
ArhiPlex. Руководство пользователя

Выпуск 2.7.0.260887

окт. 06, 2026

1	Введение	2
2	Установка	3
2.1	Требования	3
2.2	Linux	4
3	Лицензии Солвера ArhiPlex 2.6	8
3.1	Демо лицензия (Time-limited Evaluation License)	8
3.2	Лицензия одной машины (Single-Machine License)	9
3.3	Лицензия сервера вычислений (Compute Server License)	10
4	Установка лицензии	11
4.1	Сервер лицензирования	11
4.2	Вариант файла лицензии	13
5	Быстрый старт	15
5.1	Запуск из командной строки	15
5.2	Примеры входных файлов (LP/MPS)	16
5.3	Быстрый старт на Python	17
5.4	Что дальше	18
6	Основные понятия	19
6.1	Как выполняется расчёт	19
6.2	Разрыв оптимальности (gap)	20
6.3	Допуски (tolerances)	20
6.4	Начальное решение	20
6.5	Двойственные оценки	21
6.6	Диагностика недостижимости (IIS)	21
6.7	Интерпретация результата	21
7	Глоссарий	22
8	Параметры расчёта	25
8.1	Способы задания параметров	25
8.2	Сводная таблица	26
8.3	Ограничение ресурсов	27
8.4	Управление алгоритмом	28

8.5	Точности и разрывы (MIP)	28
8.6	Диагностика	30
8.7	Логирование	30
8.8	Вывод решения	31
8.9	Удалённый расчёт (ArhiCloud)	31
8.10	Многопроцессный режим и параллельный запуск	32
9	Статусы и коды возврата	33
9.1	Статус запуска расчёта	33
9.2	Статус решения	34
9.3	Текстовые статусы в логе и отчёте	35
9.4	Статусы вычисления IIS	36
9.5	Статус базиса переменных и ограничений	36
10	Форматы файлов	37
10.1	LP	37
10.2	MPS	42
10.3	SOL	43
11	Лог вычислений	44
11.1	Вводная часть	44
11.2	Табличная часть	45
11.3	Результирующая часть	46
12	Импорт и экспорт моделей	47
12.1	Импорт моделей из файла	47
12.2	Экспорт модели в файл	47
13	Создание модели с использованием матриц	49
13.1	Введение	49
13.2	Примеры использования	50
14	Двойственные решения	60
14.1	Пример	60
15	Начальные решения	63
15.1	Частичный старт MILP и бюджет достроения	64
16	Анализ недостижимости модели	66
16.1	Введение	66
16.2	Использование IIS API в Arhiplex	67
17	Python API	73
17.1	Подключение	73
17.2	Сквозной пример	74
17.3	Ключевые типы	75
17.4	Справочник (пакет)	76
17.5	Справочник (native-модуль)	76
18	C++ API	119
18.1	Подключение	119
18.2	Сквозной пример	120
18.3	Справочник API	123
19	C# API	144
19.1	Подключение	144

19.2	Сквозной пример	145
19.3	Справочник API	147
20	Java API	166
20.1	Требования	166
20.2	Установка (готовый JAR)	166
20.3	Сборка из исходников (необязательно)	167
20.4	Native-библиотека	167
20.5	Управление ресурсами	167
20.6	Быстрый старт	167
20.7	Синтаксис выражений	169
20.8	Квадратичные (QCQP/MIQCQP) выражения	169
20.9	Matrix API	169
20.10	Общие (нелинейные) выражения	170
20.11	IIS (недостижимость)	171
20.12	Bulk-векторы и remote result	171
20.13	Логирование	171
20.14	Тесты	171
20.15	Справочник API	172
21	Julia API	173
21.1	Подключение пакета	173
21.2	Базовый пример	174
21.3	Поддерживаемые конструкции	174
21.4	Параметры решателя	174
21.5	Чтение результатов	175
21.6	Начальное решение (MIP start)	175
22	CP-решатель (ArhiPlex CP)	176
22.1	Когда использовать ArhiPlex CP	176
22.2	Доступность в сборке	177
22.3	Импорт CP API	177
22.4	Базовый пример выполнимости	177
22.5	Пример оптимизации	178
22.6	Логические и комбинаторные ограничения	179
22.7	Интервалы и задачи расписаний	179
22.8	Callback и поиск всех решений	180
22.9	Статусы решения	181
22.10	Параметры решателя	182
22.11	Ограничения и совместимость	182
22.12	См. также	182
23	История изменений	183
23.1	Версия 2.7.0	183
23.2	Версия 2.6.2: допустимость решений MIP	184
23.3	Версия 2.0.11	185
23.4	Версия 2.1.0	186
23.5	Версия 2.2.0	186
23.6	Версия 2.4.0	186
23.7	Версия 2.5.0	187
23.8	Версия 2.6.0	187
23.9	Версия 2.6.1	188
23.10	Версия 2.6.2	188
24	Техническая поддержка	189

24.1	Возникновение проблемы и создание обращения пользователя	189
24.2	Обработка и анализ обращений	189
24.3	Разработка новой версии (спецверсии) солвера	190
24.4	Выпуск новой версии (спецверсии) солвера	190
24.5	Информирование клиента	190
25	Контактная информация	192
	Содержание модулей Python	193

ArhiPlex 2.6 — коммерческий решатель задач оптимизации (LP, MILP, QP, QCQP, NLP, MINLP и constraint programming). Настоящее руководство описывает установку, лицензирование, запуск расчётов, программные интерфейсы и справочные материалы по параметрам и статусам.

Если вы работаете с ArhiPlex впервые, начните с разделов *Введение* и *Быстрый старт*. Базовые понятия оптимизации и термины руководства собраны в разделах *Основные понятия* и *Глоссарий*.

ArhiPlex 2.6 — коммерческий решатель задач оптимизации. Он позволяет решать задачи линейного программирования (LP), смешанно-целочисленного линейного программирования (MILP), выпуклого и невыпуклого программирования с квадратичными ограничениями (QCQP) и квадратичной целевой функцией (QP), а также общего нелинейного программирования (NLP), в том числе с целочисленными переменными (MINLP). Кроме того, поддерживаются задачи программирования в ограничениях (CP), включая комбинаторные модели и задачи планирования, решаемые CP-решателем *ArhiPlex CP*.

Определения используемых терминов приведены в разделе *Глоссарий*, базовые понятия — в разделе *Основные понятия*.

Солвер устанавливается на следующих операционных системах: Ubuntu (20.04, 22.04), Debian (10, 11, 12), Astra Linux (1.7, 1.8), RHEL (8), Redos (7.3). Для взаимодействия с солвером реализованы программные интерфейсы для следующих языков:

- C++
- Python
- C#
- Java
- Julia

2.1 Требования

Для запуска и работы солвера **ArhiPlex 2.6** необходима поддерживаемая ОС Linux (Ubuntu 20.04/22.04, Debian 10/11/12, Astra Linux 1.7/1.8, RHEL 8, Redos 7.3). Дистрибутив поставляется в виде архива `linux-x64`.

Примечание: Полноценная локальная установка солвера поддерживается только на Linux. В Windows доступен удалённый клиент (ArhiPlex Remote Client): модель строится локально, а расчёт выполняется на сервере ArhiCloud. См. разделы *C++ API*, *C# API*.

Ниже приведены аппаратные характеристики, **рекомендуемые для крупных задач/нагрузочных расчётов**. Фактическое потребление ресурсов зависит от размера модели и настроек запуска.

Таблица 2.1: Рекомендуемые характеристики (для крупных задач)

Компонент	Минимальное значение
Процессор	24 CPU
Оперативная память	256 GB и выше
Дисковое пространство	Не менее 50 GB

2.2 Linux

Ниже приведён пошаговый сценарий установки до «первого успешного запуска» (CLI и Python API).

Что понадобится:

- `tar` для распаковки архива
- `python3` и `pip` (для Python API)
- поддержка `venv` (проверьте, что выполняется `python3 -m venv --help`)
- JDK 11+ и Maven 3.6+ (для Java API)

2.2.1 Распаковка дистрибутива

Распакуйте архив в удобную папку:

```
tar -xzf linux-x64-licensed-arhiplex-<version>.tar.gz
```

После распаковки появится папка `ARHIPLEX-<version>-Linux`.

Куда установить:

- В системную директорию (например, `/opt`) — потребуется `sudo`:

```
sudo mv ARHIPLEX-<version>-Linux /opt/
```

- В домашнюю директорию (без `sudo`):

```
mv ARHIPLEX-<version>-Linux "${HOME}/"
```

2.2.2 Переменные окружения

Добавьте в `~/.bashrc` (или `~/.profile`) следующие строки, подставив корректный путь:

```
export ARHIPLEX_HOME=/opt/ARHIPLEX-<version>-Linux
export PATH=${ARHIPLEX_HOME}/bin:${PATH}
export LD_LIBRARY_PATH=${ARHIPLEX_HOME}/lib64:${LD_LIBRARY_PATH}
```

Примечание: Если вы установили дистрибутив в домашнюю директорию, используйте, например: `export ARHIPLEX_HOME="${HOME}/ARHIPLEX-<version>-Linux"`.

Примените изменения в текущем терминале:

```
source ~/.bashrc
```

Проверьте, что переменные установлены:

```
echo "${ARHIPLEX_HOME}"
echo "${PATH}"
echo "${LD_LIBRARY_PATH}"
```

2.2.3 Проверка установки CLI

Проверьте, что исполняемый файл доступен в PATH:

```
arhiplex --help
```

Дальнейшие примеры запуска см. в *Быстрый старт*.

2.2.4 Установка Python API (рекомендуется через venv)

Примечание: Рекомендуется устанавливать Python API в виртуальное окружение, чтобы избежать конфликтов версий и установки «в другой Python».

Создайте и активируйте виртуальное окружение:

```
python3 -m venv .venv
source .venv/bin/activate
```

Обновите pip в окружении:

```
python -m pip install --upgrade pip
```

Установите Python API из дистрибутива:

```
cd "${ARHIPLEX_HOME}/lib64/python"
python -m pip install .
```

Предупреждение: На текущий момент для Linux поддерживается Python версий 3.7, 3.8, 3.9, 3.10, 3.11, 3.12.

2.2.5 Smoke test (Python API)

Проверьте, что модуль импортируется и установлен именно в активное окружение:

```
python -c "import arhiplexpy; print('module:', arhiplexpy.__file__); print('doc:',
↳(arhiplexpy.__doc__ or '')[:80])"
```

Если требуется, ознакомьтесь с интерфейсом в *Python API*.

2.2.6 Установка Java API

Требования:

- JDK 11 или новее
- Maven 3.6+ — только для необязательной сборки JAR из исходников

Java API поставляется готовым самодостаточным файлом `${ARHIPLEX_HOME}/lib64/java/arhiplex-java-<версия>.jar` (с версией продукта): положите его на classpath. JAR платформо-независим и уже содержит зависимость JNA — собирать из исходников не нужно.

Необязательно, JAR можно собрать самостоятельно из исходников дистрибутива:

```
cd "${ARHIPLEX_HOME}/include/arhiplex/java"
mvn package
```

Если JAR собран через CMake (`-DBUILD_JAVA_API=ON`), он устанавливается в `${ARHIPLEX_HOME}/lib64/java/arhiplex-java-<версия>.jar`.

Убедитесь, что `libarhiplex_cpp` доступна JVM (`LD_LIBRARY_PATH` в Linux, `PATH` в Windows) — в JAR она не входит и ставится отдельно как часть дистрибутива солвера.

2.2.7 Smoke test (Java API)

Компиляция и запуск примера из дистрибутива:

```
cd "${ARHIPLEX_HOME}/examples"
javac -cp "${ARHIPLEX_HOME}/lib64/java/arhiplex-java-<версия>.jar" example_java.java
java -cp ".:${ARHIPLEX_HOME}/lib64/java/arhiplex-java-<версия>.jar" ExampleJava
```

Подробнее об API см. *Java API*.

2.2.8 ArhiPlex CP (опционально)

CP-решатель доступен только в тех сборках, где был включён `BUILD_CP_SOLVER`. Для Python это означает, что вместе с пакетом `arhiplexpy` должны быть установлены подмодули `arhiplexpy.arhitools`.

Быстрая проверка доступности:

```
python -c "import arhiplexpy; print('CP available:', arhiplexpy.has_cp_solver())"
```

Если вывод содержит `CP available: True`, можно использовать модуль `from arhiplexpy.arhitools.sat.python import cp_model`. Подробности и примеры см. в разделе *CP-решатель (ArhiPlex CP)*.

2.2.9 Лицензия (обязательно для расчётов)

Для выполнения расчётов требуется лицензия. Как правило, в корне `$ARHIPLEX_HOME` должен находиться файл конфигурации `config.toml`.

Подробное описание вариантов и шагов активации см. в разделе *Лицензии Солвера ArhiPlex 2.6*.

2.2.10 Типовые проблемы

- `pip/python` не совпадают (установилось «не туда») — используйте `python -m pip ...` в активированном окружении `venv`.
- `ModuleNotFoundError: No module named 'arhiplexpy'` — убедитесь, что вы активировали `venv` (`source .venv/bin/activate`) и устанавливали пакет командой `python -m pip install .` из `${ARHIPLEX_HOME}/lib64/python`.
- `ImportError: ... cannot open shared object file` (Linux) — проверьте, что задан `LD_LIBRARY_PATH=${ARHIPLEX_HOME}/lib64` и применены изменения (`source ~/.bashrc`), а `ARHIPLEX_HOME` указывает на корректную папку установки.

- Нет прав на `/opt` — установите в домашнюю директорию (без `sudo`) или используйте `sudo mv ...` при переносе в `/opt`.
- Корпоративный проху/закрытый интернет — настройте проху для `pip` в вашем окружении или используйте офлайн-репозиторий/локальное зеркало пакетов (в соответствии с правилами вашей организации).

Лицензии Солвера **ArhiPlex 2.6**

Солвер **ArhiPlex 2.6** имеет различные варианты лицензирования.

3.1 Демо лицензия (Time-limited Evaluation License)

Данная лицензия предназначена только для оценки производительности и возможности использования Пользователем. Лицензия не может быть использована после окончания срока действия.

Расчеты при данном виде лицензии производятся на машинах Пользователя, роль Лицензионного Сервера, проверяющего наличие активной лицензии при запуске расчета, выполняет машина, расположенная на облачных ресурсах **ARNITEX**. Программа может быть установлена и запущена на любой машине с операционной системой, которая входит в число поддерживаемых платформ.

Данная Лицензия допускает только один расчет одновременно на единицу времени и дает право на использование Программы в соответствии с заявленными в технической документации функциональными возможностями на неограниченном количестве вычислительных машин, где установлена Программа. Под расчетом подразумевается запуск оптимизации или расчета файла *iis*, в случае прерывания расчета Пользователем или из-за аварийного завершения работы программы Пользователя, может потребоваться до 6 минут, чтобы данный расчет стал неактивным для Лицензионного сервера и освободил место для следующего запуска.

Демо лицензия предоставляется на 30 дней и может быть продлена по соглашению. Запрещено использование в коммерческих целях, а также запрещена передача и публикация результатов тестирования.

Назначение: Предназначена для оценки производительности и функциональности программного обеспечения.

Условия активации лицензии: Лицензия активируется на машине Пользователя через Лицензионный сервер, расположенный на облачных ресурсах **ARNITEX** посредством прописывания предоставленного Вендором токена в файл конфигурации.

Условия использования Солвера: Может быть установлена и запущена на любой поддерживаемой операционной системе. Действует на неограниченном количестве вычислительных машин.

Ограничения:

- Одновременно разрешен только один расчет.
- Запрещено использование в коммерческих целях, передача и публикация результатов тестирования.

Срок действия: 30 дней, продление по соглашению.

3.2 Лицензия одной машины (Single-Machine License)

Лицензия привязана к физической или виртуальной машине и ее использование ограничено количеством потоков (CPU) которые доступны в рамках лицензии. Эта лицензия может работать на машинах с количеством ядер, превышающим указанное в лицензии, однако задействовано будет только количество ядер/потоков, указанное в Договоре.

Определяющим параметром для данного типа Лицензии являются аппаратная информация машины, на которой планируется использование Программы. Для целей сбора аппаратной информации Вендор передает Пользователю программное решение для генерации файла аппаратной информации в срок не более 3 дней с даты подписания Договора. Файл аппаратной информации передается Вендору для генерации ключа активации лицензии в срок не более чем 5 (пять) рабочих дней с даты получения программного решения для генерации данного файла.

Лицензия одной машины предоставляется на определенный срок или бессрочно с момента передачи активационного ключа. Запрещено использование в платформах контейнеризации.

Назначение: Подходит

- Компаниям, имеющим единичные установки и использования систем с оптимизационными решениями.
- Компаниям, не имеющим выделенной ИТ-инфраструктуры проведения высоконагруженных расчетов.

Условия активации лицензии: Активация осуществляется через передачу аппаратной информации Вендору и дальнейшим предоставлением Пользователю файла с лицензионным ключом.

Условия использования Солвера: Работа возможна на машине с количеством ядер, превышающим указанное в лицензии, но задействуются только прописанное в Договоре количество ядер/потоков.

Ограничения:

- Запрещено использование в платформах контейнеризации.
- Может быть размещена только на физической машине.
- Расчеты ограничены производительность физической машины.

Срок действия: Определен Договором с Вендором.

3.3 Лицензия сервера вычислений (Compute Server License)

Лицензия сервера вычислений подразумевает два варианта взаимодействия, в зависимости от того, на чьей стороне расположена машина – Лицензионный сервер, выполняющая проверку наличия лицензии:

- Лицензия действует в одной заданной локальной сети машин Пользователя, одна конкретная физическая машина в этом наборе выступает в роли Лицензионного Сервера. Программа может быть установлена и запущена на любой машине в заданной локальной сети.
- Машина, выполняющая роль Лицензионного Сервера, расположена на облачных ресурсах **ARNITEX**. Каждый раз при запуске расчета на машине Пользователя, Программа подключается к Лицензионному серверу **ARNITEX** и проверяет наличие действующей лицензии для Пользователя, если все корректно, то на машине Пользователя запускается расчет.

Лицензия сервера вычислений разрешает применение функциональных возможностей согласно технической документации на вычислительных машинах с поддерживаемыми операционными системами, однако ограничивает число одновременных расчетов, прописанное в Договоре. Под расчетом подразумевается запуск оптимизации или расчета файла *iis*, в случае прерывания расчета Пользователем или из-за аварийного завершения работы программы Пользователя, может потребоваться до 6 минут, чтобы данный расчет стал неактивным для Лицензионного сервера и освободил место для других запусков.

Лицензия сервера вычислений предоставляется на определенный срок с момента Активации. Лицензия сервера вычислений допускает использование в платформах контейнеризации при условии установки Сервера Лицензирования на физическую или виртуальную машину.

Активация Лицензии сервера вычислений на стороне Пользователя осуществляется самим Пользователем в соответствии с Руководством Администратора, включенного в состав передаваемых компонент ПО.

Назначение: Позволяет использовать программное обеспечение на неограниченном количестве вычислительных машин в условиях локальной сети, если сервер Лицензирования расположен на стороне Пользователя или на любой машине с доступом в интернет, если проверка лицензии осуществляется через облачные ресурсы **ARNITEX**. Что подходит:

- Компаниям, имеющим множество систем с оптимизационными задачами.
- Компаниям, имеющим централизованную и масштабную ИТ-инфраструктуру для проведения высоконагруженных вычислений.
- Компаниям, где ведется разработка, эксплуатация и поддержка оптимизационных решений.

Условия активации лицензии: Активация выполняется Пользователем в соответствии с Руководством Администратора, если сервер Лицензирования находится на стороне Пользователя. Или при помощи полученного токена и прописывания его в конфигурационном файле, если сервер Лицензирования находится на стороне **ArhiPlex**.

Условия использования Солвера: Используется в рамках заявленных функциональных возможностей и доступно на поддерживаемых платформах.

Ограничения:

- Установленный лимит на количество одновременных расчетов (указывается в Договоре).
- Допускается использование в платформах контейнеризации только при установке Сервера Лицензирования на физическую или виртуальную машину.

Срок действия: Определен Договором с Вендором.

Установка лицензии

Для корректного функционирования **ArhiPlex 2.6** и выполнения расчетов требуется наличие лицензии. Лицензирование может осуществляться двумя способами: через подключение к серверу лицензирования или через файл лицензии с ключом.

4.1 Сервер лицензирования

Сервер лицензирования необходим для управления количеством активных экземпляров **ArhiPlex 2.6**. Доступны следующие варианты лицензий:

- Демо лицензия (Time-limited Evaluation License): лицензия, проверяемая через сервер на стороне компании **ARNITEX**.
- Лицензия сервера вычислений (Compute Server License) и Лицензия приложений (Application-Specific License): сервер лицензирования может быть установлен либо в контуре клиента, либо на облачных ресурсах **ARNITEX**.

4.1.1 Сервер лицензирования расположен в контуре клиента

Возможны два варианта установки сервера лицензирования, с использованием *kerberos* и без:

Для установки сервера лицензирования в локальной сети Пользователя с использованием *kerberos* необходимо:

1. Установить сервер лицензирования администратором согласно инструкции *Инструкция_по_установке_для_Системного_администратора*, переданной Пользователю.
2. Настроить подключение локально установленного экземпляра **ArhiPlex 2.6** к серверу лицензирования согласно инструкции *Инструкция_по_установке_для_Администратора*, переданной Пользователю.

Для установки сервера лицензирования в локальной сети Пользователя без использования *kerberos* необходимо:

1. Установить сервер лицензирования администратором согласно инструкции *Инструкция по установке для Системного администратора*, переданной Пользователю.
2. Настроить подключение локально установленного экземпляра **ArhiPlex 2.6** к серверу лицензирования.

Для этого необходимо:

- Поместить файл *rootCA.crt*, полученный у системного администратора в корневую директорию установленного Солвера.
- Отредактировать файл конфигурации *config.toml*, это можно сделать при помощи любого текстового редактора. Конфигурационный файл *config.toml* уже расположен в корневой папке Солвера, адрес которой должен совпадать с переменной окружения *ARHIPLEX_HOME*. Значение переменной можно узнать с помощью команды:

```
echo ${ARHIPLEX_HOME}
```

Если есть расхождение, то измените значение переменной *ARHIPLEX_HOME*, см. *Установка*.

Файл конфигурации *config.toml* должен содержать следующие параметры:

```
SERVER = "astra-1c.dev.tr-support.ru"
PORT = "8443"
CACERT = "/opt/arhiplex/rootCA.crt"
AUTH_TYPE = "access_token"
ACCESS_TOKEN = "some_access_token"
```

Где:

- SERVER - адрес Сервера Лицензирования (FQDN)
- PORT - порт Сервера Лицензирования
- CACERT - путь к сертификату корневого центра сертификации
- AUTH_TYPE - способ аутентификации
- ACCESS_TOKEN - сгенерированный токен

Необходимо убедиться, что указан верный адрес и порт Сервера Лицензирования и путь к сертификату корневого центра сертификации. Проверить, что аутентификация происходит через токен, для этого *AUTH_TYPE* = «*access_token*», а также вписать в поле *ACCESS_TOKEN* токен, состоящий из буквенно-цифровых символов, генерируется Пользователем по инструкции *Инструкция по установке для Системного администратора*.

4.1.2 Сервер лицензирования расположен на облачных ресурсах компании ARHITEX

Для успешной работы и осуществления вычислений требуется подключение каждого установленного экземпляра Солвера к Серверу Лицензирования, расположенного на облачных ресурсах компании **ARHITEX**. Для этого необходимо отредактировать файл конфигурации *config.toml*, это можно сделать при помощи любого текстового редактора. Конфигурационный файл *config.toml* уже расположен в корневой папке Солвера, адрес которой должен совпадать с переменной окружения *ARHIPLEX_HOME*, значение переменной можно узнать с помощью команды:

```
echo ${ARHIPLEX_HOME}
```

Если есть расхождение, то измените значение переменной *ARHIPLEX_HOME*, см. *Установка*.

Файл конфигурации *config.toml* содержит следующие параметры:

```
SERVER="web-license.arhitex.com"
ACCESS_TOKEN="token_name"
```

Где **SERVER** - это адрес (имя) Сервера Лицензирования (FQDN), а **ACCESS_TOKEN** - токен, предоставляемый компанией **ARHITEX** с поставкой ПО.

4.2 Вариант файла лицензии

Для установки лицензии для одной машины (Single-Machine License) необходимо выполнить следующие шаги:

1. Получить от Вендора файл-скрипт *get_machine_params* для генерации информации, необходимой для создания лицензии.
2. Запустить скрипт в терминале на машине, для которой будет установлен **ArhiPlex 2.6**, находясь в директории, где лежит *get_machine_params*:

```
./get_machine_params
```

Примечание: При возникновении ошибок с идентификатором оборудования *hardware_uid* на виртуальной машине, например,

```
$ ./get_machine_params
Идентификатор физического оборудования не найден
open /sys/firmware/dmi/tables/smbios_entry_point: permission denied
Идентификаторы оборудования не найдены
```

следует использовать суперпользователя *root*.

3. Результат выполнения скрипта – файл *machine_params.json* будет лежать в директории, откуда запускался скрипт. Данный файл нужно отправить Вендору для генерации файла лицензии.

Содержание файла:

```
{
  "hostname": "***",
  "host_id": "***",
  "os": "***",
  "architecture": "***",
  "username": "***",
  "physical_cores": ***,
  "cpu": "***",
  "hardware_uid": "***"
}
```

4. После получения файла лицензии *license.dat*, его следует поместить в корневую папку установленного программного обеспечения, расположение файла должно совпадать с переменной окружения *ARHIPLEX_HOME*, значение переменной можно узнать вызовом команды:

```
echo ${ARHIPLEX_HOME}
```

Если есть расхождение, то измените значение переменной *ARHIPLEX_HOME*, см. *Установка*.

4.2.1 Перенос лицензии Single-Machine License на новую машину

Данная процедура используется в случаях, предусмотренных договором на программное обеспечение. Последовательность действий:

1. Направление запроса.
 - Отправьте запрос на перенос лицензии на адрес **support@arhitex.com** или по телефону **+7 (812) 245-68-00**.
 - Используйте только почту и телефон, указанные в лицензионном договоре.
 - В запросе укажите причину переноса и описание новой машины.
2. Сбор системной информации.
 - Получите от службы поддержки скрипт сбора данных о системе *get_machine_params*.
 - Выполните скрипт на целевой машине согласно инструкции, описанной выше.
 - Отправьте результат выполнения скрипта в ответном письме.
3. Активация новой лицензии.
 - Получите новый файл лицензии *license.dat* на указанную в договоре почту.
 - При необходимости запросите актуальный дистрибутив ПО.
 - Выполните установку согласно руководству пользователя.

Важно: Все взаимодействие осуществляется исключительно через официальные каналы связи, указанные в договоре.

Быстрый старт

Этот раздел показывает, как за несколько минут выполнить первый расчёт — из командной строки и из Python. Предполагается, что солвер уже установлен (см. [Установка](#)) и активирована лицензия (см. [Лицензии Солвера ArhiPlex 2.6](#)). Развёрнутые примеры по каждому языку приведены в разделах API ([Python API](#), [C++ API](#), [C# API](#), [Java API](#), [Julia API](#)).

5.1 Запуск из командной строки

Справка по исполняемому файлу:

```
arhiplex --help
```

Запуск расчёта модели с параметрами по умолчанию:

```
arhiplex path_to_problem_file
```

где `path_to_problem_file` — путь к файлу модели. Поддерживаются форматы **LP** и **MPS** (см. [Форматы файлов](#)).

Типичные варианты запуска:

```
# лимит времени 50 секунд
arhiplex path_to_problem_file --time_limit=50

# запись журнала расчёта в файл
arhiplex --model_file=problem.mps --log_file=problem.log

# выгрузка решения и «горячий» старт из файла решения
arhiplex --model_file=problem.mps \
    --hotstart_solution=init.sol --solution_file=out.sol
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# лимит времени 10 минут и фиксированное зерно генератора
arhiplex --time_limit=600 --random_seed=5 --model_file=problem.mps
```

Полный перечень параметров — в разделе *Параметры расчёта*.

5.2 Примеры входных файлов (LP/MPS)

Небольшие модели для быстрой проверки установки. Синтаксис форматов подробно описан в разделе *Форматы файлов*.

Линейная задача (LP). Файл `toy_lp.lp`:

```
Minimize
  obj: x + 2 y
Subject To
  c1: x + y >= 3
  c2: x - y = 1
Bounds
  x free
  0 <= y <= 10
End
```

Запуск:

```
arhiplex --model_file=toy_lp.lp --solution_file=toy_lp.sol
```

Смешанно-целочисленная задача (MIP). Файл `toy_mip.lp`:

```
Maximize
  obj: 3 x + 2 y
Subject To
  c1: x + y <= 1
Binary
  x y
End
```

Запуск (с логом и лимитом времени):

```
arhiplex --model_file=toy_mip.lp --log_file=toy_mip.log --time_limit=30
```

Квадратичная задача (QP/QCP). Файл `toy_qp.lp` (квадратичная цель и квадратичное ограничение):

```
Minimize
  obj: a + b + [a^2 + 4 a * b + 7 b^2]/2
Subject To
  c1: a + b >= 10
  c2: [a^2 + b^2] <= 100
Bounds
  0 <= a <= 100
  0 <= b <= 100
End
```

Запуск:

```
arhiplex --model_file=toy_qp.lp --solution_file=toy_qp.sol
```

5.3 Быстрый старт на Python

Построение небольшой модели, запуск расчёта и чтение результата:

```
import arhiplexpy as ap

model = ap.Model()

# переменные: (нижняя граница, верхняя граница, коэфф. в ЦФ, тип, имя)
x = model.add_variable(0.0, ap.kArhiplexInf, 1.0, ap.variable_type.continuous, "x")
y = model.add_variable(0.0, 10.0, 2.0, ap.variable_type.continuous, "y")

# ограничения
model.add_constraint(x + y >= 3, "c1")
model.add_constraint(x - y == 1, "c2")

# направление оптимизации и запуск
model.set_objective_sense(ap.objective_sense.minimize)
result = model.solve()

# интерпретация результата
if result.get_solve_result() == ap.solve_result.success:
    print("Статус:", result.get_solution_status())
    print("Цель:", result.get_objective_value())
    print("x =", result.get_variable_value(x))
    print("y =", result.get_variable_value(y))
```

Предупреждение: Прежде чем читать значения переменных, проверьте статус запуска и статус решения (см. *Интерпретация результата*, *Статусы и коды возврата*).

Запуск расчёта из файла модели:

```
1 import arhiplexpy
2
3
4 if __name__ == "__main__":
5     model = arhiplexpy.Model()
6
7     model.read("blp-ic98.mps")
8     model.set_log_file("arhiplex.log")
9
10    result = model.solve()
11
12    result.write_solution("blp-ic98.sol")
```

Модель читается из файла методом `read()`, лог направляется в файл, решение записывается методом расчёта. Развёрнутый разбор Python API — в разделе *Python API*.

5.4 Что дальше

- *Основные понятия* — базовые понятия: разрыв оптимальности, допуски, интерпретация результата.
- *Параметры расчёта* — управление расчётом.
- *Импорт и экспорт моделей, Создание модели с использованием матриц* — построение и обмен моделями.
- *CP-решатель (ArhiPlex CP)* — задачи программирования в ограничениях (ArhiPlex CP).

В этом разделе собраны понятия, общие для всех классов задач и всех программных интерфейсов ArhiPlex: как устроен процесс решения, что такое разрыв оптимальности и допуски, как правильно интерпретировать результат расчёта. Определения терминов вынесены в *Глоссарий*.

6.1 Как выполняется расчёт

Работа с ArhiPlex укладывается в одну и ту же последовательность независимо от языка:

1. **Построение модели** — переменные, ограничения и целевая функция. Модель можно построить программно или прочитать из файла (*Импорт и экспорт моделей*, *Форматы файлов*).
2. **Настройка параметров** расчёта (*Параметры расчёта*): лимит времени, точности, объём вывода.
3. **Запуск решения** — метод `solve()`.
4. **Проверка статусов и чтение результата** — сначала статус запуска, затем статус решения, затем значения переменных (см. *Интерпретация результата*).

Класс задачи (LP, MIP, QP, QCQP, NLP) определяется автоматически по составу модели: наличию целочисленных переменных, квадратичных или нелинейных выражений. Отдельный класс — задачи программирования в ограничениях, решаемые *ArhiPlex CP* (*CP-решатель (ArhiPlex CP)*).

6.2 Разрыв оптимальности (gap)

Для задач с целочисленными переменными (*MIP*) решатель поддерживает две величины:

- **лучшее решение** (*upper bound*, *ub* при минимизации — лучшее найденное допустимое значение целевой функции);
- **оценка оптимума** (*lower bound*, *lb* — граница, ниже которой оптимум быть не может).

Оптимум заключён между ними. *Разрыв оптимальности* показывает, насколько они разошлись:

- **относительный разрыв** — $|ub - lb| / |ub|$;
- **абсолютный разрыв** — $|ub - lb|$.

По мере расчёта разрыв сокращается. Расчёт завершается со статусом *optimal*, как только разрыв опускается до порога *mip_rel_gap* (относительный) **или** *mip_abs_gap* (абсолютный). Нулевой разрыв означает доказанную оптимальность.

Разрыв — основной рычаг компромисса «время/качество»: увеличив *mip_rel_gap*, можно завершить расчёт раньше, приняв решение, гарантированно отстоящее от оптимума не более чем на заданную долю. Текущий разрыв виден в столбце «Точность» табличного лога (*Лог вычислений*).

Примечание: Для непрерывных задач (LP/QP) понятие разрыва оптимальности не применяется: решение либо оптимально, либо задача недостижима/неограничена.

6.3 Допуски (tolerances)

Расчёты выполняются в арифметике с плавающей точкой, поэтому равенства и целочисленность проверяются не точно, а в пределах *допусков* — малых порогов:

- **допуск достижимости** (*primal_feasibility_tolerance*) — насколько строка ограничения или граница переменной может быть нарушена, чтобы решение считалось допустимым;
- **допуск целочисленности** (*mip_feasibility_tolerance*) — насколько значение целочисленной переменной может отличаться от целого.

Ужесточение допусков повышает строгость решения, но может замедлить расчёт или сделать модель формально недостижимой из-за противоречий на уровне округления. Ослабление ускоряет расчёт ценой точности. В большинстве задач значения по умолчанию менять не требуется.

6.4 Начальное решение

Для *MIP*-задач решателю можно передать готовое допустимое *начальное решение*. Это не меняет оптимум, но часто ускоряет сходимость: решатель сразу получает хорошую верхнюю границу и быстрее сокращает *разрыв*. Подробности и примеры — в разделе *Начальные решения*.

6.5 Двойственные оценки

Для непрерывных задач (LP/QP) помимо значений переменных доступны *двойственные решения* — теневые цены ограничений и приведённые стоимости переменных. Они показывают чувствительность оптимума к изменению правых частей и границ. Для задач с целочисленными переменными двойственные оценки не определены. См. *Двойственные решения*.

6.6 Диагностика недостижимости (IIS)

Если модель получила статус `infeasible`, полезно понять, какие именно ограничения противоречат друг другу. Для этого служит *IIS* — несократимое недостижимое множество: минимальный набор ограничений и границ, который сам по себе недостижим, но становится достижимым при удалении любого элемента. Вычисление IIS включается параметром `compute_iis`. Полное описание и примеры — в разделе *Анализ недостижимости модели*.

