数规划模型求解的一组初始解。文件中变量的取值可不完全指定（默认值为 0），若某变量的取值指定了多次，则采用最后一次指定的值。

参量

filename

待读取文件名。

示例

```
# 按照结果文件格式读取文件"testmip.sol"
m.readSol('testmip.sol')
# 按照结果文件格式读取文件"testmip.txt"
m.readSol('testmip.txt')
```

Model.readBasis()

摘要

readBasis(filename)

描述

按照基解文件格式读取变量和线性约束的基状态到模型中，仅适用于线性规划模型。

参量

filename

待读取文件名。

示例

```
# 按照基解文件格式读取文件"testmip.bas"
m.readBasis('testmip.bas')
# 按照基解文件格式读取文件"testmip.txt"
m.readBasis('testmip.txt')
```

Model.readMst()

摘要

readMst(filename)

描述

按照初始解文件格式读取初始解到模型中。

注意：若读取成功，则读取的值将作为整数规划模型求解的一组初始解。文件中变量的取值可不完全指定，若某变量的取值指定了多次，则采用最后一次指定的值。

参量

filename

待读取文件名。

示例

```
# 按照初始解文件格式读取文件"testmip.mst"
m.readMst('testmip.mst')
# 按照初始解文件格式读取文件"testmip.txt"
m.readMst('testmip.txt')
```

Model.readParam()

摘要

readParam(filename)

描述

按照参数文件格式读取优化参数到模型中。

注意：若某优化参数的取值指定了多次，则采用最后一次指定的值。

参量

filename

待读取文件名。

示例

```
# 按照参数文件格式读取文件"testmip.par"
m.readParam('testmip.par')
# 按照参数文件格式读取文件"testmip.txt"
m.readParam('testmip.txt')
```

Model.readTune()

摘要

readTune(filename)

描述

按照调优文件格式读取调优参数组合到模型中。

参量

filename

待读取文件名。

示例

```
# 按照调优文件格式读取文件"testmip.tune"
m.readTune('testmip.tune')
# 按照调优文件格式读取文件"testmip.txt"
m.readTune('testmip.txt')
```

Model.write()

摘要

write(filename)

描述

根据文件后缀名判断文件类型并写出到磁盘。

目前支持 MPS 格式模型文件（后缀为 '.mps'）、LP 格式模型文件（后缀为 '.lp'）、CBF 格式模型文件（后缀为 '.cbf'）、COPT 二进制格式文件（后缀为 '.bin'）、基解文件（后缀为 '.bas'）、结果文件（后缀为 '.sol'）、初始解文件（后缀为 '.mst'）和参数文件（后缀为 '.par'）。

参量

filename

待输出文件名。

示例

```
# 输出MPS格式模型文件
m.write('test.mps')
# 输出LP格式模型文件
m.write('test.lp')
# 输出COPT二进制格式模型文件
m.write('test.bin')
# 输出基解文件
m.write('testlp.bas')
# 输出结果文件
m.write('testmip.sol')
# 输出初始解文件
m.write('testmip.mst')
# 输出参数文件
m.write('test.par')
```

Model.writeMps()

摘要

writeMps(filename)

描述

将当前模型输出到 MPS 格式模型文件中。

参量

filename

待输出 MPS 格式模型文件名。

示例

```
# 输出MPS格式模型文件"test.mps"
m.writeMps('test.mps')
```

Model.writeMpsStr()

摘要

writeMpsStr()

描述

将当前模型以 MPS 格式输出到缓存对象。

示例

```
# 将当前模型以MPS格式输出到缓存对象buff中并打印模型内容
buff = m.writeMpsStr()
print(buff.getData())
```

Model.writeLp()

摘要

writeLp(filename)

描述

将当前模型输出到 LP 格式模型文件中。

参量

filename

待输出 LP 格式模型文件名。

示例

```
# 输出LP格式模型文件"test.lp"
m.writeLp('test.lp')
```

Model.writeCbf()

摘要

writeCbf(filename)

描述

将当前模型输出到 CBF 格式模型文件中。

参量

filename

待输出 CBF 格式模型文件名。

示例

```
# 输出CBF格式模型文件"test.cbf"
m.writeCbf('test.cbf')
```

Model.writeBin()

摘要

writeBin(filename)

描述

将当前模型输出到 COPT 二进制格式模型文件中。

参量

filename

待输出 COPT 二进制格式模型文件名。

示例

```
# 输出COPT二进制格式模型文件"test.bin"
m.writeBin('test.bin')
```

Model.writeIIS()

摘要

writeIIS(filename)

描述

将当前最小冲突模型写入 IIS 格式模型文件中。

参量

filename

待输出 IIS 格式模型文件名。

示例

```
# 输出IIS格式模型文件"test.iis"
m.writeIIS('test.iis')
```

Model.writeRelax()

摘要

writeRelax(filename)

描述

将可行化松弛模型输出到 Relax 格式模型文件中。

参量

filename

待输出 Relax 格式模型文件名。

示例

```
# 输出Relax格式模型文件"test.relax"
m.writeRelax('test.relax')
```

Model.writeSol()

摘要

writeSol(filename)

描述

将模型的结果输出到结果文件中。

参量

filename

待输出结果文件名。

示例

```
# 输出结果文件"test.sol"
m.writeSol('test.sol')
```

Model.writePoolSol()

摘要

writePoolSol(isol, filename)

描述

将指定的解池中的解写到文件中。

参量

isol

解池中解的索引。

filename

待输出结果文件名。

示例

```
# 输出解池中第1组解到结果文件"poolsol_1.sol"
m.writePoolSol(1, 'poolsol_1.sol')
```

Model.writeBasis()

摘要

writeBasis(filename)

描述

将线性规划模型的基解输出到基解文件中。

参量

filename

待输出基解文件名。

示例

```
# 输出基解文件"testlp.bas"
m.writeBasis('testlp.bas')
```

Model.writeMst()

摘要

writeMst(filename)

描述

对于整数规划模型，输出当前最好整数解到初始解文件中。若没有整数解，则输出模型中存储的第一组初始解。

参量

filename

待输出初始解文件名。

示例

```
# 输出初始解文件"testmip.mst"
m.writeMst('testmip.mst')
```

Model.writeParam()

摘要

writeParam(filename)

描述

输出与默认参数值不相同的参数到参数文件中。

参量

filename

待输出参数文件名。

示例

```
# 输出参数文件"testmip.par"
m.writeParam('testmip.par')
```

Model.writeTuneParam()

摘要

writeTuneParam(idx, filename)

描述

输出指定编号的参数调优结果到参数文件中。

参量

idx

参数调优结果编号。

filename

待输出参数文件名。

示例

```
# 输出指定编号的参数调优结果到参数文件"testmip.par"
m.writeTuneParam(0, 'testmip.par')
```

Model.setLogFile()

摘要

setLogFile(logfile)

描述

设置求解器日志文件。

参量

logfile

日志文件。

示例

```
# 设置日志文件为"copt.log"
m.setLogFile('copt.log')
```

Model.setLogCallback()

摘要

setLogCallback(logcb)

描述

设置求解日志回调函数。

参量

logcb

求解日志回调函数。

示例

```
# 设置日志回调函数为Python函数'logcbfun'
m.setLogCallback(logcbfun)
```

Model.solve()

摘要

solve()

描述

求解优化模型。

示例

```
# 求解优化模型
m.solve()
```

Model.solveLP()

摘要

solveLP()

描述

求解线性规划模型。若是整数规划模型，则当作线性规划模型求解。

示例

```
# 调用线性规划求解器求解模型
m.solveLP()
```

Model.computeIIS()

摘要

computeIIS()

描述

计算不可行模型的 IIS。

示例

```
# 计算不可行模型的IIS
m.computeIIS()
```

Model.feasRelax()

摘要

feasRelax(vars, lbpen, ubpen, constrs, rhspen, uppen=None)

描述

计算不可行模型的可行化松弛。

参量

vars

待松弛变量。

lbpen

变量下界的惩罚因子。若为 None，则表示不松弛下界；若惩罚因子为 COPT.INFINITY，则表示不松弛相应的变量下界。

ubpen

变量上界的惩罚因子。若为 None，则表示不松弛上界；若惩罚因子为 COPT.INFINITY，则表示不松弛相应的变量上界。

constrs

待松弛约束。

rhspen

约束边界的惩罚因子。若为 None，则表示不松弛约束边界；若惩罚因子为 COPT.INFINITY，则表示不松弛相应的约束边界。

uppen

若约束存在双边约束，则表示约束上界的惩罚因子。若为 None，则惩罚因子由 rhspen 指定；若惩罚因子为 COPT.INFINITY，则表示不松弛相应的约束上界。

示例

```
# 计算不可行模型的可行化松弛
m.feasRelax(vars, lbpen, ubpen, constrs, rhspen)
```

Model.feasRelaxS()

摘要

feasRelaxS(vrelax, crelax)

描述

计算不可行模型的可行化松弛。

参量

vrelax

是否松弛变量。

crelax

是否松弛约束。

示例

```
# 计算不可行模型的可行化松弛
m.feasRelaxS(True, True)
```

Model.tune()

摘要

tune()

描述

对模型进行参数调优。

示例

```
# 对模型进行参数调优
m.tune()
```

Model.loadTuneParam()

摘要

loadTuneParam(idx)

描述

加载指定编号的参数调优结果到模型。

示例

```
# 加载最佳参数调优结果到模型中
m.loadTuneParam(0)
```

Model.interrupt()

摘要

interrupt()

描述

中断当前求解的模型。

示例

```
# 中断当前求解的模型
m.interrupt()
```

Model.remove()

摘要

remove(args)

描述

从模型中移除变量或约束。

若移除变量，则参数 args 可取值为*Var 类* 对象、*VarArray 类* 对象、列表、字典或*tupledict 类* 对象；

若移除线性约束，则参数 args 可取值为*Constraint 类* 对象、*ConstrArray 类* 对象、列表、字典或*tupledict 类* 对象；

若移除 SOS 约束，则参数 args 可取值为*SOS 类* 对象、*SOSArray 类* 对象、列表、字典或*tupledict 类* 对象；

若移除二阶锥约束，则参数 args 可取值为*Cone 类* 对象、*ConeArray 类* 对象、列表、字典或*tupledict 类* 对象；

若移除二次约束，则参数 args 可取值为*QConstraint 类* 对象、*QConstrArray 类* 对象、列表、字典或*tupledict 类* 对象；

若移除半定约束，则参数 args 可取值为*PsdConstraint 类* 对象、*PsdConstrArray 类* 对象、列表、字典或*tupledict 类* 对象；

若移除 Indicator 约束，则参数 args 可取值为*GenConstr 类* 对象、*GenConstrArray 类* 对象、列表、字典或*tupledict 类* 对象；

若移除 LMI 约束，则参数 args 可取值为*LmiConstraint 类* 对象、*LmiConstrArray 类* 对象、列表、字典或*tupledict 类* 对象。

参量

args

待移除变量或约束。

示例

```
# 移除线性约束conx
m.remove(conx)
# 移除变量x和y
m.remove([x, y])
```

Model.resetAll()

摘要

resetAll()

描述

重置模型中的结果信息和初始解信息。

示例

```
# 重置模型中的结果信息和初始解信息
m.resetAll()
```

Model.reset()

摘要

reset()

描述

重置模型中的结果信息。

示例

```
# 重置模型中的结果信息
m.reset()
```

Model.clear()

摘要

clear()

描述

清空模型中的内容。

示例

```
# 清空模型中的内容
m.clear()
```

Model.clone()

摘要

clone()

描述

创建模型的深拷贝，返回一个*Model 类* 对象。

示例

```
# 创建模型的深拷贝
mcopy = m.clone()
```

Model.setCallback()

Synopsis

setCallback(cb, cbctx)

Description

在 COPT 模型中，设置用户自定义的回调。

Arguments

cb

用户自定义的回调类对象，继承*CallbackBase 类* 。

cbctx

回调的触发点，可取值详见*Callback context* 。

Example

```
cb = CoptCallback()
model.setCallback(cb, COPT.CBCONTEXT_MIPSOL)
```

23.2.4 Var 类

为了方便用户访问变量的相关信息，Var 类提供了形如 `Var.LB` 的访问方式。目前支持的信息详见信息部分。对于每个信息名，大小写无关。

此外，还可以通过 `Var.x` 访问变量的取值，通过 `Var.vtype` 访问变量类型，通过 `Var.name` 访问变量的名称，通过 `Var.rc` 访问线性规划中变量的 Reduced cost 值，通过 `Var.basis` 访问变量的基状态信息，以及通过 `Var.index` 访问变量在系数矩阵中的下标。

对于变量的模型相关信息，以及变量类型和名称，用户还可以通过形如 `"Var.LB = 0.0"` 的方式设置相应的信息值。

Var 类是杉数求解器变量的相关操作的封装，提供了以下成员方法：

Var.getType()

摘要

getType()

描述

获取变量的类型。

示例

```
# 获取变量v的类型
vtype = v.getType()
```

Var.getName()

摘要

getName()

描述

获取变量的名字。

示例

```
# 获取变量v的名字
varname = v.getName()
```

Var.getBasis()

摘要

getBasis()

描述

获取变量的基状态。

示例

```
# 获取变量v的基状态
varbasis = v.getBasis()
```

Var.getLowerIIS()

摘要

getLowerIIS()

描述

获取变量下边界的 IIS 状态。

示例

```
# 获取变量v下边界的IIS状态
lowerIIS = v.getLowerIIS()
```

Var.getUpperIIS()

摘要

getUpperIIS()

描述

获取变量上边界的 IIS 状态。

示例

```
# 获取变量v上边界的IIS状态
upperIIS = v.getUpperIIS()
```

Var.getIdx()

摘要

getIdx()

描述

获取变量在系数矩阵中的下标。

示例

```
# 获取变量v的下标
vindex = v.getIdx()
```

Var.setType()

摘要

setType(newtype)

描述

设置变量的类型。

参量

newtype

变量的新类型。可取值详见变量类型 部分。

示例

```
# 设置变量v的类型
v.setType(COPT.BINARY)
```

Var.setName()

摘要

setName(newname)

描述

设置变量的名称。

参量

newname

变量的新名称。

示例

```
# 设置变量v的名称
v.setName('v')
```

Var.getInfo()

摘要

getInfo(infoname)

描述

获取变量指定的信息值，返回一个常数。

参量

infoname

待获取信息名。可取值详见信息 部分。

示例

```
# 获取变量x的下界
lb = x.getInfo(COPT.Info.LB)
```

Var.setInfo()

摘要

setInfo(infoname, newval)

描述

给变量设置新的信息值。

参量

infoname

待设置信息名。可取值详见信息 部分。

newval

待设置新信息值。

示例

```
# 设置变量x的下界
x.setInfo(COPT.Info.LB, 1.0)
```

Var.remove()

摘要

remove()

描述

从模型中删除当前变量。

示例

```
# 删除变量x
x.remove()
```

23.2.5 VarArray 类

为方便用户对一组*Var 类*对象进行操作，杉数求解器的 Python 接口设计了 VarArray 类，提供了以下成员方法：

VarArray()

摘要

VarArray(vars=None)

描述

创建一个*VarArray 类*对象。

若参数 vars 为 None，则创建一个空的*VarArray 类*对象，否则以参数 vars 初始化新创建的*VarArray 类*对象。

参量

vars

待添加变量。可选参量，默认为 None。可取值为*Var 类*对象、*VarArray 类*对象、列表、字典或*tupledict 类*对象。

示例

```
# 创建一个空的VarArray类对象
vararr = VarArray()
# 创建一个VarArray类对象，并使用变量x和y初始化
vararr = VarArray([x, y])
```

VarArray.pushBack()

摘要

pushBack(var)

描述

添加单个或多个*Var 类*对象。

参量

var

待添加变量。可取值为*Var 类*对象、*VarArray 类*对象、列表、字典或*tupledict 类*对象。

示例

```
# 添加变量x到vararr中
vararr.pushBack(x)
# 添加变量x和y到vararr中
vararr.pushBack([x, y])
```

VarArray.getVar()

摘要

getVar(idx)

描述

根据变量在*VarArray 类* 对象中的下标获取相应的变量，返回一个*Var 类* 对象。

参量

idx

变量在*VarArray 类* 对象中的下标。起始为 0。

示例

```
# 获取vararr中下标为1的变量
var = vararr.getVar(1)
```

VarArray.getAll()

摘要

getAll()

描述

获取*VarArray 类* 对象中的的全部变量，返回一个列表对象。

示例

```
# 获取vararr中的全部变量
varall = vararr.getAll()
```

VarArray.getSize()

摘要

getSize()

描述

获取*VarArray 类* 对象中元素的个数。

示例

```
# 获取vararr中变量的个数
arrsize = vararr.getSize()
```

23.2.6 PsdVar 类

PsdVar 类是杉数求解器半定变量的相关操作的封装，提供了以下成员方法：

PsdVar.getName()

摘要

getName()

描述

获取半定变量的名字。

示例

```
# 获取半定变量v的名字
varname = v.getName()
```

PsdVar.getIdx()

摘要

getIdx()

描述

获取半定变量在模型中的下标。

示例

```
# 获取半定变量v的下标
vindex = v.getIdx()
```

PsdVar.getDim()

摘要

getDim()

描述

获取半定变量的维度。

示例

```
# 获取半定变量v的维度
vdim = v.getDim()
```

PsdVar.getLen()

摘要

getLen()

描述

获取半定变量展开后的长度。

示例

```
# 获取半定变量v展开后的长度
vlen = v.getLen()
```

PsdVar.setName()

摘要

setName(newname)

描述

设置半定变量的名称。

参量

newname

半定变量的新名称。

示例

```
# 设置半定变量v的名称
v.setName('v')
```

PsdVar.getInfo()

摘要

getInfo(infoname)

描述

获取半定变量指定的信息值，返回一个列表。

参量

infoname

待获取信息名。可取值详见信息 部分。

示例

```
# 获取半定变量x的取值
sol = x.getInfo(COPT.Info.Value)
```

PsdVar.remove()

摘要

remove()

描述

从模型中删除当前半定变量。

示例

```
# 删除半定变量x
x.remove()
```

PsdVar.shape

摘要

shape

描述

PsdVar 对象的形状。

返回值

整型元组。

PsdVar.size

摘要

size

描述

PsdVar 对象的形状。

返回值

整型元组。

PsdVar.dim

摘要

dim

描述

PsdVar 对象的维度。

返回值

整型值。

PsdVar.len

摘要

len

描述

PsdVar 对象展开后的长度。

返回值

整型值。

23.2.7 PsdVarArray 类

为方便用户对一组*PsdVar 类* 对象进行操作，杉数求解器的 Python 接口设计了 PsdVarArray 类，提供了以下成员方法：

PsdVarArray()

摘要

PsdVarArray(vars=None)

描述

创建一个*PsdVarArray 类* 对象。

若参数 vars 为 None，则创建一个空的*PsdVarArray 类* 对象，否则以参数 vars 初始化新创建的*PsdVarArray 类* 对象。

参量

vars

待添加半定变量。可选参量，默认为 None。可取值为*PsdVar 类* 对象、*PsdVarArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 创建一个空的PsdVarArray类对象
vararr = PsdVarArray()
# 创建一个PsdVarArray类对象，并使用半定变量x和y初始化
vararr = PsdVarArray([x, y])
```

PsdVarArray.pushBack()

摘要

pushBack(var)

描述

添加单个或多个*PsdVar 类* 对象。

参量

var

待添加半定变量。可取值为*PsdVar 类* 对象、*PsdVarArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 添加半定变量x到vararr中
vararr.pushBack(x)
# 添加半定变量x和y到vararr中
vararr.pushBack([x, y])
```

PsdVarArray.getPsdVar()

摘要

getPsdVar(idx)

描述

根据半定变量在*PsdVarArray 类* 对象中的下标获取相应的半定变量，返回一个*PsdVar 类* 对象。

参量

idx

半定变量在*PsdVarArray 类* 对象中的下标。起始为 0。

示例

```
# 获取vararr中下标为1的半定变量
var = vararr.getPsdVar(1)
```

PsdVarArray.getSize()

摘要

getSize()

描述

获取*PsdVarArray 类* 对象中元素的个数。

示例

```
# 获取vararr中半定变量的个数
arrsize = vararr.getSize()
```

23.2.8 SymMatrix 类

SymMatrix 类是半定规划模型中对称矩阵相关操作的封装，提供了以下成员方法：

SymMatrix.getIdx()

摘要

getIdx()

描述

获取对称矩阵在模型中的下标。

示例

```
# 获取对称矩阵mat的下标
matidx = mat.getIdx()
```

SymMatrix.getDim()

摘要

getDim()

描述

获取对称矩阵在模型中的维度。

示例

```
# 获取对称矩阵mat的维度
matdim = mat.getDim()
```

23.2.9 SymMatrixArray 类

为方便用户对一组*SymMatrix 类* 对象进行操作，杉数求解器的 Python 接口设计了 SymMatrixArray 类，提供了以下成员方法：

SymMatrixArray()

摘要

SymMatrixArray(mats=None)

描述

创建一个*SymMatrixArray 类* 对象。

若参数 mats 为 None，则创建一个空的*SymMatrixArray 类* 对象，否则以参数 mats 初始化新创建的*SymMatrixArray 类* 对象。

参量

mats

待添加对称矩阵。可选参量，默认为 None。可取值为*SymMatrix 类* 对象、*SymMatrixArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 创建一个空的SymMatrixArray类对象
matarr = SymMatrixArray()
# 创建一个SymMatrixArray类对象，并使用对称矩阵matx和maty初始化
matarr = SymMatrixArray([matx, maty])
```

SymMatrixArray.pushBack()

摘要

pushBack(mat)

描述

添加单个或多个*SymMatrix 类* 对象。

参量

mat

待添加对称矩阵。可取值为*SymMatrix 类* 对象、*SymMatrixArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 添加对称矩阵matx到matarr中
matarr.pushBack(matx)
```

(续下页)

(接上页)

```
# 添加对称矩阵matx和maty到matarr中
matarr.pushBack([matx, maty])
```

SymMatrixArray.getMatrix()

摘要

getMatrix(idx)

描述

根据对称矩阵在*SymMatrixArray 类* 对象中的下标获取相应的对称矩阵，返回一个*SymMatrix 类* 对象。

参量

idx

对称矩阵在*SymMatrixArray 类* 对象中的下标。起始为 0。

示例

```
# 获取matarr中下标为1的对称矩阵
mat = matarr.getMatrix(1)
```

SymMatrixArray.getSize()

摘要

getSize()

描述

获取*SymMatrixArray 类* 对象中元素的个数。

示例

```
# 获取matarr中对称矩阵的个数
arrsize = matarr.getSize()
```

23.2.10 Constraint 类

为了方便用户访问约束的相关信息，Constraint 类提供了形如 `Constraint.LB` 的访问方式。目前支持的信息详见信息 部分。对于每个信息名，大小写无关。

此外，还可以通过 `Constraint.name` 访问约束的名称，通过 `Constraint.pi` 访问线性规划中对偶变量的取值，通过 `Constraint.basis` 访问约束的基状态信息，以及通过 `Constraint.index` 访问约束在系数矩阵中的下标。

对于约束的模型相关信息和约束名称，用户还可以通过形如 `"Constraint.lb = -100` 的方式设置相应的信息值。

Constraint 类是杉数求解器线性约束的相关操作的封装，提供了以下成员方法：

Constraint.getName()

摘要

getName()

描述

获取线性约束的名称。

示例

```
# 获取线性约束con的名称
conname = con.getName()
```

Constraint.getBasis()

摘要

getBasis()

描述

获取线性约束的基状态。

示例

```
# 获取线性约束con的基状态
conbasis = con.getBasis()
```

Constraint.getLowerIIS()

摘要

getLowerIIS()

描述

获取约束下边界的 IIS 状态。

示例

```
# 获取约束con下边界的IIS状态
lowerIIS = con.getLowerIIS()
```

Constraint.getUpperIIS()

摘要

getUpperIIS()

描述

获取约束上边界的 IIS 状态。

示例

```
# 获取约束con上边界的IIS状态
upperIIS = con.getUpperIIS()
```

Constraint.getIdx()

摘要

getIdx()

描述

获取线性约束在系数矩阵中的下标。

示例

```
# 获取线性约束con的下标
conidx = con.getIdx()
```

Constraint.setName()

摘要

setName(newname)

描述

设置线性约束的名称。

参量

newname

约束的新名称。

示例

```
# 设置线性约束con的名称
con.setName('con')
```

Constraint.getInfo()

摘要

getInfo(infoname)

描述

获取指定的信息值，返回一个常数。

参量

infoname

待获取信息名。可取值详见信息 部分。

示例

```
# 获取线性约束con的下界
conlb = con.getInfo(COPT.Info.LB)
```

Constraint.setInfo()

摘要

setInfo(infoname, newval)

描述

设置新的信息值给指定线性约束。

参量

infoname

待设置信息名。可取值详见信息 部分。

newval

待设置新信息值。

示例

```
# 设置线性约束con的下界
con.setInfo(COPT.Info.LB, 1.0)
```

Constraint.remove()

摘要

remove()

描述

从模型中删除当前线性约束。

示例

```
# 删除线性约束conx
conx.remove()
```

23.2.11 ConstrArray 类

为方便用户对一组*Constraint 类* 对象进行操作，杉数求解器的 Python 接口设计了 ConstrArray 类，提供了以下成员方法：

ConstrArray()

摘要

ConstrArray(constrs=None)

描述

创建一个*ConstrArray 类* 对象。

若参数 constrs 为 None，则创建一个空的*ConstrArray 类* 对象，否则以参数 constrs 初始化新创建的*ConstrArray 类* 对象。

参量

constrs

待添加线性约束。可选参量，默认为 None。可取值为*Constraint 类* 对象、*ConstrArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 创建一个空的ConstrArray类对象
conarr = ConstrArray()
# 创建一个ConstrArray类对象，并使用线性约束conx和cony初始化
conarr = ConstrArray([conx, cony])
```

ConstrArray.pushBack()

摘要

pushBack(constr)

描述

添加单个或多个*Constraint 类* 对象。

参量

constr

待添加线性约束。可取值为*Constraint 类* 对象、*ConstrArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 添加线性约束r到conarr中
conarr.pushBack(r)
# 添加线性约束r0和r1到conarr中
conarr.pushBack([r0, r1])
```

ConstrArray.getConstr()

摘要

getConstr(idx)

描述

根据线性约束在*ConstrArray 类* 对象中的下标获取相应的线性约束，返回一个*Constraint 类* 对象。

参量

idx

线性约束在*ConstrArray 类* 对象中的下标。起始为 0。

示例

```
# 获取conarr中下标为1的线性约束
con = conarr.getConstr(1)
```

ConstrArray.getAll()

摘要

getAll()

描述

获取*ConstrArray 类* 对象中的全部线性约束，返回一个列表对象。

示例

```
# 获取conarr中的全部线性约束
cons = conarr.getAll()
```

ConstrArray.getSize()

摘要

getSize()

描述

获取*ConstrArray 类* 对象中元素的个数。

示例

```
# 获取conarr中线性约束的个数
arrsize = conarr.getSize()
```

23.2.12 ConstrBuilder 类

ConstrBuilder 类是杉数求解器中构建线性约束时的构建器的封装，提供了以下成员方法：

ConstrBuilder()

摘要

ConstrBuilder()

描述

创建一个空的*ConstrBuilder 类* 对象。

示例

```
# 创建一个空的线性约束构建器
constrbuilder = ConstrBuilder()
```

ConstrBuilder.setBuilder()

摘要

setBuilder(expr, sense)

描述

设置线性约束构建器的表达式和约束类型。

参量

expr

待设置表达式。可取值为*Var 类* 对象或*LinExpr 类* 对象。

sense

约束类型。可取值详见*约束类型* 部分。

示例

```
# 设置线性约束构建器的表达式为：x + y - 1，约束类型为等于
constrbuilder.setBuilder(x + y - 1, COPT.EQUAL)
```

ConstrBuilder.getExpr()

摘要

getExpr()

描述

获取线性约束构建器对象的表达式。

示例

```
# 获取线性约束构建器的表达式
linexpr = constrbuilder.getExpr()
```

ConstrBuilder.getSense()

摘要

getSense()

描述

获取线性约束构建器对象的约束类型。

示例

```
# 获取线性约束构建器的约束类型
consense = constrbuilder.getSense()
```

23.2.13 ConstrBuilderArray 类

为方便用户对一组*ConstrBuilder 类* 对象进行操作，杉数求解器的 Python 接口设计了 ConstrBuilderArray 类，提供了以下成员方法：

ConstrBuilderArray()

摘要

ConstrBuilderArray(constrbuilders=None)

描述

创建一个*ConstrBuilderArray 类* 对象。

若参数 constrbuilders 为 None，则创建一个空的*ConstrBuilderArray 类* 对象，否则以参数 constrbuilders 初始化新创建的*ConstrBuilderArray 类* 对象。

参量

constrbuilders

待添加线性约束构建器。可选参量，默认为 None。可取值为*ConstrBuilder 类* 对象、*ConstrBuilderArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 创建一个空的ConstrBuilderArray类对象
conbuilderarr = ConstrBuilderArray()
# 创建一个ConstrBuilderArray类对象，并使用构建器对象conbuilderx和conbuildery初始化
conbuilderarr = ConstrBuilderArray([conbuilderx, conbuildery])
```

ConstrBuilderArray.pushBack()

摘要

pushBack(constrbuilder)

描述

添加单个或多个*ConstrBuilder 类* 对象。

参量

constrbuilder

待添加线性约束构建器。可取值为*ConstrBuilder 类* 对象、*ConstrBuilderArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 添加线性约束构建器conbuilderx到conbuilderarr中
conbuilderarr.pushBack(conbuilderx)
# 添加线性约束构建器conbuilderx和conbuildery到conbuilderarr中
conbuilderarr.pushBack([conbuilderx, conbuildery])
```

ConstrBuilderArray.getBuilder()

摘要

getBuilder(idx)

描述

根据线性约束构建器在*ConstrBuilderArray 类* 对象中的下标获取相应的构建器对象。

参量

idx

线性约束构建器在*ConstrBuilderArray 类* 对象中的下标。起始为 0。

示例

```
# 获取conbuilderarr中下标为1的构建器
conbuilder = conbuilderarr.getBuilder(1)
```

ConstrBuilderArray.getSize()

摘要

getSize()

描述

获取*ConstrBuilderArray 类* 对象中元素的个数。

示例

```
# 获取conbuilderarr中构建器的个数
arrsize = conbuilderarr.getSize()
```

23.2.14 QConstraint 类

为了方便用户访问二次约束的相关信息，QConstraint 类提供了形如 `QConstraint.Slack` 的访问方式。目前支持的信息详见信息 部分。对于每个信息名，大小写无关。

此外，还可以通过 `QConstraint.name` 访问二次约束的名称，通过 `QConstraint.index` 访问二次约束在模型中的下标。

对于二次约束的模型相关信息和名称，用户还可以通过形如 `"QConstraint.rhs = -100` 的方式设置相应的信息值。

QConstraint 类是杉数求解器二次约束的相关操作的封装，提供了以下成员方法：

QConstraint.getName()

摘要

getName()

描述

获取二次约束的名称。

示例

```
# 获取二次约束qcon的名称
qconname = qcon.getName()
```

QConstraint.getRhs()

摘要

getRhs()

描述

获取二次约束的右端项。

示例

```
# 获取二次约束qcon的右端项
qconrhs = qcon.getRhs()
```

QConstraint.getSense()

摘要

getSense()

描述

获取二次约束的约束类型。

示例

```
# 获取二次约束qcon的约束类型
qconsense = qcon.getSense()
```

QConstraint.getIdx()

摘要

getIdx()

描述

获取二次约束的下标。

示例

```
# 获取二次约束qcon的下标
qconidx = qcon.getIdx()
```

QConstraint.setName()

摘要

setName(newname)

描述

设置二次约束的名称。

参量

newname

二次约束的新名称。

示例

```
# 设置二次约束qcon的名称
qcon.setName('qcon')
```

QConstraint.setRhs()

摘要

setRhs(rhs)

设置二次约束的右端项。

参量

rhs

二次约束的新右端项。

示例

```
# 设置二次约束qcon的右端项为0.0
qcon.setRhs(0.0)
```

QConstraint.setSense()

摘要

setSense(sense)

描述

设置二次约束的约束类型。

参量

sense

二次约束的新约束类型。

示例

```
# 设置二次约束qcon的约束类型为小于等于
qcon.setSense(COPT.LESS_EQUAL)
```

QConstraint.getInfo()

摘要

getInfo(infoname)

描述

获取指定的信息值，返回一个常数。

参量

infoname

待获取信息名。可取值详见信息 部分。

示例

```
# 获取二次约束qcon的当前取值
qconlb = qcon.getInfo(COPT.Info.Slack)
```

QConstraint.setInfo()

摘要

setInfo(infoname, newval)

描述

设置新的信息值给指定二次约束。

参量

infoname

待设置信息名。可取值详见信息 部分。

newval

待设置新信息值。

示例

```
# 设置二次约束qcon的下界
qcon.setInfo(COPT.Info.LB, 1.0)
```

Constraint.remove()

摘要

remove()

描述

从模型中删除当前二次约束。

示例

```
# 删除二次约束qconx
qconx.remove()
```

23.2.15 QConstrArray 类

为方便用户对一组*QConstraint 类* 对象进行操作，杉数求解器的 Python 接口设计了 QConstrArray 类，提供了以下成员方法：

QConstrArray()

摘要

QConstrArray(qconstrs=None)

描述

创建一个*QConstrArray 类* 对象。

若参数 qconstrs 为 None，则创建一个空的*QConstrArray 类* 对象，否则以参数 qconstrs 初始化新创建的*QConstrArray 类* 对象。

参量

qconstrs

待添加二次约束。可选参量，默认为 None。可取值为*QConstraint 类* 对象、*QConstrArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 创建一个空的QConstrArray类对象
qconarr = QConstrArray()
# 创建一个QConstrArray类对象，并使用二次约束qconx和qcony初始化
qconarr = QConstrArray([qconx, qcony])
```

QConstrArray.pushBack()

摘要

pushBack(constr)

描述

添加单个或多个*QConstraint 类* 对象。

参量

constr

待添加二次约束。可取值为*QConstraint 类* 对象、*QConstrArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 添加二次约束qr到qconarr中
qconarr.pushBack(qr)
# 添加二次约束qr0和qr1到qconarr中
qconarr.pushBack([qr0, qr1])
```

QConstrArray.getQConstr()

摘要

getQConstr(idx)

描述

根据二次约束在*QConstrArray 类* 对象中的下标获取相应的二次约束，返回一个*QConstraint 类* 对象。

参量

idx

二次约束在*QConstrArray 类* 对象中的下标。起始为 0。

示例

```
# 获取qconarr中下标为1的二次约束
qcon = qconarr.getQConstr(1)
```

QConstrArray.getSize()

摘要

getSize()

描述

获取*QConstrArray 类* 对象中元素的个数。

示例

```
# 获取qconarr中二次约束的个数
qarrsize = qconarr.getSize()
```

23.2.16 QConstrBuilder 类

QConstrBuilder 类是杉数求解器中构建二次约束时的构建器的封装，提供了以下成员方法：

QConstrBuilder()

摘要

QConstrBuilder()

描述

创建一个空的*QConstrBuilder 类* 对象。

示例

```
# 创建一个空的二次约束构建器
qconstrbuilder = QConstrBuilder()
```

QConstrBuilder.setBuilder()

摘要

setBuilder(expr, sense, rhs)

描述

设置二次约束构建器的表达式、约束类型和右端项。

参量

expr

待设置表达式。可取值为*Var 类* 对象、*LinExpr 类* 对象或*QuadExpr 类* 对象。

sense

二次约束类型。可取值详见*约束类型* 部分。

rhs

二次约束右端项。

示例

```
# 设置二次约束构建器的表达式为：x + y，约束类型为小于等于，右端项为1.0
qconstrbuilder.setBuilder(x + y, COPT.LESS_EQUAL, 1.0)
```

QConstrBuilder.getQuadExpr()

摘要

getQuadExpr()

描述

获取二次约束构建器对象的表达式。

示例

```
# 获取二次约束构建器的表达式
quadexpr = constrbuilder.getQuadExpr()
```

QConstrBuilder.getSense()

摘要

getSense()

描述

获取二次约束构建器对象的约束类型。

示例

```
# 获取二次约束构建器的约束类型
qconsense = qconstrbuilder.getSense()
```

23.2.17 QConstrBuilderArray 类

为方便用户对一组*QConstrBuilder 类*对象进行操作，杉数求解器的 Python 接口设计了 QConstrBuilderArray 类，提供了以下成员方法：

QConstrBuilderArray()

摘要

QConstrBuilderArray(qconstrbuilders=None)

描述

创建一个*QConstrBuilderArray 类*对象。

若参数 qconstrbuilders 为 None，则创建一个空的*QConstrBuilderArray 类*对象，否则以参数 qconstrbuilders 初始化新创建的*QConstrBuilderArray 类*对象。

参量

qconstrbuilders

待添加二次约束构建器。可选参量，默认为 None。可取值为*QConstrBuilder 类*对象、*QConstrBuilderArray 类*对象、列表、字典或*tupledict 类*对象。

示例

```
# 创建一个空的QConstrBuilderArray类对象
qconbuilderarr = QConstrBuilderArray()
# 创建一个QConstrBuilderArray类对象，并使用构建器对象qconbuilderx和qconbuildery初始化
qconbuilderarr = QConstrBuilderArray([qconbuilderx, qconbuildery])
```

QConstrBuilderArray.pushBack()

摘要

pushBack(qconstrbuilder)

描述

添加单个或多个*QConstrBuilder 类* 对象。

参量

qconstrbuilder

待添加二次约束构建器。可取值为*QConstrBuilder 类* 对象、*QConstrBuilderArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 添加二次约束构建器qconbuilderx到qconbuilderarr中
qconbuilderarr.pushBack(qconbuilderx)
# 添加二次约束构建器qconbuilderx和qconbuildery到qconbuilderarr中
qconbuilderarr.pushBack([qconbuilderx, qconbuildery])
```

QConstrBuilderArray.getBuilder()

摘要

getBuilder(idx)

描述

根据二次约束构建器在*QConstrBuilderArray 类* 对象中的下标获取相应的构建器对象。

参量

idx

二次约束构建器在*QConstrBuilderArray 类* 对象中的下标。起始为 0。

示例

```
# 获取qconbuilderarr中下标为1的构建器
qconbuilder = qconbuilderarr.getBuilder(1)
```

QConstrBuilderArray.getSize()

摘要

getSize()

描述

获取*QConstrBuilderArray 类* 对象中元素的个数。

示例

```
# 获取qconbuilderarr中构建器的个数
qarrsize = qconbuilderarr.getSize()
```

23.2.18 PsdConstraint 类

PsdConstraint 类是杉数求解器半定约束的相关操作的封装，提供了以下成员方法：

PsdConstraint.getName()

摘要

getName()

描述

获取半定约束的名称。

示例

```
# 获取半定约束con的名称
conname = con.getName()
```

PsdConstraint.getIdx()

摘要

getIdx()

描述

获取半定约束在模型中的下标。

示例

```
# 获取半定约束con的下标
conidx = con.getIdx()
```

PsdConstraint.setName()

摘要

setName(newname)

描述

设置半定约束的名称。

参量

newname

半定约束的新名称。

示例

```
# 设置半定约束con的名称
con.setName('con')
```

PsdConstraint.getInfo()

摘要

getInfo(infoname)

描述

获取指定的信息值，返回一个常数。

参量

infoname

待获取信息名。可取值详见信息 部分。

示例

```
# 获取半定约束con的下界
conlb = con.getInfo(COPT.Info.LB)
```

PsdConstraint.setInfo()

摘要

setInfo(infoname, newval)

描述

设置新的信息值给指定半定约束。

参量

infoname

待设置信息名。可取值详见信息 部分。

newval

待设置新信息值。

示例

```
# 设置半定约束con的下界
con.setInfo(COPT.Info.LB, 1.0)
```

PsdConstraint.remove()

摘要

remove()

描述

从模型中删除当前半定约束。

示例

```
# 删除半定约束conx
conx.remove()
```

23.2.19 PsdConstrArray 类

为方便用户对一组*PsdConstraint 类*对象进行操作，杉数求解器的 Python 接口设计了 PsdConstrArray 类，提供了以下成员方法：

PsdConstrArray()

摘要

PsdConstrArray(constrs=None)

描述

创建一个*PsdConstrArray 类*对象。

若参数 constrs 为 None，则创建一个空的*PsdConstrArray 类*对象，否则以参数 constrs 初始化新创建的*PsdConstrArray 类*对象。

参量

constrs

待添加半定约束。可选参量，默认为 None。可取值为*PsdConstraint 类*对象、*PsdConstrArray 类*对象、列表、字典或*tupledict 类*对象。

示例

```
# 创建一个空的PsdConstrArray类对象
conarr = PsdConstrArray()
# 创建一个PsdConstrArray类对象，并使用半定约束conx和cony初始化
conarr = PsdConstrArray([conx, cony])
```

PsdConstrArray.pushBack()

摘要

pushBack(constr)

描述

添加单个或多个*PsdConstraint 类* 对象。

参量

constr

待添加半定约束。可取值为*PsdConstraint 类* 对象、*PsdConstrArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 添加半定约束r到conarr中
conarr.pushBack(r)
# 添加半定约束r0和r1到conarr中
conarr.pushBack([r0, r1])
```

PsdConstrArray.getPsdConstr()

摘要

getPsdConstr(idx)

描述

根据半定约束在*PsdConstrArray 类* 对象中的下标获取相应的半定约束，返回一个*PsdConstraint 类* 对象。

参量

idx

半定约束在*PsdConstrArray 类* 对象中的下标。起始为 0。

示例

```
# 获取conarr中下标为1的半定约束
con = conarr.getPsdConstr(1)
```

PsdConstrArray.getSize()

摘要

getSize()

描述

获取*PsdConstrArray 类* 对象中元素的个数。

示例

```
# 获取conarr中半定约束的个数
arrsize = conarr.getSize()
```

23.2.20 PsdConstrBuilder 类

PsdConstrBuilder 类是杉数求解器中构建线性约束时的构建器的封装，提供了以下成员方法：

PsdConstrBuilder()

摘要

PsdConstrBuilder()

描述

创建一个空的*PsdConstrBuilder 类* 对象。

示例

```
# 创建一个空的半定约束构建器
constrbuilder = PsdConstrBuilder()
```

PsdConstrBuilder.setBuilder()

摘要

setBuilder(expr, sense, rhs)

描述

设置半定约束构建器的表达式、约束类型和右端项。

参量

expr

待设置表达式。可取值为*PsdVar 类* 对象或*PsdExpr 类* 对象。

sense

约束类型。可取值详见*约束类型* 部分。

rhs

约束右端项。

示例

```
# 设置半定约束构建器的表达式为：x + y == 1，约束类型为等于
constrbuilder.setBuilder(x + y, COPT.EQUAL, 1)
```

PsdConstrBuilder.setRange()

摘要

setRange(expr, range)

描述

设置半定约束构建器的表达式和右端项，形式为 expr 小于等于 0，且大于等于 -range.

参量

expr

待设置表达式。可取值为*PsdVar 类* 对象或*PsdExpr 类* 对象。

range

Range 约束的右端项，非负。

示例

```
# 设置半定约束构建器的表达式为：x + y - 1，range右端项为1
constrbuilder.setRange(x + y - 1, 1)
```

PsdConstrBuilder.getPsdExpr()

摘要

getPsdExpr()

描述

获取半定约束构建器对象的表达式。

示例

```
# 获取半定约束构建器的表达式
psdexpr = constrbuilder.getPsdExpr()
```

PsdConstrBuilder.getSense()

摘要

getSense()

描述

获取半定约束构建器对象的约束类型。

示例

```
# 获取线性约束构建器的约束类型
consense = constrbuilder.getSense()
```

PsdConstrBuilder.getRange()

摘要

getRange()

描述

获取半定约束构建器对象的 range 右端项，即约束上下界之差。

示例

```
# 获取线性约束构建器的range右端项
rngval = constrbuilder.getRange()
```

23.2.21 PsdConstrBuilderArray 类

为方便用户对一组*PsdConstrBuilder 类* 对象进行操作，杉数求解器的 Python 接口设计了 PsdConstrBuilderArray 类，提供了以下成员方法：

PsdConstrBuilderArray()

摘要

PsdConstrBuilderArray(builders=None)

描述

创建一个*PsdConstrBuilderArray 类* 对象。

若参数 builders 为 None，则创建一个空的*PsdConstrBuilderArray 类* 对象，否则以参数 builders 初始化新创建的*PsdConstrBuilderArray 类* 对象。

参量

builders

待添加半定约束构建器。可选参量，默认为 None。可取值为*PsdConstrBuilder 类* 对象、*PsdConstrBuilderArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 创建一个空的PsdConstrBuilderArray类对象
conbuilderarr = PsdConstrBuilderArray()
# 创建一个PsdConstrBuilderArray类对象，并使用构建器对象conbuilderx和conbuildery初始化
conbuilderarr = PsdConstrBuilderArray([conbuilderx, conbuildery])
```

PsdConstrBuilderArray.pushBack()

摘要

pushBack(builder)

描述

添加单个或多个*PsdConstrBuilder 类* 对象。

参量

builder

待添加半定约束构建器。可取值为*PsdConstrBuilder 类* 对象、*PsdConstrBuilderArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 添加半定约束构建器conbuilderx到conbuilderarr中
conbuilderarr.pushBack(conbuilderx)
# 添加半定约束构建器conbuilderx和conbuildery到conbuilderarr中
conbuilderarr.pushBack([conbuilderx, conbuildery])
```

PsdConstrBuilderArray.getBuilder()

摘要

getBuilder(idx)

描述

根据半定约束构建器在*PsdConstrBuilderArray 类* 对象中的下标获取相应的构建器对象。

参量

idx

半定约束构建器在*PsdConstrBuilderArray 类* 对象中的下标。起始为 0。

示例

```
# 获取conbuilderarr中下标为1的构建器
conbuilder = conbuilderarr.getBuilder(1)
```

PsdConstrBuilderArray.getSize()

摘要

getSize()

描述

获取*PsdConstrBuilderArray 类* 对象中元素的个数。

示例

```
# 获取conbuilderarr中构建器的个数
arrsize = conbuilderarr.getSize()
```

23.2.22 LmiConstraint 类

LmiConstraint 类是杉数求解器 LMI 约束的相关操作的封装，提供了以下成员方法：

LmiConstraint.getName()

摘要

getName()

描述

获取 LMI 约束的名称。

示例

```
# 获取LMI约束con的名称
conname = con.getName()
```

LmiConstraint.getIdx()

摘要

getIdx()

描述

获取 LMI 约束在模型中的下标。

示例

```
# 获取LMI约束con的下标
conidx = con.getIdx()
```

LmiConstraint.getDim()

摘要

getDim()

描述

获取 LMI 约束的维度。

示例

```
# 获取LMI约束con的维度
conidx = con.getDim()
```

LmiConstraint.getLen()

摘要

getLen()

描述

获取 LMI 约束展开后的长度。

示例

```
# 获取LMI约束展开后的长度
conidx = con.getLen()
```

LmiConstraint.setName()

摘要

setName(newname)

描述

设置 LMI 约束的名称。

参量

newname

LMI 约束的新名称。

示例

```
# 设置LMI约束con的名称
con.setName('con')
```

LmiConstraint.setRhs()

摘要

setRhs(mat)

描述

设置 LMI 约束的常数项对称矩阵。

参量

mat

LMI 约束的常数项对称矩阵，需要为*SymMatrix 类* 类对象。

示例

```
# 设置LMI约束con的常数项对称矩阵
D = m.addSparseMat(2, [0, 1], [0, 1], [1.0, 1.0])
con.setRhs(D)
```

LmiConstraint.getInfo()

摘要

getInfo(infoname)

描述

获取 LMI 约束指定信息 infoname 的取值。

参量

infoname

待获取信息名。可取值为 "Slack" 或 "Dual" 。

示例

```
# 获取LMI约束con松弛变量的取值
con_slack = con.getInfo("Slack")
# 获取LMI约束con的对偶变量取值
con_dual = con.getInfo("Dual")
```

LmiConstraint.remove()

摘要

remove()

描述

从模型中删除当前 LMI 约束。

示例

```
# 删除LMI约束conx
conx.remove()
```

LmiConstraint.shape

摘要

shape

描述

LmiConstraint 对象的形状。

返回值

整型元组。

LmiConstraint.size

摘要

size

描述

LmiConstraint 对象的形状。

返回值

整型元组。

LmiConstraint.dim

摘要

dim

描述

LmiConstraint 对象的维度。

返回值

整型值。

LmiConstraint.len

摘要

len

描述

LmiConstraint 对象展开后的长度。

返回值

整型值。

23.2.23 LmiConstrArray 类

为方便用户对一组*LmiConstraint 类*对象进行操作，杉数求解器的 Python 接口设计了 LmiConstrArray 类，提供了以下成员方法：

LmiConstrArray()

摘要

LmiConstrArray(constrs=None)

描述

创建一个*LmiConstrArray 类*对象。

若参数 constrs 为 None，则创建一个空的*LmiConstrArray 类*对象，否则以参数 constrs 初始化新创建的*LmiConstrArray 类*对象。

参量

constrs

待添加的 LMI 约束。可选参量，默认为 None。可取值为*LmiConstraint 类*对象、*LmiConstrArray 类*对象、列表、字典或*tupledict 类*对象。

示例

```
# 创建一个空的LmiConstrArray类对象
conarr = LmiConstrArray()
# 创建一个LmiConstrArray类对象，并使用LMI约束conx和cony初始化
conarr = LmiConstrArray([conx, cony])
```

LmiConstrArray.pushBack()

摘要

pushBack(constr)

描述

向 LmiConstrArray 中添加单个或多个*LmiConstraint 类* 对象。

参量

constr

待添加 LMI 约束。可取值为*LmiConstraint 类* 对象、*LmiConstrArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 添加LMI约束r到conarr中
conarr.pushBack(r)
# 添加LMI约束r0和r1到conarr中
conarr.pushBack([r0, r1])
```

LmiConstrArray.getLmiConstr()

摘要

getLmiConstr(idx)

描述

根据 LMI 约束在*LmiConstrArray 类* 对象中的下标获取相应的 LMI 约束，返回一个*LmiConstraint 类* 对象。

参量

idx

LMI 约束在*LmiConstrArray 类* 对象中的下标。起始为 0。

示例

```
# 获取conarr中下标为1的LMI约束
con = conarr.getLmiConstr(1)
```

LmiConstrArray.getSize()

摘要

getSize()

描述

获取*LmiConstrArray 类* 对象中元素的个数。

示例

```
# 获取conarr中Lmi约束的个数
arrsize = conarr.getSize()
```

LmiConstrArray.reserve()

摘要

reserve(n)

描述

为大小为 n 的*LmiConstrArray 类* 对象保留空间。

参量

n

LmiConstrArray 类 对象中元素的个数。

23.2.24 SOS 类

SOS 类是杉数求解器的 SOS 约束的相关操作的封装，目前提供了以下成员方法：

关于 SOS 约束的介绍请参考特殊约束：*SOS 约束章节* 。

SOS.getIdx()

摘要

getIdx()

描述

获取 SOS 约束在模型中的下标。

示例

```
# 获取SOS约束sosx的下标
sosidx = sosx.getIdx()
```

SOS.remove()

摘要

remove()

描述

从模型中删除当前 SOS 约束。

示例

```
# 删除SOS约束sosx
sosx.remove()
```

23.2.25 SOSArray 类

为方便用户对一组*SOS 类* 对象进行操作，杉数求解器的 Python 接口设计了 SOSArray 类，提供了以下成员方法：

SOSArray()

摘要

SOSArray(soss=None)

描述

创建一个*SOSArray 类* 对象。

若参数 soss 为 None，则创建一个空的*SOSArray 类* 对象，否则以参数 soss 初始化新创建的*SOSArray 类* 对象。

参量

soss

待添加 SOS 约束。可选参量，默认为 None。可取值为*SOS 类* 对象、*SOSArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 创建一个新的SOSArray对象
sosarr = SOSArray()
# 创建一个SOSArray对象，并使用SOS约束sosx和sosy初始化
sosarr = SOSArray([sosx, sosy])
```

SOSArray.pushBack()

摘要

pushBack(sos)

描述

添加单个或多个*SOS 类* 对象。

参量

sos

待添加 SOS 约束。可取值为*SOS 类* 对象、*SOSArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 添加SOS约束sosx到sosarr中
sosarr.pushBack(sosx)
# 添加SOS约束sosx和sosy到sosarr中
sosarr.pushBack([sosx, sosy])
```

SOSArray.getSOS()

摘要

getSOS(idx)

描述

根据 SOS 约束在*SOSArray 类* 对象中的下标获取相应的 SOS 约束，返回一个*SOS 类* 对象。

参量

idx

SOS 约束在*SOSArray 类* 对象中的下标。起始为 0。

示例

```
# 获取sosarr中下标为1的SOS约束
sos = sosarr.getSOS(1)
```

SOSArray.getSize()

摘要

getSize()

描述

获取*SOSArray 类* 对象中元素的个数。

示例

```
# 获取sosarr中SOS约束的个数
arrsize = sosarr.getSize()
```

23.2.26 SOSBuilder 类

SOSBuilder 类是杉数求解器中构建 SOS 约束的构建器的封装，提供了以下成员方法：

关于 SOS 约束的介绍请参考特殊约束：*SOS 约束章节* 。

SOSBuilder()

摘要

SOSBuilder()

描述

创建一个空的*SOSBuilder 类* 对象。

示例

```
# 创建一个空的SOSBuilder对象
sosbuilder = SOSBuilder()
```

SOSBuilder.setBuilder()

摘要

setBuilder(sostype, vars, weights=None)

描述

设置*SOSBuilder 类* 对象的类型、变量和变量权重。

参量

sostype

SOS 约束类型，可取值详见*SOS 约束类型* 。

vars

SOS 约束的变量，可取值为*VarArray 类* 对象、列表、字典或*tupledict 类* 对象。

`weights`

SOS 约束的变量的权重。可选取值，默认为 None。可取值为列表、字典或*tupledict 类* 对象。

示例

```
# 设置SOS约束构建器的类型为SOS1，变量为x和y，变量的权重分别为1和2
sosbuilder.setBuilder(COPT.SOS_TYPE1, [x, y], [1, 2])
```

SOSBuilder.getType()

摘要

getType()

描述

获取*SOSBuilder 类* 对象的 SOS 约束类型。

示例

```
# 获取SOS约束构建器sosx的类型
sostype = sosbuilder.getType(sosx)
```

SOSBuilder.getVar()

摘要

getVar(idx)

描述

根据变量在*SOSBuilder 类* 对象的下标获取相应的变量，返回一个*Var 类* 对象。

参量

`idx`

变量在*SOSBuilder 类* 对象中的下标。起始为 0。

示例

```
# 获取SOS约束构建器sosx中下标为1的变量
sosvar = sosx.getVar(1)
```

SOSBuilder.getVars()

摘要

getVars()

描述

获取*SOSBuilder 类* 对象的所有变量，返回一个*VarArray 类* 对象。

示例

```
# 获取SOS约束构建器sosx中的所有变量
sosvars = sosx.getVars()
```

SOSBuilder.getWeight()

摘要

getWeight(idx)

描述

根据变量在*SOSBuilder 类* 对象的下标获取相应的变量权重。

参量

idx

变量在*SOSBuilder 类* 对象中的下标。起始为 0。

示例

```
# 获取SOS约束构建器sosx中下标为1的变量相应的权重
sosweight = sosx.getWeight(1)
```

SOSBuilder.getWeights()

摘要

getWeights()

描述

获取*SOSBuilder 类* 对象中所有变量的权重。

示例

```
# 获取SOS约束构建器sosx中所有变量的权重
sosweights = sosx.getWeights()
```

SOSBuilder.getSize()

摘要

getSize()

描述

获取*SOSBuilder 类* 对象中元素的个数。

示例

```
# 获取SOS约束构建器sosx中元素的个数
sossize = sosx.getSize()
```

23.2.27 SOSBuilderArray 类

为方便用户对一组*SOSBuilder 类* 对象进行操作，杉数求解器的 Python 接口设计了 SOSBuilderArray 类，提供了以下成员方法：

SOSBuilderArray()

摘要

SOSBuilderArray(sosbuilders=None)

描述

创建一个*SOSBuilderArray 类* 对象。

若参数 sosbuilders 为 None，则创建一个空的*SOSBuilderArray 类* 对象，否则以参数 sosbuilders 初始化新创建的*SOSBuilderArray 类* 对象。

参量

sosbuilders

待添加 SOS 约束构建器。可选参量，默认为 None。可取值为*SOSBuilder 类* 对象、*SOSBuilderArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 创建一个空的SOSBuilderArray对象
sosbuilderarr = SOSBuilderArray()
# 创建一个SOSBuilderArray对象，并使用SOS约束构建器sosx和sosy初始化
sosbuilderarr = SOSBuilderArray([sosx, sosy])
```

SOSBuilderArray.pushBack()

摘要

pushBack(sosbuilder)

描述

添加单个或多个*SOSBuilder 类* 对象。

参量

sosbuilder

待添加 SOS 约束构建器。可取值为*SOSBuilder 类* 对象、*SOSBuilderArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 添加SOS约束构建器sosx到sosbuilderarr中
sosbuilderarr.pushBack(sosx)
```

SOSBuilderArray.getBuilder()

摘要

getBuilder(idx)

描述

根据 SOS 约束构建器在*SOSBuilderArray 类* 对象中的下标获取相应的构建器。

参量

idx

SOS 约束构建器在*SOSBuilderArray 类* 对象中的下标。起始为 0。

示例

```
# 获取sosbuilderarr中下标为1的SOS约束构建器
sosbuilder = sosbuilderarr.getBuilder(1)
```

SOSBuilderArray.getSize()

摘要

getSize()

描述

获取*SOSBuilderArray 类* 对象中元素的个数。

示例

```
# 获取sosbuilderarr中元素的个数
sosbuildersize = sosbuilderarr.getSize()
```

23.2.28 Cone 类

Cone 类是杉数求解器的二阶锥约束的相关操作的封装，目前提供了以下成员方法：

Cone.getIdx()

摘要

getIdx()

描述

获取二阶锥约束在模型中的下标。

示例

```
# 获取二阶锥约束cone的下标
coneidx = cone.getIdx()
```

Cone.remove()

摘要

remove()

描述

从模型中删除当前二阶锥约束。

示例

```
# 删除二阶锥约束cone
cone.remove()
```

23.2.29 ConeArray 类

为方便用户对一组*Cone 类* 对象进行操作，杉数求解器的 Python 接口设计了 ConeArray 类，提供了以下成员方法：

ConeArray()

摘要

ConeArray(cones=None)

描述

创建一个*ConeArray 类* 对象。

若参数 cones 为 None，则创建一个空的*ConeArray 类* 对象，否则以参数 cones 初始化新创建的*ConeArray 类* 对象。

参量

cones

待添加二阶锥约束。可选参量，默认为 None。可取值为*Cone 类* 对象、*ConeArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 创建一个新的ConeArray对象
conearr = ConeArray()
# 创建一个ConeArray对象，并使用二阶锥约束conex和coney初始化
conearr = ConeArray([conex, coney])
```

ConeArray.pushBack()

摘要

pushBack(cone)

描述

添加单个或多个*Cone 类* 对象。

参量

cone

待添加二阶锥约束。可取值为*Cone 类* 对象、*ConeArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 添加二阶锥约束conex到conearr中
conearr.pushBack(conex)
# 添加二阶锥约束conex和coney到conearr中
conearr.pushBack([conex, coney])
```

ConeArray.getCone()

摘要

getCone(idx)

描述

根据二阶锥约束在*ConeArray 类* 对象中的下标获取相应的二阶锥约束，返回一个*Cone 类* 对象。

参量

idx

二阶锥约束在*ConeArray 类* 对象中的下标。起始为 0。

示例

```
# 获取conearr中下标为1的二阶锥约束
cone = conearr.getCone(1)
```

ConeArray.getSize()

摘要

getSize()

描述

获取*ConeArray 类* 对象中元素的个数。

示例

```
# 获取conearr中二阶锥约束的个数
arrsize = conearr.getSize()
```

23.2.30 ConeBuilder 类

ConeBuilder 类是杉数求解器中构建二阶锥约束的构建器的封装，提供了以下成员方法：

ConeBuilder()

摘要

ConeBuilder()

描述

创建一个空的*ConeBuilder 类* 对象。

示例

```
# 创建一个空的ConeBuilder对象
conebuilder = ConeBuilder()
```

ConeBuilder.setBuilder()

摘要

setBuilder(conetype, vars)

描述

设置*ConeBuilder 类* 对象的类型、变量。

参量

conetype

二阶锥约束类型，可取值详见二阶锥约束类型 。

vars

二阶锥约束的变量，可取值为*VarArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 设置二阶锥约束构建器的类型为标准二阶锥
conebuilder.setBuilder(COPT.CONE_QUAD, [z, x, y])
```

ConeBuilder.getType()

摘要

getType()

描述

获取*ConeBuilder 类* 对象的二阶锥约束类型。

示例

```
# 获取二阶锥约束构建器conex的类型
conetype = conebuilder.getType(conex)
```

ConeBuilder.getVar()

摘要

getVar(idx)

描述

根据变量在*ConeBuilder 类* 对象的下标获取相应的变量，返回一个*Var 类* 对象。

参量

idx

变量在*ConeBuilder 类* 对象中的下标。起始为 0。

示例

```
# 获取二阶锥约束构建器conex中下标为1的变量
conevar = conex.getVar(1)
```

ConeBuilder.getVars()

摘要

getVars()

描述

获取*ConeBuilder 类* 对象的所有变量，返回一个*VarArray 类* 对象。

示例

```
# 获取二阶锥约束构建器conex中的所有变量
conevars = conex.getVars()
```

ConeBuilder.getSize()

摘要

getSize()

描述

获取*ConeBuilder 类* 对象中元素的个数。

示例

```
# 获取二阶锥约束构建器conex中元素的个数
conesize = conex.getSize()
```

23.2.31 ConeBuilderArray 类

为方便用户对一组*ConeBuilder 类*对象进行操作，杉数求解器的 Python 接口设计了 ConeBuilderArray 类，提供了以下成员方法：

ConeBuilderArray()

摘要

ConeBuilderArray(conebuilders=None)

描述

创建一个*ConeBuilderArray 类*对象。

若参数 conebuilders 为 None，则创建一个空的*ConeBuilderArray 类*对象，否则以参数 conebuilders 初始化新创建的*ConeBuilderArray 类*对象。

参量

conebuilders

待添加二阶锥约束构建器。可选参量，默认为 None。可取值为*ConeBuilder 类*对象、*ConeBuilderArray 类*对象、列表、字典或*tupledict 类*对象。

示例

```
# 创建一个空的ConeBuilderArray对象
conebuilderarr = ConeBuilderArray()
# 创建一个ConeBuilderArray对象，并使用二阶锥约束构建器conex和coney初始化
conebuilderarr = ConeBuilderArray([conex, coney])
```

ConeBuilderArray.pushBack()

摘要

pushBack(conebuilder)

