
Справочник API

Выпуск 2.6.1.242987

июл. 15, 2026

1	Введение	1
2	Установка	2
2.1	Windows	2
2.2	Linux	4
2.3	Установка пакета arhiplexru	5
2.4	Удаление	5
3	Быстрый старт	6
3.1	Онлайн расчёт	6
3.2	Python интерфейс	8
3.3	C++ интерфейс	15
4	C++ API	21
5	C# API	42
6	Python API	62
7	Параметры	64
8	Лицензии	65
8.1	Активация пробной лицензии	66
8.2	Активация лицензии «Профи» и «Бизнес»	68
8.3	Выпуск лицензионного ключа	69
9	Техническая поддержка	71
10	Контактная информация	72
	Содержание модулей Python	73

ArhiCloud – высокопроизводительный облачный солвер, решающий самые сложные задачи бизнеса, осуществляющий решение задач линейного программирования (LP) и линейного целочисленного программирования (MILP).

Создайте учетную запись на портале **ArhiCloud** (<https://arhicloud.arhitex.com/>), чтобы получить бесплатную пробную лицензию. *Обращайтесь к нам* по любым техническим или коммерческим вопросам.

- *Лицензии ArhiCloud*
- *Установка ArhiCloud*

Вы этой главе рассмотрен процесс установки клиентского приложения ArhiCloud, на операционную систему *Windows* и *Linux*. Перед началом установки необходимо:

- проверить соответствие системных требований программы характеристикам вашего компьютера;
- закрыть все активные приложения.

2.1 Windows

2.1.1 Системные требования

- Архитектура x64
- Операционная система: Windows 7 (или выше), Windows Server 2016 (или выше)
- Для библиотеки Python: Python 3.8; 3.11

Компьютер пользователя должен иметь доступ к сети интернет.

2.1.2 Установка на Windows

Для установки клиентского приложения на Windows:

1. Загрузите и запустите клиентское приложение Arhicloud – ArhiplexRemoteClient.
2. Запустите msi файл (используя специальный инструмент — «Запуск от имени администратора». Это необходимо для корректной установки программы, которая требуют доступа к системным ресурсам).
3. Нажмите «Далее» [Рис. 2.1](#).
4. Выберите место установки и нажмите кнопку «Далее», чтобы начать установку [Рис. 2.2](#). Дождитесь завершения установки.

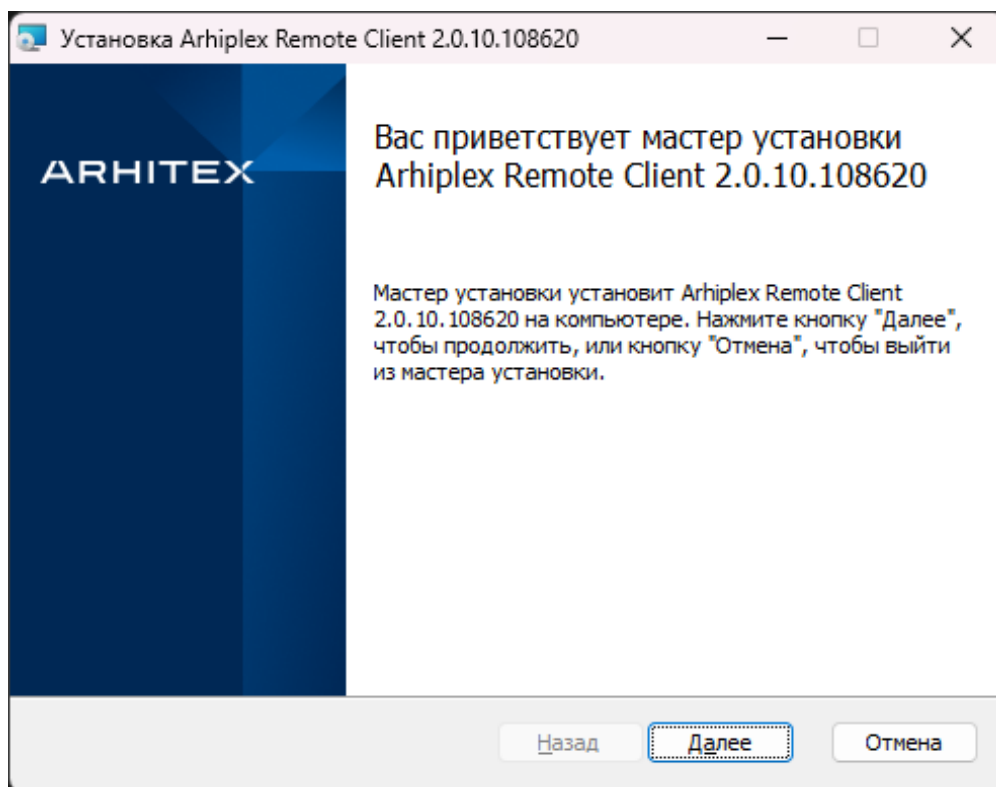


Рис. 2.1: Старт мастера установки

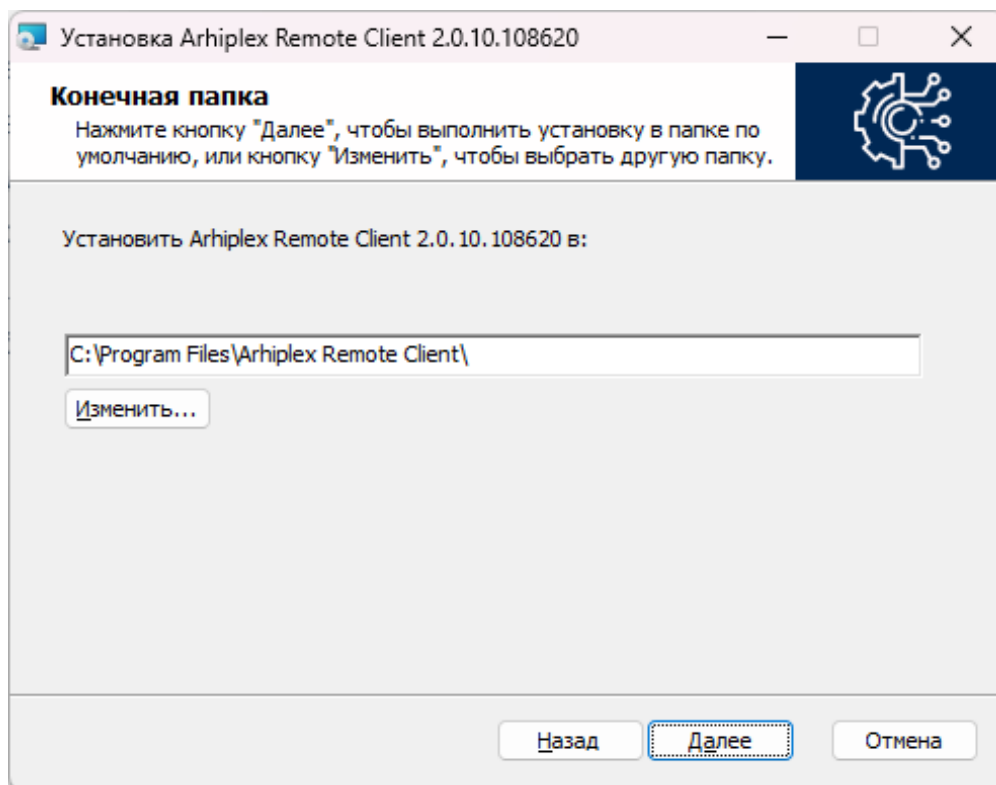


Рис. 2.2: Выбор места установки

5. Установка завершена. Нажмите «Готово» Рис. 2.3.

