Библиотека `aiwlib`

Иванов А.В., Хилков С.А. и Жданов С.А

3 августа 2017 г.

Оглавление

1 Начало работы **5**

1.1 Сборка и установка библиотеки 5

2 Ядро библиотеки **9**

2.1 Средства отладки — модуль `debug` 9

2.1.1 Общие замечания 9

2.1.2 Синтаксические ограничения 10

2.1.3 Режимы работы 11

2.1.4 Вывод информации со стека при обработке исключения 11

2.1.5 Детали реализации 12

2.2 Выделение и освобождение ресурсов в контейнерах и потоках ввода/вывода — модуль `alloc` 12

2.3 Потоки ввода/вывода — модули `iosream`, `gzstream` и `binaryio` 13

2.3.1 Общие замечания 13

2.3.2 Типобезопесный форматированный вывод 14

2.3.3 Абстрактный класс `aiw::IOstream` 14

2.3.4 Класс `aiw::File` 15

2.3.5 Класс `aiw::GzFile` 15

2.3.6 Операторы бинарного ввода/вывода — модуль `binaryio` 16

2.3.7 Детали реализации 16

2.4 Вектора и индексы — модуль `vec` 16

2.4.1 Общие замечания 16

2.4.2 Конструкторы и порождающие функции 17

2.4.3 Арифметические операции и операции сравнения 18

2.4.4 Операции и методы для доступа к элементу, изменения размерности и различные преобразования 21

2.4.5 Другие операции и методы 22

2.4.6 Детали реализации 22

2.5 Равномерные многомерные прямоугольные (картезианские) сетки — модуль `mesh` 23

2.5.1 Общие замечания 23

2.5.2 Поля и методы для получения информации о сетке 24

2.5.3 Инициализация сетки и настройка осей 24

2.5.4 Доступ к ячейкам 25

2.5.5 Обход сетки 26

2.5.6 Преобразования сеток 27

2.5.7 Сохранение и загрузка сеток . . . 28
2.5.8 Другие методы . . . 29
2.5.9 Инстацирование в Python . . . 30
2.6 Различные варинаты интерполяции — модуль inteporlations . . . 30
2.6.1 Локальный кубический сплайн . . . 30
2.6.2 Кубический B–сплайн . . . 31
2.6.3 Функции модуля interpolations . . . 32
2.7 Загрузка сеток из .dat–файлов — модуль dat2mesh . . . 32
2.8 Построение линий постоянного уровня для двумерной сетки — модуль isolines 34
2.9 Многомерный кубический массив на основе Z–кривой Мортона — модуль zcube 35
2.10 Чтение и запись сейсмических данных в формате seg-Y — модуль segy . . . 37
2.11 Контрольные точки для остановки и последующего восстановления расчета — модуль checkpoint . . . 40
2.12 Сериализация данных в формате pickle языка Python — модуль pickle . . . 41
2.13 Дискретизация разбиения Вороного на равномерной сетке в D–мерном пространстве — модуль voronoy . . . 43
2.14 Описание пользовательских типов — модуль typeinfo . . . 43
2.15 Некоторые элементы аналитической геометрии — модуль angem . . . 44
2.16 Решение СЛАУ с трехдиагональными матрицами (методы прогонки) — модуль 3diag . . . 45
2.17 Источник случайных числе и специфические операции в трехмерном декартовом пространстве — модуль gauss . . . 45
2.17.1 Определение поворота . . . 46
2.17.2 Разложение поворота в ряд . . . 47
2.18 Операции бинарного потокового ввода/вывода — модуль binaryio . . . 51
2.19 Настройка пользовательских классов — модуль objconf . . . 51
2.20 Чтение и запись конфигурационных файлов — модуль configfile . . . 52

Глава 1

Начало работы

1.1 Сборка и установка библиотеки

Для сборки библиотеки используется утилита `GNU Make`. Для работы в `C++` сборка не требуется, достаточно заголовочных файлов. Для работы в `Python` необходимо собрать ряд модулей с применением утилиты `SWIG`, что требует вызова команды

```
make
```

из директории библиотеки.

Для установки библиотеки используйте команды (из под `root`)

```
make install
```

для копирования собранных модулей и заголовочных файлов в системные директории, либо

```
make install-links|links-install
```

для создания мягких ссылок на собранные модули и заголовочные файлы в системных директориях. Второй вариант предпочтительней с точки зрения простоты обновления библиотеки, но может иметь некоторые уязвимости с точки зрения безопасности при работе в многопользовательском режиме — гипотетически пользователь установивший у себя библиотеку может подсунуть другим пользователям вредоносный код.

Предыдущая версия, библиотека `aivlib`, имела два существенных недостатка — сложную установку и проблемы при работе с несколькими версиями библиотеки на одной машине. Текущая версия допускает локальную установку произвольного числа версий/копий библиотеки, более того это рекомендуется для упрощения переноса проектов на другие машины, где библиотека `aiwlib` не установлена — в этом случае вместо копирования всей библиотоеки достаточно создать мягкую ссылку на библиотеку в директории проекта.

Для упрощения сборки проекта рекомендуется использовать файл `include/aiwlib/user.mk`. При этом пользовательский `Makefile` должен иметь вид

```
name=NAME #имя проекта
headers=... #список хидеров обратываемых SWIG-ом
modules=... #список .cpp модулей проекта
```

```
sources=... #другие исходные файлы проекта

include local-path-to-aiwlib/include/aiwlib/user.mk
```

При этом заголовочные файлы библиотеки всегда включаются как `<aiwlib/...>`, путь к локально установленной библиотеке при необходимости определяется автоматически на основе пути к файлу `user.mk`.

По умолчанию, такой `Makefile` собирает модуль для питона, и предоставляет еще ряд целей

- `sources` — выводит список исходных файлов проекта включая `make`–файл, хидеры определяются автоматически при помощи вызова `g++ -M` на основе переменной `modules`, заголовочные файлы библиотеки `aiwlib` **НЕ** включаются в список. Если при вызове `make` указать опцию `with=swig`, в список будут включены файлы `NAME.i`, `NAME.py` и `NAME_wrap.cxx`

- `all_sources` — выводит список всех исходных файлов проекта включая `make`–файл, хидеры определяются автоматически при помощи вызова `g++ -M` на основе переменной `modules`, заголовочные файлы библиотеки `aiwlib` включаются в список **ПРИ УСЛОВИИ ЕЕ ЛОКАЛЬНОЙ УСТАНОВКИ**. Если библиотека `aiwlib` установлена локально и при вызове `make` указана опция `with=aiwlib`, в список включаются необходимые файлы `.py` и `.i` библиотеки `aiwlib`. Если библиотека `aiwlib` установлена локально и при вызове `make` указана опция `with=aiwlib,swig`, (последовательность не имеет значения) в список включаются все файлы `.py`, `.i` и `_wrap.cxx` библиотеки `aiwlib`, а так же файлы `NAME.i`, `NAME.py` и `NAME_wrap.cxx`.

- `clean` — удаляет все созданные объектные файлы и файл `_NAME.so`.

- `cleanall` — удаляет все созданные объектные и `.so` модули, а так же файлы `NAME.i`, `NAME.py` и `NAME_wrap.cxx`.

- `NAME.tgz NAME.md5` — создает сжатый архив и файл с контрольной суммой **несжатого** архива со списком файлов проекта `sources`, опция `with` влияет на список файлов.

- `tar NAME-all.tgz` — создает сжатый архив `NAME-all.tgz` со списком файлов проекта `all_sources`, опция `with` влияет на список файлов.