描述

添加单个或多个*ConeBuilder 类*对象。

参量

conebuilder

待添加二阶锥约束构建器。可取值为*ConeBuilder 类*对象、*ConeBuilderArray 类*对象、列表、字典或*tupledict 类*对象。

示例

```
# 添加二阶锥约束构建器conex到conebuilderarr中
conebuilderarr.pushBack(conex)
```

ConeBuilderArray.getBuilder()

摘要

getBuilder(idx)

描述

根据二阶锥约束构建器在*ConeBuilderArray 类* 对象中的下标获取相应的构建器。

参量

idx

二阶锥约束构建器在*ConeBuilderArray 类* 对象中的下标。起始为 0。

示例

```
# 获取conebuilderarr中下标为1的二阶锥约束构建器
conebuilder = conebuilderarr.getBuilder(1)
```

ConeBuilderArray.getSize()

摘要

getSize()

描述

获取*ConeBuilderArray 类* 对象中元素的个数。

示例

```
# 获取conebuilderarr中元素的个数
conebuildersize = conebuilderarr.getSize()
```

23.2.32 GenConstr 类

GenConstr 类是杉数求解器的 Indicator 约束的相关操作的封装，提供了以下成员方法：

关于 Indicator 约束的介绍请参考*特殊约束：Indicator 约束章节* 。

GenConstr.getIdx()

摘要

getIdx()

描述

获取 Indicator 约束在模型中的下标。

示例

```
# 获取Indicator约束indicx的下标
indidx = indicx.getIdx()
```

GenConstr.remove()

摘要

remove()

描述

从模型中删除当前 Indicator 约束。

示例

```
# 删除Indicator约束indx
indx.remove()
```

23.2.33 GenConstrArray 类

为方便用户对一组*GenConstr 类* 对象进行操作，杉数求解器的 Python 接口设计了 GenConstrArray 类，提供了以下成员方法：

GenConstrArray()

摘要

GenConstrArray(genconstrs=None)

描述

创建一个*GenConstrArray 类* 对象。

若参数 genconstrs 为 None，则创建一个空的*GenConstrArray 类* 对象，否则以参数 genconstrs 初始化新创建的*GenConstrArray 类* 对象。

参量

genconstrs

待添加 Indicator 约束。可选参量，默认为 None。可取值为*GenConstr 类* 对象、*GenConstrArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 创建一个新的GenConstrArray对象
genconstrarr = GenConstrArray()
# 创建一个GenConstrArray对象，并使用Indicator约束genx和geny初始化
genconstrarr = GenConstrArray([genx, geny])
```

GenConstrArray.pushBack()

摘要

pushBack(genconstr)

描述

添加单个或多个*GenConstr 类* 对象。

参量

constrs

待添加 Indicator 约束。可取值为*GenConstr 类* 对象、*GenConstrArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 添加Indicator约束genx到genconarr中
genconarr.pushBack(genx)
# 添加Indicator约束genx和geny到genconarr中
genconarr.pushBack([genx, geny])
```

GenConstrArray.getGenConstr()

摘要

getGenConstr(idx)

描述

根据 Indicator 约束在*GenConstrArray 类* 对象中的下标获取相应的 Indicator 约束，返回一个*GenConstr 类* 对象。

参量

idx

Indicator 约束在*GenConstrArray 类* 对象中的下标。起始为 0。

示例

```
# 获取genconarr中下标为1的Indicator约束
genconstr = genconarr.getGenConstr(1)
```

GenConstrArray.getSize()

摘要

getSize()

描述

获取*GenConstrArray 类* 对象中的元素个数。

示例

```
# 获取genconarr中元素的个数
genconsize = genconarr.getSize()
```

23.2.34 GenConstrBuilder 类

GenConstrBuilder 类是杉数求解器中构建 Indicator 约束时的构建器的封装，提供了以下成员方法：

关于 Indicator 约束的介绍请参考*特殊约束：Indicator 约束章节* 。

GenConstrBuilder()

摘要

GenConstrBuilder()

描述

创建一个空的*GenConstrBuilder 类* 对象。

示例

```
# 创建一个空的GenConstrBuilder对象
genconbuilder = GenConstrBuilder()
```

GenConstrBuilder.setBuilder()

摘要

setBuilder(var, val, expr, sense)

描述

设置*GenConstrBuilder 类* 对象的 Indicator 变量、Indicator 变量的取值、线性约束表达式和线性约束的类型。

参量

var

Indicator 变量。

val

Indicator 变量的取值。

expr

线性约束的表达式。可取值为*Var 类* 对象或*LinExpr 类* 对象。

sense

线性约束的类型。可取值详见*约束类型* 。

示例

```
# 设置Indicator约束构建器的Indicator变量为x，当x为真时，线性约束x + y == 1成立
genconbuilder.setBuilder(x, True, x + y - 1, COPT.EQUAL)
```

GenConstrBuilder.getBinVar()

摘要

getBinVar()

描述

获取*GenConstrBuilder 类* 对象的 Indicator 变量。

示例

```
# 获取Indicator约束构建器genbuilderx的Indicator变量
indvar = genbuilderx.getBinVar()
```

GenConstrBuilder.getBinVal()

摘要

getBinVal()

描述

获取*GenConstrBuilder 类* 对象的 Indicator 变量的取值。

示例

```
# 获取Indicator约束构建器genbuilderx的Indicator变量有效时的取值
indval = genbuilderx.getBinVal()
```

GenConstrBuilder.getExpr()

摘要

getExpr()

描述

获取*GenConstrBuilder 类* 对象的线性约束表达式。

示例

```
# 获取Indicator约束构建器genbuilderx的线性约束表达式
linexpr = genbuilderx.getExpr()
```

GenConstrBuilder.getSense()

摘要

getSense()

描述

获取*GenConstrBuilder 类* 对象的线性约束类型。

示例

```
# 获取Indicator约束构建器genbuilderx的线性约束类型。
linsense = genbuilderx.getSense()
```

23.2.35 GenConstrBuilderArray 类

为方便用户对一组*GenConstrBuilder 类* 对象进行操作，杉数求解器的 Python 接口设计了 GenConstrBuilderArray 类，提供了以下成员方法：

GenConstrBuilderArray()

摘要

GenConstrBuilderArray(genconstrbuilders=None)

描述

创建一个*GenConstrBuilderArray 类* 对象。

若参数 genconstrbuilders 为 None，则创建一个空的*GenConstrBuilderArray 类* 对象，否则以参数 genconstrbuilders 初始化新创建的*GenConstrBuilderArray 类* 对象。

参量

genconstrbuilders

待添加 Indicator 约束构建器。可选参量，默认为 None。可取值为*GenConstrBuilder 类* 对象、*GenConstrBuilderArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 创建一个空的GenConstrBuilderArray对象
genbuilderarr = GenConstrBuilderArray()
# 创建一个GenConstrBuilderArray对象，并使用Indicator约束构建器genbuilderx和genbuildery初始化
genbuilderarr = GenConstrBuilderArray([genbuilderx, genbuildery])
```

GenConstrBuilderArray.pushBack()

摘要

pushBack(genconstrbuilder)

描述

添加单个或多个*GenConstrBuilder 类* 对象。

参量

genconstrbuilder

待添加 Indicator 约束构建器。可取值为*GenConstrBuilder 类* 对象、*GenConstrBuilderArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 添加Indicator约束构建器到genbuilderarr中
genbuilderarr.pushBack(genbuilderx)
# 添加Indicator约束构建器genbuilderx和genbuildery到genbuilderarr中
genbuilderarr.pushBack([genbuilderx, genbuildery])
```

GenConstrBuilderArray.getBuilder()

摘要

getBuilder(idx)

描述

根据 Indicator 约束构建器在*GenConstrBuilderArray 类* 对象中的下标获取相应的构建器，返回一个*GenConstrBuilder 类* 对象。

参量

idx

Indicator 约束构建器在*GenConstrBuilderArray 类* 对象中的下标。起始为 0。

示例

```
# 获取genbuilderarr中下标为1的Indicator约束构建器
genbuilder = genbuilderarr.getBuilder(1)
```

GenConstrBuilderArray.getSize()

摘要

getSize()

描述

获取*GenConstrBuilderArray 类* 对象中元素的个数。

示例

```
# 获取genbuilderarr中元素的个数
genbuildersize = genbuilderarr.getSize()
```

23.2.36 Column 类

为了方便用户采用按列建模的方式，杉数求解器的 Python 接口设计了 Column 类，提供了以下成员方法：

Column()

摘要

Column(constrs=0.0, coeffs=None)

描述

创建一个*Column 类* 对象。

若参数 constrs 为 None，参数 coeffs 为 None，则创建一个空的*Column 类* 对象，否则采用参数 constrs 和 coeffs 初始化新创建的*Column 类* 对象。若参数 constrs 为*Constraint 类* 对象或*Column 类* 对象，则参数 coeffs 为常数，当参数 coeffs 为 None，则当作常数 1.0；若参数 constrs 为列表对象，参数 coeffs 为 None，则参数 constrs 中的元素为约束、系数对；对于其它参数情形，则调用成员方法 addTerms 初始化新创建的*Column 类* 对象。

参量

constrs

线性约束。

coeffs

变量在线性约束中的系数。

示例

```
# 创建一个空的Column对象
col = Column()
# 创建一个Column对象，并添加两个项：约束conx中变量系数为2，约束cony中变量系数为3
col = Column([(conx, 2), (cony, 3)])
# 创建一个Column对象，并添加两个项：约束conxx中变量系数为1，约束conyy中变量系数为2
col = Column([conxx, conyy], [1, 2])
```

Column.getCoeff()

摘要

getCoeff(idx)

描述

根据元素在*Column 类* 对象中的下标获取相应系数。

参量

idx

元素的下标。起始为 0。

示例

```
# 获取col中下标为0的项对应的系数
coeff = col.getCoeff(0)
```

Column.getConstr()

摘要

getConstr(idx)

描述

根据元素在*Column 类* 对象中的下标获取相应线性约束。

参量

idx

元素的下标。起始为 0。

示例

```
# 获取col中下标为1的项对应的线性约束
constr = col.getConstr(1)
```

Column.getSize()

摘要

getSize()

描述

获取*Column 类* 对象中元素的个数。

示例

```
# 获取col中元素的个数
colsize = col.getSize()
```

Column.addTerm()

摘要

addTerm(constr, coeff=1.0)

描述

添加一个新的项。

参量

constr

待添加项对应的线性约束。

coeff

待添加项的系数。可选参量，默认值为 1.0。

示例

```
# 添加一个项到col，其约束为cony，系数为2.0
col.addTerm(cony, 2.0)
# 添加一个项到col，其约束为conx，系数为1.0
col.addTerm(conx)
```

Column.addTerms()

摘要

addTerms(constrs, coeffs)

描述

添加单个或多个新项。

若参数 constrs 为*Constraint 类* 对象，则参数 coeffs 为常数；若参数 constrs 为*ConstrArray 类* 对象或列表对象，则参数 coeffs 为常数或列表对象；若参数

constrs 为字典对象或*tupledict 类* 对象，则参数 coeffs 为常数、字典或*tupledict 类* 对象。

参量

constrs

待添加项对应的约束。

coeffs

待添加项对应的系数。

示例

```
# 添加两个项：约束conx中变量系数为2.0，约束cony中变量系数为3.0
col.addTerms([conx, cony], [2.0, 3.0])
```

Column.addColumn()

摘要

addColumn(col, mult=1.0)

描述

添加新的列到当前列中。

参量

col

待添加列对象。

mult

待添加列的放大系数。可选参量，默认值为 1.0。

示例

```
# 添加列coly中的项到列colx中，coly中的项放大系数为2.0
colx.addColumn(coly, 2.0)
```

Column.clone()

摘要

clone()

描述

创建列对象的深拷贝。

示例

```
# 创建列col的深拷贝
colcopy = col.clone()
```

Column.remove()

摘要

```
remove(item)
```

描述

从列对象中移除指定的项。

若参数 `item` 为常数，则移除指定下标对应的项；否则参数 `item` 为*Constraint* 类对象。

参量

`item`

常数下标或待移除项相应的线性约束。

示例

```
# 从列col中移除下标为2相应的项
col.remove(2)
# 从列col中移除线性约束conx所在的项
col.remove(conx)
```

Column.clear()

摘要

```
clear()
```

描述

清空列对象中的内容。

示例

```
# 清空列col中的内容
col.clear()
```

23.2.37 ColumnArray 类

为方便用户对一组*Column 类* 对象进行操作，杉数求解器的 Python 接口设计了 ColumnArray 类，提供了以下成员方法：

ColumnArray()

摘要

ColumnArray(columns=None)

描述

创建一个*ColumnArray 类* 对象。

若参数 columns 为 None，则创建一个空的*ColumnArray 类* 对象，否则以参数 columns 初始化新创建的*ColumnArray 类* 对象。

参量

columns

待添加列。可选参量，默认为 None。可取值为*Column 类* 对象、*ColumnArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 创建一个空的ColumnArray对象
colarr = ColumnArray()
# 创建一个ColumnArray对象，并使用列colx和coly初始化
colarr = ColumnArray([colx, coly])
```

ColumnArray.pushBack()

摘要

pushBack(column)

描述

添加单个或多个*Column 类* 对象。

参量

column

待添加列。可取值为*Column 类* 对象、*ColumnArray 类* 对象、列表、字典或*tupledict 类* 对象。

示例

```
# 添加列colx到colarr中
colarr.pushBack(colx)
```

(续下页)

(接上页)

```
# 添加列colx和coly到colarr中
colarr.pushBack([colx, coly])
```

ColumnArray.getColumn()

摘要

getColumn(idx)

描述

根据列在*ColumnArray 类* 对象中的下标获取相应的列，返回一个*Column 类* 对象。

参量

idx

列在*ColumnArray 类* 对象中的下标。起始为 0。

示例

```
# 获取colarr中下标为1的列
col = colarr.getColumn(1)
```

ColumnArray.getSize()

摘要

getSize()

描述

获取*ColumnArray 类* 对象中元素的个数。

示例

```
# 获取colarr中元素的个数
colsize = colarr.getSize()
```

ColumnArray.clear()

摘要

clear()

描述

清空*ColumnArray 类* 对象中的内容。

示例

```
# 清空colarr中的内容
colarr.clear()
```

23.2.38 MVar 类

MVar 类是杉数求解器中用于构建多维变量，并支持 NumPy 的多维数组运算的类。我们建议通过模型类的方法 addVars 或者 addMVar 来生成，虽然也可以通过 fromlist 和 fromvar 这两个内置类方法转换生成。提供了以下成员方法：

MVar.fromlist()

摘要

fromlist(vars)

描述

从一组 *Var 类* 对象生成一个 *MVar 类* 对象。这是类生成方法，可以直接调用，无需 MVar 对象。

参量

vars

一组 Var 对象，可以是多维的 list，或者 ndarray。

返回值

新的 MVar 对象，其维度取决于参量 vars 的维度。

示例

```
vars = model.addVars(4)
mx_1d = MVar.fromlist(vars)
mx_2d = MVar.fromlist([vars[0], vars[1]], [vars[2], vars[3]])
```

MVar.fromvar()

摘要

fromvar(var)

描述

从一个 *Var 类* 对象生成一个 0 维的 *MVar 类* 对象。这是类生成方法，可以直接调用，无需 MVar 对象。

参量

var

一个 Var 对象。

返回值

新的 0 维 MVar 对象。

示例

```
x = model.addVar()
mx_0d = MVar.fromvar(x)
```

MVar.clone()

摘要

clone()

描述

深度复制一个*MVar 类* 对象。

返回值

新的 MVar 对象

示例

```
# 创建一个2-D变量，并复制一份。注意实际变量没有增加。
mx = model.addMVar((3, 2), nameprefix="mx")
mx_copy = mx.clone()
```

MVar.diagonal()

摘要

diagonal(offset=0, axis1=0, axis2=1)

描述

生成一个*MVar 类* 对象，其元素是原 MVar 对象对角线上的元素。

参量

offset

可选参量，表示对角线的偏移量，默认值为 0。如果值大于 0，表示对角线向上的偏移量；如果值小于 0，表示对角线向下的偏移量。

axis1

可选参量，用来作为二维子 MVar 的第一轴的轴, 对角线应该从这里开始。默认第一轴为 0。

axis2

可选参量，用来作为二维子 MVar 的第二轴的轴, 对角线应该从这里开始。默认第二轴为 1。

返回值

新的 MVar 对象

示例

```
mx = model.addMVar((5, 5), nameprefix="mx")
diag_m0 = mx.diagonal()
diag_a1 = mx.diagonal(1)
diag_b1 = mx.diagonal(-1)
```

MVar.getInfo()

摘要

getInfo(infoname)

描述

获取 MVar 内每个变量的信息值。

参量

infoname

待查询信息的名称。可取值详见信息章节 。

返回值

返回一个 NumPy 的 ndarray，维度和 MVar 对象维度相同，其元素是对应变量信息的取值。

示例

```
mx = model.addMVar(3)
print(mx.getInfo("LB"))
```

MVar.item()

摘要

item()

描述

获取 0 维 MVar 内的 Var 变量。如果 MVar 对象不是 0 维，触发 ValueError 异常。

返回值

返回 Var 对象。

示例

```
mx = model.addMVar(3)
var = mx[0].item()
```

MVar.reshape()

摘要

reshape(shape, order='C')

描述

返回一个新的 MVar 对象，其元素保持不变，但形状按参量 shape 转换。

参量

shape

取值为整数，或者整数元组。表示新 MVar 对象的形状。

order

可选参量，默认为字符'C'，表示兼容 C 语言，即按行存储；也可设为字符'F'，即按列存储，兼容 Fortune 语言。

返回值

返回一个元素和原 MVar 对象相同，但形状不同的新 MVar 对象。

示例

```
mx = model.addMVar(6)
mx_2x2 = mx.reshape((2, 3))
```

MVar.setInfo()

摘要

setInfo(infoname, newval)

描述

设置 MVar 内每个变量的信息值。

参量

infoname

待设置的信息名称。可取值详见信息章节 。

newval

待设置新的信息取值。

示例

```
mx = model.addMVar(3)
mx.setInfo("UB", 9.0)
mx.setInfo(COPT.Info.LB, 0.0)
```

MVar.sum()

摘要

sum(axis=None)

描述

对 MVar 里的变量求和，返回一个新的*MLinExpr 类* 对象。

参量

axis

可选整型参量，默认值为 None，即逐个变量求和。否则，按给定的轴求和。

返回值

返回 MLinExpr 对象，表示对应变量的和。

示例

```
mx = model.addMVar((3, 5))
sum_all = mx.sum() #返回0维MLinExpr对象
sum_row = mx.sum(axis = 0) #返回1维MLinExpr对象，形状为(5, )
```

MVar.tolist()

摘要

tolist()

描述

把 MVar 对象转化为 Var 对象的一维列表。

返回值

返回包含 Var 对象的一维列表。

示例

```
mx = model.addMVar((3, 5))
print(mx.tolist())
```

MVar.transpose()

摘要

transpose()

描述

生成一个新的 MVar 对象，它是原 MVar 对象的转置。

返回值

返回新的 MVar 对象。

示例

```
mx = model.addMVar((3, 5))
print(mx.transpose().shape) #其形状为(5, 3)
```

MVar.ndim

摘要

ndim

描述

MVar 对象的维度。

返回值

整型值。

示例

```
mx = model.addMVar((3, 5))
print(mx.ndim) # ndim = 2
```

MVar.shape

摘要

shape

描述

MVar 对象的形状。

返回值

整型元组。

示例

```
mx = model.addMVar((3,))
print(mx.shape) # shape = (3, )
```

MVar.size

摘要

size

描述

MVar 对象的 Var 变量个数。

返回值

整型值。

示例

```
mx = model.addMVar((3, 4))
print(mx.size) # size = 12
```

MVar.T

摘要

T

描述

MVar 对象的转置。类似于类方法 transpose()。

返回值

返回转置后的 MVar 对象。

示例

```
mx = model.addMVar((3, 4))
print(mx.T.shape) # shape = (4, 3)
```

23.2.39 MConstr 类

MConstr 类是杉数求解器中用于构建多维约束，并支持 NumPy 的多维数组运算的类。需要通过模型类的方法 addConstrs 或者 addMConstr 来生成。提供了以下成员方法：

MConstr.getInfo()

摘要

getInfo(infoname)

描述

获取 MConstr 内每个约束的信息值。可取值详见信息章节 。

参量

infoname

待获取信息的名称。

返回值

返回一个 NumPy 的 ndarray，维度和 MConstr 对象维度相同，其元素是对应约束的信息值。

示例

```
a = np.random.rand(4)
mx = m.addMVar((4, 3), nameprefix="mx")
b = np.random.rand(3)
mc = m.addConstrs(a @ mx <= b)
print(mc.getInfo("pi"))
```

MConstr.item()

摘要

item()

描述

获取 0 维 MConstr 内的约束对象。如果 MConstr 对象不是 0 维，触发 ValueError 异常。

返回值

返回线性约束对象。

示例

```
mc = m.addConstrs(a @ mx <= b)
print(mc[0].item())
```

MConstr.reshape()

摘要

reshape(shape, order='C')

描述

返回一个新的 MConstr 对象，其元素保持不变，但形状按参量 shape 转换。

参量

shape

取值为整数，或者整数元组。表示新 MConstr 对象的形状。

order

可选参量，默认为字符'C'，表示兼容 C 语言，即按行存储；也可设为字符'F'，即按列存储，兼容 Fortune 语言。

返回值

返回一个元素和原 MConstr 对象相同，但形状不同的新 MConstr 对象。

示例

```
mc = m.addConstrs(a @ mx <= b)
mc_2x2 = mc.reshape((2, 2))
```

MConstr.setInfo()

摘要

```
setInfo(infoname, newval)
```

描述

设置 MConstr 内每个约束的信息值。

参量

`infoname`

待设置的信息名称。可取值详见信息章节 。

`newval`

待设置新的信息取值。

示例

```
mc = model.addConstrs(a @ mx <= b)
mc.setInfo("obj", 9.0)
```

MConstr.tolist()

摘要

```
tolist()
```

描述

把 MConstr 对象转化为元素为约束的一维列表。

返回值

返回包含 Constraint 对象的一维列表。

示例

```
mc = m.addConstrs(a @ mx <= b)
print(mc.tolist())
```

MConstr.transpose()

摘要

transpose()

描述

生成一个新的 MConstr 对象，它是原 MConstr 对象的转置。

返回值

返回转置后的 MConstr 对象。

示例

```
mc = m.addConstrs(a @ mx <= b)
print(mc.transpose())
```

MConstr.ndim

摘要

ndim

描述

MConstr 对象的维度。

返回值

整型值。

示例

```
mc = m.addConstrs(a @ mx <= b)
print(mc.ndim)
```

MConstr.shape

摘要

shape

描述

MConstr 对象的形状。

返回值

整型元组。

示例

```
mc = m.addConstrs(a @ mx <= b)
print(mc.shape)
```

MConstr.size

摘要

size

描述

MConstr 对象的约束个数。

返回值

整型值。

示例

```
mc = m.addConstrs(a @ mx <= b)
print(mc.size)
```

MConstr.T

摘要

T

描述

MConstr 对象的转置。类似于类方法 transpose()。

返回值

返回转置后的 MConstr 对象。

示例

```
A = np.ones([2, 4])
mx = m.addMVar((4, 3), nameprefix="mx")
mc = m.addConstrs(A @ X == 0.0)
print(mc.T.shape) # shape = (3, 2)
```

23.2.40 MConstrBuilder 类

MConstrBuilder 类是杉数求解器中多维线性约束的生成器，并支持 NumPy 的多维数组运算。需要通过一组*ConstrBuilder 类*对象生成，或者*MVar 类*对象，*MLinExpr 类*对象的比较运算符重载来生成。提供了以下成员方法：

MConstrBuilder()

摘要

MConstrBuilder(args, shape=None)

描述

类构造方法，生成 MConstrBuilder 对象。

参量

args

单个或者一组*ConstrBuilder 类* 对象。可以是 Python 列表，或者 NumPy 多维数组。

shape

取值为整数，或者整数元组。表示新 MConstrBuilder 对象的形状。

示例

```
vars = m.addVars(4)
builders = [x <= 1.0 for x in vars]
mcb = MConstrBuilder(builders, (2, 2))

# 或者通过MVar对象的比较运算符来生成
mx = m.addMVar((3, 2))
mcb = mx >= 1.0

# 它可以作为addConstrs的输入
model.addConstrs(mx >= 1.0)
```

23.2.41 MQConstrBuilder 类

MQConstrBuilder 类是杉数求解器中多维二次约束的生成器，并支持 NumPy 的多维数组运算。需要通过一组*QConstrBuilder 类* 对象生成，或者*MQuadExpr 类* 对象的比较运算符重载来生成。提供了以下成员方法：

MQConstrBuilder()

摘要

MQConstrBuilder(args, shape=None)

描述

类构造方法，生成 MQConstrBuilder 对象。

参量

args

单个或者一组*QConstrBuilder 类* 对象。可以是 Python 列表，或者 NumPy 多维数组。

shape

取值为整数，或者整数元组。表示新 MQConstrBuilder 对象的形状。

示例

```
x = model.addVar()
mqcb = MQConstrBuilder(x * x <= 9.0)

# 或者通过MVar对象的矩阵乘法和MQuadExpr的比较运算符来生成
mx = model.addMVar(3, 3)
mqcb = mx @ mx >= 1.0

# 它可以作为addQConstr的输入
ma = model.addMVar(2)
A = np.full((2,3), 1)
mb = model.addMVar(3)
model.addQConstr(ma @ A @ mb <= 1.0)
```

23.2.42 MLinExpr 类

MLinExpr 类是杉数求解器中用于构建多维线性表达式，并支持 NumPy 的多维数组运算的类。可以通过类生成方法 zeros() 来生成初始值为 0.0 的 MLinExpr 对象，也可以通过对*MVar 类* 对象的线性组合来生成。提供了以下成员方法：

MLinExpr.zeros()

摘要

zeros(shape)

描述

这是类生成方法，可以直接调用，无需 MLinExpr 对象。

参量

shape

取值为整数，或者整数元组。表示新 MLinExpr 对象的形状。

返回值

新的 MLinExpr 对象。

示例

```
mexpr = MLinExpr.zeros((2,3))
x = model.addVar()
mexpr += x
```

MLinExpr.clear()

摘要

clear()

描述

重置*MLinExpr 类* 对象的每个元素为 0.0。

示例

```
mexpr = 2.0 * model.addMVar(3)
mexpr.clear()
```

MLinExpr.clone()

摘要

clone()

描述

深度复制一个*MLinExpr 类* 对象。

返回值

新的 MLinExpr 对象

示例

```
mexpr = 2.0 * model.addMVar(3)
mexpr_copy = mexpr.clone()
```

MLinExpr.getValue()

摘要

getValue()

描述

获取*MLinExpr 类* 对象内每个线性表达式的估值。

返回值

返回一个 NumPy 的 ndarray，维度和 MLinExpr 对象维度相同，其元素是对应表达式的估值。

示例

```
a = np.random.rand(4)
mx = m.addMVar((4, 3), nameprefix="mx")
mexpr = a @ mx
```

(续下页)

(接上页)

```
mc = m.addConstrs(mexpr <= 1.0)
model.solve()
print(mc.getValue())
```

MLinExpr.item()

摘要

item()

描述

获取 0 维 MLinExpr 内的约束对象。如果 MLinExpr 对象不是 0 维，触发 ValueError 异常。

返回值

返回线性约束对象。

示例

```
mexpr = 2.0 * model.addMVar(3)
print(mexpr[0].item())
```

MLinExpr.reshape()

摘要

reshape(shape, order='C')

描述

返回一个新的 MLinExpr 对象，其元素保持不变，但形状按参量 shape 转换。

参量

shape

取值为整数，或者整数元组。表示新 MLinExpr 对象的形状。

order

可选参量，默认为字符'C'，表示兼容 C 语言，即按行存储；也可设为字符'F'，即按列存储，兼容 Fortune 语言。

返回值

返回一个元素和原 MLinExpr 对象相同，但形状不同的新 MLinExpr 对象。

示例

```
mc = m.addConstrs(a @ mx <= b)
mc_2x2 = mc.reshape((2, 2))
```

MLinExpr.sum()

摘要

sum(axis=None)

描述

对 MLinExpr 对象里的变量求和，返回一个新的*MLinExpr 类* 对象。

参量

axis

可选整型参量，默认值为 None，即逐个变量求和。否则，按给定的轴求和。

返回值

返回 MLinExpr 对象，表示对应线性表达式的和。

示例

```
mexpr = 2.0 * model.addMVar((3, 5))
sum_all = mexpr.sum() #返回0维MLinExpr对象
sum_row = mexpr.sum(axis = 0) #返回1维MLinExpr对象，形状为(5, )
```

MLinExpr.tolist()

摘要

tolist()

描述

把 MLinExpr 对象转化为元素为线性表达式的一维列表。

返回值

返回包含*LinExpr 类* 的一维列表。

示例

```
mexpr = 2.0 * model.addMVar((3, 5))
print(mexpr.tolist())
```

MLinExpr.transpose()

摘要

transpose()

描述

生成一个新的 MLinExpr 对象，它是原 MLinExpr 对象的转置。

返回值

返回转置后的 MLinExpr 对象。

示例

```
mexpr = 2.0 * model.addMVar((3, 5))
print(mexpr.transpose())
```

MLinExpr.ndim

摘要

ndim

描述

MLinExpr 类 对象的维度。

返回值

整型值。

示例

```
mexpr = 2.0 * model.addMVar((3, 5))
print(mexpr.ndim)
```

MLinExpr.shape

摘要

shape

描述

MLinExpr 类 对象的形状。

返回值

整型元组。

示例

```
mexpr = 2.0 * model.addMVar((3, 5))
print(mexpr.shape)
```

MLinExpr.size

摘要

size

描述

MLinExpr 类 对象的元素个数。

返回值

整型值。

示例

```
mexpr = 2.0 * model.addMVar((3, 5))
print(mexpr.size)
```

MLinExpr.T

摘要

T

描述

MLinExpr 类 对象的转置。类似于类方法 transpose()。

返回值

返回转置后的 MLinExpr 对象。

示例

```
mexpr = 2.0 * model.addMVar((3, 5))
print(mexpr.T.shape) # 转置后的形状为(5, 3)
```

MLinExpr.__eq__()

摘要

__eq__()

描述

对运算符 == 的重载，生成一个*MConstrBuilder 类* 对象。可以作为模型类里的 *addConstrs* 的输入。

返回值

返回*MConstrBuilder 类* 对象。

示例

```
model.addConstrs(A @ x == 1.0)
```

MLinExpr.__ge__()

摘要

__ge__()

描述

对运算符 >= 的重载，生成一个*MConstrBuilder 类* 对象。可以作为模型类里的 *addConstrs* 的输入。

返回值

返回*MConstrBuilder 类* 对象。

示例

```
model.addConstrs(A @ x >= 1.0)
```

MLinExpr.__le__()

摘要

__le__()

描述

对运算符 <= 的重载，生成一个*MConstrBuilder 类* 对象。可以作为模型类里的 *addConstrs* 的输入。

返回值

返回*MConstrBuilder 类* 对象。

示例

```
model.addConstrs(A @ x <= 1.0)
```

23.2.43 MQuadExpr 类

MQuadExpr 类是杉数求解器中用于构建多维二次表达式，并支持 NumPy 的多维数组运算的类。可以通过类生成方法 zeros() 来生成初始值为 0.0 的 MQuadExpr 对象，也可以通过对两个*MVar 类* 对象的（矩阵）相乘来生成。提供了以下成员方法：

MQuadExpr.zeros()

摘要

zeros(shape)

描述

这是类生成方法，可以直接调用，无需 MQuadExpr 对象。

参量

shape

取值为整数，或者整数元组。表示新 MQuadExpr 对象的形状。

返回值

新的 MQuadExpr 对象。

示例

```
mqx = MQuadExpr.zeros((2,3)) # shape = (2, 3)
x = model.addVar()
mqx += 2.0 * x * x           # broadcast scalar
mqx += model.addMVar(3)      # broadcast MVar of shape (3,)
```

MQuadExpr.clear()

摘要

clear()

描述

重置*MQuadExpr 类* 对象的每个元素为 0.0。

示例

```
ma = model.addMVar(3, nameprefix='a')
mb = model.addMVar(3, nameprefix='b')
mqx = ma * mb    # elementwise multiply, shape = (3,)
mqx.clear()
print(mqx)       # result is [0.0, 0.0, 0.0]
```

MQuadExpr.clone()

摘要

clone()

描述

深度复制一个*MQuadExpr 类* 对象。

返回值

新的*MQuadExpr 类* 对象。

示例

```
mx = model.addMVar((3, 3), nameprefix='mx')
mqx = 2.0 * mx @ mx       # matrix multiply, shape = (3, 3)
mqx_copy = mqx.clone()
mqx_copy.clear()
print(mqx)                # mqx is untouched
```

MQuadExpr.getValue()

摘要

getValue()

描述

获取*MQuadExpr 类* 对象内每个二次表达式的估值。

返回值

返回一个 NumPy 的 ndarray，维度和 MQuadExpr 对象维度相同，其元素是对应表达式的估值。

示例

```
A = np.eye(3)
mx = m.addMVar(3, nameprefix="mx")
mqx = mx @ A @ mx         # 0-D MQuadExpr, shape = ()
m.addQConstr(mqx <= 9.0)
m.solve()
print(mqx.getValue())
```

MQuadExpr.item()

摘要

item()

描述

获取 0 维 MQuadExpr 内的约束对象。如果 MQuadExpr 对象不是 0 维，触发 ValueError 异常。

返回值

返回二次表达式对象。

示例

```
x = m.addVar()
mqx = MQuadExpr.zeros(3) + x * x
print(mqx[1].item()) # 返回 QuadExpr(x * x)
```

MQuadExpr.reshape()

摘要

reshape(shape, order='C')

描述

返回一个新的 MQuadExpr 对象，其元素保持不变，但形状按参量 shape 转换。

参量

shape

取值为整数，或者整数元组。表示新 MQuadExpr 对象的形状。

order

可选参量，默认为字符'C'，表示兼容 C 语言，即按行存储；也可设为字符'F'，即按列存储，兼容 Fortune 语言。

返回值

返回一个元素和原 MQuadExpr 对象相同，但形状不同的新 MQuadExpr 对象。

示例

```
mqx = MQuadExpr.zeros(6)
mqx_2x3 = mqx.reshape((2, 3))
```

MQuadExpr.sum()

摘要

```
sum(axis=None)
```

描述

对 MQuadExpr 对象里的二次表达式求和，返回一个新的*MQuadExpr 类* 对象。

参量

axis

可选整型参量，默认值为 None，即逐个表达式求和。否则按给定的轴求和。

返回值

返回 MQuadExpr 对象，表示对应二次表达式的和。

示例

```
ma = model.addMVar((2, 3), nameprefix='ma')
mb = model.addMVar((3, 2), nameprefix='mb')
mqx = ma @ mb
sum_all = mqx.sum()          # 返回0维MQuadExpr对象
sum_row = mqx.sum(axis = 0) # 返回1维MQuadExpr对象，形状为(2,)
```

MQuadExpr.tolist()

摘要

```
tolist()
```

描述

把 MQuadExpr 对象转化为元素为二次表达式的一维列表。

返回值

返回元素为*QuadExpr 类* 的一维列表。

示例

```
print(MQuadExpr.zeros((2,3)).tolist()) # 长度为6的列表
```

MQuadExpr.transpose()

摘要

transpose()

描述

生成一个新的 MQuadExpr 对象，它是原 MQuadExpr 对象的转置。

返回值

返回转置后的 MQuadExpr 对象。

示例

```
mqx = MQuadExpr.zeros((2,3))
print(mqx.transpose().shape) # shape = (3, 2)
```

MQuadExpr.ndim

摘要

ndim

描述

MQuadExpr 类 对象的维度。

返回值

整型值。

示例

```
mqx = MQuadExpr.zeros((2,3))
print(mqx.ndim)  # ndim = 2
```

MQuadExpr.shape

摘要

shape

描述

MQuadExpr 类 对象的形状。

返回值

整型元组。

示例

```
print(MQuadExpr.zeros((2,3)).shape) # shape = (2, 3)
```

MQuadExpr.size

摘要

size

描述

MQuadExpr 类 对象的元素个数。

返回值

整型值。

示例

```
mqx = MQuadExpr.zeros((2,3))
print(mqx.size)  # size= 6
```

MQuadExpr.T

摘要

T

描述

MQuadExpr 类 对象的转置。类似于类方法 transpose()。

返回值

返回转置后的 MQuadExpr 对象。

示例

```
mqx = MQuadExpr.zeros((2,3))
print(mqx.T.shape) # shape = (3, 2)
```

MQuadExpr.__eq__()

摘要

__eq__()

描述

对运算符 == 的重载，生成一个*MQConstrBuilder 类* 对象。可以作为模型类里的 *addQConstr* 的输入。

返回值

返回*MQConstrBuilder 类* 对象。

示例

```
model.addQConstr(x @ Q @ y == 1.0)
```

MQuadExpr.__ge__()

摘要

__ge__()

描述

对运算符 >= 的重载，生成一个*MQConstrBuilder 类* 对象。可以作为模型类里的 *addQConstr* 的输入。

返回值

返回*MQConstrBuilder 类* 对象。

示例

```
model.addQConstr(x @ Q @ y >= 1.0)
```

MQuadExpr.__le__()

摘要

__le__()

描述

对运算符 <= 的重载，生成一个*MQConstrBuilder 类* 对象。可以作为模型类里的 *addQConstr* 的输入。

返回值

返回*MQConstrBuilder 类* 对象。

示例

```
model.addQConstr(x @ Q @ y <= 1.0)
```

23.2.44 ExprBuilder 类

ExprBuilder 类是杉数求解器中用于构建线性约束的构建器，提供了以下成员方法：

ExprBuilder()

摘要

ExprBuilder(arg1=0.0, arg2=None)

描述

创建一个*ExprBuilder 类* 对象。

若参数 arg1 为常数，参数 arg2 为 None，则创建一个*ExprBuilder 类* 对象，并以参数 arg1 的值初始化；若参数 arg1 为*Var 类* 对象或*ExprBuilder 类* 对象，则参数 arg2 为常数，参数 arg2 可为 None，此时当作常数 1.0，并以参数 arg1 和 arg2 初始化新创建的*ExprBuilder 类* 对象；若参数 arg1 和 arg2 为列表对象，则分别表示组成线性表达式的变量、系数，并初始化新创建的*ExprBuilder 类* 对象。

参量

arg1

可选参量，默认值为 0.0。

arg2

可选参量，默认值为 None。

示例

```
# 创建一个新的ExprBuilder对象，并初始化为：0.0
expr0 = ExprBuilder()
# 创建一个ExprBuilder对象，并初始化为：x + 2*y
expr2 = ExprBuilder([x, y], [1, 2])
```

ExprBuilder.getSize()

摘要

getSize()

描述

获取表达式构建器中项的个数。

示例

```
# 获取线性表达式构建器expr中元素的个数
exprsize = expr.getSize()
```

ExprBuilder.getCoeff()

摘要

getCoeff(idx)

描述

根据变量在表达式构建器中的下标获取其系数。

参量

idx

变量在表达式构建器中的下标。起始为 0。

示例

```
# 获取线性表达式构建器expr第1项的系数
coeff = expr.getCoeff(1)
```

ExprBuilder.getVar()

摘要

getVar(idx)

描述

根据变量在表达式中构建器的下标获取相应的变量，返回一个 *Var 类* 对象。

参量

idx

变量在表达式构建器中的下标。起始为 0。

示例

```
# 获取线性表达式构建器expr第1项的变量
x = expr.getVar(1)
```

ExprBuilder.getConstant()

摘要

getConstant()

描述

获取表达式构建器中的常数项。

示例

```
# 获取线性表达式expr的常数项
constant = expr.getConstant()
```

ExprBuilder.addTerm()

摘要

addTerm(var, coeff=1.0)

描述

添加新的项到当前表达式构建器中。

参量

var

待添加项的变量。

coeff

待添加项的放大系数。可选参量，默认值为 1.0。

示例

```
# 添加项：2*x 到线性表达式构建器expr中
expr.addTerm(x, 2.0)
```

ExprBuilder.addExpr()

摘要

addExpr(expr, coeff=1.0)

描述

添加新的表达式构建器到当前表达式构建器中。

参量

expr

待添加表达式构建器。

coeff

待添加表达式构建器的放大系数。可选参量，默认值为 1.0。

示例

```
# 添加线性表达式构建器对象 2*x + 2*y 到线性表达式构建器expr中
expr.addExpr(x + y, 2.0)
```

ExprBuilder.clone()

摘要

clone()

描述

创建表达式构建器对象的深拷贝。

示例

```
# 创建线性表达式构建器expr的深拷贝
exprcopy = expr.clone()
```

ExprBuilder.getExpr()

摘要

getExpr()

描述

创建并获取线性表达式构建器相应的表达式，返回一个*LinExpr 类* 对象。

示例

```
# 获取线性表达式构建器exprbuilder相应的表达式
expr = exprbuilder.getExpr()
```

23.2.45 LinExpr 类

LinExpr 类是杉数求解器中用于构建线性表达式时变量的相关组合操作，提供了以下成员方法：

LinExpr()

摘要

LinExpr(arg1=0.0, arg2=None)

描述

创建一个*LinExpr 类* 对象。

若参数 arg1 为常数，参数 arg2 为 None，则创建一个*LinExpr 类* 对象，并以参数 arg1 的值初始化；若参数 arg1 为*Var 类* 对象或*LinExpr 类* 对象，则参数 arg2 为常数，参数 arg2 可为 None，此时当作常数 1.0，并以参数 arg1 和 arg2 初始化新创建的*LinExpr 类* 对象；若参数 arg1 为列表对象，参数 arg2 为 None，则参数 arg1 中的元素为变量、系数对，并以参数 arg1 和 arg2 初始化新创建的*LinExpr 类* 对象；对于其它参数情形，则调用成员方法 addTerms 初始化新创建的*LinExpr 类* 对象。

参量

arg1

可选参量，默认值为 0.0。

arg2

可选参量，默认值为 None。

示例

```
# 创建一个新的LinExpr对象，并初始化为：0.0
expr0 = LinExpr()
# 创建一个LinExpr对象，并初始化为：2*x + 3*y
expr1 = LinExpr([(x, 2), (y, 3)])
# 创建一个LinExpr对象，并初始化为：x + 2*y
expr2 = LinExpr([x, y], [1, 2])
```

LinExpr.setCoeff()

摘要

setCoeff(idx, newval)

描述

根据变量在表达式中的下标设置其系数。

参量

idx

变量在表达式中的下标。起始为 0。

newval

变量的新系数。

示例

```
# 设置线性表达式expr第0项的系数为1.0
expr.setCoeff(0, 1.0)
```

LinExpr.getCoeff()

摘要

getCoeff(idx)

描述

根据变量在表达式中的下标获取其系数。

参量

idx

变量在表达式中的下标。起始为 0。

示例

```
# 获取线性表达式expr第1项的系数
coeff = expr.getCoeff(1)
```

LinExpr.getVar()

摘要

getVar(idx)

描述

根据变量在表达式中的下标获取相应的变量，返回一个*Var 类* 对象。

参量

idx

变量在表达式中的下标。起始为 0。

示例

```
# 获取线性表达式expr第1项的变量
x = expr.getVar(1)
```

LinExpr.getConstant()

摘要

getConstant()

描述

获取表达式中的常数项。

示例

```
# 获取线性表达式expr的常数项
constant = expr.getConstant()
```

LinExpr.getValue()

摘要

getValue()

描述

获取以变量的取值计算出的表达式的值。

示例

```
# 获取线性表达式expr的当前值
val = expr.getValue()
```

LinExpr.getSize()

摘要

getSize()

描述

获取表达式中项的个数。

示例

```
# 获取线性表达式expr中元素的个数
exprsize = expr.getSize()
```

LinExpr.setConstant()

摘要

setConstant(newval)

描述

设置表达式的常数项。

参量

newval

待设置常数值。

示例

```
# 设置线性表达式expr的常数项为2.0
expr.setConstant(2.0)
```

LinExpr.addConstant()

摘要

addConstant(newval)

描述

添加常数到表达式。

参量

newval

待添加常数值。

示例

```
# 添加常数2.0到线性表达式expr中
expr.addConstant(2.0)
```

LinExpr.addTerm()

摘要

addTerm(var, coeff=1.0)

描述

添加新的项到当前表达式中。

参量

expr

待添加项的变量。

coeff

待添加项的放大系数。可选参量，默认值为 1.0。

示例

```
# 添加项：x到线性表达式expr中
expr.addTerm(x)
```

LinExpr.addTerms()

摘要

addTerms(vars, coeffs)

描述

添加单个或多个新项到表达式。

若参数 vars 为*Var 类* 对象，则参数 coeffs 为常数；若参数 vars 为*VarArray 类* 对象或列表对象，则参数 coeffs 为常数或列表对象；若参数 vars 为字典对象或*tupledict 类* 对象，则参数 coeffs 为常数、字典或*tupledict 类* 对象。

参量

vars

待添加的变量。

coeffs

待添加项的系数。

示例

```
# 添加项：2*x + 2*y到线性表达式expr中
expr.addTerms([x, y], [2.0, 3.0])
```

LinExpr.addExpr()

摘要

addExpr(expr, coeff=1.0)

描述

添加新的表达到当前表达式中。

参量

expr

待添加表达式或者表达式构建器对象。

coeff

待添加表达式的放大系数。可选参量，默认值为 1.0。

示例

```
# 添加线性表达式：2*x + 2*y到线性表达式expr中
expr.addExpr(x + y, 2.0)
```

LinExpr.clone()

摘要

clone()

描述

创建表达式对象的深拷贝。

示例

```
# 创建线性表达式expr的深拷贝
exprcopy = expr.clone()
```

LinExpr.reserve()

摘要

reserve(n)

描述

为表达式对象预分配空间。

参量

n

预分配的表达式中项的数目。

示例

```
# 预分配表达式expr中项的数目为100
expr.reserve(100)
```

LinExpr.remove()

摘要

remove(item)

描述

从表达式中移除指定项。

若参数 item 为常数，则移除指定下标对应的项；否则参数 item 为*Var 类* 对象。

参量

item

常数下标或待移除项相应的变量。

示例

```
# 从线性表达式expr中移除下标为2相应的项
expr.remove(2)
# 从线性表达式expr中移除变量x相应的项
expr.remove(x)
```

23.2.46 QuadExpr 类

QuadExpr 类是杉数求解器中用于构建线性表达式时变量的相关组合操作，提供了以下成员方法：

QuadExpr()

摘要

QuadExpr(expr=0.0)

描述

创建一个*QuadExpr 类* 对象。

参数 expr 为常数、*Var 类* 对象、*LinExpr 类* 对象或*QuadExpr 类* 对象。

参量

expr

可选参量，默认值为 0.0。

示例

```
# 创建一个新的QuadExpr对象，并初始化为：0.0
quadexpr0 = QuadExpr()
# 创建一个QuadExpr对象，并初始化为：2*x + 3*y
quadexpr1 = QuadExpr(2*x + 3*y)
# 创建一个QuadExpr对象，并初始化为：x*x + 2*y*z
quadexpr2 = QuadExpr(x*x + 2*y*z)
```

QuadExpr.setCoeff()

摘要

```
setCoeff(idx, newval)
```

描述

设置二次表达式中指定索引值对应的二次项系数。

参量

idx

指定的索引值。起始为 0。

newval

变量的新系数。

示例

```
# 设置二次表达式quadexpr第0项的系数为1.0
quadexpr.setCoeff(0, 1.0)
```

QuadExpr.getCoeff()

摘要

```
getCoeff(idx)
```

描述

根据变量在表达式中的下标获取其系数。

参量

idx

变量在表达式中的下标。起始为 0。

示例

```
# 获取二次表达式quadexpr第1项的系数
coeff = quadexpr.getCoeff(1)
```

QuadExpr.getVar1()

摘要

getVar1(idx)

描述

获取指定的二次项的第一个变量，返回一个*Var 类* 对象。

参量

idx

二次项的下标。起始为 0。

示例

```
# 获取第1个二次项的第1个变量
x = expr.getVar1(1)
```

QuadExpr.getVar2()

摘要

getVar2(idx)

描述

获取指定的二次项的第二个变量，返回一个*Var 类* 对象。

参量

idx

二次项的下标。起始为 0。

示例

```
# 获取第1个二次项的第2个变量
y = expr.getVar2(1)
```

QuadExpr.getLinExpr()

摘要

getLinExpr()

描述

获取二次表达式中的线性表达式。

示例

```
# 获取二次表达式quadexpr中的线性表达式linexpr
linexpr = quadexpr.getLinExpr()
```

QuadExpr.getConstant()

摘要

getConstant()

描述

获取表达式中的常数项。

示例

```
# 获取二次表达式quadexpr的常数项
constant = quadexpr.getConstant()
```

QuadExpr.getValue()

摘要

getValue()

描述

获取以变量的取值计算出的表达式的值。

示例

```
# 获取二次表达式quadexpr的当前值
val = quadexpr.getValue()
```

QuadExpr.getSize()

摘要

getSize()

描述

获取表达式中项的个数。

示例

```
# 获取二次表达式quadexpr中元素的个数
exprsize = quadexpr.getSize()
```

QuadExpr.setConstant()

摘要

setConstant(newval)

描述

设置表达式的常数项。

参量

newval

待设置常数值。

示例

```
# 设置二次表达式quadexpr的常数项为2.0
quadexpr.setConstant(2.0)
```

QuadExpr.addConstant()

摘要

addConstant(newval)

描述

添加常数到表达式。

参量

newval

待添加常数值。

示例

```
# 添加常数2.0到二次表达式quadexpr中
quadexpr.addConstant(2.0)
```

QuadExpr.addTerm()

摘要

addTerm(coeff, var1, var2=None)

描述

添加新的项到当前表达式中。

参量

coeff

待添加项的系数。

var1

待添加项的第一个变量。

var2

待添加项的第二个变量。可以为 None，表示添加线性项。

示例

```
# 添加项：x到二次表达式quadexpr中
quadexpr.addTerm(1.0, x)
```

QuadExpr.addTerms()

摘要

addTerms(coeffs, vars1, vars2=None)

描述

添加单个或多个新项到表达式。

若参数 vars1 为*Var 类* 对象，则参数 vars2 为*Var 类* 对象或 None ，参数 coeffs 为常数；若参数 vars1 为*VarArray 类* 对象或列表对象，则参数 vars2 为*VarArray 类* 对象、列表对象或 None ，参数 coeffs 为常数或列表对象；若参数 vars1 为字典对象或*tupledict 类* 对象，则参数 vars2 为字典对象、*tupledict 类* 对象或 None ，参数 coeffs 为常数、字典或*tupledict 类* 对象。

参量

coeffs

待添加项的系数。

vars1

待添加项的第一个变量。

vars2

待添加项的第二个变量。可以取值 None，表示添加线性项。

示例

```
# 添加项：2*x + 3y + 2*x*x + 3*x*y 到二次表达式quadexpr中
# 注意：addTerms不支持混合线性项和二次项
quadexpr.addTerms([2.0, 3.0], [x, y])
quadexpr.addTerms([2.0, 3.0], [x, x], [x, y])
```

QuadExpr.addLinExpr()

摘要

addLinExpr(expr, mult=1.0)

描述

添加新的表达到当前表达式中。

参量

expr

待添加表达式或者表达式构建器对象。

mult

待添加表达式的放大系数。可选参量，默认值为 1.0。

示例

```
# 添加线性表达式：2*x + 2*y到二次表达式quadexpr中
quadexpr.addLinExpr(x + y, 2.0)
```

QuadExpr.addQuadExpr()

摘要

addQuadExpr(expr, mult=1.0)

描述

添加新的表达到当前表达式中。

参量

expr

待添加表达式或者表达式构建器对象。

mult

待添加表达式的放大系数。可选参量，默认值为 1.0。

示例

```
# 添加二次表达式：x*x + 2*y到二次表达式quadexpr中
quadexpr.addQuadExpr(x*x + 2*y, 2.0)
```

QuadExpr.clone()

摘要

clone()

描述

创建表达式对象的深拷贝。

示例

```
# 创建二次表达式quadexpr的深拷贝
exprcopy = quadexpr.clone()
```

QuadExpr.reserve()

摘要

reserve(n)

描述

为表达式对象预分配空间。

参量

n

预分配的表达式中项的数目。

示例

```
# 预分配表达式expr中项的数目为100
expr.reserve(100)
```

QuadExpr.remove()

摘要

remove(item)

描述

从表达式中移除指定项。

若参数 item 为常数，则移除指定下标对应的项；否则参数 item 为*Var 类* 对象。

参量

item

常数下标或待移除项相应的变量。

示例

```
# 从二次表达式quadexpr中移除下标为2相应的项
quadexpr.remove(2)
# 从二次表达式quadexpr中移除变量x相应的项
quadexpr.remove(x)
```

23.2.47 PsdExpr 类

PsdExpr 类是杉数求解器中用于构建半定表达式时变量的相关组合操作，提供了以下成员方法：

PsdExpr()

摘要

```
PsdExpr(expr=0.0)
```

描述

创建一个*PsdExpr 类* 对象。

参量

expr

可选参量，默认值为 0.0。可取值为常数、*Var 类* 对象、*LinExpr 类* 对象和*PsdExpr 类* 对象。

示例

```
# 创建一个新的PsdExpr对象，并初始化为：0.0
expr0 = PsdExpr()
# 创建一个PsdExpr对象，并初始化为：2*x + 3*y
expr1 = PsdExpr(2*x + 3*y)
```

PsdExpr.setCoeff()

摘要

```
setCoeff(idx, mat)
```

描述

根据半定变量在表达式中的下标设置其对称矩阵系数。

参量

idx

半定变量在表达式中的下标。起始为 0。

mat

半定变量的新对称矩阵系数。

示例

```
# 设置半定表达式expr第0项的系数为对称矩阵mat
expr.setCoeff(0, mat)
```

PsdExpr.getCoeff()

摘要

getCoeff(idx)

描述

根据半定变量在表达式中的下标获取其对称矩阵系数。

参量

idx

半定变量在表达式中的下标。起始为 0。

示例

```
# 获取半定表达式expr第1项的对称矩阵系数
mat = expr.getCoeff(1)
```

PsdExpr.getPsdVar()

摘要

getPsdVar(idx)

描述

根据半定变量在表达式中的下标获取相应的半定变量，返回一个*PsdVar 类* 对象。

参量

idx

半定变量在表达式中的下标。起始为 0。

示例

```
# 获取半定表达式expr第1项的半定变量
x = expr.getPsdVar(1)
```

PsdExpr.getLinExpr()

摘要

getLinExpr()

描述

获取表达式中的线性表达式。

示例

```
# 获取半定表达式expr的线性表达式
linexpr = expr.getLinExpr()
```

PsdExpr.getConstant()

摘要

getConstant()

描述

获取表达式中的常数项。

示例

```
# 获取半定表达式expr的常数项
constant = expr.getConstant()
```

PsdExpr.getValue()

摘要

getValue()

描述

获取以半定变量和变量的取值计算出的表达式的值。

示例

```
# 获取半定表达式expr的当前值
val = expr.getValue()
```

PsdExpr.getSize()

摘要

getSize()

描述

获取半定表达式中项的个数。

示例

```
# 获取半定表达式expr中元素的个数
exprsize = expr.getSize()
```

PsdExpr.setConstant()

摘要

setConstant(newval)

描述

设置半定表达式的常数项。

参量

newval

待设置常数值。

示例

```
# 设置半定表达式expr的常数项为2.0
expr.setConstant(2.0)
```

PsdExpr.addConstant()

摘要

addConstant(newval)

描述

添加常数到半定表达式。

参量

newval

待添加常数值。

示例

```
# 添加常数2.0到半定表达式expr中
expr.addConstant(2.0)
```

PsdExpr.addTerm()

摘要

addTerm(var, mat)

描述

添加新的半定项到当前半定表达式中。

参量

var

待添加半定项的半定变量。

mat

待添加半定项的对称矩阵。

示例

```
# 添加半定项 C1 * X到半定表达式expr中
expr.addTerm(X, C1)
```

PsdExpr.addTerms()

摘要

addTerms(vars, mats)

描述

添加单个或多个新半定项到半定表达式。

若参数 vars 为*PsdVar 类* 对象，则参数 mats 为*SymMatrix 类* 对象；若参数 vars 为*PsdVarArray 类* 对象或列表对象，则参数 mats 为*SymMatrixArray 类* 对象或列表对象；

参量

vars

待添加半定项的半定变量。

mats

待添加半定项的对称矩阵。

示例

```
# 添加项：C1 * X1 + C2 * X2到半定表达式expr中
expr.addTerms([X1, X2], [C1, C2])
```

PsdExpr.addLinExpr()

摘要

addLinExpr(expr, mult=1.0)

描述

添加新的线性表达式到当前半定表达式中。

参量

expr

待添加线性表达式或者线性表达式构建器对象。

mult

待添加线性表达式的放大系数。可选参量，默认值为 1.0。

示例

```
# 添加线性表达式：2*x + 2*y到半定表达式expr中
expr.addLinExpr(x + y, 2.0)
```

PsdExpr.addPsdExpr()

摘要

addPsdExpr(expr, mult=1.0)

描述

添加新的半定表达式到当前半定表达式中。

参量

expr

待添加半定表达式。

mult

待添加半定表达式的放大系数。可选参量，默认值为 1.0。

示例

```
# 添加半定表达式：C * X到半定表达式expr中
expr.addPsdExpr(C*X)
```

PsdExpr.clone()

摘要

clone()

描述

创建表达式对象的深拷贝。

示例

```
# 创建半定表达式expr的深拷贝
exprcopy = expr.clone()
```

PsdExpr.reserve()

摘要

reserve(n)

描述

为半定表达式对象预分配空间。

参量

n

预分配的半定表达式中项的数目。

示例

```
# 预分配半定表达式expr中项的数目为100
expr.reserve(100)
```

PsdExpr.remove()

摘要

remove(item)

描述

从半定表达式中移除指定项。

若参数 item 为常数，则移除指定下标对应的项；否则参数 item 为*PsdVar 类* 对象。

参量

item

常数下标或待移除项相应的半定变量。

示例

```
# 从半定表达式expr中移除下标为2相应的项
expr.remove(2)
# 从半定表达式expr中移除半定变量x相应的项
expr.remove(x)
```

23.2.48 LmiExpr 类

LmiExpr 类是杉数求解器中用于构建 LMI 约束表达式时变量的相关组合操作，提供了以下成员方法：

LmiExpr()

摘要

LmiExpr(arg1=None, arg2=None)

描述

创建一个*LmiExpr 类* 对象。

参量

arg1 默认值为 None ，可取值为：*Var 类* 对象，或*SymMatrix 类* 对象。

若参数 arg1 为*Var 类* 对象，则参数 arg2 为*SymMatrix 类* 对象。

LmiExpr.setCoeff()

摘要

setCoeff(idx, mat)

描述

设置 LMI 表达式中指定索引 idx 对应项的系数矩阵。

参量

idx

指定的索引值。起始为 0。

mat

待设置变量的新系数对称矩阵，需为*SymMatrix 类* 类对象。

示例

```
# 设置LMI表达式expr第0项的系数为对称矩阵mat
expr.setCoeff(0, mat)
```

LmiExpr.getCoeff()

摘要

getCoeff(idx)

描述

获取 LMI 表达式中指定索引 idx 对应项的系数矩阵。

参量

idx

变量在表达式中的下标。起始为 0。

示例

```
# 获取LMI表达式expr第1项的对称矩阵系数
mat = expr.getCoeff(1)
```

LmiExpr.getVar()

摘要

getVar(idx)

描述

获取 LMI 表达式中指定索引 idx 对应项中的变量。

参量

idx

指定的索引值。起始为 0。

示例

```
# 获取LMI表达式expr第1项的变量
mat = expr.getVar(1)
```

LmiExpr.getConstant()

摘要

getConstant()

描述

获取 LMI 表达式中的常数项对称矩阵。

示例

```
# 获取LMI表达式expr的常数项对称矩阵
constant = expr.getConstant()
```

LmiExpr.getSize()

摘要

getSize()

描述

获取 LMI 表达式中的项数。

示例

```
# 获取表达式expr中项的个数
val = expr.getSize()
```

LmiExpr.setConstant()

摘要

setConstant(mat)

描述

设置 LMI 表达式的常数项对称矩阵。

参量

mat

常数项对应的对称矩阵，需为*SymMatrix 类* 对象。

示例

```
# 设置LMI表达式expr的常数项为对称矩阵D1
expr.setConstant(D1)
```

LmiExpr.addConstant()

摘要

addConstant(mat)

描述

将对称矩阵添加到 LMI 表达式中的常数项中。

参量

mat

加到常数项的对称矩阵对象。

示例

```
# 将对称矩阵D2添加到LMI表达式expr的常数项中
expr.addConstant(D2)
```

LmiExpr.addTerm()

摘要

addTerm(var, mat)

描述

向 LMI 表达式中添加一个新项。

参量

var

新项中的变量。

mat

新项中作为变量系数的对称矩阵，需为*SymMatrix 类* 类对象。

示例

```
# 添加项 x * C1 到LMI表达式expr中
expr.addTerm(x, C1)
```

LmiExpr.addTerms()

摘要

addTerms(vars, mats)

描述

向 LMI 表达式中添加多个新项。

若参数 vars 为*Var 类* 对象，则参数 mats 为*SymMatrix 类* 对象；若参数 vars 为*VarArray 类* 对象或列表对象，则参数 mats 为*SymMatrixArray 类* 对象或列表对象；

参量

vars

待添加新项的变量数组。

mats

待添加新项的对称矩阵数组。

示例

```
# 添加项：x1 * C1 + x2 * C2到LMI表达式expr中
expr.addTerms([x1, x2], [C1, C2])
```

LmiExpr.addLmiExpr()

摘要

addLmiExpr(expr, mult=1.0)

描述

向当前的 LMI 表达式中，再添加一个新的 LMI 表达式。

参量

expr

待添加的 LMI 表达式。

mult

可选的系数倍数，为常数，默认值为 1.0。

示例

```
# 添加半定表达式：2 * x * C到半定表达式expr中
expr.addLmiExpr(x * C, 2.0)
```

LmiExpr.clone()

摘要

clone()

描述

创建 LMI 表达式对象的深拷贝。

示例

```
# 创建LMI表达式expr的深拷贝
exprcopy = expr.clone()
```

LmiExpr.reserve()

摘要

reserve(n)

描述

为 LMI 表达式对象预分配空间。

参量

n

预分配的 LMI 表达式中项的数目。

示例

```
# 预分配LMI表达式expr中项的数目为100
expr.reserve(100)
```

LmiExpr.remove()

摘要

remove(item)

描述

从 LMI 表达式中移除指定项。

若参数 item 为常数，则移除指定下标对应的项；否则参数 item 为*Var 类* 对象。

参量

item

常数下标或待移除项相应的变量。

示例

```
# 从LMI表达式expr中移除下标为2相应的项
expr.remove(2)
# 从LMI表达式expr中移除变量x相应的项
expr.remove(x)
```

23.2.49 CallbackBase 类

CallbackBase 类是对杉数求解器 COPT 的 Callback 相关操作的封装。CallbackBase 是一个抽象类，用户需要通过实现函数*CallbackBase.callback()* 来创建一个实例，该实例会作为参数传入*Model.setCallback()* 。CallbackBase 类提供以下可以被继承的成员方法：

CallbackBase.where()

摘要

where()

描述

获取回调函数可能的触发点。

返回值

返回一个整数值。

CallbackBase.callback()

摘要

callback()

描述

回调函数，为纯虚函数。用户覆盖实现求解进程中需要获取的信息或需执行的操作。

示例