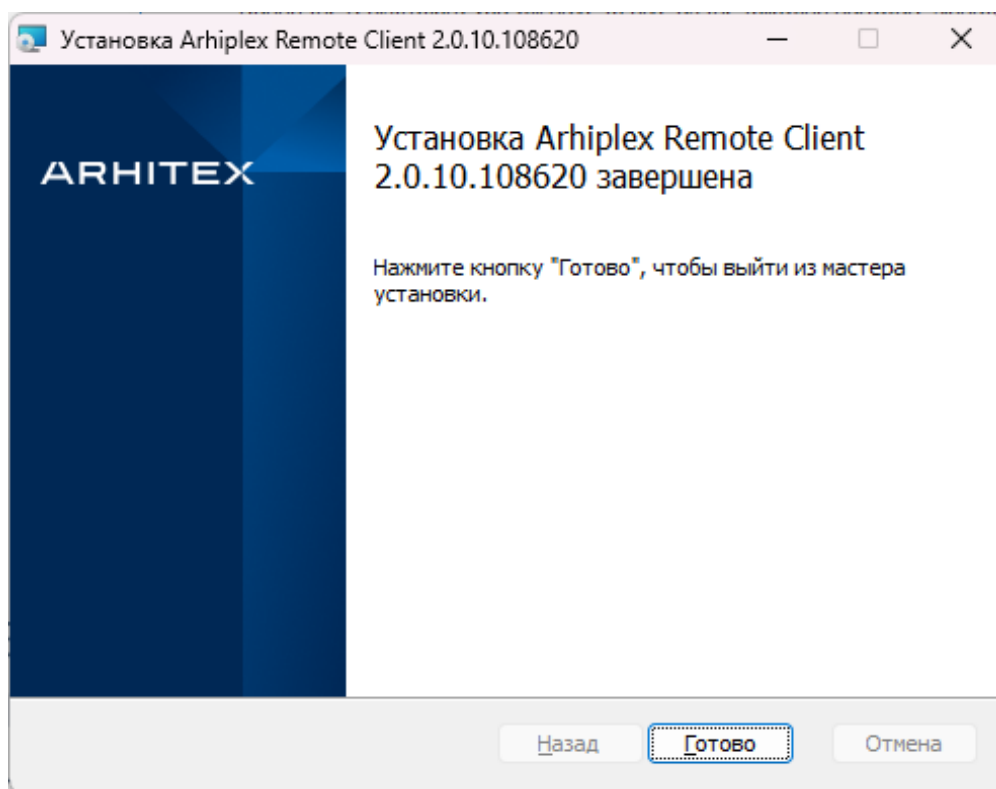


Рис. 2.3: Установка завершена

2.2 Linux

2.2.1 Системные требования

- Архитектура x64
- Операционная система: Ubuntu (20.04, 22.04), Debian (10, 11, 12), Astra Linux (1.7), RHEL (8), Redos (7.3)
- Для библиотеки Python: Python 3.8 - 3.11

2.2.2 Установка на Linux

1. Для установки необходимо распаковать архив в любую папку

```
tar -xzf ArhiplexRemoteClient-<version>-Linux.tar.gz
```

2. После распаковки появится папка ArhiplexRemoteClient-<version>-Linux. Ее можно перенести в любое удобное место, например в /opt:

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

3. Далее необходимо установить необходимые переменные окружения. Для этого необходимо добавить в файл .bashrc в вашей \${HOME} папке следующие строки:

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

4. После добавления нужно открыть новый терминал и проверить:

```
echo ${ARHIPLEX_HOME}
echo ${LD_LIBRARY_PATH}
```

2.3 Установка пакета arhiplexpy

Если необходим интерфейс к солверу ArhiCloud на Python (arhiplexpy), его нужно дополнительно установить.

1. Для того чтобы установить Python API на Windows:

Перейти в каталог программы: «C:\Program Files\Arhiplex Remote Client\lib\python», из командной строки Windows выполнить команду:

```
pip install .
```

где «C:\Program Files\Arhiplex Remote Client\» – стандартный путь, который может быть изменен в процессе установки.

2. Для того чтобы установить Python API на Linux:

Необходимо выполнить команду:

```
cd lib64/python
pip3 install .
```

2.4 Удаление

Чтобы удалить клиентское приложение ArhiCloud, перейдите в «Панель управления\программы и компоненты» и запустите Удаление программы.

3.1 Онлайн расчёт

Онлайн расчёт на портале доступен на вкладке «Расчёты» в левом меню Рис. 3.1.

ARHICLOUD

Расчёты

История расчетов [Загрузить файл](#) [Посмотреть инструкцию](#)

Данные для расчета

Загрузите файлы в формате *.lr или *.mps, не более 10 файлов

[Загрузите](#) или перетащите файлы сюда
Файлы в формате *.lr или *.mps, не более 10 файлов

Ключ лицензии
Выберите ключ

Лимит времени расчета задачи, сек.

Начать расчет

Рис. 3.1: Расчёт на портале ArhiCloud

Для запуска расчёта:

1. Загрузите или перетащите в выделенную область нужное количество файлов формата *.lr или *.mps. Количество файлов в рамках одного расчёта - не более 10.

Примечание: Для тестирования сервиса вы можете использовать свои файлы задач или воспользоваться библиотекой MIPLIB2017 с большим количеством эталонных задач для

тестирования солверов <https://miplib.zib.de/download.html>

2. Выберите из выпадающего списка «Ключ лицензии», к которому будет привязан ваш расчёт Рис. 3.2.

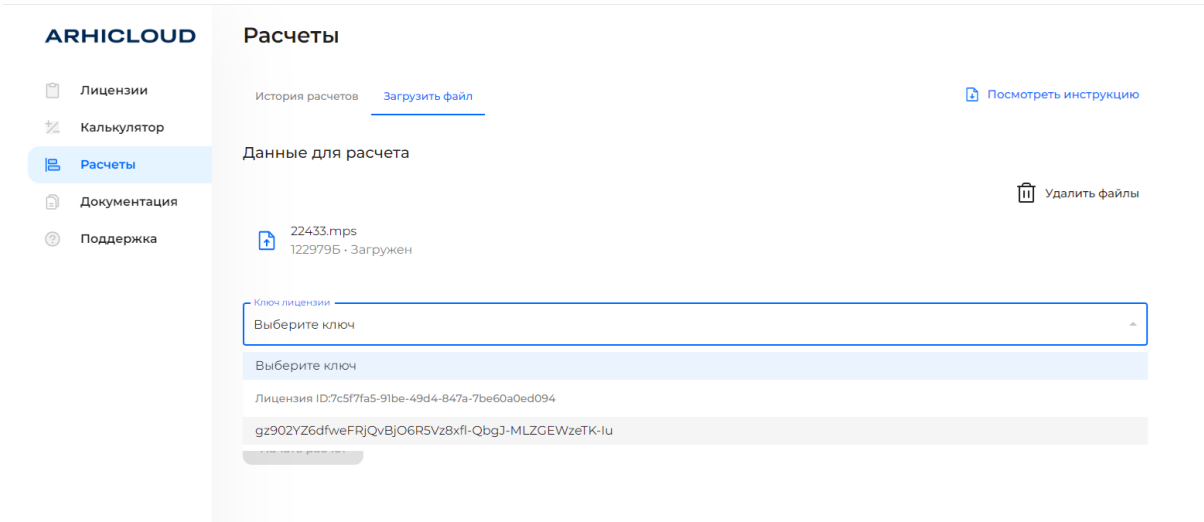


Рис. 3.2: Ввод данных для расчёт на портале ArhiCloud

3. Укажите лимит времени расчёта задачи в секундах (значение не должно превышать лимит на решение одной задачи в рамках используемой лицензии).
4. Нажмите кнопку «Начать расчёт».

После начала расчёта вы будете перенаправлены на страницу расчёта, на которой представлен список задач и статус их решения. Чтобы скачать решение задачи, воспользуйтесь кнопкой «Решение». Файлы с решениями содержат значения переменных, описанные в загруженных задачах. Скачать лог решения вы можете, нажав на кнопку «Лог» напротив завершённого расчёта Рис. 3.3.