- `export to=...` — переносит проект в другую директорию (если опция `to` имеет вид `to=PATH`) либо на другую машину (если опция `to` имеет вид `to=HOST:PATH`). Для переноса используется список файлов `all_sources`, опция `with` влияет на список файлов. Перенос на другую машину осуществляется при помощи утилиты `SSH`, если у Вас не настроена авторизация по открытому ключу то пароль придется вводить трижды. Директория `PATH` создается при необходимости, включая родительские каталоги. Корректный перенос файлов библиотеки `aiwlib` возможен только при условии ее установки в поддиректорию проекта.

Таким образом, опция `with=swig` необходима, если на целевой машине отсутствует утилита `SWIG`. Опция `with=aiwlib` необходима, если библиотека `aiwlib` была установлена локально в поддиректорию проекта.

В питоне, при импорте модулей библиотеки `aiwlib` будут импортироваться модули из локальной копии библиотеки, при условии что до этого был импортирован собранный пользовательский модуль.

Глава 2

Ядро библиотеки

2.1 Средства отладки — модуль `debug`

2.1.1 Общие замечания

Модуль `debug` предоставляет ряд макросов для вывода отладочной информации в процессе исполнения (фактически удобную альтернативу традиционным отладочным `printf`) и генерации исключений:

- `WOUT(expressions...)` — вывод информации в `std::cout`;
- `WERR(expressions...)` — вывод информации в `std::cerr`;
- `WSTR(S, expressions...)` — вывод информации в поток `S`, являющийся наследником `std::ostream`;
- `WEXC(expressions...)` — вывод информации со стека в `std::cerr` при обработке исключения (потокобезопасный вариант);
- `WEXT(expressions...)` — вывод информации со стека в `std::cerr` при обработке исключения или ошибке сегментирования (**не** потокобезопасный вариант);
- `WASSERT(condition, message, expressions...)` — вывод инфомации в `std::cerr` при нарушении условия `condition`;
- `WARNING(message, expressions...)` — вывод информации в `std::cerr`;
- `WRAISE(message, expressions...)` — вывод информации в `std::cerr` и генерация исключения типа `const char *` содержащего выведенную информацию.

Например

```
int a; double b[3];
...
WOUT(a, a*b[1], b[0]+b[2]);
```

Все макросы выводят информацию в виде

```
#filename function() LLL: expr1=value1, expr2=value2 ...
```

или

```
#filename function() LLL: message [SYSERR] expr1=value1, expr2=value2 ...
```

где `LLL` — номер строки в файле `filename` в которой был сгенерирован вывод сообщения, `function()` — имя функции в которой был сгенерирован вывод сообщения, `expr` — выражение, `value` — значение выражения.

Макрос `WRAISE` допонительно включает в сообщение информацию о системной ошибке `[SYSERR]` (результат работы фунцкии `strerror(errno)`), если состояние ошибки установлено.

Кроме того, модуль `debug` предоставляет функцию

```
void init_segfault_hook();
```

устанавливающий перехватчик сигнала `SIGSEGV` возбуждающегося при ошибке сегментирования. В этом случае на `stderr` выводится информация со стека исключения (помещенная туда при помощи макроса `WEXT`), выводится стек вызовов и инструкция (команда для `bash`) по его раскрутке.

2.1.2 Синтаксические ограничения

В качестве выражений (аргументов макросов) могут использоваться любые `rvalue` выражения, для значений которых реализованы операторы вывода в поток

```
std::ostream& operator << (std::ostream&, expr_type)
```

В выражениях могут присутствовать скобки `()[]{}`, операции `<>` скобками **не** считаются. Если в выражении есть запятые, части содержащие запятые так же должны быть в скобках, например

```
pow(a, b)
```

В противном случае макросы сохраняют работоспособность, но вывод может иметь странный вид, например

```
template <int D, typename T> struct V{
    T p[D];
    V(T x){ for(int i=0; i<D; ++i) p[i] = x; }
};

template <int D, typename T>
std::ostream& operator << (std::ostream& out, const V<D,T> &v){
    out<<"{"<<v.p[0];
    for(int i=1; i<D; ++i) out<<" "<<v.p[i];
    return out<<"}";
}
...
    int a=2;
    WOUT(V<3,int>(a), a*2);
```

даст вывод

```
# ... : V<3={2 2 2}, int>(a), a*2=4
```

вместо ожидаемого

```
# ... : V<3, int>(a)={2 2 2}, a*2=4
```

Для корректного вывода необходимо использовать дополнительные скобки

```
WOUT((V<3,int>(a)), a*2);
```

В одной строке может использоваться только один макрос WEXC.

2.1.3 Режимы работы

Макросы WOUT, WERR, WSTR, WEXC, WEXT и WASSERT работают только если определен макрос EBUG (например при помощи опции компилятора -DEBUG). При сборке на основе шаблонного aiwlib/Makefile макрос EBUG по умолчанию отключен, для его подключения необходимо использовать команду

```
make -DEBUG ...
```

Отличие между вызовом

```
WASSERT(condition, ...)
```

и

```
if(!(condition)) WRAISE(...)
```

заключается в том, что при отключенном режиме отладке макрос WASSERT игнорируется полностью (включая проверку условия).

Макросы WARNIG и WRAISE работают всегда, вне зависимости от макроса EBUG.

2.1.4 Вывод информации со стека при обработке исключения

Макросы WEXC и WEXT выводят свои аргументы на стандартный поток ошибок std::cerr при обработке исключения. Типовой ситуацией является возбуждение исключения в C++ функции, вызываемой из Python, если модуль был собран при помощи шаблонного aiwlib/Makefile — в этом случае в Python происходит вызов стандартного обработчика исключений.

Выводятся только аргументы макросов WEXC и WEXT, размещенных на стеке **до** возбуждения исключения. Для каждого аргумента выводится значение, которое принимал аргумент в момент вызова макросов WEXC и WEXT.

В одной строке может использоваться только один макрос WEXC/WEXT — это связано с размещением в строке экземпляра класса aiw::DebugStackTupleFrame, с именем формируемым на основе номера строки. При этом копии аргументов макроса хранятся внутри экземпляра класса в виде std::tuple, сообщение формируется и выводится в std::cerr дестуктором экземпляра класса только при необходимости, если возникла исключительная ситуация

или сигнал SIGSEGV. Наличие исключительной ситуации проверяется при помощи фунцкии std::uncaught_exception().

Если режим отладки выключен (макрос EBUG не определен), макросы WEXC и WEXT игнорируются.

При работе макросов WEXC и WEXT экземпляры классов aiw::DebugStackTupleFrame размещаются на стеке, но макрос WEXT дополнительно размещается указатели на создаваемые экземпляры классов в глобальной таблице, что не является потокобезопасным. В случае возниковноения сигнала SIGSEGV (и если до этого была вызвана функция init_segfault_hook()), все зарегистристрированные в глобальной таблице экземпляры классов aiw::DebugStackTupleFrame выводят сообщения в поток stderr.

2.1.5 Детали реализации

Модуль debug это легкий (звголовчный файл около 100 строк и файл src/debug.cpp около 50-ти строк), независимый от остальных частей библиотеки aiwlib модуль.

Вывод выражений построен на рекурсивной функции

```
template <typename ... Args>
void aiw::debug_out(std::ostream& out, const char* str, Args ... args);
```

вызываемой из макросов, в качестве str подставляются аргументы макроса в виде строки и затем еще раз в виде аргументов (уже значений соответствующих выражений).

Функция aiw::debug_out разбирает str по запятым, учитывая при этом скобки ()[]{}.

При выводе сообщений от всех макросов метод потока вывода flush **не** вызывается.

2.2 Выделение и освобождение ресурсов в контейнерах и потоках ввода/вывода — модуль alloc

Для безопасного копирования экземпляров потоков ввода/вывода aiw::IOstream, контейнеров библиотеки aiwlib и освобождения ресурсов используются «умные» указатели std::shared_ptr.