6.7 Интерпретация результата

Корректный разбор результата расчёта выполняется по шагам:

1. **Статус запуска** (*Статус запуска расчёта*). Если он не `success` — решение отсутствует, обращаться к значениям переменных нельзя; смотрите причину и лог. При `fail` также проверяйте `solution_status`: значение `imprecise` сообщает о превышении допусков найденным кандидатом.
2. **Статус решения** (*Статус решения*):
 - `optimal` — решение оптимально, можно использовать;
 - `feasible` — решение допустимо, но не доказанно оптимально (типично при достижении лимита времени); при необходимости продолжите расчёт;
 - `infeasible` — модель недостижима, перейдите к диагностике *IIS*;
 - `unbounded / infeasible_or_unbounded` — проверьте, не пропущены ли ограничения или границы переменных;
 - `imprecise` — кандидат не прошёл проверку заданных допусков и другого допустимого решения нет; `solve_result=fail`, значения переменных использовать нельзя. Это не доказательство недостижимости модели.
3. **Показатели качества** — значение целевой функции, *разрыв* (для МIP), время расчёта.
4. **Значения переменных и двойственные оценки** — только после того, как статусы подтвердили наличие решения.

Полный перечень статусов с рекомендациями приведён в разделе *Статусы и коды возврата*.

В этом разделе собраны основные термины, используемые в руководстве. Термины приведены в единой русской терминологии; в скобках указан устоявшийся английский эквивалент, встречающийся в API, логике расчёта и файлах моделей.

ArhiPlex CP

Решатель задач программирования в ограничениях (*Constraint Programming*): логических, комбинаторных задач и задач планирования. См. *CP-решатель (ArhiPlex CP)*.

IIS

Несократимое недостижимое множество (*Irreducible Infeasible Set*) — минимальный набор ограничений и границ, делающий модель недостижимой. Инструмент диагностики недостижимости. См. *Анализ недостижимости модели*.

LP

Линейное программирование (*Linear Programming*) — задача с линейной целевой функцией и линейными ограничениями над непрерывными переменными.

LP-формат

Текстовый формат описания моделей (не путать с классом задач *LP*), более наглядный, чем *MPS*. См. *Форматы файлов*.

lp_method

Параметр выбора метода решения непрерывной задачи. См. *lp_method*.

MIP

Смешанно-целочисленное (линейное) программирование (*Mixed-Integer Programming*, MILP) — задача, в которой часть переменных ограничена целыми значениями.

mip_abs_gap

Параметр абсолютного *разрыва оптимальности*. См. *mip_abs_gap*.

mip_rel_gap

Параметр относительного *разрыва оптимальности* — критерий остановки для *MIP*. См. *mip_rel_gap*.

MPS

Текстовый формат описания моделей оптимизации. Поддерживается на чтение и запись. См. *Форматы файлов*.

NLP

Нелинейное программирование (*Nonlinear Programming*) — задача с произвольными нелинейными выражениями в цели или ограничениях. Вариант с целочисленными переменными обозначается MINLP.

presolve

Параметр, управляющий *предобработкой* модели. См. *presolve*.

QCQP

Квадратичное программирование с квадратичными ограничениями (*Quadratically Constrained Quadratic Programming*).

QP

Квадратичное программирование (*Quadratic Programming*) — задача с квадратичной целевой функцией и линейными ограничениями.

time_limit

Параметр ограничения времени расчёта в секундах. См. *time_limit*.

Двойственное решение

Двойственные оценки (теневые цены) ограничений и приведённые стоимости переменных, доступные для непрерывных задач (*LP/QP*). См. *Двойственные решения*.

Допуски

Численные пороги, в пределах которых решение считается удовлетворяющим условиям задачи (*tolerances*). Термин используется вместо «толеранты». Подробнее — в разделе *Допуски (tolerances)*.

Достижимость

Свойство модели иметь хотя бы одно решение, удовлетворяющее всем ограничениям (*feasibility*). Модель без таких решений называется недостижимой (*infeasible*).

Модель

Формальная постановка задачи оптимизации: набор *переменных*, *ограничений* и *целевой функции*. В API модель представлена классом `Model` (`arhiplexpy.Model` в Python, `arhiplex::Model` в C++ и т. д.).

Начальное решение

Готовое допустимое решение, передаваемое решателю вместе с моделью для ускорения расчёта *MIP* (*MIP start*, «горячий старт»). Термин используется вместо «warm start». См. *Начальные решения*.

Ограничение

Условие, которому должно удовлетворять решение: линейное или квадратичное равенство либо неравенство над переменными модели.

Ограниченность

Свойство задачи иметь конечный оптимум. Если целевую функцию можно неограниченно улучшать, задача называется неограниченной (*unbounded*).

Переменная

Неизвестная величина модели, значение которой подбирает решатель. Переменные бывают непрерывными, целочисленными и бинарными (см. *тип переменной*).

Предобработка

Автоматическое упрощение модели перед основным расчётом: удаление избыточных ограниче-

ний, фиксация переменных, уплотнение матрицы. Управляется параметром *presolve*. Термин используется вместо «пресолв».

Разрыв оптимальности

Разница между лучшим найденным допустимым решением и оценкой оптимума (*gap*). Используется как критерий остановки для *MIP*-задач. Подробнее — в разделе *Разрыв оптимальности (gap)*. См. *mip_rel_gap*, *mip_abs_gap*.

Решатель

Программный компонент, который выполняет расчёт математической модели и возвращает решение. В ArhiPlex расчёт может выполняться разными решателями (симплекс-метод, метод внутренней точки и др.); выбор решателя управляется параметрами *lp_method* и *solver*. В руководстве «решатель» и «солвер» используются как синонимы.

Тип переменной

Характеристика переменной, определяющая множество её допустимых значений: непрерывная (любое вещественное число в границах), целочисленная (целые значения) или бинарная (0 или 1).

Удалённый расчёт

Выполнение расчёта на сервере ArhiCloud с передачей модели по сети, а не локально на машине пользователя.

Целевая функция

Функция переменных, значение которой решатель минимизирует или максимизирует. Направление оптимизации задаётся смыслом цели (*objective_sense*: минимизация или максимизация).

Параметры расчёта

Параметры управляют поведением решателя: ограничением ресурсов, точностью решения, объёмом вывода и режимом удалённого расчёта. В этом разделе описан универсальный набор параметров, применимый независимо от того, каким решателем выполняется расчёт.

Примечание: Перечисленные параметры действуют при любом выбранном решателе. Помимо них в поставке могут присутствовать дополнительные внутренние настройки; они не входят в поддерживаемый пользовательский интерфейс, их состав и поведение могут меняться между версиями и в этом руководстве не описаны.

8.1 Способы задания параметров

Параметр задаётся одним из следующих способов.

Через API модели. Для каждого типа значения предусмотрен свой метод. В Python:

```
import arhiplexpy as ap

model = ap.Model()
model.set_dbl_param("time_limit", 60.0)      # вещественные
model.set_int_param("random_seed", 5)        # целые
model.set_string_param("lp_method", "dual")   # строковые
model.set_bool_param("log_to_console", False) # логические
```

В C++ используются одноимённые методы фасада `arhiplex::Model`: `SetDblParam`, `SetIntParam`, `SetStringParam`, `SetBoolParam` (аналоги см. в соответствующих разделах *C#*, *Java*, *Julia*).

Через командную строку исполняемого файла `arhiplex`:

```
arhiplex --model_file=problem.mps --time_limit=60 --random_seed=5
```

Через файл настроек. Путь к файлу параметров передаётся исполняемому файлу опцией `--options_file`. Формат — по одному параметру в строке в виде **имя = значение**.

Получить актуальный список параметров и их описания в конкретной сборке можно командой:

```
arhiplex --help
```

8.2 Сводная таблица

Таблица 8.1: Универсальные параметры расчёта

Параметр	Тип	По умолчанию	Назначение
<i>time_limit</i>	веществ.	без лимита	Лимит времени расчёта, с.
<i>threads</i>	целый	0 (авто)	Число потоков расчёта.
<i>random_seed</i>	целый	1	Инициализация генератора случайных чисел.
<i>presolve</i>	строка	on	Режим предобработки модели.
<i>lp_method</i>	строка	auto	Метод решения непрерывной задачи.
<i>presolve_conflict_localization</i>	логич.	true	Локализация конфликтов при предобработке.
<i>mip_rel_gap</i>	веществ.	1e-4	Относительный разрыв оптимальности (MIP).
<i>mip_abs_gap</i>	веществ.	1e-6	Абсолютный разрыв оптимальности (MIP).
<i>mip_feasibility_tolerance</i>	веществ.	1e-6	Допуск целочисленности для MIP.
<i>primal_feasibility_tolerance</i>	веществ.	1e-6	Допуск выполнения ограничений и границ переменных.
<i>compute_iis</i>	логич.	false	Вычислять IIS при недостижимости.
<i>log_to_console</i>	логич.	true	Вывод лога в консоль.
<i>log_display_systime</i>	логич.	false	Показывать системное время в логе.
<i>write_zeroes_to_solution</i>	логич.	true	Записывать нулевые значения в файл решения.
<i>remote_timeout</i>	целый	300	Таймаут сервера удалённого расчёта, с.
<i>http_proxy</i>	строка	(пусто)	HTTP-проxy для удалённого расчёта.

8.3 Ограничение ресурсов

8.3.1 time_limit

Тип

вещественный

По умолчанию

без лимита (расчёт не ограничен по времени)

Диапазон

[0, +inf)

Максимальное время расчёта в секундах. По истечении лимита решатель завершает работу и возвращает лучшее найденное решение (если оно есть); модель при этом получает статус *достижения лимита времени*. Параметр применим и к удалённому расчёту.

```
model.set_dbl_param("time_limit", 600.0) # 10 минут
```

8.3.2 threads

Тип

целый

По умолчанию

0

Диапазон

[0, макс]

Число потоков, используемых для расчёта. Значение 0 означает автоматический выбор по числу доступных ядер хоста. Один расчёт выполняется в общей памяти в несколько потоков.

Примечание: Не следует путать многопоточность одного расчёта с многопроцессным режимом (см. *Многопроцессный режим и параллельный запуск*): это независимые механизмы.

8.3.3 random_seed

Тип

целый

По умолчанию

1

Диапазон

[0, макс]

Начальное значение генератора случайных чисел. Влияет на решения, в которых используется случайность (например, порядок обхода ветвлений в МПР). Фиксация значения делает расчёт воспроизводимым; изменение может привести к другому, но столь же корректному решению.

8.4 Управление алгоритмом

8.4.1 presolve

Тип

строка

По умолчанию

on

Допустимые значения

off, on, weak, strong

Режим *предобработки* модели — автоматического упрощения задачи перед основным расчётом. Значения **weak** и **strong** регулируют интенсивность упрощений; **off** полностью отключает предобработку. В большинстве случаев рекомендуется оставлять значение **on**.

8.4.2 lp_method

Тип

строка

По умолчанию

auto

Допустимые значения

auto, primal, dual, barrier

Метод решения непрерывной (LP/QP) задачи: прямой симплекс-метод (**primal**), двойственный симплекс-метод (**dual**), метод внутренней точки/барьерный (**barrier**) или автоматический выбор (**auto**). Для MIP-задач параметр определяет метод решения непрерывных подзадач.

8.4.3 presolve_conflict_localization

Тип

логический

По умолчанию

true

Включает локализацию конфликтов при предобработке недостижимых моделей. Помогает точнее указать источник недостижимости. Отключение может незначительно ускорить предобработку ценой менее информативной диагностики.

8.5 Точности и разрывы (MIP)

Общие сведения о разрыве оптимальности и допусках приведены в разделах *Разрыв оптимальности (gap)* и *Допуски (tolerances)*.

8.5.1 mip_rel_gap

Тип	вещественный
По умолчанию	1e-4
Диапазон	[0, +inf)

Относительный *разрыв оптимальности* для МПР: $|ub - lb| / |ub|$, где ub — лучшее найденное решение, lb — оценка оптимума. Как только разрыв опускается до указанного значения, расчёт считается завершённым и возвращается решение. Значение 1e-4 соответствует гарантии оптимальности в пределах 0,01 %.

8.5.2 mip_abs_gap

Тип	вещественный
По умолчанию	1e-6
Диапазон	[0, +inf)

Абсолютный *разрыв оптимальности* для МПР: $|ub - lb|$. Расчёт завершается при достижении относительного **или** абсолютного разрыва. Абсолютный критерий удобен для задач, где значение целевой функции близко к нулю и относительный разрыв плохо определён.

8.5.3 mip_feasibility_tolerance

Тип	вещественный
По умолчанию	1e-6
Диапазон	[1e-9, +inf)

Допуск отклонения целочисленной переменной от ближайшего целого числа. За выполнение строк ограничений и границ переменных отвечает отдельный параметр *primal_feasibility_tolerance*. Изменение одного допуски не изменяет другой.

8.5.4 primal_feasibility_tolerance

Тип

вещественный

По умолчанию

1e-6

Диапазон общего API

[1e-10, +inf)

Абсолютный допуск нарушения строк ограничений и границ переменных, в том числе при проверке решения МПР. Допустимый диапазон конкретного вычислительного модуля может быть уже диапазона общего API. Если модуль не может применить заданное значение, его результат не принимается с молча ослабленным допуском.

Например, для проверки целочисленности, строк и границ с допуском 1e-9 задайте оба параметра:

```
model.set_dbl_param("mip_feasibility_tolerance", 1e-9)
model.set_dbl_param("primal_feasibility_tolerance", 1e-9)
```

Уменьшение допусков повышает строгость проверки и может замедлить расчёт. Параметр `mip_rel_gap` регулирует качество целевой функции и не заменяет проверку допустимости. Если кандидат нарушает допуски и другого допустимого решения нет, возвращается *imprecise*.

8.6 Диагностика

8.6.1 compute_iis

Тип

логический

По умолчанию

false

Если модель оказалась недостижимой, при включённом параметре автоматически вычисляется *IIS* — несократимое недостижимое множество ограничений и границ. См. *Анализ недостижимости модели*.

8.7 Логирование

8.7.1 log_to_console

Тип

логический

По умолчанию

true

Управляет выводом лога расчёта в консоль. Установите `false`, чтобы отключить вывод (например, при пакетных запусках). Структуру лога см. в разделе *Лог вычислений*.

8.7.2 log_display_systime

Тип

логический

По умолчанию

false

Добавляет в строки лога отметку системного времени. Полезно для сопоставления хода расчёта с внешними событиями.

8.8 Вывод решения

8.8.1 write_zeroes_to_solution

Тип

логический

По умолчанию

true

Записывать ли в файл решения переменные с нулевым значением. Отключение уменьшает размер файла решения для задач с большим числом нулевых переменных.

8.9 Удалённый расчёт (ArhiCloud)

8.9.1 remote_timeout

Тип

целый

По умолчанию

300

Диапазон

[10, 600]

Максимальное время ожидания ответа сервера удалённых вычислений в секундах. Относится к сетевому взаимодействию с сервером, а не к длительности самого расчёта (её ограничивает *time_limit*).

8.9.2 http_proxy

Тип

строка

По умолчанию

(пусто)

Адрес HTTP-прокси для удалённого расчёта в закрытом сетевом контуре. Пустое значение означает прямое соединение.

8.10 Многопроцессный режим и параллельный запуск

Для МПР-задач ArhiPlex может использовать многопроцессный режим: один расчёт распределяется на несколько рабочих процессов по числу ядер хоста. Числом рабочих процессов на расчёт управляет параметр `multiprocess_max_process_count` (целое; 0 — автоматически, по числу ядер хоста, но не более 16).

Если на одном хосте одновременно выполняется несколько независимых расчётов (внешняя параллелизация), каждый из них по умолчанию претендует на все ядра хоста и не учитывает другие расчёты. Суммарный запрос рабочих процессов может превысить число ядер и системные лимиты хоста (число процессов, объём памяти, файловые дескрипторы), что приводит к сбою запуска части расчётов.

Рекомендация: произведение «числа параллельных задач» и «числа рабочих процессов на задачу» должно укладываться в число ядер и системные лимиты хоста. Основной рычаг — ограничить `multiprocess_max_process_count` для каждого расчёта. Например, при нескольких одновременных расчётах на одном хосте задайте `multiprocess_max_process_count = 2`.

Статусы и коды возврата

После расчёта ArhiPlex сообщает о его исходе двумя независимыми характеристиками:

- **статус запуска расчёта** — удалось ли вообще выполнить процесс решения;
- **статус решения** — что известно о полученном решении (оптимально, допустимо, недостижимо и т. д.).

Всегда сначала проверяйте статус запуска, затем — статус решения, и только после этого читайте значения переменных. Общая логика интерпретации результата описана в разделе *Интерпретация результата*.

9.1 Статус запуска расчёта

Возвращается методом `get_solve_result()` результата (перечисление `solve_result`). Сообщает, получен ли результат, который можно интерпретировать как допустимое решение или вывод о модели. Значение `fail` также используется для численно неточного кандидата; причину уточняет `get_solution_status()`.

Значение	Смысл	Что делать
<code>success</code>	Получено допустимое решение либо вывод о недостижимости или неограниченности модели. Само значение <code>success</code> не означает, что доступны значения переменных.	Перейти к проверке <i>статуса решения</i> .
<code>fail</code>	Результат, пригодный для использования, не получен.	Проверить <code>solution_status</code> : он отличает численно неточный кандидат (<code>imprecise</code>) от отсутствующего результата (<code>invalid_solution</code>).
<code>remote_invalid_api_key</code>	Недействительный ключ для удалённого расчёта.	Проверить ключ доступа к ArhiCloud (см. <i>Лицензии Солвера ArhiPlex 2.6</i>).
<code>remote_api_key_not_set</code>	Не задана переменная окружения с ключом API.	Задать ключ доступа перед удалённым расчётом.
<code>remote_time_amount_is_over</code>	Исчерпан доступный лимит времени удалённых расчётов.	Обратиться к условиям лицензии/тарифа.
<code>remote_time_per_calc_violated</code>	Значение <i>time_limit</i> превышает ограничение лицензии на один удалённый расчёт.	Уменьшить <code>time_limit</code> до допустимого значения.
<code>remote_fail</code>	Удалённый расчёт завершился ошибкой.	Проверить сеть и параметры удалённого расчёта, см. <i>Лог вычислений</i> .

9.2 Статус решения

Возвращается методом `get_solution_status()` результата (перечисление `solution_status`). Проверьте его и при `success`, и при `fail`: сочетание `fail / imprecise` означает, что кандидат не прошёл проверку допустимости. Значения переменных доступны только при `success` в сочетании с `optimal` или `feasible`.

Значение	Смысл	Что делать
<code>optimal</code>	Найдено оптимальное решение (в пределах заданных <i>допусков</i> и <i>разрыва</i>).	Использовать решение; при необходимости проверить значение целевой функции.
<code>feasible</code>	Найдено допустимое, но не доказанно оптимальное решение (например, при достижении <i>лимита времени</i>).	Решение можно использовать; для поиска более качественного решения увеличить <code>time_limit</code> . Ослабление <code>mip_rel_gap</code> снижает требование к качеству, а не улучшает найденную точку.
<code>infeasible</code>	Модель недостижима: допустимого решения не существует.	Проверить ограничения и границы; выполнить диагностику <i>IIS</i> (<i>Анализ недостижимости модели</i>).
<code>unbounded</code>	Модель неограничена: целевую функцию можно бесконечно улучшать.	Проверить, не пропущены ли ограничения или границы переменных.
<code>infeasible_or_unbounded</code>	Модель недостижима либо неограничена (точный случай не определён).	Проверить постановку; при необходимости зафиксировать границы и повторить расчёт для уточнения.
<code>imprecise</code>	Найден кандидат с конечной целевой функцией, но проверка выявила превышение заданных допусков, и другого допустимого решения нет. Это не допустимое решение и не доказательство недостижимости модели; <code>solve_result=fail</code> .	Не использовать значения переменных; проверить численное масштабирование модели и повторить расчёт с другим вычислительным модулем или настройками.
<code>invalid_solution</code>	Значение не определено (решение отсутствует).	Не обращаться к значениям переменных; проверить статус запуска и лог.

9.3 Текстовые статусы в логe и отчёте

В консольном логe и итоговом отчёте статус выводится текстом. Ниже приведено соответствие текстовых статусов их смыслу.

Текст в отчёте	Смысл
Оптимально	Найдено оптимальное решение.
Достижимо	Найдено допустимое решение (см. <code>feasible</code> выше).
Недостижимо	Модель не имеет допустимого решения.
Неограничено	Целевая функция не ограничена.
Первичная недостижимость или неограниченность	Модель недостижима либо неограничена.
Найден численно неточный кандидат: нарушения превышают заданные допуски	Конечная целевая функция получена, но кандидат не прошёл проверку допустимости ArhiPlex.
Пусто	Модель не содержит переменных/ограничений после предобработки.

При параллельном расчёте отклонённый кандидат одного вычислительного модуля не отменяет допустимое решение другого. ArhiPlex выбирает лучшее из проверенных допустимых решений. Наличие конечной целевой функции в промежуточном прогрессе само по себе не подтверждает допустимость.

При `imprecise` значения переменных и файл решения не выдаются; целевая функция возвращается как `+inf` для минимизации или `-inf` для максимизации, разрыв — `inf`. Эти значения обозначают отсутствие принятого решения, а не неограниченность модели. Числовое значение `solution_status.imprecise` — 6; значения ранее существовавших элементов перечисления сохранены.

Причина остановки	Смысл
Лимит времени исчерпан	Достигнут <i>time_limit</i> . Возвращается лучшее найденное решение (статус решения — <code>feasible</code> , если оно есть).
Лимит количества итераций исчерпан	Достигнут лимит итераций. Возвращается текущее лучшее решение.

9.4 Статусы вычисления IIS

При вычислении *IIS* (*Анализ недостижимости модели*) возвращаются два статуса.

Статус процесса (`compute_iis_result`, метод `get_compute_iis_result()`):

Значение	Смысл
<code>success</code>	Процесс завершён: найдено несократимое недостижимое множество либо установлена достижимость модели.
<code>fail</code>	Процесс не удался: IIS не найдено и достижимость не установлена.

Статус модели (`model_status`, метод `get_model_status()`):

Значение	Смысл
<code>feasible</code>	Модель достижима — IIS отсутствует.
<code>infeasible</code>	Модель недостижима — найдено несократимое недостижимое множество.
<code>unknown</code>	Статус не определён.

9.5 Статус базиса переменных и ограничений

Для непрерывных задач (LP/QP) при запросе базисного решения каждой переменной и ограничению сопоставляется статус базиса. Он показывает положение переменной относительно её границ в оптимальном базисе и используется, в частности, при анализе чувствительности и «горячем» старте.

Статус	Смысл
на нижней границе	Небазисная переменная зафиксирована на нижней границе.
базисная	Переменная входит в базис.
на верхней границе	Небазисная переменная зафиксирована на верхней границе.
ноль (свободная)	Свободная небазисная переменная со значением 0.
небазисная	Небазисная переменная без привязки к конкретной границе.

См. также:

Интерпретация результата — как связать статусы между собой при разборе результата; *Анализ недостижимости модели* — что делать со статусом `infeasible`.

В **ArhiPlex** поддерживаются три основных формата файла.

Форматы *.lp*, *.mps* — форматы записи моделей в файл для последующего хранения и использования. Расширение *.ilp* распознаётся ArhiPlex как файл в формате **LP**.

Формат *.sol* — файл решения. Используется для записи решения задачи в файл и использования его далее, например, в качестве начального решения в других задачах.

10.1 LP

LP формат предоставляет возможность ввода задачи в естественной алгебраической формулировке. Задачу можно изменить, сохранить и повторно прочитать в ArhiPlex.

Формат **LP** может представлять не только линейное программирование (**LP**), но и смешанное целочисленное программирование (**MIP**).

Формат **LP** предоставляется в качестве входной альтернативы формату файлов **MPS**. Файл формата **LP** проще создать, чем файл **MPS**, если ваша задача уже существует в алгебраическом формате или если у вас есть приложение, которое создает файл задачи.

10.1.1 Длина строки

Максимальная длина любого имени — 255 символов. Максимальная длина любой строки — 560 символов.

10.1.2 Комментарии

Все, что следует за обратной косой чертой \, является комментарием и игнорируется, пока не встретится перенос строки. Пустые строки также игнорируются. Пустые строки и строки комментариев могут быть размещены в любом месте любое число раз.

10.1.3 Пробелы

Пробелы между символами не имеют значения, при чтении файла они пропускаются. Пробелы не допускаются в ключевых словах, используемых для введения нового раздела, таких как *MAX*, *MIN*, *ST* или *BOUNDS*. Также ключевые слова должны быть отделены пробелом от остальной части файла и должны находиться в начале строки.

Пропуск пробелов может привести к тому, что солвер неправильно интерпретирует запись, например следующую:

```
x1 x2 = 0
```

Если пользователь собирался использовать этот пример для задания нелинейного ограничения, солвер по ошибке распознает его как ограничение, устанавливающее значение переменной *x1x2* равным нулю.

Чтобы указать квадратичное ограничение в этом разделе, используйте явное обозначение для умножения и возведения в степень.

10.1.4 Направление оптимизации

Для указания направления оптимизации формулировка задачи должна начинаться со слова *MINIMIZE* или *MAXIMIZE*, *MINIMUM* или *MAXIMUM*, или аббревиатуры *MIN* или *MAX*, в любой комбинации символов верхнего и нижнего регистра. Эти ключевые слова начинают раздел целевой функции.

10.1.5 Наименования переменных

Переменные могут иметь любое имя при условии, что имя не превышает 255 символов, все из которых должны быть буквенно-цифровыми (*a-z*, *A-Z*, *0-9*) или одним из следующих символов: *! « # \$ % & () , . : ; ? @ _ „ „ { } ~*. Более длинные имена усекаются до 255 символов. Имя переменной не может начинаться с цифры или точки.

Букву *E* или *e*, отдельно или за которой следуют другие допустимые символы, следует избегать, так как это обозначение зарезервировано для экспоненциальных записей. Таким образом, переменные не могут называться *e9*, *E-24*, *E8cats* или другими именами, которые можно интерпретировать как показатель степени. Даже имена переменных, такие как *eels* или *example* могут вызвать ошибку чтения, в зависимости от их размещения в строке файла.

Совет: Имена переменных начинаются с буквы (кроме *E* или *e*) или с символа подчеркивания.

10.1.6 Целевая функция

Определение целевой функции должно следовать после слов *MINIMIZE* или *MAXIMIZE*. Целевую функцию можно ввести в несколько строк, если ни одна переменная, константа не разделены символом новой строки. Например, целевую функцию $1x1 + 2x2 + 3x3$ можно ввести так:

```
1x1 + 2x2
+ 3x3
```

но не так:

```
1x1 + 2x
2 + 3x3
```

Целевая функция может быть поименована путем ввода имени целевой функции и двоеточия перед самой целевой функцией. Имя целевой функции и двоеточие должны стоять в одной строке. Имена целевых функций должны соответствовать тем же рекомендациям, что и имена переменных (см. правило об именах переменных).

Если имя целевой функции не указано, солвер присваивает имя *obj*.

Смещение целевой функции

Вы можете дополнительно включить смещение в качестве конечного постоянного члена в целевую функцию. Например, следующая строка объявляет целевое смещение *3.1415*:

```
obj: x1 + 2 x2 + 3.1415
```

Точно так же целевое смещение следует за любыми квадратичными выражениями в формате файла **LP** квадратичной программы, например, в следующей строке:

```
obj: x + 2 y + [ x ^2 - 3 x * y ] / 2 + 3.1415
```

10.1.7 Ограничения

Раздел ограничений вводится ключевым словом *subjectto*. Это выражение также может отображаться как *suchthat*, *st*, *S.T.* или *ST.* в любой комбинации символов верхнего и нижнего регистра. Любое из этих выражений должно предшествовать первому ограничению и отделяться от него хотя бы одним пробелом.

Каждое определение ограничения должно начинаться с новой строки. Ограничение может быть названо путем ввода имени и двоеточия перед самим выражением ограничения. Имя ограничения и двоеточие должны стоять в одной строке. Имена ограничений должны соответствовать тем же правилам, что и имена переменных (см. правило об именах переменных). Если имена ограничений не указаны, солвер присваивает имена *c1*, *c2*, *c3*, и т.д.

Ограничения вводятся так же, как и целевая функция; однако за ограничением должно следовать указание его направления и коэффициент (правая часть). Правая часть должна быть напечатана в той же строке, что и направление ограничения. Приемлемыми направлениями ограничений являются $<$, $<=$, $=<$, $>$, $>=$, $=>$ и $=$. Они интерпретируются как $<=$, $<=$, $<=$, $>=$, $>=$, $>=$ и $=$ соответственно.

Например, вот именованное ограничение:

```
time: x1 + x2 <= 10
```

Также, разрешены квадратичные ограничения. Квадратичные выражения указываются в квадратных скобках `//`, как указано в правиле о квадратичных выражениях. Спецификация квадратичного ограничения отличается от спецификации квадратичной целевой функции одним важным моментом: в квадратичном ограничении члены не делятся на два, то есть они не умножаются на $1/2$, как это должно быть в квадратичной целевой функции.

10.1.8 Границы

Необязательный раздел (*bounds*) следует за разделом ограничений. Ему предшествует слово *bounds* или *bound* любое сочетание строчных и прописных символов.

Каждое определение должно начинаться с новой строки. Формат границы следующий: $lb \leq xn \leq ub$, за исключением следующих случаев. Верхняя и нижняя границы также могут быть введены отдельно, как:

```
lb <= xn
xn <= ub
```

с нижней границей по умолчанию, равной 0 (нулю), и верхней границей по умолчанию, равной $+inf$, которая остается в силе до тех пор, пока граница не будет явно изменена.

Границы, фиксирующие переменную, можно вводить в виде простых равенств. Например, $x5 = 5.6$ эквивалентно $5.6 \leq x5 \leq 5.6$.

Границы $+inf$ (положительная бесконечность) и $-inf$ (отрицательная бесконечность) должны быть введены как слова: *+infinity*, *-infinity*, *+inf*, *inf*.

Переменная с нижней границей, равной отрицательной бесконечности, и верхней границей, равной положительной бесконечности, может быть введена как *free*, в любом сочетании символов верхнего и нижнего регистра, с пробелом, разделяющим имя переменной и слово *free*. Например, $x7 \text{ free}$ эквивалентно $infinity \leq x7 \leq +infinity$.

Последняя введенная граница имеет приоритет над ранее введенными границами.

10.1.9 Конец файла

LP файл должен заканчиваться ключевым словом *end* в любой комбинации символов верхнего и нижнего регистра, в отдельной строке.

10.1.10 Особенности MIP

Целочисленные переменные

Перечень целочисленных переменных приводится в разделе *GENERAL*, а бинарные - в разделе *BINARY*. Данные разделы следуют после раздела *BOUNDS*, либо после раздела ограничений, если *BOUNDS* отсутствует.

Раздел *GENERAL* начинается со слова *GENERAL*, *GENERALS* или *GEN* в любой комбинации символов верхнего и нижнего регистра. В следующей строке или строках должны быть перечислены имена всех переменных, которые должны быть ограничены общими целочисленными значениями, разделенные хотя бы одним пробелом.

Раздел *BINARY* начинается со слова *BINARY*, *BINARIES* или *BIN* в любой комбинации символов верхнего и нижнего регистра. В следующих строках должны быть перечислены имена всех переменных, которые должны быть ограничены двоичными целыми значениями, разделенные хотя бы одним пробелом.

Пример

```
Maximize
  obj: x1 + 2 x2 + 3 x3 + x4
Subject To
  c1: - x1 + x2 + x3 + 10 x4 <= 20
  c2: x1 - 3 x2 + x3 <= 30
  c3: x2 - 3.5 x4 = 0
Bounds
  0 <= x1 <= 40
  2 <= x4 <= 3
General
  x4
End
```

10.1.11 Особенности квадратичных задач

Данный раздел описывает особенности задач с квадратичными членами, то есть квадратичные задачи программирования или с квадратичными ограничениями (**QP**, **QCP**, **MIQP**, **MIQCP**). Квадратичные коэффициенты могут появляться в целевой функции и в ограничениях.

В квадратных скобках (//) указаны алгебраические коэффициенты функции. За квадратными скобками должен следовать знак деления (/), за которым следует число 2. Это означает, что все коэффициенты внутри квадратных скобок будут делиться на 2 при оценке квадратичных членов целевой функции. Все квадратичные коэффициенты должны быть заключены в квадратные скобки.

Для записи произведения недиагональных членов матрицы Q используется символ звездочки (*). Например, $[4x * y]$ указывает, что коэффициенты обоих недиагональных членов Q , соответствующих переменным x и y в модели, равны 2, поскольку $4x * y$ равны $2x * y + 2x * y$. Каждая пара недиагональных членов Q указывается только один раз. Солвер автоматически создает обе недиагональные записи Q .

Диагональные члены в Q (то есть члены с показателем степени 2) обозначаются знаком (^), за которым следует цифра 2. Например, $4x^2$, указывает, что коэффициент диагонального члена Q соответствующий переменной x равен 4.

Пример

В качестве примера приведём задачу минимизации функции при условии $a + b \geq 10$ и $a, b \geq 0$.

Запись в формате **LP** выглядит следующим образом.

```
Minimize
  obj: a + b + [a^2 + 4 a * b + 7 b^2]/2
Subject To
  c1: a + b >= 10
End
```

10.1.12 Нелинейные ограничения

ArhiPlex поддерживает расширения формата, позволяющие задавать *общие ограничения* (general constraints), включая **нелинейные ограничения** (Nonlinear Constraints, часто обозначаются как *NL*).

Идея следующая:

- Нелинейное ограничение задаётся не “алгебраической” строкой, а через **дерево выражения** (expression tree): набор узлов-операций и их связей (родитель/потомок).
- В дереве используются узлы вида *CONSTANT* (константа), *VARIABLE* (переменная), а также операции (*PLUS*, *MULTIPLY*, *SIN*, *EXP*, *POW* и т.д.).
- Такие ограничения располагаются в отдельном разделе **General Constraint Section** и читаются соответствующим расширением формата ArhiPlex.

Пример записи нелинейного ограничения (дерево выражения):

```
GCO: y = NL : ( PLUS , -1 , -1 ) ( SIN , -1 , 0 ) ( MULTIPLY , -1 , 1 )
( CONSTANT , 2.5 , 2 ) ( VARIABLE , x1 , 2 ) ( VARIABLE , x2 , 0 )
```

Важные синтаксические детали:

- Нелинейное ограничение **должно завершаться переводом строки**.
- Все токены (включая запятые) **должны быть разделены пробелами**.

10.2 MPS

Формат **MPS** — это старейший и наиболее широко используемый формат для хранения моделей математического программирования. Существует два широко используемых варианта этого формата. В фиксированном формате различные поля всегда должны начинаться с фиксированных столбцов в файле. Свободный формат очень похож на фиксированный, но поля разделяются пробелами, а не располагаются в определённых столбцах. Одно важное практическое различие между этими двумя форматами заключается в длине имён. В фиксированном формате имена строк и столбцов состоят ровно из 8 символов, а пробелы являются частью имени. В свободном формате имена могут быть произвольно длинными, и в именах не должно быть пробелов.

ArhiPlex поддерживает чтение **MPS** файлов, построенных с общепринятыми правилами. Со спецификацией файлов формата **MPS** можно ознакомиться на следующих ресурсах:

1. [Wikipedia: формат MPS](#)

10.2.1 Нелинейные (и квадратичные) расширения MPS

Помимо базовых секций *NAME*, *ROWS*, *COLUMNS*, *RHS*, *BOUNDS* и *ENDATA*, в спецификации для **MPS** описаны дополнительные секции, которые используются для задач с квадратичными и более общими нелинейными членами:

- **QUADOBJ**: квадратичная часть целевой функции (матрица (Q) квадратичной формы).
- **QCMATRIX**: квадратичные части квадратичных ограничений (QCP); линейные части ограничений при этом остаются в *COLUMNS*, а правые части — в *ROWS/RHS*.
- **General Constraint Section**: может содержать как “простые” general constraints (ABS, MIN/MAX, логические и т.п.), так и **нелинейные ограничения** (Nonlinear Constraints, *NL*), заданные через дерево выражения (expression tree).