```
class CoptCallback(CallbackBase):
    def __init__(self):
        super().__init__()
    def callback(self):
        # Get the objective value when finding a feasible MIP solution
        if self.where() == COPT.CBCONTEXT_MIPSOL:
            db = self.getInfo(COPT.CBInfo.MipCandObj)
```

CallbackBase.interrupt()

摘要

interrupt()

描述

中断回调中的求解进程。

CallbackBase.addUserCut()

摘要

addUserCut(lhs, sense = None, rhs = None)

描述

向模型中添加一个割平面。

参量

lhs

割平面约束的左端项。

可取值为*Var 类* 对象，*LinExpr 类* 对象，或者约束构建器（*ConstrBuilder 类* 对象）。

sense

割平面的约束类型。

可选参量，默认为 None 。可取值为：LESS_EQUAL, GREATER_EQUAL, EQUAL 和 FREE 。

通过*CallbackBase 类* 添加的割平面，仅支持单边约束。

`rhs`

割平面约束的右端项。

可选参量，默认为 None 。可取值为常数、*Var 类* 对象，或者*LinExpr 类* 对象。

示例

```
self.addUserCut(x+y <= 1)
```

CallbackBase.addUserCuts()

摘要

`addUserCuts(generator)`

描述

向模型中批量添加多个割平面。

参量

`generator`

一组割平面生成器。

可取值为一组线性约束构建器*ConstrBuilderArray 类* 对象，或多维线性约束生成器*MConstrBuilder 类* 对象。

示例

```
self.addUserCuts(x[i]+y[i] <= 1 for i in range(10))
```

CallbackBase.addLazyConstr()

摘要

`addLazyConstr(lhs, sense = None, rhs = None)`

描述

向模型中添加一个惰性约束。

参量

`lhs`

惰性约束的左端项。

可取值为*Var 类* 对象，*LinExpr 类* 对象，或者约束构建器（*ConstrBuilder 类* 对象）。

`sense`

惰性约束的约束类型。

可选参量，默认为 None 。可取值为：LESS_EQUAL, GREATER_EQUAL, EQUAL 和 FREE 。

通过*CallbackBase 类* 添加的惰性约束，仅支持单边约束。

rhs

惰性约束的右端项。

可选参量，默认为 None 。可取值为常数、*Var 类* 对象，或者*LinExpr 类* 对象。

示例

```
self.addLazyConstr(x+y <= 1)
```

CallbackBase.addLazyConstrs()

摘要

addLazyConstrs(generator)

描述

向模型中批量添加多个惰性约束。

参量

generator

一组惰性约束生成器。

可取值为一组线性约束构建器*ConstrBuilderArray 类* 对象，或多维线性约束生成器*MConstrBuilder 类* 对象。

示例

```
self.addLazyConstrs(x[i]+y[i] <= 1 for i in range(10))
```

CallbackBase.getInfo()

摘要

getInfo(cbinfo)

描述

在 MIP 求解过程中，根据回调触发条件，获取指定的模型信息值。

参量

cbinfo

指定的信息名称。可取值详见*回调信息*

返回值

返回一个常数（整数型或双精度浮点型）。

示例

```
db = self.getInfo(COPT.CBInfo.BestBnd)
```

CallbackBase.getRelaxSol()

摘要

getRelaxSol(vars)

描述

在 MIP 求解过程中，获取当前节点线性松弛解中指定变量的值。

注意：此方法仅适用于 CallbackBase.where() == COPT.CBCONTEXT_MIPRELAX（即找到 LP 线性松弛解时）。

参量

vars

指定的变量。

返回值

若参数 args 为*Var 类* 对象，则返回指定变量的信息值常数；

若参数 args 为列表或*VarArray 类* 对象，则返回指定变量的信息值组成的一个列表对象；

若参数 args 为字典或*tupledict 类* 对象，则返回*tupledict 类* 对象（键为指定变量的下标，值为指定变量的信息值）；

若参数 args 为 None，则返回全部变量当前线性松弛解的信息值。

示例

```
vals = self.getRelaxSol(vars)
```

CallbackBase.getIncumbent()

摘要

getIncumbent(vars)

描述

在 MIP 求解过程中，获取当前最优可行解中指定变量的值。

参量

vars

指定的变量。

返回值

若参数 args 为*Var 类*对象，则返回指定变量的信息值常数；

若参数 args 为列表或*VarArray 类*对象，则返回指定变量的信息值组成的一个列表对象；

若参数 args 为字典或*tupledict 类*对象，则返回*tupledict 类*对象（键为指定变量的下标，值为指定变量的信息值）；

若参数 args 为 None，则返回全部变量当前最优可行解的信息值。

示例

```
vals = self.getIncumbent(vars)
```

CallbackBase.getSolution()

摘要

getSolution(vars)

描述

在 MIP 求解过程中，获取当前可行解中指定变量的值。

注意：此方法仅适用于 CallbackBase.where() == COPT.CBCONTEXT_MIPSOL（即找到 MIP 可行解时）。

参量

vars

指定的变量。

返回值

若参数 args 为*Var 类*对象，则返回指定变量的信息值常数；

若参数 args 为列表或*VarArray 类*对象，则返回指定变量的信息值组成的一个列表对象；

若参数 args 为字典或*tupledict 类*对象，则返回*tupledict 类*对象（键为指定变量的下标，值为指定变量的信息值）；

若参数 args 为 None，则返回全部变量当前可行解的信息值。

示例

```
vals = self.getSolution(vars)
```

CallbackBase.setSolution()

摘要

setSolution(vars, vals)

描述

在 MIP 求解过程中，为指定变量设置自定义的可行解，（可以是用户通过任意方式找到的可行解，例如通过启发式算法）。

若参数 vars 为*Var 类* 对象，则参数 vals 为常量；

若参数 vars 为字典或*tupledict 类* 对象，则参数 vals 可为常量、字典或*tupledict 类* 对象；

若参数 vars 为列表或 *VarArray 类* 对象，则参数 vals 可为常量或列表对象。

参量

vars

指定的变量。

vals

自定义解的值。

示例

```
self.setSolution(x, 1)
```

CallbackBase.loadSolution()

摘要

loadSolution()

描述

将当前自定义的解加载入模型中。

注意：当前仅支持完整的自定义解。

示例

```
self.loadSolution()
```

23.2.50 GenConstrX 类

在 Model 类中，通过 addGenConstrXXX（如：addGenConstrMax ）添加的约束，会返回一个 GenConstrX 对象。

GenConstrX.getAttr()

摘要

getAttr(attrname)

描述

获取 GenConstrX 类对象的属性值，支持获取 GenConstrX 类对象的类型和名称。

示例

```
# Get the name of con_max
con_max.getAttr("name")
# Get the type of con_max
con_max.getAttr("type")
```

GenConstrX.setAttr()

摘要

setAttr(attrname)

描述

设置 GenConstrX 类对象的属性值，支持设置 GenConstrX 类对象的名称。

示例

```
# Set the name of con_max
con_max.setAttr("name")
```

23.2.51 CoptError 类

CoptError 类是杉数求解器的错误处理相关操作的封装。当方法调用对应的杉数求解器底层接口发生错误时，则抛出 CoptError 类的异常，提供了以下属性值访问相应的错误信息：

- CoptError.retcode

  错误值代码。

- CoptError.message

  错误值信息。

23.3 辅助函数与工具类

辅助函数与工具类基于 Python 的基本数据类型进行封装，提供了易用的数据类型，便于快速构建复杂的优化模型。本节将阐述其功能与使用方法。

23.3.1 辅助函数

multidict()

摘要

`multidict(data)`

描述

将输入的字典对象拆分为键与多个字典对象并返回。

参量

`data`

待拆分字典对象，该字典对象中每个键映射 n 个值。

示例

```
keys, dict1, dict2 = multidict({
  "hello": [0, 1],
  "world": [2, 3]})
```

quicksum()

摘要

`quicksum(data)`

描述

快速构建表达式，返回一个*LinExpr 类* 对象。

参量

`data`

生成表达式待加项。

示例

```
expr = quicksum(m.getVars())
```

23.3.2 tuplelist 类

tuplelist 类是基于 Python 列表类的封装，提供了以下成员方法：

tuplelist()

摘要

tuplelist(list)

描述

创建并返回一个*tuplelist 类* 对象。

参量

list

Python 列表对象。

示例

```
tl = tuplelist([(0, 1), (1, 2)])
tl = tuplelist([('a', 'b'), ('b', 'c')])
```

tuplelist.add()

摘要

add(item)

描述

向*tuplelist 类* 对象中添加项。

参量

item

待添加项，可取值为 Python 元组对象。

示例

```
tl = tuplelist([(0, 1), (1, 2)])
tl.add((2, 3))
```

tuplelist.select()

摘要

select(pattern)

描述

根据指定的模式筛选得到符合条件的项，返回一个*tuplelist 类* 对象。

参量

pattern

指定的匹配模式。

示例

```
tl = tuplelist([(0, 1), (0, 2), (1, 2)])
tl.select(0, '*')
```

23.3.3 tupledict 类

tupledict 类是基于 Python 字典类的封装，提供了以下成员方法：

tupledict()

摘要

tupledict(args, kwargs)

描述

创建并返回一个*tupledict 类* 对象。

参量

args

位置参量。

kwargs

命名参量。

示例

```
d = tupledict([(0, "hello"), (1, "world")])
```

tupledict.select()

摘要

select(pattern)

描述

根据指定的模式筛选得到符合条件的项，返回一个*tupledict 类* 对象。

参量

pattern

指定的匹配模式。

示例

```
d = tupledict([(0, "hello"), (1, "world")])
d.select()
```

tupledict.sum()

摘要

sum(pattern)

描述

根据指定的模式筛选累加项，返回一个*LinExpr 类* 对象。

参量

pattern

指定的匹配模式。

示例

```
expr = x.sum()
```

tupledict.prod()

摘要

prod(coeff, pattern)

描述

根据指定的模式筛选，并与乘积系数累乘项，返回一个*LinExpr 类* 对象。

参量

coeff

乘积系数。可取值为字典或*tupledict 类* 对象。

pattern

指定的匹配模式。

示例

```
coeff = dict([(1, 0.1), (2, 0.2)])
expr  = x.prod(coeff)
```

23.3.4 ProbBuffer 类

ProbBuffer 类是字符流缓冲区的封装，提供了以下成员方法：

ProbBuffer()

摘要

ProbBuffer(buff)

描述

创建并返回一个*ProbBuffer 类* 对象。

参量

buff

缓冲区大小，默认为 None，即缓冲区大小为 0。

示例

```
# 创建大小为100的字符流缓冲区
buff = ProbBuffer(100)
```

ProbBuffer.getData()

摘要

getData()

描述

获取字符流缓冲区中的内容。

示例

```
# 打印字符流缓冲区buff中的内容
print(buff.getData())
```

ProbBuffer.getSize()

摘要

getSize()

描述

获取字符流缓冲区的大小。

示例

```
# 获取字符流缓冲区buff的大小
print(buff.getSize())
```

ProbBuffer.resize()

摘要

resize(sz)

描述

调整字符流缓冲区的大小。

参量

sz

缓冲区新的大小。

示例

```
# 调整字符流缓冲区buff的大小
buff.resize(100)
```

第 24 章　C++ API 参考手册

杉数优化求解器提供了 C++ API 库函数，适用于基于 COPT 的 C++ 应用程序。本章节描述了 C++ 方面的常量、类、参数、属性。

24.1 常量

所有的 C++ 常量都和 C API 里描述的一样。请参看*C API 参考手册：常量章节* 。

24.2 属性

所有的 C++ 属性都和 C API 里描述的一样。请参看*C API 参考手册：属性章节* 。

在 C++ API 中，通过指定属性名称获取属性取值。提供获取属性取值的 2 个函数如下，具体请参考*C++ Model 类* 。

- `Model::GetIntAttr()`：获取整数属性取值
- `Model::GetDblAttr()`：获取浮点型属性取值

24.3 参数

所有的 C++ 参数都和 C API 里描述的一样。请参看*C API 参考手册：参数章节* 。

在 C++ API 中，通过指定参数名称获取和设置参数取值。提供的函数如下，具体请参考*C++ Model 类* 。

- 获取指定参数的详细信息（当前值/最大值/最小值）：`Model::GetParamInfo()`
- 获取指定整数型/浮点型参数当前取值：`Model::GetIntParam()` / `Model::GetDblParam()`
- 设置指定整数型/浮点型参数值：`Model::SetIntParam()` / `Model::SetDblParam()`

24.4 C++ 优化建模类

本章节详细描述杉数优化求解器 C++ 接口的优化建模类，方便用户在快速构建复杂场景下的优化模型时对其功能和使用的查询。

24.4.1 Envr 类

创建求解环境对象是每个求解过程中必不可少的第一步。而每个求解模型都和一个 Envr 类关联。用户必须首先创建一个求解环境，才能在此基础上创建一个或者多个求解模型。

Envr::Envr()

COPT Envr 类的构造函数。

概要

```
Envr()
```

Envr::Envr()

COPT Envr 类的构造函数。

概要

```
Envr(const char *szLicDir)
```

参量

`szLicDir`: 用户指定的路径，包含本地授权文档或者客户端配置文件。

Envr::Envr()

COPT Envr 类的构造函数。

概要

```
Envr(const EnvrConfig &config)
```

参量

`config`: COPT Envr 配置类，包含远程连接的设置。

Envr::Close()

关闭远程连接。之前获得的远程授权失效，对当前环境类下创建的全部问题立即生效。

概要

```
void Close()
```

Envr::CreateModel()

创建 COPT 模型。

概要

```
Model CreateModel(const char *szName)
```

参量

`szName`: 自定义的模型名称。

返回值

COPT 优化模型。

24.4.2 EnvrConfig 类

如果用户通过连接远程服务的方式启动杉数优化求解器，可以创建环境配置类来设置 COPT 作为客户端的配置。

EnvrConfig::EnvrConfig()

COPT 环境配置类的构造函数。

概要

```
EnvrConfig()
```

EnvrConfig::Set()

设置环境配置类里的内容。

概要

```
void Set(const char *szName, const char *szValue)
```

参量

`szName`: 配置的关键词。

`szValue`: 配置的内容。

24.4.3 Model 类

Model 类是杉数优化求解器模型相关操作的封装，提供了以下成员方法：

Model::AddCone()

向模型中增加一个给定大小锥约束。

概要

```
Cone AddCone(
    int dim,
    int type,
    char *pvtype,
    const char *szPrefix)
```

参量

`dim`: 锥约束的维度。

`type`: 锥约束的类型。

`pvtype`: 锥约束中变量的类型。

`szPrefix`: 锥约束中变量的名称前缀。

返回值

新的锥约束。

Model::AddCone()

向模型中增加一个锥约束。

概要

```
Cone AddCone(const ConeBuilder &builder)
```

参量

`builder`: 锥约束生成器。

返回值

新的锥约束。

Model::AddCone()

向模型中增加一个锥约束。

概要

```
Cone AddCone(const VarArray &vars, int type)
```

参量

vars: 参与锥约束的变量。

type: 锥约束的类型。

返回值

新的锥约束。

Model::AddConstr()

向模型中增加一个线性约束。

概要

```
Constraint AddConstr(
    const Expr &expr,
    char sense,
    double rhs,
    const char *szName)
```

参量

expr: 新的约束表达式。

sense: 约束的类型。

rhs: 新约束的右侧值。

szName: 可选，新约束的名称。

返回值

新约束。

Model::AddConstr()

向模型中增加一个线性约束。

概要

```
Constraint AddConstr(
    const Expr &lhs,
    char sense,
```

```
    const Expr &rhs,
    const char *szName)
```

参量

lhs: 新约束的左侧值。

sense: 约束的类型。

rhs: 新约束的右侧值。

szName: 可选，新约束的名称。

返回值

新约束。

Model::AddConstr()

向模型中增加一个线性约束。

概要

```
Constraint AddConstr(
    const Expr &expr,
    double lb,
    double ub,
    const char *szName)
```

参量

expr: 新的约束表达式。

lb: 约束的下界。

ub: 约束的上界。

szName: 可选，新约束的名称。

返回值

新约束。

Model::AddConstr()

向模型中增加一个线性约束。

概要

```
Constraint AddConstr(const ConstrBuilder &builder, const char
*szName)
```

参量

builder: 新约束生成器。

szName: 可选，新约束的名称。

返回值

新约束。

Model::AddConstrs()

向模型中增加线性约束。

概要

```
ConstrArray AddConstrs(
    int count,
    char *pSense,
    double *pRhs,
    const char *szPrefix)
```

参量

count: 添加的线性约束数目。

pSense: 约束类型，而不是范围类型。

pRhs: 新约束的右侧值。

szPrefix: 新约束的名称前缀。

返回值

新约束构成的 ConstrArray 类。

Model::AddConstrs()

向模型中增加线性约束。

概要

```
ConstrArray AddConstrs(
    int count,
    double *pLower,
    double *pUpper,
    const char *szPrefix)
```

参量

count: 添加的线性约束数目。

pLower: 新约束的下界。

pUpper: 新约束的上界。

`szPrefix`: 新约束的名称前缀。

返回值

新约束构成的 ConstrArray 类。

Model::AddConstrs()

向模型中增加线性约束。

概要

```
ConstrArray AddConstrs(
    int count,
    double *pLower,
    double *pUpper,
    const char *szNames,
    size_t len)
```

参量

`count`: 添加的线性约束数目。

`pLower`: 新约束的下界。

`pUpper`: 新约束的上界。

`szNames`: 新约束的名称缓冲区。

`len`: 名称缓冲区的长度。

返回值

新约束构成的 ConstrArray 类。

Model::AddConstrs()

向模型中增加线性约束。

概要

```
ConstrArray AddConstrs(const ConstrBuilderArray &builders, const
char *szPrefix)
```

参量

`builders`: 线性约束生成器。

`szPrefix`: 新约束的名称前缀。

返回值

新约束构成的 ConstrArray 类。

Model::AddConstrs()

向模型中增加线性约束。

概要

```
ConstrArray AddConstrs(
    const ConstrBuilderArray &builders,
    const char *szNames,
    size_t len)
```

参量

builders: 线性约束生成器。

szNames: 新约束的名称缓冲区。

len: 名称缓冲区的长度。

返回值

新约束构成的 ConstrArray 类。

Model::AddDenseMat()

向模型中增加一个密致对称矩阵。

概要

```
SymMatrix AddDenseMat(
    int dim,
    double *pVals,
    int len)
```

参量

dim: 密致对称矩阵的维度。

pVals: 非零元值数组。按列次序填充非零元，到数组长度或者对称矩阵最大长度位置。

len: 数组长度。

返回值

新对称矩阵对象。

Model::AddDenseMat()

向模型中增加一个密致对称矩阵。

概要

```
SymMatrix AddDenseMat(int dim, double val)
```

参量

`dim`: 密致对称矩阵的维度。

`val`: 同一个非零元值，用来填充对称矩阵。

返回值

新对称矩阵对象。

Model::AddDiagMat()

向模型中增加一个对角矩阵。

概要

```
SymMatrix AddDiagMat(int dim, double val)
```

参量

`dim`: 对角矩阵的维度。

`val`: 同一个非零元值，用来填充对角元素。

返回值

新对角矩阵对象。

Model::AddDiagMat()

向模型中增加一个对角矩阵。

概要

```
SymMatrix AddDiagMat(
    int dim,
    double *pVals,
    int len)
```

参量

`dim`: 对角矩阵的维度。

`pVals`: 双精度值数组，用来填充对角元素。

`len`: 值数组的长度。

返回值

新对角矩阵对象。

Model::AddDiagMat()

向模型中增加一个对角矩阵。

概要

```
SymMatrix AddDiagMat(
    int dim,
    double val,
    int offset)
```

参量

dim: 对角矩阵的维度。

val: 同一个非零元值，用来填充对角元素。

offset: 相对于标准对角线的平移量。

返回值

新对角矩阵对象。

Model::AddDiagMat()

向模型中增加一个对角矩阵。

概要

```
SymMatrix AddDiagMat(
    int dim,
    double *pVals,
    int len,
    int offset)
```

参量

dim: 对角矩阵的维度。

pVals: 双精度值数组，用来填充对角元素。

len: 值数组的长度。

offset: 相对于标准对角线的平移量。

返回值

新对角矩阵对象。

Model::AddEyeMat()

向模型中增加一个单位矩阵。

概要

```
SymMatrix AddEyeMat(int dim)
```

参量

`dim`: 单位矩阵的维度。

返回值

新单位矩阵对象。

Model::AddGenConstrIndicator()

向模型中增加一个类型指示型的一般约束（GenConstr）。

概要

```
GenConstr AddGenConstrIndicator(const GenConstrBuilder &builder)
```

参量

`builder`: 一般约束（GenConstr）生成器。

返回值

类型指示型的新一般约束（GenConstr）。

Model::AddGenConstrIndicator()

向模型中增加一个类型指示型的一般约束（GenConstr）。

概要

```
GenConstr AddGenConstrIndicator(
    Var binVar,
    int binVal,
    const ConstrBuilder &constr)
```

参量

`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 或 1）。

`constr`: 线性约束生成器。

返回值

类型指示型的新一般约束（GenConstr）。

Model::AddGenConstrIndicator()

向模型中增加一个类型指示型的一般约束（GenConstr）。

概要

```
GenConstr AddGenConstrIndicator(
    Var binVar,
    int binVal,
    const Expr &expr,
    char sense,
    double rhs)
```

参量

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 或 1）。

expr: 新的线性约束表达式。

sense: 新的线性约束类型。

rhs: 新的线性约束右侧值。

返回值

类型指示型的新一般约束（GenConstr）。

Model::AddLazyConstr()

向模型中增加一个惰性约束。

概要

```
void AddLazyConstr(
    const Expr &lhs,
    char sense,
    double rhs,
    const char *szName)
```

参量

lhs: 惰性约束表达式。

sense: 惰性约束的类型。

rhs: 惰性约束的右侧值。

szName: 可选，新惰性约束的名称。

Model::AddLazyConstr()

向模型中增加一个惰性约束。

概要

```
void AddLazyConstr(
    const Expr &lhs,
    char sense,
    const Expr &rhs,
    const char *szName)
```

参量

lhs: 惰性约束的左侧表达式。

sense: 惰性约束的类型。

rhs: 惰性约束的右侧表达式。

szName: 可选，新惰性约束的名称。

Model::AddLazyConstr()

向模型中增加一个惰性约束。

概要

```
void AddLazyConstr(const ConstrBuilder &builder, const char *szName)
```

参量

builder: 惰性约束生成器。

szName: 可选，新惰性约束的名称。

Model::AddLazyConstrs()

向模型中增加多个惰性约束。

概要

```
void AddLazyConstrs(const ConstrBuilderArray &builders, const char
*szPrefix)
```

参量

builders: 一组惰性约束生成器。

szPrefix: 新惰性约束名称的前缀。

Model::AddLmiConstr()

向模型中添加一个 LMI 约束。

概要

```
LmiConstraint AddLmiConstr(const LmiExpr &expr, const char *szName)
```

参量

expr: 新的 LMI 约束表达式。

szName: 可选，新 LMI 约束的名称。

返回值

新的 LMI 约束对象。

Model::AddOnesMat()

向模型中增加一个用非零元 1 填充的密致对称矩阵。

概要

```
SymMatrix AddOnesMat(int dim)
```

参量

dim: 密致对称矩阵的维度。

返回值

新对称矩阵对象。

Model::AddPsdConstr()

向模型中增加一个半定约束。

概要

```
PsdConstraint AddPsdConstr(
    const PsdExpr &expr,
    char sense,
    double rhs,
    const char *szName)
```

参量

expr: 新的半定约束表达式。

sense: 半定约束的类型。

rhs: 新半定约束的右侧值。

szName: 可选，新半定约束的名称。

返回值

新半定约束对象。

Model::AddPsdConstr()

向模型中增加一个半定约束。

概要

```
PsdConstraint AddPsdConstr(
    const PsdExpr &expr,
    double lb,
    double ub,
    const char *szName)
```

参量

`expr`: 新的半定约束表达式。

`lb`: 半定约束的下界。

`ub`: 半定约束的上界。

`szName`: 可选，新半定约束的名称。

返回值

新半定约束。

Model::AddPsdConstr()

向模型中增加一个半定约束。

概要

```
PsdConstraint AddPsdConstr(
    const PsdExpr &lhs,
    char sense,
    const PsdExpr &rhs,
    const char *szName)
```

参量

`lhs`: 新半定约束的左侧表达式。

`sense`: 半定约束的类型。

`rhs`: 新半定约束的右侧表达式。

`szName`: 可选，新半定约束的名称。

返回值

新半定约束对象。

Model::AddPsdConstr()

向模型中增加一个半定约束。

概要

```
PsdConstraint AddPsdConstr(const PsdConstrBuilder &builder, const
char *szName)
```

参量

builder: 新半定约束生成器。

szName: 可选，新半定约束的名称。

返回值

新半定约束对象。

Model::AddPsdVar()

向模型中增加半定变量。

概要

```
PsdVar AddPsdVar(int dim, const char *szName)
```

参量

dim: 新半定变量的维度。

szName: 新变量的名称。

返回值

新半定变量对象。

Model::AddPsdVars()

向模型中添加一些半定变量。

概要

```
PsdVarArray AddPsdVars(
    int count,
    int *pDim,
    const char *szPrefix)
```

参量

count: 新半定变量的数目。

pDim: 整数数组，包含了半定变量的维度。

szPrefix: 新半定变量的名称前缀。

返回值

新添加的半定变量数组。

Model::AddPsdVars()

向模型中添加一些半定变量。

概要

```
PsdVarArray AddPsdVars(
    int count,
    int *pDim,
    const char *szNames,
    size_t len)
```

参量

count: 新半定变量的数目。

pDim: 整数数组，包含了半定变量的维度。

szNames: 新半定变量的命名字符串缓冲区。

len: 命名字符串缓冲区的长度。

返回值

新添加的半定变量数组。

Model::AddQConstr()

向模型中增加一个二次约束。

概要

```
QConstraint AddQConstr(
    const QuadExpr &expr,
    char sense,
    double rhs,
    const char *szName)
```

参量

expr: 新的二次约束表达式。

sense: 约束的类型。

rhs: 新约束的右侧值。

szName: 可选，新二次约束的名称。

返回值

新二次约束对象。

Model::AddQConstr()

向模型中增加一个二次约束。

概要

```
QConstraint AddQConstr(
    const QuadExpr &lhs,
    char sense,
    const QuadExpr &rhs,
    const char *szName)
```

参量

lhs: 新二次约束的左侧表达式。

sense: 二次约束的类型。

rhs: 新二次约束的右侧表达式。

szName: 可选，新二次约束的名称。

返回值

新二次约束对象。

Model::AddQConstr()

向模型中增加一个二次约束。

概要

```
QConstraint AddQConstr(const QConstrBuilder &builder, const char
*szName)
```

参量

builder: 新二次约束生成器。

szName: 可选，新二次约束的名称。

返回值

新二次约束对象。

Model::AddSos()

向模型中增加一个 SOS 约束。

概要

```
Sos AddSos(const SosBuilder &builder)
```

参量

builder: SOS 约束生成器。

返回值

新 SOS 约束。

Model::AddSos()

向模型中增加一个 SOS 约束。

概要

```
Sos AddSos(
    const VarArray &vars,
    const double *pWeights,
    int type)
```

参量

vars: 参与 SOS 约束的变量。

pWeights: 可选，参与 SOS 约束变量的权重。

type: SOS 约束的类型。

返回值

新 SOS 约束。

Model::AddSparseMat()

向模型中增加一个稀疏对称矩阵。

概要

```
SymMatrix AddSparseMat(
    int dim,
    int nElems,
    int *pRows,
    int *pCols,
    double *pVals)
```

参量

dim: 稀疏对称矩阵的维度。

nElems: 稀疏对称矩阵中的非零元个数。

pRows: 整数数组，保存了非零元的行号。

pCols: 整数数组，保存了非零元的列号。

pVals: 非零元值数组。

返回值

新对称矩阵对象。

Model::AddSymMat()

根据给定的对称矩阵表达式，向模型中增加一个对称矩阵。

概要

```
SymMatrix AddSymMat(const SymMatExpr &expr)
```

参量

expr: 对称矩阵表达式对象。

返回值

结果对称矩阵对象。

Model::AddUserCut()

向模型中增加一个割平面。

概要

```
void AddUserCut(
    const Expr &lhs,
    char sense,
    double rhs,
    const char *szName)
```

参量

lhs: 割平面表达式。

sense: 割平面的类型。

rhs: 割平面的右侧值。

szName: 可选，新割平面的名称。

Model::AddUserCut()

向模型中增加一个割平面。

概要

```
void AddUserCut(
    const Expr &lhs,
    char sense,
    const Expr &rhs,
    const char *szName)
```

参量

lhs: 割平面的左侧表达式。

sense: 割平面的类型。

rhs: 割平面的右侧表达式。

szName: 可选，新割平面的名称。

Model::AddUserCut()

向模型中增加一个割平面。

概要

```
void AddUserCut(const ConstrBuilder &builder, const char *szName)
```

参量

builder: 割平面生成器。

szName: 可选，新割平面的名称。

Model::AddUserCuts()

向模型中增加多个割平面。

概要

```
void AddUserCuts(const ConstrBuilderArray &builders, const char
*szPrefix)
```

参量

builders: 一组割平面生成器。

szPrefix: 新割平面名称的前缀。

Model::AddVar()

向模型中增加一个变量。

概要

```
Var AddVar(
    double lb,
    double ub,
    double obj,
    char vtype,
    const char *szName)
```

参量

lb: 新变量的下界。

ub: 新变量的上界。

obj: 新变量在目标函数里的系数。

vtype: 新变量的类型。

szName: 新变量的名称。

返回值

新变量。

Model::AddVar()

向模型中增加一个变量。

概要

```
Var AddVar(
    double lb,
    double ub,
    double obj,
    char vtype,
    const Column &col,
    const char *szName)
```

参量

lb: 新变量的下界。

ub: 新变量的上界。

obj: 新变量在目标函数里的系数。

vtype: 新变量的类型。

col: 和添加的变量相关联的列对象。

szName: 新变量的名称。

返回值

新变量。

Model::AddVars()

向模型中增加变量。

概要

```
VarArray AddVars(
    int count,
    char vtype,
    const char *szPrefix)
```

参量

count: 添加变量的数量。

vtype: 新变量的类型。

szPrefix: 新变量的名称前缀。

返回值

新变量构成的 VarArray 类。

Model::AddVars()

向模型中增加变量。

概要

```
VarArray AddVars(
    int count,
    char vtype,
    const char *szNames,
    size_t len)
```

参量

count: 添加变量的数量。

vtype: 新变量的类型。

szNames: 新变量的名称缓冲区。

len: 名称缓冲区的长度。

返回值

新变量构成的 VarArray 类。

Model::AddVars()

向模型中增加变量。

概要

```
VarArray AddVars(
    int count,
    double lb,
    double ub,
    double obj,
    char vtype,
    const char *szPrefix)
```

参量

`count`: 添加变量的数量。

`lb`: 新变量的下界。

`ub`: 新变量的上界。

`obj`: 新变量在目标函数里的系数。

`vtype`: 新变量的类型。

`szPrefix`: 新变量的名称前缀。

返回值

新变量构成的 VarArray 类。

Model::AddVars()

向模型中增加变量。

概要

```
VarArray AddVars(
    int count,
    double lb,
    double ub,
    double obj,
    char vtype,
    const char *szNames,
```

```
    size_t len)
```

参量

`count`: 添加变量的数量。

`lb`: 新变量的下界。

`ub`: 新变量的上界。

`obj`: 新变量在目标函数里的系数。

`vtype`: 新变量的类型。

`szNames`: 新变量的名称缓冲区。

`len`: 名称缓冲区的长度。

返回值

新变量构成的 VarArray 类。

Model::AddVars()

向模型中增加变量。

概要

```
VarArray AddVars(
    int count,
    double *plb,
    double *pub,
    double *pobj,
    char *pvtype,
    const char *szPrefix)
```

参量

`count`: 添加变量的数量。

`plb`: 新变量的下界, 如果为空，下界为 0。

`pub`: 新变量的上界. 如果为空，上界为正无穷或者 1 对于二进制变量。

`pobj`: 新变量在目标函数中的系数，如果为空，则为 0.0。

`pvtype`: 新变量的类型，如果为空，则为连续变量。

`szPrefix`: 新变量的名称前缀。

返回值

新变量构成的 VarArray 类。

Model::AddVars()

向模型中增加变量。

概要

```
VarArray AddVars(
    int count,
    double *plb,
    double *pub,
    double *pobj,
    char *pvtype,
    const char *szNames,
    size_t len)
```

参量

`count`: 添加变量的数量。

`plb`: 新变量的下界，如果为空，下界为 0。

`pub`: 新变量的上界，如果为空，上界为正无穷或者 1 对于二进制变量。

`pobj`: 新变量在目标函数中的系数，如果为空，则为 0。

`pvtype`: 新变量的类型，如果为空，则为连续变量。

`szNames`: 新变量的名称缓冲区。

`len`: 名称缓冲区的长度。

返回值

新变量构成的 VarArray 类。

Model::AddVars()

向模型中增加变量。

概要

```
VarArray AddVars(
    int count,
    double *plb,
    double *pub,
    double *pobj,
    char *pvtype,
    const ColumnArray &cols,
```

```
    const char *szPrefix)
```

参量

`count`: 添加变量的数量。

`plb`: 新变量的下界，如果为空，下界为 0。

`pub`: 新变量的上界. 如果为空，上界为正无穷或者 1 对于二进制变量。

`pobj`: 新变量在目标函数中的系数，如果为空，则为 0。

`pvtype`: 新变量的类型，如果为空，则为连续变量。

`cols`: 列对象构成的 ColumnArray 类，用于指定每个新变量所属的一组约束。

`szPrefix`: 新变量的名称前缀。

返回值

新变量构成的 VarArray 类。

Model::Clear()

删除所有设置以及问题本身。

概要

```
void Clear()
```

Model::Clone()

深度复制 COPT 模型。

概要

```
Model Clone()
```

返回值

新的模型对象。

Model::ComputeIIS()

计算不可行模型的 IIS。

概要

```
void ComputeIIS()
```

Model::DelPsdObj()

删除模型目标函数的半定部分（保留线性部分）。

概要

```
void DelPsdObj()
```

Model::DelQuadObj()

删除模型目标函数的二次部分（保留线性部分）。

概要

```
void DelQuadObj()
```

Model::FeasRelax()

计算不可行模型的可行化松弛。

概要

```
void FeasRelax(
    VarArray &vars,
    double *pColLowPen,
    double *pColUppPen,
    ConstrArray &cons,
    double *pRowBndPen,
    double *pRowUppPen)
```

参量

`vars`: 变量构成的 VarArray 类对象。

`pColLowPen`: 变量下界的惩罚系数。

`pColUppPen`: 变量上界的惩罚系数。

`cons`: 约束构成的 ConstrArray 类对象。

`pRowBndPen`: 约束右端项的惩罚系数。

`pRowUppPen`: 约束上界的惩罚系数。

Model::FeasRelax()

计算不可行模型的可行化松弛。

概要

```
void FeasRelax(int ifRelaxVars, int ifRelaxCons)
```

参量

ifRelaxVars: 是否松弛变量。

ifRelaxCons: 是否松弛约束。

Model::Get()

查询与指定变量相关的双精度型信息的值。

概要

```
int Get(
    const char *szName,
    const VarArray &vars,
    double *pOut)
```

参量

szName: 双精度型信息名称。

vars: 指定变量数组。

pOut: 输出双精度型数组，保存了信息的值。

返回值

被查询的有效变量数目。如果出错，返回-1。

Model::Get()

查询与指定约束相关的双精度型信息的值。

概要

```
int Get(
    const char *szName,
    const ConstrArray &constrs,
    double *pOut)
```

参量

szName: 双精度型信息名称。

constrs: 指定约束数组。

pOut: 输出双精度型数组，保存了信息的值。

返回值

被查询的有效约束数目。如果出错，返回-1。

Model::Get()

查询与指定二次约束相关的双精度型信息的值。

概要

```
int Get(
    const char *szName,
    const QConstrArray &constrs,
    double *pOut)
```

参量

szName: 双精度型信息的名称。

constrs: 指定二次约束数组。

pOut: 输出双精度型数组，保存了信息的值。

返回值

被查询的有效二次约束数目。如果出错，返回-1。

Model::Get()

查询与指定半定约束相关的信息的值。

概要

```
int Get(
    const char *szName,
    const PsdConstrArray &constrs,
    double *pOut)
```

参量

szName: 双精度型信息的名称。

constrs: 指定半定约束数组。

pOut: 输出双精度型数组，保存了信息的值。

返回值

被查询的有效半定约束数目。如果出错，返回-1。

Model::GetCoeff()

获取变量的系数。

概要

```
double GetCoeff(const Constraint &constr, const Var &var)
```

参量

constr: 指定的约束。

var: 指定的变量。

返回值

指定的变量系数。

Model::GetCol()

获取一个包含指定变量参与的所有约束的列。

概要

```
Column GetCol(const Var &var)
```

参量

var: 指定变量。

返回值

一个关于指定变量的列。

Model::GetColBasis()

获取列的基状态。

概要

```
int GetColBasis(int *pBasis)
```

参量

pBasis: 指向基状态的整型指针。

返回值

列的数量。

Model::GetCone()

获取指定索引值的锥约束。

概要

```
Cone GetCone(int idx)
```

参量

`idx`: 指定索引值。

返回值

想获取的锥约束。

Model::GetConeBuilders()

获取全部锥约束的生成器。

概要

```
ConeBuilderArray GetConeBuilders()
```

返回值

所有锥约束生成器构成的 ConeBuilderArray 类。

Model::GetConeBuilders()

获取给定锥约束的生成器。

概要

```
ConeBuilderArray GetConeBuilders(const ConeArray &cones)
```

参量

`cones`: 锥约束构成的 ConeArray 类。

返回值

想获取的的锥约束生成器构成的 ConeBuilderArray 类。

Model::GetCones()

获取模型中所有锥约束。

概要

```
ConeArray GetCones()
```

返回值

锥约束构成的 ConeArray 类。

Model::GetConstr()

获取模型中指定索引值的约束。

概要

```
Constraint GetConstr(int idx)
```

参量

idx: 指定索引值。

返回值

想获取的约束。

Model::GetConstrBuilder()

获取指定约束的生成器，包括变量，相关的系数，类型和 RHS。

概要

```
ConstrBuilder GetConstrBuilder(Constraint constr)
```

参量

constr: 指定约束。

返回值

约束生成器类。

Model::GetConstrBuilders()

获取模型所有约束生成器。

概要

```
ConstrBuilderArray GetConstrBuilders()
```

返回值

约束生成器构成的 ConstrBuilderArray 类。

Model::GetConstrByName()

获取模型中指定名称的约束。

概要

```
Constraint GetConstrByName(const char *szName)
```

参量

szName: 指定名称。

返回值

想获取的约束。

Model::GetConstrLowerIIS()

获取约束下界的 IIS 状态。

概要

```
int GetConstrLowerIIS(const ConstrArray &constrs, int *pLowerIIS)
```

参量

`constrs`: 指定约束构成的 ConstrArray 类对象。

`pLowerIIS`: 约束下界的 IIS 状态。

返回值

约束数目。

Model::GetConstrs()

获取模型所有约束。

概要

```
ConstrArray GetConstrs()
```

返回值

约束构成的 ConstrArray 类。

Model::GetConstrUpperIIS()

获取约束上界的 IIS 状态。

概要

```
int GetConstrUpperIIS(const ConstrArray &constrs, int *pUpperIIS)
```

参量

`constrs`: 指定约束构成的 ConstrArray 类对象。

`pUpperIIS`: 约束上界的 IIS 状态。

返回值

约束数目。

Model::GetDblAttr()

获取 COPT 双精度型属性的值。

概要

```
double GetDblAttr(const char *szAttr)
```

参量

`szAttr`: 双精度型属性的名称。

返回值

双精度型属性的值。

Model::GetDblParam()

获取 COPT 双精度型参数的值。

概要

```
double GetDblParam(const char *szParam)
```

参量

`szParam`: 双精度型参数的名称。

返回值

双精度型参数的值。

Model::GetGenConstrIndicator()

获取给定类型指示类一般约束（GenConstr）的生成器。

概要

```
GenConstrBuilder GetGenConstrIndicator(const GenConstr &indicator)
```

参量

`indicator`: 类型指示类一般约束（GenConstr）。

返回值

类型指示类一般约束（GenConstr）的生成器。

Model::GetIndicatorIIS()

获取 Indicator 约束的 IIS 状态。

概要

```
int GetIndicatorIIS(const GenConstrArray &genconstrs, int *pIIS)
```

参量

`genconstrs`: 指定 Indicator 约束构成的 GenConstrArray 类对象。

`pIIS`: Indicator 约束的 IIS 状态。

返回值

Indicator 约束数目。

Model::GetIntAttr()

获取 COPT 整型属性的值。

概要

```
int GetIntAttr(const char *szAttr)
```

参量

`szAttr`: 整型属性的名称。

返回值

整型属性的值。

Model::GetIntParam()

获取 COPT 整型参数的值。

概要

```
int GetIntParam(const char *szParam)
```

参量

`szParam`: 整型参数的名称。

返回值

整型参数的值。

Model::GetLmiCoeff()

获取 LMI 约束中指定变量的系数矩阵。

概要

```
SymMatrix GetLmiCoeff(const LmiConstraint &constr, const Var &var)
```

参量

`constr`: 给定的 LMI 约束。

`var`: 给定的变量。

返回值

对应的变量的系数矩阵。

Model::GetLmiConstr()

获取模型中指定索引值对应的 LMI 约束。

概要

```
LmiConstraint GetLmiConstr(int idx)
```

参量

`idx`: 指定的索引值。

返回值

LMI 约束。

Model::GetLmiConstrByName()

获取模型中指定名称的 LMI 约束。

概要

```
LmiConstraint GetLmiConstrByName(const char *szName)
```

参量

`szName`: 指定的 LMI 约束名称。

返回值

LMI 约束对象。

Model::GetLmiConstrs()

获取模型所有 LMI 约束。

概要

```
LmiConstrArray GetLmiConstrs()
```

返回值

LMI 约束构成的 LmiConstrArray 对象。

Model::GetLmiRhs()

获取 LMI 约束中常数项矩阵。

概要

```
SymMatrix GetLmiRhs(const LmiConstraint &constr)
```

参量

`constr`: 给定的 LMI 约束。

返回值

对应的常数项矩阵。

Model::GetLmiRow()

获取参与给定 LMI 约束的 LMI 表达式，包括变量和相关系数矩阵。

概要

```
LmiExpr GetLmiRow(const LmiConstraint &constr)
```

参量

`constr`: 给定的 LMI 约束。

返回值

指向 LMI 约束对应的表达式。

Model::GetLpSolution()

获取 LP 解决方案。

概要

```
void GetLpSolution(
    double *pValue,
    double *pSlack,
    double *pRowDual,
```

```
    double *pRedCost)
```

参量

pValue: 可选，指向解的双精度型指针。

pSlack: 可选，指向松弛值的双精度型指针。

pRowDual: 可选，指向对偶值的双精度型指针。

pRedCost: 可选，指向减少值的双精度型指针。

Model::GetObjective()

获取模型的目标函数的线性表达式。

概要

```
Expr GetObjective()
```

返回值

线性表达式。

Model::GetParamAttrType()

获取 COPT 参数或属性的类型。

概要

```
int GetParamAttrType(const char *szName)
```

参量

szName: COPT 参数或属性的类型。

返回值

参数或属性的类型。

Model::GetParamInfo()

获取当下的，默认的，最小的，最大的 COPT 整型参数。

概要

```
void GetParamInfo(
    const char *szParam,
    int *pnCur,
    int *pnDef,
    int *pnMin,
    int *pnMax)
```

参量

szParam: 整型参数的名称。

pnCur: 整型参数的当前值。

pnDef: 整型参数的默认值。

pnMin: 整型参数的最小值。

pnMax: 整型参数的最大值。

Model::GetParamInfo()

获取当下的，默认的，最小的，最大的 COPT 双精度型参数。

概要

```
void GetParamInfo(
    const char *szParam,
    double *pdCur,
    double *pdDef,
    double *pdMin,
    double *pdMax)
```

参量

szParam: 双精度型参数的名称。

pdCur: 参数的当前值。

pdDef: 双精度型参数的默认值。

pdMin: 双精度型参数的最小值。

pdMax: 双精度型参数的最大值。

Model::GetPoolObjVal()

从解池中获取第 iSol 个解的目标函数值。

概要

```
double GetPoolObjVal(int iSol)
```

参量

iSol: 解的编号。

返回值

指定的目标函数值。

Model::GetPoolSolution()

从解池中获取第 iSol 个解。

概要

```
int GetPoolSolution(
    int iSol,
    const VarArray &vars,
    double *pColVals)
```

参量

iSol: 解的编号。

vars: 指定的变量。

pColVals: 指向解数组的指针。

返回值

解数组的长度。

Model::GetPsdCoeff()

获取半定约束中指定半定变量的系数矩阵。

概要

```
SymMatrix GetPsdCoeff(const PsdConstraint &constr, const PsdVar
&var)
```

参量

constr: 给定的半定约束。

var: 指定的半定变量。

返回值

对应的半定变量的系数矩阵。

Model::GetPsdConstr()

获取模型中指定索引值的半定约束。

概要

```
PsdConstraint GetPsdConstr(int idx)
```

参量

idx: 指定的索引值。

返回值

想获取的半定约束。

Model::GetPsdConstrBuilder()

获取指定约束的生成器，包括半定变量，类型和相关的系数矩阵。

概要

```
PsdConstrBuilder GetPsdConstrBuilder(const PsdConstraint &constr)
```

参量

`constr`: 指定的半定约束。

返回值

指向半定约束生成器对象。

Model::GetPsdConstrBuilders()

获取模型所有半定约束生成器。

概要

```
PsdConstrBuilderArray GetPsdConstrBuilders()
```

返回值

指向半定约束生成器构成的 PsdConstrBuilderArray 对象。

Model::GetPsdConstrByName()

获取模型中指定名称的半定约束。

概要

```
PsdConstraint GetPsdConstrByName(const char *szName)
```

参量

`szName`: 指定的半定约束的名称。

返回值

半定约束对象。

Model::GetPsdConstrs()

获取模型所有半定约束。

概要

```
PsdConstrArray GetPsdConstrs()
```

返回值

指向半定约束构成的 PsdConstrArray 对象。

Model::GetPsdObjective()

获取模型目标函数的半定目标。

概要

```
PsdExpr GetPsdObjective()
```

返回值

半定目标对应的表达式对象。

Model::GetPsdRow()

获取指定半定约束对应的半定表达式，包括半定变量和相关系数矩阵。

概要

```
PsdExpr GetPsdRow(const PsdConstraint &constr)
```

参量

`constr`: 指定的半定约束。

返回值

指向半定约束对应的表达式对象。

Model::GetPsdRow()

获取指定半定约束对应的半定表达式，包括半定变量和相关系数矩阵。

概要

```
PsdExpr GetPsdRow(
    const PsdConstraint &constr,
    double *pLower,
    double *pUpper)
```

参量

`constr`: 指定的半定约束。

`pLower`: 输出的下界数组。

`pUpper`: 输出的上界数组。

返回值

指向半定约束对应的表达式对象。

Model::GetPsdVar()

获取模型中指定索引值的半定变量。

概要

```
PsdVar GetPsdVar(int idx)
```

参量

idx: 索引值。

返回值

想获取的半定变量。

Model::GetPsdVarByName()

获取模型中指定名称的半定变量。

概要

```
PsdVar GetPsdVarByName(const char *szName)
```

参量

szName: 指定名称。

返回值

想获取的半定变量。

Model::GetPsdVars()

获取模型所有半定变量。

概要

```
PsdVarArray GetPsdVars()
```

返回值

半定变量构成的 PsdVarArray 类对象。

Model::GetQConstr()

获取模型中指定索引值的二次约束。

概要

```
QConstraint GetQConstr(int idx)
```

参量

idx: 指定索引值。

返回值

想获取的二次约束。

Model::GetQConstrBuilder()

获取指定约束的生成器，包括变量，相关的系数，类型和 RHS。

概要

```
QConstrBuilder GetQConstrBuilder(const QConstraint &constr)
```

参量

`constr`: 指定约束。

返回值

约束生成器类。

Model::GetQConstrBuilders()

获取模型所有约束生成器。

概要

```
QConstrBuilderArray GetQConstrBuilders()
```

返回值

约束生成器构成的 QConstrBuilderArray 类。

Model::GetQConstrByName()

获取模型中指定名称的二次约束。

概要

```
QConstraint GetQConstrByName(const char *szName)
```

参量

`szName`: 指定二次约束的名称。

返回值

想获取的二次约束对象。

Model::GetQConstrs()

获取模型所有二次约束。

概要

```
QConstrArray GetQConstrs()
```

返回值

二次约束构成的 QConstrArray 类对象。

Model::GetQuadObjective()

获取模型目标函数的二次目标。

概要

```
QuadExpr GetQuadObjective()
```

返回值

二次目标对应的表达式对象。

Model::GetQuadRow()

获取参与指定二次约束的变量，以及相关系数。

概要

```
QuadExpr GetQuadRow(const QConstraint &constr)
```

参量

`constr`: 指定二次约束。

返回值

二次约束的表达式。

Model::GetQuadRow()

获取参与指定二次约束的变量，以及相关系数。

概要

```
QuadExpr GetQuadRow(
    const QConstraint &constr,
    char *pSense,
    double *pBound)
```

参量

`constr`: 指定二次约束。

`pSense`: 二次约束的类型。

`pBound`: 二次约束的右端项。

返回值

二次约束的表达式。

Model::GetRow()

获取参与指定约束的变量，以及相关的系数。

概要

```
Expr GetRow(const Constraint &constr)
```

参量

`constr`: 指定约束。

返回值

约束的表达式。

Model::GetRowBasis()

获取行的基状态。

概要

```
int GetRowBasis(int *pBasis)
```

参量

`pBasis`: 指向基状态的整型指针。

返回值

行的数量。

Model::GetSolution()

获取 MIP 解决方案。

概要

```
void GetSolution(double *pValue)
```

参量

`pValue`: 指向解的双精度型指针。

Model::GetSos()

获取指定索引值的 SOS 约束。

概要

```
Sos GetSos(int idx)
```

参量

`idx`: 指定索引值。

返回值

想获取的 SOS 约束。

Model::GetSosBuilders()

获取模型中所有 SOS 约束生成器。

概要

```
SosBuilderArray GetSosBuilders()
```

返回值

SOS 约束生成器构成的 SosBuilderArray 类。

Model::GetSosBuilders()

获取给定 SOS 约束的生成器。

概要

```
SosBuilderArray GetSosBuilders(const SosArray &soss)
```

参量

`soss`: SOS 约束构成的 SosArray 类。

返回值

想获取的的 SOS 约束生成器构成的 SosBuilderArray 类。

Model::GetSOSIIS()

获取 SOS 约束的 IIS 状态。

概要

```
int GetSOSIIS(const SosArray &soss, int *pIIS)
```

参量

`soss`: 指定 SOS 约束构成的 SosArray 类对象。

`pIIS`: SOS 约束的 IIS 状态。

返回值

SOS 约束数目。

Model::GetSoss()

获取模型中所有 SOS 约束。

概要

```
SosArray GetSoss()
```

返回值

SOS 约束构成的 SosArray 类。

Model::GetSymMat()

获取模型中指定索引值的对称矩阵。

概要

```
SymMatrix GetSymMat(int idx)
```

参量

`idx`: 指定索引值。

返回值

想获取的对称矩阵。

Model::GetVar()

获取模型中指定索引值的变量。

概要

```
Var GetVar(int idx)
```

参量

`idx`: 索引值。

返回值

想获取的变量。

Model::GetVarByName()

获取模型中指定名称的变量。

概要

```
Var GetVarByName(const char *szName)
```

参量

`szName`: 指定名称。

返回值

想获取的变量。

Model::GetVarLowerIIS()

获取变量下界的 IIS 状态。

概要

```
int GetVarLowerIIS(const VarArray &vars, int *pLowerIIS)
```

参量

`vars`: 指定变量构成的 VarArray 类对象。

`pLowerIIS`: 变量下界的 IIS 状态。

返回值

变量数目。

Model::GetVars()

获取模型中所有的变量。

概要

```
VarArray GetVars()
```

返回值

变量构成的 VarArray 类。

Model::GetVarUpperIIS()

获取变量上界的 IIS 状态。

概要

```
int GetVarUpperIIS(const VarArray &vars, int *pUpperIIS)
```

参量

`vars`: 指定变量构成的 VarArray 类对象。

`pUpperIIS`: 变量上界的 IIS 状态。

返回值

变量数目。

Model::Interrupt()

中断当前问题的求解。

概要

```
void Interrupt()
```

Model::LoadMipStart()

为问题里的变量加载最终初始值。

概要

```
void LoadMipStart()
```

Model::LoadTuneParam()

加载指定的调优参数到模型。

概要

```
void LoadTuneParam(int idx)
```

参量

`idx`: 调优参数的下标。

Model::Read()

从文件中读取问题，解决方案，基，MIP start 或者 COPT 参数。

概要

```
void Read(const char *szFileName)
```

参量

`szFileName`: 输入的文件名。

Model::ReadBasis()

从文件中读取基。

概要

```
void ReadBasis(const char *szFileName)
```

参量

`szFileName`: 输入的文件名。

Model::ReadBin()

从文件中读取 COPT 定义的二进制格式的问题。

概要

```
void ReadBin(const char *szFileName)
```

参量

szFileName: 输入的文件名。

Model::ReadCbf()

从文件中读取 CBF 格式的问题。

概要

```
void ReadCbf(const char *szFileName)
```

参量

szFileName: 输入的文件名。

Model::ReadLp()

从文件中读取 LP 格式的问题。

概要

```
void ReadLp(const char *szFileName)
```

参量

szFileName: 输入的文件名。

Model::ReadMps()

从文件中读取 MPS 格式的问题。

概要

```
void ReadMps(const char *szFileName)
```

参量

szFileName: 输入的文件名。

Model::ReadMst()

从文件中读取 MIP start 信息。

概要

```
void ReadMst(const char *szFileName)
```

参量

szFileName: 输入的文件名。

Model::ReadParam()

从文件中读取 COPT 参数。

概要

```
void ReadParam(const char *szFileName)
```

参量

szFileName: 输入的文件名。

Model::ReadSdpa()

从文件中读取 SDPA 格式的问题。

概要

```
void ReadSdpa(const char *szFileName)
```

参量

szFileName: 输入的文件名。

Model::ReadSol()

从文件中读取解决方案。

概要

```
void ReadSol(const char *szFileName)
```

参量

szFileName: 输入的文件名。

Model::ReadTune()

从文件中读取调优参数。

概要

```
void ReadTune(const char *szFileName)
```

参量

`szFileName`: 输入的文件名。

Model::Remove()

从模型中删除一系列变量。

概要

```
void Remove(VarArray &vars)
```

参量

`vars`: 变量构成的 VarArray 对象。

Model::Remove()

从模型中删除一系列约束。

概要

```
void Remove(ConstrArray &constrs)
```

参量

`constrs`: 约束构成的 ConstrArray 对象。

Model::Remove()

从模型中删除一系列 SOS 约束。

概要

```
void Remove(SosArray &soss)
```

参量

`soss`: SOS 约束构成的 SosArray 对象。

Model::Remove()

从模型中删除一系列一般约束。

概要

```
void Remove(GenConstrArray &genConstrs)
```

参量

`genConstrs`: 一般约束构成的 GenConstrArray 对象。

Model::Remove()

从模型中删除一系列二阶锥约束。

概要

```
void Remove(ConeArray &cones)
```

参量

`cones`: 二阶锥约束构成的 ConeArray 对象。

Model::Remove()

从模型中删除一系列二次约束。

概要

```
void Remove(QConstrArray &qconstrs)
```

参量

`qconstrs`: 二次约束构成的 QConstrArray 对象。

Model::Remove()

从模型中删除一批半定变量。

概要

```
void Remove(PsdVarArray &vars)
```

参量

`vars`: 半定变量构成的 PsdVarArray 对象。

Model::Remove()

从模型中删除一批半定约束。

概要

```
void Remove(PsdConstrArray &constrs)
```

参量

`constrs`: 半定约束构成的 PsdConstrArray 对象。

Model::Remove()

从模型中删除一批 LMI 约束。

概要

```
void Remove(LmiConstrArray &constrs)
```

参量

`constrs`: LMI 约束构成的 LmiConstrArray 对象。

Model::Reset()

只重新设置解。

概要

```
void Reset()
```

Model::ResetAll()

重新设置解与其他信息。

概要

```
void ResetAll()
```

Model::ResetParam()

重新设置参数为默认值。

概要

```
void ResetParam()
```

Model::Set()

设置与指定变量相关的双精度型信息的值。

概要

```
void Set(
    const char *szName,
    const VarArray &vars,
    double *pVals,
    int len)
```

参量

szName: 双精度型信息的名称。

vars: 指定变量构成的 VarArray 类。

pVals: 双精度型数组，保存了信息的值。

len: 值数组的长度。

Model::Set()

设置与指定约束相关的双精度型信息的值。

概要

```
void Set(
    const char *szName,
    const ConstrArray &constrs,
    double *pVals,
    int len)
```

参量

szName: 双精度型信息的名称。

constrs: 指定约束构成的 ConstrArray 类。

pVals: 双精度型数组，保存了信息的值。

len: 值数组的长度。

Model::Set()

设置与指定半定约束相关的双精度型信息的值。

概要

```
void Set(
    const char *szName,
    const PsdConstrArray &constrs,
    double *pVals,
    int len)
```

参量

szName: 双精度型信息的名称。

constrs: 指定半定约束构成的 PsdConstrArray 类。

pVals: 双精度型数组，保存了信息的值。

len: 值数组的长度。

Model::SetBasis()

设置行和列的基状态。

概要

```
void SetBasis(int *pColBasis, int *pRowBasis)
```

参量

pColBasis: 指向列的基状态的整型指针。

pRowBasis: 指向行的基状态的整型指针。

Model::SetCallback()

在 COPT 模型中设置用户自定义回调。

概要

```
void SetCallback(ICallback *pcb, int cbctx)
```

参量

pcb: 用户自定义回调类指针。

cbctx: COPT 回调上下文，定义参看 copt.h

Model::SetCoeff()

设置变量的系数。

概要

```
void SetCoeff(
    const Constraint &constr,
    const Var &var,
    double newVal)
```

参量

constr: 指定的约束。

var: 指定的变量。

newVal: 新系数。

Model::SetDblParam()

设置 COPT 双精度型参数的值。

概要

```
void SetDblParam(const char *szParam, double dVal)
```

参量

szParam: 双精度型参数的名称。

dVal: 双精度型参数的值。

Model::SetIntParam()

设置 COPT 整型参数的值。

概要

```
void SetIntParam(const char *szParam, int nVal)
```

参量

szParam: 整型参数的名称。

nVal: 整型参数的值。

Model::SetLmiCoeff()

设置 LMI 约束中指定变量的系数矩阵。

概要

```
void SetLmiCoeff(
    const LmiConstraint &constr,
    const Var &var,
    const SymMatrix &mat)
```

参量

`constr`: 给定的 LMI 约束。

`var`: 给定的变量。

`mat`: 新系数矩阵。

Model::SetLmiRhs()

设置 LMI 约束中指定变量的常数项矩阵。

概要

```
void SetLmiRhs(const LmiConstraint &constr, const SymMatrix &mat)
```

参量

`constr`: 给定的 LMI 约束。

`mat`: 新常数项矩阵。

Model::SetLpSolution()

设置 LP 解。

概要

```
void SetLpSolution(
    double *pValue,
    double *pSlack,
    double *pRowDual,
    double *pRedCost)
```

参量

`pValue`: 指向解的双精度型指针。

`pSlack`: 指向松弛值的双精度型指针。

`pRowDual`: 指向对偶值的双精度型指针。

pRedCost: 指向减少值的双精度型指针。

Model::SetMipStart()

设置给定数目变量的初始值，从第一个开始。

概要

```
void SetMipStart(int count, double *pVals)
```

参量

count: 设置变量的数量。

pVals: 指向初始值的指针。

Model::SetMipStart()

设置指定变量的初始值。

概要

```
void SetMipStart(const Var &var, double val)
```

参量

var: 指定变量。

val: 变量的初始值。

Model::SetMipStart()

设置一系列变量的初始值。

概要

```
void SetMipStart(const VarArray &vars, double *pVals)
```

参量

vars: 指定变量构成的 VarArray 类。

pVals: 指向初始值的指针。

Model::SetObjConst()

设置目标函数里的的常数。

概要

```
void SetObjConst(double constant)
```

参量

constant: 常数的值。

Model::SetObjective()

设置模型的目标函数。

概要

```
void SetObjective(const Expr &expr, int sense)
```

参量

expr: 目标函数的表达式。

sense: 可选，默认值 0 表示不改变模型当前的优化方向。

Model::SetObjSense()

设置目标函数的优化方向。

概要

```
void SetObjSense(int sense)
```

参量

sense: 目标函数的优化方向。

Model::SetPsdCoeff()

设置半定约束中指定半定变量的系数矩阵。

概要

```
void SetPsdCoeff(
    const PsdConstraint &constr,
    const PsdVar &var,
    const SymMatrix &mat)
```

参量

constr: 给定的半定约束。

var: 指定的半定变量。

mat: 新系数矩阵。

Model::SetPsdObjective()

设置模型的半定目标。

概要

```
void SetPsdObjective(const PsdExpr &expr, int sense)
```

参量

expr: 模型目标函数的半定表达式。

sense: 可选，默认值 0 表示不改变模型当前的优化方向。

Model::SetQuadObjective()

设置模型的二次目标。

概要

```
void SetQuadObjective(const QuadExpr &expr, int sense)
```

参量

expr: 模型目标函数的二次表达式。

sense: 可选，默认值 0 表示不改变模型当前的优化方向。

Model::SetSlackBasis()

设置松弛状态。

概要

```
void SetSlackBasis()
```

Model::SetSolverLogCallback()

为 COPT 设置日志回调。

概要

```
void SetSolverLogCallback(ILogCallback *pcb)
```

参量

pcb: 日志回调类指针。

Model::SetSolverLogFile()

为 COPT 设置日志文件。

概要

```
void SetSolverLogFile(const char *szLogFile)
```

参量

`szLogFile`: 日志文件名。

Model::Solve()

求解当前的 MIP 问题。

概要

```
void Solve()
```

Model::SolveLp()

求解当前的 LP 问题。

概要

```
void SolveLp()
```

Model::Tune()

模型调优。

概要

```
void Tune()
```

Model::Write()

将问题，解决方案，基，MIP start 或者更改后的 COPT 参数输出到文件中。

概要

```
void Write(const char *szFileName)
```

参量

`szFileName`: 输出的文件名。

Model::WriteBasis()

将最优基输出到“.bas”文件。

概要

```
void WriteBasis(const char *szFileName)
```

参量

szFileName: 输出的文件名。

Model::WriteBin()

将问题以 COPT 二进制格式输出到文件中。

概要