В заголовочном файле alloc объявлен абстрактный класс aiw::BaseAlloc, предоставляющий интерфейс для работы с выделенными ресурсом (областью памяти или мапированным файлом). Класс имеет следующие методы

- void* get_addr() — возвращает адрес контролируемой области памяти;
- size_t get_size() const — возвращает размер контролируемой области памяти в байтах;
- virtual ~BaseAlloc(){} — виртуальный деструктор;
- virtual size_t get_sizeof() const = 0 — возвращает размер элемента (ячейки массива) в байтах.

Класс template<T> aiw::MemAlloc является наследником класса aiw::BaseAlloc, и кроме перегрузки соответвующих методов предоставляет конструктор

```
template<typename ... Args> MemAlloc(size_t sz, Args ... args)
```

создающий в памяти массив размера `sz` из элементов типа `T` с аргументами конструктора `args`.

Класс `aiw::MMapAlloc` является наследником класса `aiw::BaseAlloc`, и кроме перегрузки соответвующих методов предоставляет конструктор

```
MMapAlloc(const std::shared_ptr<FILE> &pf, size_t sz, int flags)
```

мапирующий в память (с флагами `flags`) область размера `size` байт из файла `pf` от текущей позиции в файле.

Модуль `alloc` является легким (около 60-ти строк) файлом, зависящими только от модуля `debug`. Модуль `alloc` подключает и использует следующие библиотеки:

- `aiwlib/debug` — генерация исключений;
- `<memory>` — доступ к классу `std::shared_ptr`;
- `<sys/mman.h>`, `<unistd.h>`, `<fcntl.h>` — мапирование файлов.

При кросс–компиляции под ОС `Windows` компилятором `minGW` необходимо указывать опцию `-DMINGW`, при этом библиотеки `<sys/mman.h>`, `<unistd.h>`, `<fcntl.h>` не подключаются и класс `MMapAlloc` является недоступным.

2.3 Потоки ввода/вывода — модули `iosream`, `gzstream` и `binaryio`

2.3.1 Общие замечания

При создании приложений численного моделирования потоки ввода/вывода `std::iostream` из стандартной библиотеки оказываются не всегда удобны. В частности желательно:

1. обеспечить максимально возможную производительность, особенно при бинарном вводе/выводе — в этом смысле потоки `std::iostream` сделаны не вполне оптимально;
2. иметь абстрактный базовый класс потока и его наследников для работы с обычными файлами и с файлами сжатыми библиотекой `zlib.h` — такую возможность предоставляет например библиотека `boost`, но использование `boost` только ради потоков представляется черезмерным;
3. иметь возможность мапировать файл (стандартная функция `mmap`) при помощи метода потока, с текущей позиции, указав лишь размер области и режим, и обеспечивать при этом автоматическую сборку мусора;
4. иметь возможность применять для форматированного вывода типобезопасный аналог функций `fprintf`;
5. иметь возможность формировать имя файла в аргументах конструктора при помощи типобезопасного аналога функций `fprintf`;

6. использовать перегруженные операции <> для бинарного ввода/вывода — впрочем эта возможность может быть реализована и для std::iostream.

Библиотека aiwlib предоставляет свои потоки ввода/вывода — абстрактный класс aiw::IOstream и его наследников aiw::File (модуль iostream) и aiw::GzFile (модуль gsztream). В модуле binaryio перегружены операции < и > для бинарного ввода/вывода для большинства актуальных типов.

2.3.2 Типобезопесный форматированный вывод

Модуль iostream предоставляет функцию

```
template <typename S, typename ... Args>
void aiw::format2stream(S &&str, const char *format, Args ... args);
```

обеспечивающую типобезопасный форматированный вывод в поток str согласно строке format. Аргументы args подставляются вместо символов %. Для вывода символа % необходимо использовать строку %%. Может выводится любой аргумент x для которого определен оператор форматированного вывода str<<x.

2.3.3 Абстрактный класс aiw::IOstream

Абстрактный класс aiw::IOstream определен в заголовочном файле iostream.

Класс aiw::IOstream имеет поле std::string name, содержащее имя открытого файла.

Класс предоставляет следующие методы:

- virtual ~IOstream(){} — виртуальный деструктор;
- virtual void close() = 0 — закрывает поток;
- virtual size_t tell() const = 0 — возвращает текущую позицию в потоке;
- virtual void seek(size_t offset, int whence=0) = 0 — устанавливает позицию в потоке относительно точки укзаываемой параметром whence, допустимые значения: 0 (SEEK_SET) — начало файла, 1 (SEEK_CUR) — текущая позиция, 2 (SEEK_END) — конец файла;
- virtual size_t read(void* buf, size_t size) = 0 — читает size байт в буфер buf из файла, возвращает число прочитанных байт;
- virtual size_t write(const void* buf, size_t size) = 0 — записывает в файл size байт из буфера buf, возвращает число записанных байт;
- virtual void flush() = 0 — принудительно сбрасывает содержимое буфера на диск;
- virtual std::shared_ptr<BaseAlloc> mmap(size_t size, bool write_mode=false) — мапирует из файла область размерами size (начиная с текущей позиции), возвращает proxy–объект (см. описание модуля alloc), если мапирование невозможно (например при работе со сжатым файлом) происходит копирование соответствующей области в память, при этом мапирование с доступом на запись невозможно (если аргумент write_mode=true возбуждается исключение);

- `virtual int printf(const char * format, ...) = 0` — обеспечивает **не**типобезобасный форматированный вывод при помощи функции `::fprintf(...)`;

- `template<typename...Args> IOstream& operator ()(const char *format, Args...args)` — обеспечивает типобезопасный форматированный вывод вызывая функцию `aiw::format2stream(*this, format, args...)`;

- операторы форматированного вывода < и < для встроенных типов.

2.3.4 Класс `aiw::File`

Класс `aiw::File` определен в заголовочном файле `iostream`.

Класс `aiw::File` является наследником класса `aiw::IOstream`. Кроме перегрузки необходимых виртуальных методов класса `aiw::IOstream`, класс `aiw::File` предоставляет следующие методы:

- `File(){}` — конструктор по умолчанию, создает неактивный поток;

- `template<typename...Args>`
  `void open(const char *format, const char *mode, Args...args)` — открывает файл в режиме `mode` с именем, формируемым на основе строки `format` и аргументов `args` при помощи функции `aiw::format2stream()`;

- `template<typename...Args> File(const char *format, const char *mode, Args...args)` — конструктор, открывает файл при помощи описанного выше метода `open`.

2.3.5 Класс `aiw::GzFile`

Класс `aiw::GzFile` определен в заголовочном файле `gzstream`.

Класс `aiw::GzFile` является наследником класса `aiw::IOstream`. Кроме перегрузки необходимых виртуальных методов класса `aiw::IOstream`, класс `aiw::GzFile` предоставляет следующие методы:

- `GzFile(){}` — конструктор по умолчанию, создает неактивный поток;

- `template <typename ... Args>`
  `void open(const char *format, const char *mode, Args ... args)` — открывает файл в режиме `mode` с именем, формируемым на основе строки `format` и аргументов `args` при помощи функции `aiw::format2stream()`;

- `template<typename...Args> GzFile(const char *format, const char *mode, Args...args)` — конструктор, открывает файл при помощи описанного выше метода `open`.

2.3.6 Операторы бинарного ввода/вывода — модуль `binaryio`

Модуль `binaryio` предоставляет перегруженные операции `<` (для бинарного вывода) и `>` (для бинарного ввода) в потоки `aiw::IOstream`.

Встроенные типы, объекты `std::complex<T>` и вектора `aiw::Vec` выводятся обычным копированием «байт-в-байт».

Для типов `std::vector`, `std::string`, `std::list`, `std::map` сначала записывается размер контейнера (тип `uint32_t` для строк и `uint64_t` для остальных типов), затем содержимое контейнера.