Пример нелинейного ограничения (дерево выражения):

```
GCO: y = NL : ( PLUS , -1 , -1 ) ( SIN , -1 , 0 ) ( MULTIPLY , -1 , 1 )
  ( CONSTANT , 2.5 , 2 ) ( VARIABLE , x1 , 2 ) ( VARIABLE , x2 , 0 )
```

Практические замечания

- Строки, начинающиеся с *, трактуются как **комментарии** и игнорируются.
- Все токены (включая запятые) должны быть разделены пробелами; ограничение завершается переводом строки.

10.3 SOL

Сохранение и загрузка решения оптимизационной задачи в **ArhiPlex** осуществляется через файлы формата *sol*. Структура файла *.sol достаточно простая - осуществляется перечисление всех наименований переменных и их значение.

Файл состоит из пар «переменная-значение», каждая из которых находится в отдельной строке. Файл содержит по одной строке для каждой переменной в модели. Ниже приведён простой пример:

```
# Arhiplex Solution File Version 1.1

# Columns 1758
x1757      82.00000000000000
x1756     119.00000000000000
x1754     111.00000000000000
x1753      26.00000000000000
x1751      60.00000000000000
x1749      12.00000000000000
x1746      99.00000000000000
x1745     153.00000000000000
x1744      14.00000000000000
x1739     135.00000000000000
x1738     108.00000000000000
x1736     174.00000000000000
...
```

Для наблюдения и контроля за процессом вычислений Пользователю предоставляется лог решения. В зависимости от класса задачи лог имеет определенную структуру. По умолчанию лог предоставляет актуальный статус по решению задачи каждые 5 секунд, но, иногда, в связи с особенностями процесса решения, такой интервал не выдерживается и статус решения может быть предоставлен с задержкой. Лог состоит из трёх частей - вводной части, табличной части и результирующей части.

11.1 Вводная часть

Общей частью для всех классов задач является вводная часть лога. Рассмотрим на примере.

```
Запуск ArhiPlex 2.5.0.214748
Права на использование (с) 2026 Arhitex
Процессор Intel Xeon Processor (Icelake) [ AVX512, AVX2 ], физических ядер CPU: 26

Параметры запуска:
  mip_rel_gap =      0.00100000
  time_limit =    60.00000000

Оригинальная задача MILP test.lp содержит ограничений: 45907, переменных: 109618
↳ (дробных: 107486, бинарных: 2132, целых: 0)
```

Вводная часть лога выводится перед началом работы алгоритмов и содержит информацию о:

1. Версии солвера, с использованием которой осуществляется расчёт
2. Характеристиках вычислительных мощностей, используемых для вычислений
3. Параметрах расчёта, установленных Пользователем и отличными от значений по умолчанию
4. Информация о характеристиках задачи, а именно, к какому классу задача относится, сколько содержит переменных и ограничений

После начальной части Пользователю предоставляется табличная часть, в зависимости от класса задачи принимающая один из двух форматов.

11.2 Табличная часть

11.2.1 Вещественные задачи

Вещественные задачи выделяются в отдельную категорию задач, решаемых с применением симплекс-методов или методов барьерных функций. В таком случае алгоритмы не оперируют понятиями границ. Лог задач, не содержащих целочисленные переменные состоит из пяти столбцов:

1. **Целевая функция** - значение целевой функции для базиса, определенного на момент вывода строки
2. **Первичная недостижимость** - величина первичной недостижимости, вычисленная для рассматриваемого на момент вывода строки базиса (как абсолютное значение суммы всех нарушений ограничений и границ)
3. **Двойственная недостижимость** - величина недостижимости двойственной задачи, вычисленная для рассматриваемого на момент вывода строки базиса (как абсолютное значение суммы всех нарушений ограничений и границ)
4. **Итераций** - количество итераций симплекс-метода, осуществлённых для поиска решения до момента вывода строки
5. **Время** - общее время решения задачи

Пример лога:

Целевая функция	Первичная недостижимость	Двойственная недостижимость	Итераций	Время
-0	0	0	0	66.7 с
3.97702004e+13	2.87250624e+11	0	11614	71.8 с
29946175431.72	32987111	0	14438	77.9 с
29945007261.65	55960093.25	0	14991	83.8 с
29943489271.89	82265180.625	0	15391	89.9 с
29942900857.33	80380287.25	0	15770	98.1 с
29942690225.26	128997072.75	0	15961	104.0 с
29942516958.06	93684708.25	0	16139	109.9 с
29942402124.64	134066687.25	0	16321	116.2 с
29942193371.52	159955599.25	0	16515	122.7 с

11.2.2 Целочисленные задачи

Если постановка задачи содержит целочисленные или бинарные переменные - вид лога немного изменится в силу специфики применяемых алгоритмов.

Лучшая Граница	Лучшее Решение	Точность	Итераций	Время
15002.918372	30824.103938	51.33%	3064	3.0 с

После вводной части с интервалом в 5 секунд Пользователю предоставляется лог в табличном виде. Основные столбцы:

1. **Лучшая граница** - оценка наилучшего решения по узлам дерева решений на данный момент
2. **Лучшее решение** - наилучшее значение целевой функции для допустимого решения, найденное на текущий момент. Оптимальное решение находится между лучшей границей и лучшим решением
3. **Точность** - относительный разрыв между оценкой лучшей границы и лучшим значением целевой функции, он же *MIP Relative Gap*. При достижении данного показателя значения, указываемого Пользователем в параметре *mip_rel_gap* процесс решения завершается
4. **Итераций** - количество итераций симплекс-метода в рамках процесса решения задач без ограничения на целочисленность
5. **Время** - количество времени в секундах, пройденное алгоритмом после начала вычислений

11.3 Результирующая часть

После завершения решения задачи Пользователю предоставляется информация о полученном решении и итоговом статусе расчёта.

Отчет

Статус	: Оптимально
Первичная граница	: 15004.7006321
Двойственная граница	: 15002.9183725
Отклонение	: 0.0119%
Статус решения	: Достижимо
Время	: 5.1 с.
Ноды(листья)	: 57
LP итераций	: 20773

Импорт и экспорт моделей

Для импорта готовых моделей, созданных с применением других инструментов, и экспорта моделей в файл в **ArhiPlex** имеется функционал работы с текстовыми описаниями моделей.

12.1 Импорт моделей из файла

Для чтения модели из файла и запуска расчёта используется метод *Model.read()*.

```
1 import arhiplexpy
2
3 model = arhiplexpy.Model()
4 model.read("model.lp")
5
6 result = model.solve()
```

12.2 Экспорт модели в файл

ArhiPlex поддерживает два основных расширения файлов для хранения и обработки записи математических моделей – *.lp*, *.mps*. Для записи в файл *.lp* используется метод *Model.write_lp()*, а для записи в файл *.mps* – *Model.write_mps()*. В качестве параметра методу передается название файла (*file_name*).

```
1 import arhiplexpy
2
3 # Создание экземпляра модели
4 model = arhiplexpy.Model()
5
6 # Создание переменных
7 x1 = model.add_variable(0, 40, 1, arhiplexpy.variable_type.continuous, 'x1')
8 x2 = model.add_variable(0, arhiplexpy.kArhiplexInf, 2, arhiplexpy.variable_type.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
↪continuous, 'x2')
9  x3 = model.add_variable(0, arhiplexpy.kArhiplexInf, 3, arhiplexpy.variable_type.
↪continuous, 'x3')
10 x4 = model.add_variable(2, 3, 1, arhiplexpy.variable_type.integer, 'x4')
11
12 # Добавление ограничений
13 model.add_constraint(-x1 + x2 + x3 + 10 * x4 <= 20, 'c1')
14 model.add_constraint(x1 - 3 * x2 + x3 <= 30, 'c2')
15 model.add_constraint(x2 - 3.5 * x4 == 0, 'c3')
16
17 # Направление оптимизации
18 model.set_objective_sense(arhiplexpy.objective_sense.minimize)
19
20 # Сохранить в формат LP, MPS
21 model.write_lp(file_name='model_lp.lp')
22 model.write_mps(file_name='model_mps.mps')
```

Создание модели с использованием матриц

13.1 Введение

ArhiPlex предоставляет расширенные функции **MatrixAPI** в интерфейсе *ArhiPlex Python* (*arhiplexpy*), что заметно упрощает матричное моделирование и делает работу с большими массивами данных удобнее.

13.1.1 Назначение MatrixAPI

- Предназначен для упрощения и ускорения процесса построения моделей оптимизации, особенно в случаях, когда работа ведётся с большими объёмами данных.
- Позволяет эффективно использовать матричные и векторные выражения, что делает его удобным инструментом для задач, требующих обработки данных в матричной форме.

13.1.2 Преимущества использования MatrixAPI

- **Ускорение процесса моделирования.** MatrixAPI значительно ускоряет процесс проектирования модели для больших данных, которые уже представлены в бинарном виде или матричной форме.
- **Экономия памяти.** Эффективная работа с разреженными матрицами позволяет минимизировать использование памяти.
- **Интеграция с популярными библиотеками.** MatrixAPI легко интегрируется с такими библиотеками, как NumPy и SciPy, что упрощает работу с внешними данными и расширяет функциональность.
- **Оптимизация читаемости кода.** Использование матричных представлений, когда исходные данные представлены в бинарном виде или матричной форме, делает код более компактным и читаемым, а также облегчает его поддержку и модификацию.

13.1.3 Задачи, которые позволяет описать MatrixAPI

- Построение линейных и квадратичных целевых функций напрямую из матричных представлений (например, Numpy ndarrays или разреженные матрицы SciPy).
- Задание линейных ограничений напрямую из матричных представлений (например, Numpy ndarrays или разреженные матрицы SciPy).

13.1.4 Рекомендации по использованию

- Применять матричное задание в тех случаях, когда данные уже представлены в матричной форме.
- Для обеспечения работы с внешними данными можно использовать с другими библиотеками, такими как NumPy и SciPy, что позволит легко интегрировать результаты оптимизации с другими аналитическими задачами.

Таким образом, **MatrixAPI** представляет собой мощный инструмент для специалистов в области оптимизации, которым требуется высокая производительность и простота работы с матричными данными.

13.2 Примеры использования

13.2.1 Пример построения задачи с линейной ЦФ и линейными ограничениями

Рассмотрим следующую задачу **MILP**:

Vars:

$$x_1 \in \mathbb{R}$$

$$x_2 \in \mathbb{Z}$$

$$x_3 \in \mathbb{B}$$

Objective:

$$3x_1 + 5x_2 + 2x_3 \rightarrow \max_{x_1, x_2, x_3}$$

Subject to:

$$2x_1 + 4x_2 + 3x_3 \leq 10 \tag{13.1}$$

$$x_1 + 3x_2 + 2x_3 \leq 8$$

$$x_1 + x_2 \leq 5$$

$$x_2 + x_3 \leq 3$$

$$x_1 + x_3 \geq 1$$

Bounds:

$$0 \leq x_1 \leq 100.8$$

$$0 \leq x_2 < inf$$

$$0 \leq x_3 \leq 1$$

Ниже приведен код для использования **MatrixAPI**:

```
1 import arhiplexpy as ap
2 import numpy as np
3
```

(continues on next page)

(продолжение с предыдущей страницы)

```

4 def get_Data():
5     """
6     :return: Возвращает данные для создания модели
7     """
8
9     vars_type = [ap.variable_type.continuous, ap.variable_type.binary, ap.variable_type.
10    ↪integer]
11     lower_bounds = 0
12     upper_bounds = [100.8, ap.kArhiplexInf, 1]
13     obj_values = [3, 5, 2]
14     count = 3
15
16     A = np.array([
17         [2, 4, 3]
18         , [1, 3, 2]
19         , [1, 1, 0]
20         , [0, 1, 1]
21         , [1, 0, 1]
22     ])
23
24     b = [10, 8, 5, 3, 1]
25     senses = [
26         ap.constraint_sense.less_equal
27         , ap.constraint_sense.less_equal
28         , ap.constraint_sense.less_equal
29         , ap.constraint_sense.less_equal
30         , ap.constraint_sense.greater_equal]
31
32     return vars_type, lower_bounds, upper_bounds, obj_values, count, A, b, senses
33
34 if __name__ == "__main__":
35
36     # Получение данных для создания модели MILP
37     vars_type, lower_bounds, upper_bounds, obj_values, count, A, b, senses = get_Data()
38
39     # Инициализация модели
40     model = ap.MatrixModel()
41     # Добавление переменных
42     model.add_variables(
43         count=count, lower_bounds=lower_bounds, upper_bounds=upper_bounds, obj_
44    ↪values=obj_values, var_types=vars_type
45     )
46     # Добавление целевой функции
47     model.set_objective(sense=ap.objective_sense.maximize)
48     # Добавление ограничений
49     model.add_constraints(A=A, senses=senses, b=b)
50
51     # Сохранение модели в формате LP
52     model.write_lp('MILP.lp')
53     # Сохранение модели в формате MPS
54     model.write_mps('MILP.mps')

```

(continues on next page)

(продолжение с предыдущей страницы)

```

54
55     # Установка параметров расчета
56     model.set_dbl_param('time_limit', 10)
57     model.set_bool_param('log_to_console', False)
58
59     # Запуск оптимизации
60     result = model.solve()
61
62     # Вывод полученного результата
63     if result.get_solve_result() == ap.solve_result.success:
64         if (result.get_solution_status() == ap.solution_status.optimal or \
65             result.get_solution_status() == ap.solution_status.feasible):
66             print(f'Status problem: {result.get_solve_result()}')
67             print(f'Objective value: {result.get_objective_value()}')
68             print(f'GAP: {result.get_relative_gap()}')
69             print(f'TIME: {result.get_solve_time()} s')
70             print(f'Solution vector: {result.get_solution_vector()}')
71
72             # Сохранение найденного решения
73             result.write_solution('result.sol')
74         else:
75             print(f'Status problem: {result.get_solution_status()}')
76     else:
77         print(f'Status process: {result.get_solve_result()}')

```

Для начала нужно импортировать модуль *arhiplexpy*:

```
import arhiplexpy as ap
```

Далее необходимо создать матричную модель:

```
model = ap.MatrixModel()
```

Далее добавляем переменные. Для добавления используем метод матричной модели *add_variables()*.

```

model.add_variables(
    count=count, lower_bounds=lower_bounds, upper_bounds=upper_bounds, obj_
    ↪ values=obj_values, var_types=vars_type

```

Первый аргумент — количество переменных. Второй и третий аргументы — нижняя и верхняя границы создаваемых переменных соответственно. Четвёртый аргумент — коэффициент при переменной в целевой функции, пятый — тип переменной, последний — имя переменной. Если в значение параметра будет передано одно значение, а не вектор значений, он будет использоваться для всех переменных. Для всех аргументов существуют значения по умолчанию (См. *Python API*).

Далее установим тип задачи с помощью метода модели *set_objective()*:

```
model.set_objective(sense=ap.objective_sense.maximize)
```

Следующим шагом добавляем ограничения. Ограничения добавляются в матричную модель с помощью её метода *add_constraints()*:

```
model.add_constraints(A=A, senses=senses, b=b)
```

Первый аргумент – двумерная матрица линейных ограничений, где количество строк соответствует числу вводимых ограничений. Может иметь тип `scipy.sparse.csr_matrix` или `numpy.ndarray`. Второй аргумент – вектор типов ограничений. И последний – вектор констант, представляющий правые части ограничений.

Для сохранения матричной модели в формате `.lp` или `.mps` используем методы `write_lp()`, `write_mps()` соответственно, в параметры передаем полный путь к файлу:

```
model.write_lp('MILP.lp')
model.write_mps('MILP.mps')
```

Чтобы установить параметры расчёты используем матричный метод `set_dbl_param()`

```
model.set_dbl_param('time_limit', 10)
```

Для запуска расчёта применяем `solve()`, он вернет результат оптимизации, который содержит все необходимые данные после расчёта:

```
result = model.solve()
```

Предупреждение: Прежде чем получить значения переменных после расчёта необходимо убедиться, что расчёт был успешным и проблема как минимум достижима.

```
if result.get_solve_result() == ap.solve_result.success:
    if (result.get_solution_status() == ap.solution_status.optimal or \
        result.get_solution_status() == ap.solution_status.feasible):
```

Для получения вектора решения и значения целевой функции, а также различные характеристики расчёта, можно использовать следующие методы результата решения. Для детального изучения интерфейса на языке Python см. *Python API*:

```
print(f'Status problem: {result.get_solve_result()}')
print(f'Objective value: {result.get_objective_value()}')
print(f'GAP: {result.get_relative_gap()}')
print(f'TIME: {result.get_solve_time()} s')
print(f'Solution vector: {result.get_solution_vector()}')
```

Также можно сохранить решение в текстовый файл с расширением `.sol`, в аргументы передается полный путь до файла.

```
result.write_solution('result.sol')
```

13.2.2 Пример построения задачи с квадратичной ЦФ и линейными ограничениями

Рассмотрим следующую задачу **QP**:

Vars:

$$x_1 \in \mathbb{B}$$

$$x_2 \in \mathbb{Z}$$

$$x_3, x_4 \in \mathbb{R}$$

Objective:

$$x_1^2 + 2x_2^2 + 4x_3^2 + x_4^2 + x_1x_2 + x_3x_4 + 3x_1 + 5.25 \rightarrow \min_{x_1, x_2, x_3, x_4}$$

Subject to:

$$2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 20$$

$$x_1 - x_2 + 2x_3 - 3x_4 \geq 1 \quad (13.2)$$

$$x_1 + 2x_2 + 3x_3 + 4x_4 = 15$$

$$x_1 + x_2 + x_3 + x_4 \geq 5$$

$$x_1 + 3x_2 + 2x_3 + x_4 \leq 12$$

Bounds:

$$0 \leq x_1 \leq 1$$

$$0 \leq x_2 < \inf$$

$$0 \leq x_3 < \inf$$

$$0 \leq x_4 < \inf$$

Ниже приведен код для использования **MatrixAPI**:

```

1 import arhiplexpy as ap
2 import numpy as np
3
4 def get_Data():
5     """
6     :return: Возвращает данные для создания модели
7     """
8
9     vars_type = np.array(
10         [ap.variable_type.binary, ap.variable_type.integer, ap.variable_type.continuous,
11         ↪ ap.variable_type.continuous]
12         , np.int32
13     )
14     lower_bounds = np.array([0, -ap.kArhiplexInf, -ap.kArhiplexInf, -ap.kArhiplexInf])
15     upper_bounds = [1, ap.kArhiplexInf, ap.kArhiplexInf, ap.kArhiplexInf]
16     count = 4
17
18     # Диагональ должна быть удвоена относительно исходной матрицы!
19     P = np.array([
20         [2, 1, 0, 0]
21         , [1, 4, 0, 0]
22         , [0, 0, 8, 1]
23         , [0, 0, 1, 2]
24     ])

```

(continues on next page)

(продолжение с предыдущей страницы)

```

24 q = np.array([3, 0, 0, 0])
25 offset = 5.25
26
27 A = np.array([
28     [2, 3, 4, 5]
29     , [1, -1, 2, -3]
30     , [1, 2, 3, 4]
31     , [1, 1, 1, 1]
32     , [1, 3, 2, 1]
33 ])
34 b = np.array([20, 1, 15, 5, 12])
35 senses = np.array(
36     [
37         ap.constraint_sense.less_equal
38         , ap.constraint_sense.greater_equal
39         , ap.constraint_sense.equal
40         , ap.constraint_sense.greater_equal
41         , ap.constraint_sense.less_equal
42     ]
43     , np.int32
44 )
45
46 return vars_type, lower_bounds, upper_bounds, count, P, q, offset, A, b, senses
47
48
49 if __name__ == "__main__":
50     # Получение данных для создания модели MIQP
51     vars_type, lower_bounds, upper_bounds, count, P, q, offset, A, b, senses = get_Data()
52
53     # Инициализация модели
54     model = ap.MatrixModel()
55     # Добавление переменных
56     model.add_variables(
57         count=count, lower_bounds=lower_bounds, upper_bounds=upper_bounds, var_
58         ↪types=vars_type
59     )
60     # Добавление целевой функции
61     model.set_objective(P=P, q=q, sense=ap.objective_sense.minimize)
62     # Добавление константы к целевой функции
63     model.set_objective_offset(offset)
64     # Добавление ограничений
65     model.add_constraints(A=A, senses=senses, b=b)
66
67     # Сохранение модели в формате LP
68     model.write_lp('MIQP.lp')
69     # Сохранение модели в формате MPS
70     model.write_mps('MIQP.mps')
71
72     # Установление параметров расчета
73     model.set_dbl_param('time_limit', 10)
74     model.set_bool_param('log_to_console', False)

```

(continues on next page)

(продолжение с предыдущей страницы)

```

75  # Запуск оптимизации
76  result = model.solve()
77
78  # Вывод полученного результата
79  if result.get_solve_result() == ap.solve_result.success:
80      if (result.get_solution_status() == ap.solution_status.optimal or \
81          result.get_solution_status() == ap.solution_status.feasible):
82          print(f'Status problem: {result.get_solve_result()}')
83          print(f'Objective value: {result.get_objective_value()}')
84          print(f'GAP: {result.get_relative_gap()}')
85          print(f'TIME: {result.get_solve_time()} s')
86          print(f'Solution vector: {result.get_solution_vector()}')
87
88      # Сохранение найденного решения
89      result.write_solution('result.sol')
90  else:
91      print(f'Status problem: {result.get_solution_status()}')
92  else:
93      print(f'Status process: {result.get_solve_result()}')

```

Если целевая функция квадратичная, то задать ее можно при помощи метода `set_objective()`

```
model.set_objective(P=P, q=q, sense=ap.objective_sense.minimize)
```

Первый аргумент – квадратная симметричная матрица, задающая квадратичную часть целевой функции, с равным количеством строк и столбцов, соответствующим количеству переменных. При считывании используется формула

$$\frac{1}{2} \bar{x}^T P \bar{x},$$

что обуславливает необходимость удвоения элементов главной диагонали. Матрица может быть представлена в формате `scipy.sparse.csr_matrix` или `numpy.ndarray`. Затем вектор коэффициентов переменных, и наконец тип оптимизации.

Для добавления константного значения к целевой функции применяется метод матричной модели `set_objective_offset()`

```
model.set_objective_offset(offset)
```

13.2.3 Пример построения задачи с квадратичной ЦФ и квадратичными ограничениями

Рассмотрим следующую задачу **QPQC**:

Vars:

$$x_1 \in \mathbb{B}$$

$$x_2 \in \mathbb{Z}$$

$$x_3, x_4 \in \mathbb{R}$$

Objective:

$$x_1^2 + 2x_2^2 + x_3^2 + 2.5x_1x_4 + x_1 + 3x_3 \rightarrow \max_{x_1, x_2, x_3, x_4}$$

Subject to:

$$x_1^2 + x_2^2 + x_3^2 \leq 20 \quad (13.3)$$

$$x_1 + x_2 + x_3 + x_4 \leq 10$$

$$x_1 + 2x_2 + 3x_3 \geq 5$$

Bounds:

$$0 \leq x_1 \leq 1$$

$$0 \leq x_2 < \text{inf}$$

$$0 \leq x_3 < \text{inf}$$

$$0 \leq x_4 < \text{inf}$$

Ниже приведен код для использования **MatrixAPI**:

```

1 import arhiplexpy as ap
2 import numpy as np
3 from scipy.sparse import csr_matrix
4
5 def get_Data():
6     """
7     :return: Возвращает данные для создания модели
8     """
9
10    vars_type = np.array(
11        [ap.variable_type.binary, ap.variable_type.integer, ap.variable_type.continuous,
12        ↪ ap.variable_type.continuous]
13        , np.int32
14    )
15    lower_bounds = np.array([0, -ap.kArhiplexInf, -ap.kArhiplexInf, -ap.kArhiplexInf])
16    count = 4
17
18    # Диагональ должна быть удвоена относительно исходной матрицы!
19    val = np.array([2, 4, 2, 2.5, 2.5])
20    row = np.array([0, 1, 2, 0, 3])
21    col = np.array([0, 1, 2, 3, 0])
22    P = csr_matrix((val, (row, col)), shape=(count, count))
23    q = np.array([1, 0, 3, 0])
24
25    A = np.array([
26        [1, 1, 1, 1]
27        , [1, 2, 3, 0]

```

(continues on next page)

(продолжение с предыдущей страницы)

```

27     ])
28     b = np.array([10, 5])
29     senses = np.array(
30         [
31             ap.constraint_sense.less_equal
32             , ap.constraint_sense.greater_equal
33         ]
34         , np.int32
35     )
36     val = np.array([2, 2, 2])
37     row = np.array([0, 1, 2])
38     col = np.array([0, 1, 2])
39     P_constr= csr_matrix((val, (row, col)), shape=(count, count))
40
41     return vars_type, lower_bounds, count, P, q, A, b, senses, P_constr
42
43
44 if __name__ == "__main__":
45     # Получение данных для создания модели QPQC
46     vars_type, lower_bounds, count, P, q, A, b, senses, P_constr = get_Data()
47
48     # Инициализация модели
49     model = ap.MatrixModel()
50     # Добавление переменных
51     model.add_variables(
52         count=count, lower_bounds=lower_bounds, var_types=vars_type
53     )
54     # Добавление целевой функции
55     model.set_objective(P=P, q=q, sense=ap.objective_sense.maximize)
56     # Добавление квадратичного ограничения
57     model.add_constraint(P=P_constr, q=None, sense=ap.constraint_sense.less_equal, r=20)
58     # Добавление линейных ограничений
59     model.add_constraints(A=A, senses=senses, b=b)
60
61     # Сохранение модели в формате LP
62     model.write_lp('QPQP.lp')
63     # Сохранение модели в формате MPS
64     model.write_mps('QPQP.mps')
65
66     # Установление параметров расчета
67     model.set_dbl_param('time_limit', 10)
68     model.set_bool_param('log_to_console', False)
69
70     # Запуск оптимизации
71     result = model.solve()
72
73     # Вывод полученного результата
74     if result.get_solve_result() == ap.solve_result.success:
75         if (result.get_solution_status() == ap.solution_status.optimal or \
76             result.get_solution_status() == ap.solution_status.feasible):
77             print(f'Status problem: {result.get_solve_result()}')
78             print(f'Objective value: {result.get_objective_value()}')

```

(continues on next page)

(продолжение с предыдущей страницы)

```

79     print(f'GAP: {result.get_relative_gap()}')
80     print(f'TIME: {result.get_solve_time()} s')
81     print(f'Solution vector: {result.get_solution_vector()}')
82
83     # Сохранение найденного решения
84     result.write_solution('result.sol')
85     else:
86         print(f'Status problem: {result.get_solution_status()}')
87     else:
88         print(f'Status process: {result.get_solve_result()}')

```

Для добавления квадратичного ограничения применяется метод матричной модели `add_constraint()`

```
model.add_constraint(P=P_constr, q=None, sense=ap.constraint_sense.less_equal, r=20)
```

Первый аргумент – квадратная симметричная матрица, задающая квадратичную часть ограничения, с равным количеством строк и столбцов, соответствующим количеству переменных. При считывании используется формула

$$\frac{1}{2}\bar{x}^T P \bar{x},$$

что обуславливает необходимость удвоения элементов главной диагонали. Затем передается вектор коэффициентов переменных, может быть задан в формате `List[float]`, `numpy.ndarray`. Третьим аргументом передаётся тип ограничения, `constraint_sense`. И наконец константное значение правой части ограничения.

Двойственные решения

Для задач линейного (LP) и квадратичного (QP) программирования доступно получение двойственных решений.

См.также:

Понятие двойственных оценок — в разделе *Двойственные оценки*. Ниже примеры приведены на Python; аналогичные методы доступны в разделах *C++ API*, *C# API*, *Java API*, *Julia API*.

Внимание: Функционал доступен только для задач без целочисленных переменных и являющихся выпуклыми (*convex*).

Для получения двойственных решений используется метод *SolveResult.get_dual_value()* (в Python) и функция *GetDualValue()* (в C++). В качестве параметра в обоих случаях может передаваться наименование ограничения *result.get_dual_value(«constraint»)* или ссылка на экземпляр класса *Constraint*.

14.1 Пример

Рассмотрим следующую задачу линейного программирования:

```
Maximize
obj: 3 x1 + 5 x2
Subject To
c1 : x1 <= 4
c2 : 2 x2 <= 12
c3 : 3 x1 + 2 x2 <= 18
Bounds
End
```

Реализация с использованием солвера **ArhiPlex** имеет следующий вид:

```

1 import arhiplexpy
2
3 # Создание экземпляра модели
4
5 model = arhiplexpy.Model()
6
7 # Создание переменных
8
9 x1 = model.add_variable(0, arhiplexpy.kArhiplexInf, 3, arhiplexpy.variable_type.
    ↳continuous, 'x1')
10 x2 = model.add_variable(0, arhiplexpy.kArhiplexInf, 5, arhiplexpy.variable_type.
    ↳continuous, 'x2')
11
12 # Добавление ограничений
13
14 model.add_constraint(x1 <= 4, 'c1')
15 model.add_constraint(2 * x2 <= 12, 'c2')
16 model.add_constraint(3 * x1 + 2 * x2 <= 18, 'c3')
17
18 # Направление оптимизации
19
20 model.set_objective_sense(arhiplexpy.objective_sense.maximize)
21
22 # Запуск расчёта
23
24 result = model.solve()
25
26 # Вывод решений
27
28 print('Objective:', result.get_objective_value())
29 print('x1:', result.get_variable_value("x1"))
30 print('x2:', result.get_variable_value("x2"))
31
32 print('The dual value of c1:', result.get_dual_value("c1"))
33 print('The dual value of c2:', result.get_dual_value("c2"))
34 print('The dual value of c3:', result.get_dual_value("c3"))

```

Результат решения задачи:

```

Запуск ArhiCloud
Права на использование (с) 2024 Arhitek

Параметры запуска:
presolve : on
problem_type : max
time_limit : 90
abs_gap : 1e-06
gap : 0.0001
primal_feasibility_tolerance : 1e-06
mip_feasibility_tolerance : 1e-06
remote_timeout : 300
Отправка запроса на расчет . . . успешно
UID: 20241022_111823_8378ac

```

(continues on next page)

(продолжение с предыдущей страницы)

```
Начало загрузки модели . . . успешно
Ожидание начала расчета .Запуск ArhiPlex 2.0.11.123984
Права на использование (с) 2024 Arhitex
Процессор Intel Xeon Processor (Icelake) [ AVX512, AVX2 ], физических ядер CPU: 10

Параметры запуска:
    time_limit =    90.00000000

Оригинальная задача LP temp.mps содержит ограничений: 3, переменных: 2 (дробных: 2,
↪бинарных: 0, целых: 0)

Отчет
Статус                               : Оптимально
Значение целевой функции             : 36
Статус решения                       : Достижимо
Время                               : 0.0 с.
LP итераций                          : 9
Objective: 36.0
x1: 2.0
x2: 6.0
The dual value of c1: 0.0
The dual value of c2: 1.5
The dual value of c3: 1.0
```

Начальные решения

См.также:

Понятие начального решения — в разделе *Начальное решение*. Ниже примеры приведены на Python; для MIP-start в других языках см. разделы *C++ API*, *Java API*, *Julia API*.

В тех случаях, когда в процессе решения задачи **MIP** наблюдается медленная сходимость, использование функционала установки начальных решений может значительно повлиять на производительность. Разработчик модели предоставляет алгоритмам допустимое решение вместе с моделью. Это реализуется через методы *API*, такие как *Model.add_mip_start_values()* в *Python* или функцию *AddMipStartValue()* в *C++*. В качестве параметра метод принимает путь к файлу с решением (*.sol*), либо словарь со значениями (в *Python*).

Рассмотрим применение функционала на языке *Python*. Применение начального решения в задачах **MIP** может быть осуществлено с использованием файла решений **.sol*, например:

```
1 import arhiplexpy as ap
2
3 m = ap.Model()
4 m.set_dbl_param('time_limit', 60)
5
6 # Чтение модели из файла
7 m.read('model.mps')
8
9 # Решение задачи и запись решения в файл
10 result = m.solve()
11 result.write_solution('solution.sol')
12
13 # Добавление решение из файла в качестве начального и повторный запуск решения
14 m.add_mip_start_values('solution.sol')
15 result = m.solve()
```

В данном примере можно рассмотреть механику применения начального решение через повторный запуск той же задачи с начальным решением из файла. Значение целевой функции во второй части решения уже учитывает применённое начальное решение.

Кроме того, применение начального решения может быть осуществлено с использованием словаря, где ключ - наименование переменной, а значение – численное значение переменной в применяемом решении. Например:

```

1 import arhiplexpy as ap
2
3 m = ap.Model()
4 m.set_dbl_param('time_limit', 60)
5
6 # Чтение модели из файла
7 m.read('model.mps')
8
9 # Решение задачи
10 result = m.solve()
11
12 # Экспорт решения в словарь
13 start_solution = {}
14 for i in range(m.get_variables_count()):
15     var_name = m.get_variable(i).get_name()
16     var_value = result.get_variable_value(var_name)
17     start_solution[var_name] = var_value
18
19 # Добавление решение из словаря в качестве начального и повторный запуск решения
20 m.add_mip_start_values(start_solution)
21 result = m.solve()

```

В данном примере файл решения не используется, однако, оперирование начальным решением и его подстановка во второй запуск осуществляется.

15.1 Частичный старт MILP и бюджет достроения

Словарь и `.sol` могут содержать только часть переменных. Отсутствующее значение означает «не задано», включая отсутствующие нули. Неизвестные имена игнорируются с предупреждением (число и до трёх примеров); пустой старт или отсутствие совпавших имён не запускают вспомогательную задачу. Совпадение имён разных моделей не гарантирует одинаковый смысл переменных или совместимость старта.

`mip_start_completion_node_limit` — целое число:

- -1 (AUTO, по умолчанию): наследовать на момент `solve()` значение `mip_heuristic_subproblem_node_limit` (0.2000000000, по умолчанию 500).
- -2 (FULL_START_ONLY): проверять только полные старты, без достроения.
- -3 (DISABLED): пропускать все старты, включая полные.
- 0: разрешены пресолв и LP-релаксация; ветвление запрещено.
- 1..2000000000: явный предел узлов достроения, независимо от бюджета эвристики.

`mip_start_completion_time_limit` и `mip_start_completion_work_limit` — неотрицательные double, по умолчанию Infinity. Первый ограничивает суммарные секунды достроения одного старта, второй — суммарную вычислительную работу. Одна единица работы — одна завершённая итерация последовательного симплекс-метода, включая LP-релаксации и повторные LP-попытки. Счётчик не включает чтение, пресолв, распространение доменов и проверку вектора; это детерминированная мера численной работы, а не время или число узлов. Дробный остаток бюджета не разрешает целую итерацию.

Повторные попытки не обнуляют бюджет. Для воспроизводимости нужны одинаковые модель, сборка, аппаратура, параметры, seed и число потоков, без конкурирующего временного лимита. Дostroение использует последовательный симплекс; основной расчёт сохраняет свои настройки потоков.

Действуют совместно узловой, временной, work-бюджеты, общий `time_limit` и отмена. Первый достигнутый предел прекращает достроение. Локальный предел позволяет продолжить основной поиск; общий таймаут завершает весь расчёт. Проверенный допустимый старт сохраняется, даже если основной поиск не нашёл лучшего решения. Полный допустимый старт проверяется без вспомогательного MIP даже при нулевых локальных бюджетах; полный недопустимый старт отклоняется без исправления.

В отчёте указаны заданный и эффективный бюджет, фактические `nodes`, `seconds`, `work`, флаг попытки `completion` и наличия допустимого решения `feasible`. Коды результата: `accepted` — принят; `not_improving` — допустим, но не улучшает; `disabled` / `full_only` — пропущен настройкой; `no_values` — нет применимых значений; `node_limit` / `time_limit` / `work_limit` — локальный предел; `incompatible` — установлена несовместимость; `numerical_error` — численная ошибка; `global_timeout` / `cancelled` — общий таймаут / отмена. Неудача в пределах бюджета не доказывает несовместимость. Пример строки приводится, если её нарушение установлено. Дostroение выполняется один раз в родительском процессе; рабочим процессам передаётся проверенный полный вектор.

```
model.add_mip_start_values({"x24809": 1.0})
model.set_int_param("mip_start_completion_node_limit",
                    int(arhiplexpy.MipStartCompletion.AUTO))
model.set_int_param("mip_heuristic_subproblem_node_limit", 1000)
model.set_dbl_param("mip_start_completion_time_limit", 10.0)
model.set_dbl_param("mip_start_completion_work_limit", 10000.0)
result = model.solve()
```

В C++ доступны те же символы `arhiplex::MipStartCompletion` с преобразованием `static_cast<int>(...)`. В файле опций CLI используются числовые значения.