```
void WriteBin(const char *szFileName)
```

参量

szFileName: 输出的文件名。

Model::WriteIIS()

将 IIS 输出到文件中。

概要

```
void WriteIIS(const char *szFileName)
```

参量

szFileName: 输出文件名。

Model::WriteLp()

将问题以 LP 格式输出到文件中。

概要

```
void WriteLp(const char *szFileName)
```

参量

szFileName: 输出的文件名。

Model::WriteMps()

将问题以 MPS 格式输出到文件中。

概要

```
void WriteMps(const char *szFileName)
```

参量

szFileName: 输出的文件名。

Model::WriteMpsStr()

将问题以 MPS 格式输出到缓存对象。

概要

```
ProbBuffer WriteMpsStr()
```

返回值

输出的缓存对象。

Model::WriteMst()

将整数模型初始解信息输出到“.mst”文件。

概要

```
void WriteMst(const char *szFileName)
```

参量

szFileName: 输出的文件名。

Model::WriteParam()

将更改后的 COPT 参数输出到“.par”文件。

概要

```
void WriteParam(const char *szFileName)
```

参量

szFileName: 输出的文件名。

Model::WritePoolSol()

将指定的解池中的结果输出到“.sol”文件。

概要

```
void WritePoolSol(int iSol, const char *szFileName)
```

参量

`iSol`: 解池中解的索引。

`szFileName`: 输出的文件名。

Model::WriteRelax()

将可行化松弛模型输出到文件中。

概要

```
void WriteRelax(const char *szFileName)
```

参量

`szFileName`: 输出文件名。

Model::WriteSol()

将求解结果输出到“.sol”文件。

概要

```
void WriteSol(const char *szFileName)
```

参量

`szFileName`: 输出的文件名。

Model::WriteTuneParam()

将指定的调优参数输出到“.par”文件。

概要

```
void WriteTuneParam(int idx, const char *szFileName)
```

参量

`idx`: 调优参数下标。

`szFileName`: 输出的文件名。

24.4.4 Var 类

Var 类是杉数优化求解器变量的相关操作的封装，提供了以下成员方法：

Var::Get()

获取变量的属性值。支持"Value", "RedCost", "LB", "UB", and "Obj" 属性。

概要

```
double Get(const char *szAttr)
```

参量

`szAttr`: 属性的名称。

返回值

属性的值。

Var::GetIdx()

获取变量的索引值。

概要

```
inline int GetIdx()
```

返回值

索引值。

Var::GetLowerIIS()

获取变量下界的 IIS 状态。

概要

```
int GetLowerIIS()
```

返回值

变量下界的 IIS 状态。

Var::GetName()

获取变量的名称。

概要

```
const char *GetName()
```

返回值

变量名称。

Var::GetType()

获取变量的类型。

概要

```
char GetType()
```

返回值

变量类型。

Var::GetUpperIIS()

获取变量上界的 IIS 状态。

概要

```
int GetUpperIIS()
```

返回值

变量上界的 IIS 状态。

Var::Remove()

从模型中删除变量。

概要

```
void Remove()
```

Var::Set()

设置变量的属性值。支持”LB”, ”UB” 和”Obj” 属性。

概要

```
void Set(const char *szAttr, double value)
```

参量

`szAttr`: 属性的名称。

`value`: 新的值。

Var::SetName()

设置变量的名称。

概要

```
void SetName(const char *szName)
```

参量

`szName`: 变量名称。

Var::SetType()

设置变量的类型。

概要

```
void SetType(char type)
```

参量

`type`: 变量类型。

24.4.5 VarArray 类

为方便用户对一组*Var 类* 对象进行操作，杉数优化求解器的 C++ 接口设计了 VarArray 类，提供了以下成员方法：

VarArray::GetVar()

获取 VarArray 类中指定索引值的变量。

概要

```
Var &GetVar(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值的变量。

VarArray::PushBack()

向 VarArray 类中添加一个变量。

概要

```
void PushBack(const Var &var)
```

参量

`var`: 变量。

VarArray::Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

`n`: 容纳 n 项的空间。

VarArray::Size()

获取 VarArray 类中变量的数目。

概要

```
int Size()
```

返回值

VarArray 类中变量的数目。

24.4.6 Expr 类

Expr 类是杉数求解器中用于构建线性表达式时变量的相关组合操作，提供了以下成员方法：

Expr::Expr()

Expr 的构造函数。

概要

```
Expr(double constant)
```

参量

`constant`: Expr 中的常值。

Expr::Expr()

只有一项的 Expr 的构造函数。

概要

```
Expr(const Var &var, double coeff)
```

参量

`var`: 添加的这一项对应的变量。

`coeff`: 添加的这一项对应的参数。

Expr::AddConstant()

增加表达式中的常数。

概要

```
void AddConstant(double constant)
```

参量

`constant`: 表达式中的常数。

Expr::AddExpr()

添加一个表达式的项，并乘以倍数。

概要

```
void AddExpr(const Expr &expr, double mult)
```

参量

`expr`: 需要添加的表达式。

`mult`: 可选的系数倍数，默认值为 1.0。

Expr::AddTerm()

向表达式中添加一项。

概要

```
void AddTerm(const Var &var, double coeff)
```

参量

`var`: 新项中的变量。

`coeff`: 新项中的系数。

Expr::AddTerms()

向表达式中添加项。

概要

```
int AddTerms(
    const VarArray &vars,
    double *pCoeff,
    int len)
```

参量

vars: 新项中的变量。

pCoeff: 新项中的系数数组。

len: 系数数组的长度。

返回值

增加的项数。

Expr::Clone()

深拷贝表达式。

概要

```
Expr Clone()
```

返回值

表达式的深拷贝。

Expr::Evaluate()

求解后对线性表达式估值。

概要

```
double Evaluate()
```

返回值

表达式估值。

Expr::GetCoeff()

获取表达式指定索引值项数中的系数。

概要

```
double GetCoeff(int i)
```

参量

i: 指定索引值。

返回值

指定索引值项数中的系数。

Expr::GetConstant()

获取表达式中的常数。

概要

```
double GetConstant()
```

返回值

表达式中的常数。

Expr::GetVar()

获取表达式指定索引值项数中的变量。

概要

```
Var &GetVar(int i)
```

参量

i: 指定索引值。

返回值

指定索引值项数中的变量。

Expr::operator*=()

自乘一个常数。

概要

```
void operator*=(double c)
```

参量

c: 对应的双精度常数。

Expr::operator*()

乘以一个常数并返回新的表达式。

概要

```
Expr operator*(double c)
```

参量

c: 乘以的常数。

返回值

结果表达式。

Expr::operator*()

乘以一个变量并返回新的二次表达式。

概要

```
QuadExpr operator*(const Var &var)
```

参量

var: 变量对象。

返回值

二次结果表达式。

Expr::operator*()

乘以一个线性表达式并返回新的二次表达式。

概要

```
QuadExpr operator*(const Expr &other)
```

参量

other: 线性表达式对象。

返回值

二次结果表达式。

Expr::operator+=()

自加一个表达式。

概要

```
void operator+=(const Expr &expr)
```

参量

expr: 需要添加的表达式。

Expr::operator+()

增加一个表达式并返回新的表达式。

概要

```
Expr operator+(const Expr &other)
```

参量

other: 加上的表达式。

返回值

结果表达式。

Expr::operator-=()

自减一个表达式。

概要

```
void operator-=(const Expr &expr)
```

参量

expr: 需要减去的表达式。

Expr::operator-()

减去一个表达式并返回新的表达式。

概要

```
Expr operator-(const Expr &other)
```

参量

other: 减去的表达式。

返回值

结果表达式。

Expr::Remove()

删除表达式中指定索引值的项。

概要

```
void Remove(int i)
```

参量

`i`: 指定索引值。

Expr::Remove()

删除表达式中与指定变量相关的项。

概要

```
void Remove(const Var &var)
```

参量

`var`: 指定变量。

Expr::Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(size_t n)
```

参量

`n`: 容纳 n 项的空间。

Expr::SetCoeff()

设置表达式指定索引值项数中的系数。

概要

```
void SetCoeff(int i, double val)
```

参量

`i`: 指定索引值。

`val`: 指定索引值项数中的参数。

Expr::SetConstant()

设置表达式中的常数。

概要

```
void SetConstant(double constant)
```

参量

`constant`: 表达式中的常数。

Expr::Size()

获取表达式中的项数。

概要

```
size_t Size()
```

返回值

表达式中的项数。

24.4.7 Constraint 类

Constraint 类是杉数求解器线性约束的相关操作的封装，提供了以下成员方法：

Constraint::Get()

获得 Constraint 的属性值。支持"Dual", "Slack", "LB", "UB" 属性。

概要

```
double Get(const char *szAttr)
```

参量

`szAttr`: 所需要获得的属性名称。

返回值

属性值。

Constraint::GetBasis()

获得 Constraint 的基状态。

概要

```
int GetBasis()
```

返回值

Constraint 的基状态。

Constraint::GetIdx()

获取 Constraint 的索引值。

概要

```
int GetIdx()
```

返回值

Constraint 的索引值。

Constraint::GetLowerIIS()

获取约束下界的 IIS 状态。

概要

```
int GetLowerIIS()
```

返回值

约束下界的 IIS 状态。

Constraint::GetName()

获取 Constraint 的名称。

概要

```
const char *GetName()
```

返回值

Constraint 的名称。

Constraint::GetUpperIIS()

获取约束上界的 IIS 状态。

概要

```
int GetUpperIIS()
```

返回值

约束上界的 IIS 状态。

Constraint::Remove()

从模型中删除当前的 Constraint。

概要

```
void Remove()
```

Constraint::Set()

设置 Constraint 的属性值。支持"LB", "UB" 属性。

概要

```
void Set(const char *szAttr, double value)
```

参量

`szAttr`: 所需要设置的属性名称。

`value`: 所需要设置的属性值。

Constraint::SetName()

设置 Constraint 的名称。

概要

```
void SetName(const char *szName)
```

参量

`szName`: Constraint 的名称。

24.4.8 ConstrArray 类

为方便用户对一组 C++ *Constraint 类* 对象进行操作，杉数求解器的 C++ 接口设计了 ConstrArray 类，提供了以下成员方法：

ConstrArray::GetConstr()

获取 ConstrArray 中的指定索引值的 Constraint。

概要

```
Constraint &GetConstr(int i)
```

参量

`i`: 指定的索引值。

返回值

指定的 Constraint。

ConstrArray::PushBack()

向 ConstrArray 中添加一个 Constraint。

概要

```
void PushBack(const Constraint &constr)
```

参量

`constr`: 待添加的 Constraint。

ConstrArray::Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

`n`: 容纳 n 项的空间。

ConstrArray::Size()

获取 ConstrArray 中的元素个数。

概要

```
int Size()
```

返回值

ConstrArray 中的元素个数。

24.4.9 ConstrBuilder 类

ConstrBuilder 类是杉数优化求解器中构建线性约束时的构建器的封装，提供了以下成员方法：

ConstrBuilder::GetExpr()

获取线性约束生成器对象的表达式。

概要

```
const Expr &GetExpr()
```

返回值

Expression 对象。

ConstrBuilder::GetRange()

获取线性约束生成器对象的约束范围的长度（从下界到上界的长度，必须大于 0）。

概要

```
double GetRange()
```

返回值

约束范围的长度（从下界到上界的长度）。

ConstrBuilder::GetSense()

获取线性约束生成器对象的约束类型。

概要

```
char GetSense()
```

返回值

约束类型。

ConstrBuilder::Set()

设置一个约束构造类的内容。

概要

```
void Set(
    const Expr &expr,
    char sense,
    double rhs)
```

参量

`expr`: 约束一侧的表达式。

`sense`: 除了 COPT_RANGE 外的约束类型。

`rhs`: 约束另一侧的常数项

ConstrBuilder::SetRange()

设置一个范围约束（带有上下界）。

概要

```
void SetRange(const Expr &expr, double range)
```

参量

`expr`: 约束表达式。其表达式的常数项的负数其实是这个约束的上界。

`range`: 约束范围的长度（从下界到上界的长度，必须大于 0）。

24.4.10 ConstrBuilderArray 类

为方便用户对一组 C++ *ConstrBuilder 类* 对象进行操作，杉数优化求解器的 C++ 接口设计了 ConstrBuilderArray 类，提供了以下成员方法：

ConstrBuilderArray::GetBuilder()

获取 ConstrBuilderArray 中的指定索引值的 Constraint。

概要

```
ConstrBuilder &GetBuilder(int i)
```

参量

`i`: 指定的索引值。

返回值

指定的 ConstrBuilder。

ConstrBuilderArray::PushBack()

向 ConstrBuilderArray 中添加一个约束生成器。

概要

```
void PushBack(const ConstrBuilder &builder)
```

参量

`builder`: 待添加的约束生成器。

ConstrBuilderArray::Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

`n`: 容纳 n 项的空间。

ConstrBuilderArray::Size()

获取 ConstrBuilderArray 中的元素个数。

概要

```
int Size()
```

返回值

ConstrBuilderArray 中的元素个数。

24.4.11 Column 类

为了方便用户采用按列建模的方式，杉数优化求解器的 C++ 接口设计了 Column 类，提供了以下成员方法：

Column::Column()

Column 的构造函数。

概要

```
Column()
```

Column::AddColumn()

添加一个列的项，并乘以倍数。

概要

```
void AddColumn(const Column &col, double mult)
```

参量

`col`: 需要添加的列对象。

`mult`: 系数倍数。

Column::AddTerm()

添加一个新的项。

概要

```
void AddTerm(const Constraint &constr, double coeff)
```

参量

`constr`: 待添加项的线性约束。

`coeff`: 待添加项的系数。

Column::AddTerms()

添加一个或多个新的项。

概要

```
int AddTerms(
    const ConstrArray &constrs,
    double *pCoeff,
    int len)
```

参量

`constrs`: 待添加项的线性约束。

`pCoeff`: 待添加项的系数。

`len`: 待添加项的数量。

返回值

添加项的数量。

Column::Clear()

清空 Column 的内容。

概要

```
void Clear()
```

Column::Clone()

创建 Column 的深拷贝。

概要

```
Column Clone()
```

返回值

Column 的深拷贝。

Column::GetCoeff()

获得 Column 中第 i 项的系数。

概要

```
double GetCoeff(int i)
```

参量

`i`: 第 i 项的索引值。

返回值

Column 中第 i 项的系数。

Column::GetConstr()

获得 Column 中第 i 项的线性约束。

概要

```
Constraint GetConstr(int i)
```

参量

`i`: 第 i 项的索引值。

返回值

Column 中第 i 项的线性约束。

Column::Remove()

从 Column 中移除指定的项。

概要

```
void Remove(int i)
```

参量

`i`: 待移除项的索引值。

Column::Remove()

从 Column 中移除指定线性约束所在的项。

概要

```
bool Remove(const Constraint &constr)
```

参量

`constr`: 指定线性约束。

返回值

当该线性约束存在于 Column 的时候返回值为真。

Column::Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

`n`: 容纳 n 项的空间。

Column::Size()

获取 Column 中元素的个数。

概要

```
int Size()
```

返回值

Column 中元素的个数。

24.4.12 ColumnArray 类

为方便用户对一组 C++ *Column 类* 对象进行操作，杉数优化求解器的 C++ 接口设计了 ColumnArray 类，提供了以下成员方法：

ColumnArray::Clear()

清空所有的 Column。

概要

```
void Clear()
```

ColumnArray::GetColumn()

获取 ColumnArray 中的指定索引值的 Column。

概要

```
Column &GetColumn(int i)
```

参量

`i`: 指定的索引值。

返回值

指定的 Column。

ColumnArray::PushBack()

向 ColumnArray 中添加一个 Column。

概要

```
void PushBack(const Column &col)
```

参量

`col`: 待添加的 Column。

ColumnArray::Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

`n`: 容纳 n 项的空间。

ColumnArray::Size()

获取 ColumnArray 中的元素个数。

概要

```
int Size()
```

返回值

ColumnArray 中的元素个数。

24.4.13 Sos 类

SOS 类是杉数求解器的 SOS 约束的相关操作的封装，目前提供了以下成员方法：

关于 SOS 约束的介绍请参考特殊约束：*SOS* 约束章节 。

Sos::GetIdx()

获取 SOS 约束的索引值。

概要

```
int GetIdx()
```

返回值

SOS 约束的索引值。

Sos::GetIIS()

获取 SOS 约束的 IIS 状态。

概要

```
int GetIIS()
```

返回值

IIS 状态。

Sos::Remove()

从模型中删除 SOS 约束。

概要

```
void Remove()
```

24.4.14 SosArray 类

为方便用户对一组 C++ *Sos 类* 对象进行操作，杉数求解器的 C++ 接口设计了 `SosArray` 类，提供了以下成员方法：

SosArray::GetSos()

获取 SosArray 里指定索引的 SOS 约束。

概要

```
Sos &GetSos(int i)
```

参量

`i`: SOS 约束的索引。

返回值

指定索引的 SOS 约束。

SosArray::PushBack()

向 SosArray 里添加 SOS 约束。

概要

```
void PushBack(const Sos &sos)
```

参量

`sos`: SOS 约束。

SosArray::Size()

获取 SosArray 里 SOS 约束个数。

概要

```
int Size()
```

返回值

SOS 约束个数。

24.4.15 SosBuilder 类

SOSBuilder 类是杉数优化求解器中构建 SOS 约束的构建器的封装，提供了以下成员方法：

关于 SOS 约束的介绍请参考特殊约束：*SOS* 约束章节 。

SosBuilder::GetSize()

获取 SOS 约束中元素个数。

概要

```
int GetSize()
```

返回值

元素个数。

SosBuilder::GetType()

获取 SOS 约束类型。

概要

```
int GetType()
```

返回值

SOS 约束类型。

SosBuilder::GetVar()

从 SOS 约束中指定索引的元素中获取变量。

概要

```
Var GetVar(int i)
```

参量

`i`: 指定的索引值。

返回值

指定索引元素对应的变量。

SosBuilder::GetVars()

获取 SOS 约束中所有元素对应的变量。

概要

```
VarArray GetVars()
```

返回值

变量构成的 VarArray 对象。

SosBuilder::GetWeight()

从 SOS 约束中指定索引的元素中获取权重。

概要

```
double GetWeight(int i)
```

参量

`i`: 指定的索引值。

返回值

指定索引元素中对应的权重。

SosBuilder::GetWeights()

获取 SOS 约束中所有元素对应的权重。

概要

```
double GetWeights()
```

返回值

指向权重数组的指针。

SosBuilder::Set()

设置 SOS 约束的变量和权重。

概要

```
void Set(
    const VarArray &vars,
    const double *pWeights,
    int type)
```

参量

`vars`: 变量构成的 VarArray 类。

`pWeights`: 指向权重的指针。

`type`: SOS 约束的类型。

24.4.16 SosBuilderArray 类

为方便用户对一组 C++ *SosBuilder 类* 对象进行操作，杉数优化求解器的 C++ 接口设计了 SosBuilderArray 类，提供了以下成员方法：

SosBuilderArray::GetBuilder()

获取指定索引值的 SOS 约束生成器（SosBuilder）。

概要

```
SosBuilder &GetBuilder(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值的 SOS 约束生成器（SosBuilder）。

SosBuilderArray::PushBack()

向 SosBuilderArray 类中添加 SOS 约束生成器（SosBuilder）。

概要

```
void PushBack(const SosBuilder &builder)
```

参量

`builder`: SOS 约束生成器（SosBuilder）。

SosBuilderArray::Size()

获取 SosBuilderArray 类中元素个数。

概要

```
int Size()
```

返回值

SosBuilderArray 类中元素个数。

24.4.17 GenConstr 类

GenConstr 类是杉数优化求解器的 Indicator 约束的相关操作的封装，提供了以下成员方法：

关于 Indicator 约束的介绍请参考特殊约束：*Indicator 约束*章节 。

GenConstr::GetIdx()

获取 GenConstr 的索引值。

概要

```
int GetIdx()
```

返回值

GenConstr 的索引值。

GenConstr::GetIIS()

获取一般约束的 IIS 状态。

概要

```
int GetIIS()
```

返回值

IIS 状态。

GenConstr::Remove()

从模型中删除 GenConstr。

概要

```
void Remove()
```

24.4.18 GenConstrArray 类

为方便用户对一组 C++ *GenConstr 类* 对象进行操作，杉数优化求解器的 C++ 接口设计了 GenConstrArray 类，提供了以下成员方法：

GenConstrArray::GetGenConstr()

获取 GenConstrArray 中的指定索引值的 GenConstr。

概要

```
GenConstr &GetGenConstr(int i)
```

参量

`i`: 指定的索引值。

返回值

指定的 GenConstr。

GenConstrArray::PushBack()

向 GenConstrArray 中添加一个 GenConstr。

概要

```
void PushBack(const GenConstr &constr)
```

参量

`constr`: 待添加的 GenConstr。

GenConstrArray::Size()

获取 GenConstrArray 中的元素个数。

概要

```
int Size()
```

返回值

GenConstrArray 中的元素个数。

24.4.19 GenConstrBuilder 类

GenConstrBuilder 类是杉数优化求解器中构建 Indicator 约束时的构建器的封装，提供了以下成员方法：

关于 Indicator 约束的介绍请参考特殊约束：*Indicator 约束*章节 。

GenConstrBuilder::GetBinVal()

获取与 GenConstr 的相关联的二进制值。

概要

```
int GetBinVal()
```

返回值

二进制值。

GenConstrBuilder::GetBinVar()

获取与 GenConstr 的相关联的二进制变量。

概要

```
Var GetBinVar()
```

返回值

二进制变量。

GenConstrBuilder::GetExpr()

获取与 GenConstr 的相关联的表达式。

概要

```
const Expr &GetExpr()
```

返回值

表达式对象。

GenConstrBuilder::GetSense()

获取与 GenConstr 的相关联的约束类型。

概要

```
char GetSense()
```

返回值

约束类型。

GenConstrBuilder::Set()

设置 GenConstr 二进制变量，二进制值，表达式，约束类型。

概要

```
void Set(
    Var var,
    int val,
    const Expr &expr,
    char sense)
```

参量

`var`: 二进制变量。

`val`: 二进制值。

`expr`: 表达式。

`sense`: 约束类型。

24.4.20 GenConstrBuilderArray 类

为方便用户对一组 C++ *GenConstrBuilder 类* 对象进行操作，杉数优化求解器的 C++ 接口设计了 GenConstrBuilderArray 类，提供了以下成员方法：

GenConstrBuilderArray::GetBuilder()

获取 GenConstrBuilderArray 中的指定索引值的 GenConstrBuilder。

概要

```
GenConstrBuilder &GetBuilder(int i)
```

参量

`i`: 指定的索引值。

返回值

指定的 GenConstrBuilder。

GenConstrBuilderArray::PushBack()

向 GenConstrBuilderArray 中添加一个 GenConstrBuilder。

概要

```
void PushBack(const GenConstrBuilder &builder)
```

参量

`builder`: 待添加的 GenConstrBuilder。

GenConstrBuilderArray::Size()

获取 GenConstrBuilderArray 中的元素个数。

概要

```
int Size()
```

返回值

GenConstrBuilderArray 中的元素个数。

24.4.21 Cone 类

Cone 类是杉数求解器的二阶锥约束的相关操作的封装，目前提供了以下成员方法：

Cone::GetIdx()

获取锥约束的索引值。

概要

```
int GetIdx()
```

返回值

锥约束的索引值。

Cone::Remove()

从模型中删除锥约束。

概要

```
void Remove()
```

24.4.22 ConeArray 类

为方便用户对一组 C++ *Cone 类* 对象进行操作，杉数求解器的 C++ 接口设计了 ConeArray 类，提供了以下成员方法：

ConeArray::GetCone()

获取 ConeArray 里指定下标的锥约束。

概要

```
Cone &GetCone(int i)
```

参量

`i`: 锥约束的下标。

返回值

指定下标的锥约束。

ConeArray::PushBack()

向 ConeArray 里添加锥约束。

概要

```
void PushBack(const Cone &cone)
```

参量

`cone`: 锥约束。

ConeArray::Size()

获取 ConeArray 里锥约束个数。

概要

```
int Size()
```

返回值

锥约束个数。

24.4.23 ConeBuilder 类

ConeBuilder 类是杉数优化求解器中构建二阶锥约束的构建器的封装，提供了以下成员方法：

ConeBuilder::GetSize()

获取锥约束中变量个数。

概要

```
int GetSize()
```

返回值

变量个数。

ConeBuilder::GetType()

获取锥约束类型。

概要

```
int GetType()
```

返回值

锥约束类型。

ConeBuilder::GetVar()

从锥约束中获取指定下标的变量。

概要

```
Var GetVar(int i)
```

参量

`i`: 指定的下标值。

返回值

指定下标的变量。

ConeBuilder::GetVars()

获取锥约束中所有对应的变量。

概要

```
VarArray GetVars()
```

返回值

变量构成的 VarArray 对象。

ConeBuilder::Set()

设置锥约束的变量。

概要

```
void Set(const VarArray &vars, int type)
```

参量

`vars`: 变量构成的 VarArray 类。

`type`: 锥约束的类型。

24.4.24 ConeBuilderArray 类

为方便用户对一组 C++ *ConeBuilder 类* 对象进行操作，杉数优化求解器的 C++ 接口设计了 ConeBuilderArray 类，提供了以下成员方法：

ConeBuilderArray::GetBuilder()

获取指定索引值的 cone 约束生成器（ConeBuilder）。

概要

```
ConeBuilder &GetBuilder(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值的 cone 约束生成器（ConeBuilder）。

ConeBuilderArray::PushBack()

向 ConeBuilderArray 类中添加锥约束生成器（ConeBuilder）。

概要

```
void PushBack(const ConeBuilder &builder)
```

参量

`builder`: 锥约束生成器（ConeBuilder）。

ConeBuilderArray::Size()

获取 ConeBuilderArray 类中元素个数。

概要

```
int Size()
```

返回值

ConeBuilderArray 类中元素个数。

24.4.25 QuadExpr 类

COPT 二次表达式包括一个线性表达式，一些二次项相关的变量和对应系数。QuadExpr 类是杉数求解器中用于构建二次表达式时对变量的相关组合操作，提供了以下成员方法：

QuadExpr::QuadExpr()

QuadExpr 的构造函数。

概要

```
QuadExpr(double constant)
```

参量

`constant`: QuadExpr 中的常值。

QuadExpr::QuadExpr()

使用变量和其系数构造的二次表达式。

概要

```
QuadExpr(const Var &var, double coeff)
```

参量

`var`: 添加的这一项对应的变量。

`coeff`: 添加的这一项对应的参数。

QuadExpr::QuadExpr()

使用线性表达式构造的二次表达式。

概要

```
QuadExpr(const Expr &expr)
```

参量

`expr`: 初始的线性表达式

QuadExpr::QuadExpr()

使用两个线性表达式构造的二次表达式。

概要

```
QuadExpr(const Expr &expr, const Var &var)
```

参量

`expr`: 一个初始的线性表达式。

`var`: 另一个初始的变量。

QuadExpr::QuadExpr()

使用两个线性表达式构造的二次表达式。

概要

```
QuadExpr(const Expr &left, const Expr &right)
```

参量

`left`: 一个初始的线性表达式。

`right`: 另一个初始的线性表达式。

QuadExpr::AddConstant()

增加表达式中的常数。

概要

```
void AddConstant(double constant)
```

参量

`constant`: 表达式中的常数。

QuadExpr::AddLinExpr()

添加一个线性表达式的项，并乘以倍数。

概要

```
void AddLinExpr(const Expr &expr, double mult)
```

参量

`expr`: 需要添加的线性表达式。

`mult`: 可选的系数倍数，默认值为 1.0。

QuadExpr::AddQuadExpr()

添加一个二次表达式的项，并乘以倍数。

概要

```
void AddQuadExpr(const QuadExpr &expr, double mult)
```

参量

`expr`: 需要添加的二次表达式。

`mult`: 可选的系数倍数，默认值为 1.0。

QuadExpr::AddTerm()

向表达式中添加一线性项。

概要

```
void AddTerm(const Var &var, double coeff)
```

参量

`var`: 新线性项中的变量。

`coeff`: 新线性项中的系数。

QuadExpr::AddTerm()

向表达式中添加一个二次项。

概要

```
void AddTerm(
    const Var &var1,
    const Var &var2,
    double coeff)
```

参量

`var1`: 新二次项中的变量 1。

`var2`: 新二次项中的变量 2。

`coeff`: 新二次项中的系数。

QuadExpr::AddTerms()

向表达式中添加一些线性项。

概要

```
int AddTerms(
    const VarArray &vars,
    double *pCoeff,
    int len)
```

参量

vars: 新线性项中的变量数组。

pCoeff: 新线性项中的系数数组。

len: 系数数组的长度。

返回值

增加的线性项个数。

QuadExpr::AddTerms()

向表达式中添加一些二次项。

概要

```
int AddTerms(
    const VarArray &vars1,
    const VarArray &vars2,
    double *pCoeff,
    int len)
```

参量

vars1: 新二次项中的变量数组 1。

vars2: 新二次项中的变量数组 2。

pCoeff: 新二次项中的系数数组。

len: 系数数组的长度。

返回值

增加的二次项个数。

QuadExpr::Clone()

深度拷贝二次表达式。

概要

```
QuadExpr Clone()
```

返回值

新的二次表达式。

QuadExpr::Evaluate()

求解后对二次表达式估值。

概要

```
double Evaluate()
```

返回值

表达式估值。

QuadExpr::GetCoeff()

获取二次表达式中指定索引值对应项的系数。

概要

```
double GetCoeff(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值对应的二次表达式项的系数。

QuadExpr::GetConstant()

获取二次表达式中的常数。

概要

```
double GetConstant()
```

返回值

二次表达式中的常数。

QuadExpr::GetLinExpr()

获取二次表达式中的线性表达式。

概要

```
Expr &GetLinExpr()
```

返回值

线性表达式对象。

QuadExpr::GetVar1()

获取二次表达式指定索引值对应项中的第一个变量。

概要

```
Var &GetVar1(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值对应项的第一个变量对象。

QuadExpr::GetVar2()

获取二次表达式指定索引值对应项中的第二个变量。

概要

```
Var &GetVar2(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值对应项的第二个变量对象。

QuadExpr::operator*=()

自乘一个常数。

概要

```
void operator*=(double c)
```

参量

`c`: 需要乘以一个常数。

QuadExpr::operator*()

乘以一个常数并返回新的表达式。

概要

```
QuadExpr operator*(double c)
```

参量

c: 乘以的常数。

返回值

结果表达式。

QuadExpr::operator+=()

自增一个表达式。

概要

```
void operator+=(const QuadExpr &expr)
```

参量

expr: 需要增加的表达式。

QuadExpr::operator+()

增加一个表达式并返回新的表达式。

概要

```
QuadExpr operator+(const QuadExpr &other)
```

参量

other: 加上的表达式。

返回值

结果表达式。

QuadExpr::operator-=()

自减一个表达式。

概要

```
void operator-=(const QuadExpr &expr)
```

参量

expr: 需要减去的表达式。

QuadExpr::operator-()

减去一个表达式并返回新的表达式。

概要

```
QuadExpr operator-(const QuadExpr &other)
```

参量

other: 减去的表达式。

返回值

结果表达式。

QuadExpr::Remove()

删除表达式中指定索引值的项。

概要

```
void Remove(int i)
```

参量

i: 指定索引值。

QuadExpr::Remove()

删除表达式中与指定变量相关的项。

概要

```
void Remove(const Var &var)
```

参量

var: 指定变量。

QuadExpr::Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(size_t n)
```

参量

n: 容纳 n 项的空间。

QuadExpr::SetCoeff()

设置二次表达式指定索引值对应项的系数。

概要

```
void SetCoeff(int i, double val)
```

参量

`i`: 指定索引值。

`val`: 指定索引值对应项的参数。

QuadExpr::SetConstant()

设置表达式中的常数。

概要

```
void SetConstant(double constant)
```

参量

`constant`: 表达式中的常数。

QuadExpr::Size()

获取表达式中的项数。

概要

```
size_t Size()
```

返回值

表达式中的项数。

24.4.26 QConstraint 类

QConstraint 类是杉数求解器对二次约束的相关操作的封装，提供了以下成员方法：

QConstraint::Get()

获得二次约束的属性值。支持二次相关的属性。

概要

```
double Get(const char *szAttr)
```

参量

`szAttr`: 所需要获得的属性名称。

返回值

属性值。

QConstraint::GetIdx()

获取二次约束的索引值。

概要

```
int GetIdx()
```

返回值

二次约束的索引值。

QConstraint::GetName()

获取二次约束的名称。

概要

```
const char *GetName()
```

返回值

二次约束的名称。

QConstraint::GetRhs()

获得二次约束的右端值。

概要

```
double GetRhs()
```

返回值

二次约束的右端值。

QConstraint::GetSense()

获得二次约束的类型。

概要

```
char GetSense()
```

返回值

二次约束的类型。

QConstraint::Remove()

从模型中删除当前的二次约束。

概要

```
void Remove()
```

QConstraint::Set()

设置二次约束的属性值。支持二次相关的属性。

概要

```
void Set(const char *szAttr, double value)
```

参量

`szAttr`: 所需要设置的属性名称。

`value`: 所需要设置的属性值。

QConstraint::SetName()

设置二次约束的名称。

概要

```
void SetName(const char *szName)
```

参量

`szName`: 二次约束的名称。

QConstraint::SetRhs()

设置二次约束的右端值。

概要

```
void SetRhs(double rhs)
```

参量

`rhs`: 二次约束的右端值。

QConstraint::SetSense()

设置二次约束的类型。

概要

```
void SetSense(char sense)
```

参量

`sense`: 二次约束的类型。

24.4.27 QConstrArray 类

为方便用户对一组 C++ *QConstraint 类* 对象进行操作，杉数求解器的 C++ 接口设计了 QConstrArray 类，提供了以下成员方法：

QConstrArray::GetQConstr()

获取 QConstrArray 中的指定索引值的 QConstraint。

概要

```
QConstraint &GetQConstr(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 QConstraint。

QConstrArray::PushBack()

向 QConstrArray 中添加一个 QConstraint。

概要

```
void PushBack(const QConstraint &constr)
```

参量

`constr`: 待添加的 QConstraint。

QConstrArray::Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

n: 容纳 n 项的空间。

QConstrArray::Size()

获取 QConstrArray 中的元素个数。

概要

```
int Size()
```

返回值

QConstrArray 中的元素个数。

24.4.28 QConstrBuilder 类

QConstrBuilder 类是杉数优化求解器中对构建二次约束的构建器的封装，提供了以下成员方法：

QConstrBuilder::GetQuadExpr()

获取二次约束相关的表达式。

概要

```
const QuadExpr &GetQuadExpr()
```

返回值

二次表达式对象。

QConstrBuilder::GetSense()

获取二次约束相关约束类型。

概要

```
char GetSense()
```

返回值

约束类型。

QConstrBuilder::Set()

设置一个二次约束的表达式，类型和边界值。

概要

```
void Set(
    const QuadExpr &expr,
    char sense,
    double rhs)
```

参量

`expr`: 约束一侧的二次表达式。

`sense`: 二次约束类型。

`rhs`: 二次约束另一侧的常数项。

24.4.29 QConstrBuilderArray 类

为方便用户对一组 C++ *QConstrBuilder 类* 对象进行操作，杉数优化求解器的 C++ 接口设计了 QConstrBuilderArray 类，提供了以下成员方法：

QConstrBuilderArray::GetBuilder()

获取 QConstrBuilderArray 中的指定索引值的二次约束。

概要

```
QConstrBuilder &GetBuilder(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 ConstrBuilder。

QConstrBuilderArray::PushBack()

向 QConstrBuilderArray 中添加一个二次约束生成器。

概要

```
void PushBack(const QConstrBuilder &builder)
```

参量

`builder`: 待添加的二次约束生成器。

QConstrBuilderArray::Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

`n`: 容纳 n 项的空间。

QConstrBuilderArray::Size()

获取 QConstrBuilderArray 中的元素个数。

概要

```
int Size()
```

返回值

QConstrBuilderArray 中的元素个数。

24.4.30 PsdVar 类

PsdVar 类是杉数优化求解器对半定变量的相关操作的封装，提供了以下成员方法：

PsdVar::Get()

获取半定变量的属性值。

概要

```
double Get(const char *szAttr, int sz)
```

参量

`szAttr`: 属性的名称。

`sz`: 输出数组长度。

返回值

输出双精度型数组，保存了属性值。

PsdVar::GetDim()

获取半定变量的维度。

概要

```
int GetDim()
```

返回值

半定变量的维度。

PsdVar::GetIdx()

获取半定变量的索引值。

概要

```
int GetIdx()
```

返回值

半定变量的索引值。

PsdVar::GetLen()

获取半定变量展开后的长度。

概要

```
int GetLen()
```

返回值

半定变量展开后的长度。

PsdVar::GetName()

获取半定变量的名称。

概要

```
const char *GetName()
```

返回值

半定变量名称。

PsdVar::Remove()

从模型中删除半定变量。

概要

```
void Remove()
```

PsdVar::SetName()

设置半定变量的名称。

概要

```
void SetName(const char *szName)
```

参量

`szName`: 半定变量名称。

24.4.31 PsdVarArray 类

为方便用户对一组*PsdVar 类* 对象进行操作，杉数优化求解器的 C++ 接口设计了 PsdVarArray 类，提供了以下成员方法：

PsdVarArray::GetPsdVar()

获取半定变量数组里指定索引的半定变量。

概要

```
PsdVar &GetPsdVar(int idx)
```

参量

`idx`: 半定变量的索引。

返回值

指定索引的半定变量。

PsdVarArray::PushBack()

向半定变量数组里添加半定变量。

概要

```
void PushBack(const PsdVar &var)
```

参量

`var`: 半定变量。

PsdVarArray::Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

n: 容纳 n 项的空间。

PsdVarArray::Size()

获取半定变量数组里半定变量个数。

概要

```
int Size()
```

返回值

半定变量个数。

24.4.32 PsdExpr 类

COPT 半定表达式包括一个线性表达式，一些半定变量和对应的系数矩阵。PsdExpr 类是杉数求解器中用于构建半定表达式时对半定变量的相关组合操作，提供了以下成员方法：

PsdExpr::PsdExpr()

PsdExpr 的构造函数。

概要

```
PsdExpr(double constant)
```

参量

constant: PsdExpr 中的常值。

PsdExpr::PsdExpr()

使用变量和其系数构造的半定表达式。

概要

```
PsdExpr(const Var &var, double coeff)
```

参量

var: 添加的这一项对应的变量。

coeff: 添加的这一项对应的参数。

PsdExpr::PsdExpr()

使用线性表达式构造的半定表达式。

概要

```
PsdExpr(const Expr &expr)
```

参量

expr: 初始的线性表达式。

PsdExpr::PsdExpr()

使用半定变量和其系数矩阵构造的半定表达式。

概要

```
PsdExpr(const PsdVar &var, const SymMatrix &mat)
```

参量

var: 添加的这一项对应的半定变量。

mat: 添加的这一项对应的系数矩阵。

PsdExpr::PsdExpr()

使用半定变量和其系数矩阵表达式构造的半定表达式。

概要

```
PsdExpr(const PsdVar &var, const SymMatExpr &expr)
```

参量

var: 添加的这一项对应的半定变量。

expr: 添加的这一项对应的系数矩阵表达式。

PsdExpr::AddConstant()

增加半定表达式中的常数。

概要

```
void AddConstant(double constant)
```

参量

constant: 半定表达式中的常数改变量。

PsdExpr::AddLinExpr()

添加一个线性表达式的项，并乘以倍数。

概要

```
void AddLinExpr(const Expr &expr, double mult)
```

参量

`expr`: 需要添加的线性表达式。

`mult`: 可选的系数倍数，默认值为 1.0。

PsdExpr::AddPsdExpr()

添加一个半定表达式的项，并乘以倍数。

概要

```
void AddPsdExpr(const PsdExpr &expr, double mult)
```

参量

`expr`: 需要添加的半定表达式。

`mult`: 可选的系数倍数，默认值为 1.0。

PsdExpr::AddTerm()

向半定表达式中添加一线性项。

概要

```
void AddTerm(const Var &var, double coeff)
```

参量

`var`: 新线性项中的变量。

`coeff`: 新线性项中的系数。

PsdExpr::AddTerm()

向半定表达式中添加一个半定项。

概要

```
void AddTerm(const PsdVar &var, const SymMatrix &mat)
```

参量

`var`: 新半定项中的半定变量。

`mat`: 新半定项中的系数矩阵。

PsdExpr::AddTerm()

向半定表达式中添加一个半定项。

概要

```
void AddTerm(const PsdVar &var, const SymMatExpr &expr)
```

参量

var: 新半定项中的半定变量。

expr: 新半定项中对称矩阵的表达式。

PsdExpr::AddTerms()

向表达式中添加一些线性项。

概要

```
int AddTerms(
    const VarArray &vars,
    double *pCoeff,
    int len)
```

参量

vars: 新线性项中的变量数组。

pCoeff: 新线性项中的系数数组。

len: 系数数组的长度。

返回值

增加的线性项个数。

PsdExpr::AddTerms()

向表达式中添加一些半定项。

概要

```
int AddTerms(const PsdVarArray &vars, const SymMatrixArray &mats)
```

参量

vars: 新半定项中的半定变量数组。

mats: 新半定项中的系数矩阵数组。

返回值

增加的半定项个数。

PsdExpr::Clone()

深度拷贝半定表达式。

概要

```
PsdExpr Clone()
```

返回值

新的半定表达式对象。

PsdExpr::Evaluate()

求解后对半定表达式估值。

概要

```
double Evaluate()
```

返回值

表达式估值。

PsdExpr::GetCoeff()

获取半定表达式中指定索引值对应项的系数。

概要

```
SymMatExpr &GetCoeff(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值对应的半定表达式项的系数。

PsdExpr::GetConstant()

获取半定表达式中的常数项。

概要

```
double GetConstant()
```

返回值

半定表达式中的常数项。

PsdExpr::GetLinExpr()

获取半定表达式中的线性表达式。

概要

```
Expr &GetLinExpr()
```

返回值

线性表达式对象。

PsdExpr::GetPsdVar()

获取半定表达式指定索引值对应项中的半定变量。

概要

```
PsdVar &GetPsdVar(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值对应项的半定变量对象。

PsdExpr::operator*=()

自乘一个常数。

概要

```
void operator*=(double c)
```

参量

`c`: 需要乘以一个常数。

PsdExpr::operator*()

乘以一个常数并返回新的表达式。

概要

```
PsdExpr operator*(double c)
```

参量

`c`: 乘以的常数。

返回值

结果表达式。

PsdExpr::operator+=()

自增一个表达式。

概要

```
void operator+=(const PsdExpr &expr)
```

参量

expr: 需要增加的表达式。

PsdExpr::operator+()

增加一个表达式并返回新的表达式。

概要

```
PsdExpr operator+(const PsdExpr &other)
```

参量

other: 加上的表达式。

返回值

结果表达式。

PsdExpr::operator-=()

自减一个表达式。

概要

```
void operator-=(const PsdExpr &expr)
```

参量

expr: 需要减去的表达式。

PsdExpr::operator-()

减去一个表达式并返回新的表达式。

概要

```
PsdExpr operator-(const PsdExpr &other)
```

参量

other: 减去的表达式。

返回值

结果表达式。

PsdExpr::Remove()

删除半定表达式中指定索引值的项。

概要

```
void Remove(int idx)
```

参量

idx: 指定索引值。

PsdExpr::Remove()

删除半定表达式中与指定变量相关的项。

概要

```
void Remove(const Var &var)
```

参量

var: 指定变量。

PsdExpr::Remove()

删除半定表达式中与指定半定变量相关的项。

概要

```
void Remove(const PsdVar &var)
```

参量

var: 指定半定变量。

PsdExpr::Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(size_t n)
```

参量

n: 容纳 n 项的空间。

PsdExpr::SetCoeff()

设置半定表达式指定索引值对应项的系数矩阵。

概要

```
void SetCoeff(int i, const SymMatrix &mat)
```

参量

`i`: 指定索引值。

`mat`: 指定索引值对应项的系数矩阵。

PsdExpr::SetConstant()

设置半定表达式中的常数。

概要

```
void SetConstant(double constant)
```

参量

`constant`: 半定表达式中的常数。

PsdExpr::Size()

获取半定表达式中的半定项数。

概要

```
size_t Size()
```

返回值

半定表达式中的半定项数。

24.4.33 PsdConstraint 类

PsdConstraint 类是杉数求解器对半定约束的相关操作的封装，提供了以下成员方法：

PsdConstraint::Get()

获得半定约束的属性值。支持半定相关的属性。

概要

```
double Get(const char *szAttr)
```

参量

`szAttr`: 所需要获得的属性名称。

返回值

双精度属性值。

PsdConstraint::GetIdx()

获取半定约束的索引值。

概要

```
int GetIdx()
```

返回值

半定约束的索引值。

PsdConstraint::GetName()

获取半定约束的名称。

概要

```
const char *GetName()
```

返回值

半定约束的名称。

PsdConstraint::Remove()

从模型中删除当前的半定约束。

概要

```
void Remove()
```

PsdConstraint::Set()

设置半定约束的属性值。支持半定相关的属性。

概要

```
void Set(const char *szAttr, double value)
```

参量

`szAttr`: 所需要设置的属性名称。

`value`: 所需要设置的属性值。

PsdConstraint::SetName()

设置半定约束的名称。

概要

```
void SetName(const char *szName)
```

参量

`szName`: 半定约束的名称。

24.4.34 PsdConstrArray 类

为方便用户对一组 C++ *PsdConstraint 类* 对象进行操作，杉数求解器的 C++ 接口设计了 PsdConstrArray 类，提供了以下成员方法：

PsdConstrArray::GetPsdConstr()

获取半定约束数组中的指定索引值的半定约束。

概要

```
PsdConstraint &GetPsdConstr(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的半定约束。

PsdConstrArray::PushBack()

向半定约束数组中添加一个半定约束。

概要

```
void PushBack(const PsdConstraint &constr)
```

参量

`constr`: 待添加的半定约束。

PsdConstrArray::Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

n: 容纳 n 项的空间。

PsdConstrArray::Size()

获取半定约束数组中的元素个数。

概要

```
int Size()
```

返回值

半定约束数组中的元素个数。

24.4.35 PsdConstrBuilder 类

PsdConstrBuilder 类是杉数优化求解器中对构建半定约束的构建器的封装，提供了以下成员方法：

PsdConstrBuilder::GetPsdExpr()

获取半定约束相关的表达式。

概要

```
const PsdExpr &GetPsdExpr()
```

返回值

半定表达式对象。

PsdConstrBuilder::GetRange()

获取半定约束生成器对象的约束范围的长度（从下界到上界的长度，必须大于 0）。

概要

```
double GetRange()
```

返回值

半定约束范围的长度（从下界到上界的长度）。

PsdConstrBuilder::GetSense()

获取半定约束相关约束类型。

概要

```
char GetSense()
```

返回值

半定约束类型。

PsdConstrBuilder::Set()

设置一个半定约束的表达式，类型和边界值。

概要

```
void Set(
    const PsdExpr &expr,
    char sense,
    double rhs)
```

参量

`expr`: 约束一侧的半定表达式。

`sense`: 除了 COPT_RANGE 外的半定约束类型。

`rhs`: 约束另一侧的常数项。

PsdConstrBuilder::SetRange()

设置一个范围约束（带有上下界）。

概要

```
void SetRange(const PsdExpr &expr, double range)
```

参量

`expr`: 半定表达式。其表达式的常数项的负数其实是这个约束的上界。

`range`: 约束范围的长度（从下界到上界的长度，必须大于 0）。

24.4.36 PsdConstrBuilderArray 类

为方便用户对一组 C++ *PsdConstrBuilder 类* 对象进行操作，杉数优化求解器的 C++ 接口设计了 PsdConstrBuilderArray 类，提供了以下成员方法：

PsdConstrBuilderArray::GetBuilder()

获取半定约束生成器数组中的指定索引值的半定约束。

概要

```
PsdConstrBuilder &GetBuilder(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的半定约束生成器。

PsdConstrBuilderArray::PushBack()

向半定约束生成器数组中添加一个半定约束生成器。

概要

```
void PushBack(const PsdConstrBuilder &builder)
```

参量

`builder`: 待添加的半定约束生成器。

PsdConstrBuilderArray::Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

`n`: 容纳 n 项的空间。

PsdConstrBuilderArray::Size()

获取半定约束生成器数组中的元素个数。

概要

```
int Size()
```

返回值

半定约束生成器数组中的元素个数。

24.4.37 LmiConstraint 类

LmiConstraint 类是杉数求解器对 LMI 约束的相关操作的封装，提供了以下成员方法：

LmiConstraint::Get()

获得 LMI 约束的属性值。支持 LMI 相关的属性。

概要

```
double Get(const char *szAttr, int len)
```

参量

`szAttr`: 所需要获得的属性名称。

`len`: 输出数组长度。

返回值

输出的一组双精度属性值。

LmiConstraint::GetDim()

获取 LMI 约束中系数矩阵的维度。

概要

```
int GetDim()
```

返回值

系数矩阵的维度。

LmiConstraint::GetIdx()

获取 LMI 约束的索引值。

概要

```
int GetIdx()
```

返回值

LMI 约束的索引值。

LmiConstraint::GetLen()

获取 LMI 约束展开后的长度。

概要

```
int GetLen()
```

返回值

LMI 约束展开后的长度。

LmiConstraint::GetName()

获取 LMI 约束的名称。

概要

```
const char *GetName()
```

返回值

LMI 约束的名称。

LmiConstraint::Remove()

从模型中删除当前的 LMI 约束。

概要

```
void Remove()
```

LmiConstraint::SetName()

设置 LMI 约束的名称。

概要

```
void SetName(const char *szName)
```

参量

`szName`: LMI 约束的名称。

LmiConstraint::SetRhs()

设置 LMI 约束的常数项。

概要

```
void SetRhs(const SymMatrix &mat)
```

参量

`mat`: 所需要设置的对称矩阵。

24.4.38 LmiConstrArray 类

为方便用户对一组 C++ *LmiConstraint 类* 对象进行操作，杉数求解器的 C++ 接口设计了 LmiConstrArray 类，提供了以下成员方法：

LmiConstrArray::GetLmiConstr()

获取 LMI 约束数组中的指定索引的 LMI 约束。

概要

```
LmiConstraint &GetLmiConstr(int idx)
```

参量

`idx`: 指定的索引。

返回值

指定索引的 LMI 约束。

LmiConstrArray::PushBack()

向 LMI 约束数组中添加一个 LMI 约束。

概要

```
void PushBack(const LmiConstraint &constr)
```

参量

`constr`: 待添加的 LMI 约束。

LmiConstrArray::Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

n: 预分配空间的大小。

LmiConstrArray::Size()

获取 LMI 约束数组中的元素个数。

概要

```
int Size()
```

返回值

LMI 约束数组中的元素个数。

24.4.39 LmiExpr 类

COPT 的 LMI 表达式包括对称矩阵和标量变量相乘。LmiExpr 类是杉数求解器中用于构建 LMI 表达式时对变量和对称矩阵相关组合操作，提供了以下成员方法：

LmiExpr::LmiExpr()

LmiExpr 的默认构造函数。

概要

```
LmiExpr()
```

LmiExpr::LmiExpr()

使用对称矩阵构造的 LMI 表达式。

概要

```
LmiExpr(const SymMatrix &mat)
```

参量

mat: 对称矩阵，其作为 LMI 的常数项。

LmiExpr::LmiExpr()

使用对称矩阵表达式构造的 LMI 表达式。

概要

```
LmiExpr(const SymMatExpr &expr)
```

参量

expr: 对称矩阵表达式，其作为 LMI 的常数项。

LmiExpr::LmiExpr()

使用变量和其系数矩阵构造的 LMI 表达式。

概要

```
LmiExpr(const Var &var, const SymMatrix &mat)
```

参量

var: 添加的这一项对应的变量。

mat: 添加的这一项对应的系数矩阵。

LmiExpr::LmiExpr()

使用变量和其系数矩阵表达式构造的 LMI 表达式。

概要

```
LmiExpr(const Var &var, const SymMatExpr &expr)
```

参量

var: 添加的这一项对应的变量。

expr: 添加的这一项对应的系数矩阵表达式。

LmiExpr::AddConstant()

加到 LMI 表达式中的常数项。

概要

```
void AddConstant(const SymMatExpr &expr)
```

参量

expr: 加到常数项的矩阵表达式。

LmiExpr::AddLmiExpr()

添加一个 LMI 表达式的项，并乘以倍数。

概要

```
void AddLmiExpr(const LmiExpr &expr, double mult)
```

参量

`expr`: 需要添加的 LMI 表达式。

`mult`: 可选的系数倍数，默认值为 1.0。

LmiExpr::AddTerm()

向 LMI 表达式中添加一项。

概要

```
void AddTerm(const Var &var, const SymMatrix &mat)
```

参量

`var`: 新项中的变量。

`mat`: 新项中的系数矩阵。

LmiExpr::AddTerm()

向 LMI 表达式中添加一项。

概要

```
void AddTerm(const Var &var, const SymMatExpr &expr)
```

参量

`var`: 新项中的变量。

`expr`: 新项中作为系数的对称矩阵表达式。

LmiExpr::AddTerms()

向 LMI 表达式中添加一些项。

概要

```
int AddTerms(const VarArray &vars, const SymMatrixArray &mats)
```

参量

`vars`: 新项中的变量数组。

`mats`: 新项中的系数矩阵数组。

返回值

增加的 LMI 项个数。

LmiExpr::Clone()

深度拷贝 LMI 表达式。

概要

```
LmiExpr Clone()
```

返回值

新的 LMI 表达式对象。

LmiExpr::GetCoeff()

获取 LMI 表达式中指定索引值对应项的系数矩阵表达式。

概要

```
SymMatExpr &GetCoeff(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值对应的 LMI 表达式项的系数矩阵表达式。

LmiExpr::GetConstant()

获取 LMI 表达式中的常数项。

概要

```
SymMatExpr &GetConstant()
```

返回值

LMI 表达式中的常数项。

LmiExpr::GetVar()

获取 LMI 表达式指定索引值对应项中的变量。

概要

```
Var &GetVar(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值对应项的变量对象。

LmiExpr::operator*=()

自乘一个常数。

概要

```
void operator*=(double c)
```

参量

c: 自乘的倍数。

LmiExpr::operator*()

乘以一个常数并返回新的表达式。

概要

```
LmiExpr operator*(double c)
```

参量

c: 乘以的倍数。

返回值

结果表达式。

LmiExpr::operator+=()

自增一个对称矩阵或者 LMI 表达式。

概要

```
void operator+=(const LmiExpr &expr)
```

参量

expr: 需要添加的对称矩阵，或者 LMI 表达式。

LmiExpr::operator+()

加上一个对称矩阵或者 LMI 表达式，并返回新的表达式。

概要

```
LmiExpr operator+(const LmiExpr &other)
```

参量

other: 添加的对称矩阵或者 LMI 表达式。

返回值

结果表达式。

LmiExpr::operator-=()

自减一个对称矩阵或者 LMI 表达式。

概要

```
void operator-=(const LmiExpr &expr)
```

参量

expr: 需要减去的对称矩阵，或者 LMI 表达式。

LmiExpr::operator-()

减去一个对称矩阵或者 LMI 表达式，并返回新的表达式。

概要

```
LmiExpr operator-(const LmiExpr &other)
```

参量

other: 减去的对称矩阵或者 LMI 表达式。

返回值

结果表达式。

LmiExpr::Remove()

删除 LMI 表达式中指定索引值的项。

概要

```
void Remove(int idx)
```

参量

idx: 指定索引值。

LmiExpr::Remove()

删除 LMI 表达式中与指定变量相关的项。

概要

```
void Remove(const Var &var)
```

参量

var: 指定变量。

LmiExpr::Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(size_t n)
```

参量

`n`: 预分配空间的大小。

LmiExpr::SetCoeff()

设置 LMI 表达式指定索引值对应项的系数矩阵。

概要

```
void SetCoeff(int i, const SymMatrix &mat)
```

参量

`i`: 指定索引值。

`mat`: 指定索引值对应项的系数矩阵。

LmiExpr::SetConstant()

设置 LMI 表达式中的常数项。

概要

```
void SetConstant(const SymMatrix &mat)
```

参量

`mat`: 常数项对应的矩阵。

LmiExpr::Size()

获取 LMI 表达式中的项数。

概要

```
size_t Size()
```

返回值

LMI 表达式中的项数。

24.4.40 SymMatrix 类

对称矩阵作为半定项中的系数矩阵，常用在半定表达式，半定约束和半定目标函数中。SymMatrix 类是杉数优化求解器中对称矩阵的封装，提供了以下成员方法：

SymMatrix::GetDim()

获取对称矩阵的维度。

概要

```
int GetDim()
```

返回值

对称矩阵的维度。

SymMatrix::GetIdx()

获取对称矩阵的索引值。

概要

```
int GetIdx()
```

返回值

对称矩阵的索引值。

24.4.41 SymMatrixArray 类

为方便用户对一组 C++ *SymMatrix 类* 对象进行操作，杉数求解器的 C++ 接口设计了 SymMatrixArray 类，提供了以下成员方法：

SymMatrixArray::GetMatrix()

获取对称矩阵数组里指定下标的对称矩阵。

概要

```
SymMatrix &GetMatrix(int idx)
```

参量

idx: 对称矩阵的下标。

返回值

指定下标的对称矩阵。

SymMatrixArray::PushBack()

向对称矩阵数组里附加一个对称矩阵。

概要

```
void PushBack(const SymMatrix &mat)
```

参量

`mat`: 对称矩阵。

SymMatrixArray::Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

`n`: 容纳 n 项的空间。

SymMatrixArray::Size()

获取对称矩阵数组里对称矩阵个数。

概要

```
int Size()
```

返回值

对称矩阵个数。

24.4.42 SymMatExpr 类

对称矩阵表达式对于对称矩阵的线性组合，其计算结果实际还是一个对称矩阵。表达式的好处是可以延迟计算结果矩阵，直到设置半定约束或者半定目标函数时。SymMatExpr 类是杉数优化求解器中对称矩阵表达式的封装，提供了以下成员方法：

SymMatExpr::SymMatExpr()

对称矩阵表达式的默认构造函数。

概要

```
SymMatExpr()
```

SymMatExpr::SymMatExpr()

使用对称矩阵和其系数构造的表达式。

概要

```
SymMatExpr(const SymMatrix &mat, double coeff)
```

参量

`mat`: 添加的这一项对应的对称矩阵。

`coeff`: 可选，添加的这一项对应的参数。默认值为 1.0。

SymMatExpr::AddSymMatExpr()

添加一个对称矩阵表达式的项，并乘以倍数。

概要

```
void AddSymMatExpr(const SymMatExpr &expr, double mult)
```

参量

`expr`: 需要添加的对称矩阵表达式。

`mult`: 可选的系数倍数，默认值为 1.0。

SymMatExpr::AddTerm()

向对称矩阵表达式中添加一项。

概要

```
bool AddTerm(const SymMatrix &mat, double coeff)
```

参量

`mat`: 新项中的对称矩阵。

`coeff`: 新项中的系数。

返回值

布尔值，表示新项是否成功添加。

SymMatExpr::AddTerms()

向表达式中添加多个项。

概要

```
int AddTerms(
    const SymMatrixArray &mats,
    double *pCoeff,
```

```
    int len)
```

参量

`mats`: 新项中的对称矩阵数组。

`pCoeff`: 新项中的系数数组。

`len`: 系数数组的长度。

返回值

增加的项数。如果返回负值，至少有一项添加失败。

SymMatExpr::Clone()

深度拷贝对称矩阵表达式。

概要

```
SymMatExpr Clone()
```

返回值

新的表达式对象。

SymMatExpr::GetCoeff()

获取表达式中指定索引值对应项的系数。

概要

```
double GetCoeff(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值对应的表达式项的系数。

SymMatExpr::GetDim()

获取表达式中对称矩阵的维度。

概要

```
int GetDim()
```

返回值

对称矩阵的维度。

SymMatExpr::GetSymMat()

获取表达式指定索引值对应项中的对称矩阵。

概要

```
SymMatrix &GetSymMat(int i)
```

参量

i: 指定索引值。

返回值

指定索引值对应项的对称矩阵对象。

SymMatExpr::operator*=()

自乘一个常数。

概要

```
void operator*=(double c)
```

参量

c: 需要乘以一个常数。

SymMatExpr::operator*()

乘以一个常数并返回新的表达式。

概要

```
SymMatExpr operator*(double c)
```

参量

c: 乘以的常数。

返回值

结果表达式。

SymMatExpr::operator+=()

自增一个对称矩阵表达式。

概要

```
void operator+=(const SymMatExpr &expr)
```

参量

expr: 需要增加的对称矩阵表达式。

SymMatExpr::operator+()

增加一个表达式并返回新的表达式。

概要

```
SymMatExpr operator+(const SymMatExpr &other)
```

参量

`other`: 加上的表达式。

返回值

结果表达式。

SymMatExpr::operator-=()

自减一个对称矩阵表达式。

概要

```
void operator-=(const SymMatExpr &expr)
```

参量

`expr`: 需要减去的对称矩阵表达式。

SymMatExpr::operator-()

减去一个表达式并返回新的表达式。

概要

```
SymMatExpr operator-(const SymMatExpr &other)
```

参量

`other`: 减去的表达式。

返回值

结果表达式。

SymMatExpr::Remove()

删除表达式中指定索引值的项。

概要

```
void Remove(int idx)
```

参量

`idx`: 指定索引值。

SymMatExpr::Remove()

删除对称矩阵表达式中与指定对称矩阵相关的项。

概要

```
void Remove(const SymMatrix &mat)
```

参量

`mat`: 指定的对称矩阵。

SymMatExpr::Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(size_t n)
```

参量

`n`: 容纳 n 项的空间。

SymMatExpr::SetCoeff()

设置表达式指定索引值项数中的系数。

概要

```
void SetCoeff(int i, double val)
```

参量

`i`: 指定索引值。

`val`: 指定索引值项数中的参数。

SymMatExpr::Size()

获取对称矩阵表达式中的项数。

概要

```
size_t Size()
```

返回值

对称矩阵表达式中的项数。

24.4.43 CallbackBase 类

CallbackBase 类给用户提供了在求解过程中介入的接口。这个是抽象类，用户需要自己实现虚函数 `virtual void CallbackBase::callback()` 才能创建实例，用来作为 `Model::SetCallback(ICallback* ptr, int cbctx)` 方法的第一个参数传入。CallbackBase 类也提供了以下可以继承的成员方法：

CallbackBase::AddLazyConstr()

向模型中增加一个惰性约束。

概要

```
void AddLazyConstr(
    const Expr &lhs,
    char sense,
    double rhs)
```

参量

`lhs`: 惰性约束表达式。

`sense`: 惰性约束的类型。

`rhs`: 惰性约束的右侧值。

CallbackBase::AddLazyConstr()

向模型中增加一个惰性约束。

概要

```
void AddLazyConstr(
    const Expr &lhs,
    char sense,
    const Expr &rhs)
```

参量

`lhs`: 惰性约束的左侧表达式。

`sense`: 惰性约束的类型。

`rhs`: 惰性约束的右侧表达式。

CallbackBase::AddLazyConstr()

向模型中增加一个惰性约束。

概要

```
void AddLazyConstr(const ConstrBuilder &builder)
```

参量

builder: 惰性约束生成器。

CallbackBase::AddLazyConstrs()

向模型中增加多个惰性约束。

概要