Во избежании подключения лишних модулей, операции `<` и `>` в модуле `binaryio` для **не** встроенных типов перегружаются только если в единице трансляции был подключен заголовочный файл с определением соотвествующего типа **до** заголовочного файла `binaryio`.

Например, для перегрузки операций `<` и `>` для комплексных чисел `std::complex<T>`, заголовочный файл `binaryio` должен быть включен **после** заголовочного файла `complex`.

Допускается многократное включение заголовочного файла `binaryio`, при этом можно считать, что с точки зрения перегрузки операций `<` и `>` актуальным является последнее включение.

2.3.7 Детали реализации

Модули `iostream` (порядка 100 строк), `gzstream` (40 строк) и `binaryio` (порядка 100 строк) являются довольно легкими модулями, зависящими только от модулей `debug` и `alloc`. Модуль `iostream` подключает и использует следующие библиотеки:

- `aiwlib/debug` — генерация исключений;
- `aiwlib/alloc` — доступ к объекту `MMapAlloc` при мапировании файлов;
- стандартная библиотека `<cstdio>` — работа с файлами `FILE*`;
- стандартная библиотека `<string>` — доступ к классу `std::string`.

Модуль `gzstream` подключает и использует следующие библиотеки:

- `aiwlib/iostream` — доступ к абстрактному классу `aiw::IOstream`;
- стандартная библиотека `<zlib.h>` — работа со сжатыми файлами `gzFile`.

Модуль `binaryio` подключает и использует следующие библиотеки:

- `aiwlib/iostream` — доступ к абстрактному классу `aiw::IOstream`.

2.4 Вектора и индексы — модуль `vec`

2.4.1 Общие замечания

Под вектором в декартовом пространстве размерности D понимается массив объектов (по умолчанию типа `double`) длиной D, в дальнейшем мы будем кратко обозначать их как $\mathbf{a}_D, \mathbf{b}_D, \mathbf{c}_D$ или $\mathbf{a}, \mathbf{b}, \mathbf{c}$ если размерность очевидна. Вектора реализованы в виде параметризованного класса

```
template <int D, typename T=double> class Vec;
```

в заголовочном файле <aiwlib/vec>.

Под индексом понимается массив целых чисел (тип int) длиной D, в дальнейшем мы будем кратко обозначать их как $\mathfrak{m}_D, \mathfrak{n}_D, \mathfrak{l}_D$ или $\mathfrak{m}, \mathfrak{n}, \mathfrak{l}$ если размерность очевидна. Индексы реализованы в виде alias-а

```
template<int D> using Ind = Vec<D, int>;
```

в заголовочном файле <aiwlib/vec>.

Кроме того доступен alias

```
template<int D> using Vecf = Vec<D, float>;
```

Таким образом индексы и вектора имеют практически одинаковые наборы операций, поскольку являются по сути одним и тем же параметризованным классом, и все сказанное для векторов справделиво и для индексов. Для индексов дополнительно перегружены операции

```
template<int D> inline Ind<D>& operator ++ (Ind<D> &I);
template<int D> inline bool operator ^= (Ind<D> &pos, const Ind<D> &Up);
template<int D> inline Ind<D> operator % (size_t x, const Ind<D> &p);
```

применяющиеся для индексации и обхода D–мерной прямоугольной области.

Во всех бинарных операциях, при вызове конструкторов и операций копирования одномерный вектор трактуется как вектор произвольной размерности состоящий из одинаковых компонент.

Для инстацирования C++ векторов в Python при импорте модуля vec.py специальным образом подправляется таблица типов SWIG, в результате явного инстацирования (при помощи директивы %template в .i–файле) не требуется. Модуль vec.py предоставляет класс Vec, экземпляры которого в зависимости от типа и количества компонент трактуются как экземпляры соответствующих классов Vec в C++.

2.4.2 Конструкторы и порождающие функции

Для индексов и векторов реализованы следующие конструкторы — конструктор принимающий один аргумент для инициазалиции всех компонент, по умолчанию аргумент равен нулю:

```
explicit Vec(T val=0);
```

конструктор копирования:

```
template <class T2> Vec(const Vec<D, T2> &v);
```

конструктор принимающий D аргументов для инициализации:

```
template <typename ... Args> explicit Vec(const Args&... xxx);
```

конструктор принимающий одномерный вектор (трактуется как D–мерный вектор с одинаковыми компонентами):

```
Vec(const Vec<1, T> &v);
```

Кроме того реализованы порождающие функции

```
template <typename T, typename ... Args>
       inline Vec<1+sizeof...(Args), T> vec(T x, Args ... args);
template <typename ... Args> inline Vecf<sizeof...(Args)> vecf(Args ... args);
template<typename ... Args> inline Ind<sizeof...(Args)> ind(Args ... args);
```

В Python для класса Vec реализован конструктор, принимающий произвольное число аргументов (компонент) либо кортеж, список или вектор значений компонент (объектов, допускающих приведение к типам bool/int/long/float/complex). Тип вектора по умолчанию равен double либо явно задается через именованный аргумент конструктора T. Размерность вектора определятся по числу компонент, либо явно задается через именованный аргумент конструктора D. Реализована порождающая функция vec, возвращающая вектор с размерностью равной числу аргументов и типом определяемым на основе типа первого аргумента.

В Python классы Ind и Vecf определены как наследники класса Vec; определены прождающие функции ind и vecf.

Примеры использования (распространяются так же на классы Ind и Vecf):

C++:

```
Vec<3> a;    // вектор с D=3, заполненный нулями.
Vec<5> b(1.); // вектор с D=5, заполненный единицами.
// вектора с D=7 и компонентами (1,2,3,4,5,6,7)
Vec<7> c = Vctr(1,2,3,4,5,6,7);
Vec<7> e = vec(1,2,3,4,5,6,7);
Vec<7> f(1,2,3,4,5,6,7);
```

Python:

```
a = Vec(D=3)    # вектор с D=3, заполненный нулями.
b = Vec(1, D=5)   # вектор с D=5, заполненный единицами.
# вектор с D=7 и компонентами (1,2,3,4,5,6,7)
c = Vec(1,2,3,4,5,6,7)
f = vec((1,2,3,4,5,6,7))
```

2.4.3 Арифметические операции и операции сравнения

Для всех арифметических операций, кроме операций вида OP=, тип результата (тип скаляра или тип компонент вектора) определяется на основе оператора decltype (в C++11, в Python такое поведение эмулируется). Например при сложении

```
Vec<3, int>(1,2,3)+Vec<3>(1.5, 2.5, 3.5) ==> Vec<3, double>(2.5, 4.5, 6.5)
```

Исключением из этого правила является лишь операция скалярного произведения индексов.

Все арифметические операции для векторов перегружены одинаково в C++ и Python.

Во всех операциях, где это имеет смысл, вектор единичной размерности может трактоваться как D–мерный вектор составленный из одинаковых компонент.