16.1 Введение

Когда модель оптимизации не имеет достижимого решения, важно уметь определять причину этого. Для этого в солвере **ArhiPlex** реализован функционал анализа несократимого недостижимого множества — **IIS**. Этот инструмент позволяет выявить минимальный набор ограничений и границ переменных, которые делают модель недостижимой, что значительно упрощает процесс диагностики и исправления ошибок.

16.1.1 Что такое IIS?

Несократимое недостижимое множество **IIS** (*Irreducible Infeasible Set*) — это подмножество ограничений и границ переменных, обладающее следующими свойствами:

1. *Недостижимость*: Подмножество само по себе не имеет достижимого решения.
2. *Минимальность*: Если удалить хотя бы одно ограничение или границу, подсистема становится достижимой.

IIS помогает локализовать проблему, показывая, какие именно ограничения или границы вызывают конфликт.

См.также:

Концепция диагностики недостижимости — в разделе *Диагностика недостижимости (IIS)*; статус **infeasible** и порядок разбора результата — в разделе *Статусы и коды возврата*. Вычисление IIS включается параметром `compute_iis`.

16.1.2 Преимущества использования IIS

Использование **IIS API** предоставляет следующие преимущества:

1. *Локализация проблем:* Точное определение ограничений или переменных, делающих модель недостижимой.
2. *Экономия времени:* Автоматический поиск проблемных элементов вместо ручного перебора.
3. *Улучшение модели:* Возможность лучше понять структуру модели и внести корректировки.

16.1.3 Задачи, решаемые IIS API

IIS API помогает решить следующие задачи:

1. *Анализ недостижимых моделей:* Определение причин, по которым модель не имеет достижимого решения.
2. *Оптимизация модели:* Упрощение модели путем удаления избыточных или конфликтующих ограничений.
3. *Отладка:* Поиск ошибок в данных или формулировке ограничений.

16.1.4 Рекомендации по использованию

Для эффективного использования IIS API следуйте этим рекомендациям:

1. *Проверка модели:* Используйте IIS API после того, как солвер вернул статус «недостижимая модель».
 2. *Анализ результатов:* Внимательно изучите выявленные ограничения и переменные, чтобы понять причину конфликта.
 3. *Корректировка модели:* Удалите или измените проблемные ограничения, чтобы сделать модель достижимой.
 4. *Тестирование:* После внесения изменений повторно запустите модель, чтобы убедиться в ее корректности.
-

16.2 Использование IIS API в Arhiplex

16.2.1 Особенности работы с IIS

- *Сложность расчета для LP, MIP, QP и QCQP:* IIS API работает как для линейных LP, так и для смешанно-целочисленных MIP, а также квадратичных QP и QCQP задач. Тем не менее, для задач MIP время вычисления может быть значительным, а для QP и QCQP, задач с более сложной структурой, это время может увеличиться еще сильнее.
- *Несколько IIS:* Недостижимая модель может иметь несколько IIS. ArhiPlex возвращает один из них, который не обязательно является наименьшим по суммарному числу ограничений и границ переменных.

- *Прерывание вычислений*: В случае прерывания вычисления **IIS** (как вручную, так и в результате достижения заданного времени, установленного с помощью параметра *time_limit*) **ArhiPlex** возвращает недостижимую подсистему, если она уже была найдена. Такая подсистема может быть неминимальной и содержать избыточные ограничения и границы переменных. Если к моменту остановки подсистема не найдена и достижимость модели не установлена, статус процесса равен **fail**, статус модели — **unknown**, а результат содержит пустое **IIS**.

16.2.2 Вычисление IIS

Рассмотрим расчет **IIS** для следующей недостижимой квадратичной проблемы:

$$\begin{aligned}
 &\textbf{Objective:} \\
 &2x_1 + 7x_2 - 5x_3 \rightarrow \max \\
 &\textbf{Subject to:} \\
 &x_1 + x_2 + x_3 \leq 10 \\
 &x_1^2 + x_2^2 \leq 5 \\
 &x_2^2 + x_3^2 \leq 5 \\
 &\textbf{Bounds:} \\
 &6 \leq x_1 \\
 &0 \leq x_2 \\
 &6 \leq x_3
 \end{aligned} \tag{16.1}$$

Ниже приведен код для использования **IIS API**:

```

1  # Maximize
2  # 2 * x1 + 7 * x2 - 5 * x3
3  # Subject to
4  # x1 + x2 + x3 <= 10
5  # x1 ** 2 + x2 ** 2 <= 5
6  # x2 ** 2 + x3 ** 2 <= 5
7  # Bounds
8  # 6 <= x1
9  # 0 <= x2
10 # 6 <= x3
11
12
13 import arhiplexpy as ap
14
15 if __name__ == "__main__":
16     # Создаем модель
17     model = ap.Model()
18
19     # Добавляем переменные
20     x1 = model.add_variable(lower_bound=6, var_type=ap.variable_type.continuous, var_
21     ↪ name='x1', obj_value=2)
21     x2 = model.add_variable(lower_bound=0, var_type=ap.variable_type.continuous, var_
22     ↪ name='x2', obj_value=7)
22     x3 = model.add_variable(lower_bound=6, var_type=ap.variable_type.continuous, var_
23     ↪ name='x3', obj_value=-5)
23
24     # Устанавливаем тип задачи

```

(continues on next page)

(продолжение с предыдущей страницы)

```

25 model.set_objective_sense(ap.objective_sense.maximize)
26
27 # Добавляем ограничения
28 model.add_constraint(x1 + x2 + x3 <= 10, 'c1')
29 model.add_constraint(x1 ** 2 + x2 ** 2 <= 5, 'c2')
30 model.add_constraint(x2 ** 2 + x3 ** 2 <= 5, 'c3')
31
32 # Запускаем оптимизацию
33 result = model.solve()
34
35 # Проверяем статус решения задачи
36 if result.get_solve_result() == ap.solve_result.success and \
37    (result.get_solution_status() == ap.solution_status.infeasible or \
38     result.get_solution_status() == ap.solution_status.infeasible_or_unbounded):
39     # Устанавливаем ограничение на расчет iis, задается в секундах
40     model.set_dbl_param('time_limit', 1.5)
41     # Модель недостижима, вызываем расчет IIS
42     iis_result = model.compute_iis()
43     # Выведем полезные данные о расчете iis
44     print(f'Статус процесса вычислений: {iis_result.get_compute_iis_result()}')
45     print(f'Время вычисления iis, с: {iis_result.get_compute_iis_time()}')
46     print(f'Минимально ли полученное множество: {iis_result.get_minimal()}')
47     print(f'Статус модели: {iis_result.get_model_status()}')
48     print(f'Число ограничений в несократимом недостижимом множестве: {iis_result.get_
↪ num_constraints()}')
49     print(f'Число границ в несократимом недостижимом множестве: {iis_result.get_num_
↪ bounds()}')
50     # Сохраняем iis в текстовый файл с расширением .lp
51     iis_result.write_iis('iis_result.lp')
52 else:
53     print(f'Статус процесса расчета: {result.get_solve_result()}')
54     print(f'Статус модели после расчета: {result.get_solution_status()}')

```

Предупреждение: Прежде чем запустить расчет IIS необходимо убедиться, что расчет решения модели был успешным и проблема недостижима.

Для этого используем методы результата расчета: `result.get_solve_result()`, `result.get_solution_status()`, и константы ArhiPlex: `ap.solve_result.success`, `ap.solution_status.infeasible`, `ap.solution_status.infeasible_or_unbounded`

```

if result.get_solve_result() == ap.solve_result.success and \
   (result.get_solution_status() == ap.solution_status.infeasible or \
    result.get_solution_status() == ap.solution_status.infeasible_or_unbounded):

```

Вызываем расчет IIS при помощи метода модели `model.compute_iis()`:

```
iis_result = model.compute_iis()
```

Просматриваем различные характеристики расчета IIS:

```

print(f'Статус процесса вычислений: {iis_result.get_compute_iis_result()}')
print(f'Время вычисления iis, с: {iis_result.get_compute_iis_time()}')
print(f'Минимально ли полученное множество: {iis_result.get_minimal()}')
print(f'Статус модели: {iis_result.get_model_status()}')
print(f'Число ограничений в несократимом недостижимом множестве: {iis_result.get_
↪ num_constraints()}')
print(f'Число границ в несократимом недостижимом множестве: {iis_result.get_num_
↪ bounds()}')

```

Сохраняем **IIS** в формате **.lp** при помощи метода результата расчета **IIS** `iis_result.write_iis()`, в параметры передаем полный путь к файлу:

```
iis_result.write_iis('iis_result.lp')
```

16.2.3 Результат

Модель недостижима. Проанализируем содержимое файла **IIS** `iis_result.lp`:

```

File written by ArhiPlex LP filereader

Minimize

obj:

Subject To

c1 : x1 + x2 + x3 <= 10

Bounds

6 <= x1

6 <= x3

End

```

В **IIS** входит ограничение `c1`, а также нижние границы переменных `x1` и `x3`, а также граница переменной `x2`. Хотя это не указано явно, но по условию $0 \leq x2$, в файл границы вида $0 \leq x < \text{inf}$ не записываются. Если бы переменная была неограниченна, то она попала бы в раздел **Free**:

```

Free

x2

```

16.2.4 Разрешение конфликта

Для приведенного примера сделать задачу достижимой можно несколькими способами:

- увеличить правую часть ограничения $c1$.
- уменьшить нижние границы одной или нескольких переменных.

Реализуем второй вариант, уменьшив границу до 2:

$$\begin{aligned}
 &\textbf{Objective:} \\
 &2x_1 + 7x_2 - 5x_3 \rightarrow \max \\
 &\textbf{Subject to:} \\
 &x_1 + x_2 + x_3 \leq 10 \\
 &x_1^2 + x_2^2 \leq 5 \\
 &x_2^2 + x_3^2 \leq 5 \\
 &\textbf{Bounds:} \\
 &2 \leq x_1 \\
 &0 \leq x_2 \\
 &2 \leq x_3
 \end{aligned} \tag{16.2}$$

Ниже приведен код для использования **IIS API**:

```

1  # Maximize
2  # 2 * x1 + 7 * x2 - 5 * x3
3  # Subject to
4  # x1 + x2 + x3 <= 10
5  # x1 ** 2 + x2 ** 2 <= 5
6  # x2 ** 2 + x3 ** 2 <= 5
7  # Bounds
8  # 2 <= x1
9  # 0 <= x2
10 # 2 <= x3
11
12
13 import arhiplexpy as ap
14
15 if __name__ == "__main__":
16     # Создаем модель
17     model = ap.Model()
18
19     # Добавляем переменные
20     x1 = model.add_variable(lower_bound=2, var_type=ap.variable_type.continuous, var_
↪ name='x1', obj_value=2)
21     x2 = model.add_variable(lower_bound=0, var_type=ap.variable_type.continuous, var_
↪ name='x2', obj_value=7)
22     x3 = model.add_variable(lower_bound=2, var_type=ap.variable_type.continuous, var_
↪ name='x3', obj_value=-5)
23
24     # Устанавливаем тип задачи
25     model.set_objective_sense(ap.objective_sense.maximize)
26
27     # Добавляем ограничения

```

(continues on next page)

(продолжение с предыдущей страницы)

```

28 model.add_constraint(x1 + x2 + x3 <= 10, 'c1')
29 model.add_constraint(x1 ** 2 + x2 ** 2 <= 5, 'c2')
30 model.add_constraint(x2 ** 2 + x3 ** 2 <= 5, 'c3')
31
32 # Запускаем оптимизацию
33 result = model.solve()
34
35 # Проверяем статус решения задачи
36 if result.get_solve_result() == ap.solve_result.success and \
37     (result.get_solution_status() == ap.solution_status.infeasible or \
38      result.get_solution_status() == ap.solution_status.infeasible_or_unbounded):
39     # Устанавливаем ограничение на расчет iis, задается в секундах
40     model.set_dbl_param('time_limit', 1.5)
41     # Модель недостижима, вызываем расчет IIS
42     iis_result = model.compute_iis()
43     # Выведем полезные данные о расчете iis
44     print(f'Статус процесса вычислений: {iis_result.get_compute_iis_result()}')
45     print(f'Время вычисления iis, с: {iis_result.get_compute_iis_time()}')
46     print(f'Минимально ли полученное множество: {iis_result.get_minimal()}')
47     print(f'Статус модели: {iis_result.get_model_status()}')
48     print(f'Число ограничений в несократимом недостижимом множестве: {iis_result.get_
↳ num_constraints()}')
49     print(f'Число границ в несократимом недостижимом множестве: {iis_result.get_num_
↳ bounds()}')
50     # Сохраняем iis в текстовый файл с расширением .lp
51     iis_result.write_iis('iis_result.lp')
52 elif result.get_solve_result() == ap.solve_result.success and \
53     (result.get_solution_status() == ap.solution_status.optimal or \
54      result.get_solution_status() == ap.solution_status.feasible):
55     # Модель допустима, вызываем текущее найденное решение
56     print('Objective value:', round(result.get_objective_value(), 5))
57     print(f'x1: {round(result.get_variable_value(x1), 5)}')
58     print(f'x2: {round(result.get_variable_value(x2), 5)}')
59     print(f'x3: {round(result.get_variable_value(x3), 5)}')
60 else:
61     print(f'Статус процесса расчета: {result.get_solve_result()}')
62     print(f'Статус модели после расчета: {result.get_solution_status()}')

```

Получим следующее решение

```

Objective value: 1.0
x1: 2.0
x2: 1.0
x3: 2.0

```

IIS API — это мощный инструмент для анализа и исправления недостижимых моделей. Его использование позволяет:

- Локализовать проблемные ограничения и переменные.
- Упростить процесс отладки и улучшения моделей.

Пакет `arhiplexpy` предоставляет Python-интерфейс к солверу ArhiPlex (LP/MIP/QP/QCQP/NLP). Классы и перечисления реализованы в native-расширении `arhiplexpy.arhiplexpy` (pybind11) и реэкспортируются на уровень пакета, поэтому обычно достаточно:

```
import arhiplexpy as ap

model = ap.Model()
result = model.solve()
```

17.1 Подключение

Установите пакет из дистрибутива в виртуальное окружение Python 3.7–3.12 (см. [Установка](#)). Нативная библиотека должна быть доступна через `LD_LIBRARY_PATH=${ARHIPLEX_HOME}/lib64`. Проверка:

```
import arhiplexpy as ap
print(ap.__file__)
```

17.2 Сквозной пример

Рассмотрим задачу смешанно-целочисленного программирования:

$$\begin{aligned}
 &\textbf{Maximize:} \\
 &x_1 + 2x_2 + 3x_3 + x_4 \\
 &\textbf{Subject to:} \\
 &-x_1 + x_2 + x_3 + 10x_4 \leq 20 \\
 &x_1 - 3x_2 + x_3 \leq 30 \\
 &x_2 - 3.5x_4 = 0 \\
 &\textbf{Bounds:} \\
 &0 \leq x_1 \leq 40 \\
 &2 \leq x_4 \leq 3 \\
 &\textbf{Integers:} \\
 &x_4
 \end{aligned} \tag{17.1}$$

```

1  # Maximize  x1 + 2 x2 + 3 x3 + x4
2  #   Subject to
3  #       - x1 +   x2 + x3 + 10 x4  <= 20
4  #         x1 - 3 x2 + x3          <= 30
5  #           x2          - 3.5x4   = 0
6  #   Bounds
7  #       0 <= x1 <= 40
8  #       0 <= x2
9  #       0 <= x3
10 #       2 <= x4 <= 3
11 #   Integers
12 #       x4
13
14 import arhiplexpy
15
16
17 if __name__ == "__main__":
18     model = arhiplexpy.Model()
19
20     model.set_dbl_param("mip_rel_gap", 0.1)
21
22     x1 = model.add_variable(0, 40, 1, arhiplexpy.variable_type.continuous, 'x1')
23     x2 = model.add_variable(0, arhiplexpy.kArhiplexInf, 2, arhiplexpy.variable_type.
↪continuous, 'x2')
24     x3 = model.add_variable(0, arhiplexpy.kArhiplexInf, 3, arhiplexpy.variable_type.
↪continuous, 'x3')
25     x4 = model.add_variable(2, 3, 1, arhiplexpy.variable_type.integer, 'x4')
26
27     model.add_constraint(-x1 + x2 + x3 + 10 * x4 <= 20, 'c1')
28     model.add_constraint(x1 - 3 * x2 + x3 <= 30, 'c2')
29     model.add_constraint(x2 - 3.5 * x4 == 0, 'c3')
30
31     model.set_objective_sense(arhiplexpy.objective_sense.maximize)
32

```

(continues on next page)

(продолжение с предыдущей страницы)

```

33 result = model.solve()
34
35 if result.get_solve_result() == arhiplexpy.solve_result.success and \
36    (result.get_solution_status() == arhiplexpy.solution_status.optimal or \
37     result.get_solution_status() == arhiplexpy.solution_status.feasible):
38     print('Objective value:', result.get_objective_value())
39     print('x1:', result.get_variable_value(x1))
40     print('x2:', result.get_variable_value('x2'))
41     print('x3:', result.get_variable_value('x3'))
42     print('x4:', result.get_variable_value(x4))
43 else:
44     print(result.get_solution_status())

```

Основные шаги: импорт `arhiplexpy`; создание `Model`; добавление переменных методом `add_variable(нижняя, верхняя, коэфф_в_ЦФ, тип, имя)`; добавление ограничений методом `add_constraint()`; выбор направления оптимизации `set_objective_sense()`; запуск `solve()`.

Предупреждение: Прежде чем читать значения переменных методом `get_variable_value()`, убедитесь, что расчёт успешен и модель достижима (см. [Интерпретация результата](#)).

Другие сценарии показаны в разделах [Быстрый старт](#) (запуск из файла), [Создание модели с использованием матриц](#) (матричное построение) и [Анализ недостижимости модели](#) (диагностика IIS).

17.3 Ключевые типы

Класс / тип	Назначение
<code>Model</code>	Модель: переменные, ограничения, цель, <code>solve</code> , параметры
<code>MatrixModel</code>	Пакетное (матричное) построение модели
<code>Variable</code>	Переменная, границы, тип, имя
<code>Constraint</code>	Ограничение (линейное или квадратичное)
<code>LinearExpression</code>	Линейное выражение
<code>QuadExpression</code>	Квадратичное выражение
<code>GeneralExpression</code>	Общее (нелинейное) выражение
<code>GeneralConstraint</code>	Общее ограничение
<code>MathFunc</code>	Статические нелинейные функции (<code>sin</code> , <code>pow</code> , ...)
<code>SolveResult</code>	Результат локального/удалённого расчёта
<code>MatrixSolveResult</code>	Результат расчёта матричной модели
<code>ComputeIisResult</code>	Результат вычисления IIS (enterprise)

Перечисления: `solve_result`, `solution_status`, `variable_type`, `objective_sense`, `constraint_sense`, `gen_expr_type` и др.

Примеры использования — в [Быстрый старт](#), [Матричный API](#), [Недостижимость](#).

17.4 Справочник (пакет)

ArhiPlexPy - Python interface for Arhiplex LP/MIP Solver with CP-SAT support

This module provides Python bindings for: - Arhiplex LP/MIP solver (native extension *arhiplexpy.arhiplexpy*)
- CP-SAT solver from *cp_solver* / *arhitools* (*arhiplexpy.arhitools*)

Usage:

```
# LP/MIP import arhiplexpy h = arhiplexpy.Arhiplex()
# CP-SAT (when built with cp_solver) from arhiplexpy.arhitools.sat.python import cp_model
```

17.5 Справочник (native-модуль)

Ниже перечислены классы с docstrings из pybind-обёртки (источник для autodoc):

```
class arhiplexpy.Model
```

Создает экземпляр модели

```
add_constraint(*args, **kwargs)
```

Overloaded function.

1. `add_constraint(self: arhiplexpy.arhiplexpy.Model, expr: arhiplexpy.arhiplexpy.LinearExpression, sense: arhiplexpy.arhiplexpy.constraint_sense, rhs: float, constr_name: str) -> arhiplexpy.arhiplexpy.Constraint`

Добавляет ограничение в модель с заданными параметрами

param expr

выражение для ограничения

param sense

тип ограничения ($\leq$, $\geq$, $=$)

param rhs

константное значение в правой части ограничения

param constr_name

имя ограничения

return

объект нового ограничения

2. `add_constraint(self: arhiplexpy.arhiplexpy.Model, expr: arhiplexpy.arhiplexpy.QuadExpression, sense: arhiplexpy.arhiplexpy.constraint_sense, rhs: float, constr_name: str) -> arhiplexpy.arhiplexpy.Constraint`

Добавляет ограничение в модель с заданными параметрами

param expr

выражение для ограничения

param sense

тип ограничения ($\leq$, $\geq$, $=$)

param rhs

константное значение в правой части ограничения

param constr_name
имя ограничения

return
объект нового ограничения

3. `add_constraint(self: arhiplexpy.arhiplexpy.Model, lhs: arhiplexpy.arhiplexpy.LinearExpression, sense: arhiplexpy.arhiplexpy.constraint_sense, rhs: arhiplexpy.arhiplexpy.LinearExpression, constr_name: str) -> arhiplexpy.arhiplexpy.Constraint`

Добавляет ограничение в модель с заданными параметрами

param lhs
выражение для ограничения (левая часть)

param sense
переменная знака ограничения ($\leq$, $\geq$, $=$)

param rhs
константное значение в правой части ограничения

param constr_name
имя ограничения

return
объект нового ограничения

4. `add_constraint(self: arhiplexpy.arhiplexpy.Model, expr: arhiplexpy.arhiplexpy.LinearExpression, lower_bound: float, upper_bound: float, constr_name: str) -> arhiplexpy.arhiplexpy.Constraint`

Добавляет интервальное ограничение в модель с заданными параметрами, вида $lhs \leq expr \leq rhs$

param expr
выражение для ограничения

param lower_bound
нижняя граница ограничения

param upper_bound
верхняя граница ограничения

param constr_name
имя ограничения

return
объект нового ограничения

5. `add_constraint(self: arhiplexpy.arhiplexpy.Model, expr: arhiplexpy.arhiplexpy.QuadExpression, lower_bound: float, upper_bound: float, constr_name: str) -> arhiplexpy.arhiplexpy.Constraint`

Добавляет интервальное ограничение в модель с заданными параметрами, вида $lhs \leq expr \leq rhs$

param expr
выражение для ограничения

param lower_bound
нижняя граница ограничения

param upper_bound
верхняя граница ограничения

param constr_name
имя ограничения

return
объект нового ограничения

6. `add_constraint(self: arhiplexpy.arhiplexpy.Model, constr: arhiplexpy.arhiplexpy.Constraint, constr_name: str) -> arhiplexpy.arhiplexpy.Constraint`

Добавляет уже существующее ограничение в модель

param constr
выражение для ограничения

param constr_name
имя ограничения

return
объект нового ограничения

`add_constraints(self: arhiplexpy.arhiplexpy.Model, constraints: dict) → None`

Добавляет ограничения в модель

param constraints
объект dictionary(name -> constraint)

`add_general_constraint(*args, **kwargs)`

Overloaded function.

1. `add_general_constraint(self: arhiplexpy.arhiplexpy.Model, expr: arhiplexpy.arhiplexpy.GeneralExpression, var: arhiplexpy.arhiplexpy.Variable, name: str) -> arhiplexpy.arhiplexpy.GeneralConstraint`

Добавляет общее ограничение

param expr
общее выражение

param var
переменная

param name
имя ограничения

return
объект ограничения

2. `add_general_constraint(self: arhiplexpy.arhiplexpy.Model, expr: arhiplexpy.arhiplexpy.QuadExpression, var: arhiplexpy.arhiplexpy.Variable, name: str) -> arhiplexpy.arhiplexpy.GeneralConstraint`

Добавляет ограничение

param expr
квадратичное выражение

param var
переменная

param name
имя ограничения

return
объект ограничения

```
3. add_general_constraint(self: arhiplexpy.arhiplexpy.Model, expr:
    arhiplexpy.arhiplexpy.LinearExpression, var: arhiplexpy.arhiplexpy.Variable, name: str)
    -> arhiplexpy.arhiplexpy.GeneralConstraint
```

Добавляет ограничение

param expr
линейное выражение

param var
переменная

param name
имя ограничения

return
объект ограничения

```
4. add_general_constraint(self: arhiplexpy.arhiplexpy.Model, gen_constr:
    arhiplexpy.arhiplexpy.GeneralConstraint) -> None
```

Добавляет общее ограничение

param gen_constr
общее ограничение

return
объект ограничения

add_mip_start_values(*args, **kwargs)

Overloaded function.

```
1. add_mip_start_values(self: arhiplexpy.arhiplexpy.Model, start_values: dict) -> None
```

Задать начальные значения переменных в решении в качестве «подсказки». Начальные значения очищаются после начала процесса решения.

param start_values
объект словаря (имя : значение)

```
2. add_mip_start_values(self: arhiplexpy.arhiplexpy.Model, sol_file: str) -> None
```

Задать начальные значения переменных в решении, указав файл решения

param sol_file
путь к файлу решения

```
add_variable(self: arhiplexpy.arhiplexpy.Model, lower_bound: float = 0.0, upper_bound: float = inf,
    obj_value: float = 0.0, var_type: arhiplexpy.arhiplexpy.variable_type =
    <variable_type.continuous: 0>, var_name: str = '') ->
    arhiplexpy.arhiplexpy.Variable
```

Добавляет переменную в модель с заданными параметрами

param lower_bound
нижняя граница переменной

param upper_bound
верхняя граница переменной

param obj_value
коэффициент к переменной в целевой функции

param var_type
тип переменной

param var_name
имя переменной

return
объект новой переменной

`binary_var_list(self: arhiplexpy.arhiplexpy.Model, indices: list, name: str) → list`

Добавляет лист бинарных переменных

param indices
список индексов, добавляемых к начальному имени (или список кортежей)

param name
префикс имени каждой добавленной переменной

return
список переменных

`clear(self: arhiplexpy.arhiplexpy.Model) → None`

Очищает модель от всех данных. Все старые переменные и ограничения становятся невалидными

`clear_mip_start_values(self: arhiplexpy.arhiplexpy.Model) → None`

Удалить все стартовые значения для поиска решения

`compute_iis(self: arhiplexpy.arhiplexpy.Model) → arhiplexpy.arhiplexpy.ComputeIisResult`

Стартует вычисление несократимого недостижимого множества

return
объект с результатом вычисления несократимого недостижимого множества

`continuous_var_list(*args, **kwargs)`

Overloaded function.

1. `continuous_var_list(self: arhiplexpy.arhiplexpy.Model, indices: list, lb: list, ub: list, name: str) -> list`

Добавляет лист непрерывных переменных

param indices
список индексов, добавляемых к начальному имени (или список кортежей)

param lb
список нижних границ

param ub
список верхних границ

param name

префикс имени каждой добавленной переменной

return

список переменных

2. `continuous_var_list(self: arhiplexpy.arhiplexpy.Model, indicies: list, lb: float = 0.0, ub: float = inf, name: str) -> list`

Добавляет лист непрерывных переменных

param indicies

список индексов, добавляемых к начальному имени (или список кортежей)

param lb

значение нижней границы (для всех переменных)

param ub

значение верхней границы (для всех переменных)

param name

префикс имени каждой добавленной переменной

return

список переменных

3. `continuous_var_list(self: arhiplexpy.arhiplexpy.Model, indicies: list, lb: float = 0.0, ub: list, name: str) -> list`

Добавляет лист непрерывных переменных

param indicies

список индексов, добавляемых к начальному имени (или список кортежей)

param lb

значение нижней границы (для всех переменных)

param ub

список верхних границ

param name

префикс имени каждой добавленной переменной

return

список переменных

4. `continuous_var_list(self: arhiplexpy.arhiplexpy.Model, indicies: list, lb: list, ub: float = inf, name: str) -> list`

Добавляет лист непрерывных переменных

param indicies

список индексов, добавляемых к начальному имени (или список кортежей)

param lb

список нижних границ

param ub

значение верхней границы (для всех переменных)

param name

префикс имени каждой добавленной переменной

return

список переменных

`get_bool_param(self: arhiplexpy.arhiplexpy.Model, param_name: str) → bool`

Получает логический параметр

param param_name

имя параметра

return

значение параметра

`get_calc_uid(self: arhiplexpy.arhiplexpy.Model) → str`

Получить уникальный идентификатор последнего удаленного расчета. :return: строку с уникальным идентификатором

`get_constraint(*args, **kwargs)`

Overloaded function.

1. `get_constraint(self: arhiplexpy.arhiplexpy.Model, idx: int) -> arhiplexpy.arhiplexpy.Constraint`

Получает ограничение из модели по индексу

param idx

индекс ограничения

return

объект ограничения

2. `get_constraint(self: arhiplexpy.arhiplexpy.Model, constr_name: str) -> arhiplexpy.arhiplexpy.Constraint`

Получает ограничение из модели по имени

param constr_name

имя ограничения

return

объект ограничения

`get_constraints_count(self: arhiplexpy.arhiplexpy.Model) → int`

Получает количество ограничений в модели :return: количество ограничений в модели

`get_dbl_param(self: arhiplexpy.arhiplexpy.Model, param_name: str) → float`

Получает параметр с плавающей точкой

param param_name

имя параметра

return

значение параметра

`get_general_constraint(*args, **kwargs)`

Overloaded function.

1. `get_general_constraint(self: arhiplexpy.arhiplexpy.Model, idx: int) -> arhiplexpy.arhiplexpy.GeneralConstraint`

Получает общее ограничение из модели по индексу

```
param idx
    индекс ограничения
return
    объект ограничения
```

```
2. get_general_constraint(self: arhiplexpy.arhiplexpy.Model,   constr_name: str) ->
    arhiplexpy.arhiplexpy.GeneralConstraint
```

Получает общее ограничение из модели по имени

```
param constr_name
    имя ограничения
return
    объект ограничения
```

```
get_general_constraints_count(self: arhiplexpy.arhiplexpy.Model) -> int
```

Получает количество общих ограничений в модели

```
return
    количество общих ограничений в модели
```

```
get_int_param(self: arhiplexpy.arhiplexpy.Model, param_name: str) -> int
```

Получает целочисленный параметр

```
param param_name
    имя параметра
return
    значение параметра
```

```
get_name(self: arhiplexpy.arhiplexpy.Model) -> str
```

Получает имя модели

```
return
    имя модели
```

```
get_objective(self: arhiplexpy.arhiplexpy.Model) -> arhiplexpy.arhiplexpy.LinearExpression
```

Получает выражение целевой функции

```
return
    объект выражения
```

```
get_objective_sense(self: arhiplexpy.arhiplexpy.Model) -> arhiplexpy.arhiplexpy.objective_sense
```

Получает тип оптимизации целевой функции

```
return
    тип оптимизации
```

```
get_remote_solve_log(self: arhiplexpy.arhiplexpy.Model, arg0: str) -> str
```

Получает лог удалённого расчёта

:param calc_uid : идентификатор удалённого расчёта :return: лог удалённого расчёта

```
get_string_param(self: arhiplexpy.arhiplexpy.Model, param_name: str) -> str
```

Получает строковый параметр

param param_name
имя параметра

return
значение параметра

`get_variable(*args, **kwargs)`

Overloaded function.

1. `get_variable(self: arhiplexpy.arhiplexpy.Model, idx: int) -> arhiplexpy.arhiplexpy.Variable`

Получает переменную модели по индексу

param idx
индекс переменной

return
объект переменной

2. `get_variable(self: arhiplexpy.arhiplexpy.Model, var_name: str) -> arhiplexpy.arhiplexpy.Variable`

Получает переменную из модели по имени

param var_name
имя переменной

return
объект переменной

`get_variables_count(self: arhiplexpy.arhiplexpy.Model) -> int`

Получает количество переменных в модели

return
количество переменных в модели

`integer_var_list(*args, **kwargs)`

Overloaded function.

1. `integer_var_list(self: arhiplexpy.arhiplexpy.Model, indicies: list, lb: list, ub: list, name: str) -> list`

Добавляет лист целочисленных переменных

param indicies
список индексов, добавляемых к начальному имени (или список кортежей)

param lb
список нижних границ

param ub
список верхних границ

param name
префикс имени каждой добавленной переменной

return
список переменных

2. `integer_var_list(self: arhiplexpy.arhiplexpy.Model, indicies: list, lb: float = 0.0, ub: float = inf, name: str) -> list`

Добавляет лист целочисленных переменных

param indicies
 список индексов, добавляемых к начальному имени (или список кортежей)

param lb
 значение нижней границы (для всех переменных)

param ub
 значение верхней границы (для всех переменных)

param name
 префикс имени каждой добавленной переменной

return
 список переменных

3. `integer_var_list(self: arhiplexpy.arhiplexpy.Model, indicies: list, lb: float = 0.0, ub: list, name: str) -> list`

Добавляет лист целочисленных переменных

param indicies
 список индексов, добавляемых к начальному имени (или список кортежей)

param lb
 значение нижней границы (для всех переменных)

param ub
 список верхних границ

param name
 префикс имени каждой добавленной переменной

return
 список переменных

4. `integer_var_list(self: arhiplexpy.arhiplexpy.Model, indicies: list, lb: list, ub: float = inf, name: str) -> list`

Добавляет лист целочисленных переменных

param indicies
 список индексов, добавляемых к начальному имени (или список кортежей)

param lb
 список нижних границ

param ub
 значение верхней границы (для всех переменных)

param name
 префикс имени каждой добавленной переменной

return
 список переменных

`is_mip(self: arhiplexpy.arhiplexpy.Model) -> bool`

Проверяет является ли задача целочисленной

`is_nonlinear(self: arhiplexpy.arhiplexpy.Model) → bool`

Проверяет является ли задача нелинейной

`read(self: arhiplexpy.arhiplexpy.Model, file_name: str) → None`

Читает модель из файла

param file_name

имя файла

`read_lp(self: arhiplexpy.arhiplexpy.Model, file_name: str) → None`

Читает модель из файла, при этом он будет читаться как LP файл независимо от расширения

param file_name

путь к файлу модели

`read_mps(self: arhiplexpy.arhiplexpy.Model, file_name: str) → None`

Читает модель из файла, при этом он будет читаться как MPS файл независимо от расширения

param file_name

путь к файлу модели

`remove(*args, **kwargs)`

Overloaded function.

1. `remove(self: arhiplexpy.arhiplexpy.Model, constr: arhiplexpy.arhiplexpy.Constraint) -> None`

Удаляет ограничение из модели

param constr

ограничение

2. `remove(self: arhiplexpy.arhiplexpy.Model, var: arhiplexpy.arhiplexpy.Variable) -> None`

Удаляет переменную из модели

param var

переменная

3. `remove(self: arhiplexpy.arhiplexpy.Model, constr: arhiplexpy.arhiplexpy.GeneralConstraint) -> None`

Удаляет общее ограничение из модели

param constr

общее ограничение

`set_bool_param(self: arhiplexpy.arhiplexpy.Model, param_name: str, param_value: bool) → None`

Задаёт логический параметр

param param_name

имя параметра

param param_value

значение параметра

`set_dbl_param(self: arhiplexpy.arhiplexpy.Model, param_name: str, param_value: float) → None`

Задаёт параметр с плавающей точкой

param param_name

имя параметра

param param_value

значение параметра

set_int_param(*self*: arhiplexpy.arhiplexpy.Model, *param_name*: str, *param_value*: int) → None

Задаёт целочисленный параметр

param param_name

имя параметра

param param_value

значение параметра

set_log_callback(*self*: arhiplexpy.arhiplexpy.Model, *callback*: object) → None

Задаёт функтор обратного вызова для вывода сообщений от солвера

param callback

функтор обратного вызова

set_log_file(*self*: arhiplexpy.arhiplexpy.Model, *log_file*: str) → None

Задать файл для записи лога решения

param log_file

файл лога

set_name(*self*: arhiplexpy.arhiplexpy.Model, *expr_name*: str) → None

Задаёт имя модели

param expr_name

имя модели

set_objective(*args, **kwargs)

Overloaded function.

