7. Калькулятор графиков

Калькулятор графиков позволяет построить произвольную комбинацию имеющихся параметров и рядов данных модели с использованием математических функций, численного дифференцирования/интегрирования, условных операторов, циклов и прочих средств языка программирования Python.

Использование калькулятора графиков описано в учебных курсах:

- **COMMON1.4. Калькулятор графиков Python**;
- **АНМ1.12. Использование экономической модели**;

Калькулятор графиков доступен также в интерфейсе модуля Адаптации и Оптимизации, где он позволяет использовать рассчитанный график как целевую функцию при адаптации, а также в интерфейсе Дизайнера Моделей. Рассчитанные графики могут также быть показаны на секторных диаграммах (см. 5.5.4.5).

7.1. Интерфейс Калькулятора графиков

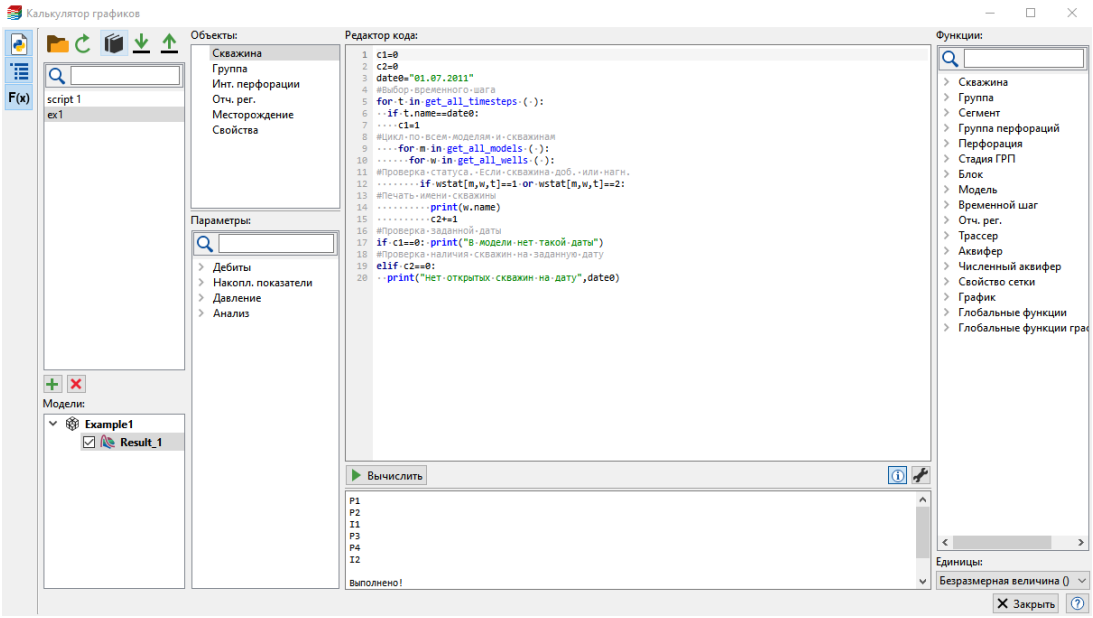

Рис. 122. *Калькулятор графиков*

Редактор кода в окне Калькулятора позволяет писать произвольный код с использованием синтаксиса Python. Код выполняется по нажатию кнопки **Вычислить**

или комбинации клавиш **Ctrl+Enter**. Если код зависает или выполняется слишком долго, его можно прервать кнопкой **Остановить**. Возможен импорт стандартных библиотек (import <имя>); см. также Импорт библиотек и настройка внешнего Python. Консольный выходной поток Python направлен в текстовое поле ниже и может использоваться для отладки.

Можно создать произвольное количество пользовательских скриптов. Скрипты сохраняются в отдельных файлах ***.py** в каталоге **USER/<имя_модели>/** при закрытии симулятора, но **не** входят в состав экспортированных шаблонов графиков. Сохранённые скрипты могут быть скопированы в другую модель.

Интерфейс Калькулятора графиков состоит из ряда панелей.

- **Панель скриптов** (включается/выключается кнопкой  слева) содержит список скриптов текущего проекта и следующие кнопки для управления им:
  -  **Задать директорию пользовательских скриптов** позволяет назначить директорию для загрузки скриптов из неё. Если в директории уже есть скрипты, при её выборе они будут автоматически прочитаны и загружены в проект. При смене директории имеется опция **Сохранить скрипты в предыдущую папку**, позволяющая сохранить все скрипты в директорию, которая была назначена ранее.
  -  **Перечитать скрипты** позволяет заново загрузить скрипты из пользовательской директории. Это необходимо в ситуации, когда скрипт скопирован в директорию уже после открытия модели.

    

    Обратите внимание, что при нажатии данной кнопки будут заново загружены **все** скрипты из директории. Если они были загружены ранее и потом изменены в графическом интерфейсе тНавигатор, то эти изменения будут потеряны.

  -  **Импорт скриптов из библиотеки** открывает библиотеку с примерами скриптов и позволяет загрузить в проект все или часть из них. Примеры включают:
    - расчёт среднего забойного давления по группам скважин;
    - расчёт добычи по стволам в многоствольных скважинах;
    - экспорт данных по добыче в текстовый файл;
    - и другие.
  -  **Импорт скрипта из файла** позволяет загрузить в проект скрипт из произвольного внешнего файла.
  -  **Добавить** создаёт новый (пустой) скрипт.
  -  **Удалить** удаляет текущий скрипт.

- **Панель объектов** (включается/выключается кнопкой  слева) содержит список типов объектов и список доступных параметров (графиков) для каждого типа. При двойном нажатии на параметр в код добавляется соответствующая мнемоника (формат ключевого слова SUMMARY, см. 12.20.1).

- **Панель кода** содержит окно редактирования кода и окно выдачи результата.

- **Панель функций** (включается/выключается кнопкой  слева) содержит список доступных функций API (см. 7.2. Функции и структуры данных.)

В результате выполнения скрипта обычно вычисляется новый график или несколько графиков. Чтобы вычисленные графики появились в интерфейсе пользователя, они должны быть экспортированы вызовом функции export(). Скрипт может содержать произвольное число вызовов этой функции. После выполнения скрипта с вызовом функции экспорта соответствующий график появляется в списке **Графики пользователя** шаблона (см. рис. 123) и может быть выбран для отображения отдельно или одновременно с другими. Его имя и размерность задаются при вызове функции экспорта. То, для какого объекта он будет отображаться (месторождение, скважина, группа, регион, или интервал перфорации) определяется его типом, который в свою очередь определяется декларацией переменной (см. описание функции graph()) или типом того графика или графиков, из которых он выведен вычислениями. Разночтения в этих типах могут привести к ошибке в скрипте.

Выбор единиц делается с помощью специальных мнемоник, которые в свою очередь, выбираются из выпадающего списка. Список содержит несколько наиболее типичных размерностей и пункт **Задать**, по нажатию на который разворачивается исчерпывающий список всех доступных единиц с возможностью поиска. Кроме того, можно использовать произвольные собственные единицы, не входящие в текущую систему единиц. Если скрипт не экспортирует ни одного графика, имеется возможность использовать опцию **Экспорт графика** в контекстном меню, вызываемом по правой кнопке мыши. Выбор данной опции добавляет в конец кода вызов функции export(), применённой к переменной из последнего по очереди оператора присваивания. Единицы измерения в этом случае не указываются, так что применяется значение по умолчанию: Безразмерная величина. Калькулятор может использоваться и без экспорта графиков, только для кратких вычислений, результат которых показывается тут же в окне консольного вывода.

Обратите внимание, что пользовательские графики из других скриптов, включая созданные в этом же шаблоне, **не становятся** автоматически доступны из скрипта по имени. Их необходимо подгрузить командой get_global_graph(). Можно также создать несколько графиков из одного скрипта.

Для скриптов можно задать две опции автозапуска (по умолчанию включена только первая): автозапуск после открытия модели и автозапуск после расчёта модели.