```
void AddLazyConstrs(const ConstrBuilderArray &builders)
```

参量

builders: 一组惰性约束生成器。

CallbackBase::AddUserCut()

向模型中增加一个割平面。

概要

```
void AddUserCut(
    const Expr &lhs,
    char sense,
    double rhs)
```

参量

lhs: 割平面表达式。

sense: 割平面的类型。

rhs: 割平面的右侧值。

CallbackBase::AddUserCut()

向模型中增加一个割平面。

概要

```
void AddUserCut(
    const Expr &lhs,
    char sense,
```

```
    const Expr &rhs)
```

参量

lhs: 割平面的左侧表达式。

sense: 割平面的类型。

rhs: 割平面的右侧表达式。

CallbackBase::AddUserCut()

向模型中增加一个割平面。

概要

```
void AddUserCut(const ConstrBuilder &builder)
```

参量

builder: 割平面生成器。

CallbackBase::AddUserCuts()

向模型中增加多个割平面。

概要

```
void AddUserCuts(const ConstrBuilderArray &builders)
```

参量

builders: 一组割平面生成器。

CallbackBase::GetDblInfo()

在回调中获取指定信息名的双精度值。

概要

```
double GetDblInfo(const char *cbinfo)
```

参量

cbinfo: 回调中的信息名。

返回值

所需的信息值。

CallbackBase::GetIncumbent()

在回调中获取指定变量的最优可行解。

概要

```
double GetIncumbent(Var &var)
```

参量

var: 指定的变量。

返回值

变量对应的最优可行解。

CallbackBase::GetIncumbent()

在回调中获取一组变量的最优可行解。

概要

```
int GetIncumbent(VarArray &vars, double *pOut)
```

参量

vars: 变量数组。

pOut: 一组变量对应的最优可行解解。

返回值

被查询的有效变量数目。如果出错，返回-1。

CallbackBase::GetIncumbent()

在回调中获取全部变量的最优可行解。

概要

```
int GetIncumbent(double *pOut, int len)
```

参量

pOut: 可选，全部变量对应的最优可行解。

len: 输出变量的数组长度。如果小于实际变量个数，最多输出到长度 len 为止。

返回值

全部变量个数。错误情况下返回-1.

CallbackBase::GetIntInfo()

在回调中获取指定信息名的整数值。

概要

```
int GetIntInfo(const char *cbinfo)
```

参量

`cbinfo`: 回调中的信息名。

返回值

所需的信息值。

CallbackBase::GetRelaxSol()

在回调中获取指定变量的线性松弛解。

概要

```
double GetRelaxSol(Var &var)
```

参量

`var`: 指定的变量。

返回值

变量对应的线性松弛解。

CallbackBase::GetRelaxSol()

在回调中获取一组变量的线性松弛解。

概要

```
int GetRelaxSol(VarArray &vars, double *pOut)
```

参量

`vars`: 变量数组。

`pOut`: 变量对应的线性松弛解。

返回值

被查询的有效变量数目。如果出错，返回-1。

CallbackBase::GetRelaxSol()

在回调中获取全部变量的线性松弛解。

概要

```
int GetRelaxSol(double *pOut, int len)
```

参量

pOut: 可选，全部变量对应的线性松弛解。

len: 输出变量的数组长度。如果小于实际变量个数，最多输出到长度 len 为止。

返回值

全部变量个数。错误情况下返回-1.

CallbackBase::GetSolution()

在回调中获取指定变量的解。

概要

```
double GetSolution(Var &var)
```

参量

var: 指定的变量。

返回值

变量对应的解。

CallbackBase::GetSolution()

在回调中获取一组变量的解。

概要

```
int GetSolution(VarArray &vars, double *pOut)
```

参量

vars: 变量数组。

pOut: 一组变量对应的解。

返回值

被查询的有效变量数目。如果出错，返回-1。

CallbackBase::GetSolution()

在回调中获取全部变量的解。

概要

```
int GetSolution(double *pOut, int len)
```

参量

`pOut`: 可选，全部变量对应的解。

`len`: 输出变量的数组长度。如果小于实际变量个数，最多输出到长度 len 为止。

返回值

全部变量个数。错误情况下返回-1.

CallbackBase::Interrupt()

中断回调中正在求解的问题。

概要

```
void Interrupt()
```

CallbackBase::LoadSolution()

向模型中添加自定义解。

概要

```
double LoadSolution()
```

返回值

解对应的目标函数值。

CallbackBase::SetSolution()

在回调中对给定的变量设置自定义解。

概要

```
void SetSolution(Var &var, double val)
```

参量

`var`: 变量对象。

`val`: 双精度值。

CallbackBase::SetSolution()

在回调中对一组变量设置自定义解。

概要

```
void SetSolution(
    VarArray &vars,
    const double *vals,
    int len)
```

参量

vars: 变量数组。

vals: 双精度值数组。

len: 双精度值数组长度。

CallbackBase::Where()

获取回调中的上下文。

概要

```
int Where()
```

返回值

上下文的整数值。

24.4.44 ProbBuffer 类

ProbBuffer 类是字符流缓冲区的封装，提供了以下成员方法：

ProbBuffer::ProbBuffer()

ProbBuffer 的构造函数。

概要

```
ProbBuffer(int sz)
```

参量

sz: ProbBuffer 的初始大小

ProbBuffer::GetData()

获取在缓存里的问题字符流。

概要

```
char *GetData()
```

返回值

缓存中的问题字符流。

ProbBuffer::Resize()

调整缓存到给定大小。

概要

```
void Resize(int sz)
```

参量

sz: 指定的缓存大小。

ProbBuffer::Size()

获取缓存大小。

概要

```
int Size()
```

返回值

缓存大小。

第 25 章　C# API 参考手册

为了方便用户对复杂应用场景进行建模并调用求解器进行求解，杉数求解器设计并提供了 C# 接口，本章节将对 C# 接口的常量和功能进行阐述。

25.1 C# 常量类

常量类定义了使用 C# 接口建模时必要的常数，包括一般常数、解状态、属性、信息和参数五大类常量。本节将依次阐述上述五类常量的内容与使用方法。

25.1.1 一般常数

一般常数定义在 `Consts` 类里．用户可以通过 `Copt` 命名空间的前缀访问一般常数，如 `Copt.Consts.XXXX`。

一般常数类中的内容，详见一般常数章节。

25.1.2 解状态

关于解状态的常数定义在 `Status` 类里．用户可以通过 `Copt` 命名空间的前缀访问解状态常数，如 `Copt.Status.XXXX`。

解状态常数中的内容，详见一般常数章节：求解状态。

25.1.3 属性

C# 属性常数中的内容，详见属性章节。

在 C# API 中，COPT 属性定义在 `DblAttr` 和 `IntAttr` 类里。用户可以通过 `Copt.DblAttr` 访问 COPT 浮点型属性；通过 `Copt.IntAttr` 访问 COPT 整型属性。

在 C# API 中，通过指定属性名称获取属性取值，提供的函数如下，具体请参考*C# Model 类* 。

- `Model.GetIntAttr()`：获取整数型属性取值
- `Model.GetDblAttr()`：获取浮点型属性取值

25.1.4 信息

C# API 信息常数中的内容，详见信息章节。

在 C# API 中，信息常数定义在 `DblInfo` 类里，用户可以通过命名空间中的前缀 `Copt`（通常可以省略）`Copt.DblInfo.` 访问信息常数。

例如，`Copt.DblInfo.Obj` 表示变量在目标函数中系数。

25.1.5 Callback 信息

C# API Callback 信息常数中的内容，详见信息章节：*Callback* 相关信息。

在 C# API 中，Callback 相关的信息常数定义在 `CbInfo` 类里，用户可以通过命名空间中的前缀 `Copt`（通常可以省略）`Copt.CbInfo.` 访问 Callback 信息常数。

例如，`Copt.CbInfo.BestObj` 表示当前最优目标函数值。

25.1.6 参数

优化参数控制 COPT 求解器优化算法的行为。

C# 参数常数中的内容，详见参数章节。

在 C# API 中，COPT 优化参数定义在 `DblParam` 和 `IntParam` 类里。用户可以通过 `Copt.DblParam` 访问 COPT 浮点型参数；通过 `Copt.IntParam` 访问 COPT 整型参数。

在 C# API 中，通过指定参数名称获取和设置参数取值。提供的函数如下，具体请参考*C# Model 类* 。

- 获取指定参数的详细信息（当前值/最大值/最小值）：`Model.GetParamInfo()`
- 获取指定整数型/浮点型参数当前取值：`Model.GetIntParam()` / `Model.GetDblParam()`
- 设置指定整数型/浮点型参数值：`Model.SetIntParam()` / `Model.SetDblParam()`

25.2 C# 优化建模类

本章节详细描述杉数优化求解器的 C# 接口的优化建模类，方便用户在快速构建复杂场景下的优化模型时对其功能和使用的查询。

25.2.1 Envr 类

创建求解环境对象是每个求解过程中必不可少的第一步。而每个求解模型都和一个环境类 Envr 关联。用户必须首先创建一个求解环境，才能在此基础上创建一个或者多个求解模型。

Envr.Envr()

COPT Envr 类的构造函数。

概要

```
Envr()
```

Envr.Envr()

COPT Envr 类的构造函数。

概要

```
Envr(string licDir)
```

参量

`licDir`: 用户指定的路径，包含本地授权文档或者客户端配置文件。

Envr.Envr()

COPT Envr 类的构造函数。

概要

```
Envr(EnvrConfig config)
```

参量

`config`: COPT Envr 配置类，包含远程连接的设置。

Envr.Close()

关闭远程连接。之前获得的远程授权失效，对当前环境类下创建的全部问题立即生效。

概要

```
void Close()
```

Envr.CreateModel()

创建模型对象。

概要

```
Model CreateModel(string name)
```

参量

`name`: 自定义的模型名称。

返回值

模型对象。

25.2.2 EnvrConfig 类

如果用户通过连接远程服务的方式启动杉数优化求解器，可以创建环境配置类来设置 COPT 作为客户端的配置。

EnvrConfig.EnvrConfig()

Envr 配置类的构造函数。

概要

```
EnvrConfig()
```

EnvrConfig.Set()

设置 Envr 配置类里的内容。

概要

```
void Set(string name, string value)
```

参量

`name`: 配置的关键词。

`value`: 配置的内容。

25.2.3 Model 类

Model 类是杉数优化求解器模型相关操作的封装，提供了以下成员方法：

Model.Model()

模型的构造函数。

概要

```
Model(Envr env, string name)
```

参量

`env`: 关联的环境对象。

`name`: 模型名。

Model.AddCone()

向模型中增加一个锥约束。

概要

```
Cone AddCone(
    int dim,
    int type,
    char[] pvtype,
    string prefix)
```

参量

dim: 锥约束的维度。

type: 锥约束的类型。

pvtype: 锥约束中变量的类型。

prefix: 可选，锥约束中变量的名称前缀，默认值为“ConeV”。

返回值

新的锥约束。

Model.AddCone()

向模型中增加一个锥约束。

概要

```
Cone AddCone(ConeBuilder builder)
```

参量

builder: 锥约束生成器。

返回值

新锥约束。

Model.AddCone()

向模型中增加一个锥约束。

概要

```
Cone AddCone(Var[] vars, int type)
```

参量

vars: 参与锥约束的变量数组。

type: 锥约束的类型。

返回值

新的锥约束。

Model.AddCone()

向模型中增加一个锥约束。

概要

```
Cone AddCone(VarArray vars, int type)
```

参量

vars: 参与锥约束的变量构成的 VarArray 类。

type: 锥约束的类型。

返回值

新的锥约束。

Model.AddConstr()

向模型中增加一个线性约束。

概要

```
Constraint AddConstr(
    Expr expr,
    char sense,
    double rhs,
    string name)
```

参量

expr: 新的约束表达式。

sense: 约束的类型。

rhs: 新约束的右侧值。

name: 可选，新约束的名称。

返回值

新约束。

Model.AddConstr()

向模型中增加一个线性约束。

概要

```
Constraint AddConstr(
    Expr expr,
    char sense,
    Var var,
    string name)
```

参量

expr: 新的约束表达式。

sense: 约束的类型。

var: 作为右侧值的变量，而不是约束范围类型。

name: 可选，新约束的名称。

返回值

新约束。

Model.AddConstr()

向模型中增加一个线性约束。

概要

```
Constraint AddConstr(
    Expr lhs,
    char sense,
    Expr rhs,
    string name)
```

参量

lhs: 新约束的左侧值。

sense: 约束的类型。

rhs: 新约束的右侧值。

name: 可选，新约束的名称。

返回值

新约束。

Model.AddConstr()

向模型中增加一个线性约束。

概要

```
Constraint AddConstr(
    Expr expr,
    double lb,
    double ub,
    string name)
```

参量

expr: 新的约束表达式。

lb: 约束的下界。

ub: 约束的上界。

name: 可选，新约束的名称。

返回值

新约束。

Model.AddConstr()

向模型中增加一个线性约束。

概要

```
Constraint AddConstr(ConstrBuilder builder, string name)
```

参量

builder: 新约束生成器。

name: 可选，新约束的名称。

返回值

新约束。

Model.AddConstrs()

向模型中增加线性约束。

概要

```
ConstrArray AddConstrs(
    int count,
    char[] senses,
```

```
    double[] rhss,
    string prefix)
```

参量

`count`: 添加的线性约束数目。

`senses`: 约束类型的数组，而不是范围类型。

`rhss`: 新约束的右侧值。

`prefix`: 新约束的名称前缀，默认值为 “R”。

返回值

新约束构成的 ConstrArray 类。

Model.AddConstrs()

向模型中增加线性约束。

概要

```
ConstrArray AddConstrs(
    int count,
    double[] lbs,
    double[] ubs,
    string prefix)
```

参量

`count`: 添加的线性约束数目。

`lbs`: 新约束的下界数组。

`ubs`: 新约束的上界数组。

`prefix`: 可选，新约束的名称前缀，默认值为 “R”。

返回值

新约束构成的 ConstrArray 类。

Model.AddConstrs()

向模型中增加线性约束。

概要

```
ConstrArray AddConstrs(ConstrBuilderArray builders, string prefix)
```

参量

`builders`: 线性约束生成器。

`prefix`: 可选，新约束的名称前缀，默认值为 “R”。

返回值

新约束构成的 ConstrArray 类。

Model.AddDenseMat()

向模型中增加一个密致对称矩阵。

概要

```
SymMatrix AddDenseMat(int dim, double[] vals)
```

参量

`dim`: 密致对称矩阵的维度。

`vals`: 非零元值数组。按列次序填充非零元，到数组长度或者对称矩阵最大长度位置。

返回值

新对称矩阵对象。

Model.AddDenseMat()

向模型中增加一个密致对称矩阵。

概要

```
SymMatrix AddDenseMat(int dim, double val)
```

参量

`dim`: 密致对称矩阵的维度。

`val`: 同一个非零元值，用来填充对称矩阵。

返回值

新对称矩阵对象。

Model.AddDiagMat()

向模型中增加一个对角矩阵。

概要

```
SymMatrix AddDiagMat(int dim, double val)
```

参量

`dim`: 对角矩阵的维度。

`val`: 同一个非零元值，用来填充对角元素。

返回值

新对角矩阵对象。

Model.AddDiagMat()

向模型中增加一个对角矩阵。

概要

```
SymMatrix AddDiagMat(int dim, double[] vals)
```

参量

`dim`: 对角矩阵的维度。

`vals`: 双精度值数组，用来填充对角元素。

返回值

新对角矩阵对象。

Model.AddDiagMat()

向模型中增加一个对角矩阵。

概要

```
SymMatrix AddDiagMat(
    int dim,
    double val,
    int offset)
```

参量

`dim`: 对角矩阵的维度。

`val`: 同一个非零元值，用来填充对角元素。

`offset`: 相对于标准对角线的平移量。

返回值

新对角矩阵对象。

Model.AddDiagMat()

向模型中增加一个对角矩阵。

概要

```
SymMatrix AddDiagMat(
    int dim,
    double[] vals,
    int offset)
```

参量

`dim`: 对角矩阵的维度。

`vals`: 双精度值数组，用来填充对角元素。

`offset`: 相对于标准对角线的平移量。

返回值

新对角矩阵对象。

Model.AddEyeMat()

向模型中增加一个单位矩阵。

概要

```
SymMatrix AddEyeMat(int dim)
```

参量

`dim`: 单位矩阵的维度。

返回值

新单位矩阵对象。

Model.AddGenConstrIndicator()

向模型中增加一个类型指示型的一般约束（GenConstr）。

概要

```
GenConstr AddGenConstrIndicator(GenConstrBuilder builder)
```

参量

`builder`: 一般约束（GenConstr）生成器。

返回值

类型指示型的新一般约束（GenConstr）。

Model.AddGenConstrIndicator()

向模型中增加一个类型指示型的一般约束（GenConstr）。

概要

```
GenConstr AddGenConstrIndicator(
    Var binvar,
    int binval,
    ConstrBuilder builder)
```

参量

`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 或 1）。

`builder`: 线性约束生成器。

返回值

类型指示型的新一般约束（GenConstr）。

Model.AddGenConstrIndicator()

向模型中增加一个类型指示型的一般约束（GenConstr）。

概要

```
GenConstr AddGenConstrIndicator(
    Var binvar,
    int binval,
    Expr expr,
    char sense,
    double rhs)
```

参量

`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 或 1）。

`expr`: 新的线性约束表达式。

`sense`: 新的线性约束类型。

`rhs`: 新的线性约束右侧值。

返回值

类型指示型的新一般约束（GenConstr）。

Model.AddLazyConstr()

向模型中增加一个惰性约束。

概要

```
void AddLazyConstr(
    Expr lhs,
    char sense,
    double rhs,
    string name)
```

参量

lhs: 惰性约束表达式。

sense: 惰性约束的类型。

rhs: 惰性约束的右侧值。

name: 可选，新惰性约束的名称。

Model.AddLazyConstr()

向模型中增加一个惰性约束。

概要

```
void AddLazyConstr(
    Expr lhs,
    char sense,
    Expr rhs,
    string name)
```

参量

lhs: 惰性约束的左侧表达式。

sense: 惰性约束的类型。

rhs: 惰性约束的右侧表达式。

name: 可选，新惰性约束的名称。

Model.AddLazyConstr()

向模型中增加一个惰性约束。

概要

```
void AddLazyConstr(ConstrBuilder builder, string name)
```

参量

builder: 惰性约束生成器。

name: 可选，新惰性约束的名称。

Model.AddLazyConstrs()

向模型中增加多个惰性约束。

概要

```
void AddLazyConstrs(ConstrBuilderArray builders, string prefix)
```

参量

`builders`: 一组惰性约束生成器。

`prefix`: 新惰性约束名称的前缀。

Model.AddLmiConstr()

向模型中增加一个 LMI 约束。

概要

```
LmiConstraint AddLmiConstr(LmiExpr expr, string name)
```

参量

`expr`: 新的 LMI 约束表达式。

`name`: 可选，新 LMI 约束的名称。

返回值

新 LMI 约束对象。

Model.AddOnesMat()

向模型中增加一个用非零元 1 填充的密致对称矩阵。

概要

```
SymMatrix AddOnesMat(int dim)
```

参量

`dim`: 密致对称矩阵的维度。

返回值

新对称矩阵对象。

Model.AddPsdConstr()

向模型中增加一个半定约束。

概要

```
PsdConstraint AddPsdConstr(
    PsdExpr expr,
    char sense,
    double rhs,
    string name)
```

参量

expr: 新的半定约束表达式。

sense: 半定约束的类型。

rhs: 新半定约束的右侧值。

name: 可选，新半定约束的名称。

返回值

新半定约束对象。

Model.AddPsdConstr()

向模型中增加一个半定约束。

概要

```
PsdConstraint AddPsdConstr(
    PsdExpr expr,
    double lb,
    double ub,
    string name)
```

参量

expr: 新的半定约束表达式。

lb: 半定约束的下界。

ub: 半定约束的上界。

name: 可选，新半定约束的名称。

返回值

新半定约束。

Model.AddPsdConstr()

向模型中增加一个半定约束。

概要

```
PsdConstraint AddPsdConstr(
    PsdExpr lhs,
    char sense,
    PsdExpr rhs,
    string name)
```

参量

lhs: 新半定约束的左侧表达式。

sense: 半定约束的类型。

rhs: 新半定约束的右侧表达式。

name: 可选，新半定约束的名称。

返回值

新半定约束对象。

Model.AddPsdConstr()

向模型中增加一个半定约束。

概要

```
PsdConstraint AddPsdConstr(PsdConstrBuilder builder, string name)
```

参量

builder: 新半定约束生成器。

name: 可选，新半定约束的名称。

返回值

新半定约束对象。

Model.AddPsdVar()

向模型中增加半定变量。

概要

```
PsdVar AddPsdVar(int dim, string name)
```

参量

dim: 新半定变量的维度。

name: 新半定变量的名称。

返回值

新半定变量对象。

Model.AddPsdVars()

向模型中添加一些半定变量。

概要

```
PsdVarArray AddPsdVars(
    int count,
    int[] dims,
    string prefix)
```

参量

count: 新半定变量的数目。

dims: 整数数组，包含了半定变量的维度。

prefix: 新半定变量的名称前缀，默认前缀是 PSD_V。

返回值

新添加的半定变量数组。

Model.AddQConstr()

向模型中增加一个二次约束。

概要

```
QConstraint AddQConstr(
    QuadExpr expr,
    char sense,
    double rhs,
    string name)
```

参量

expr: 新的二次约束表达式。

sense: 约束的类型。

rhs: 新约束的右侧值。

name: 可选，新二次约束的名称。

返回值

新二次约束对象。

Model.AddQConstr()

向模型中增加一个二次约束。

概要

```
QConstraint AddQConstr(
    QuadExpr lhs,
    char sense,
    QuadExpr rhs,
    string name)
```

参量

lhs: 新二次约束的左侧表达式。

sense: 二次约束的类型。

rhs: 新二次约束的右侧表达式。

name: 可选，新二次约束的名称。

返回值

新二次约束对象。

Model.AddQConstr()

向模型中增加一个二次约束。

概要

```
QConstraint AddQConstr(QConstrBuilder builder, string name)
```

参量

builder: 新二次约束生成器。

name: 可选，新二次约束的名称。

返回值

新二次约束对象。

Model.AddSos()

向模型中增加一个 SOS 约束。

概要

```
Sos AddSos(
    Var[] vars,
    double[] weights,
    int type)
```

参量

`vars`: 参与 SOS 约束的变量数组。

`weights`: 参与 SOS 约束变量的权重数组。

`type`: SOS 约束的类型。

返回值

新 SOS 约束。

Model.AddSos()

向模型中增加一个 SOS 约束。

概要

```
Sos AddSos(
    VarArray vars,
    double[] weights,
    int type)
```

参量

`vars`: 参与 SOS 约束的变量构成的 VarArray 类。

`weights`: 参与 SOS 约束变量的权重数组。

`type`: SOS 约束的类型。

返回值

新 SOS 约束。

Model.AddSos()

向模型中增加一个 SOS 约束。

概要

```
Sos AddSos(SosBuilder builder)
```

参量

`builder`: SOS 约束生成器。

返回值

新 SOS 约束。

Model.AddSparseMat()

向模型中增加一个稀疏对称矩阵。

概要

```
SymMatrix AddSparseMat(
    int dim,
    int nElems,
    int[] rows,
    int[] cols,
    double[] vals)
```

参量

`dim`: 稀疏对称矩阵的维度。

`nElems`: 稀疏对称矩阵中的非零元个数。

`rows`: 整数数组，保存了非零元的行号。

`cols`: 整数数组，保存了非零元的列号。

`vals`: 非零元值数组。

返回值

新对称矩阵对象。

Model.AddSymMat()

根据给定的对称矩阵表达式，向模型中增加一个对称矩阵。

概要

```
SymMatrix AddSymMat(SymMatExpr expr)
```

参量

expr: 对称矩阵表达式对象。

返回值

结果对称矩阵对象。

Model.AddUserCut()

向模型中增加一个割平面。

概要

```
void AddUserCut(
    Expr lhs,
    char sense,
    double rhs,
    string name)
```

参量

lhs: 割平面表达式。

sense: 割平面的类型。

rhs: 割平面的右侧值。

name: 可选，新割平面的名称。

Model.AddUserCut()

向模型中增加一个割平面。

概要

```
void AddUserCut(
    Expr lhs,
    char sense,
    Expr rhs,
    string name)
```

参量

lhs: 割平面的左侧表达式。

sense: 割平面的类型。

rhs: 割平面的右侧表达式。

name: 可选，新割平面的名称。

Model.AddUserCut()

向模型中增加一个割平面。

概要

```
void AddUserCut(ConstrBuilder builder, string name)
```

参量

builder: 割平面生成器。

name: 可选，新割平面的名称。

Model.AddUserCuts()

向模型中增加多个割平面。

概要

```
void AddUserCuts(ConstrBuilderArray builders, string prefix)
```

参量

builders: 一组割平面生成器。

prefix: 新割平面名称的前缀。

Model.AddVar()

向模型中增加一个变量以及相关不为零的系数作为列。

概要

```
Var AddVar(
    double lb,
    double ub,
    double obj,
    char vtype,
    string name)
```

参量

lb: 新变量的下界。

ub: 新变量的上界。

obj: 新变量在目标函数里的系数。

vtype: 新变量的类型。

name: 可选，新变量的名称。

返回值

新变量。

Model.AddVar()

向模型中增加一个变量以及相关不为零的系数作为列。

概要

```
Var AddVar(
    double lb,
    double ub,
    double obj,
    char vtype,
    Column col,
    string name)
```

参量

lb: 新变量的下界。

ub: 新变量的上界。

obj: 新变量在目标函数里的系数。

vtype: 新变量的类型。

col: 列对象，用于指定新变量所属的一组约束.

name: 可选，新变量的名称。

返回值

新变量。

Model.AddVars()

向模型中增加变量。

概要

```
VarArray AddVars(
    int count,
    char vtype,
    string prefix)
```

参量

`count`: 添加变量的数量。

`vtype`: 新变量的类型。

`prefix`: 可选，新变量的名称前缀，默认为 C。

返回值

新变量构成的 VarArray 类。

Model.AddVars()

向模型中增加变量。

概要

```
VarArray AddVars(
    int count,
    double lb,
    double ub,
    double obj,
    char vtype,
    string prefix)
```

参量

`count`: 添加变量的数量。

`lb`: 新变量的下界。

`ub`: 新变量的上界。

`obj`: 新变量在目标函数里的系数。

`vtype`: 新变量的类型。

`prefix`: 可选，新变量的名称前缀，默认为”C”。

返回值

新变量构成的 VarArray 类。

Model.AddVars()

向模型中增加变量。

概要

```
VarArray AddVars(
    int count,
    double[] lbs,
    double[] ubs,
    double[] objs,
    char[] types,
    string prefix)
```

参量

count: 添加变量的数量。

lbs: 新变量的下界数组, 如果为空，下界为 0。

ubs: 新变量的上界数组, 如果为空，上界为正无穷或者 1 对于二进制变量。

objs: 新变量在目标函数中的系数数组，如果为空，则为 0。

types: 新变量的类型，如果为空，则为连续变量。

prefix: 可选，新变量的名称前缀，默认为 C。

返回值

新变量构成的 VarArray 类。

Model.AddVars()

向模型中增加变量。

概要

```
VarArray AddVars(
    double[] lbs,
    double[] ubs,
    double[] objs,
    char[] types,
    Column[] cols,
    string prefix)
```

参量

lbs: 新变量的下界数组，如果为空，下界为 0。

ubs: 新变量的上界数组，如果为空，上界为正无穷或者 1 对于二进制变量。

objs: 新变量在目标函数中的系数数组，如果为空，则为 0。

types: 新变量的类型，如果为空，则为连续变量。

cols: 列对象数组，用于指定每个新变量所属的一组约束。

prefix: 可选，新变量的名称前缀，默认为 C。

返回值

新变量构成的 VarArray 类。

Model.AddVars()

向模型中增加变量。

概要

```
VarArray AddVars(
    double[] lbs,
    double[] ubs,
    double[] objs,
    char[] types,
    ColumnArray cols,
    string prefix)
```

参量

lbs: 新变量的下界数组，如果为空，下界为 0。

ubs: 新变量的上界数组，如果为空，上界为正无穷或者 1 对于二进制变量。

objs: 新变量在目标函数中的系数数组，如果为空，则为 0。

types: 新变量的类型，如果为空，则为连续变量。

cols: 列对象构成的 ColumnArray 类，用于指定每个新变量所属的一组约束。

prefix: 可选，新变量的名称前缀，默认为 C。

返回值

新变量构成的 VarArray 类。

Model.Clear()

删除所有设置以及问题本身。

概要

```
void Clear()
```

Model.Clone()

深度复制 COPT 模型。

概要

```
Model Clone()
```

返回值

新的 COPT 模型对象。

Model.ComputeIIS()

计息模型的 IIS。

概要

```
void ComputeIIS()
```

Model.DelPsdObj()

删除模型目标函数的半定部分（保留线性部分）。

概要

```
void DelPsdObj()
```

Model.DelQuadObj()

删除模型目标函数的二次部分（保留线性部分）。

概要

```
void DelQuadObj()
```

Model.FeasRelax()

计算不可行模型的可行化松弛。

概要

```
void FeasRelax(
    VarArray vars,
    double[] colLowPen,
    double[] colUppPen,
    ConstrArray cons,
    double[] rowBndPen,
    double[] rowUppPen)
```

参量

vars: 变量构成的 VarArray 类对象。

colLowPen: 变量下界的惩罚系数。

colUppPen: 变量上界的惩罚系数。

cons: 约束构成的 ConstrArray 类对象。

rowBndPen: 约束右端项的惩罚系数。

rowUppPen: 约束上界的惩罚系数。

Model.FeasRelax()

计算不可行模型的可行化松弛。

概要

```
void FeasRelax(int ifRelaxVars, int ifRelaxCons)
```

参量

ifRelaxVars: 是否松弛变量。

ifRelaxCons: 是否松弛约束。

Model.Get()

查询与指定变量相关的双精度型信息的值。

概要

```
double[] Get(string name, Var[] vars)
```

参量

name: 双精度型信息的名称。

vars: 指定变量数组。

返回值

信息的值。

Model.Get()

查询与指定变量相关的双精度型信息的值。

概要

```
double[] Get(string name, VarArray vars)
```

参量

name: 双精度型信息的名称。

vars: 指定变量构成的 VarArray 类。

返回值

信息的值。

Model.Get()

查询与指定约束相关的双精度型信息的值。

概要

```
double[] Get(string name, Constraint[] constrs)
```

参量

name: 双精度型信息的名称。

constrs: 指定约束数组。

返回值

信息的值。

Model.Get()

查询与指定约束相关的双精度型信息的值。

概要

```
double[] Get(string name, ConstrArray constrs)
```

参量

name: 双精度型信息的名称。

constrs: 指定约束构成的 ConstrArray 类。

返回值

信息的值。

Model.Get()

查询与指定二次约束相关的双精度型信息的值。

概要

```
double[] Get(string name, QConstraint[] constrs)
```

参量

name: 双精度型信息的名称。

constrs: 指定二次约束数组。

返回值

信息的值。

Model.Get()

查询与指定二次约束相关的双精度型信息的值。

概要

```
double[] Get(string name, QConstrArray constrs)
```

参量

name: 双精度型信息的名称。

constrs: 指定二次约束构成的 QConstrArray 类。

返回值

信息的值。

Model.Get()

查询与指定半定约束相关的信息的值。

概要

```
double[] Get(string name, PsdConstraint[] constrs)
```

参量

name: 信息的名称。

constrs: 指定半定约束数组。

返回值

输出双精度型数组，保存了信息的值。

Model.Get()

查询与指定半定约束相关的信息的值。

概要

```
double[] Get(string name, PsdConstrArray constrs)
```

参量

`name`: 信息的名称。

`constrs`: 指定半定约束数组。

返回值

输出双精度型数组，保存了信息的值。

Model.GetCoeff()

获取变量在约束中的系数。

概要

```
double GetCoeff(Constraint constr, Var var)
```

参量

`constr`: 指定的约束。

`var`: 指定的变量。

返回值

指定的变量在约束中的系数。

Model.GetCol()

获取一个包含指定变量参与的所有约束的列。

概要

```
Column GetCol(Var var)
```

参量

`var`: 指定变量。

返回值

一个关于指定变量的列。

Model.GetColBasis()

获取列的基状态。

概要

```
int[] GetColBasis()
```

返回值

列的基状态。

Model.GetCone()

获取指定索引值的锥约束。

概要

```
Cone GetCone(int idx)
```

参量

`idx`: 指定下标值。

返回值

想获取的锥约束。

Model.GetConeBuilders()

获取模型中所有锥约束生成器。

概要

```
ConeBuilderArray GetConeBuilders()
```

返回值

锥约束生成器构成的 ConeBuilderArray 类。

Model.GetConeBuilders()

获取给定锥约束的生成器。

概要

```
ConeBuilderArray GetConeBuilders(Cone[] cones)
```

参量

`cones`: 锥约束数组。

返回值

想获取的的锥约束生成器构成的 ConeBuilderArray 类。

Model.GetConeBuilders()

获取给定锥约束的生成器。

概要

```
ConeBuilderArray GetConeBuilders(ConeArray cones)
```

参量

`cones`: 锥约束构成的 ConeArray 类。

返回值

想获取的的锥约束生成器构成的 ConeBuilderArray 类。

Model.GetCones()

获取模型中所有锥约束。

概要

```
ConeArray GetCones()
```

返回值

锥约束构成的 ConeArray 类。

Model.GetConstr()

获取模型中指定索引值的约束。

概要

```
Constraint GetConstr(int idx)
```

参量

`idx`: 指定索引值。

返回值

想获取的约束。

Model.GetConstrBuilder()

获取指定约束的生成器，包括变量，相关的系数，类型和 RHS。

概要

```
ConstrBuilder GetConstrBuilder(Constraint constr)
```

参量

`constr`: 指定约束。

返回值

约束生成器类。

Model.GetConstrBuilders()

获取模型所有约束生成器。

概要

```
ConstrBuilderArray GetConstrBuilders()
```

返回值

约束生成器构成的 ConstrBuilderArray 类。

Model.GetConstrByName()

获取模型中指定名称的约束。

概要

```
Constraint GetConstrByName(string name)
```

参量

`name`: 指定名称。

返回值

想获取的约束。

Model.GetConstrLowerIIS()

获取约束下界的 IIS 状态。

概要

```
int[] GetConstrLowerIIS(ConstrArray constrs)
```

参量

`constrs`: 指定约束构成的 ConstrArray 类对象。

返回值

约束下界的 IIS 状态。

Model.GetConstrLowerIIS()

获取约束下界的 IIS 状态。

概要

```
int[] GetConstrLowerIIS(Constraint[] constrs)
```

参量

`constrs`: 指定约束构成的 ConstrArray 类对象。

返回值

约束下界的 IIS 状态。

Model.GetConstrs()

获取模型所有约束。

概要

```
ConstrArray GetConstrs()
```

返回值

约束构成的 ConstrArray 类。

Model.GetConstrUpperIIS()

获取约束上界的 IIS 状态。

概要

```
int[] GetConstrUpperIIS(ConstrArray constrs)
```

参量

`constrs`: 指定约束构成的 ConstrArray 类对象。

返回值

约束上界的 IIS 状态。

Model.GetConstrUpperIIS()

获取约束上界的 IIS 状态。

概要

```
int[] GetConstrUpperIIS(Constraint[] constrs)
```

参量

`constrs`: 指定约束构成的 ConstrArray 类对象。

返回值

约束上界的 IIS 状态。

Model.GetDblAttr()

获取 COPT 双精度型属性的值。

概要

```
double GetDblAttr(string attr)
```

参量

`attr`: 双精度型属性的名称。

返回值

双精度型属性的值。

Model.GetDblParam()

获取 COPT 双精度型参数的值。

概要

```
double GetDblParam(string param)
```

参量

`param`: 双精度型参数的名称。

返回值

双精度型参数的值。

Model.GetGenConstrIndicator()

获取给定类型指示类一般约束（GenConstr）的生成器。

概要

```
GenConstrBuilder GetGenConstrIndicator(GenConstr indicator)
```

参量

`indicator`: 类型指示类一般约束（GenConstr）。

返回值

类型指示类一般约束（GenConstr）的生成器。

Model.GetIndicatorIIS()

获取 Indicator 约束的 IIS 状态。

概要

```
int[] GetIndicatorIIS(GenConstrArray genconstrs)
```

参量

`genconstrs`: 指定 Indicator 约束构成的 GenConstrArray 类对象。

返回值

Indicator 约束的 IIS 状态。

Model.GetIndicatorIIS()

获取 Indicator 约束的 IIS 状态。

概要

```
int[] GetIndicatorIIS(GenConstr[] genconstrs)
```

参量

`genconstrs`: 指定 Indicator 约束构成的 GenConstrArray 类对象。

返回值

Indicator 约束的 IIS 状态。

Model.GetIntAttr()

获取 COPT 整型属性的值。

概要

```
int GetIntAttr(string attr)
```

参量

`attr`: 整型属性的名称。

返回值

整型属性的值。

Model.GetIntParam()

获取 COPT 整型参数的值。

概要

```
int GetIntParam(string param)
```

参量

`param`: 整型参数的名称。

返回值

整型参数的值。

Model.GetLmiCoeff()

获取 LMI 约束中指定变量的系数矩阵。

概要

```
SymMatrix GetLmiCoeff(LmiConstraint constr, Var var)
```

参量

`constr`: 给定的 LMI 约束。

`var`: 指定的变量。

返回值

指定变量的系数矩阵。

Model.GetLmiConstr()

获取模型中指定索引的 LMI 约束。

概要

```
LmiConstraint GetLmiConstr(int idx)
```

参量

`idx`: 指定的索引值。

返回值

LMI 约束对象。

Model.GetLmiConstrByName()

获取模型中指定名称的 LMI 约束。

概要

```
LmiConstraint GetLmiConstrByName(string name)
```

参量

`name`: 指定 LMI 约束的名称。

返回值

LMI 约束对象。

Model.GetLmiConstrs()

获取模型所有 LMI 约束。

概要

```
LmiConstrArray GetLmiConstrs()
```

返回值

LMI 约束构成的 LmiConstrArray 对象。

Model.GetLmiRhs()

获取 LMI 约束中常数项矩阵。

概要

```
SymMatrix GetLmiRhs(LmiConstraint constr)
```

参量

`constr`: 给定的 LMI 约束。

返回值

对应的常数项矩阵。

Model.GetLmiRow()

获取参与给定 LMI 约束的 LMI 表达式，包括变量和相关系数矩阵。

概要

```
LmiExpr GetLmiRow(LmiConstraint constr)
```

参量

`constr`: 给定的 LMI 约束。

返回值

LMI 约束对应的表达式对象。

Model.GetLpSolution()

获取 LP 解。

概要

```
void GetLpSolution(
    out double[] value,
    out double[] slack,
    out double[] rowDual,
    out double[] redCost)
```

参量

value: 输出，解的值数组。

slack: 输出，约束取值的数组。

rowDual: 输出，约束对偶值的数组。

redCost: 输出，变量对偶值的数组。

Model.GetObjective()

获取模型的目标函数的线性表达式。

概要

```
Expr GetObjective()
```

返回值

线性表达式。

Model.GetParamInfo()

获取当下的，默认的，最小的，最大的 COPT 整型参数的值。

概要

```
void GetParamInfo(
    string name,
    out int cur,
    out int def,
    out int min,
    out int max)
```

参量

name: 整型参数的名称。

cur: 整型参数的当前值。

def: 整型参数的默认值。

min: 整型参数的最小值。

max: 整型参数的最大值。

Model.GetParamInfo()

获取当下的，默认的，最小的，最大的 COPT 双精度型参数的值。

概要

```
void GetParamInfo(
    string name,
    out double cur,
    out double def,
    out double min,
    out double max)
```

参量

name: 双精度型参数的名称。

cur: 参数的当前值。

def: 双精度型参数的默认值。

min: 双精度型参数的最小值。

max: 双精度型参数的最大值。

Model.GetPoolObjVal()

从解池中获取第 idx 个解的目标函数值。

概要

```
double GetPoolObjVal(int idx)
```

参量

idx: 解的编号。

返回值

指定的目标函数值。

Model.GetPoolSolution()

从解池中获取第 idx 个解。

概要

```
double[] GetPoolSolution(int idx, VarArray vars)
```

参量

idx: 解的编号。

vars: 指定的变量。

返回值

指定的解。

Model.GetPoolSolution()

从解池中获取第 idx 个解。

概要

```
double[] GetPoolSolution(int idx, Var[] vars)
```

参量

idx: 解的编号。

vars: 指定的变量。

返回值

指定的解。

Model.GetPsdCoeff()

获取半定约束中指定半定变量的系数矩阵。

概要

```
SymMatrix GetPsdCoeff(PsdConstraint constr, PsdVar var)
```

参量

constr: 给定的半定约束。

var: 指定的半定变量。

返回值

对应的半定变量的系数矩阵。

Model.GetPsdConstr()

获取模型中指定索引值的半定约束。

概要

```
PsdConstraint GetPsdConstr(int idx)
```

参量

`idx`: 指定的索引值。

返回值

半定约束对象。

Model.GetPsdConstrBuilder()

获取指定约束的生成器，包括半定变量，类型和相关的系数矩阵。

概要

```
PsdConstrBuilder GetPsdConstrBuilder(PsdConstraint constr)
```

参量

`constr`: 指定的半定约束。

返回值

半定约束生成器对象。

Model.GetPsdConstrBuilders()

获取模型所有半定约束生成器。

概要

```
PsdConstrBuilderArray GetPsdConstrBuilders()
```

返回值

半定约束生成器构成的 PsdConstrBuilderArray 对象。

Model.GetPsdConstrByName()

获取模型中指定名称的半定约束。

概要

```
PsdConstraint GetPsdConstrByName(string name)
```

参量

`name`: 指定半定约束的名称。

返回值

半定约束对象。

Model.GetPsdConstrs()

获取模型所有半定约束。

概要

```
PsdConstrArray GetPsdConstrs()
```

返回值

半定约束构成的 PsdConstrArray 对象。

Model.GetPsdObjective()

获取模型目标函数的半定目标。

概要

```
PsdExpr GetPsdObjective()
```

返回值

半定目标对应的表达式对象。

Model.GetPsdRow()

获取指定半定约束对应的半定表达式，包括半定变量和相关系数矩阵。

概要

```
PsdExpr GetPsdRow(PsdConstraint constr)
```

参量

`constr`: 指定的半定约束。

返回值

半定约束的表达式对象。

Model.GetPsdSolution()

获取半定变量与半定约束的解。

概要

```
void GetPsdSolution(
    out double[] psdValue,
    out double[] psdSlack,
    out double[] psdRowDual,
    out double[] psdRedCost)
```

参量

`psdValue`: 输出，半定变量的解数组。

`psdSlack`: 输出，半定约束取值数组。

`psdRowDual`: 输出，半定约束的对偶解数组。

`psdRedCost`: 输出，半定变量的对偶解数组。

Model.GetPsdVar()

获取模型中指定索引值的半定变量。

概要

```
PsdVar GetPsdVar(int idx)
```

参量

`idx`: 索引值。

返回值

想获取的半定变量。

Model.GetPsdVarByName()

获取模型中指定名称的半定变量。

概要

```
PsdVar GetPsdVarByName(string name)
```

参量

`name`: 指定名称。

返回值

想获取的半定变量。

Model.GetPsdVars()

获取模型所有半定变量。

概要

```
PsdVarArray GetPsdVars()
```

返回值

半定变量构成的 PsdVarArray 对象。

Model.GetQConstr()

获取模型中指定索引值的二次约束。

概要

```
QConstraint GetQConstr(int idx)
```

参量

idx: 指定索引值。

返回值

想获取的二次约束。

Model.GetQConstrBuilder()

获取指定约束的生成器，包括变量，相关的系数，类型和 RHS。

概要

```
QConstrBuilder GetQConstrBuilder(QConstraint constr)
```

参量

constr: 指定约束。

返回值

约束生成器类。

Model.GetQConstrBuilders()

获取模型所有约束生成器。

概要

```
QConstrBuilderArray GetQConstrBuilders()
```

返回值

约束生成器构成的 QConstrBuilderArray 类。

Model.GetQConstrByName()

获取模型中指定名称的二次约束。

概要

```
QConstraint GetQConstrByName(string name)
```

参量

name: 指定二次约束的名称。

返回值

想获取的二次约束对象。

Model.GetQConstrs()

获取模型所有二次约束。

概要

```
QConstrArray GetQConstrs()
```

返回值

二次约束构成的 QConstrArray 类对象。

Model.GetQuadObjective()

获取模型目标函数的二次目标。

概要

```
QuadExpr GetQuadObjective()
```

返回值

二次目标对应的表达式对象。

Model.GetQuadRow()

获取参与指定二次约束的变量，以及相关系数。

概要

```
QuadExpr GetQuadRow(QConstraint constr)
```

参量

`constr`: 指定二次约束。

返回值

二次约束的表达式。

Model.GetRow()

获取参与指定约束的变量，以及相关的系数。

概要

```
Expr GetRow(Constraint constr)
```

参量

`constr`: 指定约束。

返回值

约束表达式。

Model.GetRowBasis()

获取行的基状态。

概要

```
int[] GetRowBasis()
```

返回值

行的基状态。

Model.GetSolution()

获取 MIP 解决方案。

概要

```
double[] GetSolution()
```

返回值

解的值数组。

Model.GetSos()

获取指定索引值的 SOS 约束。

概要

```
Sos GetSos(int idx)
```

参量

`idx`: 指定索引值。

返回值

想获取的 SOS 约束。

Model.GetSosBuilders()

获取模型中所有 SOS 约束生成器。

概要

```
SosBuilderArray GetSosBuilders()
```

返回值

SOS 约束生成器构成的 SosBuilderArray 类。

Model.GetSosBuilders()

获取给定 SOS 约束的生成器。

概要

```
SosBuilderArray GetSosBuilders(Sos[] soss)
```

参量

`soss`: SOS 约束数组。

返回值

想获取的的 SOS 约束生成器构成的 SosBuilderArray 类。

Model.GetSosBuilders()

获取给定 SOS 约束的生成器。

概要

```
SosBuilderArray GetSosBuilders(SosArray soss)
```

参量

`soss`: SOS 约束构成的 SosArray 类。

返回值

想获取的的 SOS 约束生成器构成的 SosBuilderArray 类。

Model.GetSOSIIS()

获取 SOS 约束的 IIS 状态。

概要

```
int[] GetSOSIIS(SosArray soss)
```

参量

`soss`: 指定 SOS 约束构成的 SosArray 类对象。

返回值

SOS 约束的 IIS 状态。

Model.GetSOSIIS()

获取 SOS 约束的 IIS 状态。

概要

```
int[] GetSOSIIS(Sos[] soss)
```

参量

soss: 指定 SOS 约束构成的 SosArray 类对象。

返回值

SOS 约束的 IIS 状态。

Model.GetSoss()

获取模型中所有 SOS 约束。

概要

```
SosArray GetSoss()
```

返回值

SOS 约束构成的 SosArray 类。

Model.GetSymMat()

获取模型中指定索引值的对称矩阵。

概要

```
SymMatrix GetSymMat(int idx)
```

参量

idx: 指定索引值。

返回值

想获取的对称矩阵。

Model.GetVar()

获取模型中指定索引值的变量。

概要

```
Var GetVar(int idx)
```

参量

idx: 索引值。

返回值

想获取的变量。

Model.GetVarByName()

获取模型中指定名称的变量。

概要

```
Var GetVarByName(string name)
```

参量

`name`: 指定名称。

返回值

想获取的变量。

Model.GetVarLowerIIS()

获取变量下界的 IIS 状态。

概要

```
int[] GetVarLowerIIS(VarArray vars)
```

参量

`vars`: 指定变量构成的 VarArray 类对象。

返回值

变量下界的 IIS 状态。

Model.GetVarLowerIIS()

获取变量下界的 IIS 状态。

概要

```
int[] GetVarLowerIIS(Var[] vars)
```

参量

`vars`: 指定变量构成的 VarArray 类对象。

返回值

变量下界的 IIS 状态。

Model.GetVars()

获取模型中所有的变量。

概要

```
VarArray GetVars()
```

返回值

变量构成的 VarArray 类。

Model.GetVarUpperIIS()

获取变量上界的 IIS 状态。

概要

```
int[] GetVarUpperIIS(VarArray vars)
```

参量

`vars`: 指定变量构成的 VarArray 类对象。

返回值

变量上界的 IIS 状态。

Model.GetVarUpperIIS()

获取变量上界的 IIS 状态。

概要

```
int[] GetVarUpperIIS(Var[] vars)
```

参量

`vars`: 指定变量构成的 VarArray 类对象。

返回值

变量上界的 IIS 状态。

Model.Interrupt()

中断当前问题的求解。

概要

```
void Interrupt()
```

Model.LoadMipStart()

为问题里的变量加载最终初始值。

概要

```
void LoadMipStart()
```

Model.LoadTuneParam()

加载指定的调优参数到模型。

概要

```
void LoadTuneParam(int idx)
```

参量

idx: 调优参数的下标。

Model.Read()

从文件中读取问题，解决方案，基，MIP start 或者 COPT 参数。

概要

```
void Read(string filename)
```

参量

filename: 输入的文件名。

Model.ReadBasis()

从文件中读取基。

概要

```
void ReadBasis(string filename)
```

参量

filename: 输入的文件名。

Model.ReadBin()

从文件中读取 COPT 二进制格式的问题。

概要

```
void ReadBin(string filename)
```

参量

filename: 输入的文件名。

Model.ReadCbf()

从文件中读取 CBF 格式的问题。

概要

```
void ReadCbf(string filename)
```

参量

`filename`: 输入的文件名。

Model.ReadLp()

从文件中读取 LP 格式的问题。

概要

```
void ReadLp(string filename)
```

参量

`filename`: 输入的文件名。

Model.ReadMps()

从文件中读取 MPS 格式的问题。

概要

```
void ReadMps(string filename)
```

参量

`filename`: 输入的文件名。

Model.ReadMst()

从文件中读取 MIP start 信息。

概要

```
void ReadMst(string filename)
```

参量

`filename`: 输入的文件名。

Model.ReadParam()

从文件中读取 COPT 参数。

概要

```
void ReadParam(string filename)
```

参量

`filename`: 输入的文件名。

Model.ReadSdpa()

从文件中读取 SDPA 格式的问题。

概要

```
void ReadSdpa(string filename)
```

参量

`filename`: 输入的文件名。

Model.ReadSol()

从文件中读取解决方案。

概要

```
void ReadSol(string filename)
```

参量

`filename`: 输入的文件名。

Model.ReadTune()

从文件中读取调优参数。

概要

```
void ReadTune(string filename)
```

参量

`filename`: 输入的文件名。

Model.Remove()

从模型中删除一系列变量。

概要

```
void Remove(Var[] vars)
```

参量

`vars`: 变量数组。

Model.Remove()

从模型中删除一系列变量。

概要

```
void Remove(VarArray vars)
```

参量

`vars`: 变量构成的 VarArray 对象。

Model.Remove()

从模型中删除一系列约束。

概要

```
void Remove(Constraint[] constrs)
```

参量

`constrs`: 约束数组。

Model.Remove()

从模型中删除一系列约束。

概要

```
void Remove(ConstrArray constrs)
```

参量

`constrs`: 约束构成的 ConstrArray 对象。

Model.Remove()

从模型中删除一系列 SOS 约束。

概要

```
void Remove(Sos[] soss)
```

参量

`soss`: SOS 约束数组。

Model.Remove()

从模型中删除一系列 SOS 约束。

概要

```
void Remove(SosArray soss)
```

参量

`soss`: SOS 约束构成的 SosArray 对象。

Model.Remove()

从模型中删除一系列二阶锥约束。

概要

```
void Remove(Cone[] cones)
```

参量

`cones`: 二阶锥约束数组。

Model.Remove()

从模型中删除一系列二阶锥约束。

概要

```
void Remove(ConeArray cones)
```

参量

`cones`: 二阶锥约束构成的 ConeArray 对象。

Model.Remove()

从模型中删除一系列一般约束。

概要

```
void Remove(GenConstr[] genConstrs)
```

参量

genConstrs: 一般约束数组。

Model.Remove()

从模型中删除一系列一般约束。

概要

```
void Remove(GenConstrArray genConstrs)
```

参量

genConstrs: 一般约束构成的 GenConstrArray 对象。

Model.Remove()

从模型中删除一些二次约束。

概要

```
void Remove(QConstraint[] qconstrs)
```

参量

qconstrs: 二次约束数组。

Model.Remove()

从模型中删除一些二次约束。

概要

```
void Remove(QConstrArray qconstrs)
```

参量

qconstrs: 二次约束构成的 QConstrArray 对象。

Model.Remove()

从模型中删除一批半定变量。

概要

```
void Remove(PsdVar[] vars)
```

参量

`vars`: 半定变量数组。

Model.Remove()

从模型中删除一批半定变量。

概要

```
void Remove(PsdVarArray vars)
```

参量

`vars`: 半定变量构成的 PsdVarArray 对象。

Model.Remove()

从模型中删除一批半定约束。

概要

```
void Remove(PsdConstraint[] constrs)
```

参量

`constrs`: 半定约束数组。

Model.Remove()

从模型中删除一批半定约束。

概要

```
void Remove(PsdConstrArray constrs)
```

参量

`constrs`: 半定约束构成的 PsdConstrArray 对象。

Model.Remove()

从模型中删除一批 LMI 约束。

概要

```
void Remove(LmiConstrArray constrs)
```

参量

`constrs`: LMI 约束构成的 LmiConstrArray 对象。

Model.Remove()

从模型中删除一批 LMI 约束。

概要

```
void Remove(LmiConstraint[] constrs)
```

参量

`constrs`: LMI 约束数组。

Model.Reset()

只重新设置解。

概要

```
void Reset()
```

Model.ResetAll()

重新设置解与其他信息。

概要

```
void ResetAll()
```

Model.ResetParam()

重新设置参数为默认值。

概要

```
void ResetParam()
```

Model.Set()

设置与指定变量相关的双精度型信息的值。

概要

```
void Set(
    string name,
    Var[] vars,
    double[] vals)
```

参量

name: 双精度型信息的名称。

vars: 指定变量数组。

vals: 信息的值。

Model.Set()

设置与指定变量相关的双精度型信息的值。

概要

```
void Set(
    string name,
    VarArray vars,
    double[] vals)
```

参量

name: 双精度型信息的名称。

vars: 指定变量构成的 VarArray 类。

vals: 信息的值。

Model.Set()

设置与指定约束相关的双精度型信息的值。

概要

```
void Set(
    string name,
    Constraint[] constrs,
    double[] vals)
```

参量

name: 双精度型信息的名称。

constrs: 指定约束数组。

vals: 信息的值。

Model.Set()

设置与指定约束相关的双精度型信息的值。

概要

```
void Set(
    string name,
    ConstrArray constrs,
    double[] vals)
```

参量

name: 双精度型信息的名称。

constrs: 指定约束构成的 ConstrArray 类。

vals: 信息的值。

Model.Set()

设置与指定半定约束相关的双精度型信息的值。

概要

```
void Set(
    string name,
    PsdConstraint[] constrs,
    double[] vals)
```

参量

name: 双精度型信息的名称。

constrs: 指定半定约束构成的 PsdConstrArray 类。

vals: 双精度型数组，保存了信息的值。

Model.Set()

设置与指定半定约束相关的双精度型信息的值。

概要

```
void Set(
    string name,
    PsdConstrArray constrs,
    double[] vals)
```

参量

`name`: 双精度型信息的名称。

`constrs`: 指定半定约束构成的 PsdConstrArray 类。

`vals`: 双精度型数组，保存了信息的值。

Model.SetBasis()

设置行和列的基状态。

概要

```
void SetBasis(int[] colbasis, int[] rowbasis)
```

参量

`colbasis`: 列的基状态。

`rowbasis`: 行的基状态。

Model.SetCallback()

在 COPT 模型中设置用户自定义回调。

概要

```
void SetCallback(CallbackBase cb, int cbctx)
```

参量

`cb`: 用户自定义回调对象，继承 CallbackBase 类。

`cbctx`: COPT 回调上下文。

Model.SetCoeff()

设置变量的系数。

概要

```
void SetCoeff(
    Constraint constr,
    Var var,
    double newVal)
```

参量

constr: 指定的约束。

var: 指定的变量。

newVal: 新系数。

Model.SetDblParam()

设置 COPT 双精度型参数的值。

概要

```
void SetDblParam(string param, double val)
```

参量

param: 双精度型参数的名称。

val: 双精度型参数的值。

Model.SetIntParam()

设置 COPT 整型参数的值。

概要

```
void SetIntParam(string param, int val)
```

参量

param: 整型参数的名称。

val: 整型参数的值。

Model.SetLmiCoeff()

设置 LMI 约束中指定变量的系数矩阵。

概要

```
void SetLmiCoeff(
    LmiConstraint constr,
    Var var,
    SymMatrix mat)
```

参量

`constr`: 给定的 LMI 约束。

`var`: 指定的变量。

`mat`: 新系数矩阵。

Model.SetLmiRhs()

设置 LMI 约束中指定变量的常数项矩阵。

概要

```
void SetLmiRhs(LmiConstraint constr, SymMatrix mat)
```

参量

`constr`: 给定的 LMI 约束。

`mat`: 新常数项矩阵。

Model.SetLpSolution()

设置 LP 解。

概要

```
void SetLpSolution(
    double[] value,
    double[] slack,
    double[] rowDual,
    double[] redCost)
```

参量

`value`: 解的值数组。

`slack`: 约束取值的数组。

`rowDual`: 约束对偶值的数组。

redCost: 变量对偶值的数组。

Model.SetMipStart()

设置给定数目变量的初始值，从第一个开始。

概要

```
void SetMipStart(int count, double[] vals)
```

参量

count: 设置变量的数量。

vals: 变量的值。

Model.SetMipStart()

设置指定变量的初始值。

概要

```
void SetMipStart(Var var, double val)
```

参量

var: 指定变量。

val: 变量的初始值。

Model.SetMipStart()

设置一系列变量的初始值。

概要

```
void SetMipStart(Var[] vars, double[] vals)
```

参量

vars: 指定变量数组。

vals: 变量的初始值。

Model.SetMipStart()

设置一系列变量的初始值。

概要

```
void SetMipStart(VarArray vars, double[] vals)
```

参量

vars: 指定变量构成的 VarArray 类。

vals: 变量的初始值。

Model.SetObjConst()

设置目标函数里的的常数。

概要

```
void SetObjConst(double constant)
```

参量

constant: 常数的值。

Model.SetObjective()

设置模型的目标函数。

概要

```
void SetObjective(Expr expr, int sense)
```

参量

expr: 目标函数的表达式。

sense: 可选，默认值 0 不改变模型当前的优化方向。

Model.SetObjSense()

设置目标函数的优化方向。

概要

```
void SetObjSense(int sense)
```

参量

sense: 目标函数的优化方向。

Model.SetPsdCoeff()

设置半定约束中指定半定变量的系数矩阵。

概要

```
void SetPsdCoeff(
    PsdConstraint constr,
    PsdVar var,
    SymMatrix mat)
```

参量

constr: 给定的半定约束。

var: 指定的半定变量。

mat: 新系数矩阵。

Model.SetPsdObjective()

设置模型的半定目标。

概要

```
void SetPsdObjective(PsdExpr expr, int sense)
```

参量

expr: 模型目标函数的半定表达式。

sense: 可选，默认值 0 表示不改变模型当前的优化方向。

Model.SetQuadObjective()

设置模型的二次目标。

概要

```
void SetQuadObjective(QuadExpr expr, int sense)
```

参量

expr: 模型目标函数的二次表达式。

sense: 默认值 0 表示不改变模型当前的优化方向。

Model.SetSlackBasis()

设置松弛状态。

概要

```
void SetSlackBasis()
```

Model.SetSolverLogFile()

为 COPT 设置日志文件。

概要

```
void SetSolverLogFile(string filename)
```

参量

filename: 日志文件名。

Model.Solve()

求解当前的 MIP 问题。

概要

```
void Solve()
```

Model.SolveLp()

求解当前的 LP 问题。

概要

```
void SolveLp()
```

Model.Tune()

模型调优。

概要

```
void Tune()
```

Model.Write()

向文件中输出问题，解决方案，基，MIP start 或者改变后的 COPT 参数。

概要

```
void Write(string filename)
```

参量

`filename`: 输出的文件名。

Model.WriteBasis()

将最优基输出到“.bas”文件。

概要

```
void WriteBasis(string filename)
```

参量

`filename`: 输出的文件名

Model.WriteBin()

将问题以 COPT 二进制格式输出到文件中。

概要

```
void WriteBin(string filename)
```

参量

`filename`: 输出的文件名。

Model.WriteIIS()

将 IIS 输出到文件中。

概要

```
void WriteIIS(string filename)
```

参量

`filename`: 输出文件名。

Model.WriteLp()

将问题以 LP 格式输出到文件中。

概要

```
void WriteLp(string filename)
```

参量

`filename`: 输出的文件名。

Model.WriteMps()

将问题以 MPS 格式输出到文件中。

概要

```
void WriteMps(string filename)
```

参量

`filename`: 输出的文件名。

Model.WriteMpsStr()

将问题以 MPS 格式输出到问题缓存中。

概要

```
ProbBuffer WriteMpsStr()
```

返回值

输出的 MPS 问题缓存对象。

Model.WriteMst()

将 MIP start 信息输出到“.mst”文件。

概要

```
void WriteMst(string filename)
```

参量

`filename`: 输出的文件名。

Model.WriteParam()

将更改后的 COPT 参数输出到“.par”文件。

概要

```
void WriteParam(string filename)
```

参量

`filename`: 输出的文件名。

Model.WritePoolSol()

将指定的解池中的结果输出到“.sol”文件。

概要

```
void WritePoolSol(int idx, string filename)
```

参量

`idx`: 解池中解的索引。

`filename`: 输出的文件名。

Model.WriteRelax()

将可行化松弛模型输出到文件中。

概要

```
void WriteRelax(string filename)
```

参量

`filename`: 输出文件名。

Model.WriteSol()

将解决方案输出到“.sol”文件。

概要

```
void WriteSol(string filename)
```

参量

`filename`: 输出的文件名。

Model.WriteTuneParam()

将指定的调优参数输出到“.par”文件。

概要

```
void WriteTuneParam(int idx, string filename)
```

参量

`idx`: 调优参数下标。

`filename`: 输出的文件名。

25.2.4 Var 类

Var 类是杉数优化求解器变量的相关操作的封装，提供了以下成员方法：

Var.Get()

获取变量的属性值。支持”Value”, ”RedCost”, ”LB”, ”UB”, and ”Obj” 属性。

概要

```
double Get(string attr)
```

参量

`attr`: 属性的名称。

返回值

属性的值。

Var.GetBasis()

获取变量的基状态。

概要