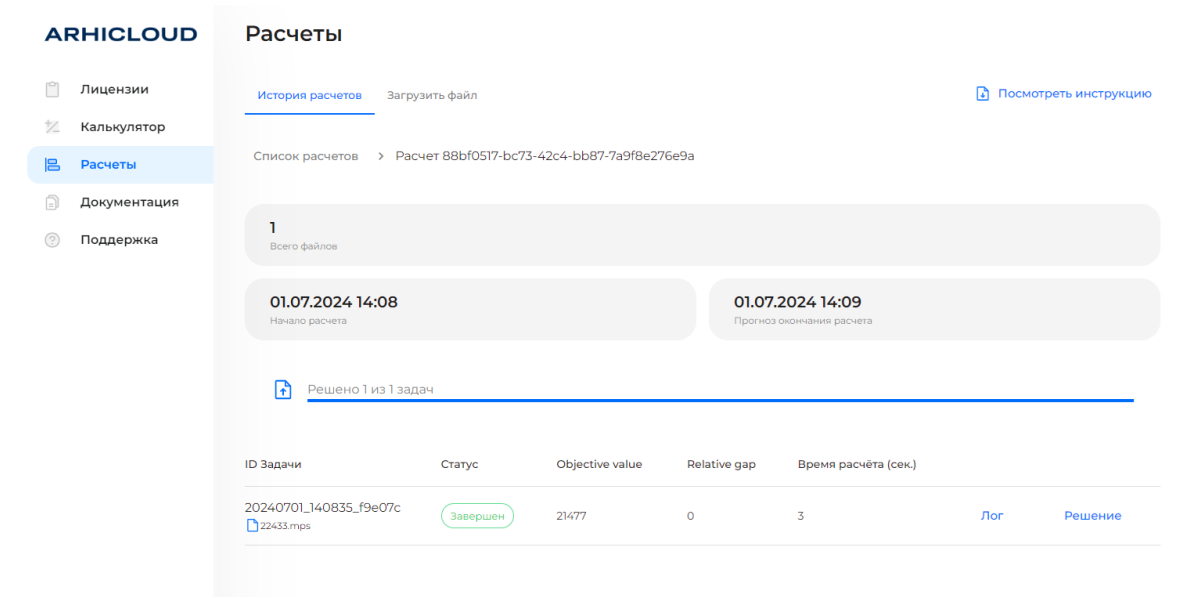


Рис. 3.3: История расчётов на портале

Для просмотра истории онлайн расчётов перейдите в раздел «Список расчётов» в верхней части экрана. Детализация каждого расчёта по задачам доступна по нажатию кнопкой мыши на строку с выбранным расчётом.

3.2 Python интерфейс

В этом разделе рассматривается использование солвера ArhiCloud, используя Python API.

Для расчёта через API нужно:

- Установить лицензионный ключ в соответствии с инструкцией для операционной системы *Windows* или *Linux*;
- Установить клиентское приложение ArhiCloud, которое содержит актуальную версию пакета *arhiplexru* (см. *Установка*);
- Установить пакет *arhiplexru* (см. *Установка пакета arhiplexru*);
- Подключить пакет *arhiplexru* (см. *Запуск расчёта*);
- Реализовать математическую модель на языке Python с использованием пакета *arhiplexru*. Пример реализации модели (см. *Запуск расчёта*);
- Запустить расчет.

3.2.1 Установка лицензионного ключа для Windows

Для запуска расчёта через API в ОС Windows необходимо установить лицензионный ключ на ПК, с которого осуществляется запуск. Установка осуществляется через запись ключа в переменную окружения `X_API_KEY` операционной системы. Доступна настройка постоянного и временного действия переменной.

Для настройки постоянного действия переменной:

1. Откройте вкладку «Дополнительно» в свойствах системы;
2. Выберите «Переменные среды...»;
3. В открывшемся окне выберите действие «Создать...»;
4. В поле «Имя переменной» введите значение: `X_API_KEY`;
5. В поле «Значение» введите значение вашего ключа лицензии;
6. Нажмите «ОК».

После сохранения этой переменной в списке переменных окружений при каждом запуске расчёта ключ будет передан на сервер расчёта.

Для настройки временного действия задайте значение ключа одним из способов:

- В командной строке с помощью команды `set`:

```
set = <значение вашего ключа лицензии>
```

- В PowerShell с помощью команды `$Env:X_API_KEY`:

```
$Env:X_API_KEY = <значение вашего ключа лицензии>>
```

3.2.2 Установка лицензионного ключа для Linux

В ОС Linux установка лицензионного ключа осуществляется через запись ключа в переменную окружения X_API_KEY операционной системы. Пример команды для добавления пользовательской переменной:

```
export X_API_KEY=<значение вашего ключа лицензии>
```

После установки ключа в переменные окружения операционной системы возможен запуск расчёта на удаленном сервере ArhiCloud.

3.2.3 Запуск расчёта

Рассмотрим следующую задачу смешанно-целочисленного программирования:

$$\begin{aligned}
 &\textbf{Maximize:} \\
 &-x_1 + 2x_2 + 3x_3 + x_4 \\
 &\textbf{Subject to:} \\
 &-x_1 + x_2 + x_3 + 10x_4 \leq 20 \\
 &x_1 - 3x_2 + x_3 \leq 30 \\
 &x_2 - 3.5x_4 = 0 \\
 &\textbf{Bounds:} \\
 &0 \leq x_1 \leq 40 \\
 &0 \leq x_2 \\
 &0 \leq x_3 \\
 &2 \leq x_4 \leq 4 \\
 &\textbf{Integers:} \\
 &x_4
 \end{aligned} \tag{3.1}$$

Ниже приведен исходный код для решения данной проблемы:

```

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

```

(continues on next page)

(продолжение с предыдущей страницы)

```

15
16 if __name__ == "__main__":
17     model = arhiplexpy.Model()
18
19     model.set_dbl_param("mip_rel_gap", 0.1)
20
21     x1 = model.add_variable(0, 40, 1, arhiplexpy.variable_type.continuous, 'x1')
22     x2 = model.add_variable(0, arhiplexpy.kArhiplexInf, 2, arhiplexpy.variable_
↪type.continuous, 'x2')
23     x3 = model.add_variable(0, arhiplexpy.kArhiplexInf, 3, arhiplexpy.variable_
↪type.continuous, 'x3')
24     x4 = model.add_variable(2, 3, 1, arhiplexpy.variable_type.integer, 'x4')
25
26     model.add_constraint(-x1 + x2 + x3 + 10 * x4 <= 20, 'c1')
27     model.add_constraint(x1 - 3 * x2 + x3 <= 30, 'c2')
28     model.add_constraint(x2 - 3.5 * x4 == 0, 'c3')
29
30     model.set_objective_sense(arhiplexpy.objective_sense.maximize)
31
32     result = model.solve_remote(name_mapping_file="test.map")
33
34     if result.get_solve_result() == arhiplexpy.solve_result.success and \
35         (result.get_solution_status() == arhiplexpy.solution_status.optimal_
↪or \
36         result.get_solution_status() == arhiplexpy.solution_status.
↪feasible):
37         print('Objective value:', result.get_objective_value())
38         print('x1:', result.get_variable_value(x1))
39         print('x2:', result.get_variable_value(x2))
40         print('x3:', result.get_variable_value(x3))
41         print('x4:', result.get_variable_value(x4))
42     else:
43         print(result.get_solution_status())

```

Для начала нужно импортировать модуль arhiplexpy:

```
import arhiplexpy
```

Далее необходимо создать объект модели, с которой пользователю требуется далее взаимодействовать.

Для модели имеется возможность устанавливать внешние параметры. В данном случае осуществляется установка вещественного параметра `mip_rel_gap` с помощью команды `set_dbl_param`.

Далее добавляем переменные. Для добавления используем метод модели `add_variable()`. Переменная всегда ассоциируется с конкретной моделью.

```

x2 = model.add_variable(0, arhiplexpy.kArhiplexInf, 2, arhiplexpy.variable_
↪type.continuous, 'x2')
x3 = model.add_variable(0, arhiplexpy.kArhiplexInf, 3, arhiplexpy.variable_
↪type.continuous, 'x3')
x4 = model.add_variable(2, 3, 1, arhiplexpy.variable_type.integer, 'x4')

```

Первый и второй аргументы – это нижняя и верхняя границы переменной соответственно. Третий аргумент – значение коэффициента при этой переменной в целевой функции. Четвёртый – тип переменной. Последний аргумент – это имя переменной. Для всех аргументов существуют значения по умолчанию (См. *Python API*).

Следующим шагом задаём ограничения. Так же как и переменные, ограничения всегда ассоциируются с конкретной моделью. Ограничения добавляются в модель с помощью её метода `add_constraint()`:

```

model.add_constraint(x1 - 3 * x2 + x3 <= 30, 'c2')
model.add_constraint(x2 - 3.5 * x4 == 0, 'c3')

```

Далее установим тип задачи (минимизация или максимизация целевой функции) с помощью метода модели `set_objective_sense()`:

Запустите расчёт на ArhiCloud. Параметр `name_mapping_file` является путем к файлу, содержащим исходные наименования переменных и ограничений перед их обфускацией и отправкой задачи на сервер.