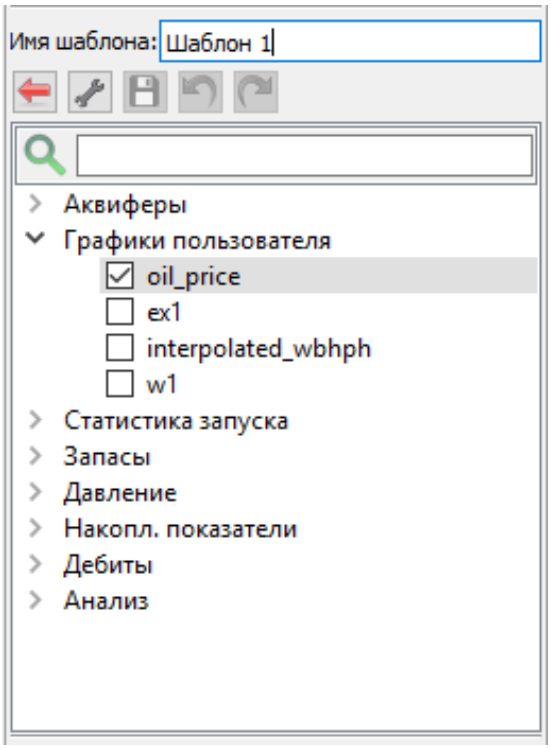

Рис. 123. *Пользовательские графики*

Опции задаются в глобальных настройках тНавигатор , см. Настройки автозапуска в Руководстве по Дизайнеру Моделей. Если обе выключены, скрипт запускается только при нажатии кнопки **Вычислить**.

7.2. Функции и структуры данных

Доступ Python к данным модели в значительной мере строится на графиках, в особенности мнемониках **SUMMARY**. График как объект программы представляет собой сложную структуру данных, содержащую записи за все временные шаги для всех объектов соответствующего типа (скважины, группы, интервалы перфорации, отчётные регионы) и для всех загруженных моделей. Графики одинакового типа можно трансформировать друг в друга и комбинировать с помощью арифметических операций и математических функций, действующих на них поэлементно. Графики можно также комбинировать со скалярными величинами или с графиками меньшей размерности. Кроме того, имеются специальные функции, осуществляющие численное дифференцирование, интегрирование, усреднение по множествам объектов и т.д.

Все доступные мнемоники SUMMARY (см. 12.20.1) можно использовать непосредственно в коде. Они интерпретируются как объекты типа график и содержат массив данных для всех объектов соответствующего типа (скважин, интервалов перфорации и т.д.) и для всех шагов по времени.

Графики, определённые для трассеров, доступны для обращения по комбинированному имени: <имя мнемоники>_<имя трассера>.

Мнемоники доступны только на тех шагах расчёта модели, для которых записаны графики. На тех шагах, когда графики не записаны, они интерполируются предыдущим значением. Варианты частичной записи результатов описаны в разделе 8.1 Технического руководства тНавигатор.

Если в модели присутствуют переменные, заданные ключевым словом UDQ (см. 12.21.215), они могут быть вызваны по имени в коде. Они тоже интерпретируются как объекты типа график.

С точки зрения извлечения отдельных значений объект типа график функционирует как многомерный массив, индексированный объектами следующих типов (в зависимости от его собственного подтипа):

Подтип графика	**Чем индексирован**
Скважина	Модель, врем. шаг, скважина
Группа	Модель, врем. шаг, группа
Инт. перф.	Модель, врем. шаг, инт. перф.
Отч. рег.	Модель, врем. шаг, отч. рег.
Аквифер	Модель, врем. шаг, аквифер
Месторождение	Модель, врем. шаг

Например, *wopr*[*m1*,*w1*,*t1*] возвращает одно число — дебит нефти для скважины *w1* в модели *m1* на шаге *t1*. Индексирующие элементы могут быть заданы в любом порядке (так, пример выше мог быть записан в виде *wopr*[*t1*,*w1*,*m1*]). Выражение, в котором задана только часть индексов, возвращает график с соответствующим подмножеством значений. Так, *wopr*[*m1*, *w1*] возвращает график дебита нефти для скважины *w1* в модели *m1* на всех временных шагах.

Для работы с графиками как с абстрактными массивами данных может оказаться удобным перевести их в формат специализированных библиотек и далее пользоваться их свойствами и методами. Так, график может быть передан как аргумент в конструктор объекта *array* из библиотеки *numpy*, а тот в свою очередь — в *pandas.DataFrame*. Сами библиотеки для этого необходимо предварительно импортировать, см. Импорт библиотек и настройка внешнего Python:

```
import numpy
import pandas
a1 = numpy.array(wopr)
print(a1.shape)
df = pandas.DataFrame(a1)
print(df.axes)
```

Примеры управления проектами тНавигатор с помощью Python и Сервера Управления приведены в учебных курсах:

- **API1.1 Использование Сервера Управления с Дизайнером Геологии;**
- **API1.2 Использование Сервера Управления с Дизайнером Моделей;**
- **API1.3 Построение поверхностной сети с помощью Сервера Управления.**
- **API1.4 Создание проекта в Дизайнере Скважин при помощи Сервера Управления.**

Кроме графиков, существуют другие типы предопределённых объектов, которые можно использовать в коде: скважины, группы, временные шаги, интервалы перфорации, отчётные регионы и т.д. Для обращения с ними доступны следующие свойства и функции:

7.2.1. Скважина

Объект скважина имеет следующие доступные свойства и методы:

- .name — свойство, содержащее имя скважины.
  Пример: *s1* = *w1*.name

Фрагменты кода, приведённые здесь и ниже, служат только для иллюстрации синтаксиса. Они не являются самодостаточными примерами и не будут работать, если их скопировать в окно ввода калькулятора. Полноценные примеры см. в разделе Примеры использования.

- .connections — свойство, содержащее массив со всеми интервалами перфорации скважины.
  Пример: for *c* in *w1*.connections: ⟨ *некоторые действия* ⟩
- .virtual_connections — свойство, содержащее массив со всеми виртуальными интервалами перфорации скважины.
  Пример: for *c* in *w1*.virtual_connections: ⟨ *некоторые действия* ⟩
- .segments — свойство, содержащее массив со всеми сегментами скважины (только для многосегментных скважин; для прочих содержит пустой массив).
  Пример: for *s* in *w1*.segments: ⟨ *некоторые действия* ⟩
- .completions — свойство, содержащее массив со всеми группами перфораций скважины.
  Пример: for *c* in *w1*.completions: ⟨ *некоторые действия* ⟩
- .group — свойство, содержащее группу, к которой принадлежит скважина.
  Пример: *g1* = *w1*.group
- .is_producer() (без аргументов) возвращает зависящий от времени график, эквивалентный True, когда скважина является добывающей, и False в противном случае.
  Пример: if *w1*.is_producer(): ⟨ *некоторые действия* ⟩
- .is_opened() (без аргументов) возвращает зависящий от времени график, эквивалентный True, когда скважина открыта, и False в противном случае.
  Пример: if *w1*.is_opened(): ⟨ *некоторые действия* ⟩
- .is_stopped() (без аргументов) возвращает зависящий от времени график, эквивалентный True, когда скважина остановлена, и False в противном случае.
  Пример: if *w1*.is_stopped(): ⟨ *некоторые действия* ⟩
- .is_shut() (без аргументов) возвращает зависящий от времени график, эквивалентный True, когда скважина закрыта, и False в противном случае.
  Пример: if *w1*.is_shut(): ⟨ *некоторые действия* ⟩

- .get_connections_from_branch(*branch_id*=<номер>) возвращает массив со всеми интервалами перфорации определённого ствола скважины. Значение аргумента, равное 0, указывает на основной или единственный ствол скважины.
  Пример: for *c* in *w1*.get_connections_from_branch(*branch_id*=1): ⟨ *некоторые действия* ⟩

- .get_connections_from_fracture_stage(*fracture_stage_name*=<имя>) возвращает массив со всеми интервалами перфорации скважины, возникающими на определённой стадии ГРП.
  Пример:
  for *c* in *w1*.get_connections_from_fracture_stage(*fracture_stage_name*='F1'): ⟨ *некоторые действия* ⟩

- .get_completions_from_branch(*branch_id*=<номер>) возвращает массив со всеми группами перфораций определённого ствола скважины. Значение аргумента, равное 0, указывает на основной или единственный ствол скважины.
  Пример:
  for *c* in *w1*.get_completions_from_branch(*branch_id*=1): ⟨ *некоторые действия* ⟩

- .get_fracture_stages_from_branch(*branch_id*=<номер>) возвращает список всех стадий трещины указанного ствола скважины. Аргумент является опциональным; если он пропущен, возвращается список стадий для скважины.
  Пример:
  for *st* in *w1.get_fracture_stages_from_branch*(*branch_id*=1): ⟨ *некоторые действия* ⟩

- .get_closure_reason(*type*=<тип>, *substatus*=<'yes'> *или* <'no'>) возвращает одну из предопределённого списка констант (см. табл. 7.1), в которых закодирована причина закрытия скважины. В результате выполнения функции возвращается числовое или текстовое представление. Значения аргументов таковы:

  - *type*: отвечает за тип значения, которое возвращает функция. Может принимать одно из следующих значений:
    - 'enum' - внутреннее представление значения;
    - 'number' - числовой код (числовое представление значения);
    - 'string' - текстовое представление значения;
    - 'description' - описание причины закрытия;

    По умолчанию: type = 'number'.