$\mathbf{a} = -\mathbf{b}$	$\to$	$a_i = -b_i$	$\mathbf{a} = +\mathbf{b}$	$\to$	$a_i = +b_i$
$\mathbf{a} = \mathbf{b} + \mathbf{c}$	$\to$	$a_i = b_i + c_i$	$\mathbf{b} \mathrel{+}= \mathbf{c}$	$\to$	$b_i = b_i + c_i$
$\mathbf{a} = \mathbf{b} - \mathbf{c}$	$\to$	$a_i = b_i - c_i$	$\mathbf{b} \mathrel{-}= \mathbf{c}$	$\to$	$b_i = b_i - c_i$
$\mathbf{a} = \mathbf{b} * x$	$\to$	$a_i = b_i x$	$\mathbf{b} \mathrel{*}= x$	$\to$	$b_i = b_i x$
$\mathbf{a} = x * \mathbf{c}$	$\to$	$a_i = x c_i$			
$\mathbf{a} = \mathbf{b}/x$	$\to$	$a_i = b_i/x$	$\mathbf{b} \mathrel{/}= x$	$\to$	$b_i = b_i/x$
$x = \mathbf{b} * \mathbf{c}$	$\to$	$x = \sum_i b_i c_i$			
$\mathbf{a} = \mathbf{b} \& \mathbf{c}$	$\to$	$a_i = b_i c_i$	$\mathbf{b} \mathrel{\&}= \mathbf{c}$	$\to$	$b_i = b_i c_i$
$\mathbf{a} = \mathbf{b}/\mathbf{c}$	$\to$	$a_i = b_i/c_i$	$\mathbf{b} \mathrel{/}= \mathbf{c}$	$\to$	$b_i = b_i/c_i$
$\mathbf{a} = x/\mathbf{c}$	$\to$	$a_i = x/c_i$			
$\mathbf{a} = \mathbf{b} \ll \mathbf{c}$	$\to$	$a_i = \min(b_i, c_i)$	$\mathbf{b} \mathrel{\ll}= \mathbf{c}$	$\to$	$b_i = \min(b_i, c_i)$
$\mathbf{a} = \mathbf{b} \gg \mathbf{c}$	$\to$	$a_i = \max(b_i, c_i)$	$\mathbf{b} \mathrel{\gg}= \mathbf{c}$	$\to$	$b_i = \max(b_i, c_i)$
$\mathbf{a}_3 = \mathbf{b}_3 \% \mathbf{c}_3$	$\to$	$\mathbf{a}_3 = [\mathbf{b}_3 \times \mathbf{c}_3]$	$\mathfrak{l} = k\%\mathfrak{m}$		см. текст
$x = \mathbf{b}_2 \% \mathbf{c}_2$	$\to$	$x = b_0 c_1 - b_1 c_0$	$\mathfrak{l} \mathrel{\hat{}}= \mathfrak{m}$		см. текст
			$++\mathfrak{m}$	$\to$	$++m_0$
$q = \mathbf{b} == \mathbf{c}$	$\to$	$q = (b_i = c_i \forall i)$	$q = \mathbf{b} \mathrel{!}= \mathbf{c}$	$\to$	$q = (\exists i) b_i \neq c_i$
$q = \mathbf{b} < \mathbf{c}$	$\to$	$q = (b_i < c_i \forall i)$	$q = \mathbf{b} <= \mathbf{c}$	$\to$	$q = (b_i \leq c_i \forall i)$
$q = \mathbf{b} > \mathbf{c}$	$\to$	$q = (b_i > c_i \forall i)$	$q = \mathbf{b} >= \mathbf{c}$	$\to$	$q = (b_i \geq c_i \forall i)$
$q = \texttt{bool}(\mathbf{b})$	$\to$	$q = (\exists i) b_i \neq 0$	$q = !\mathbf{b}$	$\to$	$q = (b_i = 0 \forall i)$

Таблица 2.1: Арифметические операции и операции сравнения класса `Vec<D,T>`

$x = \mathbf{b}[i]$	$\to$	$x = b_i$	$\mathbf{a}[i] = x$	$\to$	$a_i = x$
$x = (\mathbf{a}_D).\texttt{periodic}(i)$	$\to$	$x = \mathbf{a}_{(i\%D)}$	$(\mathbf{a}_D).\texttt{periodic}(i) = x$	$\to$	$\mathbf{a}_{(i\%D)} = x$
$\mathbf{a}_{D+1} = \mathbf{b}_D \vert x$	$\to$	$a_i = b_i,\ a_D = x$	$\mathbf{a}_D = \mathbf{b}_E(\mathfrak{m}_D)$	$\to$	$a_i = b_{\mathfrak{m}_i}$
$\mathbf{a}_{D+1} = x \vert \mathbf{b}_D$	$\to$	$a_0 = x,\ a_{i+1} = b_i$	$\mathbf{a}_D = \mathbf{b}_E(k_1, ...k_D)$	$\to$	$a_i = b_{k_i}$
$\mathbf{a}_{D_1+D_2} = \mathbf{b}_{D_1} \vert \mathbf{c}_{D_2}$	$\to$	$a_i = b_i,\ a_{(i+D_1)} = c_i$	$\mathbf{a} = \mathbf{b}.\texttt{circ}(k)$	$\to$	$a_i = b_{((i+k)\%D)}$
$x = \mathbf{b}.\texttt{abs}()$	$\to$	$x = \sqrt{\sum_i b_i^2}$	$\mathbf{a} = \mathbf{b}.\texttt{ceil}()$	$\to$	$a_i = ::\texttt{ceil}(b_i)$
$\mathbf{a} = \mathbf{b}.\texttt{fabs}()$	$\to$	$a_i = \lvert b_i \rvert$	$\mathbf{a} = \mathbf{b}.\texttt{floor}()$	$\to$	$a_i = ::\texttt{floor}(b_i)$
$\mathbf{a} = \mathbf{b}.\texttt{pow}(k)$	$\to$	$a_i = b_i^k$	$\mathbf{a} = \mathbf{b}.\texttt{round}()$	$\to$	$a_i = ::\texttt{round}(b_i)$
$x = \mathbf{b}.\texttt{sum}()$	$\to$	$x = \sum_i b_i$	$\mathbf{a} = \mathbf{b}.\texttt{fmod}(x)$	$\to$	$a_i = ::\texttt{fmod}(b_i, x)$
$x = \mathbf{b}.\texttt{prod}()$	$\to$	$x = \prod_i b_i$	$\mathbf{a} = \mathbf{b}.\texttt{fmod}(\mathbf{c})$	$\to$	$a_i = ::\texttt{fmod}(b_i, c_i)$
$x = \mathbf{b}.\texttt{min}()$	$\to$	$x = \min b_i$	$\mathfrak{m} = \mathbf{b}.\texttt{nan}()$	$\to$	$m_i = ::\texttt{isnan}(b_i)$
$x = \mathbf{b}.\texttt{max}()$	$\to$	$x = \max b_i$	$\mathfrak{m} = \mathbf{b}.\texttt{inf}()$	$\to$	$m_i = ::\texttt{isinf}(b_i)$
$k = \mathbf{b}.\texttt{imin}()$	$\to$	$x = \arg\min_i b_i$	$q = \mathbf{b}.\texttt{cknan}()$	$\to$	$q = (\exists i)::\texttt{isnan}(b_i)$
$k = \mathbf{b}.\texttt{imax}()$	$\to$	$x = \arg\max_i b_i$	$q = \mathbf{b}.\texttt{ckinf}()$	$\to$	$q = (\exists i)::\texttt{isinf}(b_i)$

Таблица 2.2: Операции и методы для доступа к элементу, изменения размерности и различные преобразования класса `Vec<D,T>`

Традиционно для векторов перегружены операции `+, -, +=, -=`, операции умножения на скаляр ($\mathbf{a} * x$, $x * \mathbf{a}$ и $\mathbf{a} *= x$), операции деления вектора на скаляр ($\mathbf{a}/x$ и $\mathbf{a}/= x$), операция доступа к элементу `[]`.

Операция умножения `*` вектора на вектор перегружена как скалярное произведение векторов[1]. Для индексов операция скалярного умножения возвращает тип `int64_t` во избежании переполнения `int32_t` при индексации многомерных массивов.

Операции $\mathbf{a}/\mathbf{b}$, $\mathbf{a}/= \mathbf{b}$, $\mathbf{a}\&\mathbf{b}$ и $\mathbf{a}\&= \mathbf{b}$ перегружены как покомпонетное деление и умножение векторов.