1. **set_objective**(*self*: arhiplexpy.arhiplexpy.Model, *var*: arhiplexpy.arhiplexpy.Variable, *sense*: arhiplexpy.arhiplexpy.objective_sense) -> None

Задаёт переменную для оптимизации и тип

param var

переменная

param sense

тип оптимизации

2. **set_objective**(*self*: arhiplexpy.arhiplexpy.Model, *expr*: arhiplexpy.arhiplexpy.LinearExpression, *sense*: arhiplexpy.arhiplexpy.objective_sense) -> None

Задаёт выражение целевой функции и тип оптимизации

param expr

выражение

param sense

тип оптимизации

3. **set_objective**(*self*: arhiplexpy.arhiplexpy.Model, *expr*: arhiplexpy.arhiplexpy.QuadExpression, *sense*: arhiplexpy.arhiplexpy.objective_sense) -> None

Задаёт выражение целевой функции и тип оптимизации

param expr
выражение

param sense
тип оптимизации

set_objective_offset(*self*: [arhiplexpy.arhiplexpy.Model](#), *offset*: *float*) → None

Задаёт константу в выражении целевой функции

param offset
значение константы

set_objective_sense(*self*: [arhiplexpy.arhiplexpy.Model](#), *constr_name*:
arhiplexpy.arhiplexpy.objective_sense) → None

Задаёт тип оптимизации целевой функции - максимизация или минимизация

param sense
тип оптимизации

set_string_param(*self*: [arhiplexpy.arhiplexpy.Model](#), *param_name*: *str*, *param_value*: *str*) → None

Задаёт строковый параметр

param param_name
имя параметра

param param_value
значение параметра

solve(*self*: [arhiplexpy.arhiplexpy.Model](#)) → *arhiplexpy.arhiplexpy.SolveResult*

Стартует оптимизацию модели

return
объект с результатом оптимизации

solve_remote(*self*: [arhiplexpy.arhiplexpy.Model](#), *name_mapping_file*: *str*) →
arhiplexpy.arhiplexpy.SolveResult

Стартует оптимизацию модели на удаленном сервере в синхронном режиме. Предварительно необходимо установить переменную окружения X_API_KEY.

param name_mapping_file
путь к файлу, который будет записан и будет содержать соответствие оригинальных имен модели и анонимизированных :return: объект с результатом оптимизации

solve_remote_async(*self*: [arhiplexpy.arhiplexpy.Model](#), *name_mapping_file*: *str*) → None

Стартует оптимизацию модели на удаленном сервере в асинхронном режиме без ожидания результата. Предварительно необходимо установить переменную окружения X_API_KEY.n

param name_mapping_file
путь к файлу, который будет записан и будет содержать соответствие оригинальных имен модели и анонимизированных

write(*self*: [arhiplexpy.arhiplexpy.Model](#), *file_name*: *str*, *name_mapping_file*: *str* = '') → None

Записывает модель в файл. Тип будет определен по расширению

param file_name
путь к записываемому файлу

```

    param name_mapping_file
        путь к файлу, который будет содержать соответствие между именами мо-
        дели при анонимизации (если пустой - запись без анонимизации)

write_lp(self: arhiplexpy.arhiplexpy.Model, file_name: str, name_mapping_file: str = '') → None
    Записывает модель в файл в формате LP

    param file_name
        путь к записываемому файлу

    param name_mapping_file
        путь к файлу, который будет содержать соответствие между именами мо-
        дели при анонимизации (если пустой - запись без анонимизации)

write_mps(self: arhiplexpy.arhiplexpy.Model, file_name: str, name_mapping_file: str = '') → None
    Записывает модель в файл в формате MPS

    param file_name
        путь к записываемому файлу

    param name_mapping_file
        путь к файлу, который будет содержать соответствие между именами мо-
        дели при анонимизации (если пустой - запись без анонимизации)

class arhiplexpy.Variable
    Работа с переменной - чтение/изменение имени, верхней и нижней границы, типа, а также уда-
    ление

    get_lower_bound(self: arhiplexpy.arhiplexpy.Variable) → float
        Получает нижнюю границу переменной

        return
            нижнюю границу переменной

    get_name(self: arhiplexpy.arhiplexpy.Variable) → str
        Получает имя переменной в модели

        return
            имя переменной

    get_type(self: arhiplexpy.arhiplexpy.Variable) → arhiplexpy.arhiplexpy.variable_type
        Получает тип переменной

        return
            тип переменной

    get_upper_bound(self: arhiplexpy.arhiplexpy.Variable) → float
        Получает верхнюю границу переменной

        return
            верхнюю границу переменной

    remove(self: arhiplexpy.arhiplexpy.Variable) → None
        Помечает переменную как удаленную. Переменная не будет участвовать в расчетах

    set_lower_bound(self: arhiplexpy.arhiplexpy.Variable, lower_bound: float) → None
        Задаёт нижнюю границу для переменной

        param lower_bound
            новая нижняя граница переменной

```

`set_name(self: arhiplexpy.arhiplexpy.Variable, var_name: str) → None`

Задаёт имя переменной в модели

param var_name

имя переменной

`set_type(self: arhiplexpy.arhiplexpy.Variable, type: arhiplexpy.arhiplexpy.variable_type) → None`

Задаёт новый тип переменной в модели

param type

новый тип переменной

`set_upper_bound(self: arhiplexpy.arhiplexpy.Variable, upper_bound: float) → None`

Задаёт верхнюю границу для переменной

param upper_bound

новая верхняя граница переменной

`class arhiplexpy.Constraint`

Работа с линейными ограничениями у модели,

позволяет работать с выражением и границами, а также удалять его

`get_linear_expression(self: arhiplexpy.arhiplexpy.Constraint) → arhiplex::LinearExpression`

Линейная часть выражения, связанного с ограничением

return

линейная часть выражения, связанного с ограничением

`get_lower_bound(self: arhiplexpy.arhiplexpy.Constraint) → float`

Получает нижнюю границу ограничения

return

нижнюю границу ограничения

`get_name(self: arhiplexpy.arhiplexpy.Constraint) → str`

Получает имя ограничения

return

имя ограничения

`get_quad_expression(self: arhiplexpy.arhiplexpy.Constraint) → arhiplex::QuadExpression`

Квадратичная часть выражения, связанного с ограничением

return

квадратичная часть выражения, связанного с ограничением

`get_range(self: arhiplexpy.arhiplexpy.Constraint) → float`

Получает интервал ограничения: upper_bound - lower_bound

return

upper_bound - lower_bound

`get_upper_bound(self: arhiplexpy.arhiplexpy.Constraint) → float`

Получает верхнюю границу ограничения

return

верхнюю границу ограничения

`remove(self: arhiplexpy.arhiplexpy.Constraint) → None`

Помечает ограничение как удаленное. Такое ограничение будет исключено из расчетов

`set_lower_bound(self: arhiplexpy.arhiplexpy.Constraint, lower_bound: float) → None`

Задаёт нижнюю границу ограничения

param lower_bound

новая нижняя граница ограничения

`set_name(self: arhiplexpy.arhiplexpy.Constraint, constr_name: str) → None`

Задаёт имя ограничения

param constr_name

имя ограничения

`set_range(self: arhiplexpy.arhiplexpy.Constraint, range: float) → None`

Задаёт интервал ограничения: $\text{lower_bound} \leq \text{expression} \leq \text{lower_bound} + \text{range}$

param range

интервал для ограничения

`set_upper_bound(self: arhiplexpy.arhiplexpy.Constraint, upper_bound: float) → None`

Задаёт верхнюю границу ограничения

param upper_bound

новая верхняя граница ограничения

`class arhiplexpy.LinearExpression`

Обеспечивает работу с линейными выражениями - добавление/удаление элементов и изменение коэффициентов. Элемент выражения - переменная * коэффициент.

`add_constant(self: arhiplexpy.arhiplexpy.LinearExpression, value: float) → None`

Добавляет константу к выражению

param

value значение константы

`add_expression(self: arhiplexpy.arhiplexpy.LinearExpression, expr: arhiplexpy.arhiplexpy.LinearExpression, mult: float) → None`

Добавляет выражение к выражению

param expr

добавляемое выражение

param mult

коэффициент к добавляемому выражению

`add_term(self: arhiplexpy.arhiplexpy.LinearExpression, var: arhiplexpy.arhiplexpy.Variable, coeff: float = 1.0) → None`

Добавляет элемент к выражению

param var

переменная

param coeff

коэффициент к переменной

`copy(self: arhiplexpy.arhiplexpy.LinearExpression) → arhiplexpy.arhiplexpy.LinearExpression`

Создаёт копию выражения, не привязанную к модели или другому ограничению

return

копия выражения

`empty(self: arhiplexpy.arhiplexpy.LinearExpression) → bool`

Возвращает истину, если выражение пустое

return
истина, если выражение пустое

`get_constant(self: arhiplexpy.arhiplexpy.LinearExpression) → float`

Возвращает константу в выражении

return
значение константы

`get_name(self: arhiplexpy.arhiplexpy.LinearExpression) → str`

Получает имя выражения

return
имя выражения

`get_quad_part(self: arhiplexpy.arhiplexpy.LinearExpression) → arhiplex::QuadExpression`

Возвращает квадратичную часть выражения

return
квадратичная часть выражения

`get_term_coeff(self: arhiplexpy.arhiplexpy.LinearExpression, ind: int) → float`

Возвращает коэффициент переменной по индексу элемента в выражении

param ind
индекс элемента в выражении

return
значение коэффициента переменной

`get_term_variable(self: arhiplexpy.arhiplexpy.LinearExpression, ind: int) →
arhiplexpy.arhiplexpy.Variable`

Возвращает переменную по индексу элемента в выражении

param ind
индекс элемента в выражении

return
объект переменной

`get_term_variable_idx(self: arhiplexpy.arhiplexpy.LinearExpression, ind: int) → int`

Возвращает порядковый индекс переменной в модели по индексу элемента в выражении

param ind
индекс элемента в выражении

return
индекс переменной

`get_terms_count(self: arhiplexpy.arhiplexpy.LinearExpression) → int`

Возвращает количество элементов в выражении

return
количество элементов в выражении

```
remove_term(self: arhiplexpy.arhiplexpy.LinearExpression, idx: int) → None
```

Удаляет элемент выражения по индексу

param idx

индекс удаляемого элемента в выражении

```
remove_variable(self: arhiplexpy.arhiplexpy.LinearExpression, var: arhiplexpy.arhiplexpy.Variable)
→ None
```

Удаляет переменную из выражения

param var

удаляемая переменная

```
set_constant(self: arhiplexpy.arhiplexpy.LinearExpression, value: float) → None
```

Задаёт константу в выражении

param value

значение константы

```
set_name(self: arhiplexpy.arhiplexpy.LinearExpression, expr_name: str) → None
```

Задаёт имя выражения

param expr_name

имя выражения

```
set_term_coeff(self: arhiplexpy.arhiplexpy.LinearExpression, ind: int, value: float) → None
```

Задаёт коэффициент переменной по индексу выражения

param ind

индекс элемента в выражении

param value

значение коэффициента при переменной в элементе

```
class arhiplexpy.QuadExpression
```

Обеспечивает работу с нелинейными выражениями - добавление/удаление элементов и изменение коэффициентов. Элемент выражения - переменная * переменная * коэффициент.

```
add_expression(*args, **kwargs)
```

Overloaded function.

```
1. add_expression(self: arhiplexpy.arhiplexpy.QuadExpression, expr:
arhiplexpy.arhiplexpy.QuadExpression, mult: float) -> None
```

Добавляет выражение к выражению

param expr

добавляемое выражение

param mult

коэффициент к добавляемому выражению

```
2. add_expression(self: arhiplexpy.arhiplexpy.QuadExpression, expr:
arhiplexpy.arhiplexpy.LinearExpression, mult: float) -> None
```

Добавляет выражение к выражению

param expr

добавляемое выражение

param mult

коэффициент к добавляемому выражению

`add_term(self: arhiplexpy.arhiplexpy.QuadExpression, var1: arhiplexpy.arhiplexpy.Variable, var2: arhiplexpy.arhiplexpy.Variable, coeff: float = 1.0) → None`

Добавляет элемент к выражению

param var1

переменная 1

param var2

переменная 2

param coeff

коэффициент

`copy(self: arhiplexpy.arhiplexpy.QuadExpression) → arhiplexpy.arhiplexpy.QuadExpression`

Создает копию выражения, не привязанную к модели или другому ограничению

return

копия выражения

`empty(self: arhiplexpy.arhiplexpy.QuadExpression) → bool`

Возвращает истину, если выражение пустое

return

истина, если выражение пустое

`get_linear_part(self: arhiplexpy.arhiplexpy.QuadExpression) →`

arhiplexpy.arhiplexpy.LinearExpression

Получает линейную часть выражения

return

линейная часть выражения

`get_name(self: arhiplexpy.arhiplexpy.QuadExpression) → str`

Получает имя выражения

return

имя выражения

`get_term_coeff(self: arhiplexpy.arhiplexpy.QuadExpression, ind: int) → float`

Возвращает коэффициент по индексу элемента в выражении

param ind

индекс элемента в выражении

return

значение коэффициента

`get_term_variable1(self: arhiplexpy.arhiplexpy.QuadExpression, ind: int) →`

arhiplexpy.arhiplexpy.Variable

Возвращает первую переменную по индексу элемента в выражении

param ind

индекс элемента в выражении

return

объект переменной

`get_term_variable2(self: arhiplexpy.arhiplexpy.QuadExpression, ind: int) →`
arhiplexpy.arhiplexpy.Variable

Возвращает вторую переменную по индексу элемента в выражении

param ind
индекс элемента в выражении

return
объект переменной

`get_terms_count(self: arhiplexpy.arhiplexpy.QuadExpression) → int`

Возвращает количество элементов в выражении

return
количество элементов в выражении

`remove_term(self: arhiplexpy.arhiplexpy.QuadExpression, idx: int) → None`

Удаляет элемент выражения по индексу

param idx
индекс удаляемого элемента в выражении

`set_name(self: arhiplexpy.arhiplexpy.QuadExpression, expr_name: str) → None`

Задаёт имя выражения

param expr_name
имя выражения

`set_term_coeff(self: arhiplexpy.arhiplexpy.QuadExpression, ind: int, value: float) → None`

Задаёт коэффициент по индексу выражения

param ind
индекс элемента в выражении
param value
значение коэффициента в элементе

`class arhiplexpy.GeneralExpression`

Выражение общего вида

`add_expr(self: arhiplexpy.arhiplexpy.GeneralExpression, expr: arhiplexpy.arhiplexpy.GeneralExpression) → None`

Добавляет выражение, для типов plus и multiply

param expr
добавляемое выражение

`copy(self: arhiplexpy.arhiplexpy.GeneralExpression) → arhiplexpy.arhiplexpy.GeneralExpression`

Создаёт копию выражения, не привязанную к модели или другому ограничению

return
копия выражения

`get_arity(self: arhiplexpy.arhiplexpy.GeneralExpression) → int`

Возвращает арность выражения

return
арность выражения

`get_constant(self: arhiplexpy.arhiplexpy.GeneralExpression) → float`

Возвращает значение константы для выражения с типом constant

return

значение константы

`get_expr(self: arhiplexpy.arhiplexpy.GeneralExpression, idx: int) → arhiplexpy.arhiplexpy.GeneralExpression`

Возвращает по индексу содержащееся выражение

param idx

индекс получаемого выражения

return

выражение

`get_expr_count(self: arhiplexpy.arhiplexpy.GeneralExpression) → int`

Возвращает количество элементов в выражении

return

количество элементов в выражении

`get_type(self: arhiplexpy.arhiplexpy.GeneralExpression) → arhiplexpy.arhiplexpy.gen_expr_type`

Возвращает тип выражения

return

тип выражения

`get_variable(self: arhiplexpy.arhiplexpy.GeneralExpression) → arhiplexpy.arhiplexpy.Variable`

Возвращает переменную для выражения с типом variable

return

объект переменной

`set_expr(self: arhiplexpy.arhiplexpy.GeneralExpression, idx: int, expr: arhiplexpy.arhiplexpy.GeneralExpression) → None`

Задаёт выражение по индексу

param idx

индекс выражения

param expr

выражение

`class arhiplexpy.GeneralConstraint`

Работа с нелинейными ограничениями у модели

`get_expression(self: arhiplexpy.arhiplexpy.GeneralConstraint) → arhiplexpy.GeneralExpression`

Выражение, связанное с ограничением

return

выражение, связанное с ограничением

`get_name(self: arhiplexpy.arhiplexpy.GeneralConstraint) → str`

Получает имя ограничения

return

имя ограничения

`get_variable(self: arhiplexpy.arhiplexpy.GeneralConstraint) → arhiplex::Variable`

Переменная, связанная с ограничением

return

переменная, связанная с ограничением

`remove(self: arhiplexpy.arhiplexpy.GeneralConstraint) → None`

Помечает ограничение как удаленное. Такое ограничение будет исключено из расчетов

`set_name(self: arhiplexpy.arhiplexpy.GeneralConstraint, constr_name: str) → None`

Задаёт имя ограничения

param constr_name

имя ограничения

`class arhiplexpy.SolveResult`

Результат расчета

`get_best_bound(self: arhiplexpy.arhiplexpy.SolveResult) → float`

Получает значение граничной (двойственной) функции

return

значение граничной (двойственной) функции

`get_dual_value(*args, **kwargs)`

Overloaded function.

1. `get_dual_value(self: arhiplexpy.arhiplexpy.SolveResult, constraint: arhiplex::Constraint) -> float`

Получает значение dual value

param constraint

ограничение

return

значение dual value в решении

2. `get_dual_value(self: arhiplexpy.arhiplexpy.SolveResult, constraint_name: str) -> float`

Получает значение dual value по имени ограничения

param constraint_name

имя ограничения

return

значение dual value в решении

`get_expression_value(*args, **kwargs)`

Overloaded function.

1. `get_expression_value(self: arhiplexpy.arhiplexpy.SolveResult, arhiplex::QuadExpression) -> float` arg0:

Получает значение выражения

param expr

объект выражения

return

значение выражения

2. `get_expression_value(self: arhiplexpy.arhiplexpy.SolveResult, arhiplex::LinearExpression) -> float` arg0:

Получает значение выражения

param expr
объект выражения

return
значение выражения

`get_iterations_count(self: arhiplexpy.arhiplexpy.SolveResult) → int`

Получает количество итераций, проведенных в процессе расчета

return
количество итераций

`get_nodes_count(self: arhiplexpy.arhiplexpy.SolveResult) → int`

Получает количество обработанных узлов дерева решений

return
количество обработанных узлов

`get_objective_value(self: arhiplexpy.arhiplexpy.SolveResult) → float`

Получает значение целевой функции

return
значение целевой функции

`get_reduced_cost(*args, **kwargs)`

Overloaded function.

1. `get_reduced_cost(self: arhiplexpy.arhiplexpy.SolveResult, var_name: str) -> float`

Получает значение reduced cost в решении по имени

param var_name
имя переменной

return
значение reduced cost в решении

2. `get_reduced_cost(self: arhiplexpy.arhiplexpy.SolveResult, variable: arhiplex::Variable) -> float`

Получает значение reduced cost в решении

param variable
переменная

return
значение reduced cost в решении

`get_relative_gap(self: arhiplexpy.arhiplexpy.SolveResult) → float`

Получает точность решения в процентах

return
точность решения в процентах

```
get_solution_status(self: arhiplexpy.arhiplexpy.SolveResult) →
    arhiplexpy.arhiplexpy.solution_status
```

Получает статус модели в итоге расчета

```
return
    значение статуса
```

```
get_solve_result(self: arhiplexpy.arhiplexpy.SolveResult) → arhiplexpy.arhiplexpy.solve_result
```

Получает статус процесса расчетов

```
return
    значение статуса
```

```
get_solve_time(self: arhiplexpy.arhiplexpy.SolveResult) → float
```

Получает общее время решения

```
return
    общее время решения [сек]
```

```
get_variable_value(*args, **kwargs)
```

Overloaded function.

1. get_variable_value(self: arhiplexpy.arhiplexpy.SolveResult, var: arhiplex::Variable) -> float

Получает значение переменной в решении

```
param var
    объект переменной

return
    значение переменной в решении
```

2. get_variable_value(self: arhiplexpy.arhiplexpy.SolveResult, var_name: str) -> float

Получает значение переменной в решении по имени

```
param var_name
    имя переменной

return
    значение переменной в решении
```

```
write_solution(self: arhiplexpy.arhiplexpy.SolveResult, file_name: str) → None
```

Записывает решение в файл

```
param file_name
    путь к файлу решения для записи
```

```
class arhiplexpy.MatrixModel
```

Создает экземпляр матричной модели

```
add_constraint(self: arhiplexpy.arhiplexpy.MatrixModel, P: object = None, q: object = None, sense:
    arhiplexpy.arhiplexpy.constraint_sense, r: float) → None
```

Добавляет ограничение в модель с заданными параметрами

```
param P
    матрица квадратичных коэффициентов

param q
    вектор коэффициентов переменных
```

param sense
тип ограничения ($\leq$, $\geq$, $=$)

param r
константное значение в правой части ограничения

add_constraints(*self*: arhiplexpy.arhiplexpy.MatrixModel, *A*: object, *senses*: object, *b*: object) → None

Добавляет ограничения в модель с заданными параметрами

param A
матрица ограничений в формате csr (sparse row matrix)

param senses
массив типов ограничений (constraint_sense)

param b
массив константных значений в правых частях ограничений

add_variables(*self*: arhiplexpy.arhiplexpy.MatrixModel, *lower_bounds*: object = 0.0, *upper_bounds*: object = inf, *obj_values*: object = 0.0, *var_types*: object = <variable_type.continuous: 0>, *count*: int) → None

Добавляет переменные в модель с заданными параметрами

param lower_bounds
нижние границы переменных

param upper_bounds
верхние границы переменных

param obj_values
коэффициенты к переменным в целевой функции

param var_types
типы переменных

param count
количество переменных

clear(*self*: arhiplexpy.arhiplexpy.MatrixModel) → None

Очищает модель от всех данных. Все старые переменные и ограничения становятся невалидными

get_bool_param(*self*: arhiplexpy.arhiplexpy.MatrixModel, *param_name*: str) → bool

Получает логический параметр

param param_name
имя параметра

return
значение параметра

getdbl_param(*self*: arhiplexpy.arhiplexpy.MatrixModel, *param_name*: str) → float

Получает параметр с плавающей точкой

param param_name
имя параметра

return
значение параметра

`get_int_param(self: arhiplexpy.arhiplexpy.MatrixModel, param_name: str) → int`

Получает целочисленный параметр

param param_name
имя параметра

return
значение параметра

`get_string_param(self: arhiplexpy.arhiplexpy.MatrixModel, param_name: str) → str`

Получает строковый параметр

param param_name
имя параметра

return
значение параметра

`get_variables_count(self: arhiplexpy.arhiplexpy.MatrixModel) → int`

Получает количество переменных в модели

return
количество переменных в модели

`set_bool_param(self: arhiplexpy.arhiplexpy.MatrixModel, param_name: str, param_value: bool) → None`

Задаёт логический параметр

param param_name
имя параметра

param param_value
значение параметра

`set_dbl_param(self: arhiplexpy.arhiplexpy.MatrixModel, param_name: str, param_value: float) → None`

Задаёт параметр с плавающей точкой

param param_name
имя параметра

param param_value
значение параметра

`set_int_param(self: arhiplexpy.arhiplexpy.MatrixModel, param_name: str, param_value: int) → None`

Задаёт целочисленный параметр

param param_name
имя параметра

param param_value
значение параметра

`set_log_file(self: arhiplexpy.arhiplexpy.MatrixModel, log_file: str) → None`

Задать файл для записи лога решения

param log_file
файл лога

`set_objective(self: arhiplexpy.arhiplexpy.MatrixModel, P: object = None, q: object = None, sense: arhiplexpy.arhiplexpy.objective_sense = <objective_sense.minimize: 0>) → None`

Задаёт выражение целевой функции и тип оптимизации

param P

квадратичная часть выражения

param q

вектор коэффициентов к переменным

param sense

тип оптимизации

`set_objective_offset(self: arhiplexpy.arhiplexpy.MatrixModel, offset: float) → None`

Задаёт константу в выражении целевой функции

param offset

значение константы

`set_string_param(self: arhiplexpy.arhiplexpy.MatrixModel, param_name: str, param_value: str) → None`

Задаёт строковый параметр

param param_name

имя параметра

param param_value

значение параметра

`solve(self: arhiplexpy.arhiplexpy.MatrixModel) → arhiplex::MatrixSolveResult`

Стартует оптимизацию модели

return

объект с результатом оптимизации

`solve_remote(self: arhiplexpy.arhiplexpy.MatrixModel) → arhiplex::MatrixSolveResult`

Стартует оптимизацию модели на удаленном сервере в синхронном режиме. Предварительно необходимо установить переменную окружения X_API_KEY.

return

объект с результатом оптимизации

`write(self: arhiplexpy.arhiplexpy.MatrixModel, file_name: str) → None`

Записывает модель в файл. Тип будет определен по расширению

param file_name

путь к записываемому файлу

`write_lp(self: arhiplexpy.arhiplexpy.MatrixModel, file_name: str) → None`

Записывает модель в файл в формате LP

param file_name

путь к записываемому файлу

`write_mps(self: arhiplexpy.arhiplexpy.MatrixModel, file_name: str) → None`

Записывает модель в файл в формате MPS

param file_name

путь к записываемому файлу

```
class arhiplexpy.MatrixSolveResult
```

Результат расчета матричной модели

```
get_best_bound(self: arhiplexpy.arhiplexpy.MatrixSolveResult) → float
```

Получает значение граничной (двойственной) функции

```
    return
```

значение граничной (двойственной) функции

```
get_dual_values_vector(self: arhiplexpy.arhiplexpy.MatrixSolveResult) →  
    numpy.ndarray[numpy.float64]
```

Получает dual values вектор

```
    return
```

вектор dual values

```
get_iterations_count(self: arhiplexpy.arhiplexpy.MatrixSolveResult) → int
```

Получает количество итераций, проведенных в процессе расчета

```
    return
```

количество итераций

```
get_nodes_count(self: arhiplexpy.arhiplexpy.MatrixSolveResult) → int
```

Получает количество обработанных узлов дерева решений

```
    return
```

количество обработанных узлов

```
get_objective_value(self: arhiplexpy.arhiplexpy.MatrixSolveResult) → float
```

Получает значение целевой функции

```
    return
```

значение целевой функции

```
get_reduced_costs_vector(self: arhiplexpy.arhiplexpy.MatrixSolveResult) →  
    numpy.ndarray[numpy.float64]
```

Получает reduced costs вектор

```
    return
```

вектор reduced costs

```
get_relative_gap(self: arhiplexpy.arhiplexpy.MatrixSolveResult) → float
```

Получает точность решения в процентах

```
    return
```

точность решения в процентах

```
get_solution_status(self: arhiplexpy.arhiplexpy.MatrixSolveResult) →  
    arhiplexpy.arhiplexpy.solution_status
```

Получает статус модели в итоге расчета

```
    return
```

значение статуса

```
get_solution_vector(self: arhiplexpy.arhiplexpy.MatrixSolveResult) →  
    numpy.ndarray[numpy.float64]
```

Получает вектор значений переменных

```
    return
```

вектор переменных

```
get_solve_result(self: arhiplexpy.arhiplexpy.MatrixSolveResult) →
    arhiplexpy.arhiplexpy.solve_result
```

Получает статус процесса расчетов

```
return
    значение статуса
```

```
get_solve_time(self: arhiplexpy.arhiplexpy.MatrixSolveResult) → float
```

Получает общее время решения

```
return
    общее время решения [сек]
```

```
write_solution(self: arhiplexpy.arhiplexpy.MatrixSolveResult, file_name: str) → None
```

Записывает решение в файл

```
param file_name
    путь к файлу решения для записи
```

```
class arhiplexpy.MathFunc
```

Математические функции для нелинейных выражений

```
cos(*args, **kwargs)
```

Overloaded function.

1. cos(self: float) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейcos

```
Результат
    выражение с функциейcos
```

2. cos(self: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейcos

```
Результат
    выражение с функциейcos
```

3. cos(self: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейcos

```
Результат
    выражение с функциейcos
```

4. cos(self: arhiplexpy.arhiplexpy.QuadExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейcos

```
Результат
    выражение с функциейcos
```

5. cos(self: arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейcos

```
Результат
    выражение с функциейcos
```

`exp(*args, **kwargs)`

Overloaded function.

1. `exp(self: float) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `exp`

Результат

выражение с функцией `exp`

2. `exp(self: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `exp`

Результат

выражение с функцией `exp`

3. `exp(self: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `exp`

Результат

выражение с функцией `exp`

4. `exp(self: arhiplexpy.arhiplexpy.QuadExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `exp`

Результат

выражение с функцией `exp`

5. `exp(self: arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `exp`

Результат

выражение с функцией `exp`

`log(*args, **kwargs)`

Overloaded function.

1. `log(self: float) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `log`

Результат

выражение с функцией `log`

2. `log(self: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `log`

Результат

выражение с функцией `log`

3. `log(self: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `log`

Результат

выражение с функциейlog

4. log(self: arhiplexpy.arhiplexpy.QuadExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейlog

Результат

выражение с функциейlog

5. log(self: arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейlog

Результат

выражение с функциейlog

log10(*args, **kwargs)

Overloaded function.

1. log10(self: float) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейlog10

Результат

выражение с функциейlog10

2. log10(self: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейlog10

Результат

выражение с функциейlog10

3. log10(self: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейlog10

Результат

выражение с функциейlog10

4. log10(self: arhiplexpy.arhiplexpy.QuadExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейlog10

Результат

выражение с функциейlog10

5. log10(self: arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейlog10

Результат

выражение с функциейlog10

`log2(*args, **kwargs)`

Overloaded function.

1. `log2(self: float) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `log2`

Результат

выражение с функцией `log2`

2. `log2(self: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `log2`

Результат

выражение с функцией `log2`

3. `log2(self: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `log2`

Результат

выражение с функцией `log2`

4. `log2(self: arhiplexpy.arhiplexpy.QuadExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `log2`

Результат

выражение с функцией `log2`

5. `log2(self: arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `log2`

Результат

выражение с функцией `log2`

`logistic(*args, **kwargs)`

Overloaded function.

1. `logistic(self: float) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `logistic`

Результат

выражение с функцией `logistic`

2. `logistic(self: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `logistic`

Результат

выражение с функцией `logistic`

3. `logistic(self: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `logistic`

Результат

выражение с функцией `logistic`

4. `logistic(self: arhiplexpy.arhiplexpy.QuadExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `logistic`

Результат

выражение с функцией `logistic`

5. `logistic(self: arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `logistic`

Результат

выражение с функцией `logistic`

`pow(*args, **kwargs)`

Overloaded function.

1. `pow(self: float, arg0: float) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `row`

Результат

выражение с функцией `row`

2. `pow(self: float, arg0: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `row`

Результат

выражение с функцией `row`

3. `pow(self: float, arg0: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `row`

Результат

выражение с функцией `row`

4. `pow(self: float, arg0: arhiplexpy.arhiplexpy.QuadExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `row`

Результат

выражение с функцией `row`

5. `pow(self: float, arg0: arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `row`

Результат

выражение с функцией `row`

6. `pow(self: arhiplexpy.arhiplexpy.Variable, arg0: float) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функциейrow

Результат

выражение с функциейrow

7. pow(self: arhiplexpy.arhiplexpy.Variable, arg0: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

8. pow(self: arhiplexpy.arhiplexpy.Variable, arg0: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

9. pow(self: arhiplexpy.arhiplexpy.Variable, arg0: arhiplexpy.arhiplexpy.QuadExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

10. pow(self: arhiplexpy.arhiplexpy.Variable, arg0: arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

11. pow(self: arhiplexpy.arhiplexpy.LinearExpression, arg0: float) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

12. pow(self: arhiplexpy.arhiplexpy.LinearExpression, arg0: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

13. pow(self: arhiplexpy.arhiplexpy.LinearExpression, arg0: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

14. pow(self: arhiplexpy.arhiplexpy.LinearExpression, arg0: arhiplexpy.arhiplexpy.QuadExpression)
-> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

15. pow(self: arhiplexpy.arhiplexpy.LinearExpression, arg0: arhiplexpy.arhiplexpy.GeneralExpression)
-> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

16. pow(self: arhiplexpy.arhiplexpy.QuadExpression, arg0: float)
arhiplexpy.arhiplexpy.GeneralExpression ->

Получает выражение с функциейrow

Результат

выражение с функциейrow

17. pow(self: arhiplexpy.arhiplexpy.QuadExpression, arg0: arhiplexpy.arhiplexpy.Variable) ->
arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

18. pow(self: arhiplexpy.arhiplexpy.QuadExpression, arg0: arhiplexpy.arhiplexpy.LinearExpression)
-> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

19. pow(self: arhiplexpy.arhiplexpy.QuadExpression, arg0: arhiplexpy.arhiplexpy.QuadExpression)
-> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

20. pow(self: arhiplexpy.arhiplexpy.QuadExpression, arg0: arhiplexpy.arhiplexpy.GeneralExpression)
-> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

21. pow(self: arhiplexpy.arhiplexpy.GeneralExpression, arg0: float) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

22. pow(self: arhiplexpy.arhiplexpy.GeneralExpression, arg0: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

23. pow(self: arhiplexpy.arhiplexpy.GeneralExpression, arg0: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

24. pow(self: arhiplexpy.arhiplexpy.GeneralExpression, arg0: arhiplexpy.arhiplexpy.QuadExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

25. pow(self: arhiplexpy.arhiplexpy.GeneralExpression, arg0: arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейrow

Результат

выражение с функциейrow

signpow(*args, **kwargs)

Overloaded function.

1. signpow(self: float, arg0: float) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

2. signpow(self: float, arg0: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

3. signpow(self: float, arg0: arhiplexpy.arhiplexpy.LinearExpression) ->
arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

4. signpow(self: float, arg0: arhiplexpy.arhiplexpy.QuadExpression) ->
arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

5. signpow(self: float, arg0: arhiplexpy.arhiplexpy.GeneralExpression) ->
arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

6. signpow(self: arhiplexpy.arhiplexpy.Variable, arg0: float) ->
arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

7. signpow(self: arhiplexpy.arhiplexpy.Variable, arg0: arhiplexpy.arhiplexpy.Variable) ->
arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

8. signpow(self: arhiplexpy.arhiplexpy.Variable, arg0: arhiplexpy.arhiplexpy.LinearExpression) ->
arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

9. signpow(self: arhiplexpy.arhiplexpy.Variable, arg0: arhiplexpy.arhiplexpy.QuadExpression) ->
arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

10. signpow(self: arhiplexpy.arhiplexpy.Variable, arg0: arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

11. signpow(self: arhiplexpy.arhiplexpy.LinearExpression, arg0: float) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

12. signpow(self: arhiplexpy.arhiplexpy.LinearExpression, arg0: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

13. signpow(self: arhiplexpy.arhiplexpy.LinearExpression, arg0: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

14. signpow(self: arhiplexpy.arhiplexpy.LinearExpression, arg0: arhiplexpy.arhiplexpy.QuadExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

15. signpow(self: arhiplexpy.arhiplexpy.LinearExpression, arg0: arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

16. signpow(self: arhiplexpy.arhiplexpy.QuadExpression, arg0: float) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

17. signpow(self: arhiplexpy.arhiplexpy.QuadExpression, arg0: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

18. signpow(self: arhiplexpy.arhiplexpy.QuadExpression, arg0: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

19. signpow(self: arhiplexpy.arhiplexpy.QuadExpression, arg0: arhiplexpy.arhiplexpy.QuadExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

20. signpow(self: arhiplexpy.arhiplexpy.QuadExpression, arg0: arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

21. signpow(self: arhiplexpy.arhiplexpy.GeneralExpression, arg0: float) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

22. signpow(self: arhiplexpy.arhiplexpy.GeneralExpression, arg0: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

23. signpow(self: arhiplexpy.arhiplexpy.GeneralExpression, arg0: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

24. signpow(self: arhiplexpy.arhiplexpy.GeneralExpression, arg0:
arhiplexpy.arhiplexpy.QuadExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

25. signpow(self: arhiplexpy.arhiplexpy.GeneralExpression, arg0:
arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsignpow

Результат

выражение с функциейsignpow

`sin(*args, **kwargs)`

Overloaded function.