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

```

(result.get_solution_status() == arhiplexpy.solution_status.optimal_
↪or \
    result.get_solution_status() == arhiplexpy.solution_status.
↪feasible):
    print('Objective value:', result.get_objective_value())

```

Обратите внимание, что получить значение переменной с помощью метода результата решения `get_variable_value()` можно используя как объект переменной, так и её имя. Для детального изучения интерфейса на языке Python см. *Python API*:

```

print('x2:', result.get_variable_value(x2))
print('x3:', result.get_variable_value(x3))
print('x4:', result.get_variable_value(x4))
else:

```

После отправки задачи отобразятся параметры запуска, с которыми работает солвер. Пример лога:

```
Права на использование (с) 2024 Arhitex

Параметры запуска:
presolve : on
problem_type : max
time_limit : 1000000.000000
abs_gap : 0.000001
gap : 0.0001
remote_timeout : 300
Отправка запроса на расчет . . . успешно
UID : 20240426_161046_575b9c
Начало загрузки модели . . . успешно
Ожидание начала расчета . . . .

Запуск Arhiplex 2.0.0
Права на использование (с) 2024 Arhitex

Оригинальная задача MILP temp.aps содержит:
  всего ограничений: 3
  всего переменных: 4
дробных переменных: 3
  целых переменных: 3
Результат решения задачи:

  Лучшая Граница   | Лучшее Решение   | Точность | Итераций |   Время
-----+-----+-----+-----+-----
          125.208333 |        122.500000 |      2.21 |          3 |         0 с
Objective value: 122.5
x1: 40.0
x2: 10.5
x3: 19.5
x4: 3.0
```

По мере решения задачи текущий статус может изменяться:

Таблица 3.1: Лог задачи

Статус решения задач	Описание статуса
Calculating	В данный момент времени задача решается
Done	Решение задачи завершено, можно получить результат
Error	Возникла ошибка во время расчёта
Queued	Задача находится в очереди на решение

По результатам решения задачи отображается информация:

- времени работы расчёта;
- значение целевой функции;
- значение показателя MIP gap

3.2.4 Управление расчётами и асинхронный запуск при работе с ArhiCloud (Python)

При решении задач на удаленных ресурсах имеются дополнительные возможности для управления расчётами.

Рассмотрим пример на Python:

```

1 import arhiplexpy
2
3
4 if __name__ == "__main__":
5     model = arhiplexpy.Model()
6
7     model.read("blp-ic98.mps")
8     model.set_dbl_param("mip_rel_gap", 0.1)
9     model.set_dbl_param("time_limit", 70)
10    model.set_log_file("arhi_cloud.log")
11
12    result = model.solve_remote("name_mapping.txt")
13    uid = model.get_calc_uid()
14
15    result.write_solution("blp-ic98.sol")
16
17    result_by_uid = arhiplexpy.SolveResult(uid, "name_mapping.txt")
18
19    print("Objective:", result_by_uid.get_objective_value())
20    print("Solve time:", result_by_uid.get_solve_time())

```

Интерфейс работы с удаленным сервером имеет несколько особенностей:

- Поддержка ограниченного набора параметров. Параметры, которые поддерживаются в разделе *Параметры*, содержат в описании ключевое слово ArhiCloud.
- Запуск расчёта осуществляется с помощью метода `solve_remote()` или `solve_remote_async()`.

`solve_remote_async` отличается от синхронного `solve_remote()` тем, что не возвращает результат расчёта (объект `SolveResult`) и не ожидает окончания расчёта. Оба метода требуют указать путь к файлу, в который будет записан список всех имен модели до анонимизации и после (все имена переменных, ограничений, а также целевой функции). Результат расчёта можно получить позднее по универсальному идентификатору расчёта и по имени файла анонимизации модели. Для получения идентификатора необходимо у модели вызвать метод `get_calc_uid()`:

Для получения результата с помощью этого идентификатора необходимо создать объект `SolveResult` и в конструктор передать идентификатор и путь к файлу анонимизации модели, созданному ранее:

```
print("Objective:", result_by_uid.get_objective_value())
```

(continues on next page)

(продолжение с предыдущей страницы)

```
print("Solve time:", result_by_uid.get_solve_time())
```

Для работы с ArhiCloud необходимо установить переменную окружения `X_API_KEY` - уникальный ключ клиента (см. инструкцию для *Windows* или *Linux*).

После отправки задачи на удаленный сервер ArhiCloud отобразятся параметры запуска, с которыми работает солвер. Пример лога:

Запуск ArhiPlex 2.0.0

Права на использование (с) 2024 Arhitex

Оригинальная задача MILP blp-ic98.mps содержит:

всего ограничений: 717

всего переменных: 13640

можнодробных переменных: 90

бинарных переменных: 13550

Лучшая Граница	Лучшее Решение	Точность	Итераций	Время
4376.298372	6553.151299	33.22 %	385	2 с
4430.396860	4611.371957	3.92 %	60334	6 с
4432.143584	4611.371957	3.89 %	60334	13 с
4466.751708	4501.323861	0.77 %	234629	16 с
4473.256688	4501.323861	0.62 %	541526	22 с
4476.533987	4491.447584	0.33 %	839431	27 с
4476.533987	4491.447584	0.33 %	839431	31 с
4482.301705	4491.447584	0.20 %	1265834	41 с
4482.301705	4491.447584	0.20 %	1265834	46 с
4486.072772	4491.447584	0.12 %	1268277	53 с
4486.072772	4491.447584	0.12 %	1268277	63 с

Отчет

Статус решения : Оптимально

Первичная граница : 4491.447584

Двойственная граница : 4488.145776

Отклонение : 0.07 %

Время расчета : 76.1 с.

Ноды(листья) : 53529

LP итераций : 1313251

3.3 C++ интерфейс

Предупреждение: На текущий момент C++ интерфейс поддерживается только для Windows и Ubuntu.

В текущем разделе будет рассмотрен пример использования солвера **Arhiplex** в пользовательском коде C++.

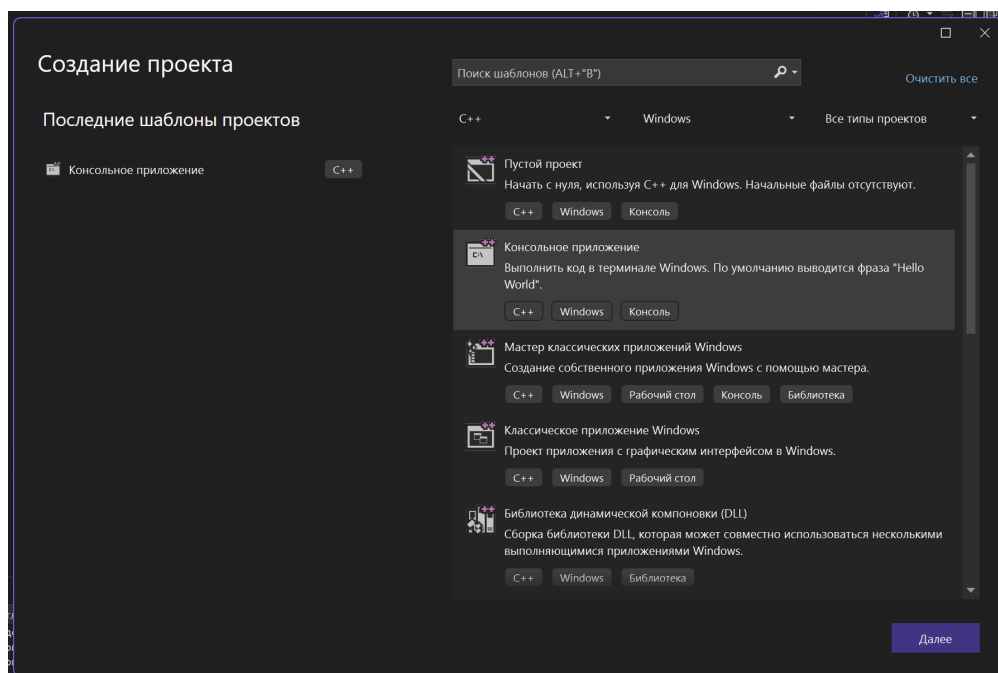
Настройка проекта в Visual Studio 2022

Для работы с ArhiCloud в приложениях на C++ выполните перечень предварительных шагов - подключите файлы заголовков и дополнительную библиотеку *libarhiplex_cpp.lib*.