  - *substatus*: отвечает за подробное описание причины закрытия скважины из-за ограничений, заданных в WECON (см. 12.21.112), и принимает одно из двух значений ('yes' или 'no'). Если аргумент принимает значение 'yes', то при закрытии скважины из-за экономических ограничений (представление *shut_by_wecon_cecon* из табл. 7.1) функция будет возвращать более подробное представление (см. табл. 7.2).

    По умолчанию: substatus = 'no'.

В **Калькуляторе графиков** для доступа к значению функции .get_closure_reason() с аргументом *type*='number' после самой функции в квадратных скобках необходимо указать временной шаг. Если же аргумент *type* принимает другие значения, то в квадратных скобках указывается **номер** временного шага (подробное описание см. в разделе Временной шаг)

Пример:
reason_number = *w*.get_closure_reason(type='number',*substatus*='yes')[*t*]
reason_enum = *w*.get_closure_reason(type='enum')[*t.step_number*]

7.2.2. Группа

Объект группа представляет группу скважин и имеет следующие доступные свойства и методы:

- .name — свойство, содержащее название группы.
  Пример: *s1* = *g1*.name
- .wells — свойство, содержащее массив со всеми скважинами данной группы. Скважины дочерних групп, если такие есть, не включаются.
  Пример: for *w* in *g1*.wells: ⟨ *некоторые действия* ⟩
- .all_wells — свойство, содержащее массив, рекурсивно включающий все скважины данной группы и её дочерних групп.
  Пример: for *w* in *g1*.all_wells: ⟨ *некоторые действия* ⟩
- .parent_group — свойство, содержащее родительскую группу данной.
  Пример: *g2* = *g1*.parent_group
- .child_groups — свойство, содержащее массив со всеми дочерними группами.
  Пример: for *g* in *g1*.child_groups: ⟨ *некоторые действия* ⟩

7.2.3. Сегмент

Объект сегмент имеет следующие доступные свойства:

- .name — свойство, содержащее имя сегмента (формируется автоматически из номера сегмента).
  Пример: print(*s1*.name)
- .well — свойство, содержащее скважину, к которой относится сегмент.
  Пример: print(*s1*.well.name)

7.2.4. Перфорация

Объект данного типа представляет интервал перфорации и имеет следующие доступные свойства:

- .name — свойство, содержащее имя интервала перфорации (формируется автоматически).
  Пример: *s1* = *c1*.name
- .i — свойство, содержащее координату сетки i данного интервала перфорации.
  Пример: i = *c1*.i
- .j — свойство, содержащее координату сетки j данного интервала перфорации.
  Пример: j = *c1*.j
- .k — свойство, содержащее координату сетки k данного интервала перфорации.
  Пример: k = *c1*.k
- .well — свойство, содержащее скважину, к которой относится интервал перфорации.
  Пример: print(*c1*.well.name)
- .branch_id — свойство, содержащее номер ствола скважины, к которому относится интервал перфорации (см. 2.17. Многосегментная скважина в Техническом руководстве тНавигатор). Для одноствольных скважин возвращает 0.
  Пример: print(*c1*.branch_id)
- .fracture_stage_name — свойство, содержащее имя стадии ГРП, к которой относится интервал перфорации.
  Пример: print(*c1*.fracture_stage_name)
- .lgr_name — свойство, содержащее имя локального измельчения сетки, к которому относится интервал перфорации (см. 5.5.9. Локальное измельчение сетки LGR в Техническом руководстве тНавигатор). Если интервал перфорации не относится ни к какому локальному измельчению, то в данном свойстве записано **GLOBAL**. Если измельчение создано неявным образом в процессе создания трещины ГРП (см. 5.7. Моделирование трещин ГРП в Техническом руководстве тНавигатор), имя присваивается ему автоматически.
  Пример: print(*c1*.lgr_name)
- .*completion* — свойство, содержащее группу перфораций, к которой принадлежит данная перфорация. Если группы перфораций не были определены явно, они создаются автоматически, по одной на каждую перфорацию.
  Пример: print(*c1.completion*.name)
- .traj_md_in, .traj_md_out — свойства, содержащие в виде списка кабельные глубины (MD) точек, с которых в блоке сетки начинаются сегменты траекторий, и глубины (MD) точек, которые являются их концами в этом же блоке.

Данные свойства доступны только тогда, когда перфорации заданы через траектории, т.е. ключевыми словами COMPDATMD (см. 12.21.9) и WELLTRACK (см. 12.21.8). В противном случае свойства содержат значение 0.0.

- .traj_tvd_in, .traj_tvd_out — свойства, содержащие в виде списка абсолютные глубины (TVD) точек, с которых в блоке сетки начинаются сегменты траекторий, и глубины (TVD) точек, которые являются их концами в этом же блоке.
- .is_opened() (без аргументов) возвращает зависящий от времени график, эквивалентный True, когда перфорация открыта, и False в противном случае.
  Пример: if *c1*.is_opened(): ⟨ *некоторые действия* ⟩
- .is_shut() (без аргументов) возвращает зависящий от времени график, эквивалентный True, когда перфорация закрыта, и False в противном случае.
  Пример: if *c1*.is_shut(): ⟨ *некоторые действия* ⟩

7.2.5. Группа перфораций

Объект данного типа представляет группу (участок) перфораций и имеет следующие доступные свойства:

- .name — свойство, содержащее имя группы перфораций (формируется автоматически).
  Пример: *s1* = *cm*.name
- .connections — свойство, содержащее массив со всеми интервалами перфорации данной группы.
  Пример: for *c* in *cm*.connections: ⟨ *некоторые действия* ⟩
- .well — свойство, содержащее скважину, к которой относится группа перфораций.
  Пример: print(*cm*.well.name)
- .branch_id — свойство, содержащее номер ствола скважины, к которому относится группа перфораций. Для одноствольных скважин возвращает 0.
  Пример: print(*cm*.branch_id)
- .is_opened() (без аргументов) возвращает зависящий от времени график, эквивалентный True, когда группа перфораций открыта, и False в противном случае.
  Пример: if *cm*.is_opened(): ⟨ *некоторые действия* ⟩
- .is_shut() (без аргументов) возвращает зависящий от времени график, эквивалентный True, когда группа перфораций закрыта, и False в противном случае.
  Пример: if *cm*.is_shut(): ⟨ *некоторые действия* ⟩

- .md_in, .md_out — свойства, содержащие в виде списка кабельные глубины (MD) начальных и конечных точек группы перфораций.

Данные свойства доступны только тогда, когда группы перфораций заданы 15-м параметром ключевого слова COMPDATMD (см. 12.21.9). В противном случае свойства содержат значение 0.0.

- .tvd_in, .tvd_out — свойства, содержащие в виде списка абсолютные глубины (TVD) начальных и конечных точек группы перфораций.

7.2.6. Стадия ГРП

Объект данного типа представляет стадию ГРП и имеет следующие доступные свойства:

- .name — свойство, содержащее имя стадии ГРП.
  Пример: *s1* = *fr*.name

7.2.7. Блок

Объект данного типа представляет блок сетки и имеет следующие доступные свойства:

- .name — свойство, содержащее имя блока (формируется автоматически).
  Пример: *s1* = *b1*.name
- .i — свойство, содержащее координату сетки i данного блока.
  Пример: *i* = *b1*.i
- .j — свойство, содержащее координату сетки j данного блока.
  Пример: *j* = *b1*.j
- .k — свойство, содержащее координату сетки k данного блока.
  Пример: *k* = *b1*.k
- .lgr_name — свойство, содержащее имя локального измельчения сетки (LGR), к которому относится блок (см. 5.5.9. Локальное измельчение сетки LGR в Техническом руководстве тНавигатор). Если блок не относится ни к какому локальному измельчению, то в данном свойстве записано **GLOBAL**. Если измельчение создано неявным образом в процессе создания трещины ГРП (см. 5.7. Моделирование трещин ГРП в Техническом руководстве тНавигатор), имя присваивается ему автоматически.
  Пример: *s1* = *b1*.lgr_name

Доступны в качестве объектов лишь те блоки сетки, для которых ключевым словом SUMMARY (см. 12.20.1) заказаны те или иные графики блочного типа.