1. sin(self: float) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsin

Результат

выражение с функциейsin

2. sin(self: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsin

Результат

выражение с функциейsin

3. sin(self: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsin

Результат

выражение с функциейsin

4. sin(self: arhiplexpy.arhiplexpy.QuadExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsin

Результат

выражение с функциейsin

5. sin(self: arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsin

Результат

выражение с функциейsin

`sqr(*args, **kwargs)`

Overloaded function.

1. `sqr(self: float) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `sqr`

Результат

выражение с функцией `sqr`

2. `sqr(self: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `sqr`

Результат

выражение с функцией `sqr`

3. `sqr(self: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `sqr`

Результат

выражение с функцией `sqr`

4. `sqr(self: arhiplexpy.arhiplexpy.QuadExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `sqr`

Результат

выражение с функцией `sqr`

5. `sqr(self: arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `sqr`

Результат

выражение с функцией `sqr`

`sqrt(*args, **kwargs)`

Overloaded function.

1. `sqrt(self: float) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `sqrt`

Результат

выражение с функцией `sqrt`

2. `sqrt(self: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `sqrt`

Результат

выражение с функцией `sqrt`

3. `sqrt(self: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `sqrt`

Результат

выражение с функциейsqrt

4. sqrt(self: arhiplexpy.arhiplexpy.QuadExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsqrt

Результат

выражение с функциейsqrt

5. sqrt(self: arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейsqrt

Результат

выражение с функциейsqrt

`tan(*args, **kwargs)`

Overloaded function.

1. tan(self: float) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейtan

Результат

выражение с функциейtan

2. tan(self: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейtan

Результат

выражение с функциейtan

3. tan(self: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейtan

Результат

выражение с функциейtan

4. tan(self: arhiplexpy.arhiplexpy.QuadExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейtan

Результат

выражение с функциейtan

5. tan(self: arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression

Получает выражение с функциейtan

Результат

выражение с функциейtan

`tanh(*args, **kwargs)`

Overloaded function.

1. `tanh(self: float) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `tanh`

Результат

выражение с функцией `tanh`

2. `tanh(self: arhiplexpy.arhiplexpy.Variable) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `tanh`

Результат

выражение с функцией `tanh`

3. `tanh(self: arhiplexpy.arhiplexpy.LinearExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `tanh`

Результат

выражение с функцией `tanh`

4. `tanh(self: arhiplexpy.arhiplexpy.QuadExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `tanh`

Результат

выражение с функцией `tanh`

5. `tanh(self: arhiplexpy.arhiplexpy.GeneralExpression) -> arhiplexpy.arhiplexpy.GeneralExpression`

Получает выражение с функцией `tanh`

Результат

выражение с функцией `tanh`

17.5.1 ArhiPlex CP API

Примечание: В данной сборке документации ArhiPlex CP не был включён (`BUILD_CP_SOLVER=OFF`). Это не означает, что ArhiPlex CP отсутствует во всех дистрибутивах ArhiPlex.

Проверьте доступность в вашей установке:

```
import arhiplexpy
print(arhiplexpy.has_cp_solver())
```

Если вывод `True`, импортируйте `from arhiplexpy.arhitools.sat.python import cp_model`. Подробнее см. *CP-решатель (ArhiPlex CP)*.

C++ API предоставляет объектный интерфейс к решателю: построение модели из переменных, ограничений и целевой функции, настройку параметров, запуск расчёта и чтение результата. Публичный фасад — класс `arhiplex::Model` и связанные типы (`Variable`, `Constraint`, `LinearExpression`, `QuadExpression`, `SolveResult`).

18.1 Подключение

Linux (основной вариант). Солвер должен быть установлен, как описано в разделе [Linux](#). Для сборки укажите каталоги заголовков и библиотеки и подключите `arhiplex_cpp`:

```
cd "${ARHIPLEX_HOME}/examples"
g++ example1.cpp -I "${ARHIPLEX_HOME}/include/arhiplex/" \
-L "${ARHIPLEX_HOME}/lib/" -larhiplex_cpp
```

В исходном файле подключается основной заголовок:

```
#include "arhiplex_client.h"
```

Примечание: В Windows C++ API поставляется в составе удалённого клиента (ArhiPlex Remote Client): заголовки и библиотека `libarhiplex_cpp.lib` подключаются к проекту (например, в Visual Studio 2022 — через «Компоновщик» -> «Ввод» -> «Дополнительные зависимости»), а расчёт выполняется на сервере. Полноценная локальная установка солвера поддерживается на Linux (см. [Установка](#)).

18.2 Сквозной пример

Рассмотрим задачу:

$$\begin{aligned}
 &\textbf{Maximize:} \\
 &x_1 + 2x_2 - 0.1x_3 - 3x_4 \\
 &\textbf{Subject to:} \\
 &x_1 + x_2 \leq 5 \\
 &2x_1 - x_2 \geq 0 \\
 &-x_1 + 3x_2 \geq 0 \\
 &x_3 + x_4 \geq 0.5 \\
 &\textbf{Bounds:} \\
 &-\infty < x_1 < +\infty \\
 &-\infty < x_2 < +\infty \\
 &1.1 \leq x_3 \leq 10.0 \\
 &-\infty < x_4 < +\infty
 \end{aligned} \tag{18.1}$$

Полный исходный код (файл `example1.cpp` из каталога `examples` установленного солвера):

```

1  #include "arhiplex_client.h"
2  #include <iostream>
3
4  // The problem to solve:
5  //
6  // Maximize:
7  //   x1 + 2 * x2 - 0.1 * x3 - 3 * x4
8  //
9  // Subject to:
10 //   c0: x1 + x2 <= 5
11 //   c1: 2 * x1 - x2 >= 0
12 //   c2: -x1 + 3 * x2 >= 0
13 //   c3: x3 + x4 >= 0.5
14 //
15 // where:
16 //   x1 Free
17 //   x2 Free
18 //   1.1 <= x3 <= 10.0
19 //   x4 Free
20
21 int main(int argc, char *argv[])
22 {
23     try
24     {
25         setlocale(LC_CTYPE, "Russian");
26         using namespace arhiplex;
27         arhiplex::Model model;
28         //Как пример - для данной простой модели не актуально
29         model.SetDb1Param("time_limit", 10.0); //10 сек
30         model.SetDb1Param("mip_rel_gap", 0.0005); //0.05%
31
32         // Добавление переменных
33         auto x1 = model.AddVariable(-arhiplex::inf, arhiplex::inf, 0.0,

```

(continues on next page)

(продолжение с предыдущей страницы)

```

34     variable_type::continuous, "x1");
35     auto x2 = model.AddVariable(-arhiplex::inf, arhiplex::inf, 0.0,
36         variable_type::continuous, "x2");
37     auto x3 = model.AddVariable(0.0, arhiplex::inf, 0.0,
38         variable_type::continuous, "x3");
39     auto x4 = model.AddVariable(-arhiplex::inf, arhiplex::inf, 0.0,
40         variable_type::continuous, "x4");
41
42     x3.SetLowerBound(1.1);
43     x3.SetUpperBound(10.0);
44     // или сразу
45     // auto x3 = model.AddVariable(1.1, 10.0, 0.0,
46     //     variable_type::continuous, "x3");
47
48     model.SetObjective(x1 + 2 * x2 - 0.1 * x3 - 3 * x4,
49         objective_sense::maximize);
50
51     model.AddConstraint(x1 + x2 <= 5, "c0");
52     model.AddConstraint(2 * x1 - x2 >= 0, "c1");
53     model.AddConstraint(-x1 + 3 * x2 >= 0, "c2");
54     model.AddConstraint(x3 + x4 >= 0.5, "c3");
55
56     auto solve_result = model.Solve();
57
58     auto solve_time = solve_result.GetSolveTime();
59     auto solve_res = solve_result.GetSolveResult();
60     auto solution_status = solve_result.GetSolutionStatus();
61
62     if (solve_res == solve_result::success &&
63         (solution_status == solution_status::optimal ||
64          solution_status == solution_status::feasible))
65     {
66         auto gap = solve_result.GetRelativeGap();
67         auto x1val = solve_result.GetVariableValue(x1);
68         auto x2val = solve_result.GetVariableValue("x2");
69         auto x3val = solve_result.GetVariableValue("x3");
70         auto x4val = solve_result.GetVariableValue(x4);
71         std::cout << "Gap: " << gap << "\n";
72         std::cout << "Variables:\n" << "x1: " << x1val <<
73             "\nx2: " << x2val << "\nx3: " << x3val <<
74             "\nx4: " << x4val << "\n";
75     }
76 }
77 catch (const arhiplex::arhiplex_exception &ex)
78 {
79     std::cout << ex.what() << std::endl;
80 }
81 catch (...)
82 {
83     std::cout << "Unknown exception occurs!";
84 }
85 return 0;

```

(continues on next page)

(продолжение с предыдущей страницы)

86

}

Разберём ключевые шаги. Создание модели:

```
arhiplex::Model model;
```

Настройка параметров (полный список — в разделе *Параметры расчёта*):

```
model.SetDbiParam("time_limit", 10.0); //10 сек
model.SetDbiParam("mip_rel_gap", 0.0005); //0.05%
```

Добавление переменных методом `AddVariable()`. Первый и второй аргументы — нижняя и верхняя границы, третий — коэффициент в целевой функции, четвёртый — тип переменной, последний — имя:

```
// Добавление переменных
auto x1 = model.AddVariable(-arhiplex::inf, arhiplex::inf, 0.0,
    variable_type::continuous, "x1");
auto x2 = model.AddVariable(-arhiplex::inf, arhiplex::inf, 0.0,
    variable_type::continuous, "x2");
auto x3 = model.AddVariable(0.0, arhiplex::inf, 0.0,
    variable_type::continuous, "x3");
auto x4 = model.AddVariable(-arhiplex::inf, arhiplex::inf, 0.0,
    variable_type::continuous, "x4");
```

Границы можно изменить позже (как здесь для x_3):

```
x3.SetLowerBound(1.1);
x3.SetUpperBound(10.0);
```

Целевая функция задаётся методом `SetObjective()` (выражение и направление оптимизации):

```
model.SetObjective(x1 + 2 * x2 - 0.1 * x3 - 3 * x4,
    objective_sense::maximize);
```

Ограничения добавляются методом `AddConstraint()` (само ограничение и его имя):

```
model.AddConstraint(x1 + x2 <= 5, "c0");
model.AddConstraint(2 * x1 - x2 >= 0, "c1");
model.AddConstraint(-x1 + 3 * x2 >= 0, "c2");
model.AddConstraint(x3 + x4 >= 0.5, "c3");
```

Запуск расчёта:

```
auto solve_result = model.Solve();
```

Метод `Solve` возвращает результат с данными расчёта — например, относительный *разрыв*:

```
auto gap = solve_result.GetRelativeGap();
```

Предупреждение: Прежде чем читать значения переменных, убедитесь, что расчёт успешен и модель достижима (см. *Интерпретация результата*).

```

if (solve_res == solve_result::success &&
    (solution_status == solution_status::optimal ||
     solution_status == solution_status::feasible))
{
    auto gap = solve_result.GetRelativeGap();
    auto x1val = solve_result.GetVariableValue(x1);
    auto x2val = solve_result.GetVariableValue("x2");
    auto x3val = solve_result.GetVariableValue("x3");
    auto x4val = solve_result.GetVariableValue(x4);
    std::cout << "Gap: " << gap << "\n";
    std::cout << "Variables:\n" << "x1: " << x1val <<
        "\nx2: " << x2val << "\nx3: " << x3val <<
        "\nx4: " << x4val << "\n";
}

```

Значение переменной методом `GetVariableValue()` можно получить как по объекту переменной, так и по её имени.

18.3 Справочник API

Полный справочник классов и перечислений формируется автоматически из исходных комментариев (Doxygen).

class `arhiplex_exception` : public `runtime_error`

arhiplex-исключение

Исключение, генерируемое всеми классами в случае ошибки

Public Functions

inline int `get_error_code()` const

Код ошибки arhiplex. Как правило, всегда -1

class `Constraint`

ограничение

Работа с ограничениями у модели, позволяет работать с выражением и границами а также удалять его

Public Functions

inline `Constraint`(const *LinearExpression* &expr, *constraint_sense* sense, double rhs)

создает ограничение вида (expr (<=, ==, >=) constant)

Параметры

- `expr` – [in] выражение, являющееся левой частью ограничения
- `sense` – [in] переменная, задающая знак (<=, ==, >=)
- `rhs` – [in] правая часть ограничения

inline *Constraint*(const *QuadExpression* &expr, *constraint_sense* sense, double rhs)
создает ограничение вида (expr ($\leq$, $=$, $\geq$) constant)

Параметры

- **expr** – [in] выражение, являющееся левой частью ограничения
- **sense** – [in] переменная, задающая знак ($\leq$, $=$, $\geq$)
- **rhs** – [in] правая часть ограничения

inline *LinearExpression* GetLinearExpression() const
Линейное выражение, связанное с ограничением

Результат

Выражение, связанное с ограничением

inline *QuadExpression* GetQuadExpression() const
Квадратичное выражение, связанное с ограничением

Результат

Выражение, связанное с ограничением

inline void SetUpperBound(double upper_bound)
Задаёт верхнюю границу ограничения

Параметры

upper_bound – [in] Новая верхняя граница ограничения

inline double GetUpperBound() const
Получает верхнюю границу ограничения

Результат

Верхнюю границу ограничения

inline void SetLowerBound(double lower_bound)
Задаёт нижнюю границу ограничения

Параметры

lower_bound – [in] Новая нижняя граница ограничения

inline double GetLowerBound() const
Получает нижнюю границу ограничения

Результат

Нижнюю границу ограничения

inline double GetRange() const
Получает интервал ограничения: upper_bound - lower_bound

Результат

upper_bound - lower_bound

inline void SetRange(double range)
Задаёт интервал ограничения: lower_bound $\leq$ expression $\leq$ lower_bound + range

Параметры

range – [in] интервал для ограничения

inline void Remove()
Помечает ограничение как удаленное. Такое ограничение будет исключено из расчетов

```
inline const char *GetName() const
```

Получает имя ограничения

Результат

имя ограничения

```
inline void SetName(const char *szName)
```

Задаёт имя ограничения

Параметры

szName – [in] имя ограничения

```
class LinearExpression
```

Линейное выражение

Обеспечивает работу с линейными выражениями - добавление/удаление элементов и изменение коэффициентов. Элемент выражения - переменная * коэффициент.

Public Functions

```
inline LinearExpression(double constant = 0.0)
```

Создает пустое выражение с возможностью задать константу

Параметры

constant – [in] константа в выражении

```
inline LinearExpression(const Variable &var, double coeff = 1.0)
```

Создает выражение на основе переменной и коэффициента

Параметры

- var – [in] переменная в выражении
- coeff – [in] коэффициент переменной в выражении

```
inline int GetTermsCount() const
```

Возвращает кол-во элементов в выражении

Результат

Кол-во элементов в выражении

```
inline bool empty() const
```

Возвращает истину, если выражение пустое

Результат

истина, если выражение пустое

```
inline void SetName(const char *expr_name)
```

Задаёт имя выражения

Параметры

expr_name – [in] имя выражения

```
inline const char *GetName() const
```

Получает имя выражения

Результат

имя выражения

```
inline Variable GetTermVariable(int i) const
```

Возвращает переменную по индексу элемента в выражении

Параметры

i – **[in]** индекс элемента в выражении

Результат

Объект переменной

```
inline int GetTermVariableIndex(int i) const
```

Возвращает индекс переменной по индексу элемента в выражении

Параметры

i – **[in]** индекс элемента в выражении

Результат

индекс переменной

```
inline double GetTermCoeff(int i) const
```

Возвращает коэффициент переменной по индексу элемента в выражении

Параметры

i – **[in]** индекс элемента в выражении

Результат

Значение коэффициента переменной

```
inline double GetConstant() const
```

Возвращает константу в выражении

Результат

значение константы

```
inline void SetTermCoeff(int i, double value)
```

Задаёт коэффициент переменной по индексу выражения

Параметры

- *i* – **[in]** Индекс элемента в выражении
- *value* – **[in]** Значение коэффициента при переменной в элементе

```
inline void SetConstant(double constant)
```

Задаёт константу в выражении

Параметры

constant – **[in]** значение константы

```
inline void AddConstant(double constant)
```

Добавляет константу к выражению

Параметры

constant – **[in]** значение константы

```
inline void AddTerm(const Variable &var, double coeff = 1.0)
```

Добавляет элемент к выражению

Параметры

- *var* – **[in]** переменная
- *coeff* – **[in]** коэффициент к переменной

```
inline void AddExpression(const LinearExpression &expr, double mult = 1.0)
```

Добавляет выражение к выражению

Параметры

- `expr` – **[in]** добавляемое выражение
- `mult` – **[in]** коэффициент к добавляемому выражению

```
inline void RemoveTerm(int idx)
```

Удаляет элемент выражения по индексу

Параметры

`idx` – **[in]** индекс удаляемого элемента в выражении

```
inline void RemoveVariable(const Variable &var)
```

Удаляет переменную из выражения

Параметры

`var` – **[in]** удаляемая переменная

```
inline LinearExpression CreateFreeCopy() const
```

Создает копию выражения, не привязанную к модели или другому ограничению

Результат

копия выражения

```
class QuadExpression
```

Квадратичное выражение

Обеспечивает работу с квадратичными выражениями - добавление/удаление элементов и изменение коэффициентов. Элемент выражения - переменная * переменная * коэффициент.

Public Functions

```
inline QuadExpression()
```

Создает пустое выражение

```
inline QuadExpression(const Variable &var1, const Variable &var2, double coeff = 1.0)
```

Создает выражение на основе переменных и коэффициента

Параметры

- `var1` – **[in]** переменная №1 в выражении
- `var2` – **[in]** переменная №2 в выражении
- `coeff` – **[in]** коэффициент в выражении

```
inline int GetTermsCount() const
```

Возвращает кол-во элементов в выражении

Результат

Кол-во элементов в выражении

```
inline bool empty() const
```

Возвращает истину, если выражение пустое

Результат

истина, если выражение пустое

```
inline void SetName(const char *expr_name)
```

Задаёт имя выражения

Параметры

`expr_name` – [in] имя выражения

```
inline const char *GetName() const
```

Получает имя выражения

Результат

имя выражения

```
inline Variable GetTermVariable1(int i) const
```

Возвращает переменную №1 по индексу элемента в выражении

Параметры

`i` – [in] индекс элемента в выражении

Результат

Объект переменной

```
inline Variable GetTermVariable2(int i) const
```

Возвращает переменную №2 по индексу элемента в выражении

Параметры

`i` – [in] индекс элемента в выражении

Результат

Объект переменной

```
inline double GetTermCoeff(int i) const
```

Возвращает коэффициент переменных по индексу элемента в выражении

Параметры

`i` – [in] индекс элемента в выражении

Результат

Значение коэффициента

```
inline void SetTermCoeff(int i, double value)
```

Задаёт коэффициент переменной по индексу выражения

Параметры

- `i` – [in] Индекс элемента в выражении
- `value` – [in] Значение коэффициента при переменных в элементе

```
inline void AddTerm(const Variable &var1, const Variable &var2, double coeff = 1.0)
```

Добавляет элемент к выражению

Параметры

- `var1` – [in] переменная
- `var2` – [in] переменная
- `coeff` – [in] коэффициент

```
inline void AddExpression(const QuadExpression &expr, double mult = 1.0)
```

Добавляет выражение к выражению

Параметры

- `expr` – **[in]** квадратичное выражение
- `mult` – **[in]** коэффициент к добавляемому выражению

`inline void AddExpression(const LinearExpression &expr, double mult = 1.0)`

Добавляет выражение к выражению

Параметры

- `expr` – **[in]** добавляемое выражение
- `mult` – **[in]** коэффициент к добавляемому выражению

`inline void RemoveTerm(int idx)`

Удаляет элемент выражения по индексу

Параметры

`idx` – **[in]** индекс удаляемого элемента в выражении

`inline QuadExpression CreateFreeCopy() const`

Создает копию выражения, не привязанную к модели или другому ограничению

Результат

копия выражения

`class Model`

Класс для работы с моделью

Предоставляет функции чтения/записи файлов, модификации модели, управления переменными, ограничениями, а также целевой функцией

Public Functions

`inline Model(const char *name = nullptr)`

Создает экземпляр модели

Параметры

`name` – **[in]** Имя модели

`inline void Read(const char *szFileName)`

Читает модель из файла, тип модели определяется по расширению

Параметры

`szFileName` – **[in]** путь к файлу модели

`inline void ReadMps(const char *szFileName)`

Читает модель из файла, при этом он будет читаться как MPS файл независимо от расширения

Параметры

`szFileName` – **[in]** путь к файлу модели

`inline void ReadLp(const char *szFileName)`

Читает модель из файла, при этом он будет читаться как LP файл независимо от расширения

Параметры

`szFileName` – **[in]** путь к файлу модели

```
inline int GetIntParam(const char *szParam)
```

Получает целочисленный параметр

Параметры

`szParam` – **[in]** имя параметра

Результат

Значение параметра

```
inline double GetDbiParam(const char *szParam)
```

Получает параметр с плавающей точкой

Параметры

`szParam` – **[in]** имя параметра

Результат

Значение параметра

```
inline std::string GetStringParam(const char *szParam)
```

Получает строковый параметр

Параметры

`szParam` – **[in]** имя параметра

Результат

Значение параметра

```
inline bool GetBoolParam(const char *szParam)
```

Получает логический параметр

Параметры

`szParam` – **[in]** имя параметра

Результат

Значение параметра

```
inline void SetIntParam(const char *szParam, int nVal)
```

Задаёт целочисленный параметр

Параметры

- `szParam` – **[in]** имя параметра
- `nVal` – **[in]** Новое значение параметра

```
inline void SetDbiParam(const char *szParam, double dVal)
```

Задаёт параметр с плавающей точкой

Параметры

- `szParam` – **[in]** имя параметра
- `dVal` – **[in]** Новое значение параметра

```
inline void SetStringParam(const char *szParam, const char *cVal)
```

Задаёт строковый параметр

Параметры

- `szParam` – **[in]** имя параметра
- `cVal` – **[in]** Новое значение параметра

```
inline void SetBoolParam(const char *szParam, bool bVal)
```

Задаёт логический параметр

Параметры

- `szParam` – [in] имя параметра
- `bVal` – [in] Новое значение параметра

```
inline void Write(const char *szFileName, const char *name_mapping_file = "")
```

Записывает модель в файл. Тип будет определен по расширению

Параметры

- `szFileName` – [in] путь к записываемому файлу
- `name_mapping_file` – [in] путь к файлу, который будет содержать соответствие между именами модели при анонимизации (если пустой - запись без анонимизации)

```
inline void WriteMps(const char *szFileName, const char *name_mapping_file = "")
```

Записывает модель в файл в формате MPS

Параметры

- `szFileName` – [in] путь к записываемому файлу
- `name_mapping_file` – [in] путь к файлу, который будет содержать соответствие между именами модели при анонимизации (если пустой - запись без анонимизации)

```
inline void WriteLp(const char *szFileName, const char *name_mapping_file = "")
```

Записывает модель в файл в формате LP

Параметры

- `szFileName` – [in] путь к записываемому файлу
- `name_mapping_file` – [in] путь к файлу, который будет содержать соответствие между именами модели при анонимизации (если пустой - запись без анонимизации)

```
inline Variable AddVariable(double lb = 0, double ub = inf, double obj = 0.0, variable_type  
var_type = variable_type::continuous, const char *szName = "")
```

Добавляет переменную в модель с заданными параметрами

Параметры

- `lb` – [in] нижняя граница переменной
- `ub` – [in] верхняя граница переменной
- `obj` – [in] коэффициент к переменной в целевой функции
- `var_type` – [in] тип переменной
- `szName` – [in] имя переменной

Результат

объект новой переменной

```
inline Constraint AddConstraint(const LinearExpression &expr, constraint_sense sense, double rhs,  
const char *szName)
```

Добавляет ограничение в модель с заданными параметрами

Параметры

- **expr** – [in] линейное выражение для ограничения
- **sense** – [in] тип ограничения ($\leq$, $\geq$, $=$)
- **rhs** – [in] константное значение в правой части ограничения
- **szName** – [in] имя ограничения. Если передана пустая строка или nullptr, имя ограничения будет сгенерировано

Результат

объект нового ограничения

```
inline Constraint AddConstraint(const QuadExpression &expr, constraint_sense sense, double rhs,
                               const char *szName)
```

Добавляет ограничение в модель с заданными параметрами

Параметры

- **expr** – [in] квадратичное выражение для ограничения
- **sense** – [in] тип ограничения ($\leq$, $\geq$, $=$)
- **rhs** – [in] константное значение в правой части ограничения
- **szName** – [in] имя ограничения. Если передана пустая строка или nullptr, имя ограничения будет сгенерировано

Результат

объект нового ограничения

```
inline Constraint AddConstraint(const LinearExpression &lhs, constraint_sense sense, const
                               LinearExpression &rhs, const char *szName)
```

Добавляет ограничение в модель с заданными параметрами

Параметры

- **lhs** – [in] выражение для ограничения (левая часть)
- **sense** – [in] переменная знака ограничения ($\leq$, $\geq$, $=$)
- **rhs** – [in] выражение в правой части ограничения
- **szName** – [in] имя ограничения

Результат

объект нового ограничения

```
inline Constraint AddConstraint(const LinearExpression &expr, double lb, double ub, const char
                               *szName)
```

Добавляет интервальное ограничение в модель с заданными параметрами, вида $lhs \leq expr \leq rhs$

Параметры

- **expr** – [in] выражение для ограничения
- **lb** – [in] нижняя граница ограничения
- **ub** – [in] верхняя граница ограничения
- **szName** – [in] имя ограничения

Результат

объект нового ограничения

```
inline Constraint AddConstraint(const QuadExpression &expr, double lb, double ub, const char
                               *szName)
```

Добавляет интервальное ограничение в модель с заданными параметрами, вида $lhs \leq expr$
 $\leq rhs$

Параметры

- `expr` – [in] выражение для ограничения
- `lb` – [in] нижняя граница ограничения
- `ub` – [in] верхняя граница ограничения
- `szName` – [in] имя ограничения

Результат

объект нового ограничения

```
inline Constraint AddConstraint(const Constraint &constr, const char *szName)
```

Добавляет уже существующее ограничение в модель

Параметры

- `constr` – [in] ограничение
- `szName` – [in] имя ограничения

Результат

объект нового ограничения

```
inline Variable GetVariable(int idx) const
```

Получает переменную модели по индексу

Параметры

`idx` – [in] индекс переменной

Результат

объект переменной

```
inline int GetVariablesCount() const
```

Получает количество переменных в модели

Результат

количество переменных в модели

```
inline Variable GetVariableByName(const char *name)
```

Получает переменную из модели по имени

Параметры

`name` – [in] имя переменной

Результат

объект переменной

```
inline Constraint GetConstraint(int idx) const
```

Получает ограничение из модели по индексу

Параметры

`idx` – [in] индекс ограничения

Результат

объект ограничения

```
inline int GetConstraintsCount() const
```

Получает количество ограничений модели

Результат

количество ограничений модели

```
inline Constraint GetConstrByName(const char *szName)
```

Получает ограничение из модели по имени

Параметры

`szName` – [in] имя ограничения

Результат

объект ограничения

```
inline void SetObjectiveSense(objective_sense sense)
```

Задаёт тип оптимизации целевой функции - максимизация или минимизация

Параметры

`sense` – [in] тип оптимизации

```
inline objective_sense GetObjectiveSense() const
```

Получает тип оптимизации целевой функции

Результат

тип оптимизации

```
inline void SetObjectiveOffset(double constant)
```

Задаёт константу в выражении целевой функции

Параметры

`constant` – [in] значение константы в целевой функции

```
inline void SetObjective(const LinearExpression &expr, objective_sense sense =  
                        objective_sense::minimize)
```

Задаёт выражение целевой функции и тип оптимизации

Параметры

- `expr` – [in] линейное выражение
- `sense` – [in] тип оптимизации

```
inline void SetObjective(const QuadExpression &expr, objective_sense sense =  
                        objective_sense::minimize)
```

Задаёт выражение целевой функции и тип оптимизации

Параметры

- `expr` – [in] квадратичное выражение
- `sense` – [in] тип оптимизации

```
inline LinearExpression GetObjective() const
```

Получает выражение целевой функции

Результат

Объект выражения

```
inline SolveResult Solve()
```

Стартует оптимизацию модели

Результат

Объект с результатом оптимизации

```
inline SolveResult SolveRemote(const char *name_mapping_file)
```

Стартует оптимизацию модели на удаленном сервере в синхронном режиме. Предварительно необходимо установить переменную окружения X_API_KEY.

Параметры

`name_mapping_file` – **[in]** путь к файлу, который будет записан и будет содержать соответствие оригинальных имен модели и анонимизированных

Результат

Объект с результатом оптимизации

```
inline void SolveRemoteAsync(const char *name_mapping_file)
```

Стартует оптимизацию модели на удаленном сервере в асинхронном режиме без ожидания результата. Предварительно необходимо установить переменную окружения X_API_KEY.

Параметры

`name_mapping_file` – **[in]** путь к файлу, который будет записан и будет содержать соответствие оригинальных имен модели и анонимизированных

```
inline ComputeIisResult ComputeIis()
```

Стартует вычисление несократимого недостижимого множества

Результат

Объект с результатом вычисления несократимого недостижимого множества

```
inline const char *GetRemoteSolveLog(const char *szCalcUID) const
```

Получает лог удалённого расчета

Параметры

`szCalcUID` – **[in]** идентификатор удалённого расчета

Результат

лог удалённого расчета

```
inline void Remove(Variable &var)
```

Удаляет переменную из модели

Параметры

`var` – **[in]** переменная

```
inline void Remove(Constraint &constr)
```

Удаляет ограничение из модели

Параметры

`constr` – **[in]** ограничение

```
inline void Clear()
```

Очищает модель от всех данных. Все старые переменные и ограничения становятся невалидными

```
inline void SetLogFile(const char *szLogFile)
```

Задать файл для записи лога решения

Параметры

`szLogFile` – **[in]** файл лога

```
inline void SetLogCallback(ILogCallback *pcb)
```

Задать интерфейс для обратного вызова при записи лога

Параметры

pcb – [in] интерфейс для обратного вызова

```
inline void AddMipStartValue(const char *var_name, double var_value)
```

Добавить возможное значение переменной в решении в качестве «подсказки». Стартовые значения очищаются после начала процесса решения.

Параметры

- var_name – [in] имя переменной
- var_value – [in] значение переменной

```
inline void AddMipStartValue(const Variable &var, double var_value)
```

Добавить возможное значение переменной в решении в качестве «подсказки». Стартовые значения очищаются после начала процесса решения.

Параметры

- var – [in] переменная
- var_value – [in] значение переменной

```
inline void AddMipStartValues(const char *sol_file)
```

Добавить возможные значения переменных в решении в качестве «подсказки». Стартовые значения очищаются после начала процесса решения.

Параметры

sol_file – [in] путь к файлу с решением

```
inline void ClearMipStartValues()
```

Удалить все стартовые значения для поиска решения

```
inline const char *GetName() const
```

Получает имя модели

Результат

имя модели

```
inline void SetName(const char *szName)
```

Задаёт имя модели

Параметры

szName – [in] имя модели

```
inline std::string GetCalcUID() const
```

Получить уникальный идентификатор последнего удаленного расчета.

Результат

идентификатор удалённого расчета

```
inline bool IsMip() const
```

Проверяет является ли задача целочисленной.

Результат

true если задача целочисленная, иначе false.

```
inline bool IsNonlinear() const
```

Проверяет является ли задача нелинейной.