В качестве примера рассмотрим процесс запуска решения с использованием IDE MS Visual Studio 2022.

- Подключение файлов заголовков

Создайте проект консольного приложения на C++ в Microsoft Visual Studio 2022.

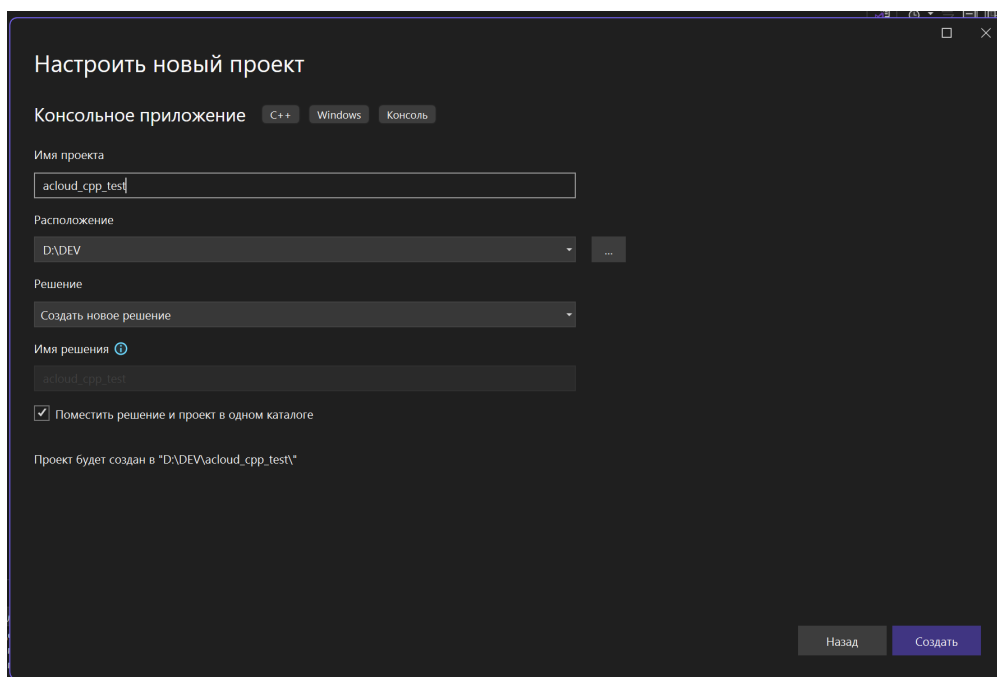


Для использования возможностей построения моделей и запуска решения на C++ скопируйте папку *include* с файлами заголовков, расположенную в директории, в которую было установлено приложение ArhiPlex Remote Client (обычно *C:/Program Files/ArhiPlex Remote Client/*) в каталог проекта. Для подключения в исполняемом файле укажите путь к основному файлу *arhiplex_client.h*:

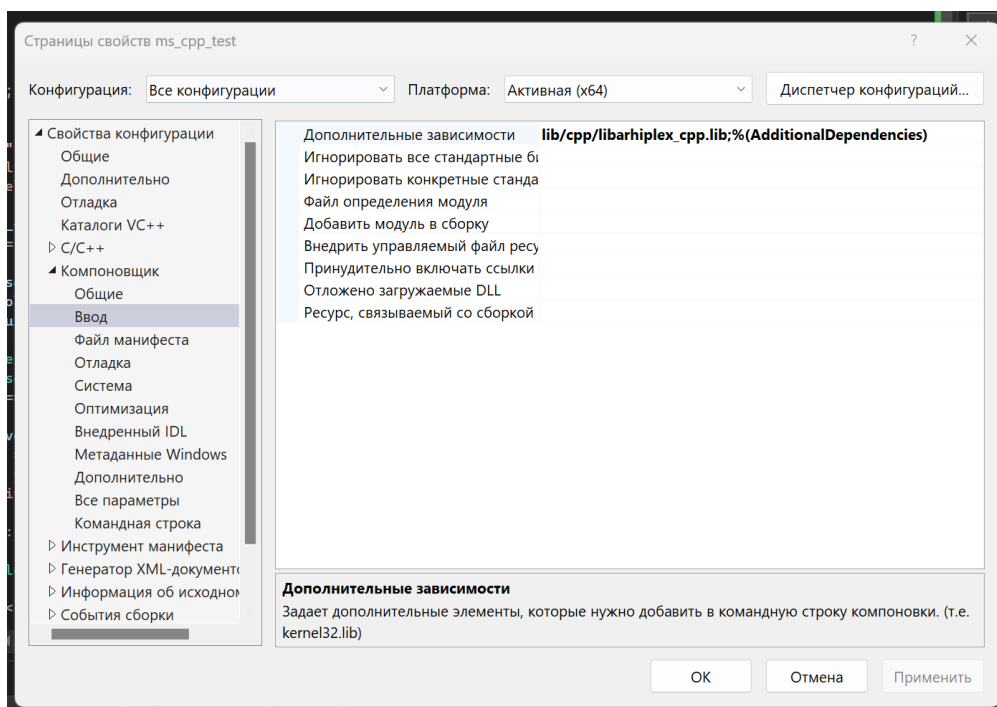
```
#include "include/arhiplex_client.h"
```

Подключение библиотеки

Для построения проекта обязательно подключение библиотеки *libarhiplex_cpp.lib*. Для этого скопируем папку *lib* из директории, в которую установлено приложение ArhiPlex Remote Client в каталог проекта.



Далее в Свойствах проекта -> «Компоновщик» -> «Ввод» в поле «Дополнительные зависимости» укажите путь к файлу с библиотекой. Предварительно в выпадающем списке «Конфигурация» выберите «Все конфигурации».



Нажмите «ОК».

После подключения файлов заголовков и библиотеки возможно построение моделей и запуск решения

Практическое использование на примере

Рассмотрим решение следующей проблемы:

$$\begin{aligned}
 &\textbf{Maximize:} \\
 &x_1 + 2x_2 - 0.1x_3 - 3x_4 \\
 &\textbf{Subject to:} \\
 &x_1 + x_2 \leq 5 \\
 &2x_1 - x_2 \geq 0 \\
 &-x_1 + 3x_2 \geq 0 \\
 &x_3 + x_4 \geq 0.5 \\
 &\textbf{Bounds:} \\
 &-\infty < x_1 < +\infty \\
 &-\infty < x_2 < +\infty \\
 &1.1 \leq x_3 \leq 10.0 \\
 &-\infty < x_4 < +\infty
 \end{aligned} \tag{3.2}$$

Ниже приведен исходный код для решения данной проблемы. Этот файл (*example1.cpp*) можно найти в папке *examples* после установки клиентского приложения ArhiCloud (см. *Установка*).

```

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.SetDbiParam("time_limit", 10.0); //10 сек
30         model.SetDbiParam("mip_rel_gap", 0.0005); //0.05%

```

(continues on next page)

(продолжение с предыдущей страницы)

```

31
32 // Добавление переменных
33 auto x1 = model.AddVariable(-arhiplex::inf, arhiplex::inf, 0.0,
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;

```

(continues on next page)

(продолжение с предыдущей страницы)

```

80 }
81 catch (...)
82 {
83     std::cout << "Unknown exception occurs!";
84 }
85 return 0;
86 }

```

Для начала нужно подключить заголовочный файл `arhiplex_client.h`. Далее необходимо создать модель:

```
using namespace arhiplex;
```

Далее выставляем параметры:

```

arhiplex::Model model;
//Как пример - для данной простой модели не актуально
model.SetDbiParam("time_limit", 10.0); //10 сек

```

Полный список параметров можно посмотреть в разделе *Параметры*.

Следующий шаг – добавление переменных. Переменные добавляются с помощью метода модели `AddVariable()`. Переменная всегда ассоциируется с конкретной моделью.

```

// 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");

x3.SetLowerBound(1.1);
x3.SetUpperBound(10.0);
// или сразу
// auto x3 = model.AddVariable(1.1, 10.0, 0.0,
//     variable_type::continuous, "x3");

model.SetObjective(x1 + 2 * x2 - 0.1 * x3 - 3 * x4,

```

Первый и второй аргументы – это нижняя и верхняя границы переменной соответственно. Третий аргумент – значение коэффициента при этой переменной в целевой функции. Четвёртый – тип переменной. Последний аргумент – это имя переменной. Обратите внимание, что значения нижней и верхней границ можно поменять позже как в этом примере для переменной `x3`.