7.2.8. Модель

Объект модель имеет следующие доступные свойства:

- .name — свойство, содержащее название модели (актуально, когда загружены результаты расчёта нескольких моделей).
  Пример: *s1* = *m1*.name

7.2.9. Временной шаг

Объект данного типа представляет один шаг по времени, и имеет следующие доступные свойства и методы:

- .name — свойство, содержащее представление даты шага в виде строки согласно шаблону (шаблон выбирается из выпадающего списка **Формат дат** ниже).
  Пример: *s1* = *t1*.name
- .*step_number* — свойство, содержащее номер данного временного шага.
  Пример: if *t1.step_number* == 5: ⟨ *некоторые действия* ⟩
- .to_datetime() (без аргументов) возвращает объект datetime, имеющий стандартные для Python свойства и методы.
  Пример:
  dt1 = *t1*.to_datetime()
  if *dt1.year* > 2014: ⟨ *некоторые действия* ⟩

7.2.10. Отч. рег.

Объект, представляющий отчётный регион, имеет следующие доступные свойства:

- .name — свойство, содержащее имя региона (состоит из названия семейства и номера региона в семействе).
  Пример: *s1* = *reg1*.name
- .family — свойство, содержащее название семейства, к которому принадлежит регион.
  Пример: *s1* = *reg1*.family
- .number — свойство, содержащее номер региона в семействе.
  Пример: *i* = *reg1*.number

- .alias — свойство, содержащее псевдоним региона (для моделей МатБаланса — название резервуара, иначе — имя, присвоенное ключевым словом ALIAS (см. 12.4.36), а при его отсутствии — назначенное автоматически).
  Пример: *s* = *reg1*.alias

7.2.11. Трассер

Объект трассер имеет следующее доступное свойство:

- .name — свойство, содержащее название трассера.
  Пример: *s1* = *tr1*.name

7.2.12. Аквифер

Объект аквифер (представляющий аналитический аквифер) имеет следующее доступное свойство:

- .name — свойство, содержащее название аквифера.
  Пример: *s1* = *aq1*.name

7.2.13. Численный аквифер

Объект численный аквифер имеет следующее доступное свойство:

- .name — свойство, содержащее название численного аквифера.
  Пример: *s1* = *aqn1*.name

7.2.14. Свойство сетки

Объект типа **Свойство сетки** имеет следующие свойства и методы:

- .name содержит название свойства.
- .compute_statistics(*type* = <тип>, *arithmetics* = <выражение>, *timestep* = <врем. шаг>) возвращает значение выбранной статистики свойства. Аргумент type принимает следующие значения:
  - 'min': минимальное значение свойства;
  - 'max': максимальное значение;
  - 'mean': среднее значение;
  - 'weighted_poro': средневзвешенное значение с весом пористости блока;
  - 'sum: сумма значений;
  - 'sum_of_squares': сумма квадратов значений;

- 'rms' или 'root_mean_square': среднеквадратичное отклонение от среднего значения;
- 'coefficient_of_variation': отношение среднеквадратичного отклонения к среднему значению;
- 'mean_square': средний квадрат значения;
- 'mean_absolute': среднее абсолютное значение;
- 'entries': число блоков, где определено значение.

Аргумент *arithmetics* является необязательным. Выражение играет роль фильтра; синтаксически оно должно состоять из трёх элементов:

Свойство сетки;

Знак сравнения (">", "<" или "==");

Число, с которым происходит сравнение.

Статистика будет вычислена по тем блокам, где сравнение верно.
Значение свойства вычисляется на временном шаге, соответствующем параметру *timestep*.

```
# Статистика свойства с фильтрацией по другому свойству
perm_avg = permx.compute_statistics(type = 'mean',
    arithmetics = 'INIT_PORO>0.1')
# Будет вычислено среднее значение свойства permx по блокам с пористостью
  более 0.1.
```

7.2.15. График

Объект типа график представляет один из графиков, как предопределённых, так и полученных в результате вычислений. Результат выполнения скрипта также является объектом этого типа. Он имеет следующие доступные методы:

- .fix(*model*=<модель>,*object*=<объект>,*date*=<врем. шаг>) возвращает значение указанного графика для указанной модели, объекта и временного шага, которые должны быть заданы как соответствующие объекты Python, т.е. не по имени. Тип объекта (скважина, группа, интервал перфорации или отчётный регион) должен соответствовать типу графика. Все аргументы опциональны. Если каких-то из них нет, функция возвратит структуру данных, содержащую значения графика для всех возможных значений пропущенных аргументов.
  Пример:
  graph2 = *graph1*.fix(*object*=get_well_by_name('PROD1'))
  берёт график для всех скважин и возвращает график только для одной скважины, а именно 'PROD1'.

- .max,.min,.avg,.sum(*models*=<модели>,*objects*=<объекты>,*dates*=<врем. шаги>) извлекают подмножество значений для указанных моделей, объектов и временных шагов (все аргументы могут содержать как массив, так и одно значение), и затем возвращают минимум, максимум, среднее, или сумму этого подмножества. Аргументы должны быть заданы как соответствующие объекты Python, т.е. не по имени. Тип объектов должен соответствовать типу графика. Все аргументы опциональны. Если каких-то из них нет, функция возвращает объект, содержащий значения минимума, максимума, среднего, или суммы по всем значениям заданных аргументов для всех возможных значений пропущенных аргументов.
  Пример:
  graph2 = *graph1*.max(*objects*=get_wells_by_mask('WELL3*'))
  возвращает график, содержащий максимум из значений первоначального графика для скважин с именами 'WELL3*', т.е. 'WELL31', 'WELL32', 'WELL33' и т.д.;
  graph2 = *graph1*.avg(*dates*=get_all_timesteps()[15:25])
  возвращает график, содержащий значение первоначального графика, усреднённое по временным шагам с 15-го по 24-й.

- .aggregate_by_time_interval(*interval*=<интервал>, *type*=<тип>) возвращает кусочно-постоянный график, интервалы постоянства которого задаются аргументом *interval*, а значения вычисляются из значений первоначального графика на этих интервалах по закону, указанному в аргументе *type*:

  - 'avg': среднее;
  - 'min': минимум;
  - 'max': максимум;
  - 'last': последнее значение;
  - 'sum': сумма значений;
  - 'total': разность последнего и первого значений.

  Возможные значения аргумента *interval*:

  - 'month' — месяц;
  - 'quarter' — квартал;
  - 'year' — год.

  Пример:
  w1 = *wopr*.aggregate_by_time_interval(*interval* = 'year', type = 'avg')
  возвращает график, кусочно-постоянный по годовым интервалам, где значение в каждом году равно среднему значению первоначального графика (*wopr*, т.е. дебит нефти) за этот год.

- .to_list() (без аргументов) возвращает список значений графика. Эта функция работает только в случае, если график является одномерным, т.е. зависит только от времени; в противном случае будет выдана ошибка. Чтобы получить одномерный график, необходимо исключить зависимость от модели и скважины, либо указав их явно через .fix(), либо найдя значение .min(), .max() и т.п. по всем моделям и/или скважинам.
  Пример: *x=fopr*.fix(*model*='BRUGGE_VAR_1').to_list()
  возвращает массив значений дебита нефти всего месторождения на всех временных шагах.

- .to_dataframe() (без аргументов) возвращает значения графика в формате Pandas DataFrame (многомерный массив с дополнительными методами). Пример:
  x = *wopr*().to_dataframe()
  возвращает массив значений дебита нефти для всех скважин на всех временных шагах.
  В следующем примере два графика сливаются в один объект DataFrame:

```
gr1 = wopr.to_dataframe()
gr2 = wwpr.to_dataframe()
gr_both = gr1.merge(gr2, how='left')
print(gr_both)
```

Обратите внимание, что обратная операция graph_from_dataframe() оформлена как **глобальная** функция.

7.2.16. Глобальные функции

Функции общего назначения, включая:

- get_all_wells(*type*=<тип>) возвращает массив, содержащий все скважины соответствующего типа. Тип может принимать следующие значения: 'prod', 'inje', 'open', 'shut' и 'stop'. Тип скважины проверяется по состоянию на последний временной шаг. Аргумент является опциональным; если он пропущен, возвращаются скважины всех типов.
  Пример: for *w* in get_all_wells(): ⟨*некоторые действия*⟩

Функция get_all_wells(), вызванная на любом шаге, возвращает все скважины, которые когда-либо существовали или **будут существовать** в проекте.