```
int GetBasis()
```

返回值

变量的基状态。

Var.GetIdx()

获取变量的索引值。

概要

```
int GetIdx()
```

返回值

索引值。

Var.GetLowerIIS()

获取变量下界的 IIS 状态。

概要

```
int GetLowerIIS()
```

返回值

变量下界的 IIS 状态。

Var.GetName()

获取变量的名称。

概要

```
string GetName()
```

返回值

变量名称。

Var.GetType()

获取变量的类型。

概要

```
char GetType()
```

返回值

变量类型。

Var.GetUpperIIS()

获取变量上界的 IIS 状态。

概要

```
int GetUpperIIS()
```

返回值

变量上界的 IIS 状态。

Var.Remove()

从模型中删除变量。

概要

```
void Remove()
```

Var.Set()

设置变量的属性值。支持”LB”, ”UB” 和”Obj” 属性。

概要

```
void Set(string attr, double val)
```

参量

`attr`: 属性的名称。

`val`: 新的值。

Var.SetName()

设置变量的名称。

概要

```
void SetName(string name)
```

参量

`name`: 变量名称。

Var.SetType()

设置变量的类型。

概要

```
void SetType(char vtype)
```

参量

`vtype`: 变量类型。

25.2.5 VarArray 类

为方便用户对一组 C# *Var 类* 对象进行操作，杉数优化求解器的 C# 接口设计了 VarArray 类，提供了以下成员方法：

VarArray.VarArray()

VarArray 的构造函数。

概要

```
VarArray()
```

VarArray.GetVar()

获取 VarArray 类中指定索引值的变量。

概要

```
Var GetVar(int idx)
```

参量

`idx`: 指定索引值。

返回值

指定索引值的变量。

VarArray.PushBack()

向 VarArray 类中添加一个变量。

概要

```
void PushBack(Var var)
```

参量

`var`: 变量。

VarArray.Size()

获取 VarArray 类中变量的数目。

概要

```
int Size()
```

返回值

VarArray 类中变量的数目。

25.2.6 Expr 类

Expr 类是杉数求解器中用于构建线性表达式时变量的相关组合操作，提供了以下成员方法：

Expr.Expr()

线性表达式的构造函数。

概要

```
Expr(double constant)
```

参量

`constant`: 表达式对象中的常量。

Expr.Expr()

只有一项的 Expr 的构造函数。

概要

```
Expr(Var var, double coeff)
```

参量

`var`: 添加的这一项对应的变量。

`coeff`: 添加的这一项对应的参数。

Expr.AddConstant()

增加表达式中的常数。

概要

```
void AddConstant(double constant)
```

参量

`constant`: 表达式中的常数改变量。

Expr.AddExpr()

添加一个表达式的项，并乘以倍数。

概要

```
void AddExpr(Expr expr, double mult)
```

参量

`expr`: 需要添加的表达式。

`mult`: 系数倍数。

Expr.AddTerm()

向表达式中添加一项。

概要

```
void AddTerm(Var var, double coeff)
```

参量

`var`: 新项中的变量。

`coeff`: 新项中的系数。

Expr.AddTerms()

向表达式中添加项。

概要

```
void AddTerms(Var[] vars, double coeff)
```

参量

`vars`: 新项中的变量数组。

`coeff`: 新项中的系数。

Expr.AddTerms()

向表达式中添加项。

概要

```
void AddTerms(Var[] vars, double[] coeffs)
```

参量

vars: 新项中的变量数组。

coeffs: 新项中的系数数组。

Expr.AddTerms()

向表达式中添加项。

概要

```
void AddTerms(VarArray vars, double coeff)
```

参量

vars: 新项中的变量构成的 VarArray 类。

coeff: 新项中的系数。

Expr.AddTerms()

向表达式中添加项。

概要

```
void AddTerms(VarArray vars, double[] coeffs)
```

参量

vars: 新项中的变量构成的 VarArray 类。

coeffs: 新项中的系数数组。

Expr.Clone()

深拷贝表达式。

概要

```
Expr Clone()
```

返回值

复制的表达式对象。

Expr.GetCoeff()

获取表达式指定索引值项数中的系数。

概要

```
double GetCoeff(int i)
```

参量

i: 指定索引值。

返回值

指定索引值项数中的系数。

Expr.GetConstant()

获取表达式中的常数。

概要

```
double GetConstant()
```

返回值

表达式中的常数。

Expr.GetVar()

获取表达式指定索引值项数中的变量。

概要

```
Var GetVar(int i)
```

参量

i: 指定索引值。

返回值

指定索引值项数中的变量。

Expr.Remove()

删除表达式中指定索引值的项。

概要

```
void Remove(int idx)
```

参量

idx: 指定索引值。

Expr.Remove()

删除表达式中与指定变量相关的项。

概要

```
void Remove(Var var)
```

参量

var: 指定变量。

Expr.SetCoeff()

获取表达式指定索引值项数中的系数。

概要

```
void SetCoeff(int i, double val)
```

参量

i: 指定索引值。

val: 指定索引值项数中的参数。

Expr.SetConstant()

设置表达式中的常数。

概要

```
void SetConstant(double constant)
```

参量

constant: 表达式中的常数。

Expr.Size()

获取表达式中的项数。

概要

```
long Size()
```

返回值

表达式中的项数。

25.2.7 Constraint 类

Constraint 类是杉数求解器线性约束的相关操作的封装，提供了以下成员方法：

Constraint.Get()

获得 Constraint 的属性值。支持"Dual", "Slack", "LB", "UB" 属性。

概要

```
double Get(string attr)
```

参量

`attr`: 所需要获得的属性名称。

返回值

属性值。

Constraint.GetBasis()

获得 Constraint 的基状态。

概要

```
int GetBasis()
```

返回值

Constraint 的基状态。

Constraint.GetIdx()

获取 Constraint 的索引值。

概要

```
int GetIdx()
```

返回值

Constraint 的索引值。

Constraint.GetLowerIIS()

获取约束下界的 IIS 状态。

概要

```
int GetLowerIIS()
```

返回值

约束下界的 IIS 状态。

Constraint.GetName()

获取 Constraint 的名称。

概要

```
string GetName()
```

返回值

Constraint 的名称。

Constraint.GetUpperIIS()

获取约束上界的 IIS 状态。

概要

```
int GetUpperIIS()
```

返回值

约束上界的 IIS 状态。

Constraint.Remove()

从模型中删除当前的 Constraint。

概要

```
void Remove()
```

Constraint.Set()

设置 Constraint 的属性值。支持"LB", "UB" 属性。

概要

```
void Set(string attr, double val)
```

参量

`attr`: 所需要设置的属性名称。

`val`: 所需要设置的属性值。

Constraint.SetName()

设置 Constraint 的名称。

概要

```
void SetName(string name)
```

参量

`name`: Constraint 的名称。

25.2.8 ConstrArray 类

为方便用户对一组 C# *Constraint 类* 对象进行操作，杉数求解器的 C# 接口设计了 ConstrArray 类，提供了以下成员方法：

ConstrArray.ConstrArray()

ConstrArray 的构造函数。

概要

```
ConstrArray()
```

ConstrArray.GetConstr()

获取 ConstrArray 中的指定索引值的 Constraint。

概要

```
Constraint GetConstr(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 Constraint。

ConstrArray.PushBack()

向 ConstrArray 中添加一个 Constraint。

概要

```
void PushBack(Constraint constr)
```

参量

`constr`: 待添加的 Constraint。

ConstrArray.Size()

获取 ConstrArray 中的元素个数。

概要

```
int Size()
```

返回值

ConstrArray 中的元素个数。

25.2.9 ConstrBuilder 类

ConstrBuilder 类是杉数优化求解器中构建线性约束时的构建器的封装，提供了以下成员方法：

ConstrBuilder.ConstrBuilder()

ConstrBuilder 的构造函数。

概要

```
ConstrBuilder()
```

ConstrBuilder.GetExpr()

获取线性约束生成器对象的表达式。

概要

```
Expr GetExpr()
```

返回值

Expression 对象。

ConstrBuilder.GetRange()

获取线性约束生成器对象的约束范围的长度（从下界到上界的长度，必须大于 0）。

概要

```
double GetRange()
```

返回值

约束范围的长度（从下界到上界的长度）。

ConstrBuilder.GetSense()

获取线性约束生成器对象的约束类型。

概要

```
char GetSense()
```

返回值

约束类型。

ConstrBuilder.Set()

设置一个约束构造类的内容。

概要

```
void Set(
    Expr expr,
    char sense,
    double rhs)
```

参量

`expr`: 约束一侧的表达式。

`sense`: 除了 COPT_RANGE 外的约束类型。

`rhs`: 约束另一侧的常数项

ConstrBuilder.SetRange()

设置一个范围约束（带有上下界）。

概要

```
void SetRange(Expr expr, double range)
```

参量

`expr`: 约束表达式。其表达式的常数项的负数其实是这个约束的上界。

`range`: 约束范围的长度（从下界到上界的长度，必须大于 0）。

25.2.10 ConstrBuilderArray 类

为方便用户对一组 C# *ConstrBuilder 类* 对象进行操作，杉数优化求解器的 C# 接口设计了 ConstrBuilderArray 类，提供了以下成员方法：

ConstrBuilderArray.ConstrBuilderArray()

ConstrBuilderArray 的构造函数。

概要

```
ConstrBuilderArray()
```

ConstrBuilderArray.GetBuilder()

获取 ConstrBuilderArray 中的指定索引值的 Constraint。

概要

```
ConstrBuilder GetBuilder(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 ConstrBuilder。

ConstrBuilderArray.PushBack()

向 ConstrBuilderArray 中添加一个 ConstrBuilder。

概要

```
void PushBack(ConstrBuilder builder)
```

参量

`builder`: 待添加的 ConstrBuilder。

ConstrBuilderArray.Size()

获取 ConstrBuilderArray 中的元素个数。

概要

```
int Size()
```

返回值

ConstrBuilderArray 中的元素个数。

25.2.11 Column 类

为了方便用户采用按列建模的方式，杉数优化求解器的 C# 接口设计了 Column 类，提供了以下成员方法：

Column.Column()

Column 的构造函数。

概要

```
Column()
```

Column.AddColumn()

添加一个列的项，并乘以倍数。

概要

```
void AddColumn(Column col, double mult)
```

参量

`col`: 需要添加的列对象。

`mult`: 系数倍数。

Column.AddTerm()

添加一个新的项。

概要

```
void AddTerm(Constraint constr, double coeff)
```

参量

`constr`: 待添加项的线性约束。

`coeff`: 待添加项的系数。

Column.AddTerms()

添加一个或多个新的项。

概要

```
void AddTerms(Constraint[] constrs, double coeff)
```

参量

`constrs`: 待添加项的线性约束数组。

`coeff`: 待添加项的系数。

Column.AddTerms()

添加一个或多个新的项。

概要

```
void AddTerms(Constraint[] constrs, double[] coeffs)
```

参量

`constrs`: 待添加项的线性约束数组。

`coeffs`: 待添加项的系数数组。

Column.AddTerms()

添加一个或多个新的项。

概要

```
void AddTerms(ConstrArray constrs, double coeff)
```

参量

`constrs`: 待添加项的线性约束构成的 ConstrArray 类。

`coeff`: 待添加项的系数。

Column.AddTerms()

添加一个或多个新的项。

概要

```
void AddTerms(ConstrArray constrs, double[] coeffs)
```

参量

`constrs`: 待添加项的线性约束构成的 ConstrArray 类。

`coeffs`: 待添加项的系数数组。

Column.Clear()

清空 Column 的内容。

概要

```
void Clear()
```

Column.Clone()

创建 Column 的深度拷贝。

概要

```
Column Clone()
```

返回值

Column 的深度拷贝对象。

Column.GetCoeff()

获得 Column 中第 i 项的线性约束。

概要

```
double GetCoeff(int i)
```

参量

`i`: 第 i 项的索引值。

返回值

Column 中第 i 项的线性约束。

Column.GetConstr()

获得 Column 中第 i 项的系数。

概要

```
Constraint GetConstr(int i)
```

参量

`i`: 第 i 项的索引值。

返回值

Column 中第 i 项的系数。

Column.Remove()

从 Column 中移除指定的项。

概要

```
void Remove(int idx)
```

参量

`idx`: 待移除项的索引值。

Column.Remove()

从 Column 中移除指定线性约束所在的项。

概要

```
void Remove(Constraint constr)
```

参量

`constr`: 指定线性约束。

Column.Size()

获取 Column 中元素的个数。

概要

```
int Size()
```

返回值

Column 中元素的个数。

25.2.12 ColumnArray 类

为方便用户对一组 C# *Column 类* 对象进行操作，杉数优化求解器的 C# 接口设计了 ColumnArray 类，提供了以下成员方法：

ColumnArray.ColumnArray()

ColumnArray 的构造函数。

概要

```
ColumnArray()
```

ColumnArray.Clear()

清空所有的 Column。

概要

```
void Clear()
```

ColumnArray.GetColumn()

获取 ColumnArray 中的指定索引值的 Column。

概要

```
Column GetColumn(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 Column。

ColumnArray.PushBack()

向 ColumnArray 中添加一个 Column。

概要

```
void PushBack(Column col)
```

参量

`col`: 待添加的 Column。

ColumnArray.Size()

获取 ColumnArray 中的元素个数。

概要

```
int Size()
```

返回值

ColumnArray 中的元素个数。

25.2.13 Sos 类

SOS 类是杉数求解器的 SOS 约束的相关操作的封装，特殊约束：*SOS* 约束章节 目前提供了以下成员方法：

关于 SOS 约束的介绍请参考特殊约束：*SOS* 约束章节 。

Sos.GetIdx()

获取 SOS 约束的索引值。

概要

```
int GetIdx()
```

返回值

SOS 约束的索引值。

Sos.GetIIS()

获取 SOS 约束的 IIS 状态。

概要

```
int GetIIS()
```

返回值

IIS 状态。

Sos.Remove()

从模型中删除 SOS 约束。

概要

```
void Remove()
```

25.2.14 SosArray 类

为方便用户对一组 C# *Sos 类* 对象进行操作，杉数求解器的 C# 接口设计了 SosArray 类，提供了以下成员方法：

SosArray.SosArray()

SosArray 的构造函数。

概要

```
SosArray()
```

SosArray.GetSos()

获取 SosArray 中的指定索引值的 Sos 约束。

概要

```
Sos GetSos(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 Sos 约束。

SosArray.PushBack()

向 SosArray 里添加 SOS 约束。

概要

```
void PushBack(Sos sos)
```

参量

`sos`: SOS 约束。

SosArray.Size()

获取 SosArray 里 SOS 约束个数。

概要

```
int Size()
```

返回值

SOS 约束个数。

25.2.15 SosBuilder 类

SOSBuilder 类是杉数优化求解器中构建 SOS 约束的构建器的封装，提供了以下成员方法：

关于 SOS 约束的介绍请参考特殊约束：*SOS* 约束章节 。

SosBuilder.SosBuilder()

SosBuilder 的构造函数。

概要

```
SosBuilder()
```

SosBuilder.GetSize()

获取 SOS 约束中元素个数。

概要

```
int GetSize()
```

返回值

元素个数。

SosBuilder.GetType()

获取 SOS 约束类型。

概要

```
int GetType()
```

返回值

SOS 约束类型。

SosBuilder.GetVar()

从 SOS 约束中指定索引的元素中获取变量。

概要

```
Var GetVar(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定索引元素对应的变量。

SosBuilder.GetVars()

获取 SOS 约束中的全部变量。

概要

```
VarArray GetVars()
```

返回值

一组变量。

SosBuilder.GetWeight()

从 SOS 约束中指定索引的元素中获取权重。

概要

```
double GetWeight(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定索引元素中对应的权重。

SosBuilder.GetWeights()

获取 SOS 约束中所有元素对应的权重。

概要

```
double[] GetWeights()
```

返回值

权重数组。

SosBuilder.Set()

设置 SOS 约束的变量和权重。

概要

```
void Set(
    VarArray vars,
    double[] weights,
    int type)
```

参量

`vars`: 变量构成的 VarArray 类。

`weights`: 权重数组。

`type`: SOS 约束的类型。

25.2.16 SosBuilderArray 类

为方便用户对一组 C# *SosBuilder 类* 对象进行操作，杉数优化求解器的 C# 接口设计了 SosBuilderArray 类，提供了以下成员方法：

SosBuilderArray.SosBuilderArray()

SosBuilderArray 的构造函数。

概要

```
SosBuilderArray()
```

SosBuilderArray.GetBuilder()

获取指定索引值的 SOS 约束生成器（SosBuilder）。

概要

```
SosBuilder GetBuilder(int idx)
```

参量

`idx`: 指定索引值。

返回值

指定索引值的 SOS 约束生成器（SosBuilder）。

SosBuilderArray.PushBack()

向 SosBuilderArray 类中添加 SOS 约束生成器（SosBuilder）。

概要

```
void PushBack(SosBuilder builder)
```

参量

`builder`: SOS 约束生成器（SosBuilder）。

SosBuilderArray.Size()

获取 SosBuilderArray 类中元素个数。

概要

```
int Size()
```

返回值

SosBuilderArray 类中元素个数。

25.2.17 GenConstr 类

GenConstr 类是杉数优化求解器的 Indicator 约束的相关操作的封装，提供了以下成员方法：

关于 Indicator 约束的介绍请参考特殊约束：*Indicator 约束章节* 。

GenConstr.GetIdx()

获取 GenConstr 的索引值。

概要

```
int GetIdx()
```

返回值

GenConstr 的索引值。

GenConstr.GetIIS()

获取一般约束的 IIS 状态。

概要

```
int GetIIS()
```

返回值

IIS 状态。

GenConstr.Remove()

从模型中删除 GenConstr。

概要

```
void Remove()
```

25.2.18 GenConstrArray 类

为方便用户对一组 C# *GenConstr 类* 对象进行操作，杉数优化求解器的 C# 接口设计了 GenConstrArray 类，提供了以下成员方法：

GenConstrArray.GenConstrArray()

GenConstrArray 的构造函数。

概要

```
GenConstrArray()
```

GenConstrArray.GetGenConstr()

获取 GenConstrArray 中的指定索引值的 GenConstr。

概要

```
GenConstr GetGenConstr(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 GenConstr。

GenConstrArray.PushBack()

向 GenConstrArray 中添加一个 GenConstr。

概要

```
void PushBack(GenConstr genconstr)
```

参量

`genconstr`: 待添加的 GenConstr。

GenConstrArray.Size()

获取 GenConstrArray 中的元素个数。

概要

```
int Size()
```

返回值

GenConstrArray 中的元素个数。

25.2.19 GenConstrBuilder 类

GenConstrBuilder 类是杉数优化求解器中构建 Indicator 约束时的构建器的封装，提供了以下成员方法：

关于 Indicator 约束的介绍请参考特殊约束：*Indicator* 约束章节 。

GenConstrBuilder.GenConstrBuilder()

GenConstrBuilder 的构造函数。

概要

```
GenConstrBuilder()
```

GenConstrBuilder.GetBinVal()

获取与 GenConstr 的相关联的二进制值。

概要

```
int GetBinVal()
```

返回值

二进制值。

GenConstrBuilder.GetBinVar()

获取与 GenConstr 的相关联的二进制变量。

概要

```
Var GetBinVar()
```

返回值

二进制变量。

GenConstrBuilder.GetExpr()

获取与 GenConstr 的相关联的表达式。

概要

```
Expr GetExpr()
```

返回值

表达式对象。

GenConstrBuilder.GetSense()

获取与 GenConstr 的相关联的约束类型。

概要

```
char GetSense()
```

返回值

约束类型。

GenConstrBuilder.Set()

设置 GenConstr 二进制变量，二进制变量取值，表达式，约束类型。

概要

```
void Set(
    Var binvar,
    int binval,
    Expr expr,
    char sense)
```

参量

`binvar`: 二进制变量。

`binval`: 二进制变量取值。

`expr`: 表达式。

`sense`: 约束类型。

25.2.20 GenConstrBuilderArray 类

为方便用户对一组 C# *GenConstrBuilder 类* 对象进行操作，杉数优化求解器的 C# 接口设计了 GenConstrBuilderArray 类，提供了以下成员方法：

GenConstrBuilderArray.GenConstrBuilderArray()

GenConstrBuilderArray 的构造函数。

概要

```
GenConstrBuilderArray()
```

GenConstrBuilderArray.GetBuilder()

获取 GenConstrBuilderArray 中的指定索引值的 GenConstrBuilder。

概要

```
GenConstrBuilder GetBuilder(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 GenConstrBuilder。

GenConstrBuilderArray.PushBack()

向 GenConstrBuilderArray 中添加一个 GenConstrBuilder。

概要

```
void PushBack(GenConstrBuilder builder)
```

参量

`builder`: 待添加的 GenConstrBuilder。

GenConstrBuilderArray.Size()

获取 GenConstrBuilderArray 中的元素个数。

概要

```
int Size()
```

返回值

GenConstrBuilderArray 中的元素个数。

25.2.21 Cone 类

Cone 类是杉数求解器的二阶锥约束的相关操作的封装，目前提供了以下成员方法：

Cone.GetIdx()

获取锥约束的索引值。

概要

```
int GetIdx()
```

返回值

锥约束的索引值。

Cone.Remove()

从模型中删除锥约束。

概要

```
void Remove()
```

25.2.22 ConeArray 类

为方便用户对一组 C# *Cone 类* 对象进行操作，杉数求解器的 C# 接口设计了 ConeArray 类，提供了以下成员方法：

ConeArray.ConeArray()

ConeArray 的构造函数。

概要

```
ConeArray()
```

ConeArray.GetCone()

获取 ConeArray 中的指定索引值的 Cone。

概要

```
Cone GetCone(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 Cone。

ConeArray.PushBack()

向 ConeArray 里添加锥约束。

概要

```
void PushBack(Cone cone)
```

参量

`cone`: 锥约束。

ConeArray.Size()

获取 ConeArray 里锥约束个数。

概要

```
int Size()
```

返回值

锥约束个数。

25.2.23 ConeBuilder 类

ConeBuilder 类是杉数优化求解器中构建二阶锥约束的构建器的封装，提供了以下成员方法：

ConeBuilder.ConeBuilder()

ConeBuilder 的构造函数。

概要

```
ConeBuilder()
```

ConeBuilder.GetSize()

获取锥约束中变量个数。

概要

```
int GetSize()
```

返回值

变量个数。

ConeBuilder.GetType()

获取锥约束类型。

概要

```
int GetType()
```

返回值

锥约束类型。

ConeBuilder.GetVar()

从锥约束中获取指定下标的变量。

概要

```
Var GetVar(int idx)
```

参量

`idx`: 指定的下标值。

返回值

指定下标对应的变量。

ConeBuilder.GetVars()

获取锥约束中的全部变量。

概要

```
VarArray GetVars()
```

返回值

一组变量。

ConeBuilder.Set()

设置锥约束的变量和类型。

概要

```
void Set(VarArray vars, int type)
```

参量

`vars`: 变量构成的 VarArray 类。

`type`: 锥约束的类型。

25.2.24 ConeBuilderArray 类

为方便用户对一组 C# *ConeBuilder 类* 对象进行操作，杉数优化求解器的 C# 接口设计了 ConeBuilderArray 类，提供了以下成员方法：

ConeBuilderArray.ConeBuilderArray()

ConeBuilderArray 的构造函数。

概要

```
ConeBuilderArray()
```

ConeBuilderArray.GetBuilder()

获取指定索引值的锥约束生成器（ConeBuilder）。

概要

```
ConeBuilder GetBuilder(int idx)
```

参量

`idx`: 指定索引值。

返回值

指定索引值的锥约束生成器（ConeBuilder）。

ConeBuilderArray.PushBack()

向 ConeBuilderArray 类中添加锥约束生成器（ConeBuilder）。

概要

```
void PushBack(ConeBuilder builder)
```

参量

`builder`: 锥约束生成器（ConeBuilder）。

ConeBuilderArray.Size()

获取 ConeBuilderArray 类中元素个数。

概要

```
int Size()
```

返回值

ConeBuilderArray 类中元素个数。

25.2.25 QuadExpr 类

COPT 二次表达式包括一个线性表达式，一些二次项相关的变量和对应系数。QuadExpr 类是杉数求解器中用于构建二次表达式时对变量的相关组合操作，提供了以下成员方法：

QuadExpr.QuadExpr()

二次表达式的构造函数。

概要

```
QuadExpr(double constant)
```

参量

`constant`: 二次表达式对象中的常量。

QuadExpr.QuadExpr()

包含一线性项的二次表达式的构造函数。

概要

```
QuadExpr(Var var, double coeff)
```

参量

`var`: 添加的一线性项对应的变量。

`coeff`: 添加的一线性项对应的参数，默认值为 1.0。

QuadExpr.QuadExpr()

二次表达式的构造函数，初始值是给定的线性表达式。

概要

```
QuadExpr(Expr expr)
```

参量

`expr`: 二次表达式中的线性部分。

QuadExpr.QuadExpr()

使用两个线性表达式构造的二次表达式。

概要

```
QuadExpr(Expr expr, Var var)
```

参量

`expr`: 一个初始的线性表达式。

`var`: 另一个初始的变量。

QuadExpr.QuadExpr()

使用两个线性表达式构造的二次表达式。

概要

```
QuadExpr(Expr left, Expr right)
```

参量

`left`: 一个初始的线性表达式。

`right`: 另一个初始的线性表达式。

QuadExpr.AddConstant()

增加二次表达式中的常数。

概要

```
void AddConstant(double constant)
```

参量

`constant`: 二次表达式中的常数改变量。

QuadExpr.AddLinExpr()

在二次表达式中添加一个线性表达式。

概要

```
void AddLinExpr(Expr expr)
```

参量

`expr`: 需要添加的线性表达式。

QuadExpr.AddLinExpr()

添加一个线性表达式的项，并乘以倍数。

概要

```
void AddLinExpr(Expr expr, double mult)
```

参量

`expr`: 需要添加的线性表达式

`mult`: 倍数参数。

QuadExpr.AddQuadExpr()

在二次表达式中添加一个二次表达式。

概要

```
void AddQuadExpr(QuadExpr expr)
```

参量

expr: 需要添加的二次表达式。

QuadExpr.AddQuadExpr()

添加一个二次表达式的项，并乘以倍数。

概要

```
void AddQuadExpr(QuadExpr expr, double mult)
```

参量

expr: 需要添加的二次表达式

mult: 倍数参数。

QuadExpr.AddTerm()

向二次表达式中添加一线性项。

概要

```
void AddTerm(Var var, double coeff)
```

参量

var: 新线性项中的变量。

coeff: 新线性项中的系数。

QuadExpr.AddTerm()

向二次表达式中添加一个二次项。

概要

```
void AddTerm(
    Var var1,
    Var var2,
    double coeff)
```

参量

var1: 新二次项中的变量 1。

var2: 新二次项中的变量 2。

coeff: 新二次项中的系数。

QuadExpr.AddTerms()

向二次表达式中添加一些线性项。

概要

```
void AddTerms(Var[] vars, double coeff)
```

参量

vars: 新线性项中的变量数组。

coeff: 新线性项中的公共系数。

QuadExpr.AddTerms()

向二次表达式中添加一些线性项。

概要

```
void AddTerms(Var[] vars, double[] coeffs)
```

参量

vars: 新线性项中的变量数组。

coeffs: 新线性项中的系数数组。

QuadExpr.AddTerms()

向二次表达式中添加线性项。

概要

```
void AddTerms(VarArray vars, double coeff)
```

参量

vars: 新线性项中的变量构成的 VarArray 类。

coeff: 新线性项中的公共系数。

QuadExpr.AddTerms()

向二次表达式中添加线性项。

概要

```
void AddTerms(VarArray vars, double[] coeffs)
```

参量

vars: 新线性项中的变量构成的 VarArray 类。

coeffs: 新线性项中的系数数组。

QuadExpr.AddTerms()

向二次表达式中添加一些二次项。

概要

```
void AddTerms(
    VarArray vars1,
    VarArray vars2,
    double[] coeffs)
```

参量

vars1: 新二次项中的变量数组 1。

vars2: 新二次项中的变量数组 2。

coeffs: 新二次项中的系数数组。

QuadExpr.AddTerms()

向二次表达式中添加一些二次项。

概要

```
void AddTerms(
    Var[] vars1,
    Var[] vars2,
    double[] coeffs)
```

参量

vars1: 新二次项中的变量数组 1。

vars2: 新二次项中的变量数组 2。

coeffs: 新二次项中的系数数组。

QuadExpr.Clone()

深度拷贝二次表达式对象。

概要

```
QuadExpr Clone()
```

返回值

复制的二次表达式对象。

QuadExpr.Evaluate()

求解后对二次表达式估值。

概要

```
double Evaluate()
```

返回值

表达式估值。

QuadExpr.GetCoeff()

获取二次表达式指定索引值对应项数中的系数。

概要

```
double GetCoeff(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值对应项数中的系数。

QuadExpr.GetConstant()

获取二次表达式中的常数。

概要

```
double GetConstant()
```

返回值

二次表达式中的常数。

QuadExpr.GetLinExpr()

获取二次表达式中的线性表达式。

概要

```
Expr GetLinExpr()
```

返回值

线性表达式对象。

QuadExpr.GetVar1()

获取二次表达式指定索引值对应项数中的第一个变量。

概要

```
Var GetVar1(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值对应项数中的第一个变量。

QuadExpr.GetVar2()

获取表达式指定索引值对应项数中的第二个变量。

概要

```
Var GetVar2(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值对应项数中的第二个变量。

QuadExpr.Remove()

删除二次表达式中指定索引值的项。

概要

```
void Remove(int idx)
```

参量

`idx`: 指定索引值。

QuadExpr.Remove()

删除二次表达式中与指定变量相关的项。

概要

```
void Remove(Var var)
```

参量

`var`: 指定删除的变量。

QuadExpr.SetCoeff()

设置二次表达式指定索引值项数中的系数。

概要

```
void SetCoeff(int i, double val)
```

参量

`i`: 指定索引值。

`val`: 指定索引值项数中的参数。

QuadExpr.SetConstant()

设置二次表达式中的常数。

概要

```
void SetConstant(double constant)
```

参量

`constant`: 二次表达式中的常数。

QuadExpr.Size()

获取二次表达式中的项数。

概要

```
long Size()
```

返回值

二次表达式中的二次项数。

25.2.26 QConstraint 类

QConstraint 类是杉数求解器对二次约束的相关操作的封装，提供了以下成员方法：

QConstraint.Get()

获得二次约束的属性值。

概要

```
double Get(string attr)
```

参量

`attr`: 所需要获得的属性名称。

返回值

属性值。

QConstraint.GetIdx()

获取二次约束的索引值。

概要

```
int GetIdx()
```

返回值

二次约束的索引值。

QConstraint.GetName()

获取二次约束的名称。

概要

```
string GetName()
```

返回值

二次约束的名称。

QConstraint.GetRhs()

获得二次约束的右端值。

概要

```
double GetRhs()
```

返回值

二次约束的右端值。

QConstraint.GetSense()

获得二次约束的右端值。

概要

```
char GetSense()
```

返回值

二次约束的右端值。

QConstraint.Remove()

从模型中删除当前的 Constraint。

概要

```
void Remove()
```

QConstraint.Set()

设置 Constraint 的属性值。

概要

```
void Set(string attr, double val)
```

参量

`attr`: 所需要设置的属性名称。

`val`: 所需要设置的属性值。

QConstraint.SetName()

设置二次约束的名称。

概要

```
void SetName(string name)
```

参量

`name`: 二次约束的名称。

QConstraint.SetRhs()

设置二次约束的右端值。

概要

```
void SetRhs(double rhs)
```

参量

`rhs`: 二次约束的右端值。

QConstraint.SetSense()

设置二次约束的类型。

概要

```
void SetSense(char sense)
```

参量

`sense`: 二次约束的类型。

25.2.27 QConstrArray 类

为方便用户对一组 C# *QConstraint 类* 对象进行操作，杉数求解器的 C# 接口设计了 QConstrArray 类，提供了以下成员方法：

QConstrArray.QConstrArray()

QConstrArray 的构造函数。

概要

```
QConstrArray()
```

QConstrArray.GetQConstr()

获取 QConstrArray 中的指定索引值的二次约束。

概要

```
QConstraint GetQConstr(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的二次约束。

QConstrArray.PushBack()

向 QConstrArray 中添加一个二次约束。

概要

```
void PushBack(QConstraint constr)
```

参量

`constr`: 待添加的二次约束。

QConstrArray.Size()

获取 QConstrArray 中的元素个数。

概要

```
int Size()
```

返回值

QConstrArray 中的元素个数。

25.2.28 QConstrBuilder 类

QConstrBuilder 类是杉数优化求解器中对构建二次约束的构建器的封装，提供了以下成员方法：

QConstrBuilder.QConstrBuilder()

QConstrBuilder 的构造函数。

概要

```
QConstrBuilder()
```

QConstrBuilder.GetQuadExpr()

获取二次约束相关的表达式。

概要

```
QuadExpr GetQuadExpr()
```

返回值

二次表达式对象。

QConstrBuilder.GetSense()

获取二次约束相关的约束类型。

概要

```
char GetSense()
```

返回值

约束类型。

QConstrBuilder.Set()

设置一个二次约束的表达式，类型和边界值。

概要

```
void Set(
    QuadExpr expr,
    char sense,
    double rhs)
```

参量

`expr`: 约束一侧的二次表达式。

`sense`: 二次约束类型。

`rhs`: 二次约束另一侧的常数项。

25.2.29 QConstrBuilderArray 类

为方便用户对一组 C# *QConstrBuilder 类* 对象进行操作，杉数优化求解器的 C# 接口设计了 QConstrBuilderArray 类，提供了以下成员方法：

QConstrBuilderArray.QConstrBuilderArray()

QConstrBuilderArray 的构造函数。

概要

```
QConstrBuilderArray()
```

QConstrBuilderArray.GetBuilder()

获取 QConstrBuilderArray 中的指定索引值的 QConstraintBuilder 对象。

概要

```
QConstrBuilder GetBuilder(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 QConstrBuilder 对象。

QConstrBuilderArray.PushBack()

向 QConstrBuilderArray 中添加一个 QConstrBuilder。

概要

```
void PushBack(QConstrBuilder builder)
```

参量

`builder`: 待添加的 QConstrBuilder。

QConstrBuilderArray.Size()

获取 QConstrBuilderArray 中的元素个数。

概要

```
int Size()
```

返回值

QConstrBuilderArray 中的元素个数。

25.2.30 PsdVar 类

PsdVar 类是杉数优化求解器对半定变量的相关操作的封装，提供了以下成员方法：

PsdVar.Get()

获取半定变量的属性值。

概要

```
double[] Get(string attr)
```

参量

`attr`: 属性的名称。

返回值

输出双精度型数组，保存了属性值。

PsdVar.GetDim()

获取半定变量的维度。

概要

```
int GetDim()
```

返回值

半定变量的维度。

PsdVar.GetIdx()

获取半定变量展开后的索引。

概要

```
int GetIdx()
```

返回值

半定变量的索引。

PsdVar.GetLen()

获取半定变量展开后的长度。

概要

```
int GetLen()
```

返回值

半定变量展开后的长度。

PsdVar.GetName()

获取半定变量的名称。

概要

```
string GetName()
```

返回值

半定变量名称。

PsdVar.Remove()

从模型中删除半定变量。

概要

```
void Remove()
```

25.2.31 PsdVarArray 类

为方便用户对一组*PsdVar 类* 对象进行操作，杉数优化求解器的 C# 接口设计了 PsdVarArray 类，提供了以下成员方法：

PsdVarArray.PsdVarArray()

PsdVarArray 的构造函数。

概要

```
PsdVarArray()
```

PsdVarArray.GetPsdVar()

获取半定变量数组里指定索引的半定变量。

概要

```
PsdVar GetPsdVar(int idx)
```

参量

`idx`: 半定变量的索引。

返回值

指定索引的半定变量。

PsdVarArray.PushBack()

向半定变量数组里添加半定变量。

概要

```
void PushBack(PsdVar var)
```

参量

`var`: 半定变量。

PsdVarArray.Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

n: 容纳 n 项的空间。

PsdVarArray.Size()

获取半定变量数组里半定变量个数。

概要

```
int Size()
```

返回值

半定变量个数。

25.2.32 PsdExpr 类

COPT 半定表达式包括一个线性表达式，一些半定变量和对应的系数矩阵。PsdExpr 类是杉数求解器中用于构建半定表达式时对半定变量的相关组合操作，提供了以下成员方法：

PsdExpr.PsdExpr()

半定表达式的构造函数。

概要

```
PsdExpr(double constant)
```

参量

constant: 半定表达式对象中的常量。

PsdExpr.PsdExpr()

使用变量和其系数构造的半定表达式。

概要

```
PsdExpr(Var var, double coeff)
```

参量

var: 添加的这一项对应的变量。

coeff: 添加的这一项对应的参数，默认值为 1.0。

PsdExpr.PsdExpr()

使用线性表达式构造的半定表达式。

概要

```
PsdExpr(Expr expr)
```

参量

expr: 初始的线性表达式。

PsdExpr.PsdExpr()

使用半定变量和其系数矩阵构造的半定表达式。

概要

```
PsdExpr(PsdVar var, SymMatrix mat)
```

参量

var: 添加的这一项对应的半定变量。

mat: 添加的这一项对应的系数矩阵。

PsdExpr.PsdExpr()

使用半定变量和其系数矩阵构造的半定表达式。

概要

```
PsdExpr(PsdVar var, SymMatExpr expr)
```

参量

var: 添加的这一项对应的半定变量。

expr: 新半定项中对称矩阵的表达式。

PsdExpr.AddConstant()

增加半定表达式中的常数。

概要

```
void AddConstant(double constant)
```

参量

constant: 半定表达式中的常数改变量。

PsdExpr.AddLinExpr()

在半定表达式中添加一个线性表达式。

概要

```
void AddLinExpr(Expr expr)
```

参量

expr: 需要添加的线性表达式。

PsdExpr.AddLinExpr()

在半定表达式中添加一个线性表达式的项，并乘以倍数。

概要

```
void AddLinExpr(Expr expr, double mult)
```

参量

expr: 需要添加的线性表达式

mult: 倍数参数。

PsdExpr.AddPsdExpr()

添加一个半定表达式的项，并乘以倍数。

概要

```
void AddPsdExpr(PsdExpr expr)
```

参量

expr: 需要添加的半定表达式。

PsdExpr.AddPsdExpr()

添加一个半定表达式的项，并乘以倍数。

概要

```
void AddPsdExpr(PsdExpr expr, double mult)
```

参量

expr: 需要添加的半定表达式。

mult: 倍数参数。

PsdExpr.AddTerm()

向半定表达式中添加一线性项。

概要

```
void AddTerm(Var var, double coeff)
```

参量

var: 新线性项中的变量。

coeff: 新线性项中的系数，默认值为 1.0。

PsdExpr.AddTerm()

向半定表达式中添加一个半定项。

概要

```
void AddTerm(PsdVar var, SymMatrix mat)
```

参量

var: 新半定项中的半定变量。

mat: 新半定项中的系数矩阵。

PsdExpr.AddTerm()

向半定表达式中添加一个半定项。

概要

```
void AddTerm(PsdVar var, SymMatExpr expr)
```

参量

var: 新半定项中的半定变量。

expr: 新半定项中对称矩阵的表达式。

PsdExpr.AddTerms()

向半定表达式中添加一些线性项。

概要

```
void AddTerms(Var[] vars, double coeff)
```

参量

vars: 新线性项中的变量数组。

coeff: 新线性项中的公共系数，默认值为 1.0。

PsdExpr.AddTerms()

向表达式中添加一些线性项。

概要

```
void AddTerms(Var[] vars, double[] coeffs)
```

参量

`vars`: 新线性项中的变量数组。

`coeffs`: 新线性项中的系数数组。

PsdExpr.AddTerms()

向半定表达式中添加线性项。

概要

```
void AddTerms(VarArray vars, double coeff)
```

参量

`vars`: 新线性项中的变量构成的 VarArray 类。

`coeff`: 新线性项中的公共系数，默认值为 1.0。

PsdExpr.AddTerms()

向半定表达式中添加线性项。

概要

```
void AddTerms(VarArray vars, double[] coeffs)
```

参量

`vars`: 新线性项中的变量构成的 VarArray 类。

`coeffs`: 新线性项中的系数数组。

PsdExpr.AddTerms()

向表达式中添加一些半定项。

概要

```
void AddTerms(PsdVarArray vars, SymMatrixArray mats)
```

参量

`vars`: 新半定项中的半定变量数组。

`mats`: 新半定项中的系数矩阵数组。

PsdExpr.AddTerms()

向表达式中添加一些半定项。

概要

```
void AddTerms(PsdVar[] vars, SymMatrix[] mats)
```

参量

vars: 新半定项中的半定变量数组。

mats: 新半定项中的系数矩阵数组。

PsdExpr.Clone()

深度拷贝半定表达式对象。

概要

```
PsdExpr Clone()
```

返回值

复制的半定表达式对象。

PsdExpr.Evaluate()

求解后对半定表达式估值。

概要

```
double Evaluate()
```

返回值

表达式估值。

PsdExpr.GetCoeff()

获取半定表达式中指定索引值对应项的系数。

概要

```
SymMatExpr GetCoeff(int i)
```

参量

i: 指定索引值。

返回值

指定索引值对应的半定表达式项的系数矩阵表达式。

PsdExpr.GetConstant()

获取半定表达式中的常数项。

概要

```
double GetConstant()
```

返回值

半定表达式中的常数项。

PsdExpr.GetLinExpr()

获取半定表达式中的线性表达式。

概要

```
Expr GetLinExpr()
```

返回值

线性表达式对象。

PsdExpr.GetPsdVar()

获取半定表达式指定索引值对应项中的半定变量。

概要

```
PsdVar GetPsdVar(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值对应项的半定变量对象。

PsdExpr.Multiply()

对半定表达式乘以常数。

概要

```
void Multiply(double c)
```

参量

`c`: 常数操作数。

PsdExpr.Remove()

删除半定表达式中指定索引值的项。

概要

```
void Remove(int idx)
```

参量

idx: 指定索引值。

PsdExpr.Remove()

删除半定表达式中与指定变量相关的项。

概要

```
void Remove(Var var)
```

参量

var: 指定变量。

PsdExpr.Remove()

删除半定表达式中与指定半定变量相关的项。

概要

```
void Remove(PsdVar var)
```

参量

var: 指定半定变量。

PsdExpr.SetCoeff()

设置半定表达式指定索引值对应项的系数矩阵。

概要

```
void SetCoeff(int i, SymMatrix mat)
```

参量

i: 指定索引值。

mat: 指定索引值对应项的系数矩阵。

PsdExpr.SetConstant()

设置半定表达式中的常数。

概要

```
void SetConstant(double constant)
```

参量

`constant`: 半定表达式中的常数。

PsdExpr.Size()

获取半定表达式中的半定项数。

概要

```
long Size()
```

返回值

半定表达式中的半定项数。

25.2.33 PsdConstraint 类

PsdConstraint 类是杉数求解器对半定约束的相关操作的封装，提供了以下成员方法：

PsdConstraint.Get()

获得半定约束的属性值。支持半定相关的属性。

概要

```
double Get(string attr)
```

参量

`attr`: 所需要获得的属性名称。

返回值

双精度属性值。

PsdConstraint.GetIdx()

获取半定约束的索引值。

概要

```
int GetIdx()
```

返回值

半定约束的索引值。

PsdConstraint.GetName()

获取半定约束的名称。

概要

```
string GetName()
```

返回值

半定约束的名称。

PsdConstraint.Remove()

从模型中删除当前的半定约束。

概要

```
void Remove()
```

PsdConstraint.Set()

设置半定约束的属性值。支持半定相关的属性。

概要

```
void Set(string attr, double value)
```

参量

`attr`: 所需要设置的属性名称。

`value`: 所需要设置的属性值。

PsdConstraint.SetName()

设置半定约束的名称。

概要

```
void SetName(string name)
```

参量

`name`: 半定约束的名称。

25.2.34 PsdConstrArray 类

为方便用户对一组 C# *PsdConstraint 类* 对象进行操作，杉数求解器的 C# 接口设计了 PsdConstrArray 类，提供了以下成员方法：

PsdConstrArray.PsdConstrArray()

PsdConstrArray 的构造函数。

概要

```
PsdConstrArray()
```

PsdConstrArray.GetPsdConstr()

获取半定约束数组中的指定索引值的半定约束。

概要

```
PsdConstraint GetPsdConstr(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的半定约束。

PsdConstrArray.PushBack()

向半定约束数组中添加一个半定约束。

概要

```
void PushBack(PsdConstraint constr)
```

参量

`constr`: 待添加的半定约束。

PsdConstrArray.Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

`n`: 容纳 n 项的空间。

PsdConstrArray.Size()

获取半定约束数组中的元素个数。

概要

```
int Size()
```

返回值

半定约束数组中的元素个数。

25.2.35 PsdConstrBuilder 类

PsdConstrBuilder 类是杉数优化求解器中对构建半定约束的构建器的封装，提供了以下成员方法：

PsdConstrBuilder.PsdConstrBuilder()

PsdConstrBuilder 的构造函数。

概要

```
PsdConstrBuilder()
```

PsdConstrBuilder.GetPsdExpr()

获取半定约束相关的表达式。

概要

```
PsdExpr GetPsdExpr()
```

返回值

半定表达式对象。

PsdConstrBuilder.GetRange()

获取半定约束生成器对象的约束范围的长度（从下界到上界的长度，必须大于 0）。

概要

```
double GetRange()
```

返回值

半定约束范围的长度（从下界到上界的长度）。

PsdConstrBuilder.GetSense()

获取半定约束相关约束类型。

概要

```
char GetSense()
```

返回值

半定约束类型。

PsdConstrBuilder.Set()

设置一个半定约束的表达式，类型和边界值。

概要

```
void Set(
    PsdExpr expr,
    char sense,
    double rhs)
```

参量

`expr`: 约束一侧的半定表达式。

`sense`: 除了 COPT_RANGE 外的半定约束类型。

`rhs`: 约束另一侧的常数项。

PsdConstrBuilder.SetRange()

设置一个范围约束（带有上下界）。

概要

```
void SetRange(PsdExpr expr, double range)
```

参量

`expr`: 半定表达式。其表达式的常数项的负数其实是这个约束的上界。

`range`: 约束范围的长度（从下界到上界的长度，必须大于 0）。

25.2.36 PsdConstrBuilderArray 类

为方便用户对一组 C# *PsdConstrBuilder 类* 对象进行操作，杉数优化求解器的 C# 接口设计了 PsdConstrBuilderArray 类，提供了以下成员方法：

PsdConstrBuilderArray.PsdConstrBuilderArray()

PsdConstrBuilderArray 的构造函数。

概要

```
PsdConstrBuilderArray()
```

PsdConstrBuilderArray.GetBuilder()

获取半定约束生成器数组中的指定索引值的半定约束。

概要

```
PsdConstrBuilder GetBuilder(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的半定约束生成器。

PsdConstrBuilderArray.PushBack()

向半定约束生成器数组中添加一个半定约束生成器。

概要

```
void PushBack(PsdConstrBuilder builder)
```

参量

`builder`: 待添加的半定约束生成器。

PsdConstrBuilderArray.Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

`n`: 容纳 n 项的空间。

PsdConstrBuilderArray.Size()

获取半定约束生成器数组中的元素个数。

概要

```
int Size()
```

返回值

半定约束生成器数组中的元素个数。

25.2.37 LmiConstraint 类

LmiConstraint 类是杉数求解器对 LMI 约束的相关操作的封装，提供了以下成员方法：

LmiConstraint.Get()

获取 LMI 约束的属性值。

概要

```
double[] Get(string attr)
```

参量

`attr`: 属性的名称。

返回值

输出双精度型数组，保存了属性值。

LmiConstraint.GetDim()

获取 LMI 约束的维度。

概要

```
int GetDim()
```

返回值

LMI 约束的维度。

LmiConstraint.GetIdx()

获取 LMI 约束的索引。

概要

```
int GetIdx()
```

返回值

LMI 约束的索引。

LmiConstraint.GetLen()

获取 LMI 约束展开后的长度。

概要

```
int GetLen()
```

返回值

LMI 约束展开后的长度。

LmiConstraint.GetName()

获取 LMI 约束的名称。

概要

```
string GetName()
```

返回值

LMI 约束名称。

LmiConstraint.Remove()

从模型中删除当前的 LMI 约束。

概要

```
void Remove()
```

LmiConstraint.SetRhs()

设置 LMI 约束的常数项。

概要

```
void SetRhs(SymMatrix mat)
```

参量

`mat`: 所需要设置的对称矩阵。

25.2.38 LmiConstrArray 类

为方便用户对一组 C# *LmiConstraint 类* 对象进行操作，杉数求解器的 C# 接口设计了 LmiConstrArray 类，提供了以下成员方法：

LmiConstrArray.LmiConstrArray()

LmiConstrArray 的构造函数。

概要

```
LmiConstrArray()
```

LmiConstrArray.GetLmiConstr()

获取 LMI 约束数组里指定索引的 LMI 约束。

概要

```
LmiConstraint GetLmiConstr(int idx)
```

参量

`idx`: 指定的 LMI 约束的索引。

返回值

指定索引的 LMI 约束。

LmiConstrArray.PushBack()

在 LMI 约束数组里添加 LMI 约束。

概要

```
void PushBack(LmiConstraint constr)
```

参量

`constr`: LMI 约束。

LmiConstrArray.Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

`n`: 预分配空间的大小。

LmiConstrArray.Size()

获取 LMI 约束数组里 LMI 约束个数。

概要

```
int Size()
```

返回值

LMI 约束个数。

25.2.39 LmiExpr 类

COPT 的 LMI 表达式包括对称矩阵和标量变量相乘。LmiExpr 类是杉数求解器中用于构建 LMI 表达式时对变量和对称矩阵相关组合操作，提供了以下成员方法：

LmiExpr.LmiExpr()

LMI 表达式的默认构造函数。

概要

```
LmiExpr()
```

LmiExpr.LmiExpr()

使用对称矩阵构造的 LMI 表达式。

概要

```
LmiExpr(SymMatrix mat)
```

参量

`mat`: 作为常量项的对称矩阵。

LmiExpr.LmiExpr()

使用矩阵表达式构造的 LMI 表达式。

概要

```
LmiExpr(SymMatExpr expr)
```

参量

`expr`: 作为常量项的矩阵表达式。

LmiExpr.LmiExpr()

使用 LMI 约束和其系数矩阵构造的 LMI 表达式。

概要

```
LmiExpr(Var var, SymMatrix mat)
```

参量

`var`: 添加的这一项对应的变量。

`mat`: 添加的这一项对应的系数矩阵。

LmiExpr.LmiExpr()

使用 LMI 约束和其系数矩阵构造的 LMI 表达式。

概要

```
LmiExpr(Var var, SymMatExpr expr)
```

参量

`var`: 添加的这一项对应的变量。

`expr`: 新 LMI 项中对应的对称矩阵表达式。

LmiExpr.AddConstant()

加到 LMI 表达式中的常数项。

概要

```
void AddConstant(SymMatExpr expr)
```

参量

`expr`: 加到常数项的矩阵表达式对象。

LmiExpr.AddLmiExpr()

添加一个 LMI 表达式的项，并乘以倍数。

概要

```
void AddLmiExpr(LmiExpr expr, double mult)
```

参量

`expr`: 需要添加的 LMI 表达式。

`mult`: 可选的系数倍数，默认值为 1.0。

LmiExpr.AddTerm()

向 LMI 表达式中添加一项。

概要

```
void AddTerm(Var var, SymMatrix mat)
```

参量

`var`: 新项中的变量。

`mat`: 新项中的系数矩阵。

LmiExpr.AddTerm()

向 LMI 表达式中添加一项。

概要

```
void AddTerm(Var var, SymMatExpr expr)
```

参量

`var`: 新项中的变量。

`expr`: 新项中作为系数的对称矩阵表达式。

LmiExpr.AddTerms()

向 LMI 表达式中添加一些项。

概要

```
void AddTerms(VarArray vars, SymMatrixArray mats)
```

参量

`vars`: 新项中的变量数组。

`mats`: 新项中的系数矩阵数组。

LmiExpr.AddTerms()

向 LMI 表达式中添加一些项。

概要

```
void AddTerms(Var[] vars, SymMatrix[] mats)
```

参量

`vars`: 新项中的变量数组。

`mats`: 新项中的系数矩阵数组。

LmiExpr.Clone()

深度复制 LMI 表达式。

概要

```
LmiExpr Clone()
```

返回值

新的 LMI 表达式对象。

LmiExpr.GetCoeff()

获取 LMI 表达式中指定索引值对应项的系数矩阵表达式。

概要

```
SymMatExpr GetCoeff(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值对应的 LMI 表达式项的系数矩阵表达式。

LmiExpr.GetConstant()

获取 LMI 表达式中的常数项。

概要

```
SymMatExpr GetConstant()
```

返回值

对称矩阵表达式对象。

LmiExpr.GetVar()

获取 LMI 表达式指定索引值对应项中的变量。

概要

```
Var GetVar(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值对应项的变量对象。

LmiExpr.Multiply()

对 LMI 表达式自乘常数。

概要

```
void Multiply(double c)
```

参量

c: 自乘的倍数。

LmiExpr.Remove()

删除 LMI 表达式中指定索引对应的项。

概要

```
void Remove(int idx)
```

参量

idx: 指定的索引。

LmiExpr.Remove()

删除 LMI 表达式中与指定变量相关的项。

概要

```
void Remove(Var var)
```

参量

var: 指定的变量。

LmiExpr.SetCoeff()

设置 LMI 表达式指定索引值对应项的系数矩阵。

概要

```
void SetCoeff(int i, SymMatrix mat)
```

参量

i: 指定的索引值。

mat: 指定索引值对应项的系数矩阵。

LmiExpr.SetConstant()

设置 LMI 表达式中的常数项。

概要

```
void SetConstant(SymMatrix mat)
```

参量

`mat`: 新的对称矩阵。

LmiExpr.Size()

获取 LMI 表达式中的项数。

概要

```
long Size()
```

返回值

LMI 表达式中的项数。

25.2.40 SymMatrix 类

对称矩阵作为半定项中的系数矩阵，常用在半定表达式，半定约束和半定目标函数中。SymMatrix 类是杉数优化求解器中对称矩阵的封装，提供了以下成员方法：

SymMatrix.GetDim()

获取对称矩阵的维度。

概要

```
int GetDim()
```

返回值

对称矩阵的维度。

SymMatrix.GetIdx()

获取对称矩阵的索引值。

概要

```
int GetIdx()
```

返回值

对称矩阵的索引值。

25.2.41 SymMatrixArray 类

为方便用户对一组 C# *SymMatrix 类* 对象进行操作，杉数求解器的 C# 接口设计了 SymMatrixArray 类，提供了以下成员方法：

SymMatrixArray.SymMatrixArray()

SymMatrixArray 的构造函数。

概要

```
SymMatrixArray()
```

SymMatrixArray.GetMatrix()

获取对称矩阵数组里指定下标的对称矩阵。

概要

```
SymMatrix GetMatrix(int idx)
```

参量

`idx`: 对称矩阵的下标。

返回值

指定下标的对称矩阵。

SymMatrixArray.PushBack()

向对称矩阵数组里附加一个对称矩阵。

概要

```
void PushBack(SymMatrix mat)
```

参量

`mat`: 对称矩阵。

SymMatrixArray.Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

`n`: 容纳 n 项的空间。

SymMatrixArray.Size()

获取对称矩阵数组里对称矩阵个数。

概要

```
int Size()
```

返回值

对称矩阵个数。

25.2.42 SymMatExpr 类

对称矩阵表达式对于对称矩阵的线性组合，其计算结果实际还是一个对称矩阵。表达式的好处是可以延迟计算结果矩阵，直到设置半定约束或者半定目标函数时。SymMatExpr 类是杉数优化求解器中对称矩阵表达式的封装，提供了以下成员方法：

SymMatExpr.SymMatExpr()

对称矩阵表达式的默认构造函数。

概要

```
SymMatExpr()
```

SymMatExpr.SymMatExpr()

使用对称矩阵和其系数构造的表达式。

概要

```
SymMatExpr(SymMatrix mat, double coeff)
```

参量

`mat`: 添加的这一项对应的对称矩阵。

`coeff`: 可选，添加的这一项对应的参数。默认值为 1.0。

SymMatExpr.AddSymMatExpr()

添加一个对称矩阵表达式的项，并乘以倍数。

概要

```
void AddSymMatExpr(SymMatExpr expr, double mult)
```

参量

`expr`: 需要添加的对称矩阵表达式。

`mult`: 可选的系数倍数，默认值为 1.0。

SymMatExpr.AddTerm()

向对称矩阵表达式中添加一项。

概要

```
bool AddTerm(SymMatrix mat, double coeff)
```

参量

`mat`: 新项中的对称矩阵。

`coeff`: 新项中的系数。

返回值

布尔值，表示新项是否成功添加。

SymMatExpr.AddTerms()

向表达式中添加多个项。

概要

```
int AddTerms(SymMatrixArray mats, double[] coeffs)
```

参量

`mats`: 新项中的对称矩阵数组。

`coeffs`: 新项中的系数数组。

返回值

增加的项数。如果返回负值，至少有一项添加失败。

SymMatExpr.AddTerms()

向表达式中添加多个项。

概要

```
int AddTerms(SymMatrix[] mats, double[] coeffs)
```

参量

`mats`: 新项中的对称矩阵数组。

`coeffs`: 新项中的系数数组。

返回值

增加的项数。如果返回负值，至少有一项添加失败。

SymMatExpr.AddTerms()

向表达式中添加多个项。

概要

```
int AddTerms(SymMatrix[] mats, double coeff)
```

参量

`mats`: 新项中的对称矩阵数组。

`coeff`: 新项中的公用系数，默认值为 1.0。

返回值

增加的项数。如果返回负值，至少有一项添加失败。

SymMatExpr.Clone()

深度拷贝对称矩阵表达式。

概要

```
SymMatExpr Clone()
```

返回值

新的表达式对象。

SymMatExpr.GetCoeff()

获取表达式中指定索引值对应项的系数。

概要

```
double GetCoeff(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值对应的表达式项的系数。

SymMatExpr.GetDim()

获取表达式中对称矩阵的维度。

概要

```
int GetDim()
```

返回值

对称矩阵的维度。

SymMatExpr.GetSymMat()

获取表达式指定索引值对应项中的对称矩阵。

概要

```
SymMatrix GetSymMat(int i)
```

参量

i: 指定索引值。

返回值

指定索引值对应项的对称矩阵对象。

SymMatExpr.Multiply()

对对称矩阵表达式自乘常数。

概要

```
void Multiply(double c)
```

参量

c: 常数操作数。

SymMatExpr.Remove()

删除表达式中指定索引值的项。

概要

```
void Remove(int idx)
```

参量

idx: 指定索引值。

SymMatExpr.Remove()

删除对称矩阵表达式中与指定对称矩阵相关的项。

概要

```
void Remove(SymMatrix mat)
```

参量

mat: 指定的对称矩阵。

SymMatExpr.Reserve()

预分配大小为 n 项的空间。

概要

```
void Reserve(int n)
```

参量

`n`: 容纳 n 项的空间。

SymMatExpr.SetCoeff()

设置表达式指定索引值项数中的系数。

概要

```
void SetCoeff(int i, double val)
```

参量

`i`: 指定索引值。

`val`: 指定索引值项数中的参数。

SymMatExpr.Size()

获取对称矩阵表达式中的项数。

概要

```
long Size()
```

返回值

对称矩阵表达式中的项数。

25.2.43 CallbackBase 类

CallbackBase 类给用户提供了在求解过程中介入的接口。这个是抽象类，用户需要自己实现虚函数 `virtual void CallbackBase::callback()` 才能创建实例，用来作为 `Model::SetCallback(CallbackBase cb, int cbctx)` 方法的第一个参数传入。CallbackBase 类也提供了以下可以继承的成员方法：

CallbackBase.CallbackBase()

CallbackBase 的构造函数，实现 ICallback 接口。

概要

```
CallbackBase()
```

CallbackBase.AddLazyConstr()

向模型中增加一个惰性约束。

概要

```
void AddLazyConstr(
    Expr lhs,
    char sense,
    Expr rhs)
```

参量

`lhs`: 惰性约束的左侧表达式。

`sense`: 惰性约束的类型。

`rhs`: 惰性约束的右侧表达式。

CallbackBase.AddLazyConstr()

向模型中增加一个惰性约束。

概要

```
void AddLazyConstr(ConstrBuilder builder)
```

参量

`builder`: 惰性约束生成器。

CallbackBase.AddLazyConstr()

向模型中增加一个惰性约束。

概要

```
void AddLazyConstr(
    Expr lhs,
    char sense,
    double rhs)
```

参量

lhs: 惰性约束表达式。

sense: 惰性约束的类型。

rhs: 惰性约束的右侧值。

CallbackBase.AddLazyConstrs()

向模型中增加多个惰性约束。

概要

```
void AddLazyConstrs(ConstrBuilderArray builders)
```

参量

builders: 一组惰性约束生成器。

CallbackBase.AddUserCut()

向模型中增加一个割平面。

概要

```
void AddUserCut(
    Expr lhs,
    char sense,
    double rhs)
```

参量

lhs: 割平面表达式。

sense: 割平面的类型。

rhs: 割平面的右侧值。

CallbackBase.AddUserCut()

向模型中增加一个割平面。

概要

```
void AddUserCut(
    Expr lhs,
    char sense,
    Expr rhs)
```

参量

`lhs`: 割平面的左侧表达式。

`sense`: 割平面的类型。

`rhs`: 割平面的右侧表达式。

CallbackBase.AddUserCut()

向模型中增加一个割平面。

概要

```
void AddUserCut(ConstrBuilder builder)
```

参量

`builder`: 割平面生成器。

CallbackBase.AddUserCuts()

向模型中增加多个割平面。

概要

```
void AddUserCuts(ConstrBuilderArray builders)
```

参量

`builders`: 一组割平面生成器。

CallbackBase.callback()

定义在 ICallback 接口里的纯虚函数，需要用户覆盖对应实现。

概要

```
void callback()
```

CallbackBase.GetDblInfo()

在回调中获取指定信息名的双精度值。

概要

```
double GetDblInfo(string cbinfo)
```

参量

`cbinfo`: 回调中的信息名。

返回值

所需的信息值。

CallbackBase.GetIncumbent()

在回调中获取指定变量的最优可行解。

概要

```
double GetIncumbent(Var var)
```

参量

var: 指定的变量。

返回值

给定变量对应的最优可行解。

CallbackBase.GetIncumbent()

在回调中获取一组变量的最优可行解。

概要

```
double[] GetIncumbent(VarArray vars)
```

参量

vars: 变量数组。

返回值

一组变量对应的最优可行解。

CallbackBase.GetIncumbent()

在回调中获取一组变量的最优可行解。

概要

```
double[] GetIncumbent(Var[] vars)
```

参量

vars: 变量数组。

返回值

一组变量对应的最优可行解。

CallbackBase.GetIncumbent()

在回调中获取全部变量的最优可行解。

概要

```
double[] GetIncumbent()
```

返回值

全部变量对应的最优可行解。

CallbackBase.GetIntInfo()

在回调中获取指定信息名的整数值。

概要

```
int GetIntInfo(string cbinfo)
```

参量

`cbinfo`: 回调中的信息名。

返回值

所需的信息值。

CallbackBase.GetRelaxSol()

在回调中获取指定变量的线性松弛解。

概要

```
double GetRelaxSol(Var var)
```

参量

`var`: 指定的变量。

返回值

给定变量对应的线性松弛解。

CallbackBase.GetRelaxSol()

在回调中获取一组变量的线性松弛解。

概要