Перегружены операторы побитового сдвига $\ll, \gg, \ll=, \gg=$. Выражение $c = a \ll b$ интерпретируется как покомпонетный результат вычисления выражения $c_i = \min(a_i, b_i)$. Такие операторы позволяют эффективно определять границы области значений множества векторов.

Сравнение векторов проводится нетрадиционным образом. Перегружены операторы `<, >, <=, >=, ==, !=`. Считается, что $\mathbf{a} < \mathbf{b}$ при $a_i < b_i \quad \forall i$ и $\mathbf{a} \le \mathbf{b}$ при $a_i \le b_i \quad \forall i$. Аналогично, $\mathbf{a} > \mathbf{b}$ при $a_i > b_i \quad \forall i$ и $\mathbf{a} \ge \mathbf{b}$ при $a_i \ge b_i \quad \forall i$. Такие операции сравнения позволяют эффективно задавать прямоугольные области как множество индексов/векторов $\mathbf{x}$ удовлетворяющих условию $\mathbf{a} < \mathbf{x} < \mathbf{b}$.

Для использования векторов в качестве ключей контейнера `std::map` библиотеки `STL` специализирована структура `std::less`, сравнивающая вектора в обратном лексикографическом порядке. Во избежании влияния ошибок округления, структура `std::less` для векторов использует структуры `std::less` специализированные для типа компонент вектора. В частности, для `double` и `float` в модуле `vec` специализированны структуры `std::less`, не учитывающие при сравнении последний байт (для `float`) и последние два байта (для `double`) мантиссы.

Для использования векторов в качестве ключей словаря `Python` реализован метод `__hash__`, возвращающий `hash`–значение кортежа, составленного из компонент вектора.

Реализован оператор приведения вектора к типу `bool` (возвращает истину если хотя бы одна компонента не равна нулю) и оператор отрицания `!` (возвращает истину если все компоненты равны нулю).

Для двумерных и трехмерных векторов перегружена операция `%` как операция векторного умножения.

Для индексов перегружена операция

```
Ind<D> operator % (size_t x, const Ind<D> &m);
```

возвращающая индекс — позицию i–го (по счету) элемента в D–мерной области размера $\mathtt{m}$, первая компонента считается самой быстрой осью. Операция может использоваться для организации обхода D–мерной области в одном цикле:

```
Ind<D> m = ...;
size_t sz = m.prod();
for(size_t i=0; i<sz; ++i){
    Ind<D> pos = i%m;
    ...
}
```

[1] В отличии от системы `Matlab` транспонирования вектора при этом не требуется

Операция относительно дорогая и такой вариант не очень эффективен, но зато цикл может быть легко распараллелен средствами библиотеки OpenMP.

Для эффективного обхода D–мерных областей в C++ у индексов перегружены операции префиксного инкремента и $\hat{}=$:

```
Ind<D>& operator ++ (Ind<D> &l);
bool operator ^= (Ind<D> &l, const Ind<D> &m);
```

Операция инкремента всегда увеличивает нулевую компоненту индекса. Операция $\mathfrak{l}\hat{}=\mathfrak{m}$ проверяет, не вышла ли нулевая компонента индекса $\mathfrak{l}$ за размеры области $\mathfrak{m}$ (правая граница не включается) — при необходимости компонента обнуляется, следующая компонента инкрементируется и проверяется ее выход за пределы области. Оператор возвращает истину, если проверенный (и измененный при необходимости) индекс находится внутри области, и ложь если индекс вышел за пределы области (стал равен $\mathfrak{m}$). Обход области выглядит как

```
Ind<D> m = ...;
for(Ind<D> l=0; l^=m; ++l){ ... }
```

Такой обход куда эффективнее, но не может быть легко распараллелен средствами библиотеки OpenMP.

Арифметические операции и операции сравнения приведены в таблице 2.1.

2.4.4 Операции и методы для доступа к элементу, изменения размерности и различные преобразования

Для доступа к элементу традиционно перегружена операция [], в C++ при включенном режиме отладки (опция debug=on при вызове make либо опция -DEBUG компилятора) проверяется корректность номера компоненты вектора. В Python для операции [] традиционно реализовано взятие среза и адресация с конца при отрицательном значении аргумента. Кроме того, в C++ реализован метод

```
T  periodic(int i) const;
T& periodic(int i);
```

использующий положительный отстаток от деления i на размерность вектора.

Метод

```
Vec circ(int l) const;
```

возвращает вектор с циклически переставленными на l позиций компонентами.

Операция | (побитовое или) обеспечивает «склейку» числа и вектора, вектора и числа или двух векторов.

Операция () принимает номера компонент вектора (произвольное количество аргументов) либо индекс произвольной длины с номерами компонент вектора (в Python так же список и кортеж), и возвращает вектор составленный из указанных компонент.

Метод abs() возвращает модуль (длину) вектора как корень из суммы квадратов компонент.

Метод `pow(n)` возвращает вектор покомпонентно возведенный в степень n. Предоставляется эффективная реализация для целочисленных степеней, как положительных так и отрицательных.

Методы `fabs()`, `ceil()`, `floor()`, `round()` возвращают вектор с результатами покомпонетного применения соответствующих функций библиотеки `math.h`.

Метод `fmod(y)` принимает скаляр либо вектор и возвращает вектор с результатами покомпонетного применения функции `::fmod(x, y)` библиотеки `math.h`, тип результирующего вектора опередеяется на основе оператора `decltype` (в `C++11`, в `Python` такое поведение эмулируется).

Методы `sum()` и `prod()` возвращают скаляр — сумму и произведение компонент вектора. На случай расчета размера больших D–мерных прямоугольных областей, во избежании переполнения `int32_t`, предоставляется метод

```
template <typename T2> inline void prod(T2 &res) const;
```

Методы `min()` и `max()` возвращают значение минимальной и максимальной компоненты вектора. Методы `imin()` и `imax()` возвращают номер минимальной и максимальной компоненты вектора (первой по счету из минимальной/максимальной, если есть компоненты с одинаковыми значениями).

Методы `nan()` и `inf()` возвращают индекс, содержащий результаты проверки компонент вектора при помощи функций `::isnan()` и `::isinf()` библиотеки `math.h`. Методы `cknan()` и `ckinf()` возвращают истину, если хотя бы одна из компонент вектора содержит значение `NAN` или `INF` (проверяется функциями `::isnan()` и `::isinf()` библиотеки `math.h`).

Операции и методы для доступа к элементу, изменения размерности и различные преобразования приведены в таблице 2.2.

2.4.5 Другие операции и методы

Для сериализации векторов в `Python` перегружены специальные методы `__get/setstate__`.

Для векторов перегружены операции форматированного ввода/вывода в потоки `std::iostream` и операция форматированного вывода в потоки `aiw::IOstream`.

Для векторов перегружены операции бинарного ввода/вывода `<>` в потоки `aiw::IOstream`.

2.4.6 Детали реализации

При написании модуля `vec` основной проблемой являлась необходимость инстацирования шаблона `Vec` в `Python` при помощи `SWIG`. Ситуация усугублялась тем, что модуль `vec` написан с широким использованием возможностей `C++11` (шаблонов с переменным числом аргументов, оператора `decltype`, принципа `SFINAE`) — в момент написания модуля утилита `SWIG` эти возможности не поддерживала. Кроме того, сама необходимость инстацирования большого числа шаблонов `Vec` с разными наборами параметров существенно усложняла сборку и эксплуатацию библиотеки.

В итоге было решено отказаться от прямого инстацирования шаблонов `Vec` при помощи директивы `%template` утилиты `SWIG`. Вместо этого на `Python` был написан отдельный модуль `vec.py`, использующий служебный класс `PVec` и несколько функций из заголовочного файла `aiwlib/swig`. При импорте модуль `vec.py` анализирует таблицу типов `SWIG` и устанавливает

дополнительные связи между всеми использованиями шаблонов `Vec` в импортируемом `C++`–коде и служебным классом `PVec`. В итоге, поведение шаблонов класса `Vec` полностью эмулируется в `Python`, единственным ограничением является размер вектора в памяти, ограниченный размером памяти выделяемой под класс `PVec` — в настоящий момент он не должен превышать 1024 байта.