Результат

true если задача нелинейная, иначе false.

```
class SolveResult
```

Класс для работы с итогами расчета

Предоставляет доступ к информации о расчете (количество итераций, статус, значение целевой функции и пр.), а также к значениям переменных

Public Functions

```
inline solve_result GetSolveResult() const
```

Получает статус процесса расчетов

Результат

Значение статуса

```
inline solution_status GetSolutionStatus() const
```

Получает статус модели в итоге расчета

Результат

Значение статуса

```
inline double GetRelativeGap() const
```

Получает точность решения в процентах

Результат

точность решения в процентах

```
inline unsigned int GetIterationsCount() const
```

Получает количество итераций, проведенных в процессе расчета

Результат

Кол-во итераций

```
inline std::int64_t GetProcessedNodesCount() const
```

Получает количество обработанных узлов дерева решений

Результат

Кол-во обработанных узлов

```
inline double GetObjectiveFunctionValue() const
```

Получает значение целевой функции

Результат

Значение целевой функции

```
inline double GetBestBoundValue() const
```

Получает значение граничной (двойственной) функции

Результат

Значение граничной (двойственной) функции

```
inline double GetSolveTime() const
```

Получает общее время решения

Результат

Общее время решения, секунды

```
inline double GetVariableValue(const Variable &var) const
```

Получает значение переменной в решении

Параметры

`var` – **[in]** переменная

Результат

Значение переменной в решении

```
inline double GetVariableValue(const char *var_name) const
```

Получает значение переменной в решении по имени

Параметры

`var_name` – **[in]** имя переменной

Результат

Значение переменной в решении

```
inline double GetDualValue(const Constraint &constr) const
```

Получает значение dual value

Параметры

`constr` – **[in]** ограничение

Результат

Значение dual value

```
inline double GetDualValue(const char *constr_name) const
```

Получает значение dual value по имени ограничения

Параметры

`constr_name` – **[in]** имя ограничения

Результат

Значение dual value

```
inline double GetReducedCost(const Variable &var) const
```

Получает значение reduced cost

Параметры

`var` – **[in]** переменная

Результат

Значение reduced cost

```
inline double GetReducedCost(const char *var_name) const
```

Получает значение reduced cost по имени переменной

Параметры

`var_name` – **[in]** имя переменной

Результат

Значение reduced cost

```
inline double GetExpressionValue(const LinearExpression &expr) const
```

Получает значение выражения

Параметры

`expr` – [in] линейное выражение

Результат

Значение выражения

```
inline double GetExpressionValue(const QuadExpression &expr) const
```

Получает значение выражения

Параметры

`expr` – [in] квадратичное выражение

Результат

Значение выражения

```
inline void WriteSolution(const char *file_name) const
```

Записывает решение в файл

Параметры

`file_name` – [in] путь к файлу решения для записи

```
class ComputeIisResult
```

Класс для работы с итогами вычисления несократимого недостижимого множества

Предоставляет доступ к информации о вычислении несократимого недостижимого множества

Public Functions

```
inline compute_iis_result GetComputeIisResult() const
```

Получает статус процесса вычислений несократимого недостижимого множества

Результат

Значение статуса

```
inline model_status GetModelStatus() const
```

Получает статус модели в итоге вычисления несократимого недостижимого множества

Результат

Значение статуса

```
inline bool GetMinimal() const
```

Получает, минимально ли несократимое недостижимое множество

Результат

Минимально ли несократимое недостижимое множество

```
inline int GetNumConstraints() const
```

Получает число ограничений в несократимом недостижимом множестве

Результат

Число ограничений в несократимом недостижимом множестве

```
inline int GetNumBounds() const
```

Получает число границ в несократимом недостижимом множестве

Результат

Число границ в несократимом недостижимом множестве

```
inline double GetComputeIisTime() const
```

Получает время вычисления несократимого недостижимого множества

Результат

Время вычисления несократимого недостижимого множества, секунды

```
inline void WriteIis(const char *file_name) const
```

Записывает несократимое недостижимое множество в файл

Параметры

`file_name` – **[in]** путь к файлу несократимого недостижимого множества для записи

```
class Variable
```

Переменная

Работа с переменной - чтение/изменение имени, верхней и нижней границы, типа, а также удаление

Public Functions

```
inline const char *GetName() const
```

Получает имя переменной в модели

Результат

имя переменной

```
inline void SetName(const char *szName)
```

Задаёт имя переменной в модели

Параметры

`szName` – **[in]**

```
inline arhiplex::variable_type GetType() const
```

Получает тип переменной

Результат

тип переменной

```
inline void SetType(arhiplex::variable_type type)
```

Задаёт тип переменной в модели

Параметры

`type` – **[in]** тип переменной

```
inline void SetUpperBound(double upper_bound)
```

Задаёт верхнюю границу для переменной

Параметры

`upper_bound` – **[in]** новая верхняя граница переменной

```
inline double GetUpperBound() const
```

Получает верхнюю границу переменной

Результат

верхняя граница переменной

```
inline void SetLowerBound(double lower_bound)
```

Задаёт нижнюю границу для переменной

Параметры

`lower_bound` – **[in]** новая нижняя граница переменной

```
inline double GetLowerBound() const
```

Получает нижнюю границу переменной

Результат

нижняя граница переменной

```
inline void Remove()
```

Помечает переменную как удалённую. Переменная не будет участвовать в расчётах

```
enum class arhiplex::variable_type
```

Тип переменной

Values:

```
enumerator continuous
```

непрерывная

```
enumerator binary
```

бинарная

```
enumerator integer
```

целочисленная

```
enumerator semicontinuous
```

полу-непрерывная

```
enumerator semiinteger
```

полу-целая

```
enum class arhiplex::objective_sense
```

Направление оптимизации целевой функции

Values:

```
enumerator minimize
```

Минимизация

```
enumerator maximize
```

Максимизация

```
enum class arhiplex::constraint_sense
```

Знак ограничения между левой частью (выражением) и правой частью (константой)

Values:

enumerator equal

„==“

enumerator less_equal

„<=“

enumerator greater_equal

„>=“

enum class arhiplex::solve_result

Результат процесса расчетов

Values:

enumerator success

Процесс решения успешен - найдено некоторое решение или обнаружена недостижимость модели

enumerator fail

Процесс решения неудачен - допустимое решение или доказательство недостижимости не получено; подробность доступна в solution_status

enumerator remote_invalid_api_key

Невалидный ключ для удаленного расчета

enumerator remote_api_key_not_set

Переменная окружения X_API_KEY не установлена

enumerator remote_time_amount_is_over

Не осталось доступного времени для осуществления удаленного расчета

enumerator remote_time_per_calc_violated

Превышен лимит времени для единичного удаленного расчета. Параметр time_limit нарушает ограничения лицензии

enumerator remote_fail

Удаленный расчет неудачен

enum class arhiplex::solution_status

Статус решения по результатам расчета

Values:

enumerator invalid_solution

неопределенное/невалидное значение

enumerator `optimal`

решение оптимально (погрешность в рамках заданного значения)

enumerator `feasible`

решение найдено, но не оптимально (погрешность $>$ заданного значения)

enumerator `infeasible`

модель не имеет решения (решение недостижимо)

enumerator `unbounded`

модель неограничена (целевая функция может бесконечно неограниченно увеличиваться/уменьшаться)

enumerator `infeasible_or_unbounded`

модель недостижима или неограничена

enumerator `imprecise`

найден кандидат с конечной целевой функцией, но нарушения ограничений, границ или целочисленности превышают заданные допуски; это не допустимое решение

enum class `arhiplex::compute_iis_result`

Результат процесса вычислений несократимого недостижимого множества

Values:

enumerator `success`

Процесс вычисления несократимого недостижимого множества успешен - найдено некоторое несократимое недостижимое множество или обнаружена достижимость модели

enumerator `fail`

Процесс вычисления несократимого недостижимого множества неудачен - не найдено никаких несократимых недостижимых множеств и не обнаружена достижимость модели

enum class `arhiplex::model_status`

Статус модели по результатам вычисления несократимого недостижимого множества

Values:

enumerator `unknown`

неизвестное значение

enumerator `feasible`

модель достижима (модель не имеет несократимого недостижимого множества)

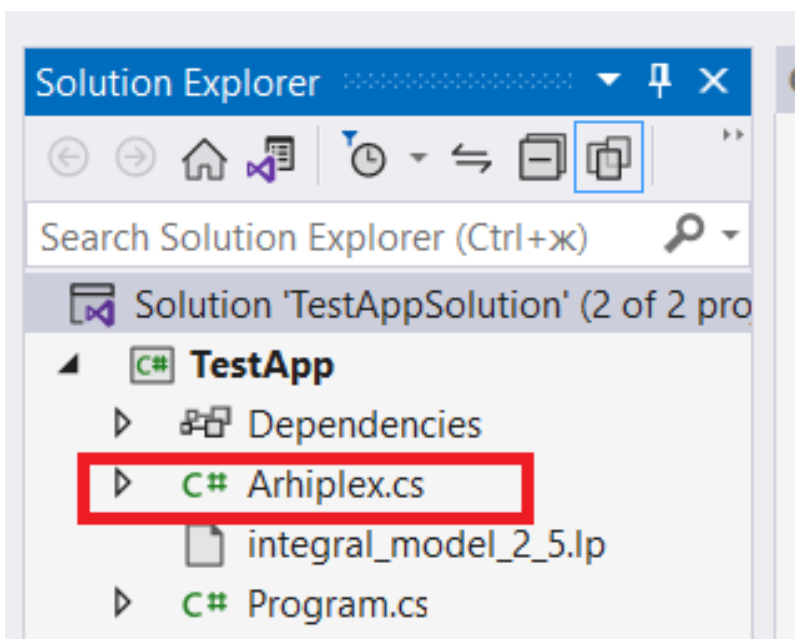
enumerator `infeasible`

модель недостижима (найденое некоторое несократимое недостижимое множество)

C# API повторяет объектную модель C++ API: класс `Model` и связанные типы (`Variable`, `Constraint`, `LinearExpression`, `QuadExpression`, `SolveResult`). В C# доступна перегрузка операторов, поэтому выражения записываются естественно ($x + y \leq 10$, $2 * x + y$).

19.1 Подключение

Добавьте в проект файл `ArhiPlex.cs` из каталога `include/arhiplex` установленного солвера. Нативная библиотека `libarhiplex_cpp` должна быть доступна среде выполнения (`LD_LIBRARY_PATH` в Linux).



Примечание: Локальная установка солвера поддерживается на Linux (см. [Установка](#)). В Windows расчёт выполняется через удалённый клиент.

19.2 Сквозной пример

Рассмотрим задачу:

$$\begin{aligned}
 &\textbf{Maximize:} \\
 &x_1 + 2x_2 - 0.1x_3 - 3x_4 \\
 &\textbf{Subject to:} \\
 &x_1 + x_2 \leq 5 \\
 &2x_1 - x_2 \geq 0 \\
 &-x_1 + 3x_2 \geq 0 \\
 &x_3 + x_4 \geq 0.5 \\
 &\textbf{Bounds:} \\
 &-\infty < x_1 < +\infty \\
 &-\infty < x_2 < +\infty \\
 &1.1 \leq x_3 \leq 10.0 \\
 &-\infty < x_4 < +\infty
 \end{aligned} \tag{19.1}$$

Полный исходный код (файл `example_dot_net.cs` из каталога `examples`):

```

1 using arhiplex;
2 internal class Program
3 {
4     private static void Main(string[] args)
5     {
6         var m = new Model();
7
8         // Устанавливаем наименование модели
9         m.SetName("model_1");
10
11        // Добавляем переменные
12        var x1 = m.AddVariable(-Model.Inf, Model.Inf, 0, Variable_type.continuous, "x1");
13        var x2 = m.AddVariable(-Model.Inf, Model.Inf, 0, Variable_type.continuous, "x2");
14        var x3 = m.AddVariable(1.1, 10, 0, Variable_type.continuous, "x3");
15        var x4 = m.AddVariable(-Model.Inf, Model.Inf, 0, Variable_type.continuous, "x4");
16
17        // Добавляем ограничения
18        m.AddConstraint(x1 + x2 <= 5, "c1");
19        m.AddConstraint(2 * x1 - x2 >= 0, "c2");
20        m.AddConstraint(-x1 + 3 * x2 >= 0, "c3");
21        m.AddConstraint(x3 + x4 >= 0.5, "c4");
22
23        // Устанавливаем целевую функцию
24        m.SetObjective(x1 + 2 * x2 - 0.1 * x3 - 3 * x4, Objective_sense.maximize);
25
26        // Устанавливаем лимит времени расчета задачи

```

(continues on next page)

(продолжение с предыдущей страницы)

```

27     m.SetDbiParam("time_limit", 30);
28
29     var solution = m.Solve();
30
31     var solution_status = solution.GetSolutionStatus();
32     var solve_result = solution.GetSolveResult();
33
34     if (solve_result == Solve_result.success &&
35         (solution_status == Solution_status.optimal ||
36          solution_status == Solution_status.feasible))
37     {
38         var gap = solution.GetRelativeGap();
39         var time = solution.GetSolveTime();
40         var x1_value = solution.GetVariableValue(x1);
41         var x2_value = solution.GetVariableValue("x2");
42         var x3_value = solution.GetVariableValue("x3");
43         var x4_value = solution.GetVariableValue(x4);
44     }
45
46 }
47 }

```

Создание модели:

```
var m = new Model();
```

Добавление переменных методом `AddVariable()` (границы, коэффициент в целевой функции, тип, имя):

```

var x1 = m.AddVariable(-Model.Inf, Model.Inf, 0, Variable_type.continuous, "x1");
var x2 = m.AddVariable(-Model.Inf, Model.Inf, 0, Variable_type.continuous, "x2");
var x3 = m.AddVariable(1.1, 10, 0, Variable_type.continuous, "x3");
var x4 = m.AddVariable(-Model.Inf, Model.Inf, 0, Variable_type.continuous, "x4");

```

Целевая функция — метод `SetObjective()` (выражение и направление оптимизации):

```
m.SetObjective(x1 + 2 * x2 - 0.1 * x3 - 3 * x4, Objective_sense.maximize);
```

Ограничения — метод `AddConstraint()` (ограничение и его имя):

```

m.AddConstraint(x1 + x2 <= 5, "c1");
m.AddConstraint(2 * x1 - x2 >= 0, "c2");
m.AddConstraint(-x1 + 3 * x2 >= 0, "c3");
m.AddConstraint(x3 + x4 >= 0.5, "c4");

```

Запуск расчёта:

```
var solution = m.Solve();
```

Результат содержит данные расчёта — например, время в секундах:

```
var time = solution.GetSolveTime();
```

Предупреждение: Прежде чем читать значения переменных, убедитесь, что расчёт успешен и модель достижима (см. *Интерпретация результата*).

```
if (solve_result == Solve_result.success &&
    (solution_status == Solution_status.optimal ||
     solution_status == Solution_status.feasible))
```

Значение переменной методом `GetVariableValue()` можно получить как по объекту переменной, так и по её имени. Простой пример MIP-модели см. в файле `simple_mip.cs` каталога `examples`.

19.3 Справочник API

Полный справочник классов и перечислений формируется автоматически из исходных комментариев.

```
class arhiplex.ArhiplexException : Exception
```

Исключение, генерируемое всеми классами в случае ошибки

```
class arhiplex.Constraint : IDisposable
```

Работа с ограничениями у модели, позволяет работать с выражением и границами

Public Functions

```
string? GetName ()
```

Получает имя ограничения

```
return
```

имя ограничения

```
void SetName (string name)
```

Задаёт имя ограничения в модели

```
param name
```

имя ограничение

```
void Remove ()
```

Помечает ограничение как удаленное. Такое ограничение будет исключено из расчетов

```
LinearExpression GetLinearExpression ()
```

Линейное выражение, связанное с ограничением

```
return
```

Линейное выражение

```
QuadExpression GetQuadExpression ()
```

Квадратичное выражение, связанное с ограничением

```
return
```

Квадратичное выражение

```
void SetUpperBound (double upper_bound)
```

Задаёт верхнюю границу для ограничения

param upper_bound

новая верхняя граница ограничения

```
double GetUpperBound ()
```

Получает верхнюю границу ограничения

return

верхняя граница ограничения

```
void SetLowerBound (double lower_bound)
```

Задаёт нижнюю границу для ограничения

param lower_bound

новая нижняя граница ограничения

```
double GetLowerBound ()
```

Получает нижнюю границу ограничения

return

нижняя граница ограничения

```
double GetRange ()
```

Получает интервал ограничения: $upper_bound - lower_bound$

return

$upper_bound - lower_bound$

```
void SetRange (double range)
```

Задаёт интервал ограничения: $lower_bound \leq expression \leq lower_bound + range$

param range

интервал для ограничения

```
class arhiplex.LinearExpression : IDisposable
```

Обеспечивает работу с линейными выражениями - добавление/удаление элементов и изменение коэффициентов. Элемент выражения - переменная * коэффициент.

Public Functions

```
LinearExpression (double offset)
```

Создает пустое выражение с возможностью задать константу

param offset

константа в выражении

```
LinearExpression (Variable var, double coeff = 1.0)
```

Создает выражение на основе переменной и коэффициента

param var

переменная в выражении

param coeff

коэффициент переменной в выражении

`int GetTermsCount ()`

Возвращает кол-во элементов в выражении

return

кол-во элементов в выражении

Variable `GetTermVariable (int term_idx)`

Возвращает переменную по индексу элемента в выражении

param term_idx

индекс элемента в выражении

return

Объект переменной

`double GetTermCoeff (int term_idx)`

Возвращает коэффициент переменной по индексу элемента в выражении

param term_idx

индекс элемента в выражении

return

Значение коэффициента переменной

`void SetTermCoeff (int term_idx, double value)`

Задаёт коэффициент переменной по индексу выражения

param term_idx

индекс элемента в выражении

param value

Значение коэффициента при переменной в элементе

`double GetConstant ()`

Возвращает константу в выражении

return

значение константы

`void SetConstant (double constant)`

Задаёт константу в выражении

param constant

значение константы

`void AddConstant (double constant)`

Добавляет константу к выражению

param constant

значение константы

`void AddTerm (Variable var, double coeff)`

Добавляет элемент к выражению

param var

переменная

param coeff

коэффициент к переменной

```
void AddExpression (LinearExpression expr, double mult = 1.0)
```

Добавляет выражение к выражению

param expr

добавляемое выражение

param mult

коэффициент к добавляемому выражению

```
void RemoveTerm (int term_idx)
```

Удаляет элемент выражения по индексу

param term_idx

индекс удаляемого элемента в выражении

```
void RemoveVariable (Variable var)
```

Удаляет переменную из выражения

param var

удаляемая переменная

```
QuadExpression GetQuadPart ()
```

Возвращает квадратичную часть выражения

return

квадратичная часть выражения

```
LinearExpression CreateFreeCopy ()
```

Создает копию выражения, не привязанную к модели или другому ограничению

return

копия выражения

```
class arhiplex.QuadExpression : IDisposable
```

Обеспечивает работу с квадратичными выражениями - добавление/удаление элементов и изменение коэффициентов. Элемент выражения - переменная * переменная * коэффициент.

Public Functions

```
QuadExpression ()
```

Создает пустое выражение

```
QuadExpression (Variable var1, Variable var2, double coeff = 1.0)
```

Создает выражение на основе переменных и коэффициента

param var1

переменная №1 в выражении

param var2

переменная №2 в выражении

param coeff

коэффициент в выражении

```
int GetTermsCount ()
```

Возвращает кол-во элементов в выражении

return

Кол-во элементов в выражении

Variable GetTermVariable1 (int *term_idx*)

Возвращает переменную №1 по индексу элемента в выражении

param term_idx
индекс элемента в выражении

return
объект переменной

Variable GetTermVariable2 (int *term_idx*)

Возвращает переменную №2 по индексу элемента в выражении

param term_idx
индекс элемента в выражении

return
объект переменной

double GetTermCoeff (int *term_idx*)

Возвращает коэффициент переменных по индексу элемента в выражении

param term_idx
индекс элемента в выражении

return
Значение коэффициента

void SetTermCoeff (int *term_idx*, double *value*)

Задаёт коэффициент переменной по индексу выражения

param term_idx
индекс элемента в выражении

param value
Значение коэффициента при переменных в элементе

void AddTerm (*Variable* *var1*, *Variable* *var2*, double *coeff*)

Добавляет элемент к выражению

param var1
переменная

param var2
переменная

param coeff
коэффициент

void AddExpression (*QuadExpression* *quad_expr*, double *mult* = 1.0)

Добавляет выражение к выражению

param quad_expr
квадратичное выражение

param mult
коэффициент к добавляемому выражению

`void RemoveTerm (int term_idx)`

Удаляет элемент выражения по индексу

param *term_idx*

индекс удаляемого элемента в выражении

LinearExpression `GetLinearPart ()`

Возвращает линейную часть выражения

return

линейная часть

QuadExpression `CreateFreeCopy ()`

Создает копию выражения, не привязанную к модели или другому ограничению

return

копия выражения

`class arhiplex.Model : IDisposable`

Предоставляет функции чтения/записи файлов, модификации модели, управления переменными, ограничениями, а также целевой функцией

Public Functions

`Model (string name = "")`

Создает экземпляр модели

param *name*

Имя модели

`void Read (string file_name)`

Читает модель из файла, тип модели определяется по расширению

param *file_name*

путь к файлу модели

`void ReadMps (string file_name)`

Читает модель из файла, при этом он будет читаться как MPS файл независимо от расширения

param *file_name*

путь к файлу модели

`void ReadLp (string file_name)`

Читает модель из файла, при этом он будет читаться как LP файл независимо от расширения

param *file_name*

путь к файлу модели

`void Write (string file_name, string name_mapping_file = "")`

Записывает модель в файл. Тип будет определен по расширению

param *file_name*

путь к записываемому файлу

param *name_mapping_file*

путь к файлу, который будет содержать соответствие между именами модели при анонимизации (если пустой - запись без анонимизации)

```
void WriteMps (string file_name, string name_mapping_file = "")
```

Записывает модель в файл в формате MPS

param file_name

путь к записываемому файлу

param name_mapping_file

путь к файлу, который будет содержать соответствие между именами модели при анонимизации (если пустой - запись без анонимизации)

```
void WriteLp (string file_name, string name_mapping_file = "")
```

Записывает модель в файл в формате LP

param file_name

путь к записываемому файлу

param name_mapping_file

путь к файлу, который будет содержать соответствие между именами модели при анонимизации (если пустой - запись без анонимизации)

```
int GetIntParam (string param_name)
```

Получает целочисленный параметр

param param_name

имя параметра

return

Значение параметра

```
double GetDb1Param (string param_name)
```

Получает параметр с плавающей точкой

param param_name

имя параметра

return

Значение параметра

```
string GetStringParam (string param_name)
```

Получает строковый параметр

param param_name

имя параметра

return

Значение параметра

```
bool GetBoolParam (string param_name)
```

Получает логический параметр

param param_name

имя параметра

return

Значение параметра

```
void SetIntParam (string param_name, int nVal)
```

Задаёт целочисленный параметр

param *param_name*

имя параметра

param *nVal*

Новое значение параметра

```
void SetDb1Param (string param_name, double dVal)
```

Задаёт параметр с плавающей точкой

param *param_name*

имя параметра

param *dVal*

Новое значение параметра

```
void SetStringParam (string param_name, string cVal)
```

Задаёт строковый параметр

param *param_name*

имя параметра

param *cVal*

Новое значение параметра

```
void SetBoolParam (string param_name, bool bVal)
```

Задаёт логический параметр

param *param_name*

имя параметра

param *bVal*

Новое значение параметра

```
string? GetName ()
```

Получает имя модели

return

Имя модели

```
void SetName (string name)
```

Задаёт имя модели

param *name*

Имя модели

```
Variable AddVariable (double lb, double ub, double obj, Variable_type var_type, string name)
```

Добавляет переменную в модель с заданными параметрами

param *lb*

нижняя граница переменной

param *ub*

верхняя граница переменной

param *obj*

коэффициент к переменной в целевой функции

param *var_type*

тип переменной

param name
имя переменной

Constraint AddConstraint (*LinearExpression* lin_expr, *Constraint_sense* sense, double rhs, string name)

Добавляет ограничение в модель с заданными параметрами

param lin_expr
линейное выражение для ограничения

param sense
тип ограничения (<=, >= , ==)

param rhs
константное значение в правой части ограничения

param name
имя ограничения. Если передана пустая строка или null, имя ограничения будет сгенерировано

return
объект нового ограничения

Constraint AddConstraint (*QuadExpression* quad_expr, *Constraint_sense* sense, double rhs, string name)

Добавляет ограничение в модель с заданными параметрами

param quad_expr
квадратичное выражение для ограничения

param sense
тип ограничения (<=, >= , ==)

param rhs
константное значение в правой части ограничения

param name
имя ограничения. Если передана пустая строка или null, имя ограничения будет сгенерировано

return
объект нового ограничения

Constraint AddConstraint (*LinearExpression* lin_expr, double lb, double ub, string name)

Добавляет ограничение в модель с заданными параметрами

param lin_expr
линейное выражение для ограничения

param lb
нижняя граница ограничения

param ub
верхняя граница ограничения

param name
имя ограничения. Если передана пустая строка или null, имя ограничения будет сгенерировано

return
объект нового ограничения

Constraint AddConstraint (*QuadExpression* quad_expr, double lb, double ub, string name)

Добавляет ограничение в модель с заданными параметрами

param quad_expr

квадратичное выражение для ограничения

param lb

нижняя граница ограничения

param ub

верхняя граница ограничения

param name

имя ограничения. Если передана пустая строка или null, имя ограничения будет сгенерировано

return

объект нового ограничения

Constraint AddConstraint (*Constraint* constr, string name)

Добавляет уже существующее ограничение в модель

param constr

ограничение

param name

имя ограничения

return

объект нового ограничения

int GetVariablesCount ()

Получает количество переменных в модели

return

количество переменных в модели

Variable GetVariable (int var_idx)

Получает переменную модели по индексу

param var_idx

индекс переменной

return

объект переменной

Variable GetVariable (string name)

Получает переменную модели по имени

param name

имя переменной

return

объект переменной

int GetConstraintsCount ()

Получает количество ограничений модели

return

количество ограничений модели

Constraint GetConstraint (int *constr_idx*)

Получает ограничение из модели по индексу

param constr_idx

индекс ограничения

return

объект ограничения

Constraint GetConstraint (string *name*)

Получает ограничение из модели по имени

param name

имя ограничения

return

объект ограничения

void SetObjectiveSense (*Objective_sense* *sense*)

Задаёт тип оптимизации целевой функции - максимизация или минимизация

param sense

тип оптимизации

Objective_sense GetObjectiveSense ()

Получает тип оптимизации целевой функции

return

тип оптимизации целевой функции

void SetObjectiveOffset (double *offset*)

Задаёт константу в выражении целевой функции

param offset

значение константы в целевой функции

void SetObjective (*LinearExpression* *expr*, *Objective_sense* *sense* = *Objective_sense.minimize*)

Задаёт выражение целевой функции и тип оптимизации

param expr

линейное выражение

param sense

тип оптимизации

void SetObjective (*QuadExpression* *quad_expr*, *Objective_sense* *sense* = *Objective_sense.minimize*)

Задаёт выражение целевой функции и тип оптимизации

param quad_expr

квадратичное выражение

param sense

тип оптимизации

void ClearObjective ()

Удаляет целевую функцию из модели

LinearExpression GetObjective ()

Получает выражение целевой функции

return

Объект выражения

QuadExpression GetQuadObjective ()

Получает выражение целевой функции

return

Объект выражения

SolveResult Solve ()

Стартует оптимизацию модели

return

Объект с результатом оптимизации

SolveResult SolveRemote (string *name_mapping_file*)

Стартует оптимизацию модели на удаленном сервере в синхронном режиме. Предварительно необходимо установить переменную окружения X_API_KEY.

param name_mapping_file

путь к файлу, который будет записан и будет содержать соответствие оригинальных имен модели и анонимизированных

return

Объект с результатом оптимизации

void SolveRemoteAsync (string *name_mapping_file*)

Стартует оптимизацию модели на удаленном сервере в асинхронном режиме без ожидания результата. Предварительно необходимо установить переменную окружения X_API_KEY.

param name_mapping_file

путь к файлу, который будет записан и будет содержать соответствие оригинальных имен модели и анонимизированных

string? GetRemoteSolveLog (string *calc_uid*)

Получает лог удалённого расчета

param calc_uid

идентификатор удалённого расчета

return

лог удалённого расчета

void Remove (*Variable* *var*)

Удаляет переменную из модели

param var

переменная

void Remove (*Constraint* *constr*)

Удаляет ограничение из модели

param constr

ограничение

void Clear ()

Очищает модель от всех данных. Все старые переменные и ограничения становятся невалидными

void SetLogFile (string *log_file*)

Задать файл для записи лога решения

param log_file

файл лога

void AddMipStartValue (string *var_name*, double *var_value*)

Добавить возможное значение переменной в решении в качестве «подсказки». Стартовые значения очищаются после начала процесса решения.

param var_name

имя переменной

param var_value

значение переменной

void AddMipStartValue (*Variable* *var*, double *var_value*)

Добавить возможное значение переменной в решении в качестве «подсказки». Стартовые значения очищаются после начала процесса решения.

param var

переменная

param var_value

значение переменной

void AddMipStartValues (string *sol_file*)

Добавить возможное значение переменной в решении в качестве «подсказки». Стартовые значения очищаются после начала процесса решения.

param sol_file

путь к файлу с решением

void ClearMipStartValues ()

Удалить все стартовые значения для поиска решения

string? GetCalcUID ()

Получить уникальный идентификатор последнего удаленного расчета.

return

идентификатор удаленного расчета.

bool IsMip ()

Проверяет является ли задача целочисленной.

return

true если задача целочисленная, иначе false.

bool IsNonlinear ()

Проверяет является ли задача нелинейной.

return

true если задача нелинейная, иначе false.

```
class arhiplex.SolveResult : IDisposable
```

Предоставляет доступ к информации о расчете (количество итераций, статус, значение целевой функции и пр.), а также к значениям переменных

Public Functions

```
Solve_result GetSolveResult ()
```

Получает статус процесса расчетов

```
return
```

Значение статуса

```
Solution_status GetSolutionStatus ()
```

Получает статус модели в итоге расчета

```
return
```

Значение статуса

```
double GetRelativeGap ()
```

Получает точность решения в процентах

```
return
```

точность решения в процентах

```
uint GetIterationsCount ()
```

Получает количество итераций, проведенных в процессе расчета

```
return
```

Кол-во итераций

```
Int64 GetProcessedNodesCount ()
```

Получает количество обработанных узлов дерева решений

```
return
```

Кол-во обработанных узлов

```
double GetObjectiveFunctionValue ()
```

Получает значение целевой функции

```
return
```

Значение целевой функции

```
double GetBestBoundValue ()
```

Получает значение граничной (двойственной) функции

```
return
```

Значение граничной (двойственной) функции

```
double GetSolveTime ()
```

Получает общее время решения

```
return
```

Общее время решения, секунды

```
double GetVariableValue (Variable var)
```

Получает значение переменной в решении

```
param var
```

объект переменной

return

Значение переменной в решении

double **GetVariableValue** (string *var_name*)

Получает значение переменной в решении по имени

param var_name

имя переменной

return

Значение переменной в решении

double **GetDualValue** (*Constraint* *constr*)

Получает значение dual value

param constr

ограничение

return

Значение dual value

double **GetDualValue** (string *constr_name*)

Получает значение dual value по имени ограничения

param constr_name

имя ограничения

return

Значение dual value

double **GetReducedCost** (*Variable* *var*)

Получает значение reduced cost

param var

переменная

return

Значение reduced cost

double **GetReducedCost** (string *var_name*)

Получает значение reduced cost

param var

имя переменной

return

Значение reduced cost

double **GetExpressionValue** (*LinearExpression* *expression*)

Получает значение выражения

param expression

линейное выражение

return

Значение выражения

```
double GetExpressionValue (QuadExpression expression)
```

Получает значение выражения

param expression

квадратичное выражение

return

Значение выражения

```
void WriteSolution (string file_name)
```

Записывает решение в файл

param file_name

путь к файлу решения для записи

```
class arhiplex.Variable : IDisposable
```

Класс для работа с переменной - чтение/изменение имени, верхней и нижней границы, типа, а также удаление

Public Functions

```
void SetUpperBound (double upper_bound)
```

Задаёт верхнюю границу для переменной

param upper_bound

новая верхняя граница переменной

```
double GetUpperBound ()
```

Получает верхнюю границу переменной

return

верхняя граница переменной

```
void SetLowerBound (double lower_bound)
```

Задаёт нижнюю границу для переменной

param lower_bound

новая нижняя граница переменной

```
double GetLowerBound ()
```

Получает нижнюю границу переменной

return

нижняя граница переменной

```
void Remove ()
```

Помечает переменную как удалённую. Переменная не будет участвовать в расчетах

```
Variable_type GetVarType ()
```

Получает тип переменной

return

тип переменной

```
void SetType (Variable_type type)
```

Задаёт тип переменной в модели

param type

тип переменной

```
string? GetName ()
```

Получает имя переменной

return

имя переменной

```
void SetName (string name)
```

Задаёт имя переменной в модели

param name

имя переменной

```
enum arhiplex.Variable_type
```

Тип переменной

Values:

continuous

непрерывная

binary

бинарная

integer

целочисленная

semicontinuous

полу-непрерывная

semiinteger

полу-целая

```
enum arhiplex.Objective_sense
```

Направление оптимизации целевой функции

Values:

minimize

Минимизация

maximize

Максимизация

```
enum arhiplex.Constraint_sense
```

Знак ограничения между левой частью (выражением) и правой частью (константой)

Values:

`equal`

„==“

`less_equal`

„<=“

`greater_equal`

„>=“

`enum arhiplex.Solve_result`

Результат процесса расчетов

Values:

`success`

Процесс решения успешен - найдено некоторое решение или обнаружена недостижимость модели

`fail`

Процесс решения неудачен - допустимое решение или доказательство недостижимости не получено; подробность доступна в `solution_status`

`remote_invalid_api_key`

Невалидный ключ для удаленного расчета

`remote_api_key_not_set`

Переменная окружения `X_API_KEY` не установлена

`remote_time_amount_is_over`

Не осталось доступного времени для осуществления удаленного расчета

`remote_time_per_calc_violated`

Превышен лимит времени для единичного удаленного расчета. Параметр `time_limit` нарушает ограничения лицензии

`remote_fail`

Удаленный расчет неудачен

`enum arhiplex.Solution_status`

Статус решения по результатам расчета

Values:

`invalid_solution`

неопределенное/невалидное значение

optimal

решение оптимально (погрешность в рамках заданного значения)

feasible

решение найдено, но не оптимально (погрешность $>$ заданного значения)

infeasible

модель не имеет решения (решение недостижимо)

unbounded

модель неограничена (целевая функция может бесконечно неограниченно увеличиваться/уменьшаться)

infeasible_or_unbounded

модель недостижима или неограничена

imprecise

найден кандидат с конечной целевой функцией, но его нарушения превышают заданные допуски; это не допустимое решение

Java API использует тот же C ABI, что и C# API, через [JNA](#). Он поставляется готовым **самодостаточным JAR** — собирать его из исходников дистрибутива не требуется.

20.1 Требования

- JDK 11 или новее
- Собранная native-библиотека `libarhiplex_cpp` (`.dll` в Windows, `.so` в Linux) — входит в дистрибутив солвера
- Maven — только если вы собираете JAR из исходников (см. ниже)

20.2 Установка (готовый JAR)

Готовый файл `arhiplex-java-<версия>.jar` входит в дистрибутив солвера и находится в `${ARHIPLEX_HOME}/lib64/java/`. Положите его на classpath своего приложения. JAR платформонезависим и уже содержит зависимость JNA — ничего дополнительно скачивать не нужно.

```
javac -cp arhiplex-java-<версия>.jar MyApp.java
java -cp .:arhiplex-java-<версия>.jar MyApp      # Windows: разделитель ';'

```

Версия JAR совпадает с версией продукта.

20.3 Сборка из исходников (необязательно)

Если нужно собрать JAR самостоятельно, потребуется Maven 3.6+:

```
cd src/interfaces/java
mvn package          # результат: target/arhiplex-java-<версия>.jar
```

Через CMake (опциональная сборка JAR при сборке дистрибутива):

```
cmake -DBUILD_JAVA_API=ON ..
cmake --build . --target arhiplex_java_jar
```

20.4 Native-библиотека

Убедитесь, что `libarhiplex_cpp` доступна JVM (в JAR она не входит и ставится отдельно как часть дистрибутива солвера):

- **Linux:** `export LD_LIBRARY_PATH=${ARHIPLEX_HOME}/lib64:${LD_LIBRARY_PATH}`
- **Windows:** добавьте каталог с `libarhiplex_cpp.dll` в `PATH` или укажите `-Djava.library.path=.`
..

20.5 Управление ресурсами

Все классы с native-дескриптором реализуют `AutoCloseable`. Используйте `try-with-resources` для `Model`, `SolveResult`, `Constraint` и временных выражений.