Метод `SetObjective()` у модели задаёт целевую функцию. Первый аргумент – это линейное выражение, а второй – тип проблемы (максимизация или минимизация).

Далее добавим ограничения в модель, используя метод модели `AddConstraint()`:

```
model.AddConstraint(x1 + x2 <= 5, "c0");
model.AddConstraint(2 * x1 - x2 >= 0, "c1");
model.AddConstraint(-x1 + 3 * x2 >= 0, "c2");
```

Первый аргумент – это само ограничение, а второй – имя ограничения. Теперь, когда модель создана можно запустить процесс оптимизации:

Метод `Solve` вернёт результат оптимизации, который содержит все необходимые данные после расчёта. Например, можно получить время расчёта в сек:

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

```
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";
}
```

Обратите внимание, что получить значение переменной с помощью метода результата решения `GetVariableValue()` можно используя как объект переменной, так и её имя. Для детального изучения интерфейса на языке C++ см. [C++ API](#).

Для компиляции данной программы, например, на Ubuntu используя GNU компилятор, необходимо выполнить:

```
cd /opt/ARHIPLEX-<version>-Linux/examples
g++ example1.cpp -I ${ARHIPLEX_HOME}/include/arhiplex/ -L ${ARHIPLEX_HOME}/
lib/ -larhiplex_cpp
```

```
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 ($\leq$, $=$, $\geq$) constant)

Параметры

- **expr** – **[in]** выражение, являющееся левой частью ограничения
- **sense** – **[in]** переменная, задающая знак ($\leq$, $=$, $\geq$)
- **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: $\text{upper_bound} - \text{lower_bound}$

Результат

$\text{upper_bound} - \text{lower_bound}$

inline void SetRange(double range)
Задаёт интервал ограничения: $\text{lower_bound} \leq \text{expression} \leq \text{lower_bound} + \text{range}$

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 GetDblParam(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 SetDblParam(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 const char *GetRemoteSolveLog(const char *szCalcUID) const
```

Получает лог удалённого расчёта

Параметры

szCalcUID – **[in]** идентификатор удалённого расчёта

Результат

лог удалённого расчёта

```
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 **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 удаленную. Переменная не будет участвовать в расчетах

`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 достижимость модели

enumerator **fail**

Процесс решения неудачен - не найдено никаких решений и не обнаружена достижимость модели

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

```
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* <= *expression* <= *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 GetDblParam (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 SetDblParam (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 удаленную. 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 недо-
стижимость модели

fail

Процесс решения неудачен - не найдено никаких решений и не обнаружена недо-
стижимость модели

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

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
```

```
class arhiplexpy.Model
```

Модель для удаленного расчета.