- get_wells_by_mask(<маска>) возвращает массив, содержащий все скважины, подходящие под маску. Маска может содержать символы **?** (любой символ) и * (любое количество символов, возможно, 0).
  Пример: for *w* in get_wells_by_mask('prod1*'): ⟨*некоторые действия*⟩

- get_well_by_name(<имя>) — возвращает объект скважины по её имени.
  Пример: *w1* = get_well_by_name('prod122')

- get_well_by_criterion(*method* = <'метод'>, *crit* = <график>, *group* = <'группа'>, *status* = <'состояние'>, *recursive* = <'тип обхода'>, *model* = <модель>, *timestep* = <временной шаг>) возвращает скважину, на которой достигает минимального или максимального значения выбранный график для указанной модели на выбранную дату. Значения аргументов:

  - *method*: либо 'min', либо 'max';
  - *crit*: график, по которому сравниваются скважины (должен иметь тип графика по скважинам);
  - *group*: группа, внутри которой происходит поиск;
  - *status*: состояние скважин, по которым происходит поиск. Возможные значения: 'prod', 'inje', 'open', 'shut', 'stop'. Является необязательным параметром. Если состояние скважины не указано, перебор происходит по скважинам всех состояний;
  - *recursive*: тип обхода. Возможные значения: 'no' — поиск только в указанной группе, 'yes' — рекурсивный обход по всем дочерним группам;
    По умолчанию: 'no'.
  - *model*: модель, в которой происходит поиск скважины. Если аргумент *model* не указан, то выбирается текущая модель;
  - *timestep*: временной шаг, на котором происходит поиск скважины. Если аргумент *timestep* не выбран, перебор происходит по всем временным шагам.

  Пример: *w1* = get_well_by_criterion(*method*='max', *crit*=*wopr*, *group*=*g1*, *status*='open', *recursive*='yes')

- get_all_groups() (без аргументов) возвращает массив, содержащий все группы.
  Пример: for *g* in get_all_groups(): ⟨*некоторые действия*⟩

- get_group_by_name(<имя>) — возвращает объект группы по её имени.
  Пример: *g1* = get_group_by_name('group21')

- get_group_by_criterion(*method* = <'метод'>, *crit* = <график>, *group* = <'группа'>, *recursive* = <'тип обхода'>, *model* = <модель>, *timestep* = <временной шаг>) возвращает группу, на которой достигает минимального или максимального значения выбранный график для указанной модели на выбранную дату. Значения параметров:

- *method*: либо 'min', либо 'max';
- *crit*: график, по которому сравниваются группы (должен иметь тип графика по группам);
- *group*: родительская группа, внутри которой происходит поиск;
- *recursive*: тип обхода. Возможные значения: 'no' — поиск только в указанной группе, 'yes' — рекурсивный обход по всем дочерним группам;
  По умолчанию: 'no'.
- *model*: модель, в которой происходит поиск группы. Если аргумент *model* не указан, то выбирается текущая модель;
- *timestep*: временной шаг, на котором происходит поиск группы. Если аргумент *timestep* не выбран, перебор происходит по всем временным шагам.

Пример: *g3* = get_group_by_criterion(*method*='min', *crit*=*gwpr*, *group*=*g2*)

- get_all_segments() возвращает массив, содержащий все сегменты во всех многосегментных скважинах (если таких нет, возвращает пустой массив.)
  Пример: for *s* in get_all_segments(): ⟨*некоторые действия*⟩
- get_segment_by_number(*well* = <'имя скважины'>, *number* = <номер>) возвращает сегмент многосегментной скважины по имени скважины и его номеру в ней.
  Пример: *s* = get_segment_by_number(*well*='W1',*number*=10)
- get_all_connections(*type*=<тип>) возвращает массив, содержащий все интервалы перфорации соответствующего типа во всех скважинах. Тип может принимать следующие значения:
  - 'virtual': возвращаются только виртуальные интервалы перфорации;
  - 'non-virtual': возвращаются только реальные интервалы перфорации;
  - 'edfm': возвращаются только интервалы перфорации с трещинами EDFM;
  - 'all': возвращаются все интервалы перфорации.

  Пример: for *c* in get_all_connections(type='all'): ⟨*некоторые действия*⟩
- get_connection_by_name(<имя>, *well* = <'имя скважины'>, *branch_id* = <номер ствола>) возвращает интервал перфорации по его имени, имени скважины и номеру ствола (обратите внимание, что имена интервалов перфорации сгенерированы автоматически и могут не быть уникальными в рамках всей модели).
  Пример:
  c1 = get_connection_by_name('[13,10,10]',*well*='Well1',*branch_id*= 1)
- get_connection_by_criterion(*method* = <'метод'>, *crit* = <график>, *well* = <'имя скважины'>, *model* = <модель>, *timestep* = <временной шаг>) возвращает интервал перфорации

выбранной скважины из указанной модели на выбранную дату, на котором достигает минимального или максимального значения выбранный график. Значения параметров:

- *method*: либо 'min', либо 'max';
- *crit*: график, по которому сравниваются интервалы перфорации (должен иметь тип графика по интервалам перфорации);
- *well*: скважина, по интервалам перфорации которой происходит поиск;
- *model*: модель, в которой происходит поиск интервала перфорации указанной скважины. Если аргумент *model* не указан, то выбирается текущая модель;
- *timestep*: временной шаг, на котором происходит поиск интервала перфорации указанной скважины. Если аргумент *timestep* не выбран, перебор происходит по всем временным шагам.

Пример: *c1* = get_connection_by_criterion(*method*='min', *crit*=*cglr*,*well*=*w1*)

- get_all_completions() (без аргументов) возвращает массив, содержащий список всех участков перфорации.
  Пример: for *cm* in get_all_completions(): ⟨*некоторые действия*⟩
- get_completion_by_name(<имя>, *well* = <'имя скважины'>) возвращает группу перфораций по её имени и имени скважины (обратите внимание, что имена групп перфораций сгенерированы автоматически и могут не быть уникальными в рамках всей модели).
  Пример: *cm* = get_completion_by_name('#11',*well*='Well1')
- get_completion_by_criterion(*method* = <'метод'>, *crit* = <график>, *well* = <'имя скважины'>, *model* = <модель>, *timestep* = <временной шаг>) возвращает группу перфораций выбранной скважины из указанной модели на выбранную дату, на котором достигает минимального или максимального значения выбранный график. Значения параметров:
  - *method*: либо 'min', либо 'max';
  - *crit*: график, по которому сравниваются группы перфораций (должен иметь тип графика по группам перфораций);
  - *well*: скважина, по группам перфораций которой происходит поиск;
  - *model*: модель, в которой происходит поиск группы перфораций указанной скважины. Если аргумент *model* не указан, то выбирается текущая модель;
  - *timestep*: временной шаг, на котором происходит поиск группы перфораций указанной скважины. Если аргумент *timestep* не выбран, перебор происходит по всем временным шагам.

Пример:
cm = get_completion_by_criterion(*method*='max', *crit*=*lcopr*,*well*=*w1*)

- get_all_well_filters() (без аргументов) возвращает массив, содержащий имена всех фильтров по скважинам. Обратите внимание, что в данном контексте фильтр по скважинам не существует как объект Python со своими характерными свойствами и методами, поэтому обращение к нему производится только по имени.
  Пример: for *wf* in get_all_well_filters(): ⟨*некоторые действия*⟩

- get_wells_from_filter(<имя фильтра>) возвращает массив, содержащий все скважины, включённые в фильтр по скважинам. Фильтр должен быть создан заранее кнопкой **Фильтр по скважинам** или иным способом.Пример: for *w* in get_wells_from_filter('first'): ⟨*некоторые действия*⟩

- create_filter(*name*=<имя фильтра>, *wells*=<массив>, *type*=<static/dynamic>) создаёт фильтр по скважинам, включающий указанные скважины.
  Имена фильтров, в отличие от имён многих других объектов, необязательно уникальны. Если фильтры с таким именем уже существуют, последний из них будет перезаписан.
  Массив *wells* должен содержать не имена, а сами объекты скважин. Третий аргумент необязателен.
  Когда фильтр создан, он становится активным.
  Пример: create_filter(*name*= 'Injectors',*wells*= [*inj1*, *inj2*, *inj3*])

- get_all_fracture_stages() (без аргументов) возвращает массив, содержащий все стадии ГРП.
  Пример: for *f* in get_all_fracture_stages(): ⟨*некоторые действия*⟩

- get_fracture_stage_by_name(<имя>) — возвращает объект стадии ГРП по его имени.
  Пример: *f1* = get_fracture_stage_by_name('frac21')

- get_all_blocks() (без аргументов) возвращает массив, содержащий список всех доступных блоков сетки.
  Пример: for *b* in get_all_blocks(): ⟨*некоторые действия*⟩

Доступны в качестве объектов лишь те блоки сетки, для которых ключевым словом SUMMARY (см. 12.20.1) заказаны те или иные графики блочного типа.