```
double[] GetRelaxSol(VarArray vars)
```

参量

`vars`: 变量数组。

返回值

一组变量对应的线性松弛解。

CallbackBase.GetRelaxSol()

在回调中获取一组变量的线性松弛解。

概要

```
double[] GetRelaxSol(Var[] vars)
```

参量

vars: 变量数组。

返回值

一组变量对应的线性松弛解。

CallbackBase.GetRelaxSol()

在回调中获取全部变量的线性松弛解。

概要

```
double[] GetRelaxSol()
```

返回值

全部变量对应的线性松弛解。

CallbackBase.GetSolution()

在回调中获取指定变量的解。

概要

```
double GetSolution(Var var)
```

参量

var: 指定的变量。

返回值

给定变量对应的解。

CallbackBase.GetSolution()

在回调中获取一组变量的解。

概要

```
double[] GetSolution(VarArray vars)
```

参量

vars: 变量数组。

返回值

一组变量对应的解。

CallbackBase.GetSolution()

在回调中获取一组变量的解。

概要

```
double[] GetSolution(Var[] vars)
```

参量

`vars`: 变量数组。

返回值

一组变量对应的解。

CallbackBase.GetSolution()

在回调中获取全部变量的解。

概要

```
double[] GetSolution()
```

返回值

全部变量对应的解。

CallbackBase.Interrupt()

中断回调中正在求解的问题。

概要

```
void Interrupt()
```

CallbackBase.LoadSolution()

向模型中添加自定义解。

概要

```
double LoadSolution()
```

返回值

解对应的目标函数值。

CallbackBase.SetSolution()

在回调中对给定的变量设置自定义解。

概要

```
void SetSolution(Var var, double val)
```

参量

var: 变量对象。

val: 双精度值。

CallbackBase.SetSolution()

在回调中对一组变量设置自定义解。

概要

```
void SetSolution(VarArray vars, double[] vals)
```

参量

vars: 变量数组。

vals: 双精度值数组。

CallbackBase.SetSolution()

在回调中对一组变量设置自定义解。

概要

```
void SetSolution(Var[] vars, double[] vals)
```

参量

vars: 变量数组。

vals: 双精度值数组。

CallbackBase.Where()

获取回调中的上下文。

概要

```
int Where()
```

返回值

上下文对应的整数值。

25.2.44 ProbBuffer 类

ProbBuffer 类是字符流缓冲区的封装，提供了以下成员方法：

ProbBuffer.ProbBuffer()

ProbBuffer 的构造函数。

概要

```
ProbBuffer(int sz)
```

参量

sz: ProbBuffer 的初始大小

ProbBuffer.GetData()

获取在缓存里的问题字符流。

概要

```
string GetData()
```

返回值

缓存中的问题字符流。

ProbBuffer.Resize()

调整缓存到给定大小。

概要

```
void Resize(int sz)
```

参量

sz: 指定的缓存大小。

ProbBuffer.Size()

获取缓存大小。

概要

```
int Size()
```

返回值

缓存大小。

25.2.45 CoptException 类

CoptException 类是杉数优化求解器的错误处理相关操作的封装。当方法调用对应的杉数求解器底层接口发生错误时，则抛出 CoptException 类的异常，提供了以下属性值访问相应的错误信息：

CoptException.CoptException()

CoptException 的构造函数。

概要

```
CoptException(int code, string msg)
```

参量

`code`: 异常错误代码。

`msg`: 异常的错误信息。

CoptException.GetCode()

获取异常错误代码。

概要

```
int GetCode()
```

返回值

异常错误代码。

第 26 章　Java API 参考手册

为了方便用户对复杂应用场景进行建模并调用求解器进行求解，杉数求解器设计并提供了 Java 接口，本章节将对 Java 接口的常量和功能进行阐述。

26.1 Java 常量类

常量类定义了使用 Java 接口建模时必要的常数，包括一般常数、解状态、属性、信息和参数五大类常量。本节将依次阐述上述五类常量的内容与使用方法。

26.1.1 一般常数

一般常数定义在 `Consts` 类里. 用户可以通过 `copt` 导入包名作为前缀访问一般常数，如 `copt.Consts.XXXX`。

一般常数中的内容，详见一般常数章节。

26.1.2 解状态

关于解状态的常数定义在 `Status` 类里. 用户可以通过 `copt` 导入包名的前缀访问解状态常数，如 `copt.Status.XXXX`。

解状态常数中的内容，详见一般常数章节：求解状态。

26.1.3 属性

Java API 属性常数中的内容，详见属性章节。

在 Java API 中, COPT 属性定义在 `DblAttr` 和 `IntAttr` 类里。用户可以通过 `copt.DblAttr` 访问 COPT 浮点型属性；通过 `copt.IntAttr` 访问 COPT 整型属性。

在 Java API 中，通过指定属性名称获取属性取值，提供的函数如下，具体请参考*Java Model 类* 。

- `Model.getIntAttr()`：获取整数型属性取值
- `Model.getDblAttr()`：获取浮点型属性取值

26.1.4 信息

Java API 信息常数中的内容，详见信息章节。

在 Java API 中，信息常数定义在 `DblInfo` 类里。用户可以通过命名空间中的前缀 `copt`（通常可以省略）`copt.DblInfo.` 访问信息常数。

例如，`copt.DblInfo.Obj` 表示变量在目标函数中系数。

26.1.5 Callback 信息

Java API Callback 信息常数中的内容，详见信息章节：*Callback* 相关信息。

在 Java API 中，Callback 相关的信息常数定义在 `CbInfo` 类里，用户可以通过命名空间中的前缀 `copt`（通常可以省略）`copt.CbInfo.` 访问 Callback 信息常数。

例如，`copt.CbInfo.BestObj` 表示当前最优目标函数值。

26.1.6 参数

优化参数控制 COPT 求解器优化算法的行为。

Java API 参数常数中的内容，详见参数章节。

在 Java API 中，COPT 优化参数定义在 `DblParam` 和 `IntParam` 类里。用户可以通过 `copt.DblParam` 访问 COPT 浮点型参数；通过 `copt.IntParam` 访问 COPT 整型参数。

在 Java API 中，通过指定参数名称获取和设置参数取值。提供的函数如下，具体请参考*Java Model 类* 。

- 获取指定参数的详细信息（当前值/最大值/最小值）：`Model.getParamInfo()`
- 获取指定整数型/浮点型参数当前取值：`Model.getIntParam()` / `Model.getDblParam()`
- 设置指定整数型/浮点型参数值：`Model.setIntParam()` / `Model.setDblParam()`

26.2 Java 建模类

本章节详细描述杉数优化求解器的 Java 接口的优化建模类，方便用户在快速构建复杂场景下的优化模型时对其功能和使用的查询。

26.2.1 Envr 类

创建求解环境对象是每个求解过程中必不可少的第一步。而每个求解模型都和一个 Envr 类关联。用户必须首先创建一个求解环境，才能在此基础上创建一个或者多个求解模型。

Envr.Envr()

COPT Envr 类的构造函数。

概要

```
Envr()
```

Envr.Envr()

COPT Envr 类的构造函数。

概要

```
Envr(String licDir)
```

参量

`licDir`: 用户指定的路径，包含本地授权文档或者客户端配置文件。

Envr.Envr()

COPT Envr 类的构造函数。

概要

```
Envr(EnvrConfig config)
```

参量

`config`: COPT Envr 配置类，包含远程连接的设置。

Envr.close()

关闭远程连接。之前获得的远程授权失效，对当前环境类下创建的全部问题立即生效。

概要

```
void close()
```

Envr.createModel()

创建模型。

概要

```
Model createModel(String name)
```

参量

`name`: 自定义的模型名称。

返回值

模型。

26.2.2 EnvrConfig 类

如果用户通过连接远程服务的方式启动杉数优化求解器，可以创建环境配置类来设置 COPT 作为客户端的配置。

EnvrConfig.EnvrConfig()

Envr 配置类的构造函数。

概要

```
EnvrConfig()
```

EnvrConfig.set()

设置 Envr 配置类里的内容。

概要

```
void set(String name, String value)
```

参量

`name`: 配置的关键词。

`value`: 配置的内容。

26.2.3 Model 类

Model 类是杉数优化求解器模型相关操作的封装，提供了以下成员方法：

Model.Model()

模型的构造函数。

概要

```
Model(Envr env, String name)
```

参量

`env`: 关联的环境对象。

`name`: 模型名。

Model.addCone()

向模型中增加一个锥约束。

概要

```
Cone addCone(
    int dim,
    int type,
    char[] pvtype,
    String prefix)
```

参量

`dim`: 锥约束的维度。

`type`: 锥约束的类型。

`pvtype`: 锥约束中变量的类型。

`prefix`: 锥约束中变量的名称前缀。

返回值

新的锥约束。

Model.addCone()

向模型中增加一个锥约束。

概要

```
Cone addCone(ConeBuilder builder)
```

参量

`builder`: 锥约束生成器。

返回值

新的锥约束。

Model.addCone()

向模型中增加一个锥约束。

概要

```
Cone addCone(Var[] vars, int type)
```

参量

`vars`: 参与锥约束的变量数组。

`type`: 锥约束的类型。

返回值

新的锥约束。

Model.addCone()

向模型中增加一个锥约束。

概要

```
Cone addCone(VarArray vars, int type)
```

参量

vars: 参与锥约束的变量数组。

type: 锥约束的类型。

返回值

新的锥约束。

Model.addConstr()

向模型中增加一个线性约束。

概要

```
Constraint addConstr(
    Expr expr,
    char sense,
    double rhs,
    String name)
```

参量

expr: 新的约束表达式。

sense: 约束的类型。

rhs: 新约束的右侧值。

name: 新约束的名称。

返回值

新约束。

Model.addConstr()

向模型中增加一个线性约束。

概要

```
Constraint addConstr(
    Expr expr,
    char sense,
    Var var,
    String name)
```

参量

`expr`: 新的约束表达式。

`sense`: 约束的类型。

`var`: 作为右侧值的变量，而不是约束范围类型。

`name`: 新约束的名称。

返回值

新约束。

Model.addConstr()

向模型中增加一个线性约束。

概要

```
Constraint addConstr(
    Expr lhs,
    char sense,
    Expr rhs,
    String name)
```

参量

`lhs`: 新约束的左侧值。

`sense`: 约束的类型。

`rhs`: 新约束的右侧值。

`name`: 新约束的名称。

返回值

新约束。

Model.addConstr()

向模型中增加一个线性约束。

概要

```
Constraint addConstr(
    Expr expr,
    double lb,
    double ub,
    String name)
```

参量

`expr`: 新的约束表达式。

`lb`: 约束的下界。

`ub`: 约束的上界。

`name`: 新约束的名称。

返回值

新约束。

Model.addConstr()

向模型中增加一个线性约束。

概要

```
Constraint addConstr(ConstrBuilder builder, String name)
```

参量

`builder`: 新约束生成器。

`name`: 新约束的名称。

返回值

新约束。

Model.addConstrs()

向模型中增加线性约束。

概要

```
ConstrArray addConstrs(
    int count,
    char[] senses,
```

```
    double[] rhss,
    String prefix)
```

参量

`count`: 添加的线性约束数目。

`senses`: 约束类型的数组，而不是范围类型。

`rhss`: 新约束的右侧值。

`prefix`: 新约束的名称前缀。

返回值

新约束构成的 ConstrArray 类。

Model.addConstrs()

向模型中增加线性约束。

概要

```
ConstrArray addConstrs(
    int count,
    double[] lbs,
    double[] ubs,
    String prefix)
```

参量

`count`: 添加的线性约束数目。

`lbs`: 新约束的下界数组。

`ubs`: 新约束的上界数组。

`prefix`: 可选，新约束的名称前缀。

返回值

新约束构成的 ConstrArray 类。

Model.addConstrs()

向模型中增加线性约束。

概要

```
ConstrArray addConstrs(ConstrBuilderArray builders, String prefix)
```

参量

`builders`: 线性约束生成器。

`prefix`: 新约束的名称前缀。

返回值

新约束构成的 ConstrArray 类。

Model.addDenseMat()

向模型中增加一个密致对称矩阵。

概要

```
SymMatrix addDenseMat(int dim, double[] vals)
```

参量

`dim`: 密致对称矩阵的维度。

`vals`: 非零元值数组。按列次序填充非零元，到数组长度或者对称矩阵最大长度位置。

返回值

新对称矩阵对象。

Model.addDenseMat()

向模型中增加一个密致对称矩阵。

概要

```
SymMatrix addDenseMat(int dim, double val)
```

参量

`dim`: 密致对称矩阵的维度。

`val`: 同一个非零元值，用来填充对称矩阵。

返回值

新对称矩阵对象。

Model.addDiagMat()

向模型中增加一个对角矩阵。

概要

```
SymMatrix addDiagMat(int dim, double val)
```

参量

`dim`: 对角矩阵的维度。

`val`: 同一个非零元值，用来填充对角元素。

返回值

新对角矩阵对象。

Model.addDiagMat()

向模型中增加一个对角矩阵。

概要

```
SymMatrix addDiagMat(int dim, double[] vals)
```

参量

`dim`: 对角矩阵的维度。

`vals`: 双精度值数组，用来填充对角元素。

返回值

新对角矩阵对象。

Model.addDiagMat()

向模型中增加一个对角矩阵。

概要

```
SymMatrix addDiagMat(
    int dim,
    double val,
    int offset)
```

参量

`dim`: 对角矩阵的维度。

`val`: 同一个非零元值，用来填充对角元素。

`offset`: 相对于标准对角线的平移量。

返回值

新对角矩阵对象。

Model.addDiagMat()

向模型中增加一个对角矩阵。

概要

```
SymMatrix addDiagMat(
    int dim,
    double[] vals,
    int offset)
```

参量

`dim`: 对角矩阵的维度。

`vals`: 双精度值数组，用来填充对角元素。

`offset`: 相对于标准对角线的平移量。

返回值

新对角矩阵对象。

Model.addEyeMat()

向模型中增加一个单位矩阵。

概要

```
SymMatrix addEyeMat(int dim)
```

参量

`dim`: 单位矩阵的维度。

返回值

新单位矩阵对象。

Model.addGenConstrIndicator()

向模型中增加一个类型指示型的一般约束（GenConstr）。

概要

```
GenConstr addGenConstrIndicator(GenConstrBuilder builder)
```

参量

`builder`: 一般约束（GenConstr）生成器。

返回值

类型指示型的新一般约束（GenConstr）。

Model.addGenConstrIndicator()

向模型中增加一个类型指示型的一般约束（GenConstr）。

概要

```
GenConstr addGenConstrIndicator(
    Var binvar,
    int binval,
    ConstrBuilder builder)
```

参量

`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 或 1）。

`builder`: 线性约束生成器。

返回值

类型指示型的新一般约束（GenConstr）。

Model.addGenConstrIndicator()

向模型中增加一个类型指示型的一般约束（GenConstr）。

概要

```
GenConstr addGenConstrIndicator(
    Var binvar,
    int binval,
    Expr expr,
    char sense,
    double rhs)
```

参量

`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 或 1）。

`expr`: 新的线性约束表达式。

`sense`: 新的线性约束类型。

`rhs`: 新的线性约束右侧值。

返回值

类型指示型的新一般约束（GenConstr）。

Model.addLazyConstr()

向模型中增加一个惰性约束。

概要

```
void addLazyConstr(
    Expr lhs,
    char sense,
    double rhs,
    String name)
```

参量

lhs: 惰性约束表达式。

sense: 惰性约束的类型。

rhs: 惰性约束的右侧值。

name: 可选，新惰性约束的名称。

Model.addLazyConstr()

向模型中增加一个惰性约束。

概要

```
void addLazyConstr(
    Expr lhs,
    char sense,
    Expr rhs,
    String name)
```

参量

lhs: 惰性约束的左侧表达式。

sense: 惰性约束的类型。

rhs: 惰性约束的右侧表达式。

name: 可选，新惰性约束的名称。

Model.addLazyConstr()

向模型中增加一个惰性约束。

概要

```
void addLazyConstr(ConstrBuilder builder, String name)
```

参量

builder: 惰性约束生成器。

name: 可选，新惰性约束的名称。

Model.addLazyConstrs()

向模型中增加多个惰性约束。

概要

```
void addLazyConstrs(ConstrBuilderArray builders, String prefix)
```

参量

`builders`: 一组惰性约束生成器。

`prefix`: 新惰性约束名称的前缀。

Model.addLmiConstr()

向模型中增加一个 LMI 约束。

概要

```
LmiConstraint addLmiConstr(LmiExpr expr, String name)
```

参量

`expr`: 新的 LMI 约束表达式。

`name`: 可选，新 LMI 约束的名称。

返回值

新 LMI 约束对象。

Model.addOnesMat()

向模型中增加一个用非零元 1 填充的密致对称矩阵。

概要

```
SymMatrix addOnesMat(int dim)
```

参量

`dim`: 密致对称矩阵的维度。

返回值

新对称矩阵对象。

Model.addPsdConstr()

向模型中增加一个半定约束。

概要

```
PsdConstraint addPsdConstr(
    PsdExpr expr,
    char sense,
    double rhs,
    String name)
```

参量

`expr`: 新的半定约束表达式。

`sense`: 半定约束的类型。

`rhs`: 新半定约束的右侧值。

`name`: 可选，新半定约束的名称。

返回值

新半定约束对象。

Model.addPsdConstr()

向模型中增加一个半定约束。

概要

```
PsdConstraint addPsdConstr(
    PsdExpr expr,
    double lb,
    double ub,
    String name)
```

参量

`expr`: 新的半定约束表达式。

`lb`: 半定约束的下界。

`ub`: 半定约束的上界。

`name`: 可选，新半定约束的名称。

返回值

新半定约束。

Model.addPsdConstr()

向模型中增加一个半定约束。

概要

```
PsdConstraint addPsdConstr(
    PsdExpr lhs,
    char sense,
    PsdExpr rhs,
    String name)
```

参量

lhs: 新半定约束的左侧表达式。

sense: 半定约束的类型。

rhs: 新半定约束的右侧表达式。

name: 可选，新半定约束的名称。

返回值

新半定约束对象。

Model.addPsdConstr()

向模型中增加一个半定约束。

概要

```
PsdConstraint addPsdConstr(PsdConstrBuilder builder, String name)
```

参量

builder: 新半定约束生成器。

name: 可选，新半定约束的名称。

返回值

新半定约束对象。

Model.addPsdVar()

向模型中增加半定变量。

概要

```
PsdVar addPsdVar(int dim, String name)
```

参量

dim: 新半定变量的维度。

name: 新半定变量的名称。

返回值

新半定变量对象。

Model.addPsdVars()

向模型中添加一些半定变量。

概要

```
PsdVarArray addPsdVars(
    int count,
    int[] dims,
    String prefix)
```

参量

count: 新半定变量的数目。

dims: 整数数组，包含了半定变量的维度。

prefix: 新半定变量的名称前缀。

返回值

新添加的半定变量数组。

Model.addQConstr()

向模型中增加一个二次约束。

概要

```
QConstraint addQConstr(
    QuadExpr expr,
    char sense,
    double rhs,
    String name)
```

参量

expr: 新的二次约束表达式。

sense: 约束的类型。

rhs: 新约束的右侧值。

name: 可选，新二次约束的名称。

返回值

新二次约束对象。

Model.addQConstr()

向模型中增加一个二次约束。

概要

```
QConstraint addQConstr(
    QuadExpr lhs,
    char sense,
    QuadExpr rhs,
    String name)
```

参量

lhs: 新二次约束的左侧表达式。

sense: 二次约束的类型。

rhs: 新二次约束的右侧表达式。

name: 可选，新二次约束的名称。

返回值

新二次约束对象。

Model.addQConstr()

向模型中增加一个二次约束。

概要

```
QConstraint addQConstr(QConstrBuilder builder, String name)
```

参量

builder: 新二次约束生成器。

name: 可选，新二次约束的名称。

返回值

新二次约束对象。

Model.addSos()

向模型中增加一个 SOS 约束。

概要

```
Sos addSos(
    VarArray vars,
    double[] weights,
    int type)
```

参量

`vars`: 参与 SOS 约束的变量构成的 VarArray 类。

`weights`: 参与 SOS 约束变量的权重数组。

`type`: SOS 约束的类型。

返回值

新 SOS 约束。

Model.addSos()

向模型中增加一个 SOS 约束。

概要

```
Sos addSos(SosBuilder builder)
```

参量

`builder`: SOS 约束生成器。

返回值

新 SOS 约束。

Model.addSos()

向模型中增加一个 SOS 约束。

概要

```
Sos addSos(
    Var[] vars,
    double[] weights,
    int type)
```

参量

vars: 参与 SOS 约束的变量数组。

weights: 参与 SOS 约束变量的权重数组。

type: SOS 约束的类型。

返回值

新 SOS 约束。

Model.addSparseMat()

向模型中增加一个稀疏对称矩阵。

概要

```
SymMatrix addSparseMat(
    int dim,
    int nElems,
    int[] rows,
    int[] cols,
    double[] vals)
```

参量

dim: 稀疏对称矩阵的维度。

nElems: 稀疏对称矩阵中的非零元个数。

rows: 整数数组，保存了非零元的行号。

cols: 整数数组，保存了非零元的列号。

vals: 非零元值数组。

返回值

新对称矩阵对象。

Model.addSymMat()

根据给定的对称矩阵表达式，向模型中增加一个对称矩阵。

概要

```
SymMatrix addSymMat(SymMatExpr expr)
```

参量

expr: 对称矩阵表达式对象。

返回值

结果对称矩阵对象。

Model.addUserCut()

向模型中增加一个割平面。

概要

```
void addUserCut(
    Expr lhs,
    char sense,
    double rhs,
    String name)
```

参量

lhs: 割平面表达式。

sense: 割平面的类型。

rhs: 割平面的右侧值。

name: 可选，新割平面的名称。

Model.addUserCut()

向模型中增加一个割平面。

概要

```
void addUserCut(
    Expr lhs,
    char sense,
    Expr rhs,
    String name)
```

参量

lhs: 割平面的左侧表达式。

sense: 割平面的类型。

rhs: 割平面的右侧表达式。

name: 可选，新割平面的名称。

Model.addUserCut()

向模型中增加一个割平面。

概要

```
void addUserCut(ConstrBuilder builder, String name)
```

参量

builder: 割平面生成器。

name: 可选，新割平面的名称。

Model.addUserCuts()

向模型中增加多个割平面。

概要

```
void addUserCuts(ConstrBuilderArray builders, String prefix)
```

参量

builders: 一组割平面生成器。

prefix: 新割平面名称的前缀。

Model.addVar()

向模型中增加一个变量以及相关不为零的系数作为列。

概要

```
Var addVar(
    double lb,
    double ub,
    double obj,
    char vtype,
    String name)
```

参量

lb: 新变量的下界。

ub: 新变量的上界。

obj: 新变量在目标函数里的系数。

vtype: 新变量的类型。

name: 可选，新变量的名称。

返回值

新变量。

Model.addVar()

向模型中增加一个变量以及相关不为零的系数作为列。

概要

```
Var addVar(
    double lb,
    double ub,
    double obj,
    char vtype,
    Column col,
    String name)
```

参量

`lb`: 新变量的下界。

`ub`: 新变量的上界。

`obj`: 新变量在目标函数里的系数。

`vtype`: 新变量的类型。

`col`: 列对象，用于指定新变量所属的一组约束.

`name`: 新变量的名称。

返回值

新变量。

Model.addVars()

向模型中增加变量。

概要

```
VarArray addVars(
    int count,
    char vtype,
    String prefix)
```

参量

`count`: 添加变量的数量。

`vtype`: 新变量的类型。

`prefix`: 新变量的名称前缀。

返回值

新变量构成的 VarArray 类。

Model.addVars()

向模型中增加变量。

概要

```
VarArray addVars(
    int count,
    double lb,
    double ub,
    double obj,
    char vtype,
    String prefix)
```

参量

`count`: 添加变量的数量。

`lb`: 新变量的下界。

`ub`: 新变量的上界。

`obj`: 新变量在目标函数里的系数。

`vtype`: 新变量的类型。

`prefix`: 新变量的名称前缀。

返回值

新变量构成的 VarArray 类。

Model.addVars()

向模型中增加变量。

概要

```
VarArray addVars(
    int count,
    double[] lbs,
    double[] ubs,
    double[] objs,
    char[] types,
    String prefix)
```

参量

`count`: 添加变量的数量。

`lbs`: 新变量的下界数组, 如果为空，下界为 0。

`ubs`: 新变量的上界数组, 如果为空，上界为正无穷或者 1 对于二进制变量。

`objs`: 新变量在目标函数中的系数数组，如果为空，则为 0。

`types`: 新变量的类型，如果为空，则为连续变量。

`prefix`: 新变量的名称前缀。

返回值

新变量构成的 VarArray 类。

Model.addVars()

向模型中增加变量。

概要

```
VarArray addVars(
    double[] lbs,
    double[] ubs,
    double[] objs,
    char[] types,
    Column[] cols,
    String prefix)
```

参量

`lbs`: 新变量的下界数组, 如果为空，下界为 0。

`ubs`: 新变量的上界数组, 如果为空，上界为正无穷或者 1 对于二进制变量。

`objs`: 新变量在目标函数中的系数数组，如果为空，则为 0。

`types`: 新变量的类型，如果为空，则为连续变量。

`cols`: 列对象数组，用于指定每个新变量所属的一组约束。

`prefix`: 新变量的名称前缀。

返回值

新变量构成的 VarArray 类。

Model.addVars()

向模型中增加变量。

概要

```
VarArray addVars(
    double[] lbs,
    double[] ubs,
    double[] objs,
    char[] types,
    ColumnArray cols,
    String prefix)
```

参量

`lbs`: 新变量的下界数组, 如果为空，下界为 0。

`ubs`: 新变量的上界数组, 如果为空，上界为正无穷或者 1 对于二进制变量。

`objs`: 新变量在目标函数中的系数数组，如果为空，则为 0。

`types`: 新变量的类型，如果为空，则为连续变量。

`cols`: 列对象构成的 ColumnArray 类，用于指定每个新变量所属的一组约束。

`prefix`: 新变量的名称前缀。

返回值

新变量构成的 VarArray 类。

Model.clear()

删除所有设置以及问题本身。

概要

```
void clear()
```

Model.clone()

深度复制 COPT 模型。

概要

```
Model clone()
```

返回值

新的模型对象。

Model.computeIIS()

计算不可行模型的 IIS。

概要

```
void computeIIS()
```

Model.delPsdObj()

删除模型目标函数的半定部分（保留线性部分）。

概要

```
void delPsdObj()
```

Model.delQuadObj()

删除模型目标函数的二次部分（保留线性部分）。

概要

```
void delQuadObj()
```

Model.feasRelax()

计算不可行模型的可行化松弛。

概要

```
void feasRelax(
    VarArray vars,
    double[] colLowPen,
    double[] colUppPen,
    ConstrArray cons,
    double[] rowBndPen,
    double[] rowUppPen)
```

参量

`vars`: 变量构成的 VarArray 类对象。

`colLowPen`: 变量下界的惩罚系数。

`colUppPen`: 变量上界的惩罚系数。

`cons`: 约束构成的 ConstrArray 类对象。

`rowBndPen`: 约束右端项的惩罚系数。

`rowUppPen`: 约束上界的惩罚系数。

Model.feasRelax()

计算不可行模型的可行化松弛。

概要

```
void feasRelax(int ifRelaxVars, int ifRelaxCons)
```

参量

ifRelaxVars: 是否松弛变量。

ifRelaxCons: 是否松弛约束。

Model.get()

查询与指定变量相关的双精度型信息的值。

概要

```
double[] get(String name, Var[] vars)
```

参量

name: 双精度型信息的名称。

vars: 指定变量数组。

返回值

信息的值。

Model.get()

查询与指定变量相关的双精度型信息的值。

概要

```
double[] get(String name, VarArray vars)
```

参量

name: 双精度型信息的名称。

vars: 指定变量构成的 VarArray 类。

返回值

信息的值。

Model.get()

查询与指定约束相关的双精度型信息的值。

概要

```
double[] get(String name, Constraint[] constrs)
```

参量

`name`: 双精度型信息的名称。

`constrs`: 指定约束数组。

返回值

信息的值。

Model.get()

查询与指定约束相关的双精度型信息的值。

概要

```
double[] get(String name, ConstrArray constrs)
```

参量

`name`: 双精度型信息的名称。

`constrs`: 指定约束构成的 ConstrArray 类。

返回值

信息的值。

Model.get()

查询与指定二次约束相关的双精度型信息的值。

概要

```
double[] get(String name, QConstraint[] constrs)
```

参量

`name`: 双精度型信息的名称。

`constrs`: 指定二次约束数组。

返回值

信息的值。

Model.get()

查询与指定二次约束相关的双精度型信息的值。

概要

```
double[] get(String name, QConstrArray constrs)
```

参量

`name`: 双精度型信息的名称。

`constrs`: 指定二次约束构成的 QConstrArray 类。

返回值

信息的值。

Model.get()

查询与指定半定约束相关的信息的值。

概要

```
double[] get(String name, PsdConstraint[] constrs)
```

参量

`name`: 信息的名称。

`constrs`: 指定半定约束数组。

返回值

输出双精度型数组，保存了信息的值。

Model.get()

查询与指定半定约束相关的信息的值。

概要

```
double[] get(String name, PsdConstrArray constrs)
```

参量

`name`: 信息的名称。

`constrs`: 指定半定约束数组。

返回值

输出双精度型数组，保存了信息的值。

Model.getCoeff()

获取变量在约束中的系数。

概要

```
double getCoeff(Constraint constr, Var var)
```

参量

`constr`: 指定的约束。

`var`: 指定的变量。

返回值

指定的变量在约束中的系数。

Model.getCol()

获取一个包含指定变量参与的所有约束的列。

概要

```
Column getCol(Var var)
```

参量

`var`: 指定变量。

返回值

一个关于指定变量的列。

Model.getColBasis()

获取列的基状态。

概要

```
int[] getColBasis()
```

返回值

列的基状态。

Model.getCone()

获取指定下标值的锥约束。

概要

```
Cone getCone(int idx)
```

参量

`idx`: 指定下标值。

返回值

想获取的锥约束。

Model.getConeBuilders()

获取模型中所有锥约束生成器。

概要

```
ConeBuilderArray getConeBuilders()
```

返回值

锥约束生成器构成的 ConeBuilderArray 类。

Model.getConeBuilders()

获取给定锥约束的生成器。

概要

```
ConeBuilderArray getConeBuilders(Cone[] cones)
```

参量

`cones`: 锥约束数组。

返回值

想获取的的锥约束生成器构成的 ConeBuilderArray 类。

Model.getConeBuilders()

获取给定锥约束的生成器。

概要

```
ConeBuilderArray getConeBuilders(ConeArray cones)
```

参量

`cones`: 锥约束构成的 ConeArray 类。

返回值

想获取的的锥约束生成器构成的 ConeBuilderArray 类。

Model.getCones()

获取模型中所有锥约束。

概要

```
ConeArray getCones()
```

返回值

锥约束构成的 ConeArray 类。

Model.getConstr()

获取模型中指定索引值的约束。

概要

```
Constraint getConstr(int idx)
```

参量

`idx`: 指定索引值。

返回值

想获取的约束。

Model.getConstrBuilder()

获取指定约束的生成器，包括变量，相关的系数，类型和 RHS。

概要

```
ConstrBuilder getConstrBuilder(Constraint constr)
```

参量

`constr`: 指定约束。

返回值

约束生成器类。

Model.getConstrBuilders()

获取模型所有约束生成器。

概要

```
ConstrBuilderArray getConstrBuilders()
```

返回值

约束生成器构成的 ConstrBuilderArray 类。

Model.getConstrByName()

获取模型中指定名称的约束。

概要

```
Constraint getConstrByName(String name)
```

参量

`name`: 指定名称。

返回值

想获取的约束。

Model.getConstrLowerIIS()

获取约束下界的 IIS 状态。

概要

```
int[] getConstrLowerIIS(ConstrArray constrs)
```

参量

`constrs`: 指定约束构成的 ConstrArray 类对象。

返回值

约束下界的 IIS 状态。

Model.getConstrLowerIIS()

获取约束下界的 IIS 状态。

概要

```
int[] getConstrLowerIIS(Constraint[] constrs)
```

参量

`constrs`: 指定约束构成的 ConstrArray 类对象。

返回值

约束下界的 IIS 状态。

Model.getConstrs()

获取模型所有约束。

概要

```
ConstrArray getConstrs()
```

返回值

约束构成的 ConstrArray 类。

Model.getConstrUpperIIS()

获取约束上界的 IIS 状态。

概要

```
int[] getConstrUpperIIS(ConstrArray constrs)
```

参量

`constrs`: 指定约束构成的 ConstrArray 类对象。

返回值

约束上界的 IIS 状态。

Model.getConstrUpperIIS()

获取约束上界的 IIS 状态。

概要

```
int[] getConstrUpperIIS(Constraint[] constrs)
```

参量

`constrs`: 指定约束构成的 ConstrArray 类对象。

返回值

约束上界的 IIS 状态。

Model.getDblAttr()

获取 COPT 双精度型属性的值。

概要

```
double getDblAttr(String attr)
```

参量

`attr`: 双精度型属性的名称。

返回值

双精度型属性的值。

Model.getDblParam()

获取 COPT 双精度型参数的值。

概要

```
double getDblParam(String param)
```

参量

`param`: 双精度型参数的名称。

返回值

双精度型参数的值。

Model.getDblParamInfo()

获取当下的，默认的，最小的，最大的 COPT 双精度型参数的值。

概要

```
double[] getDblParamInfo(String name)
```

参量

`name`: 整型参数的名称。

返回值

当下的，默认的，最小的，最大的 COPT 双精度型参数的值。

Model.getGenConstrIndicator()

获取给定类型指示类一般约束（GenConstr）的生成器。

概要

```
GenConstrBuilder getGenConstrIndicator(GenConstr indicator)
```

参量

`indicator`: 类型指示类一般约束（GenConstr）。

返回值

类型指示类一般约束（GenConstr）的生成器。

Model.getIndicatorIIS()

获取 Indicator 约束的 IIS 状态。

概要

```
int[] getIndicatorIIS(GenConstrArray genconstrs)
```

参量

`genconstrs`: 指定 Indicator 约束构成的 GenConstrArray 类对象。

返回值

Indicator 约束的 IIS 状态。

Model.getIndicatorIIS()

获取 Indicator 约束的 IIS 状态。

概要

```
int[] getIndicatorIIS(GenConstr[] genconstrs)
```

参量

`genconstrs`: 指定 Indicator 约束构成的 GenConstrArray 类对象。

返回值

Indicator 约束的 IIS 状态。

Model.getIntAttr()

获取 COPT 整型属性的值。

概要

```
int getIntAttr(String attr)
```

参量

`attr`: 整型属性的名称。

返回值

整型属性的值。

Model.getIntParam()

获取 COPT 整型参数的值。

概要

```
int getIntParam(String param)
```

参量

param: 整型参数的名称。

返回值

整型参数的值。

Model.getIntParamInfo()

获取当下的，默认的，最小的，最大的 COPT 整型参数的值。

概要

```
int[] getIntParamInfo(String name)
```

参量

name: 整型参数的名称。

返回值

当下的，默认的，最小的，最大的 COPT 整型参数的值。

Model.getLmiCoeff()

获取 LMI 约束中指定变量的系数矩阵。

概要

```
SymMatrix getLmiCoeff(LmiConstraint constr, Var var)
```

参量

constr: 给定的 LMI 约束。

var: 指定的变量。

返回值

指定变量的系数矩阵。

Model.getLmiConstr()

获取模型中指定索引的 LMI 约束。

概要

```
LmiConstraint getLmiConstr(int idx)
```

参量

`idx`: 指定的索引。

返回值

LMI 约束对象。

Model.getLmiConstrByName()

获取模型中指定名称的 LMI 约束。

概要

```
LmiConstraint getLmiConstrByName(String name)
```

参量

`name`: 指定 LMI 约束的名称。

返回值

LMI 约束对象对象。

Model.getLmiConstrs()

获取模型所有 LMI 约束。

概要

```
LmiConstrArray getLmiConstrs()
```

返回值

LMI 约束构成的 LmiConstrArray 对象。

Model.getLmiRhs()

获取 LMI 约束中常数项矩阵。

概要

```
SymMatrix getLmiRhs(LmiConstraint constr)
```

参量

`constr`: 给定的 LMI 约束。

返回值

对应的常数项矩阵。

Model.getLmiRow()

获取指定 LMI 约束对应的 LMI 表达式，包括变量和相关系数矩阵。

概要

```
LmiExpr getLmiRow(LmiConstraint constr)
```

参量

`constr`: 指定的 LMI 约束。

返回值

LMI 约束对应的表达式对象。

Model.getLmiSolution()

获取 LMI 约束的解。

概要

```
Object[] getLmiSolution()
```

返回值

LMI 约束取值和对偶值。

Model.getLpSolution()

获取 LP 解。

概要

```
Object[] getLpSolution()
```

返回值

解的值，约束取值，约束对偶值，变量对偶值。

Model.getObjective()

获取模型的目标函数的线性表达式。

概要

```
Expr getObjective()
```

返回值

线性表达式。

Model.getPoolObjVal()

从解池中获取第 idx 个解的目标函数值。

概要

```
double getPoolObjVal(int idx)
```

参量

idx: 解的编号。

返回值

指定的目标函数值。

Model.getPoolSolution()

从解池中获取第 idx 个解。

概要

```
double[] getPoolSolution(int idx, VarArray vars)
```

参量

idx: 解的编号。

vars: 指定的变量。

返回值

指定的解。

Model.getPoolSolution()

从解池中获取第 idx 个解。

概要

```
double[] getPoolSolution(int idx, Var[] vars)
```

参量

idx: 解的编号。

vars: 指定的变量。

返回值

指定的解。

Model.getPsdCoeff()

获取半定约束中指定半定变量的系数矩阵。

概要

```
SymMatrix getPsdCoeff(PsdConstraint constr, PsdVar var)
```

参量

`constr`: 给定的半定约束。

`var`: 指定的半定变量。

返回值

对应的半定变量的系数矩阵。

Model.getPsdConstr()

获取模型中指定索引值的半定约束。

概要

```
PsdConstraint getPsdConstr(int idx)
```

参量

`idx`: 指定的索引值。

返回值

半定约束对象。

Model.getPsdConstrBuilder()

获取指定约束的生成器，包括半定变量，类型和相关的系数矩阵。

概要

```
PsdConstrBuilder getPsdConstrBuilder(PsdConstraint constr)
```

参量

`constr`: 指定的半定约束。

返回值

半定约束生成器对象。

Model.getPsdConstrBuilders()

获取模型所有半定约束生成器。

概要

```
PsdConstrBuilderArray getPsdConstrBuilders()
```

返回值

半定约束生成器构成的 PsdConstrBuilderArray 对象。

Model.getPsdConstrByName()

获取模型中指定名称的半定约束。

概要

```
PsdConstraint getPsdConstrByName(String name)
```

参量

`name`: 指定的半定约束的名称。

返回值

半定约束对象。

Model.getPsdConstrs()

获取模型所有半定约束。

概要

```
PsdConstrArray getPsdConstrs()
```

返回值

半定约束构成的 PsdConstrArray 类对象。

Model.getPsdObjective()

获取模型目标函数的半定目标。

概要

```
PsdExpr getPsdObjective()
```

返回值

半定目标对应的表达式对象。

Model.getPsdRow()

获取指定半定约束对应的半定表达式，包括半定变量和相关系数矩阵。

概要

```
PsdExpr getPsdRow(PsdConstraint constr)
```

参量

`constr`: 指定的半定约束。

返回值

半定约束的表达式。

Model.getPsdSolution()

获取半定变量与半定约束的解。

概要

```
Object[] getPsdSolution()
```

返回值

解的值，约束取值，约束对偶解，变量对偶解。

Model.getPsdVar()

获取模型中指定索引值的半定变量。

概要

```
PsdVar getPsdVar(int idx)
```

参量

`idx`: 索引值。

返回值

想获取的半定变量。

Model.getPsdVarByName()

获取模型中指定名称的半定变量。

概要

```
PsdVar getPsdVarByName(String name)
```

参量

`name`: 指定名称。

返回值

想获取的半定变量。

Model.getPsdVars()

获取模型所有半定变量。

概要

```
PsdVarArray getPsdVars()
```

返回值

半定变量构成的 PsdVarArray 类对象。

Model.getQConstr()

获取模型中指定索引值的二次约束。

概要

```
QConstraint getQConstr(int idx)
```

参量

`idx`: 指定索引值。

返回值

想获取的二次约束。

Model.getQConstrBuilder()

获取指定约束的生成器，包括变量，相关的系数，类型和 RHS。

概要

```
QConstrBuilder getQConstrBuilder(QConstraint constr)
```

参量

`constr`: 指定约束。

返回值

约束生成器类。

Model.getQConstrBuilders()

获取模型所有约束生成器。

概要

```
QConstrBuilderArray getQConstrBuilders()
```

返回值

约束生成器构成的 QConstrBuilderArray 类。

Model.getQConstrByName()

获取模型中指定名称的二次约束。

概要

```
QConstraint getQConstrByName(String name)
```

参量

`name`: 指定二次约束的名称。

返回值

想获取的二次约束对象。

Model.getQConstrs()

获取模型所有二次约束。

概要

```
QConstrArray getQConstrs()
```

返回值

二次约束构成的 QConstrArray 类对象。

Model.getQuadObjective()

获取模型目标函数的二次目标。

概要

```
QuadExpr getQuadObjective()
```

返回值

二次目标对应的表达式对象。

Model.getQuadRow()

获取参与指定二次约束的变量，以及相关系数。

概要

```
QuadExpr getQuadRow(QConstraint constr)
```

参量

`constr`: 指定二次约束。

返回值

二次约束的表达式。

Model.getRow()

获取参与指定约束的变量，以及相关的系数。

概要

```
Expr getRow(Constraint constr)
```

参量

`constr`: 指定约束。

返回值

约束表达式。

Model.getRowBasis()

获取行的基状态。

概要

```
int[] getRowBasis()
```

返回值

行的基状态。

Model.getSolution()

获取 MIP 解。

概要

```
double[] getSolution()
```

返回值

解的值数组。

Model.getSos()

获取指定索引值的 SOS 约束。

概要

```
Sos getSos(int idx)
```

参量

`idx`: 指定索引值。

返回值

想获取的 SOS 约束。

Model.getSosBuilders()

获取模型中所有 SOS 约束生成器。

概要

```
SosBuilderArray getSosBuilders()
```

返回值

SOS 约束生成器构成的 SosBuilderArray 类。

Model.getSosBuilders()

获取给定 SOS 约束的生成器。

概要

```
SosBuilderArray getSosBuilders(Sos[] soss)
```

参量

`soss`: SOS 约束数组。

返回值

想获取的的 SOS 约束生成器构成的 SosBuilderArray 类。

Model.getSosBuilders()

获取给定 SOS 约束的生成器。

概要

```
SosBuilderArray getSosBuilders(SosArray soss)
```

参量

`soss`: SOS 约束构成的 SosArray 类。

返回值

想获取的的 SOS 约束生成器构成的 SosBuilderArray 类。

Model.getSOSIIS()

获取 SOS 约束的 IIS 状态。

概要

```
int[] getSOSIIS(SosArray soss)
```

参量

`soss`: 指定 SOS 约束构成的 SosArray 类对象。

返回值

SOS 约束的 IIS 状态。

Model.getSOSIIS()

获取 SOS 约束的 IIS 状态。

概要

```
int[] getSOSIIS(Sos[] soss)
```

参量

`soss`: 指定 SOS 约束构成的 SosArray 类对象。

返回值

SOS 约束的 IIS 状态。

Model.getSoss()

获取模型中所有 SOS 约束。

概要

```
SosArray getSoss()
```

返回值

SOS 约束构成的 SosArray 类。

Model.getSymMat()

获取模型中指定索引值的对称矩阵。

概要

```
SymMatrix getSymMat(int idx)
```

参量

`idx`: 指定索引值。

返回值

想获取的对称矩阵。

Model.getVar()

获取模型中指定索引值的变量。

概要

```
Var getVar(int idx)
```

参量

`idx`: 索引值。

返回值

想获取的变量。

Model.getVarByName()

获取模型中指定名称的变量。

概要

```
Var getVarByName(String name)
```

参量

`name`: 指定名称。

返回值

想获取的变量。

Model.getVarLowerIIS()

获取变量下界的 IIS 状态。

概要

```
int[] getVarLowerIIS(VarArray vars)
```

参量

`vars`: 指定变量构成的 VarArray 类对象。

返回值

变量下界的 IIS 状态。

Model.getVarLowerIIS()

获取变量下界的 IIS 状态。

概要

```
int[] getVarLowerIIS(Var[] vars)
```

参量

`vars`: 指定变量构成的 VarArray 类对象。

返回值

变量下界的 IIS 状态。

Model.getVars()

获取模型中所有的变量。

概要

```
VarArray getVars()
```

返回值

变量构成的 VarArray 类。

Model.getVarUpperIIS()

获取变量上界的 IIS 状态。

概要

```
int[] getVarUpperIIS(VarArray vars)
```

参量

`vars`: 指定变量构成的 VarArray 类对象。

返回值

变量上界的 IIS 状态。

Model.getVarUpperIIS()

获取变量上界的 IIS 状态。

概要

```
int[] getVarUpperIIS(Var[] vars)
```

参量

`vars`: 指定变量构成的 VarArray 类对象。

返回值

变量上界的 IIS 状态。

Model.interrupt()

中断当前问题的求解。

概要

```
void interrupt()
```

Model.loadMipStart()

为问题里的变量加载最终初始值。

概要

```
void loadMipStart()
```

Model.loadTuneParam()

加载指定的调优参数到模型。

概要

```
void loadTuneParam(int idx)
```

参量

idx: 调优参数的下标。

Model.read()

从文件中读取问题，解决方案，基，MIP start 或者 COPT 参数。

概要

```
void read(String filename)
```

参量

filename: 输入的文件名。

Model.readBasis()

从文件中读取基。

概要

```
void readBasis(String filename)
```

参量

filename: 输入的文件名。

Model.readBin()

从文件中读取 COPT 二进制格式的问题。

概要

```
void readBin(String filename)
```

参量

filename: 输入的文件名

Model.readCbf()

从文件中读取 CBF 格式的问题。

概要

```
void readCbf(String filename)
```

参量

`filename`: 输入的文件名。

Model.readLp()

从文件中读取 LP 格式的问题。

概要

```
void readLp(String filename)
```

参量

`filename`: 输入的文件名。

Model.readMps()

从文件中读取 MPS 格式的问题。

概要

```
void readMps(String filename)
```

参量

`filename`: 输入的文件名

Model.readMst()

从文件中读取 MIP start 信息。

概要

```
void readMst(String filename)
```

参量

`filename`: 输入的文件名。

Model.readParam()

从文件中读取 COPT 参数。

概要

```
void readParam(String filename)
```

参量

`filename`: 输入的文件名。

Model.readSdpa()

从文件中读取 SDPA 格式的问题。

概要

```
void readSdpa(String filename)
```

参量

`filename`: 输入的文件名。

Model.readSol()

从文件中读取解决方案。

概要

```
void readSol(String filename)
```

参量

`filename`: 输入的文件名。

Model.readTune()

从文件中读取调优参数。

概要

```
void readTune(String filename)
```

参量

`filename`: 输入的文件名。

Model.remove()

从模型中删除一系列变量。

概要

```
void remove(Var[] vars)
```

参量

`vars`: 变量数组。

Model.remove()

从模型中删除一系列变量。

概要

```
void remove(VarArray vars)
```

参量

`vars`: 变量构成的 VarArray 对象。

Model.remove()

从模型中删除一系列约束。

概要

```
void remove(Constraint[] constrs)
```

参量

`constrs`: 约束数组。

Model.remove()

从模型中删除一系列约束。

概要

```
void remove(ConstrArray constrs)
```

参量

`constrs`: 约束构成的 ConstrArray 对象。

Model.remove()

从模型中删除一系列 SOS 约束。

概要

```
void remove(Sos[] soss)
```

参量

`soss`: SOS 约束数组。

Model.remove()

从模型中删除一系列 SOS 约束。

概要

```
void remove(SosArray soss)
```

参量

`soss`: SOS 约束构成的 SosArray 对象。

Model.remove()

从模型中删除一系列二阶锥约束。

概要

```
void remove(Cone[] cones)
```

参量

`cones`: 二阶锥约束数组。

Model.remove()

从模型中删除一系列二阶锥约束。

概要

```
void remove(ConeArray cones)
```

参量

`cones`: 二阶锥约束构成的 ConeArray 对象。

Model.remove()

从模型中删除一系列一般约束。

概要

```
void remove(GenConstr[] genConstrs)
```

参量

genConstrs: 一般约束数组。

Model.remove()

从模型中删除一系列一般约束。

概要

```
void remove(GenConstrArray genConstrs)
```

参量

genConstrs: 一般约束构成的 GenConstrArray 对象。

Model.remove()

从模型中删除一些二次约束。

概要

```
void remove(QConstraint[] qconstrs)
```

参量

qconstrs: 二次约束数组。

Model.remove()

从模型中删除一些二次约束。

概要

```
void remove(QConstrArray qconstrs)
```

参量

qconstrs: 二次约束构成的 QConstrArray 对象。

Model.remove()

从模型中删除一批半定变量。

概要

```
void remove(PsdVar[] vars)
```

参量

`vars`: 半定变量数组。

Model.remove()

从模型中删除一批半定变量。

概要

```
void remove(PsdVarArray vars)
```

参量

`vars`: 半定变量构成的 PsdVarArray 对象。

Model.remove()

从模型中删除一批半定约束。

概要

```
void remove(PsdConstraint[] constrs)
```

参量

`constrs`: 半定约束数组。

Model.remove()

从模型中删除一批半定约束。

概要

```
void remove(PsdConstrArray constrs)
```

参量

`constrs`: 半定约束构成的 PsdConstrArray 对象。

Model.remove()

从模型中删除一批 LMI 约束。

概要

```
void remove(LmiConstrArray constrs)
```

参量

`constrs`: LMI 约束构成的 LmiConstrArray 对象。

Model.remove()

从模型中删除一批 LMI 约束。

概要

```
void remove(LmiConstraint[] constrs)
```

参量

`constrs`: LMI 约束数组。

Model.reset()

只重新设置解。

概要

```
void reset()
```

Model.resetAll()

重新设置解与其他信息。

概要

```
void resetAll()
```

Model.resetParam()

重新设置参数为默认值。

概要

```
void resetParam()
```

Model.set()

设置与指定变量相关的双精度型信息的值。

概要

```
void set(
    String name,
    Var[] vars,
    double[] vals)
```

参量

name: 双精度型信息的名称。

vars: 指定变量数组。

vals: 信息的值。

Model.set()

设置与指定变量相关的双精度型信息的值。

概要

```
void set(
    String name,
    VarArray vars,
    double[] vals)
```

参量

name: 双精度型信息的名称。

vars: 指定变量构成的 VarArray 类。

vals: 信息的值。

Model.set()

设置与指定约束相关的双精度型信息的值。

概要

```
void set(
    String name,
    Constraint[] constrs,
    double[] vals)
```

参量

name: 双精度型信息的名称。

constrs: 指定约束数组。

vals: 信息的值。

Model.set()

设置与指定约束相关的双精度型信息的值。

概要

```
void set(
    String name,
    ConstrArray constrs,
    double[] vals)
```

参量

name: 双精度型信息的名称。

constrs: 指定约束构成的 ConstrArray 类。

vals: 信息的值。

Model.set()

设置与指定半定约束相关的双精度型信息的值。

概要

```
void set(
    String name,
    PsdConstraint[] constrs,
    double[] vals)
```

参量

name: 双精度型信息的名称。

constrs: 指定半定约束构成的 PsdConstrArray 类。

vals: 双精度型数组，保存了信息的值。

Model.set()

设置与指定半定约束相关的双精度型信息的值。

概要

```
void set(
    String name,
    PsdConstrArray constrs,
    double[] vals)
```

参量

`name`: 双精度型信息的名称。

`constrs`: 指定半定约束构成的 PsdConstrArray 类。

`vals`: 双精度型数组，保存了信息的值。

Model.setBasis()

设置行和列的基状态。

概要

```
void setBasis(int[] colbasis, int[] rowbasis)
```

参量

`colbasis`: 列的基状态。

`rowbasis`: 行的基状态。

Model.setCallback()

在 COPT 模型中设置用户自定义回调。

概要

```
void setCallback(CallbackBase cb, int cbctx)
```

参量

`cb`: 用户自定义回调对象，继承 CallbackBase 类。

`cbctx`: COPT 回调上下文。

Model.setCoeff()

设置变量的系数。

概要

```
void setCoeff(
    Constraint constr,
    Var var,
    double newVal)
```

参量

`constr`: 指定的约束。

`var`: 指定的变量。

`newVal`: 新系数。

Model.setDblParam()

设置 COPT 双精度型参数的值。

概要

```
void setDblParam(String param, double val)
```

参量

`param`: 双精度型参数的名称。

`val`: 双精度型参数的值。

Model.setIntParam()

设置 COPT 整型参数的值。

概要

```
void setIntParam(String param, int val)
```

参量

`param`: 整型参数的名称。

`val`: 整型参数的值。

Model.setLmiCoeff()

设置 LMI 约束中指定变量的系数矩阵。

概要

```
void setLmiCoeff(
    LmiConstraint constr,
    Var var,
    SymMatrix mat)
```

参量

`constr`: 给定的 LMI 约束。

`var`: 指定的变量。

`mat`: 新系数矩阵。

Model.setLmiRhs()

设置 LMI 约束中指定变量的常数项矩阵。

概要

```
void setLmiRhs(LmiConstraint constr, SymMatrix mat)
```

参量

`constr`: 给定的 LMI 约束。

`mat`: 新常数项矩阵。

Model.setLpSolution()

设置 LP 解。

概要

```
void setLpSolution(
    double[] value,
    double[] slack,
    double[] rowDual,
    double[] redCost)
```

参量

`value`: 变量的解。

`slack`: 约束的取值。

`rowDual`: 约束的对偶值。

redCost: 变量的对偶值。

Model.setMipStart()

设置给定数目变量的初始值，从第一个开始。

概要

```
void setMipStart(int count, double[] vals)
```

参量

count: 设置变量的数量。

vals: 变量的值。

Model.setMipStart()

设置指定变量的初始值。

概要

```
void setMipStart(Var var, double val)
```

参量

var: 指定变量。

val: 变量的初始值。

Model.setMipStart()

设置一系列变量的初始值。

概要

```
void setMipStart(Var[] vars, double[] vals)
```

参量

vars: 指定变量数组。

vals: 变量的初始值。

Model.setMipStart()

设置一系列变量的初始值。

概要

```
void setMipStart(VarArray vars, double[] vals)
```

参量

vars: 指定变量构成的 VarArray 类。

vals: 变量的初始值。

Model.setObjConst()

设置目标函数里的的常数。

概要

```
void setObjConst(double constant)
```

参量

constant: 常数的值。

Model.setObjective()

设置模型的目标函数。

概要

```
void setObjective(Expr expr, int sense)
```

参量

expr: 目标函数的表达式。

sense: 对最小化问题设置 COPT_MINIMIZE；对最大化问题设置 COPT_MAXIMIZE。

Model.setObjSense()

设置目标函数的类型。

概要

```
void setObjSense(int sense)
```

参量

sense: 目标函数的类型。

Model.setPsdCoeff()

设置半定约束中指定半定变量的系数矩阵。

概要

```
void setPsdCoeff(
    PsdConstraint constr,
    PsdVar var,
    SymMatrix mat)
```

参量

`constr`: 给定的半定约束。

`var`: 指定的半定变量。

`mat`: 新系数矩阵。

Model.setPsdObjective()

设置模型的半定目标。

概要

```
void setPsdObjective(PsdExpr expr, int sense)
```

参量

`expr`: 模型目标函数的半定表达式。

`sense`: 模型当前的优化方向。

Model.setQuadObjective()

设置模型的二次目标。

概要

```
void setQuadObjective(QuadExpr expr, int sense)
```

参量

`expr`: 模型目标函数的二次表达式。

`sense`: 默认值 0 表示不改变模型当前的优化方向。

Model.setSlackBasis()

设置松弛状态。

概要

```
void setSlackBasis()
```

Model.setSolverLogFile()

设置 COPT 的日志文件。

概要

```
void setSolverLogFile(String filename)
```

参量

`filename`: 日志文件名。

Model.solve()

求解当前的 MIP 问题。

概要

```
void solve()
```

Model.solveLp()

求解当前的 LP 问题。

概要

```
void solveLp()
```

Model.tune()

模型调优。

概要

```
void tune()
```

Model.write()

向文件中输出问题，解决方案，基，MIP start 或者改变后的 COPT 参数。

概要

```
void write(String filename)
```

参量

`filename`: 输出的文件名。

Model.writeBasis()

将最优基输出到“.bas”文件。

概要

```
void writeBasis(String filename)
```

参量

`filename`: 输出的文件名

Model.writeBin()

将问题以 COPT 二进制格式输出到文件中。

概要

```
void writeBin(String filename)
```

参量

`filename`: 输出的文件名。

Model.writeIIS()

将 IIS 输出到文件中。

概要

```
void writeIIS(String filename)
```

参量

`filename`: 输出文件名。

Model.writeLp()

将问题以 LP 格式输出到文件中。

概要

```
void writeLp(String filename)
```

参量

`filename`: 输出的文件名。

Model.writeMps()

将问题以 MPS 格式输出到文件中。

概要

```
void writeMps(String filename)
```

参量

`filename`: 输出的文件名。

Model.writeMpsStr()

将问题以 MPS 格式输出到问题缓存中。

概要

```
ProbBuffer writeMpsStr()
```

返回值

输出的 MPS 问题缓存对象。

Model.writeMst()

将 MIP start 信息输出到“.mst”文件。

概要

```
void writeMst(String filename)
```

参量

`filename`: 输出的文件名。

Model.writeParam()

将更改后的 COPT 参数输出到“.par”文件。

概要

```
void writeParam(String filename)
```

参量

`filename`: 输出的文件名。

Model.writePoolSol()

将指定的解池中的结果输出到“.sol”文件。

概要

```
void writePoolSol(int idx, String filename)
```

参量

`idx`: 解池中解的索引。

`filename`: 输出的文件名。

Model.writeRelax()

将可行化松弛模型输出到文件中。

概要

```
void writeRelax(String filename)
```

参量

`filename`: 输出文件名。

Model.writeSol()

将解决方案输出到“.sol”文件。

概要

```
void writeSol(String filename)
```

参量

`filename`: 输出的文件名。

Model.writeTuneParam()

将指定的调优参数输出到“.par”文件。

概要

```
void writeTuneParam(int idx, String filename)
```

参量

`idx`: 调优参数下标。

`filename`: 输出的文件名。

26.2.4 Var 类

Var 类是杉数优化求解器变量的相关操作的封装，提供了以下成员方法：

Var.get()

获取变量的属性值。支持”Value”, ”RedCost”, ”LB”, ”UB”, and ”Obj” 属性。

概要

```
double get(String attr)
```

参量

`attr`: 属性的名称。

返回值

属性的值。

Var.getBasis()

获取变量的基状态。

概要

```
int getBasis()
```

返回值

变量的基状态。

Var.getIdx()

获取变量的索引值。

概要

```
int getIdx()
```

返回值

索引值。

Var.getLowerIIS()

获取变量下界的 IIS 状态。

概要

```
int getLowerIIS()
```

返回值

变量下界的 IIS 状态。

Var.getName()

获取变量的名称。

概要

```
String getName()
```

返回值

变量名称。

Var.getType()

获取变量的类型。

概要

```
char getType()
```

返回值

变量类型。

Var.getUpperIIS()

获取变量上界的 IIS 状态。

概要

```
int getUpperIIS()
```

返回值

变量上界的 IIS 状态。

Var.remove()

从模型中删除变量。

概要

```
void remove()
```

Var.set()

设置变量的属性值。支持”LB”, ”UB” 和”Obj” 属性。

概要

```
void set(String attr, double val)
```

参量

`attr`: 属性的名称。

`val`: 新的值。

Var.setName()

设置变量的名称。

概要

```
void setName(String name)
```

参量

`name`: 变量名称。

Var.setType()

设置变量的类型。

概要

```
void setType(char vtype)
```

参量

`vtype`: 变量类型。

26.2.5 VarArray 类

为方便用户对一组 Java *Var 类* 对象进行操作，杉数优化求解器的 Java 接口设计了 VarArray 类，提供了以下成员方法：

VarArray.VarArray()

VarArray 的构造函数。

概要

```
VarArray()
```

VarArray.getVar()

获取 VarArray 类中指定索引值的变量。

概要

```
Var getVar(int idx)
```

参量

`idx`: 指定索引值。

返回值

指定索引值的变量。

VarArray.pushBack()

向 VarArray 类中添加一个变量。

概要

```
void pushBack(Var var)
```

参量

`var`: 变量。

VarArray.size()

获取 VarArray 类中变量的数目。

概要

```
int size()
```

返回值

VarArray 类中变量的数目。

26.2.6 Expr 类

Expr 类是杉数求解器中用于构建线性表达式时变量的相关组合操作，提供了以下成员方法：

Expr.Expr()

Expr 的构造函数。

概要

```
Expr()
```

Expr.Expr()

Expr 的构造函数。

概要

```
Expr(double constant)
```

参量

`constant`: Expr 中的常值。

Expr.Expr()

只有一项的 Expr 的构造函数。

概要

```
Expr(Var var)
```

参量

`var`: 添加的这一项对应的变量。

Expr.Expr()

只有一项的 Expr 的构造函数。

概要

```
Expr(Var var, double coeff)
```

参量

`var`: 添加的这一项对应的变量。

`coeff`: 添加的这一项对应的参数。

Expr.addConstant()

增加表达式中的常数。

概要

```
void addConstant(double constant)
```

参量

`constant`: 表达式中的常数改变量。

Expr.addExpr()

添加一个线性表达式的项。

概要

```
void addExpr(Expr expr)
```

参量

`expr`: 需要添加的线性表达式。

Expr.addExpr()

添加一个线性表达式的项，并乘以倍数。

概要

```
void addExpr(Expr expr, double mult)
```

参量

`expr`: 需要添加的线性表达式。

`mult`: 倍数参数。

Expr.addTerm()

向表达式中添加一项。

概要

```
void addTerm(Var var, double coeff)
```

参量

`var`: 新项中的变量。

`coeff`: 新项中的系数。

Expr.addTerms()

向表达式中添加项。

概要

```
void addTerms(Var[] vars, double coeff)
```

参量

`vars`: 新项中的变量数组。

`coeff`: 新项中的公共系数。

Expr.addTerms()

向表达式中添加项。

概要

```
void addTerms(Var[] vars, double[] coeffs)
```

参量

`vars`: 新项中的变量数组。

`coeffs`: 新项中的系数数组。

Expr.addTerms()

向表达式中添加项。

概要

```
void addTerms(VarArray vars, double coeff)
```

参量

`vars`: 新项中的变量构成的 VarArray 类。

`coeff`: 新项中的公共系数。

Expr.addTerms()

向表达式中添加项。

概要

```
void addTerms(VarArray vars, double[] coeffs)
```

参量

`vars`: 新项中的变量构成的 VarArray 类。

`coeffs`: 新项中的系数数组。

Expr.clone()

深度拷贝表达式对象。

概要