```
Model.solve_remote()
```

```
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

```
Model.solve_remote_async()
```

```
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`.

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

`Model.get_remote_solve_log()`

`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

`class arhiplexpy.SolveResult`

Результат удалённого расчёта модели.

`class arhiplexpy.MatrixModel`

Создает экземпляр матричной модели

`MatrixModel.solve_remote()`

`solve_remote(self: arhiplexpy.arhiplexpy.MatrixModel) -> arhiplex::MatrixSolveResult`

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

return

объект с результатом оптимизации

`class arhiplexpy.MatrixSolveResult`

Результат удалённого расчёта матричной модели.

Список параметров формируется во время сборки документации.

Раздел содержит описание и настройку предусмотренных типов лицензий облачного сервера **ArhiCloud**.

Чтобы выпустить лицензию, необходимо перейти на портал **ArhiCloud** (<https://arhicloud.arhitex.com/>) и зарегистрироваться, либо *связаться с нами* удобным для вас способом.

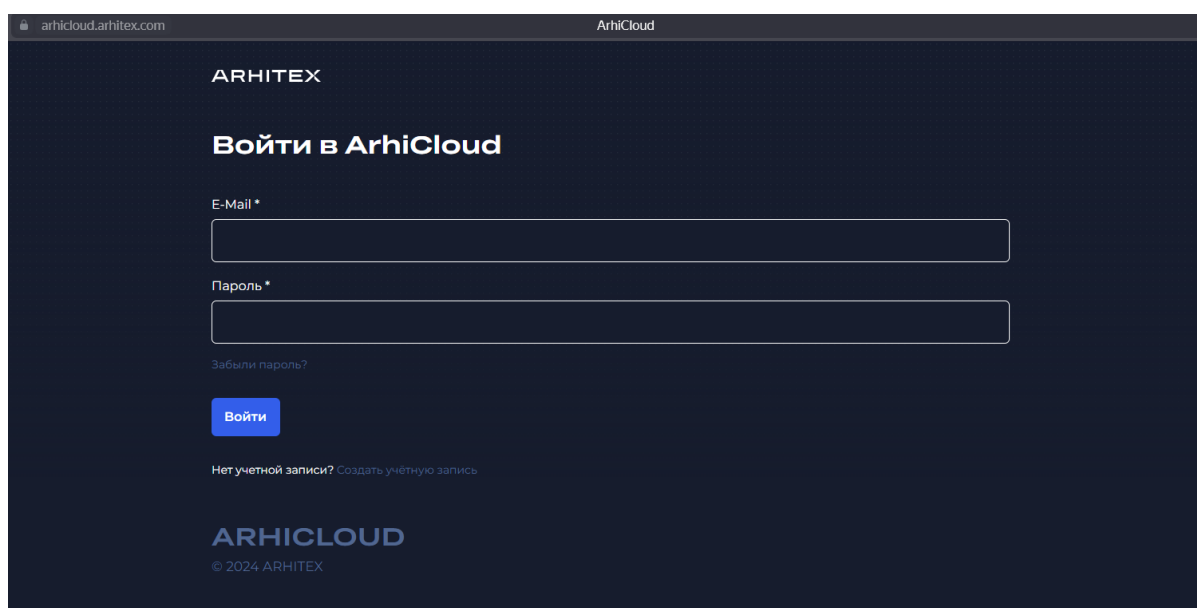


Рис. 8.1: Портал ArhiCloud

8.1 Активация пробной лицензии

Пробная лицензия позволяет пользователю оценить возможности облачного сервиса расчёта задач математической оптимизации. В рамках пробной лицензии пользователю доступно решение задач с лимитом по времени не более 30 минут на решение одной задачи, а общее доступное время расчёта – не более 10 часов. Срок действия пробной лицензии составляет 7 дней.

Активация пробной лицензии бесплатна и доступна только один раз. Для активации пробной лицензии:

1. Перейдите на вкладку «Лицензии», либо на вкладку «Калькулятор» в левом меню портала, Рис. 8.2 и Рис. 8.3 соответственно (далее действия будут производиться из вкладки «Калькулятор»);

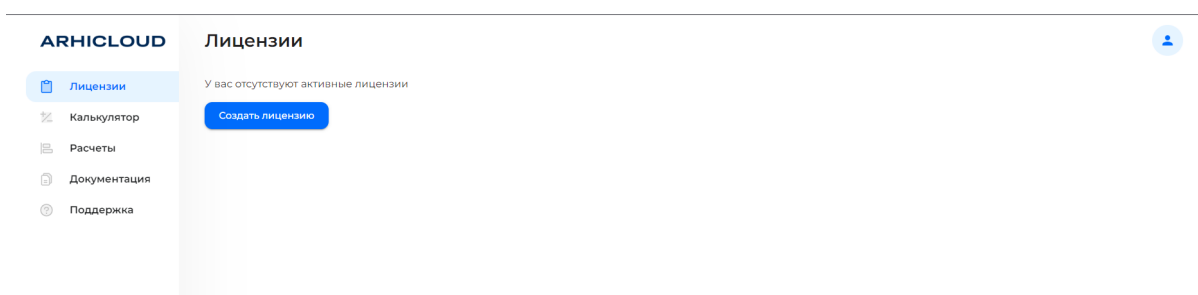


Рис. 8.2: Вкладка «Лицензии»

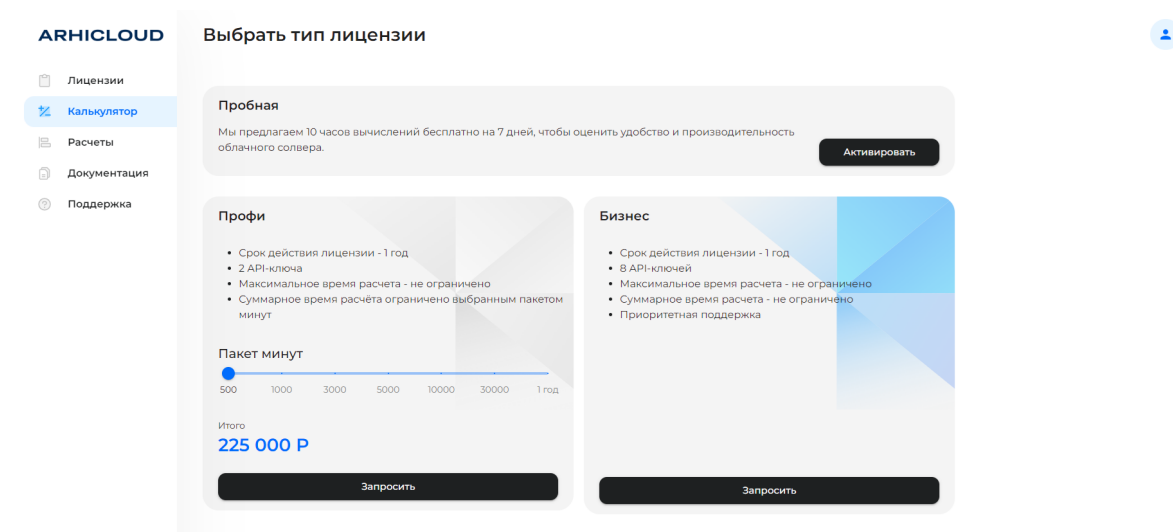


Рис. 8.3: Вкладка «Калькулятор»

2. Нажмите кнопку «Активировать» в блоке описания пробной лицензии;
3. В открывшемся модальном окне нажмите кнопку «Активировать» Рис. 8.4.

Во вкладке «Лицензии» в разделе «Активные» будет отображена информация об активированной пробной лицензии Рис. 8.5.

Пробная лицензия позволяет пользователю использовать «Онлайн расчёт» на портале, а также расчёты через API. Описание онлайн расчёта и расчёта через API представлено в данной инструкции: Раздел 3.1 и Раздел 3.2 .

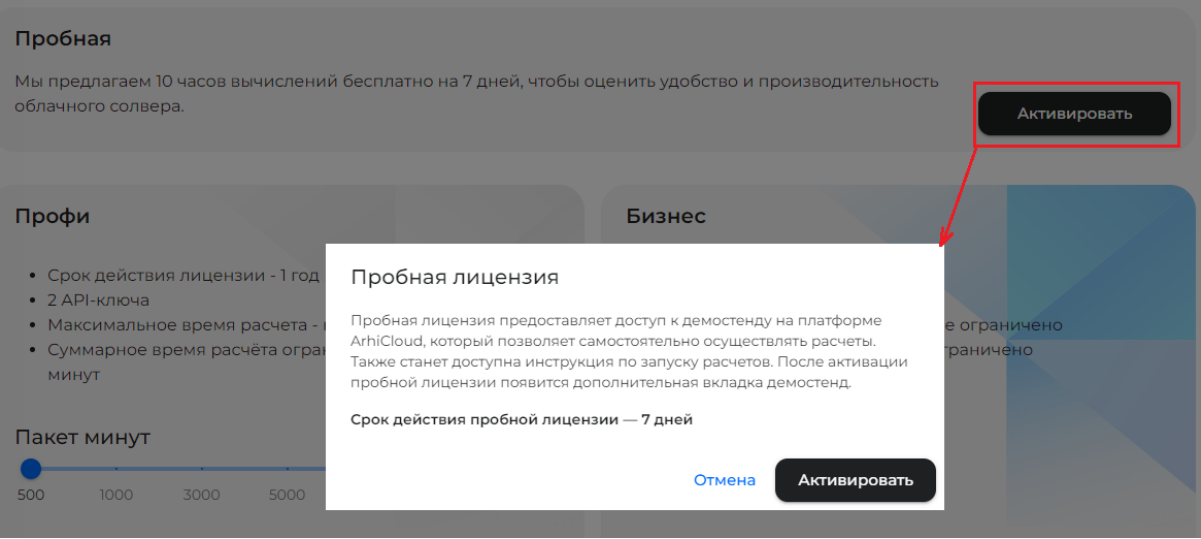


Рис. 8.4: Модальное окно для активации пробной лицензии

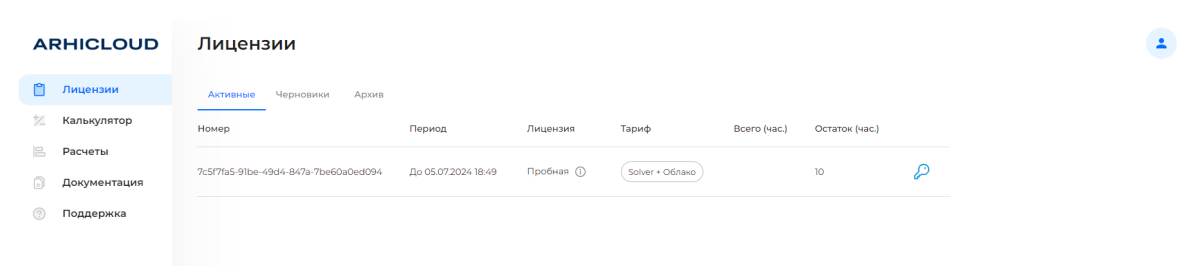


Рис. 8.5: Активные лицензии

8.2 Активация лицензии «Профи» и «Бизнес»

Лицензии типа «Профи» и «Бизнес» позволяют полноценно использовать функционал облачного сервиса расчёта задач математической оптимизации.

8.2.1 Лицензия «Профи»

Лицензия «Профи» может быть выпущена с разным количеством пакета минут в зависимости от нужд пользователя. В рамках лицензии «Профи» пользователю доступно:

- Решение задач с неограниченным лимитом по времени на решение одной задачи;
- Общее доступное время расчёта ограничено выбранным пакетом минут;
- Два API-ключа;
- Срок действия лицензии «Профи» составляет 1 год или по исчерпанию пакета минут.

Для активации лицензии «Профи»:

1. Перейдите на вкладку *Калькулятор* в левом меню на портале;
2. При помощи ползунка в области «Профи» настройте нужное количество минут расчёта, цена за лицензию будет пересчитана автоматически.
3. Нажмите кнопку «Запросить».
4. Дождитесь обратной связи от менеджера продукта для индивидуальной настройки лицензии.