- get_block_by_name(<имя>) — возвращает объект блока сетки по его имени.
  Пример: *b1* = get_block_by_name('[10,10,1]')

- get_block_by_ijk(<i>,<j>,<k>) — возвращает объект блока сетки по его координатам IJK.
  Пример: *b2* = get_block_by_ijk(20,30,5)

- get_local_block_by_ijk(<имя LGR>,<i>,<j>,<k>) — возвращает объект блока локальной сетки по его координатам IJK в LGR.
  Пример: *local_b1* = get_local_block_by_ijk('LGR_1',10,20,3)

- get_all_models() (без аргументов) возвращает массив, содержащий все модели (актуально, когда загружены результаты расчёта нескольких моделей).
  Пример: for *m* in get_all_models(): ⟨*некоторые действия*⟩

- get_model_by_name(<имя>) — возвращает объект модели по её имени.
  Пример: *m1* = get_model_by_name('Result_1')

- get_all_timesteps() (без аргументов) возвращает массив, содержащий список всех временных шагов.
  Пример: for *t* in get_all_timesteps(): ⟨*некоторые действия*⟩

- get_timestep_from_datetime(<дата>,*mode* = <'режим'>) возвращает временной шаг по указанной дате, которая должна являться объектом Python типа date или *datetime*. В зависимости от значения параметра *mode* поиск шага происходит следующим образом:

  - 'exact_match': берётся шаг с датой, в точности совпадающей с указанной;
  - 'nearest': берётся ближайший шаг к указанной дате;
  - 'nearest_before': берётся ближайший шаг, предшествующий указанной дате;
  - 'nearest_after': берётся ближайший шаг, следующий за указанной датой;

  По умолчанию: 'exact_match'.
  Если шаг не может быть найден с учётом заданного режима поиска, или если дата находится за пределами интервала моделирования, возвращается ошибка.
  Пример:
  t1 = get_timestep_from_datetime(date(2012,7,1), *mode*='nearest_after')

  

  Большинство манипуляций с объектом *datetime* требуют предварительной загрузки соответствующей библиотеки (см. Импорт библиотек и настройка внешнего Python). Это делается так:
  from *datetime* import *datetime*

- get_all_fip_regions() (без аргументов) возвращает массив, содержащий все FIP-регионы всех семейств.
  Пример: for *reg* in get_all_fip_regions(): ⟨*некоторые действия*⟩

- get_fip_regions_from_family(<семейство>) возвращает массив, содержащий все FIP-регионы данного семейства.
  Пример: for *reg* in get_fip_regions_from_family('FIPNUM'): ⟨*некоторые действия*⟩

- get_fip_region(*family_name*=<семейство>, *fip_num*=<номер>) возвращает объект FIP-региона по имени семейства и номеру.
  Пример: *reg* = get_fip_region(*family_name*='FIPNUM', *fip_num*=1)

- get_fip_region_by_alias(<имя>) возвращает объект FIP-региона по его псевдониму.
  Пример: *reg* = get_fip_region_by_alias('FIPNUM_1')

- get_all_tracers() (без аргументов) возвращает массив, содержащий список всех трассеров.
  Пример: for *tr* in get_all_tracers(): ⟨ *некоторые действия* ⟩

- get_tracer_by_name(<имя>) — возвращает объект трассера по его имени.
  Пример: *tr1* = get_tracer_by_name('TR1')

- get_all_aquifers() (без аргументов) возвращает массив, содержащий список всех аналитических аквиферов.
  Пример: for *aq* in get_all_aquifers(): ⟨ *некоторые действия* ⟩

- get_aquifer_by_name(<имя>) — возвращает объект аквифера по его имени.
  Пример: *aq1* = get_aquifer_by_name('AQ1')

- get_all_numerical_aquifers() (без аргументов) возвращает массив, содержащий список всех численных аквиферов.
  Пример: for *aq* in get_all_numerical_aquifers(): ⟨ *некоторые действия* ⟩

- get_numerical_aquifer_by_name(<имя>) — возвращает объект численного аквифера по его имени.
  Пример: *aq1* = get_numerical_aquifer_by_name('AQN1')

- get_all_properties() (без аргументов) возвращает массив, содержащий список всех свойств сетки.
  Пример: for *p* in get_all_properties(): ⟨ *некоторые действия* ⟩

- get_property_by_name(*name*=<имя>) возвращает свойство сетки по его имени. Аргумент может быть передан как именным, так и позиционным.
  Пример: *prop* = get_property_by_name('PRESSURE')

- get_all_components() (без аргументов) возвращает массив, содержащий список всех компонент.
  Пример: for *comp* in get_all_components(): ⟨ *некоторые действия* ⟩

- get_component_by_name(<имя>) — возвращает компонент по его имени.
  Пример: *comp1* = get_component_by_name('OIL')

- get_project_folder() (без аргументов) возвращает полный путь к директории, содержащей данную модель. Он может потребоваться для операции записи на диск.
  Пример: *path* = get_project_folder()

- get_project_name() (без аргументов) возвращает имя файла текущей модели (без расширения).
  Пример: *fn* = get_project_name()

- print(<аргументы>) работает как стандартная функция Python, выдавая любое количество аргументов в окно консольного вывода.
  Пример:
  ```
  for w in get_all_wells():
      print('In well', w.name, 'BHP =', wbhp[w])
  ```

7.2.17. Глобальные функции графиков

Глобальные функции для работы с графиками, включая:

- graph(*type*=<'тип'>, *default_value*=<значение>) — возвращает график указанного типа/*type* ('field', 'well', 'group', 'conn' — интервалы перфорации или 'fip' — отчётные регионы), заполненный значениями по умолчанию.
  Пример: *tmp* = graph(type='field', *default_value*=1)

- diff(<ряд>) — численное дифференцирование временного ряда, т.е. его преобразование в ряд разностей.
  Пример: *graph2* = diff(*graph1*)
  Здесь мы рассчитываем значения нефтедобычи за временной шаг по накопленным значениям:

$$465, 1165, 2188, 3418, 4968, \ldots \longmapsto 465, 700, 1023, 1230, 1550, \ldots$$

- diff_t(<ряд>) — то же, что diff, с последующим делением результатов на длины временных шагов в днях. Пример: *graph2* = diff_t(*graph1*)
  Здесь мы рассчитываем значения дебитов скважин по накопленным значениям. Пусть шаги по времени соответствуют месяцам и имеют длину 31, 28, 31, 30, 31, ... дней. Тогда:

$$465, 1165, 2188, 3418, 4968, \ldots \longmapsto 15, 25, 33, 41, 50, \ldots$$

- cum_sum(<ряд>) — численное интегрирование временного ряда, т.е. его преобразование в ряд сумм.
  Пример: *graph3* = cum_sum(*graph1*)
  Здесь мы рассчитываем накопленные значения нефтедобычи по значениям за временной шаг:

$$465, 700, 1023, 1230, 1550, \ldots \longmapsto 465, 1165, 2188, 3418, 4968, \ldots$$

- cum_sum_t(<ряд>) — то же, что cum_sum, с умножением добавляемых величин на длины временных шагов в днях.
  Пример: *graph3* = cum_sum_t(*graph1*)
  Здесь мы рассчитываем накопленные значения нефтедобычи по дебитам. Пусть шаги по времени соответствуют месяцам и имеют длину 31, 28, 31, 30, 31, ... дней. Тогда:
  $$15, 25, 33, 41, 50, \ldots \longmapsto 465, 1165, 2188, 3418, 4968, \ldots$$

- exp(<число>), ln(<число>), sqrt(<число>), abs(<число>) — математические функции (экспонента, логарифм, квадратный корень и модуль, соответственно). При передаче графика в качестве аргумента применяются к нему поэлементно.
  Примеры:
  t = ln(*y*)
  x = exp(*r*)

- if_then_else(<условие>,<вариант если да>,<вариант если нет>) — условный оператор, который применяется к графикам поэлементно.
  Пример: *graph1* = if_then_else(*wopr* > 10, 1, 0)

- create_table_vs_time(<ряд>) возвращает график, содержащий кусочно-линейную интерполяцию данного ряда значений. Ряд должен представлять собой массив кортежей из двух элементов **(дата,значение)**. Дата должна являться объектом Python типа date или *datetime*.
  Пример:
  oil_price_list = []
  oil_price_list.append((date(2011,1,1),107.5))
  oil_price_list.append((date(2012,1,1),109.5))
  oil_price_list.append((date(2013,1,1),105.9))
  oil_price_list.append((date(2014,1,1), 96.3))
  oil_price_list.append((date(2015,1,1), 49.5))
  oil_price_list.append((date(2016,1,1), 40.7))
  oil_price = create_table_vs_time(*oil_price_list*)

  Здесь мы строим график цен на нефть. Для наглядности точки в массив добавляются по одной.