Для корректной работы с `C++` методами имеющими аргументы (возвращающими значения) и переменными типа `Vec`, достаточно проимпортировать модуль `aiwlib.vec` как

```
import aiwlib.vec
```

или

```
from aiwlib.vec import *
```

Для удобства работы рекомендуется второй вариант, хотя и в первом варианте переменные и возвращаемые значения типа `Vec` оказываются полностью работоспособными в `Python`.

Модуль `vec` подключает и использует следующие библиотеки:

- `<math.h>` — стандартные математические функции;
- `"aiwlib/debug"` — средства отладки (проверка диапазона номеров компонент).

2.5 Равномерные многомерные прямоугольные (картезианские) сетки — модуль `mesh`

2.5.1 Общие замечания

Равномерные прямоугольные сетки реализованы в заголовочном файле `<aiwlib/mesh>` в виде параметризованного класса `Mesh<`T`,`D`>`, где T — тип ячейки массива, D — размерность массива.

Многомерные сетки поддерживают настройку осей координат — для каждой оси могут быть заданы пределы, шаг и опционально логарифмический масштаб. Поддерживается обращение к ячейкам сетки как по индексу (номеру по всем осям) так и по координате, которая пересчитывается в индекс на основе настроек осей. Кроме того, возможно использование интерполяции различных порядков, задание периодических граничных условий, продолжение сетки за область ее определения на основе граничных значений.

Многомерная сетка (массив) эмулируется при помощи одномерного массива, смещение в котором пересчитывается с учетом размеров многомерной области. Предоставляются средства для организации эффективного обхода содержимого сетки с учетом локальности данных.

Многомерные сетки `aiw::Mesh` обеспечивают упорядоченный доступ к некоторому участку памяти. Возможно создание сеток с другими размерами, обеспечивающих доступ к тому же участку. В частности, конструкторы копирования сеток не выделяют новых участков памяти под данные — копии ссылаются на тот же участок. Сборка мусора призводится на основе подсчета ссылок. С одной стороны это существенно ускоряет копирование объектов, с другой стороны копии не являются независимыми, т.е. изменение данных в одной копии влечет за

собой изменение всех остальных копий. Для полноценного копирования с выделением нового участка памяти под данные используется метод `copy()`.

Многомерные сетки допускают проведение ряда преобразований — разворот и изменение порядка нумерации осей координат, вырезание подобластей, построение срезов и т.д. При этом не происходит копирование данных исходной сетки, а лишь предоставляется альтернативный способ доступа к исходным данным, что открывает широкие возможности для манипуляций с данными.

Многомерные сетки обеспечивают запись и чтение данных на диск в бинарном формате (сейчас используется старый формат библиотеки `aivlib`) и форматированный вывод данных в текстовом виде для `gnuplot`.

2.5.2 Поля и методы для получения информации о сетке

Класс `Mesh<`T`,`D`>` содежит следующие открытые поля:

```
std::string head;  // произвольный текстовый заголовок
T out_value;       // значение (ячейка) за пределами сетки
aiw::Vec<D> bmin;  // координаты левого нижнего угла области
aiw::Vec<D> bmax;  // координаты правого верхнего угла области
aiw::Vec<D> step;  // размер ячейки сетки
aiw::Vec<D> rstep; // обратный размер ячейки
int logscale;      // битовая маска отмечающая логарифмические масштабы осей
```

Класс `Mesh<`T`,`D`>` предоставляет следующие методы для получения информации о состоянии сетки

```
size_t size() const;       // общее число элементов сетки
aiw::Ind<D> bbox() const;  // размеры сетки по всем осям
size_t mem_size() const;   // размер области памяти в ячейках
size_t mem_sizeof() const; // размер ячейки в байтах
```

Методы `mem_size()` и `mem_sizeof()` выдают информацию об области памяти сетки без учета проведенных преобразований.

2.5.3 Инициализация сетки и настройка осей

Для настройки осей сетки предназначен метод

```
void set_axes(const aiw::Vec<D> &bmin, const aiw::Vec<D> &bmax, int logscale=0);
```

Метод настраивает оси на основе текущих размеров сетки в ячейках.

Для инициализации сетки (выделения памяти) служат методы

```
void init(const aiw::Ind<D> &box);
void init(const aiw::Ind<D> &box,
          const aiw::Vec<D> &bmin, const aiw::Vec<D> &bmax, int logscale=0);
```

Вторая версия метода `init` производит настройку осей после выделения памяти. Первая версия метода `init` настраивает оси по умолчанию — размеры области от нуля до `box` (шаг равен еденице), логарифмического масштаба нет.

Каждый вызов метода `init` приводит к выделению новой области памяти под данные, однако старая область памяти может оказаться используемой другой сеткой и не обязательно будет освобождена, см. раздел 2.5.6

Все настройки осей хранятся в открытых полях `bmin`, `bmax`, `step`, `rstep` и `logscale`. Несогласованное изменение этих полей может привести к некорректному преобразованию координат точки в индекс ячейки сетки.

Кроме того, для настройки граничных условий и интерполяции используются поля

```
int periodic; // битовая маска, задающая периодические граничные условия для осей
Ind<2> bound_min, bound_max;  // битовые маски, задающие обработку границ слева/справа
                      // 0   - ничего не делать (при выходе за границу выкидывает искл
                      // 1,0 - возвращает out_value
                      // 1,1 - возвращает крайнее значение
Ind<3> Itype; // битовые маски, задающие типы интерполяции по осям:
              // 0 - без интерполяции, 1,0 - линейная, 1,1,0 - кубическая, 1,1,1 - B-c
```

2.5.4 Доступ к ячейкам

Сетки обеспечивают доступ к ячейкам по координате (вектору) или индексу (набору номеров ячейки по всем осям).

Базовыми являются методы

```
inline aiw::Ind<D> coord2pos(const aiw::Vec<D> &r) const;
inline double pos2coord(int pos, int axe) const;
inline aiw::Vec<D> pos2coord(const aiw::Ind<D> &p) const;
inline aiw::Vec<D> cell_angle(const aiw::Ind<D> &p, bool up) const;
```

пересчитывающие координаты в индексы и обратно согласно настройкам осей. Метод `pos2coord` возвращает координаты центра ячейки. Метод `cell_angle` в зависимости от аргумента `up` возвращает координаты левого нижнего или правого верхнего угла ячейки.

Для доступа к ячейкам служат методы

```
inline const T& get(const aiw::Ind<D> &pos) const;
inline const T& get(const aiw::Vec<D> &r) const;
```

В зависимости от значений полей `periodic`, `bound_min` и `bound_max` при промахе (выходе за границы сетки) реализуются периодические граничные условия (если поднят соответствующий бит маски `periodic`), выкидывается исключение (если не поднят соответствующий бит в маске `bound_min/max[0]`), обеспечивается доступ к полю `out_value` (если поднят соответствующий бит в маске `bound_min/max[0]`) либо к граничной ячейке (если поднят соответствующий бит в маске `bound_min/max[1]`). В настоящий момент функция `get` обеспечивает довольно гибкое управление поведением сетки при промахах, но при этом проводиться довольно много проверок.

Метод доступа по вектору требует дополнительных вычислений для перевода вектора в индекс ячейки. При промахе (если вектор попадает за пределы обоасти сетки) обеспечивается доступ к открытому полю сетки `out_value`.