```
Expr clone()
```

返回值

复制的表达式对象。

Expr.evaluate()

求解后对线性表达式估值。

概要

```
double evaluate()
```

返回值

表达式估值。

Expr.getCoeff()

获取表达式指定索引值项数中的系数。

概要

```
double getCoeff(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值项数中的系数。

Expr.getConstant()

获取表达式中的常数。

概要

```
double getConstant()
```

返回值

表达式中的常数。

Expr.getVar()

获取表达式指定索引值项数中的变量。

概要

```
Var getVar(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值项数中的变量。

Expr.remove()

删除表达式中指定索引值的项。

概要

```
void remove(int idx)
```

参量

`idx`: 指定索引值。

Expr.remove()

删除表达式中与指定变量相关的项。

概要

```
void remove(Var var)
```

参量

`var`: 指定变量。

Expr.setCoeff()

设置表达式指定索引值项数中的系数。

概要

```
void setCoeff(int i, double val)
```

参量

`i`: 指定索引值。

`val`: 指定索引值项数中的参数。

Expr.setConstant()

设置表达式中的常数。

概要

```
void setConstant(double constant)
```

参量

`constant`: 表达式中的常数。

Expr.size()

获取表达式中的项数。

概要

```
long size()
```

返回值

表达式中的项数。

26.2.7 Constraint 类

Constraint 类是杉数求解器线性约束的相关操作的封装，提供了以下成员方法：

Constraint.get()

获得 Constraint 的属性值。支持"Dual", "Slack", "LB", "UB" 属性。

概要

```
double get(String attr)
```

参量

`attr`: 所需要获得的属性名称。

返回值

属性值。

Constraint.getBasis()

获得 Constraint 的基状态。

概要

```
int getBasis()
```

返回值

Constraint 的基状态。

Constraint.getIdx()

获取 Constraint 的索引值。

概要

```
int getIdx()
```

返回值

Constraint 的索引值。

Constraint.getLowerIIS()

获取约束下界的 IIS 状态。

概要

```
int getLowerIIS()
```

返回值

约束下界的 IIS 状态。

Constraint.getName()

获取 Constraint 的名称。

概要

```
String getName()
```

返回值

Constraint 的名称。

Constraint.getUpperIIS()

获取约束上界的 IIS 状态。

概要

```
int getUpperIIS()
```

返回值

约束上界的 IIS 状态。

Constraint.remove()

从模型中删除当前的 Constraint。

概要

```
void remove()
```

Constraint.set()

设置 Constraint 的属性值。支持"LB", "UB" 属性。

概要

```
void set(String attr, double val)
```

参量

`attr`: 所需要设置的属性名称。

`val`: 所需要设置的属性值。

Constraint.setName()

设置 Constraint 的名称。

概要

```
void setName(String name)
```

参量

`name`: Constraint 的名称。

26.2.8 ConstraArray 类

为方便用户对一组 Java *Constraint 类* 对象进行操作，杉数求解器的 Java 接口设计了 ConstrArray 类，提供了以下成员方法：

ConstrArray.ConstrArray()

ConstrArray 的构造函数。

概要

```
ConstrArray()
```

ConstrArray.getConstr()

获取 ConstrArray 中的指定索引值的 Constraint。

概要

```
Constraint getConstr(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 Constraint。

ConstrArray.pushBack()

向 ConstrArray 中添加一个 Constraint。

概要

```
void pushBack(Constraint constr)
```

参量

`constr`: 待添加的 Constraint。

ConstrArray.size()

获取 ConstrArray 中的元素个数。

概要

```
int size()
```

返回值

ConstrArray 中的元素个数。

26.2.9 ConstraBuilder 类

ConstrBuilder 类是杉数优化求解器中构建线性约束时的构建器的封装，提供了以下成员方法：

ConstrBuilder.ConstrBuilder()

ConstrBuilder 的构造函数。

概要

```
ConstrBuilder()
```

ConstrBuilder.getExpr()

获取线性约束生成器对象的表达式。

概要

```
Expr getExpr()
```

返回值

Expression 对象。

ConstrBuilder.getRange()

获取线性约束生成器对象的约束范围的长度（从下界到上界的长度，必须大于 0）。

概要

```
double getRange()
```

返回值

约束范围的长度（从下界到上界的长度）。

ConstrBuilder.getSense()

获取线性约束生成器对象的约束类型。

概要

```
char getSense()
```

返回值

约束类型。

ConstrBuilder.set()

设置一个约束构造类的内容。

概要

```
void set(
    Expr expr,
    char sense,
    double rhs)
```

参量

`expr`: 约束一侧的表达式。

`sense`: 除了 COPT_RANGE 外的约束类型。

`rhs`: 约束另一侧的常数项

ConstrBuilder.setRange()

设置一个范围约束（带有上下界）。

概要

```
void setRange(Expr expr, double range)
```

参量

`expr`: 约束表达式。其表达式的常数项的负数其实是这个约束的上界。

`range`: 约束范围的长度（从下界到上界的长度，必须大于 0）。

26.2.10 ConstrBuilderArray 类

为方便用户对一组 Java *ConstraBuilder 类* 对象进行操作，杉数优化求解器的 Java 接口设计了 ConstrBuilderArray 类，提供了以下成员方法：

ConstrBuilderArray.ConstrBuilderArray()

ConstrBuilderArray 的构造函数。

概要

```
ConstrBuilderArray()
```

ConstrBuilderArray.getBuilder()

获取 ConstrBuilderArray 中的指定索引值的 Constraint。

概要

```
ConstrBuilder getBuilder(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 ConstrBuilder。

ConstrBuilderArray.pushBack()

向 ConstrBuilderArray 中添加一个 ConstrBuilder。

概要

```
void pushBack(ConstrBuilder builder)
```

参量

`builder`: 待添加的 ConstrBuilder。

ConstrBuilderArray.size()

获取 ConstrBuilderArray 中的元素个数。

概要

```
int size()
```

返回值

ConstrBuilderArray 中的元素个数。

26.2.11 Column 类

为了方便用户采用按列建模的方式，杉数优化求解器的 Java 接口设计了 Column 类，提供了以下成员方法：

Column.Column()

Column 的构造函数。

概要

```
Column()
```

Column.addColumn()

添加一个列的项，并乘以倍数。

概要

```
void addColumn(Column col)
```

参量

`col`: 需要添加的列对象。

Column.addColumn()

添加一个列的项，并乘以倍数。

概要

```
void addColumn(Column col, double mult)
```

参量

`col`: 需要添加的列对象。

`mult`: 系数倍数。

Column.addTerm()

添加一个新的项。

概要

```
void addTerm(Constraint constr, double coeff)
```

参量

`constr`: 待添加项的线性约束。

`coeff`: 待添加项的系数。

Column.addTerms()

添加一个或多个新的项。

概要

```
void addTerms(Constraint[] constrs, double coeff)
```

参量

`constrs`: 待添加项的线性约束数组。

`coeff`: 待添加项的系数。

Column.addTerms()

添加一个或多个新的项。

概要

```
void addTerms(Constraint[] constrs, double[] coeffs)
```

参量

`constrs`: 待添加项的线性约束数组。

`coeffs`: 待添加项的系数数组。

Column.addTerms()

添加一个或多个新的项。

概要

```
void addTerms(ConstrArray constrs, double coeff)
```

参量

`constrs`: 待添加项的线性约束构成的 ConstrArray 类。

`coeff`: 待添加项的系数。

Column.addTerms()

添加一个或多个新的项。

概要

```
void addTerms(ConstrArray constrs, double[] coeffs)
```

参量

`constrs`: 待添加项的线性约束构成的 ConstrArray 类。

`coeffs`: 待添加项的系数数组。

Column.clear()

清空 Column 的内容。

概要

```
void clear()
```

Column.clone()

创建 Column 的深度拷贝。

概要

```
Column clone()
```

返回值

Column 的深度拷贝对象。

Column.getCoeff()

获得 Column 中第 i 项的系数。

概要

```
double getCoeff(int i)
```

参量

`i`: 第 i 项的索引值。

返回值

Column 中第 i 项的系数。

Column.getConstr()

获得 Column 中第 i 项的线性约束。

概要

```
Constraint getConstr(int i)
```

参量

`i`: 第 i 项的索引值。

返回值

Column 中第 i 项的线性约束。

Column.remove()

从 Column 中移除指定的项。

概要

```
void remove(int idx)
```

参量

`idx`: 待移除项的索引值。

Column.remove()

从 Column 中移除指定线性约束所在的项。

概要

```
void remove(Constraint constr)
```

参量

`constr`: 指定线性约束。

Column.size()

获取 Column 中元素的个数。

概要

```
int size()
```

返回值

Column 中元素的个数。

26.2.12 ColumnArray 类

为方便用户对一组 Java *Column 类* 对象进行操作，杉数优化求解器的 Java 接口设计了 ColumnArray 类，提供了以下成员方法：

ColumnArray.ColumnArray()

ColumnArray 的构造函数。

概要

```
ColumnArray()
```

ColumnArray.clear()

清空所有的 Column。

概要

```
void clear()
```

ColumnArray.getColumn()

获取 ColumnArray 中的指定索引值的 Column。

概要

```
Column getColumn(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 Column。

ColumnArray.pushBack()

向 ColumnArray 中添加一个 Column。

概要

```
void pushBack(Column col)
```

参量

`col`: 待添加的 Column。

ColumnArray.size()

获取 ColumnArray 中的元素个数。

概要

```
int size()
```

返回值

ColumnArray 中的元素个数。

26.2.13 Sos 类

SOS 类是杉数求解器的 SOS 约束的相关操作的封装，目前提供了以下成员方法：

关于 SOS 约束的介绍请参考特殊约束：*SOS* 约束章节 。

Sos.getIdx()

获取 SOS 约束的索引值。

概要

```
int getIdx()
```

返回值

SOS 约束的索引值。

Sos.getIIS()

获取 SOS 约束的 IIS 状态。

概要

```
int getIIS()
```

返回值

IIS 状态。

Sos.remove()

从模型中删除 SOS 约束。

概要

```
void remove()
```

26.2.14 SosArray 类

为方便用户对一组 Java *Sos* 类 对象进行操作，杉数求解器的 Java 接口设计了 SosArray 类，提供了以下成员方法：

SosArray.SosArray()

SosArray 的构造函数。

概要

```
SosArray()
```

SosArray.getSos()

获取 SosArray 中的指定索引值的 Sos 约束。

概要

```
Sos getSos(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 Sos 约束。

SosArray.pushBack()

向 SosArray 里添加 SOS 约束。

概要

```
void pushBack(Sos sos)
```

参量

`sos`: SOS 约束。

SosArray.size()

获取 SosArray 里 SOS 约束个数。

概要

```
int size()
```

返回值

SOS 约束个数。

26.2.15 SosBuilder 类

SOSBuilder 类是杉数优化求解器中构建 SOS 约束的构建器的封装，提供了以下成员方法：

关于 SOS 约束的介绍请参考特殊约束：*SOS* 约束章节 。

SosBuilder.SosBuilder()

SosBuilder 的构造函数。

概要

```
SosBuilder()
```

SosBuilder.getSize()

获取 SOS 约束中元素个数。

概要

```
int getSize()
```

返回值

元素个数。

SosBuilder.getType()

获取 SOS 约束类型。

概要

```
int getType()
```

返回值

SOS 约束类型。

SosBuilder.getVar()

从 SOS 约束中指定索引的元素中获取变量。

概要

```
Var getVar(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定索引元素对应的变量。

SosBuilder.GetVars()

获取 SOS 约束中的全部变量。

概要

```
VarArray GetVars()
```

返回值

一组变量。

SosBuilder.getWeight()

从 SOS 约束中指定索引的元素中获取权重。

概要

```
double getWeight(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定索引元素中对应的权重。

SosBuilder.getWeights()

获取 SOS 约束中所有元素对应的权重。

概要

```
double[] getWeights()
```

返回值

权重数组。

SosBuilder.set()

设置 SOS 约束的变量和权重。

概要

```
void set(
    VarArray vars,
    double[] weights,
    int type)
```

参量

`vars`: 变量构成的 VarArray 类。

`weights`: 权重数组。

`type`: SOS 约束的类型。

26.2.16 SosBuilderArray 类

为方便用户对一组 Java *SosBuilder 类* 对象进行操作，杉数优化求解器的 Java 接口设计了 SosBuilderArray 类，提供了以下成员方法：

SosBuilderArray.SosBuilderArray()

SosBuilderArray 的构造函数。

概要

```
SosBuilderArray()
```

SosBuilderArray.getBuilder()

获取指定索引值的 SOS 约束生成器（SosBuilder）。

概要

```
SosBuilder getBuilder(int idx)
```

参量

`idx`: 指定索引值。

返回值

指定索引值的 SOS 约束生成器（SosBuilder）。

SosBuilderArray.pushBack()

向 SosBuilderArray 类中添加 SOS 约束生成器（SosBuilder）。

概要

```
void pushBack(SosBuilder builder)
```

参量

`builder`: SOS 约束生成器（SosBuilder）。

SosBuilderArray.size()

获取 SosBuilderArray 类中元素个数。

概要

```
int size()
```

返回值

SosBuilderArray 类中元素个数。

26.2.17 GenConstr 类

GenConstr 类是杉数优化求解器的 Indicator 约束的相关操作的封装，提供了以下成员方法：

关于 Indicator 约束的介绍请参考特殊约束：*Indicator 约束章节* 。

GenConstr.getIdx()

获取 GenConstr 的索引值。

概要

```
int getIdx()
```

返回值

GenConstr 的索引值。

GenConstr.getIIS()

获取一般约束的 IIS 状态。

概要

```
int getIIS()
```

返回值

IIS 状态。

GenConstr.remove()

从模型中删除 GenConstr。

概要

```
void remove()
```

26.2.18 GenConstrArray 类

为方便用户对一组 Java *GenConstr 类* 对象进行操作，杉数优化求解器的 Java 接口设计了 GenConstrArray 类，提供了以下成员方法：

GenConstrArray.GenConstrArray()

GenConstrArray 的构造函数。

概要

```
GenConstrArray()
```

GenConstrArray.getGenConstr()

获取 GenConstrArray 中的指定索引值的 GenConstr。

概要

```
GenConstr getGenConstr(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 GenConstr。

GenConstrArray.pushBack()

向 GenConstrArray 中添加一个 GenConstr。

概要

```
void pushBack(GenConstr genconstr)
```

参量

`genconstr`: 待添加的 GenConstr。

GenConstrArray.size()

获取 GenConstrArray 中的元素个数。

概要

```
int size()
```

返回值

GenConstrArray 中的元素个数。

26.2.19 GenConstrBuilder 类

GenConstrBuilder 类是杉数优化求解器中构建 Indicator 约束时的构建器的封装，提供了以下成员方法：

关于 Indicator 约束的介绍请参考特殊约束：*Indicator* 约束章节 。

GenConstrBuilder.GenConstrBuilder()

GenConstrBuilder 的构造函数。

概要

```
GenConstrBuilder()
```

GenConstrBuilder.getBinVal()

获取与 GenConstr 的相关联的二进制值。

概要

```
int getBinVal()
```

返回值

二进制值。

GenConstrBuilder.getBinVar()

获取与 GenConstr 的相关联的二进制变量。

概要

```
Var getBinVar()
```

返回值

二进制变量。

GenConstrBuilder.getExpr()

获取与 GenConstr 的相关联的表达式。

概要

```
Expr getExpr()
```

返回值

表达式对象。

GenConstrBuilder.getSense()

获取与 GenConstr 的相关联的约束类型。

概要

```
char getSense()
```

返回值

约束类型。

GenConstrBuilder.set()

设置 GenConstr 二进制变量，二进制变量取值，表达式，约束类型。

概要

```
void set(
    Var binvar,
    int binval,
    Expr expr,
    char sense)
```

参量

`binvar`: 二进制变量。

`binval`: 二进制变量取值。

`expr`: 表达式。

`sense`: 约束类型。

26.2.20 GenConstrBuilderArray 类

为方便用户对一组 Java *GenConstrBuilder 类* 对象进行操作，杉数优化求解器的 Java 接口设计了 GenConstrBuilderArray 类，提供了以下成员方法：

GenConstrBuilderArray.GenConstrBuilderArray()

GenConstrBuilderArray 的构造函数。

概要

```
GenConstrBuilderArray()
```

GenConstrBuilderArray.getBuilder()

获取 GenConstrBuilderArray 中的指定索引值的 GenConstrBuilder。

概要

```
GenConstrBuilder getBuilder(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 GenConstrBuilder。

GenConstrBuilderArray.pushBack()

向 GenConstrBuilderArray 中添加一个 GenConstrBuilder。

概要

```
void pushBack(GenConstrBuilder builder)
```

参量

`builder`: 待添加的 GenConstrBuilder。

GenConstrBuilderArray.size()

获取 GenConstrBuilderArray 中的元素个数。

概要

```
int size()
```

返回值

GenConstrBuilderArray 中的元素个数。

26.2.21 Cone 类

Cone 类是杉数求解器的二阶锥约束的相关操作的封装，目前提供了以下成员方法：

Cone.getIdx()

获取锥约束的下标值。

概要

```
int getIdx()
```

返回值

锥约束的下标值。

Cone.remove()

从模型中删除锥约束。

概要

```
void remove()
```

26.2.22 ConeArray 类

为方便用户对一组 Java *Cone 类* 对象进行操作，杉数求解器的 Java 接口设计了 ConeArray 类，提供了以下成员方法：

ConeArray.ConeArray()

ConeArray 的构造函数。

概要

```
ConeArray()
```

ConeArray.getCone()

获取 ConeArray 中的指定索引值的 Cone。

概要

```
Cone getCone(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 Cone。

ConeArray.pushBack()

向 ConeArray 里添加锥约束。

概要

```
void pushBack(Cone cone)
```

参量

`cone`: 锥约束。

ConeArray.size()

获取 ConeArray 里锥约束个数。

概要

```
int size()
```

返回值

锥约束个数。

26.2.23 ConeBuilder 类

ConeBuilder 类是杉数优化求解器中构建二阶锥约束的构建器的封装，提供了以下成员方法：

ConeBuilder.ConeBuilder()

ConeBuilder 的构造函数。

概要

```
ConeBuilder()
```

ConeBuilder.getSize()

获取锥约束中变量个数。

概要

```
int getSize()
```

返回值

变量个数。

ConeBuilder.getType()

获取锥约束类型。

概要

```
int getType()
```

返回值

锥约束类型。

ConeBuilder.getVar()

从锥约束中指定下标的变量。

概要

```
Var getVar(int idx)
```

参量

`idx`: 指定的下标值。

返回值

指定下标对应的变量。

ConeBuilder.getVars()

获取锥约束中的全部变量。

概要

```
VarArray getVars()
```

返回值

一组变量。

ConeBuilder.set()

设置锥约束的变量和类型。

概要

```
void set(VarArray vars, int type)
```

参量

`vars`: 变量构成的 VarArray 类。

`type`: 锥约束的类型。

26.2.24 ConeBuilderArray 类

为方便用户对一组 Java *ConeBuilder 类* 对象进行操作，杉数优化求解器的 Java 接口设计了 ConeBuilderArray 类，提供了以下成员方法：

ConeBuilderArray.ConeBuilderArray()

ConeBuilderArray 的构造函数。

概要

```
ConeBuilderArray()
```

ConeBuilderArray.getBuilder()

获取指定索引值的锥约束生成器（ConeBuilder）。

概要

```
ConeBuilder getBuilder(int idx)
```

参量

`idx`: 指定索引值。

返回值

指定索引值的锥约束生成器（ConeBuilder）。

ConeBuilderArray.pushBack()

向 ConeBuilderArray 类中添加锥约束生成器（ConeBuilder）。

概要

```
void pushBack(ConeBuilder builder)
```

参量

`builder`: 锥约束生成器（ConeBuilder）。

ConeBuilderArray.size()

获取 ConeBuilderArray 类中元素个数。

概要

```
int size()
```

返回值

ConeBuilderArray 类中元素个数。

26.2.25 QuadExpr 类

COPT 二次表达式包括一个线性表达式，一些二次项相关的变量和对应系数。QuadExpr 类是杉数求解器中用于构建二次表达式时对变量的相关组合操作，提供了以下成员方法：

QuadExpr.QuadExpr()

二次表达式的构造函数。

概要

```
QuadExpr()
```

QuadExpr.QuadExpr()

二次表达式的构造函数。

概要

```
QuadExpr(double constant)
```

参量

`constant`: 二次表达式中的常值。

QuadExpr.QuadExpr()

只有一线性项的二次表达式的构造函数。

概要

```
QuadExpr(Var var)
```

参量

`var`: 添加的一线性项对应的变量。

QuadExpr.QuadExpr()

只有一线性项的二次表达式的构造函数。

概要

```
QuadExpr(Var var, double coeff)
```

参量

`var`: 添加的一线性项对应的变量。

`coeff`: 添加的一线性项对应的参数。

QuadExpr.QuadExpr()

二次表达式的构造函数，初始值是给定的线性表达式。

概要

```
QuadExpr(Expr expr)
```

参量

`expr`: 二次表达式中的线性部分。

QuadExpr.QuadExpr()

使用两个线性表达式构造的二次表达式。

概要

```
QuadExpr(Expr expr, Var var)
```

参量

`expr`: 一个初始的线性表达式。

`var`: 另一个初始的变量。

QuadExpr.QuadExpr()

使用两个线性表达式构造的二次表达式。

概要

```
QuadExpr(Expr left, Expr right)
```

参量

`left`: 一个初始的线性表达式。

`right`: 另一个初始的线性表达式。

QuadExpr.addConstant()

增加二次表达式中的常数。

概要

```
void addConstant(double constant)
```

参量

`constant`: 二次表达式中的常数改变量。

QuadExpr.addLinExpr()

在二次表达式中添加一个线性表达式。

概要

```
void addLinExpr(Expr expr)
```

参量

`expr`: 需要添加的线性表达式。

QuadExpr.addLinExpr()

添加一个线性表达式的项，并乘以倍数。

概要

```
void addLinExpr(Expr expr, double mult)
```

参量

`expr`: 需要添加的线性表达式

`mult`: 倍数参数。

QuadExpr.addQuadExpr()

在二次表达式中添加一个二次表达式。

概要

```
void addQuadExpr(QuadExpr expr)
```

参量

`expr`: 需要添加的二次表达式。

QuadExpr.addQuadExpr()

添加一个二次表达式的项，并乘以倍数。

概要

```
void addQuadExpr(QuadExpr expr, double mult)
```

参量

`expr`: 需要添加的二次表达式

`mult`: 倍数参数。

QuadExpr.addTerm()

向二次表达式中添加一线性项。

概要

```
void addTerm(Var var, double coeff)
```

参量

var: 新线性项中的变量。

coeff: 新线性项中的系数。

QuadExpr.addTerm()

向二次表达式中添加一个二次项。

概要

```
void addTerm(
    Var var1,
    Var var2,
    double coeff)
```

参量

var1: 新二次项中的变量 1。

var2: 新二次项中的变量 2。

coeff: 新二次项中的系数。

QuadExpr.addTerms()

向二次表达式中添加一些线性项。

概要

```
void addTerms(Var[] vars, double coeff)
```

参量

vars: 新线性项中的变量数组。

coeff: 新线性项中的公共系数。

QuadExpr.addTerms()

向二次表达式中添加一些线性项。

概要

```
void addTerms(Var[] vars, double[] coeffs)
```

参量

vars: 新线性项中的变量数组。

coeffs: 新线性项中的系数数组。

QuadExpr.addTerms()

向二次表达式中添加线性项。

概要

```
void addTerms(VarArray vars, double coeff)
```

参量

vars: 新线性项中的变量构成的 VarArray 类。

coeff: 新线性项中的公共系数。

QuadExpr.addTerms()

向二次表达式中添加线性项。

概要

```
void addTerms(VarArray vars, double[] coeffs)
```

参量

vars: 新线性项中的变量构成的 VarArray 类。

coeffs: 新线性项中的系数数组。

QuadExpr.addTerms()

向二次表达式中添加一些二次项。

概要

```
void addTerms(
    VarArray vars1,
    VarArray vars2,
    double[] coeffs)
```

参量

vars1: 新二次项中的变量数组 1。

vars2: 新二次项中的变量数组 2。

coeffs: 新二次项中的系数数组。

QuadExpr.addTerms()

向二次表达式中添加一些二次项。

概要

```
void addTerms(
    Var[] vars1,
    Var[] vars2,
    double[] coeffs)
```

参量

vars1: 新二次项中的变量数组 1。

vars2: 新二次项中的变量数组 2。

coeffs: 新二次项中的系数数组。

QuadExpr.clone()

深度拷贝二次表达式对象。

概要

```
QuadExpr clone()
```

返回值

复制的二次表达式对象。

QuadExpr.evaluate()

求解后对二次表达式估值。

概要

```
double evaluate()
```

返回值

表达式估值。

QuadExpr.getCoeff()

获取二次表达式指定索引值对应项数中的系数。

概要

```
double getCoeff(int i)
```

参量

i: 指定索引值。

返回值

指定索引值对应项数中的系数。

QuadExpr.getConstant()

获取二次表达式中的常数。

概要

```
double getConstant()
```

返回值

二次表达式中的常数。

QuadExpr.getLinExpr()

获取二次表达式中的线性表达式。

概要

```
Expr getLinExpr()
```

返回值

线性表达式对象。

QuadExpr.getVar1()

获取二次表达式指定索引值对应项数中的第一个变量。

概要

```
Var getVar1(int i)
```

参量

i: 指定索引值。

返回值

指定索引值对应项数中的第一个变量。

QuadExpr.getVar2()

获取表达式指定索引值对应项数中的第二个变量。

概要

```
Var getVar2(int i)
```

参量

i: 指定索引值。

返回值

指定索引值对应项数中的第二个变量。

QuadExpr.remove()

删除二次表达式中指定索引值的项。

概要

```
void remove(int idx)
```

参量

idx: 指定索引值。

QuadExpr.remove()

删除二次表达式中与指定变量相关的项。

概要

```
void remove(Var var)
```

参量

var: 指定删除的变量。

QuadExpr.setCoeff()

设置二次表达式指定索引值项数中的系数。

概要

```
void setCoeff(int i, double val)
```

参量

i: 指定索引值。

val: 指定索引值项数中的参数。

QuadExpr.setConstant()

设置二次表达式中的常数。

概要

```
void setConstant(double constant)
```

参量

`constant`: 二次表达式中的常数。

QuadExpr.size()

获取二次表达式中的项数。

概要

```
long size()
```

返回值

二次表达式中的二次项数。

26.2.26 QConstraint 类

QConstraint 类是杉数求解器对二次约束的相关操作的封装，提供了以下成员方法：

QConstraint.get()

获得二次约束的属性值。

概要

```
double get(String attr)
```

参量

`attr`: 所需要获得的属性名称。

返回值

属性值。

QConstraint.getIdx()

获取二次约束的索引值。

概要

```
int getIdx()
```

返回值

二次约束的索引值。

QConstraint.getName()

获取二次约束的名称。

概要

```
String getName()
```

返回值

二次约束的名称。

QConstraint.getRhs()

获得二次约束的右端值。

概要

```
double getRhs()
```

返回值

二次约束的右端值。

QConstraint.getSense()

获得二次约束的右端值。

概要

```
char getSense()
```

返回值

二次约束的右端值。

QConstraint.remove()

从模型中删除当前的 QConstraint。

概要

```
void remove()
```

QConstraint.set()

设置 QConstraint 的属性值。

概要

```
void set(String attr, double val)
```

参量

attr: 所需要设置的属性名称。

val: 所需要设置的属性值。

QConstraint.setName()

设置二次约束的名称。

概要

```
void setName(String name)
```

参量

name: 二次约束的名称。

QConstraint.setRhs()

设置二次约束的右端值。

概要

```
void setRhs(double rhs)
```

参量

rhs: 二次约束的右端值。

QConstraint.setSense()

设置二次约束的类型。

概要

```
void setSense(char sense)
```

参量

sense: 二次约束的类型。

26.2.27 QConstrArray 类

为方便用户对一组 Java *QConstraint 类* 对象进行操作，杉数求解器的 Java 接口设计了 QConstrArray 类，提供了以下成员方法：

QConstrArray.QConstrArray()

QConstrArray 的构造函数。

概要

```
QConstrArray()
```

QConstrArray.getQConstr()

获取 QConstrArray 中的指定索引值的 QConstraint。

概要

```
QConstraint getQConstr(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 QConstraint。

QConstrArray.pushBack()

向 QConstrArray 中添加一个 QConstraint。

概要

```
void pushBack(QConstraint constr)
```

参量

`constr`: 待添加的 QConstraint。

QConstrArray.size()

获取 QConstrArray 中的元素个数。

概要

```
int size()
```

返回值

QConstrArray 中的元素个数。

26.2.28 QConstrBuilder 类

QConstrBuilder 类是杉数优化求解器中对构建二次约束的构建器的封装，提供了以下成员方法：

QConstrBuilder.QConstrBuilder()

QConstrBuilder 的构造函数。

概要

```
QConstrBuilder()
```

QConstrBuilder.getQuadExpr()

获取二次约束相关的表达式。

概要

```
QuadExpr getQuadExpr()
```

返回值

二次表达式对象。

QConstrBuilder.getSense()

获取二次约束相关的约束类型。

概要

```
char getSense()
```

返回值

约束类型。

QConstrBuilder.set()

设置一个二次约束的表达式，类型和边界值。

概要

```
void set(
    QuadExpr expr,
    char sense,
    double rhs)
```

参量

`expr`: 约束一侧的二次表达式。

`sense`: 二次约束类型。

`rhs`: 二次约束另一侧的常数项。

26.2.29 QConstrBuilderArray 类

为方便用户对一组 Java *QConstrBuilder 类* 对象进行操作，杉数优化求解器的 Java 接口设计了 QConstrBuilderArray 类，提供了以下成员方法：

QConstrBuilderArray.QConstrBuilderArray()

QConstrBuilderArray 的构造函数。

概要

```
QConstrBuilderArray()
```

QConstrBuilderArray.getBuilder()

获取 QConstrBuilderArray 中的指定索引值的 QConstraint。

概要

```
QConstrBuilder getBuilder(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的 QConstrBuilder。

QConstrBuilderArray.pushBack()

向 QConstrBuilderArray 中添加一个 QConstrBuilder。

概要

```
void pushBack(QConstrBuilder builder)
```

参量

`builder`: 待添加的 QConstrBuilder。

QConstrBuilderArray.size()

获取 QConstrBuilderArray 中的元素个数。

概要

```
int size()
```

返回值

QConstrBuilderArray 中的元素个数。

26.2.30 PsdVar 类

PsdVar 类是杉数优化求解器对半定变量的相关操作的封装，提供了以下成员方法：

PsdVar.get()

获取半定变量的属性值。

概要

```
double[] get(String attr)
```

参量

`attr`: 属性的名称。

返回值

输出双精度型数组，保存了属性值。

PsdVar.getDim()

获取半定变量的维度。

概要

```
int getDim()
```

返回值

半定变量的维度。

PsdVar.getIdx()

获取半定变量的索引。

概要

```
int getIdx()
```

返回值

半定变量的索引。

PsdVar.getLen()

获取半定变量展开后的长度。

概要

```
int getLen()
```

返回值

半定变量展开后的长度。

PsdVar.getName()

获取半定变量的名称。

概要

```
String getName()
```

返回值

半定变量名称。

PsdVar.remove()

从模型中删除半定变量。

概要

```
void remove()
```

26.2.31 PsdVarArray 类

为方便用户对一组*PsdVar 类* 对象进行操作，杉数优化求解器的 Java 接口设计了 PsdVarArray 类，提供了以下成员方法：

PsdVarArray.PsdVarArray()

PsdVarArray 的构造函数。

概要

```
PsdVarArray()
```

PsdVarArray.getPsdVar()

获取半定变量数组里指定索引的半定变量。

概要

```
PsdVar getPsdVar(int idx)
```

参量

idx: 半定变量的索引。

返回值

指定索引的半定变量。

PsdVarArray.pushBack()

向半定变量数组里添加半定变量。

概要

```
void pushBack(PsdVar var)
```

参量

var: 半定变量。

PsdVarArray.reserve()

预分配大小为 n 项的空间。

概要

```
void reserve(int n)
```

参量

n: 容纳 n 项的空间。

PsdVarArray.size()

获取半定变量数组里半定变量个数。

概要

```
int size()
```

返回值

半定变量个数。

26.2.32 PsdExpr 类

COPT 半定表达式包括一个线性表达式，一些半定变量和对应的系数矩阵。PsdExpr 类是杉数求解器中用于构建半定表达式时对半定变量的相关组合操作，提供了以下成员方法：

PsdExpr.PsdExpr()

半定表达式的构造函数。

概要

```
PsdExpr(double constant)
```

参量

constant: 半定表达式对象中的常量。

PsdExpr.PsdExpr()

使用变量和其系数构造的半定表达式。

概要

```
PsdExpr(Var var)
```

参量

var: 添加的这一项对应的变量。

PsdExpr.PsdExpr()

使用变量和其系数构造的半定表达式。

概要

```
PsdExpr(Var var, double coeff)
```

参量

var: 添加的这一项对应的变量。

coeff: 添加的这一项对应的参数，默认值为 1.0。

PsdExpr.PsdExpr()

使用线性表达式构造的半定表达式。

概要

```
PsdExpr(Expr expr)
```

参量

expr: 初始的线性表达式。

PsdExpr.PsdExpr()

使用半定变量和其系数矩阵构造的半定表达式。

概要

```
PsdExpr(PsdVar var, SymMatrix mat)
```

参量

var: 添加的这一项对应的半定变量。

mat: 添加的这一项对应的系数矩阵。

PsdExpr.PsdExpr()

使用半定变量和其系数矩阵表达式构造的半定表达式。

概要

```
PsdExpr(PsdVar var, SymMatExpr expr)
```

参量

var: 添加的这一项对应的半定变量。

expr: 新半定项中对称矩阵的表达式。

PsdExpr.addConstant()

增加半定表达式中的常数。

概要

```
void addConstant(double constant)
```

参量

constant: 半定表达式中的常数改变量。

PsdExpr.addLinExpr()

在半定表达式中添加一个线性表达式。

概要

```
void addLinExpr(Expr expr)
```

参量

expr: 需要添加的线性表达式。

PsdExpr.addLinExpr()

在半定表达式中添加一个线性表达式的项，并乘以倍数。

概要

```
void addLinExpr(Expr expr, double mult)
```

参量

expr: 需要添加的线性表达式

mult: 倍数。

PsdExpr.addPsdExpr()

添加一个半定表达式的项。

概要

```
void addPsdExpr(PsdExpr expr)
```

参量

`expr`: 需要添加的半定表达式。

PsdExpr.addPsdExpr()

添加一个半定表达式的项，并乘以倍数。

概要

```
void addPsdExpr(PsdExpr expr, double mult)
```

参量

`expr`: 需要添加的半定表达式。

`mult`: 倍数。

PsdExpr.addTerm()

向半定表达式中添加一线性项。

概要

```
void addTerm(Var var, double coeff)
```

参量

`var`: 新线性项中的变量。

`coeff`: 新线性项中的系数。

PsdExpr.addTerm()

向半定表达式中添加一个半定项。

概要

```
void addTerm(PsdVar var, SymMatrix mat)
```

参量

`var`: 新半定项中的半定变量。

`mat`: 新半定项中的系数矩阵。

PsdExpr.addTerm()

向半定表达式中添加一个半定项。

概要

```
void addTerm(PsdVar var, SymMatExpr expr)
```

参量

var: 新半定项中的半定变量。

expr: 新半定项中对称矩阵的表达式。

PsdExpr.addTerms()

向半定表达式中添加一些线性项。

概要

```
void addTerms(Var[] vars, double coeff)
```

参量

vars: 新线性项中的变量数组。

coeff: 新线性项中的公共系数。

PsdExpr.addTerms()

向表达式中添加一些线性项。

概要

```
void addTerms(Var[] vars, double[] coeffs)
```

参量

vars: 新线性项中的变量数组。

coeffs: 新线性项中的系数数组。

PsdExpr.addTerms()

向半定表达式中添加线性项。

概要

```
void addTerms(VarArray vars, double coeff)
```

参量

vars: 新线性项中的变量构成的 VarArray 类。

coeff: 新线性项中的公共系数。

PsdExpr.addTerms()

向半定表达式中添加线性项。

概要

```
void addTerms(VarArray vars, double[] coeffs)
```

参量

`vars`: 新线性项中的变量构成的 VarArray 类。

`coeffs`: 新线性项中的系数数组。

PsdExpr.addTerms()

向表达式中添加一些半定项。

概要

```
void addTerms(PsdVarArray vars, SymMatrixArray mats)
```

参量

`vars`: 新半定项中的半定变量数组。

`mats`: 新半定项中的系数矩阵数组。

PsdExpr.addTerms()

向表达式中添加一些半定项。

概要

```
void addTerms(PsdVar[] vars, SymMatrix[] mats)
```

参量

`vars`: 新半定项中的半定变量数组。

`mats`: 新半定项中的系数矩阵数组。

PsdExpr.clone()

深度拷贝半定表达式对象。

概要

```
PsdExpr clone()
```

返回值

复制的半定表达式对象。

PsdExpr.evaluate()

求解后对半定表达式估值。

概要

```
double evaluate()
```

返回值

表达式估值。

PsdExpr.getCoeff()

获取半定表达式中指定索引值对应项的系数。

概要

```
SymMatExpr getCoeff(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值对应的半定表达式项的系数矩阵表达式。

PsdExpr.getConstant()

获取半定表达式中的常数项。

概要

```
double getConstant()
```

返回值

半定表达式中的常数项。

PsdExpr.getLinExpr()

获取半定表达式中的线性表达式。

概要

```
Expr getLinExpr()
```

返回值

线性表达式对象。

PsdExpr.getPsdVar()

获取半定表达式指定索引值对应项中的半定变量。

概要

```
PsdVar getPsdVar(int i)
```

参量

i: 指定索引值。

返回值

指定索引值对应项的半定变量对象。

PsdExpr.multiply()

对半定表达式乘以常数。

概要

```
void multiply(double c)
```

参量

c: 常数操作数。

PsdExpr.remove()

删除半定表达式中指定索引值的项。

概要

```
void remove(int idx)
```

参量

idx: 指定索引值。

PsdExpr.remove()

删除半定表达式中与指定变量相关的项。

概要

```
void remove(Var var)
```

参量

var: 指定变量。

PsdExpr.remove()

删除半定表达式中与指定半定变量相关的项。

概要

```
void remove(PsdVar var)
```

参量

var: 指定半定变量。

PsdExpr.setCoeff()

设置半定表达式指定索引值对应项的系数矩阵。

概要

```
void setCoeff(int i, SymMatrix mat)
```

参量

i: 指定索引值。

mat: 指定索引值对应项的系数矩阵。

PsdExpr.setConstant()

设置半定表达式中的常数。

概要

```
void setConstant(double constant)
```

参量

constant: 半定表达式中的常数。

PsdExpr.size()

获取半定表达式中的半定项数。

概要

```
long size()
```

返回值

半定表达式中的半定项数。

26.2.33 PsdConstraint 类

PsdConstraint 类是杉数求解器对半定约束的相关操作的封装，提供了以下成员方法：

PsdConstraint.get()

获得半定约束的属性值。支持半定相关的属性。

概要

```
double get(String attr)
```

参量

`attr`: 所需要获得的属性名称。

返回值

双精度属性值。

PsdConstraint.getIdx()

获取半定约束的索引值。

概要

```
int getIdx()
```

返回值

半定约束的索引值。

PsdConstraint.getName()

获取半定约束的名称。

概要

```
String getName()
```

返回值

半定约束的名称。

PsdConstraint.remove()

从模型中删除当前的半定约束。

概要

```
void remove()
```

PsdConstraint.set()

设置半定约束的属性值。支持半定相关的属性。

概要

```
void set(String attr, double value)
```

参量

`attr`: 所需要设置的属性名称。

`value`: 所需要设置的属性值。

PsdConstraint.setName()

设置半定约束的名称。

概要

```
void setName(String name)
```

参量

`name`: 半定约束的名称。

26.2.34 PsdConstrArray 类

为方便用户对一组 Java *PsdConstraint 类* 对象进行操作，杉数求解器的 Java 接口设计了 PsdConstrArray 类，提供了以下成员方法：

PsdConstrArray.PsdConstrArray()

PsdConstrArray 的构造函数。

概要

```
PsdConstrArray()
```

PsdConstrArray.getPsdConstr()

获取半定约束数组中的指定索引值的半定约束。

概要

```
PsdConstraint getPsdConstr(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的半定约束。

PsdConstrArray.pushBack()

向半定约束数组中添加一个半定约束。

概要

```
void pushBack(PsdConstraint constr)
```

参量

`constr`: 待添加的半定约束。

PsdConstrArray.reserve()

预分配大小为 n 项的空间。

概要

```
void reserve(int n)
```

参量

`n`: 容纳 n 项的空间。

PsdConstrArray.size()

获取半定约束数组中的元素个数。

概要

```
int size()
```

返回值

半定约束数组中的元素个数。

26.2.35 PsdConstrBuilder 类

PsdConstrBuilder 类是杉数优化求解器中对构建半定约束的构建器的封装，提供了以下成员方法：

PsdConstrBuilder.PsdConstrBuilder()

PsdConstrBuilder 的构造函数。

概要

```
PsdConstrBuilder()
```

PsdConstrBuilder.getPsdExpr()

获取半定约束相关的表达式。

概要

```
PsdExpr getPsdExpr()
```

返回值

半定表达式对象。

PsdConstrBuilder.getRange()

获取半定约束生成器对象的约束范围的长度（从下界到上界的长度，必须大于 0）。

概要

```
double getRange()
```

返回值

半定约束范围的长度（从下界到上界的长度）。

PsdConstrBuilder.getSense()

获取半定约束相关约束类型。

概要

```
char getSense()
```

返回值

半定约束类型。

PsdConstrBuilder.set()

设置一个半定约束的表达式，类型和边界值。

概要

```
void set(
    PsdExpr expr,
    char sense,
    double rhs)
```

参量

`expr`: 约束一侧的半定表达式。

`sense`: 除了 COPT_RANGE 外的半定约束类型。

`rhs`: 约束另一侧的常数项。

PsdConstrBuilder.setRange()

设置一个范围约束（带有上下界）。

概要

```
void setRange(PsdExpr expr, double range)
```

参量

`expr`: 半定表达式。其表达式的常数项的负数其实是这个约束的上界。

`range`: 约束范围的长度（从下界到上界的长度，必须大于 0）。

26.2.36 PsdConstrBuilderArray 类

为方便用户对一组 Java *PsdConstrBuilder 类* 对象进行操作，杉数优化求解器的 Java 接口设计了 PsdConstrBuilderArray 类，提供了以下成员方法：

PsdConstrBuilderArray.PsdConstrBuilderArray()

PsdConstrBuilderArray 的构造函数。

概要

```
PsdConstrBuilderArray()
```

PsdConstrBuilderArray.getBuilder()

获取半定约束生成器数组中的指定索引值的半定约束。

概要

```
PsdConstrBuilder getBuilder(int idx)
```

参量

`idx`: 指定的索引值。

返回值

指定的半定约束生成器。

PsdConstrBuilderArray.pushBack()

向半定约束生成器数组中添加一个半定约束生成器。

概要

```
void pushBack(PsdConstrBuilder builder)
```

参量

`builder`: 待添加的半定约束生成器。

PsdConstrBuilderArray.reserve()

预分配大小为 n 项的空间。

概要

```
void reserve(int n)
```

参量

`n`: 容纳 n 项的空间。

PsdConstrBuilderArray.size()

获取半定约束生成器数组中的元素个数。

概要

```
int size()
```

返回值

半定约束生成器数组中的元素个数。

26.2.37 LmiConstraint 类

LmiConstraint 类是杉数求解器对 LMI 约束的相关操作的封装，提供了以下成员方法：

LmiConstraint.get()

获取 LMI 约束的属性值。

概要

```
double[] get(String attr)
```

参量

`attr`: 属性的名称。

返回值

输出双精度型数组，保存了属性值。

LmiConstraint.getDim()

获取 LMI 约束的维度。

概要

```
int getDim()
```

返回值

LMI 约束的维度。

LmiConstraint.getIdx()

获取 LMI 约束的索引。

概要

```
int getIdx()
```

返回值

LMI 约束的索引。

LmiConstraint.getLen()

获取 LMI 约束展开后的长度。

概要

```
int getLen()
```

返回值

LMI 约束展开后的长度。

LmiConstraint.getName()

获取 LMI 约束的名称。

概要

```
String getName()
```

返回值

LMI 约束名称。

LmiConstraint.remove()

从模型中删除当前的 LMI 约束。

概要

```
void remove()
```

LmiConstraint.setRhs()

设置 LMI 约束的常数项。

概要

```
void setRhs(SymMatrix mat)
```

参量

`mat`: 所需要设置的对称矩阵。

26.2.38 LmiConstrArray 类

为方便用户对一组 Java *LmiConstraint 类* 对象进行操作，杉数求解器的 Java 接口设计了 LmiConstrArray 类，提供了以下成员方法：

LmiConstrArray.LmiConstrArray()

LmiConstrArray 的构造函数。

概要

```
LmiConstrArray()
```

LmiConstrArray.getLmiConstr()

获取 LMI 约束数组里指定索引的 LMI 约束。

概要

```
LmiConstraint getLmiConstr(int idx)
```

参量

`idx`: 指定的 LMI 约束的索引。

返回值

指定索引的 LMI 约束。

LmiConstrArray.pushBack()

向 LMI 约束数组里添加 LMI 约束。

概要

```
void pushBack(LmiConstraint constr)
```

参量

`constr`: LMI 约束。

LmiConstrArray.reserve()

预分配大小为 n 项的空间。

概要

```
void reserve(int n)
```

参量

`n`: 预分配空间的大小。

LmiConstrArray.size()

获取 LMI 约束数组里 LMI 约束个数。

概要

```
int size()
```

返回值

LMI 约束个数。

26.2.39 LmiExpr 类

COPT 的 LMI 表达式包括对称矩阵和标量变量相乘。LmiExpr 类是杉数求解器中用于构建 LMI 表达式时对变量和对称矩阵相关组合操作，提供了以下成员方法：

LmiExpr.LmiExpr()

LMI 表达式的默认构造函数。

概要

```
LmiExpr()
```

LmiExpr.LmiExpr()

使用对称矩阵构造的 LMI 表达式。

概要

```
LmiExpr(SymMatrix mat)
```

参量

`mat`: 作为常量项的对称矩阵。

LmiExpr.LmiExpr()

使用矩阵表达式构造的 LMI 表达式。

概要

```
LmiExpr(SymMatExpr expr)
```

参量

`expr`: 作为常量项的矩阵表达式。

LmiExpr.LmiExpr()

使用变量和其系数矩阵构造的 LMI 表达式。

概要

```
LmiExpr(Var var, SymMatrix mat)
```

参量

`var`: 添加的这一项对应的变量。

`mat`: 添加的这一项对应的系数矩阵。

LmiExpr.LmiExpr()

使用变量和其系数矩阵表达式构造的 LMI 表达式。

概要

```
LmiExpr(Var var, SymMatExpr expr)
```

参量

`var`: 添加的这一项对应的变量。

`expr`: 新 LMI 项中的对称矩阵表达式。

LmiExpr.addConstant()

加到 LMI 表达式中的常数项。

概要

```
void addConstant(SymMatExpr expr)
```

参量

`expr`: 加到常数项的矩阵表达式对象。

LmiExpr.addLmiExpr()

添加一个 LMI 表达式的项。

概要

```
void addLmiExpr(LmiExpr expr)
```

参量

`expr`: 需要添加的 LMI 表达式。

LmiExpr.addLmiExpr()

添加一个 LMI 表达式的项，并乘以倍数。

概要

```
void addLmiExpr(LmiExpr expr, double mult)
```

参量

`expr`: 需要添加的 LMI 表达式。

`mult`: 倍数。

LmiExpr.addTerm()

向 LMI 表达式中添加一个新项。

概要

```
void addTerm(Var var, SymMatrix mat)
```

参量

`var`: 新项中的变量。

`mat`: 新项中的系数矩阵。

LmiExpr.addTerm()

向 LMI 表达式中添加一项。

概要

```
void addTerm(Var var, SymMatExpr expr)
```

参量

`var`: 新项中的变量。

`expr`: 新项中作为系数的对称矩阵表达式。

LmiExpr.addTerms()

向 LMI 表达式中添加一些项。

概要

```
void addTerms(VarArray vars, SymMatrixArray mats)
```

参量

`vars`: 新项中的变量数组。

`mats`: 新项中的系数矩阵数组。

LmiExpr.addTerms()

向 LMI 表达式中添加一些项。

概要

```
void addTerms(Var[] vars, SymMatrix[] mats)
```

参量

`vars`: 新项中的变量数组。

`mats`: 新项中的系数矩阵数组。

LmiExpr.clone()

深度复制 LMI 表达式。

概要

```
LmiExpr clone()
```

返回值

新的 LMI 表达式对象。

LmiExpr.getCoeff()

获取 LMI 表达式中指定索引值对应项的系数。

概要

```
SymMatExpr getCoeff(int i)
```

参量

`i`: 指定索引值。

返回值

指定索引值对应的 LMI 表达式项的系数矩阵表达式。

LmiExpr.getConstant()

获取 LMI 表达式中的常量项。

概要

```
SymMatExpr getConstant()
```

返回值

对称矩阵表达式对象。

LmiExpr.getVar()

获取 LMI 表达式指定索引值对应项中的变量。

概要

```
Var getVar(int i)
```

参量

i: 指定索引值。

返回值

指定索引值对应项的变量对象。

LmiExpr.multiply()

对 LMI 表达式自乘常数。

概要

```
void multiply(double c)
```

参量

c: 自乘的倍数。

LmiExpr.remove()

删除 LMI 表达式中指定索引值的项。

概要

```
void remove(int idx)
```

参量

idx: 指定的索引。

LmiExpr.remove()

删除 LMI 表达式中与指定变量相关的项。

概要

```
void remove(Var var)
```

参量

var: 指定的变量。

LmiExpr.setCoeff()

设置 LMI 表达式指定索引值对应项的系数矩阵。

概要

```
void setCoeff(int i, SymMatrix mat)
```

参量

`i`: 指定索引值。

`mat`: 指定索引值对应项的系数矩阵。

LmiExpr.setConstant()

设置 LMI 表达式中的常数项。

概要

```
void setConstant(SymMatrix mat)
```

参量

`mat`: 新的对称矩阵。

LmiExpr.size()

获取 LMI 表达式中的项数。

概要

```
long size()
```

返回值

LMI 表达式中的项数。

26.2.40 SymMatrix 类

对称矩阵作为半定项中的系数矩阵，常用在半定表达式，半定约束和半定目标函数中。SymMatrix 类是杉数优化求解器中对称矩阵的封装，提供了以下成员方法：

SymMatrix.getDim()

获取对称矩阵的维度。

概要

```
int getDim()
```

返回值

对称矩阵的维度。

SymMatrix.getIdx()

获取对称矩阵的索引值。

概要

```
int getIdx()
```

返回值

对称矩阵的索引值。

26.2.41 SymMatrixArray 类

为方便用户对一组 Java *SymMatrix 类* 对象进行操作，杉数求解器的 Java 接口设计了 SymMatrixArray 类，提供了以下成员方法：

SymMatrixArray.SymMatrixArray()

SymMatrixArray 的构造函数。

概要

```
SymMatrixArray()
```

SymMatrixArray.getMatrix()

获取对称矩阵数组里指定下标的对称矩阵。

概要

```
SymMatrix getMatrix(int idx)
```

参量

`idx`: 对称矩阵的下标。

返回值

指定下标的对称矩阵。

SymMatrixArray.pushBack()

向对称矩阵数组里附加一个对称矩阵。

概要

```
void pushBack(SymMatrix mat)
```

参量

`mat`: 对称矩阵。

SymMatrixArray.reserve()

预分配大小为 n 项的空间。

概要

```
void reserve(int n)
```

参量

`n`: 容纳 n 项的空间。

SymMatrixArray.size()

获取对称矩阵数组里对称矩阵个数。

概要

```
int size()
```

返回值

对称矩阵个数。

26.2.42 SymMatExpr 类

对称矩阵表达式对于对称矩阵的线性组合，其计算结果实际还是一个对称矩阵。表达式的好处是可以延迟计算结果矩阵，直到设置半定约束或者半定目标函数时。SymMatExpr 类是杉数优化求解器中对称矩阵表达式的封装，提供了以下成员方法：

SymMatExpr.SymMatExpr()

对称矩阵表达式的默认构造函数。

概要

```
SymMatExpr()
```

SymMatExpr.SymMatExpr()

使用对称矩阵和其系数构造的表达式。

概要

```
SymMatExpr(SymMatrix mat, double coeff)
```

参量

`mat`: 添加的这一项对应的对称矩阵。

`coeff`: 添加的这一项对应的参数。

SymMatExpr.addSymMatExpr()

添加一个对称矩阵表达式的项，并乘以倍数。

概要

```
void addSymMatExpr(SymMatExpr expr, double mult)
```

参量

`expr`: 需要添加的对称矩阵表达式。

`mult`: 系数倍数。

SymMatExpr.addTerm()

向对称矩阵表达式中添加一项。

概要

```
Boolean addTerm(SymMatrix mat, double coeff)
```

参量

`mat`: 新项中的对称矩阵。

`coeff`: 新项中的系数。

返回值

布尔值，表示新项是否成功添加。

SymMatExpr.addTerms()

向表达式中添加多个项。

概要

```
int addTerms(SymMatrixArray mats, double[] coeffs)
```

参量

`mats`: 新项中的对称矩阵数组。

`coeffs`: 新项中的系数数组。

返回值

增加的项数。如果返回负值，至少有一项添加失败。

SymMatExpr.addTerms()

向表达式中添加多个项。

概要

```
int addTerms(SymMatrix[] mats, double[] coeffs)
```

参量

`mats`: 新项中的对称矩阵数组。

`coeffs`: 新项中的系数数组。

返回值

增加的项数。如果返回负值，至少有一项添加失败。

SymMatExpr.addTerms()

向表达式中添加多个项。

概要

```
int addTerms(SymMatrix[] mats, double coeff)
```

参量

`mats`: 新项中的对称矩阵数组。

`coeff`: 新项中的公用系数。

返回值

增加的项数。如果返回负值，至少有一项添加失败。

SymMatExpr.clone()

深度拷贝对称矩阵表达式。

概要

```
SymMatExpr clone()
```

返回值

新的表达式对象。

SymMatExpr.getCoeff()

获取表达式中指定索引值对应项的系数。

概要

```
double getCoeff(int i)
```

参量

i: 指定索引值。

返回值

指定索引值对应的表达式项的系数。

SymMatExpr.getDim()

获取表达式中对称矩阵的维度。

概要

```
int getDim()
```

返回值

对称矩阵的维度。

SymMatExpr.getSymMat()

获取表达式指定索引值对应项中的对称矩阵。

概要

```
SymMatrix getSymMat(int i)
```

参量

i: 指定索引值。

返回值

指定索引值对应项的对称矩阵对象。

SymMatExpr.multiply()

对对称矩阵表达式乘以常数。

概要

```
void multiply(double c)
```

参量

c: 常数操作数。

SymMatExpr.remove()

删除表达式中指定索引值的项。

概要

```
void remove(int idx)
```

参量

`idx`: 指定索引值。

SymMatExpr.remove()

删除对称矩阵表达式中与指定对称矩阵相关的项。

概要

```
void remove(SymMatrix mat)
```

参量

`mat`: 指定的对称矩阵。

SymMatExpr.reserve()

预分配大小为 n 项的空间。

概要

```
void reserve(int n)
```

参量

`n`: 容纳 n 项的空间。

SymMatExpr.setCoeff()

设置表达式指定索引值项数中的系数。

概要

```
void setCoeff(int i, double val)
```

参量

`i`: 指定索引值。

`val`: 指定索引值项数中的参数。

SymMatExpr.size()

获取对称矩阵表达式中的项数。

概要

```
long size()
```

返回值

对称矩阵表达式中的项数。

26.2.43 CallbackBase 类

CallbackBase 类给用户提供了在求解过程中介入的接口。这个是抽象类，用户需要自己实现虚函数 `virtual void CallbackBase::callback()` 才能创建实例，用来作为 `Model::SetCallback(CallbackBase cb, int cbctx)` 方法的第一个参数传入。CallbackBase 类也提供了以下可以继承的成员方法：

CallbackBase.CallbackBase()

CallbackBase 的构造函数，实现 ICallback 接口。

概要

```
CallbackBase()
```

CallbackBase.addLazyConstr()

向模型中增加一个惰性约束。

概要

```
void addLazyConstr(
    Expr lhs,
    char sense,
    double rhs)
```

参量

`lhs`: 惰性约束表达式。

`sense`: 惰性约束的类型。

`rhs`: 惰性约束的右侧值。

CallbackBase.addLazyConstr()

向模型中增加一个惰性约束。

概要

```
void addLazyConstr(
    Expr lhs,
    char sense,
    Expr rhs)
```

参量

lhs: 惰性约束的左侧表达式。

sense: 惰性约束的类型。

rhs: 惰性约束的右侧表达式。

CallbackBase.addLazyConstr()

向模型中增加一个惰性约束。

概要

```
void addLazyConstr(ConstrBuilder builder)
```

参量

builder: 惰性约束生成器。

CallbackBase.addLazyConstrs()

向模型中增加多个惰性约束。

概要

```
void addLazyConstrs(ConstrBuilderArray builders)
```

参量

builders: 一组惰性约束生成器。

CallbackBase.addUserCut()

向模型中增加一个割平面。

概要

```
void addUserCut(
    Expr lhs,
    char sense,
```

```
    double rhs)
```

参量

lhs: 割平面表达式。

sense: 割平面的类型。

rhs: 割平面的右侧值。

CallbackBase.addUserCut()

向模型中增加一个割平面。

概要

```
void addUserCut(
    Expr lhs,
    char sense,
    Expr rhs)
```

参量

lhs: 割平面的左侧表达式。

sense: 割平面的类型。

rhs: 割平面的右侧表达式。

CallbackBase.addUserCut()

向模型中增加一个割平面。

概要

```
void addUserCut(ConstrBuilder builder)
```

参量

builder: 割平面生成器。

CallbackBase.addUserCuts()

向模型中增加多个割平面。

概要

```
void addUserCuts(ConstrBuilderArray builders)
```

参量

builders: 一组割平面生成器。

CallbackBase.callback()

定义在 ICallback 接口里的纯虚函数，需要用户覆盖对应实现。

概要

```
void callback()
```

CallbackBase.getDblInfo()

在回调中获取指定信息名的双精度值。

概要

```
double getDblInfo(String cbinfo)
```

参量

`cbinfo`: 回调中的信息名。

返回值

所需的信息值。

CallbackBase.getIncumbent()

在回调中获取指定变量的最优可行解。

概要

```
double getIncumbent(Var var)
```

参量

`var`: 指定的变量。

返回值

变量对应的最优可行解。

CallbackBase.getIncumbent()

在回调中获取一组变量的最优可行解。

概要

```
double[] getIncumbent(VarArray vars)
```

参量

`vars`: 变量数组。

返回值

一组变量对应的最优可行解。

CallbackBase.getIncumbent()

在回调中获取一组变量的最优可行解。

概要

```
double[] getIncumbent(Var[] vars)
```

参量

vars: 变量数组。

返回值

一组变量对应的最优可行解。

CallbackBase.getIncumbent()

在回调中获取全部变量的最优可行解。

概要

```
double[] getIncumbent()
```

返回值

全部变量对应的最优可行解。

CallbackBase.getIntInfo()

在回调中获取指定信息的整数值。

概要

```
int getIntInfo(String cbinfo)
```

参量

cbinfo: 回调中的信息名。

返回值

所需的信息值。

CallbackBase.getRelaxSol()

在回调中获取指定变量的线性松弛解。

概要

```
double getRelaxSol(Var var)
```

参量

var: 指定的变量。

返回值

变量对应的线性松弛解。

CallbackBase.getRelaxSol()

在回调中获取一组变量的线性松弛解。

概要

```
double[] getRelaxSol(VarArray vars)
```

参量

vars: 变量数组。

返回值

一组变量对应的线性松弛解。

CallbackBase.getRelaxSol()

在回调中获取一组变量的线性松弛解。

概要

```
double[] getRelaxSol(Var[] vars)
```

参量

vars: 变量数组。

返回值

一组变量对应的线性松弛解。

CallbackBase.getRelaxSol()

在回调中获取全部变量的线性松弛解。

概要

```
double[] getRelaxSol()
```

返回值

全部变量对应的线性松弛解。

CallbackBase.getSolution()

在回调中获取指定变量的解。

概要

```
double getSolution(Var var)
```

参量

var: 指定的变量。

返回值

变量对应的解。

CallbackBase.getSolution()

在回调中获取一组变量的解。

概要

```
double[] getSolution(VarArray vars)
```

参量

`vars`: 变量数组。

返回值

一组变量对应的解。

CallbackBase.getSolution()

在回调中获取一组变量的解。

概要

```
double[] getSolution(Var[] vars)
```

参量

`vars`: 变量数组。

返回值

一组变量对应的解。

CallbackBase.getSolution()

在回调中获取全部变量的解。

概要

```
double[] getSolution()
```

返回值

全部变量对应的解。

CallbackBase.interrupt()

中断回调中正在求解的问题。

概要

```
void interrupt()
```

CallbackBase.loadSolution()

向模型中添加自定义解。

概要

```
double loadSolution()
```

返回值

解对应的目标函数值。

CallbackBase.setSolution()

在回调中对给定的变量设置自定义解。

概要

```
void setSolution(Var var, double val)
```

参量

`var`: 变量对象。

`val`: 双精度值。

CallbackBase.setSolution()

在回调中对一组变量设置自定义解。

概要

```
void setSolution(VarArray vars, double[] vals)
```

参量

`vars`: 变量数组。

`vals`: 双精度值数组。

CallbackBase.setSolution()

在回调中对一组变量设置自定义解。

概要

```
void setSolution(Var[] vars, double[] vals)
```

参量

`vars`: 变量数组。

`vals`: 双精度值数组。

CallbackBase.where()

获取回调中的上下文。

概要

```
int where()
```

返回值

上下文对应的整数值。

26.2.44 ProbBuffer 类

ProbBuffer 类是字符流缓冲区的封装，提供了以下成员方法：

ProbBuffer.ProbBuffer()

ProbBuffer 的构造函数。

概要

```
ProbBuffer(int sz)
```

参量

`sz`: ProbBuffer 的初始大小

ProbBuffer.getData()

获取在缓存里的问题字符流。

概要

```
String getData()
```

返回值

缓存中的问题字符流。

ProbBuffer.resize()

调整缓存到给定大小。

概要

```
void resize(int sz)
```

参量

sz: 指定的缓存大小。

ProbBuffer.size()

获取缓存大小。

概要

```
int size()
```

返回值

缓存大小。

26.2.45 CoptException 类

CoptException 类是杉数优化求解器的错误处理相关操作的封装。当方法调用对应的杉数求解器底层接口发生错误时，则抛出 CoptException 类的异常，提供了以下属性值访问相应的错误信息：

CoptException.CoptException()

CoptException 的构造函数。

概要

```
CoptException(int code, String msg)
```

参量

code: 异常错误代码。

msg: 异常的错误信息。

CoptException.getCode()

获取异常错误代码。

概要

```
int getCode()
```

返回值

异常错误代码。