- shift_t(*graph*=<график>, *shift*=<сдвиг>, *default*=<значение по умолчанию>) возвращает первоначальный график, сдвинутый на указанное число шагов. Пропущенные позиции заполняются указанным значением по умолчанию.
  Пример: *woprh_corr* = shift_t(*graph*=*woprh*, shift=3, *default*=10)
  В этом примере мы смещаем ряд исторических значений дебита нефти, который был ошибочно введён с неправильными датами. Смещение производится на 3 шага вправо, пустые позиции заполняются значением дебита на первом шаге (10):
  $$\underbrace{10, 12, 19, 24, 30, 33, 31, 27, 25, \ldots}_{woprh} \longmapsto \underbrace{10, 10, 10, 10, 12, 19, 24, 30, 33, 31, 27, 25, \ldots}_{\text{shift\_t}(graph=woprh,shift=3,default=10)}$$

- export(<выражение>, *name* = <'имя'>, *units* = <'единицы'>) экспортирует переданное выражение в пользовательский график, задавая также его имя и (опционально) единицы измерения.
  Выражение должно при вычислении давать результат типа график, иначе будет возвращена ошибка.
  Единицы измерения задаются мнемоническим именем, которое можно выбрать из выпадающего списка справа.
  Пример: export(*w1*, *name*='graph1')

- get_global_graph(*name*=<'имя'>) импортирует и возвращает пользовательский график с указанным именем, который мог быть создан, например, в другом скрипте.
  Пример: *gr1* = get_global_graph(*name*='graph1')

- graph_from_dataframe(*type* = <'тип'>, *default_value* = <значение>, *dataframe* = <объект DataFrame>, *object* = <'имя колонки объектов'>, *date* = <'имя колонки дат'>, *value* = <'имя колонки значений'>) — создаёт график, имеющий указанный тип/*type* ('field', 'well', 'group' и т.д.) и заполненный значениями из объекта Pandas DataFrame — массива произвольной размерности, который может содержать значения для ряда объектов и дат.
  Пример: набор данных прочитывается из CSV файла с помощью встроенных средств Pandas и преобразуется в график.

```
import pandas as pd
data = pd.read_csv(get_project_folder()+'/hist.txt',
  header=0, sep=',', names=['Date','Well','Oil_rate'])
data['Date'] = pd.to_datetime(data['Date'],
  format="%Y-%m-%d")
hist = graph_from_dataframe(type='well',
  default_value = 0.0, dataframe = data, object = 'Well',
  date = 'Date', value = 'Oil_rate')
export(hist, name = 'dp_historical')
```

Обратите внимание, что обратная операция .to_dataframe() оформлена как **метод** объекта График, а не глобальная функция.

Таблица 7.1. *Возможные причины закрытия скважины*

<number>	<enum>	**Состояние скважины и причина закрытия**
0	*well_open*	Открыта (т.е. не закрыта)
1	*stopped_by_kw*	Остановлена ключевым словом
2	*shut_by_kw*	Закрыта ключевым словом
3	*shut_by_physical_reason*	Закрыта по физическим критериям
4	*shut_by_wecon_cecon*	Закрыта из-за нарушения экономических ограничений
5	*shut_by_conn_econ*	Закрыта из-за нарушения экономических ограничений на уровне перфорации
6	*shut_by_group_econ*	Закрыта из-за нарушения экономических ограничений на уровне группы
7	*shut_by_priority_control*	Закрыта по причинам приоритета
8	*shut_by_wthpmax*	Закрыта из-за превышения максимального THP
9	*shut_by_wcycle*	Закрыта в порядке циклической работы (WCYCLE, см. 12.21.74)
10	*well_undrilled*	Ещё не пробурена
11	*well_undefined*	Ещё не существует (это состояние любой скважины до появления первых ключевых слов, относящихся к ней)

Таблица 7.2. *Подробные причины закрытия скважины из-за экономических ограничений*

<number>	<enum>	**Причина закрытия и параметр** WECON (см. 12.21.112)
401	*shut_by_wecon_oil_rate*	Закрыта из-за нарушения экономического ограничения по дебиту нефти (2-ой параметр)
402	*shut_by_wecon_gas_rate*	Закрыта из-за нарушения экономического ограничения по дебиту газа (3-ий параметр)
403	*shut_by_wecon_watercut*	Закрыта из-за нарушения экономического ограничения по обводнённости (4-ый параметр)
404	*shut_by_wecon_GOR*	Закрыта из-за нарушения экономического ограничения по газовому фактору (5-ый параметр)
405	*shut_by_wecon_WGR*	Закрыта из-за нарушения экономического ограничения по водо-газовому фактору (6-ой параметр)
406	*shut_by_wecon_GLR*	Закрыта из-за нарушения экономического ограничения по газо-жидкостному фактору (13-ый параметр)
407	*shut_by_wecon_liquid_rate*	Закрыта из-за нарушения экономического ограничения по дебиту жидкости (14-ый параметр)

7.3. Импорт библиотек и настройка внешнего Python

тНавигатор содержит внутреннюю копию Python версии 3.11.3 с набором стандартных библиотек. В частности, набор включает:

- joblib-1.0.0
- numpy-1.19.4
- OpenBLAS-0.3.13
- pandas-1.1.5
- python-dateutil-2.8.1
- pytz-2020.4
- scikit-learn-0.24.0
- scipy-1.5.4
- six-1.15.0
- threadpoolctl-2.1.0
- openpyxl-3.1.2

Чтобы увидеть список всех установленных библиотек, наберите следующее в редакторе кода и нажмите **Вычислить**:

```
help('modules')
```

Эти библиотеки могут быть подключены непосредственно:

```
import itertools
```

Для использования других библиотек можно установить внешний Python и сконфигурировать тНавигатор, чтобы использовать данный экземпляр Python в графическом калькуляторе.

Для возможности использования внешнего Python необходимо, чтобы тНавигатор был установлен с **внутренним** Python.

Обратите внимание, что произведённая настройка внешнего Python влияет одновременно на скрипты в SCHEDULE (см. ключевое слово APPLYSCRIPT, см. 12.21.218), выполнение workflow и графический калькулятор.

Допускается использовать 64-битную версию Python 3.11.x, скачанную с официального сайта https://www.python.org. В более новых версиях возможны проблемы бинарной совместимости.

ИРМ не несет ответственности за последствия использования других дистрибутивов Python или непроверенных сторонних библиотек.

Чтобы подключить собственные или сторонние библиотеки, необходимо проделать следующее:

1. Установите Python версии 3.11.x (предпочтительно 3.11.3). В случае ОС Windows при использовании официального дистрибутива рекомендуется установка для всех пользователей. Для сторонних дистрибутивов (например, Anaconda) установка для всех пользователей не требуется. В случае ОС Linux Python устанавливается в домашний каталог пользователя.

2. Рекомендуется (хотя не обязательно) добавить Python в системную переменную **PATH**. В случае ОС Windows при использовании официального дистрибутива это может быть выполнено в процессе установки или позднее. Для ОС Linux, а также для сторонних дистрибутивов под Windows (например, Anaconda) требуется ручное добавление Python в **PATH**.

   - Если переменная **PATH** изменена, когда тНавигатор был запущен, может потребоваться перезапуск тНавигатор, чтобы изменения вступили в силу.
   - Чтобы посмотреть или отредактировать **PATH** в Windows, нажмите **Мой компьютер** → **Свойства** → **Дополнительные параметры** → **Переменные среды** → **PATH**.
   - Чтобы посмотреть **PATH** в Linux, используйте команду: echo $PATH. Способ редактирования **PATH** зависит от используемой командной оболочки.

3. В этом экземпляре Python установите нужные Вам библиотеки.

После установки Python рекомендуется выйти и повторно войти в систему и проверить установленную версию командой: python --version
(Предполагается, что либо директория Python добавлена в **PATH**, либо данная команда выполнена из неё.)

4. В главном окне тНавигатор перейдите на **Настройки** → **Опции** → **Пути**.

5. Измените следующие параметры:

5.1. Отметьте **Исп. внешнюю библиотеку Python** и введите путь к исполняемым файлам и библиотекам для нового экземпляра Python.

Чтобы узнать путь к исполняемому файлу python под Linux, следует выполнить команду:
which python3.11

С помощью кнопки **Автоопределение** можно ввести путь к Python автоматически. Для работы этой функциональности Python должен быть добавлен в **PATH** (см. выше).