Методы `getVariable`, `getConstraint`, `getObjective`, вложенные `getter`'ы выражений и аналогичные возвращают **новый** native-обёрточный объект на каждый вызов; его нужно закрывать (как и объекты, возвращаемые `addVariable` / `addConstraint`). **Не** вызывайте цепочки вида `model.getConstraint("c").getName()` — каждый промежуточный `wrapper` нужно закрывать отдельно. `addConstraints(Map)` закрывает обёртки автоматически; для одиночного `addConstraint(...)` используйте `try-with-resources`, если возвращаемый `Constraint` не сохраняете.

```
try (Model model = new Model()) {
    try (SolveResult result = model.solve()) {
        // ...
    }
}
```

20.6 Быстрый старт

```
1 import com.arhiplex.*;
2
3 /**
4  * ArhiPlex Java API example (mirrors example_dot_net.cs).
5  */
6 class ExampleJava {
```

(continues on next page)

(продолжение с предыдущей страницы)

```

7  public static void main(String[] args) {
8      try (Model m = new Model("model_1")) {
9          Variable x1 = m.addVariable(-Model.getInf(), Model.getInf(), 0,
10             VariableType.CONTINUOUS, "x1");
11          Variable x2 = m.addVariable(-Model.getInf(), Model.getInf(), 0,
12             VariableType.CONTINUOUS, "x2");
13          Variable x3 = m.addVariable(1.1, 10, 0, VariableType.CONTINUOUS, "x3");
14          Variable x4 = m.addVariable(-Model.getInf(), Model.getInf(), 0,
15             VariableType.CONTINUOUS, "x4");
16
17          m.addConstraint(x1.plus(x2).le(5), "c1");
18          m.addConstraint(x1.times(2).minus(x2).ge(0), "c2");
19          m.addConstraint(x1.negate().plus(x2.times(3)).ge(0), "c3");
20          m.addConstraint(x3.plus(x4).ge(0.5), "c4");
21
22          m.setObjective(
23              x1.plus(x2.times(2)).minus(x3.times(0.1)).minus(x4.times(3)),
24              ObjectiveSense.MAXIMIZE);
25
26          m.setDblParam("time_limit", 30);
27
28          try (SolveResult solution = m.solve()) {
29              SolveResultStatus solveResult = solution.getSolveResult();
30              SolutionStatus solutionStatus = solution.getSolutionStatus();
31
32              if (solveResult == SolveResultStatus.SUCCESS
33                  && (solutionStatus == SolutionStatus.OPTIMAL
34                      || solutionStatus == SolutionStatus.FEASIBLE)) {
35                  double gap = solution.getRelativeGap();
36                  double time = solution.getSolveTime();
37                  double x1Value = solution.getVariableValue(x1);
38                  double x2Value = solution.getVariableValue("x2");
39                  double x3Value = solution.getVariableValue("x3");
40                  double x4Value = solution.getVariableValue("x4");
41              }
42          }
43      }
44  }
45  }

```

20.7 Синтаксис выражений

В Java нет перегрузки операторов. Используйте fluent-методы:

C# / C++	Java
$x + y \leq 10$	<code>x.plus(y).le(10)</code>
$2 * x + y$	<code>x.times(2).plus(y)</code>
$x * y$	<code>x.times(y)</code>
$x * y == z$	<code>x.times(y).eq(z)</code>
$x_4 * x_{10} - x_{24} * x_{38} == 0$	<code>x4.times(x10).minus(x24.times(x38)).eq(0)</code>
$x_{10} - x_2 * x_{16} == 0$	<code>x10.minus(x2.times(x16)).eq(0)</code>
<code>model.SetObjective(expr, sense)</code>	<code>model.setObjective(expr, sense)</code>

20.8 Квадратичные (QCQP/MIQCQP) выражения

ArhiPlex поддерживает смешанные линейные и квадратичные цели и ограничения:

Сценарий	Java API
Смешанная цель	<code>LinearExpression + setObjective(expr, sense)</code>
Чисто квадратичная	<code>QuadExpression + setObjective(quad, sense)</code>
Билинейное ограничение	<code>x.times(y).eq(z)</code> или <code>q1.minus(q2).eq(0)</code>
Диапазон на quad	<code>model.addConstraint(quad, lb, ub)</code>

Пример смешанной квадратичной цели:

```
LinearExpression obj = x2.times(-0.0139904)
    .plus(x2.times(x4).times(0.0187028))
    .plus(x3.times(-0.0661588));
model.setObjective(obj, ObjectiveSense.MINIMIZE);
```

Статические хелперы доступны в Expressions (`constant`, `product`, `constantMinus`, `sum`).

20.9 Matrix API

Используйте `MatrixModel` для пакетного построения моделей (аналог `arhiplexpy.MatrixModel`):

```
1 import com.arhiplex.*;
2
3 /**
4  * Matrix MILP example (mirrors matrixMILP.py).
5  */
6 class MatrixMilpExample {
7     public static void main(String[] args) {
8         try (MatrixModel model = new MatrixModel()) {
9             model.addVariables(
10                 3,
11                 new double[] {0.0},
12                 new double[] {100.8, Model.getInf(), 1.0},
```

(continues on next page)

(продолжение с предыдущей страницы)

```

13         new double[] {3.0, 5.0, 2.0},
14         new VariableType[] {
15             VariableType.CONTINUOUS,
16             VariableType.BINARY,
17             VariableType.INTEGER
18         });
19
20     model.setObjective(ObjectiveSense.MAXIMIZE);
21     model.addConstraints(
22         new double[][] {
23             {2, 4, 3},
24             {1, 3, 2},
25             {1, 1, 0},
26             {0, 1, 1},
27             {1, 0, 1}
28         },
29         new ConstraintSense[] {
30             ConstraintSense.LESS_EQUAL,
31             ConstraintSense.LESS_EQUAL,
32             ConstraintSense.LESS_EQUAL,
33             ConstraintSense.LESS_EQUAL,
34             ConstraintSense.GREATER_EQUAL
35         },
36         new double[] {10, 8, 5, 3, 1});
37
38     model.setDblParam("time_limit", 10);
39     try (MatrixSolveResult result = model.solve()) {
40         if (result.getSolveResult() == SolveResultStatus.SUCCESS) {
41             System.out.println("objective: " + result.
42 ↪getObjectiveFunctionValue());
43             System.out.println("solution: " + java.util.Arrays.toString(result.
44 ↪getSolutionVector()));
45         }
46     }
47 }

```

20.10 Общие (нелинейные) выражения

```
model.addGeneralConstraint(MathFunc.sin(x1.times(2.5)).plus(x2).eq(y, "n11"));
```

20.11 IIS (недостижимость)

```
try (ComputeIisResult iis = model.computeIis()) {
    iis.writeIis("iis.lp");
}
```

20.12 Bulk-векторы и remote result

После локального или матричного решения можно получить полный вектор значений переменных и (для непрерывных LP/QP задач) векторы двойственных оценок и reduced costs одним вызовом, без обхода переменных по одной:

```
try (SolveResult result = model.solve()) {
    double[] x = result.getSolutionVector();
    double[] duals = result.getDualValuesVector();
    double[] rc = result.getReducedCostsVector();
}
```

`getSolutionVector()` возвращает копию значений всех переменных модели в порядке индексов. `getDualValuesVector()` и `getReducedCostsVector()` имеют смысл для задач без целочисленных переменных (аналогично точечным методам `getDualValue(constraint)` / `getReducedCost(variable)`).

Для асинхронного удалённого расчёта (`solveRemoteAsync`) результат можно загрузить по UID расчёта и опциональному файлу сопоставления имён:

```
try (SolveResult remote = new SolveResult(calcUid, nameMappingFile)) {
    double[] x = remote.getSolutionVector();
}
```

Объекты `SolveResult` владеют native-результатом: закрывайте их через try-with-resources.

20.13 Логирование

```
model.setLogCallback(msg -> System.out.println(msg));
```

20.14 Тесты

Интеграционные JUnit-тесты Java API находятся в модуле `check/java_api_test`:

- `ArhiplexApiTest` — базовые операции модели, выражения, `toString`, параметры, remote smoke.
- `JavaParityTest` — паритет с Python API: QCQP, `MatrixModel`, general constraints, IIS, bulk-векторы.

Требования для запуска:

- собранная `libarhiplex_cpp` в `LD_LIBRARY_PATH` / `PATH`;
- установленный JAR `arhiplex-java` (`mvn install` в `src/interfaces/java`);
- Maven и JDK 11+.

Запуск из корня репозитория:

```
cd check/java_api_test
mvn test
```

Тесты не входят в пользовательский дистрибутив: они предназначены для проверки API при разработке и в CI.

20.15 Справочник API

Java API публикуется в пакете `com.arhiplex`. Ниже приведён справочник основных типов; подробные сигнатуры см. в исходниках, поставляемых вместе с дистрибутивом (`include/arhiplex/java/src`), и в тестах `check/java_api_test`.

Таблица 20.1: Классы Java API

Тип	Назначение
<code>Model</code>	Объектная модель: переменные, ограничения, цель, параметры, запуск решения.
<code>MatrixModel</code>	Пакетное построение моделей через массивы/матрицы.
<code>Variable</code>	Переменная модели: имя, тип и границы.
<code>Constraint</code>	Линейное или квадратичное ограничение.
<code>LinearExpression</code>	Линейное выражение и операции над ним.
<code>QuadExpression</code>	Квадратичное выражение и операции над ним.
<code>GeneralExpression</code>	Нелинейное выражение для general constraints.
<code>GeneralConstraint</code>	Нелинейное ограничение.
<code>Expressions</code>	Статические хелперы для построения выражений.
<code>MathFunc</code>	Математические функции для нелинейных выражений.
<code>SolveResult</code>	Результат решения объектной модели.
<code>MatrixSolveResult</code>	Результат решения матричной модели.
<code>ComputeIisResult</code>	Результат вычисления IIS.
<code>ArhiplexException</code>	Исключение Java API.

Таблица 20.2: Перечисления Java API

Перечисление	Назначение
<code>VariableType</code>	Тип переменной: непрерывная, целочисленная, бинарная.
<code>ObjectiveSense</code>	Направление оптимизации.
<code>ConstraintSense</code>	Смысл ограничения.
<code>SolveResultStatus</code>	Статус запуска решения.
<code>SolutionStatus</code>	Статус найденного решения.
<code>GeneralExpressionType</code>	Тип нелинейного/general expression.
<code>ComputeIisResultStatus</code>	Статус вычисления IIS.
<code>ModelStatus</code>	Статус модели для IIS/диагностики.

Пакет `Arhiplex.jl` предоставляет интерфейс к решателю ArhiPlex (LP/MIP/QP) для языка Julia через `MathOptInterface` и моделировщик JuMP. Решатель используется так же, как любой другой оптимизатор JuMP — через `Model(Arhiplex.Optimizer)`.

21.1 Подключение пакета

Пакет `Arhiplex.jl` поставляется в составе дистрибутива в каталоге `lib64/Arhiplex.jl`. Нативная библиотека ищется по переменной окружения `ARHIPLEX_HOME` (в `ARHIPLEX_HOME/lib64`); единственная обязательная зависимость пакета — `MathOptInterface`, JuMP устанавливается отдельно.

```
import Pkg
# Путь к поставляемому пакету
arhiplex_jl = joinpath(ENV["ARHIPLEX_HOME"], "lib64", "Arhiplex.jl")
Pkg.develop(; path = arhiplex_jl)
Pkg.add("JuMP")

using JuMP
using Arhiplex
```

Внимание: Переменная окружения `ARHIPLEX_HOME` должна указывать на корень дистрибутива; библиотека `libarhiplex_cpp` загружается из `ARHIPLEX_HOME/lib64`. Если переменная не задана, библиотека ищется стандартным загрузчиком (`LD_LIBRARY_PATH`).

21.2 Базовый пример

Задача линейного программирования: максимизировать $3x + 4y$ при ограничениях.

```
using JuMP
using Arhiplex

model = Model(Arhiplex.Optimizer)

@variable(model, x >= 0)
@variable(model, y >= 0)

@objective(model, Max, 3x + 4y)

@constraint(model, c1, 2x + y <= 10)
@constraint(model, c2, x + 2y <= 12)
@constraint(model, c3, x + y <= 8)

optimize!(model)

println(termination_status(model))    # OPTIMAL
println(objective_value(model))        # 26.666...
println(value(x), " ", value(y))
```

21.3 Поддерживаемые конструкции

- **Типы переменных:** непрерывные, целочисленные (Int), бинарные (Bin).
- **Ограничения на переменные:** MOI.LessThan, MOI.GreaterThan, MOI.EqualTo, MOI.Interval (двусторонние границы).
- **Линейные ограничения:** ScalarAffineFunction в тех же множествах, включая диапазонные ограничения вида $lb \leq a'x \leq ub$.
- **Целевая функция:** линейная, минимизация или максимизация.

```
@variable(model, 0 <= x <= 10, Int)    # целочисленная с границами
@variable(model, z, Bin)                # бинарная
@constraint(model, 3 <= x + y <= 8)      # диапазонное ограничение
```

21.4 Параметры решателя

Параметры задаются через стандартный механизм JuMP. Имена совпадают с параметрами ArhiPlex (time_limit, mip_rel_gap, mip_abs_gap и др.).

```
set_optimizer_attribute(model, "time_limit", 60.0)
set_optimizer_attribute(model, "mip_rel_gap", 1e-4)

set_silent(model)                        # отключить вывод решателя
set_time_limit_sec(model, 60.0)          # эквивалент параметру time_limit
```

21.5 Чтение результатов

```
optimize!(model)

termination_status(model)  # OPTIMAL, INFEASIBLE, OTHER_LIMIT, ...
primal_status(model)       # FEASIBLE_POINT / NO_SOLUTION
objective_value(model)
value(x)                   # значение переменной

# Двойственные значения (только для непрерывных задач)
dual(c1)                   # теневая цена ограничения
dual(LowerBoundRef(x))     # приведённая стоимость переменной

# Доп. показатели
solve_time(model)
relative_gap(model)        # относительный разрыв (MIP)
MOI.get(unsafe_backend(model), MOI.NodeCount())
```

Примечание: Для целочисленных (MIP) задач двойственные значения не определены: `dual_status(model)` вернёт `NO_SOLUTION`.

21.6 Начальное решение (MIP start)

Для MIP-задач можно задать стартовые значения переменных.

```
@variable(model, 0 <= x <= 10, Int)
set_start_value(x, 5.0)
optimize!(model)
```

CP-решатель (ArhiPlex CP)

В дополнение к классическим методам LP/MIP/QP/MIQP/NLP/MINLP солвер **ArhiPlex** поддерживает задачи **constraint programming (CP)** через **ArhiPlex CP**.

ArhiPlex CP ориентирован на комбинаторные и логические модели, в которых важно не только значение целевой функции, но и структура допустимых комбинаций:

- булевы и целочисленные решения;
- логические связи и импликации;
- ограничения `all_different` и `exactly_one`;
- интервальные переменные и задачи расписаний;
- перебор всех решений для задач выполнимости.

22.1 Когда использовать ArhiPlex CP

CP-решатель особенно полезен, если модель содержит:

- дискретные решения и сложные логические зависимости;
- распределение ресурсов, назначение работ, календарные ограничения;
- задачи последовательности, непересечения интервалов и расписаний;
- требования перечислить несколько или все допустимые решения.

Примечание: Для классических LP/MIP/QP/NLP-задач следует продолжать использовать основной API `arhiplexу.Arhiplex`. ArhiPlex CP не заменяет линейный/квадратичный солвер, а дополняет его для другого класса задач.

22.2 Доступность в сборке

Поддержка ArhiPlex CP является **опциональной** и зависит от того, была ли ваша сборка ArhiPlex собрана с BUILD_CP_SOLVER / ARHIPLEX_WITH_CP_SOLVER.

Проверить доступность можно так:

```
import arhiplexpy
print(arhiplexpy.has_cp_solver())
```

Если функция вернула False, CP-часть не была включена в сборку или не была установлена вместе с Python-пакетом.

22.3 Импорт CP API

Основной модуль ArhiPlex CP импортируется так:

```
from arhiplexpy.arhitools.sat.python import cp_model
```

После этого становятся доступны:

- `cp_model.CpModel` для построения модели;
- `cp_model.CpSolver` для запуска решения;
- `cp_model.CpSolverSolutionCallback` для callback-обработчиков;
- константы статусов `OPTIMAL`, `FEASIBLE`, `INFEASIBLE`, `MODEL_INVALID`, `UNKNOWN`.

22.4 Базовый пример выполнимости

Ниже показан простой пример задачи выполнимости с двумя целочисленными переменными:

```
1 from arhiplexpy.arhitools.sat.python import cp_model
2
3
4 def main() -> None:
5     model = cp_model.CpModel()
6
7     rabbits = model.new_int_var(0, 100, "rabbits")
8     pheasants = model.new_int_var(0, 100, "pheasants")
9
10    model.add(rabbits + pheasants == 20)
11    model.add(4 * rabbits + 2 * pheasants == 56)
12
13    solver = cp_model.CpSolver()
14    status = solver.solve(model)
15
16    if status == cp_model.OPTIMAL:
17        print(f"rabbits = {solver.value(rabbits)}")
18        print(f"pheasants = {solver.value(pheasants)}")
19    else:
```

(continues on next page)

(продолжение с предыдущей страницы)

```

20     print(f"Unexpected status: {solver.status_name(status)}")
21
22
23 if __name__ == "__main__":
24     main()

```

В этом примере:

- создаётся объект `CpModel`;
- переменные объявляются через `new_int_var()`;
- ограничения добавляются методом `add()`;
- результат проверяется по статусу `OPTIMAL`.

22.5 Пример оптимизации

Следующий пример показывает постановку целочисленной задачи с линейной целевой функцией:

```

1  from arhiplexpy.arhitools.sat.python import cp_model
2
3
4  def main() -> None:
5      model = cp_model.CpModel()
6
7      upper_bound = max(50, 45, 37)
8      x = model.new_int_var(0, upper_bound, "x")
9      y = model.new_int_var(0, upper_bound, "y")
10     z = model.new_int_var(0, upper_bound, "z")
11
12     model.add(2 * x + 7 * y + 3 * z <= 50)
13     model.add(3 * x - 5 * y + 7 * z <= 45)
14     model.add(5 * x + 2 * y - 6 * z <= 37)
15     model.maximize(2 * x + 2 * y + 3 * z)
16
17     solver = cp_model.CpSolver()
18     status = solver.solve(model)
19
20     if status in (cp_model.OPTIMAL, cp_model.FEASIBLE):
21         print(f"objective = {solver.objective_value}")
22         print(f"x = {solver.value(x)}")
23         print(f"y = {solver.value(y)}")
24         print(f"z = {solver.value(z)}")
25         print(f"status = {solver.status_name(status)}")
26         print(f"conflicts = {solver.num_conflicts}")
27         print(f"branches = {solver.num_branches}")
28         print(f"wall_time = {solver.wall_time:.6f}")
29     else:
30         print(f"No solution: {solver.status_name(status)}")
31
32

```

(continues on next page)

(продолжение с предыдущей страницы)

```

33 if __name__ == "__main__":
34     main()

```

После решения можно получить:

- значение цели через `solver.objective_value`;
- значения переменных через `solver.value(var)`;
- статистику поиска через `solver.num_conflicts`, `solver.num_branches` и `solver.wall_time`.

22.6 Логические и комбинаторные ограничения

Наиболее часто используемые методы `CpModel`:

Таблица 22.1: Основные конструкции ArhiPlex CP

Метод	Назначение
<code>add(expr)</code>	линейные ограничения и равенства/неравенства над целочисленными выражениями
<code>add_bool_or(...)</code>	дизъюнкция над булевыми литералами
<code>add_bool_and(...)</code>	конъюнкция над булевыми литералами
<code>add_implication(a, b)</code>	логическая импликация $a \rightarrow b$
<code>add_all_different(...)</code>	попарное различие значений переменных
<code>add_exactly_one(...)</code>	ровно один литерал должен быть истинным
<code>add_at_most_one(...)</code>	не более одного истинного литерала
<code>add_element(index, values, target)</code>	выбор элемента массива по индексу
<code>add_circuit(arcs)</code>	циклы/маршруты в задачах обхода и последовательностей

22.7 Интервалы и задачи расписаний

Для scheduling-задач ArhiPlex CP предоставляет интервальные переменные и специальные ограничения непересечения.

```

1  from arhiplexpy.arhitools.sat.python import cp_model
2
3
4  def main() -> None:
5      model = cp_model.CpModel()
6      horizon = 20
7
8      start_a = model.new_int_var(0, horizon, "start_a")
9      end_a = model.new_int_var(0, horizon, "end_a")
10     task_a = model.new_interval_var(start_a, 3, end_a, "task_a")
11
12     start_b = model.new_int_var(0, horizon, "start_b")
13     end_b = model.new_int_var(0, horizon, "end_b")
14     task_b = model.new_interval_var(start_b, 4, end_b, "task_b")
15

```

(continues on next page)

(продолжение с предыдущей страницы)

```

16 start_c = model.new_int_var(0, horizon, "start_c")
17 end_c = model.new_int_var(0, horizon, "end_c")
18 task_c = model.new_interval_var(start_c, 2, end_c, "task_c")
19
20 model.add_no_overlap([task_a, task_b, task_c])
21
22 makespan = model.new_int_var(0, horizon, "makespan")
23 model.add_max_equality(makespan, [end_a, end_b, end_c])
24 model.minimize(makespan)
25
26 solver = cp_model.CpSolver()
27 status = solver.solve(model)
28
29 if status == cp_model.OPTIMAL:
30     print(f"makespan = {solver.objective_value}")
31     print(f"task_a starts at {solver.value(start_a)}")
32     print(f"task_b starts at {solver.value(start_b)}")
33     print(f"task_c starts at {solver.value(start_c)}")
34 else:
35     print(f"No optimal solution: {solver.status_name(status)}")
36
37
38 if __name__ == "__main__":
39     main()

```

В этом примере используются:

- `new_interval_var(start, size, end, name)` для задания работ;
- `add_no_overlap(...)` для запрета пересечения работ на одном ресурсе;
- `add_max_equality(...)` для вычисления makespan;
- `minimize(...)` для минимизации длины расписания.

22.8 Callback и поиск всех решений

Если требуется перечислить несколько решений, можно использовать callback-класс `CpSolverSolutionCallback` и включить полный перебор решений.

```

1 from arhiplexpy.arhitools.sat.python import cp_model
2
3
4 class NQueensPrinter(cp_model.CpSolverSolutionCallback):
5     def __init__(self, queens):
6         super().__init__()
7         self._queens = queens
8         self.solution_count = 0
9
10    def on_solution_callback(self) -> None:
11        self.solution_count += 1
12        print(f"solution #{self.solution_count}")

```

(continues on next page)

(продолжение с предыдущей страницы)

```

13     for row in range(len(self._queens)):
14         line = []
15         for col in range(len(self._queens)):
16             line.append("Q" if self.value(self._queens[col]) == row else ".")
17         print(" ".join(line))
18     print()
19
20
21 def main(board_size: int = 4) -> None:
22     model = cp_model.CpModel()
23
24     queens = [model.new_int_var(0, board_size - 1, f"q_{i}") for i in range(board_size)]
25     model.add_all_different(queens)
26     model.add_all_different(queens[i] + i for i in range(board_size))
27     model.add_all_different(queens[i] - i for i in range(board_size))
28
29     solver = cp_model.CpSolver()
30     solver.parameters.enumerate_all_solutions = True
31     printer = NQueensPrinter(queens)
32     solver.solve(model, printer)
33
34     print(f"solutions = {printer.solution_count}")
35     print(f"branches = {solver.num_branches}")
36     print(f"wall_time = {solver.wall_time:.6f}")
37
38
39 if __name__ == "__main__":
40     main()

```

Ключевые элементы:

- наследование от `CpSolverSolutionCallback`;
- реализация метода `on_solution_callback()`;
- установка `solver.parameters.enumerate_all_solutions = True`;
- получение промежуточных значений через `self.value(...)` внутри callback.

22.9 Статусы решения

Чаще всего используются следующие статусы:

- `OPTIMAL` - найдено оптимальное решение;
- `FEASIBLE` - найдено допустимое решение;
- `INFEASIBLE` - допустимых решений не существует;
- `MODEL_INVALID` - модель задана некорректно;
- `UNKNOWN` - решатель не смог завершить поиск с определённым итогом.

Человекочитаемое имя статуса можно получить методом:

```
solver.status_name(status)
```

22.10 Параметры решателя

Параметры ArhiPlex CP доступны через `solver.parameters`. На практике часто используются:

- `max_time_in_seconds` - лимит времени;
- `num_search_workers` - число потоков поиска;
- `random_seed` - инициализация генератора случайных чисел;
- `log_search_progress` - подробный лог поиска;
- `enumerate_all_solutions` - перечисление всех решений.

Пример:

```
solver = cp_model.CpSolver()  
solver.parameters.max_time_in_seconds = 60.0  
solver.parameters.num_search_workers = 8  
solver.parameters.log_search_progress = True
```

22.11 Ограничения и совместимость

При работе с ArhiPlex CP важно учитывать:

- ArhiPlex CP рассчитан прежде всего на **целочисленные** и **булевы** переменные;
- формат модели задаётся программно через `CpModel`;
- ArhiPlex CP не использует файловые форматы `.lp` / `.mps` как основной интерфейс;
- для линейных, квадратичных и общих нелинейных задач следует использовать основной солвер ArhiPlex.

22.12 См. также

- *Python API*
- *Быстрый старт*

23.1 Версия 2.7.0

Обновления упрощают интеграцию ArhiPlex в приложения и пересчёт планов, сокращают накладные расходы на серии задач и повышают надёжность работы с результатами. Релиз объединяет изменения третьего квартала 2026 года и октябрьские исправления импорта MPS и повторных расчётов.

23.1.1 Новые возможности

- Пересчёт с частично заданным решением: ArhiPlex достраивает и проверяет известную часть плана перед основным поиском. Доступны режимы использования старта и отдельные лимиты на его подготовку.
- Готовая библиотека для Java в составе дистрибутива позволяет подключать ArhiPlex к приложениям и сервисам без сборки библиотеки из исходников.
- Обновлён OEM-режим для встраивания ArhiPlex в партнёрские продукты.

23.1.2 Улучшения и исправления

- Сокращены затраты на подготовку матричных моделей, повторный запуск небольших задач и завершение рабочих процессов. Устранено накопление открытых файлов журналов в длительных сериях.
- Исправлены передача коэффициентов, преобразование выражений и проверка решений по исходным условиям модели. Исправлена нормализация квадратичной цели при чтении MPS.
- Уточнены статусы неограниченных задач, остановки по лимиту и численно неточного результата, включая интерфейс Julia. Исправлены оценка точности около нулевой цели, сохранение лучшего решения и итоговое время. При повторных расчётах устранена потеря статуса оптимального решения из-за округления.

- Исправлены поиск противоречащих ограничений, чтение и запись моделей с особыми именами. Устранены расхождения между решением в API и сохранённым файлом; уточнена обратная загрузка начальных решений.
- Улучшены сообщения об ошибках запуска и обмен прогрессом. Исправлены взаимное влияние лицензированных сессий, редкие отказы при массовом старте и освобождение ресурсов при завершении расчётов Python.
- Исправлены комплектность C++ API и обработка параметров. Обновлено руководство, справочник настроек и инструкции по анализу результатов; унифицировано выполнение расчётов и актуализирован состав поставки.

23.1.3 При обновлении

Устанавливайте дистрибутив целиком. Приложения с прямым подключением низкоуровневого C++ ядра требуют пересборки; часть прежних низкоуровневых операций больше не поддерживается.

23.1.4 Известное ограничение

Сохраняется замедление при построении больших выражений встроенным Python `sum`. Исправление этого ограничения в релиз не включено.

23.2 Версия 2.6.2: допустимость решений MILP

- Исправлена передача допусков целочисленности и выполнения ограничений вычислительным модулям. Их назначения описаны в разделе *Параметры расчёта*.
- Перед публикацией решения проверяются ограничения, границы переменных и целочисленность. Конечная целевая функция в прогрессе сама по себе не означает, что найдено допустимое решение.
- Добавлен статус `solution_status.imprecise` (код 6): найденный кандидат превышает заданные допуски, другого допустимого решения нет. При этом `get_solve_result()` возвращает `fail`, значения переменных и файл решения недоступны. Прежние числовые коды статусов сохранены.
- При параллельном поиске сохраняется лучшее проверенное допустимое решение. Результат, не достигший заданного `gap`, не останавливает работающие модули до исчерпания общего лимита времени. Если поиск завершён без достижения `gap`, допустимое решение возвращается со статусом `feasible`.
- При выборе решения ранее завершившегося модуля итоговое время отражает длительность всего параллельного поиска, а не только работу этого модуля.
- Уточнены порядок проверки *статусов* и вывод малых значений параметров в журнале расчёта.

23.3 Версия 2.0.11

23.3.1 Оптимизация производительности

- Ускорено добавление термов в линейные выражения
- Улучшена обработка больших моделей
- Ускорено создание линейных функций. Исправлена проблема с производительностью при добавлении более чем 10^4 переменных в цикле

23.3.2 Новый Matrix API

- Реализован базовый функционал для создания и работы с матричными моделями
- Добавлена поддержка remote solve для матричных моделей
- Обновлено документация. Добавлены примеры использования

23.3.3 Поддержка Python

- Добавлена поддержка Python 3.12
- Улучшена поддержка IntelliSense в Python
- Обновлено подсказки типов для основных API-функций

23.3.4 Улучшения ArhiPlex Remote Client/Оптимизация облачных вычислений

- Ускорена загрузка данных в облако
- Исправлен вывод логов при длительных удалённых расчётах
- Добавлена поддержка работы через прокси-серверы
- Оптимизировано отображение прогресса вычислений, добавлен индикатор прогресса загрузки в процентах

23.3.5 Общие улучшения

- Реализован вывод системного времени в лог-файлы

23.3.6 Документационные правки

- Разделена документация для версий 1.x и 2.x
- Обновлено информация по лицензиям и их активации
- Добавлена документация по Matrix API и примеры

23.4 Версия 2.1.0

Новая версия ArhiPlex 2.1.0

23.4.1 Новые возможности

- Добавлен расчет IIS

23.4.2 Общие улучшения

- Стабилизация механизма подстановки начального решения

23.4.3 Документационные правки

- Добавлен раздел Анализ недостижимости модели и примеры использования расчета IIS

23.5 Версия 2.2.0

Новая версия ArhiPlex 2.2.0

23.5.1 Оптимизация производительности

- Улучшение вычислительных алгоритмов, сокращающее время оптимизации задач MILP в 1,5 раза

23.5.2 Новые возможности

- Полное англоязычное сопровождение

23.5.3 Документационные правки

- Добавлено Руководство пользователя на английском языке

23.6 Версия 2.4.0

Новая версия ArhiPlex 2.4.0

23.6.1 Оптимизация производительности

- Улучшение вычислительных алгоритмов, сокращающее время оптимизации задач LP/MILP/MIQCP

23.6.2 Новые возможности

- Поддержка нелинейной оптимизации.

23.7 Версия 2.5.0

Новая версия ArhiPlex 2.5.0

23.7.1 Новые возможности

- Добавлена поддержка CP-задач через ArhiPlex CP.

23.7.2 Документационные правки

- Добавлен отдельный раздел по CP-решателю, Python API и примерам constraint programming.

23.8 Версия 2.6.0

23.8.1 Новые возможности

- Добавлен программный интерфейс на языке Java: готовая к использованию библиотека (JAR) поставляется в составе дистрибутива.
- Добавлен интерфейс для языка Julia (JuMP/MOI) с интеграцией в экосистему JuMP.

23.8.2 Общие улучшения

- Русскоязычный консольный вывод CP-решателя (ArhiPlex CP).
- Исправлены статус решения и вычисление гар для задач с оптимумом около нуля.
- Исправлено определение класса модели для задач с обобщёнными ограничениями (general constraints).

23.8.3 Документационные правки

- Добавлены разделы по Java API и Julia API с примерами использования.

23.9 Версия 2.6.1

23.9.1 Исправления

- В многопроцессном режиме сообщение об ошибке запуска решения теперь содержит конкретную причину (нехватка системных ресурсов хоста — процессы, память, файловые дескрипторы) и рекомендацию по настройке.

23.9.2 Документационные правки

- Добавлено описание параметра `multiprocess_max_process_count` и рекомендация по параллельному запуску нескольких расчётов на одном хосте.

23.10 Версия 2.6.2

23.10.1 Исправления

- Исправлено определение терминального статуса неограниченных LP, MILP и QP в многопроцессном enterprise-режиме: результат больше не подменяется статусами «Оптимально» или «Не задан».
- Исправлено вычисление IIS для недостижимых моделей: возвращается непустое корректное IIS.
- Исправлено сохранение IIS для моделей со служебными символами в именах: итоговый LP-файл остаётся корректным и повторно читается ArhiPlex. Файлы с расширением `.ilp` распознаются как формат LP.
- Исправлено отображение статусов ArhiPlex при остановке по лимиту или прерыванию без найденного допустимого решения: такие случаи больше не объявляются доказанной недостижимостью модели.

Поддержка продукта солвера **ArhiPlex** (далее — солвер) разработчиком вендором, компанией «**Лаборатория Корпоративного Сопровождения**», осуществляется на всём продолжении жизненного цикла программного обеспечения (далее — ПО) **ArhiPlex**.

24.1 Возникновение проблемы и создание обращения пользователя

При возникновении ошибок в процессе использования, в расчетах солвера, пользователи могут ознакомиться с документацией самостоятельно и, возможно, найти ответ на возникший вопрос или проблему. В случае, если проблема не освещена в документации, пользователи могут направлять обращения на почту **info@arhitex.com** или по телефону **+7 (812) 245-68-00**. При подаче обращения по телефону, необходимо сообщить наименование компании, версию солвера и суть возникшей проблемы или вопроса. При подаче обращения по почте, рекомендуется приложить информацию о ходе решения задачи (логи работы) в солвере и данные при расчёте которых, возникла проблема. В случае выявления проблем эксплуатации солвера в среде использования или взаимодействия с солвером сторонних приложений, необходимо приложить логи сторонних систем, которые содержат сообщения об ошибках.

24.2 Обработка и анализ обращений

После получения обращения (по телефону или почте), в течение 8 часов, пользователю направляется плановая дата и время предоставления результатов анализа или запрос на дополнительные данные. Ответ на обращение должен быть направлен не позднее чем через 72 часа после поступления заявки. Срок запроса дополнительных данных не может превышать 48 часов.

Результат анализа обращения должен содержать:

- указание на особенность примера, на котором возникла проблема;
- указание на массовый характер заявки;

- версию ПО, в которой будет исправлена данная проблема;
- сроки выхода данной версии.

24.3 Разработка новой версии (спецверсии) солвера

По результатам анализа проблемы выявляется ошибка, которая вносится в дорожную карту развития солвера.

В дорожной карте данная ошибка получает высокий приоритет и будет взята в первую очередь работ к выпуску новой версии солвера.

Если ошибка носит локальный характер и не включена в дорожную карту, спецверсия может быть собрана, не дожидаясь очередного выпуска в графике выпусков солвера.

24.4 Выпуск новой версии (спецверсии) солвера

Процесс выпуска новой версии включает в себя, как сборку самого ПО, так и обновление документации к ПО. В составе документации присутствуют примечания к версии продукта (release-notes), публикуемые на сайте поставщика ПО. По отдельному запросу клиента может быть предоставлен расширенный список release-notes.

В процессе выпуска спецверсии, общая документация продукта не обновляется, однако клиенту высылаются инструкция, касательно внесенных в конкретную спецверсию исправлений.

24.5 Информирование клиента

Процесс информирования клиента включает в себя:

- работу с обращениями и вопросами клиентов;
- работу по донесению информации о новых версиях, их особенностях, о существующих проблемах и уязвимостях ПО, о планируемых изменениях в ПО в будущих релизах.

Процесс информирования осуществляется по электронной почте, на сайте вендора и при передаче release-notes клиенту.

24.5.1 Информирование о возможном решении проблемы без поставки и установки новой версии

Возможны ситуации, в которых поставка новой версии ПО солвера **ArhiPlex** не требуется. Устранение проблемы может быть решено путем настройки существующего ПО или предложенным разработчиком обходным решением. В таком случае, поддержка компании разработчика сообщает клиенту о возможном пути решения проблемы по электронной почте. В случае решения проблемы/вопроса обращение закрывается.

24.5.2 Информирование клиента о существующих уязвимостях в ПО или в используемых библиотеках

Информирование клиента о найденных уязвимостях различных уровней критичности, происходит посредством включения информации о данных уязвимостях в release-notes, передаваемой клиенту версии. Для критических уязвимостей самого ПО и компонент, в него входящих, указывается дата исправления или ссылка на источники, от которых эта дата зависит. Информация о существующих уязвимостях передается в виде документа — приложения к передаваемому клиенту версии солвера релизу.

24.5.3 Информирование клиента о выходе новой версии

В ходе развития продукта, выходят новые версии солвера, которые содержат как исправления найденных ошибок, так и новый функционал.

Информация о данных версиях публикуется на сайте. Клиенты могут подписаться на рассылку сообщений по электронной почте о выходе новых версий ПО на официальном сайте вендора.

24.5.4 Информирование клиента о решении проблемы

После решения проблемы и получения подтверждения решения проблемы от клиента, обращение может быть закрыто, о чем будет направлено сообщение по электронной почте на указанный контактный адрес клиента.

Контактная информация

Обращения, вопросы и предложения по работе и доработке продукта вы можете оставить по телефону +7 (812) 245-68-00 или по электронному адресу info@arhitex.com.

a

`arhiplexpy`, [76](#)

`arhiplexpy.arhiplexpy`, [76](#)

`arhiplexpy.arhitools.sat.python.cp_model`, ??