8.2.2 Лицензия «Бизнес»

Лицензия «Бизнес» выпускается только при помощи индивидуальной настройки под конкретные нужды пользователя. В рамках лицензии «Бизнес» пользователю доступно:

- Решение задач с неограниченным лимитом по времени на решение одной задачи;
- Общее доступное время расчёта неограниченно;
- Восемь ключей API;
- Приоритетная поддержка;
- Срок действия лицензии «Бизнес» составляет 1 год.

Для активации лицензии «Бизнес»:

1. Перейдите на вкладку *Калькулятор* в левом меню на портале;
2. Нажмите кнопку «Запросить» в области «Бизнес».
3. Дождитесь обратной связи от менеджера продукта для индивидуальной настройки лицензии.

8.3 Выпуск лицензионного ключа

Выпуск лицензионного ключа доступен после активации лицензии любого типа. Лицензионный ключ нужен для: расчёта через API или онлайн расчёта.

Для выпуска лицензионного ключа:

1. Перейдите в раздел «*Лицензии*» в левом меню;
2. Нажмите на знак «*ключа*» в строке с лицензией, для которой требуется сгенерировать ключ [Рис. 8.6](#);

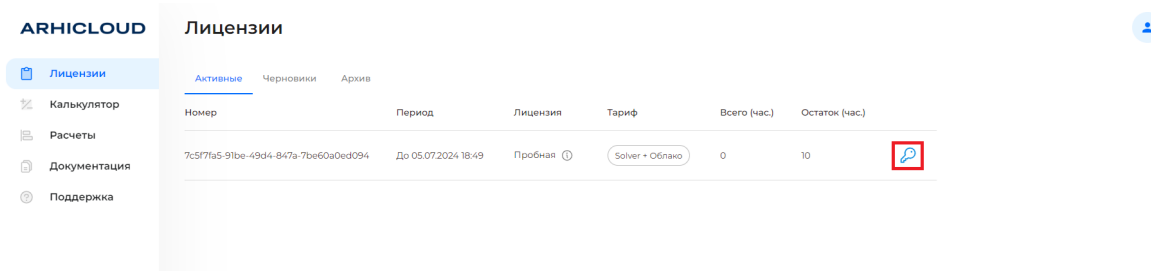


Рис. 8.6: Выпуск лицензионного ключа на портале ArhiCloud

3. В открывшемся окне нажмите кнопку «Сгенерировать ключ» [Рис. 8.7](#).

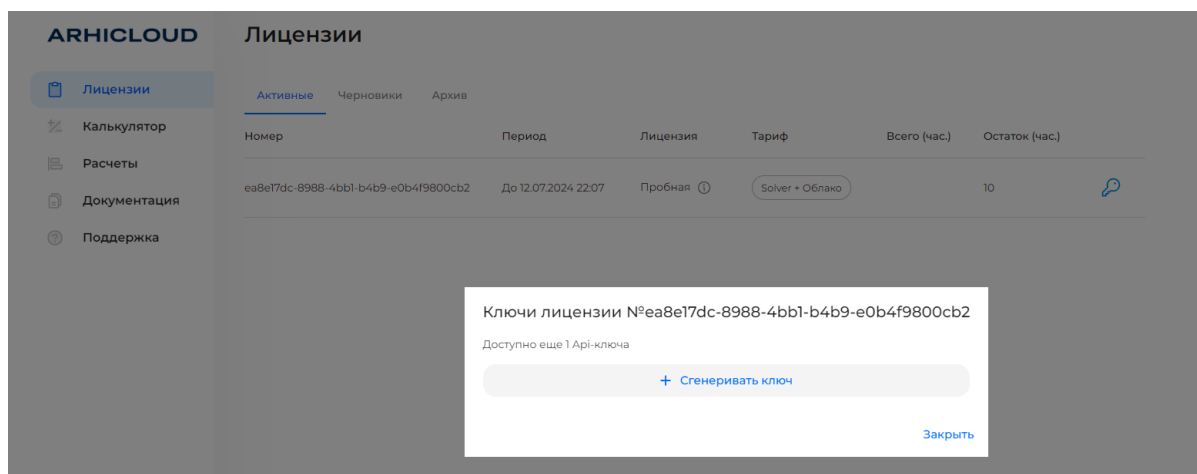


Рис. 8.7: Генерация API-ключа на портале ArhiCloud

Если кнопка генерации ключа неактивна, а на ее месте отображается сообщение о превышении лимита доступных ключей, для генерации нового ключа необходимо деактивировать один из действующих нажатием кнопки «Деактивировать» [Рис. 8.8](#).

4. Для использования лицензионного ключа необходимо будет скопировать его значение из поля «Ключ» [Рис. 8.9](#)

После получения лицензии и создания лицензионного ключа следующим шагом будет *установка* программного обеспечения.

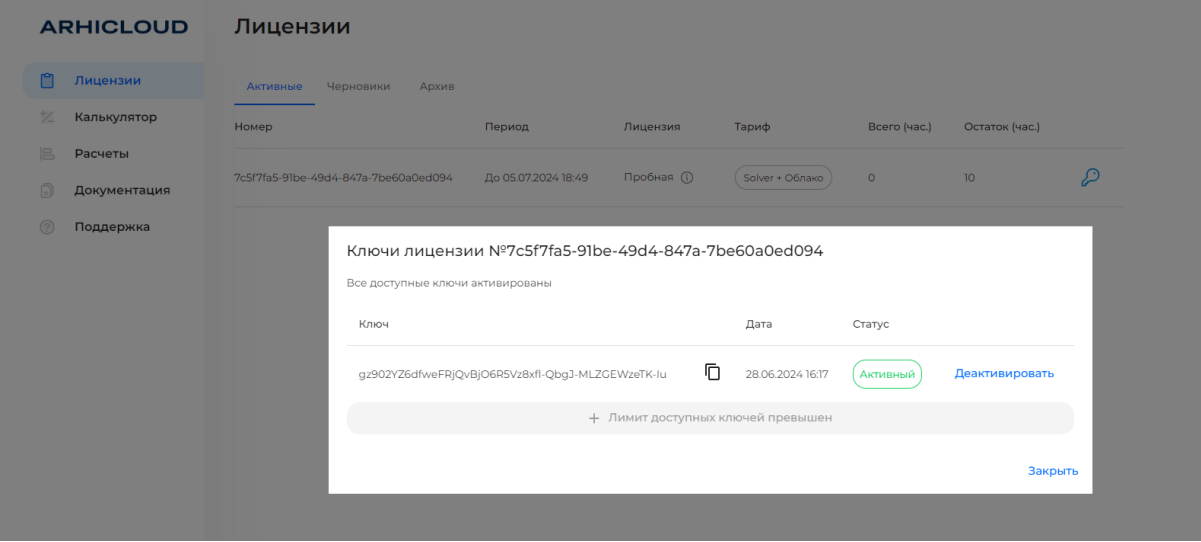


Рис. 8.8: Деактивация ключа

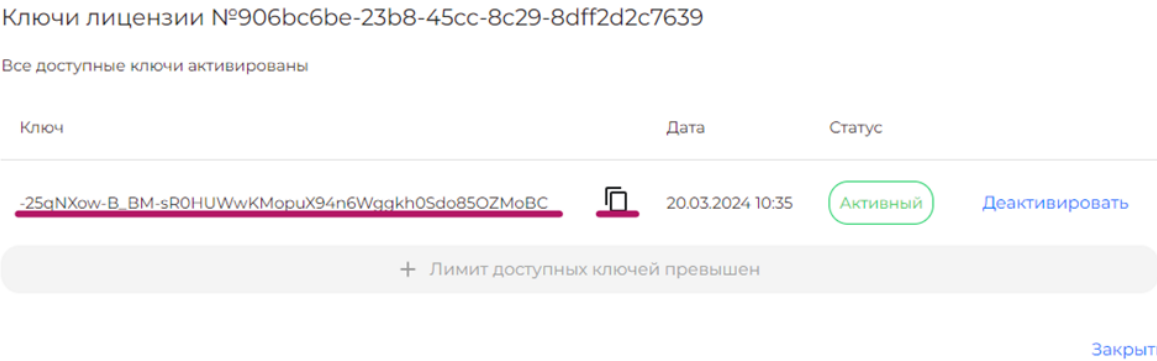


Рис. 8.9: Значение ключа

Техническая поддержка

Облачный сервис ArhiCloud предоставляет пользователям возможность обратиться за технической поддержкой через специальную вкладку «Поддержка» Рис. 9.1. Здесь пользователи могут задать вопросы и получить помощь в решении проблем.

ARHICLOUD Поддержка

Лицензии
Калькулятор
Расчеты
Документация
Поддержка

с 9:00 до 18:00
по рабочим дням
Консультирование

24 часа
в сутки
Для приоритетной поддержки

Служба поддержки консультирует пользователей по предоставлению услуг.

Перед тем, как связаться с оператором, пожалуйста, ознакомьтесь со списком часто задаваемых вопросов. Возможно, там уже есть ответ на Ваш вопрос.

Не нашли ответ на свой вопрос?
Опишите проблему и отправьте нам

Описание вопроса или проблемы

Отправить

Рис. 9.1: Старт мастера установки

Чтобы связаться с оператором поддержки:

1. Перейдите на вкладку «Поддержка» в левом меню портала;
2. Опишите вопрос или проблему в текстовом поле;
3. Дождитесь ответа оператора на почту, которую вы указывали при регистрации.

Техническая поддержка оказывается удаленно.

В разделе *Контактная информация* пользователь может найти дополнительные способы связи для обращения по вопросам и предложениям.

Контактная информация

Обращения, вопросы и предложения по работе и доработке продукта необходимо направлять по доступным каналам связи:

Таблица 10.1: Контактная информация

Тип	Информация	Описание	Время работы
Веб-сайт	https://arhicloud.arhitex.com/support	Официальный портал Поддержка	10:00 – 18:00 в рабочие дни
Электронная почта	arhicloud@arhitex.com	Служба поддержки Платформы	10:00 – 18:00 в рабочие дни
Электронная почта	info@arhitexlab.ru	Отдел информации	09:00 – 18:00 в рабочие дни

a

arhiplexpy, [62](#)