5.2. Для некоторых дистрибутивов Python (в частности, Anaconda) требуется задать путь к модулям. В этом случае отметьте **Вручную задать путь к модулям Python** и введите путь к импортированным библиотекам Python. Можно вводить несколько путей, разделённых точками с запятой.

Чтобы получить путь к библиотекам, используемым в установленном экземпляре Python, выполните в интерактивном интерпретаторе Python следующие команды:

- Для Windows:
```
import sys
print(';'.join(sys.path))
```
- Для Linux:
```
import sys
print(':'.join(sys.path))
```

Если внешний экземпляр Python удалён, тНавигатор автоматически переключается на использование внутренней копии Python.

Использование внешнего Python со сторонними библиотеками (в частности, с Win32 API) описано в учебном курсе **АНМ1.12. Использование экономической модели**;

7.3.1. Кодировка

Все файлы со скриптами Python, которые содержат символы за пределами ASCII и предназначены для взаимодействия с Python-инфраструктурой тНавигатор (через импорт или иными путями), должны сохраняться в кодировке UTF-8. В противном случае возможны нежелательные последствия, от нечитаемых текстовых сообщений до неработоспособности скрипта.

7.3.2. Известные проблемы

Модуль Python *multiprocessing* не работает в тНавигатор. Как следствие, невозможны параллельные расчёты в модулях *scipy*, *sklearn* и т.п.

7.4. Примеры использования

Данные фрагменты кода будут функционировать полноценно только при условии их запуска на модели, в которой имеются все объекты и структуры, подразумеваемые в коде. Например, если код обращается к скважине под названием **P1**, а в модели её нет, будет возвращена ошибка.

Пример 1

Предположим, мы хотим узнать, сколько нефти накоплено скважиной за определённый период, или (в зависимости от времени) за прошедшую часть этого периода.

Скрипт работает следующим образом:

1. Создать график (x) со значением дебита нефти (*wopr*) на интересующем нас интервале, и 0 за его границами. Для этого сравнить *time* (время со старта, в днях) с границами интервала и затем использовать результаты сравнений в арифметическом выражении. При этом булевские величины автоматически приводятся к целым: True становится 1, False — 0.
2. Посчитать накопленную сумму от x.
3. Экспортировать получившийся график.

```
x = wopr * (time >= 215) * (time <= 550)
w1 = cum_sum_t(x)
export(w1, name = 'PeriodProd', units = 'liquid_surface_volume')
```

Созданный график может быть отображён на секторных диаграммах.

Пример 2

Предположим, мы хотим узнать, какая часть дебита скважин идёт из слоёв сетки $70 \leqslant k < 100$.

Это возможно в интерфейсе симулятора или Дизайнера Моделей, где графический калькулятор имеет доступ к данным по отдельным интервалам перфорации, но не в интерфейсе модуля Адаптации и Оптимизации.

Скрипт работает следующим образом:

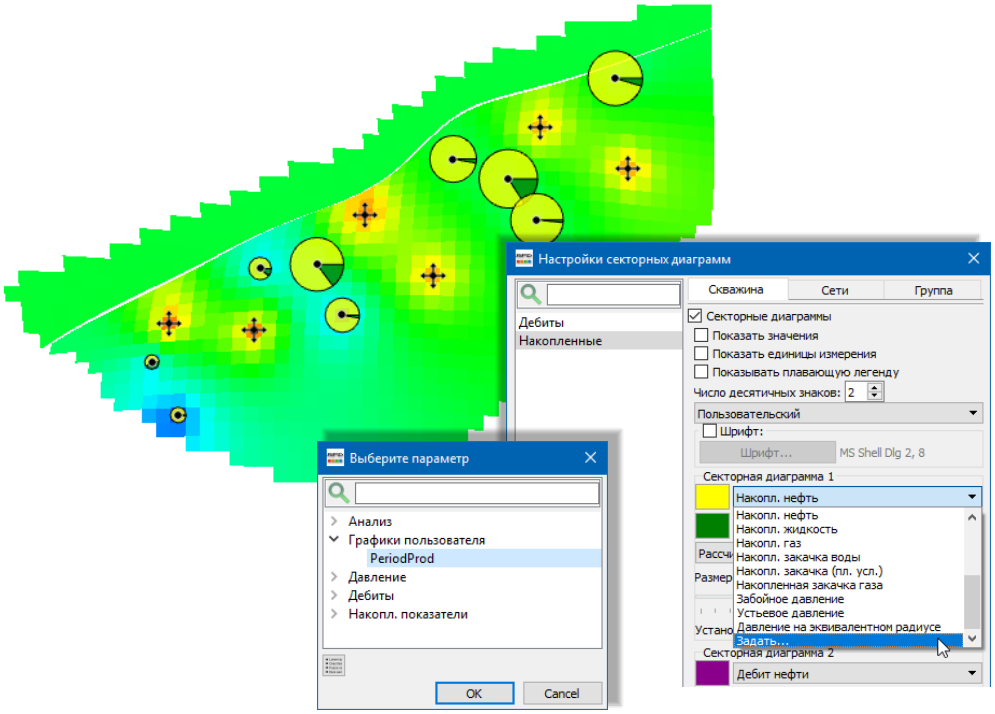

Рис. 124. *Пользовательский график на секторной диаграмме*

1. Инициализировать временную структуру данных (*tmp*) типа график по скважинам и заполнить её значением 0;

2. Итератором пройти по всем интервалам перфорации, и на каждом:

    - Проверить, находится ли интервал перфорации в нужной области, и если да, то:

        - Добавить дебит этого интервала во временную структуру данных на позицию соответствующей скважины;

3. Экспортировать временную структуру, делённую на общие дебиты по скважинам (деление графиков происходит поэлементно, т.е. сумма по интервалам перфорации для каждой скважины делится на дебит этой скважины).

```
tmp = graph(type='well', default_value=0)
for c in get_all_connections():
  if c.k in range(70,100):
    tmp[c.well] += copr[c]
export(tmp/wopr, name='wopr_layer2')
```

Обратите внимание на пробелы в начале строк. Они синтаксически важны в языке Python, но могут быть потеряны при копировании и вставке.

Пример 3

Предположим, мы хотим рассчитать средний дебит нефти для подмножества скважин, названия которых начинаются с 'WELL3', затем сравнить его с историческими данными, которые необходимо прочитать из файла, и использовать результат как целевую функцию для адаптации. Скрипт работает следующим образом:

1. Импортировать стандартную библиотеку *datetime*, которая позволяет более гибко манипулировать датами.
2. Передать в функцию avg массив с нужным подмножеством скважин, чтобы получить требуемое среднее (*obs*).
3. Открыть для чтения файл 'input.txt', находящийся в директории модели.
4. Преобразовать массив строк файла в массив кортежей **(строка,значение)**.
5. Перебрать массив ещё раз с разбором строк, чтобы получить массив кортежей **(дата,значение)**.
6. Построить график интерполяции по прочитанным данным (*hist*).
7. Построить и экспортировать график квадрата отклонения наблюдаемых данных от исторических.

```
from datetime import datetime
obs = wopr.avg(objects = get_wells_by_mask('WELL3*'))
inpf = open(get_project_folder()+'/input.txt', 'r')
raw = [(line.split()[0],float(line.split()[1]))
  for line in inpf]
arr = [(datetime.strptime(x[0], '%d.%m.%Y'),x[1])
  for x in raw]
hist = create_table_vs_time(arr)
export((obs - hist)**2, name='fuobj')
```

Пример 4

Предположим, имеется график исторического забойного давления по скважинам, где только часть измерений реальны; остальные позиции заполнены нулями. Мы хотим интерполировать данные на весь диапазон по времени. Скрипт работает следующим образом:

1. Инициализировать временную структуру данных (*tmp*) типа график по скважинам и заполнить её значением 0;
2. Итератором пройти по всем моделям и всем скважинам, и на каждой сделать следующее:
   - Извлечь BHP для данной скважины;
   - Создать пустой массив для хранения реально измеренных BHP (*observed*);
   - Итератором пройти по всем шагам по времени:
     - Если на этом шаге BHP > 0, то
     - добавить его в массив;
   - Если в массиве хотя бы 2 элемента,
   - создать интерполированный график и поместить его во временную структуру;
3. Экспортировать временную структуру.

```
tmp = graph(type = 'well', default_value = 0)
for m in get_all_models():
  for w in get_all_wells():
    current = wbhph[m,w]
observed = []
for t in get_all_timesteps():
  if current[t] > 0:
    observed.append((t.to_datetime(), current[t]))
if len(observed) >= 2:
  tmp[m,w] = create_table_vs_time(observed)
export(tmp, name = 'interpolated_wbhph')
```