Для традиционного доступа в `C++` перегружены операции

```
inline const T& operator [] (const aiw::Ind<D> &p) const;
inline       T& operator [] (const aiw::Ind<D> &p);
inline const T& operator [] (const aiw::Vec<D> &r) const;
inline       T& operator [] (const aiw::Vec<D> &r);
```

вызывающие функции `get`, те же операторы перегружены в `Python` как

```
inline const T& __getitem__(const aiw::Ind<D> &p) const;
inline const T& __getitem__(const aiw::Vec<D> &r) const;
inline void __setitem__(const aiw::Ind<D> &p, const T& v);
inline void __setitem__(const aiw::Vec<D> &r, const T& v);
```

Для реализации доступа с периодическими граничными условиями предназначены методы[2]

```
template<int P> inline const T& periodic_bc(Ind<D> pos) const;
template<int P> inline T& periodic_bc(Ind<D> pos);
```

где P — битовая маска, указывающая по каким осям необходимо создать периодичность. Например $P = 5$ задаст периодические граничные условия по осям x и z. Методы `periodic_bc` корректируют компоненты индекса по тем осям, для которых указаны периодические граничные условия, и затем вызывают метод `get`.

Для интерполяции перегружена операция

```
inline T operator () (const aiw::Vec<D> &r) const;
```

тип интерполяции задается при помощи битовых масок `Ind<3> Itype`, при этом используются функции из модуля `interpolations` (см. 2.6). Соответствующие некоторой оси `axe` биты в `Itype` означают: `Itype[0]&(1<<axe)==0` — интерполяция нулевого порядка (кусочно–постоянная в рамках ячейки), `Itype[0]&(1<<axe)==1`, `Itype[1]&(1<<axe)==0` — линейная интерполяция между центрами ячеек, `Itype[0]&(1<<axe)==1`, `Itype[1]&(1<<axe)==1`, `Itype[2]&(1<<axe)` локальный кубический сплайн, `Itype[0]&(1<<axe)==1`, `Itype[1]&(1<<axe)==1`, `Itype[2]&(1<<axe)==1` - кубический B–сплайн.

При интерполяции на границе сетки важны настройки `periodic` и `bound_min/max`. Функции из модуля `interpolations` не пороверяют выход за границы, поэтому при `bound_min/max[0]&(1<<axe)==` возможна генерация исключения.

2.5.5 Обход сетки

Для оптимального обхода сетки предназначены методы

```
inline aiw::Ind<D> inbox(size_t offset) const;
inline aiw::Ind<D> first() const;
inline bool next(aiw::Ind<D> &pos) const;
```

[2]Пока оставлено для обратной совместимости

Метод `inbox` преобразует номер элемента сетки (от начала области памяти) в его индекс. Для непреобразованной сетки его результат будет совпадать с результатами операции `offset%bbox()`, однако для преобразованной сетки это может быть неверно. Метод `inbox` является относительно дорогостоящим, но обеспечивает оптимальный (с точки зрения локальности данных) порядок обхода сетки и позволяет легко распараллеливать циклы обходы средствами библиотеки `OpenMP`

```
    Mesh<T,D> M;
    ...
    size_t sz = M.size();
#pragma omp parallel for
    for(size_t i=0; i<sz; ++i){
        Ind<D> pos = M.inbox(i);
        ...
    }
```

Аналогичный порядок обхода (с меньшими накладными расходами, но без такого простого распараллеивания) можно получить при помощи конструкции

```
    Ind<D> pos=M.first();
    do{...} while(M.next(pos));
```

Для непреобразованных сеток этот обход экивалентен конструкции

```
    for(Ind<D> pos; pos^=M.bbox(); ++pos){...}
```

но после преобразований такой вариант может оказаться неэффективным.

2.5.6 Преобразования сеток

Для преобразования сеток служат методы

```
    Mesh flip(int a, bool axe=true) const;
    Mesh transpose(int a, int b) const;
    Mesh crop(aiw::Ind<D> l, aiw::Ind<D> m, aiw::Ind<D> n=Ind<D>(1)) const;
    template <class T2, int D2>
        Mesh<T2, D2> slice(Ind<D> pos, size_t offset_in_cell) const;
```

Все эти методы не приводят к выделению новых областей памяти для данных сетки, а лишь создают альтернативные способы доступа к уже выделенной памяти в исходной сетке. После создания преобразованной сетки исходная сетка может быть удалена/перениициализирована, однако освобождение памяти произойдет лишь после уничтожения/перениициализации всех преобразованных сеток. Сборка мусора осуществляется при помощи подсчета ссылок на основе указателя `std::shared_ptr<BaseAlloc>`, исходная сетка и построенные на ее основе преобразованные сетки являются равноправными владельцами выделенной под данные памяти.

Метод `flip(int a, bool axe=true)` разворачивает (отражает) ось a, параметр `axe` указывает следует ли преобразовать так же настройки оси (пределы и шаг).

Метод `transpose(int a, int b)` меняет оси a и b местами, при этом преобразуются так же настройки осей.

Метод `crop(aiw::Ind<D> l, aiw::Ind<D> m, aiw::Ind<D> n=Ind<D>(1))` вырезает фрагмент сетки с левым нижним углом в ячейке $\mathfrak{l}$, правым верхним углом в ячейке $\mathfrak{m}$, правая верхняя граница не включается. При задании $\mathfrak{l}$ и $\mathfrak{m}$ допускается использовать отрицательные значения, которые отсчистываются от верхней границы (размера сетки) по соответствующей оси. Необязательный параметр $\mathfrak{n}$ позволяет задать шаг, т.е. использовать каждую $\mathfrak{n}$–ю ячейку внутри указанной области.

Метод `slice<D2, T2>(Ind<D> pos, size_t offset_in_cell)` позволяет строить срезы — уменьшать размерность сетки и изменять тип хранимых данных, например составляя новую сетку из отдельных полей структуры хранящейся в исходной сетке.

Для уменьшения размерности необходимо указать в аргументе `pos` значения -1 по тем осям, которые должны войти в срез (ровно D_2 штук), и положение среза по остальным осям.

Для измения типа данных необходимо указать новый тип и сдвиг данных внутри исходной структуры в байтах. Следует с осторожностью использовать этот вариант вызова метода `slice`, поскольку неверно вычисленное смещение может привести к непредсказуемым результатам. Допустим есть стуктура

```
struct Cell{ double a, b; int c[10];};
```

для которой создана сетка `Mesh<Cell,2> A;` Тогда, вызов

```
Mesh<int,1> B = A.slice<1, int>(Indx(10,-1), 2*8+2*4);
```

создаст срез `B` в виде одномерного массива развернутого по оси y, проходящего через десятые ячейки по оси x, и обеспечивающий доступ к полям `c[2]` соответствующих ячеек.

В настоящий момент метод `slice` доступен из `Python`-а как семейство методов

```
sliceX(self, pos, offset=0)
```

где `X` пробегает значения от `1` до `D-1`. Таким образом в `Python` возможно построение срезов, но невозможно изменение типа ячейки[3].

2.5.7 Сохранение и загрузка сеток

Для сохранения и загрузки содержимого сеток в бинарном формате предназначены методы

```
void dump(aiw::IOstream &&S) const;
void load(aiw::IOstream &&S, int use_mmap=0);
void dump(aiw::IOstream &S) const;
void load(aiw::IOstream &S, int use_mmap=0);
```

В настоящий момент используется старый формат библиотеки `aivlib`. Необязательны параметр `use_mmap` указывает на использование мапирования файла, 0 — не использовать мапирование, 1 — мапировать файл только на чтение, 2 — мапировать файл на чтение и запись.

Кроме того перегружены операции `<>` для бинарного ввода/вывода

[3]Поскольку изменение типа ячейки невозможно, изначальный смысл аргумента `offset` в `Python` теряется, однако возможны экзотические ситуации когда этот аргумент все же будет востребован

```
IOstream& operator < (IOstream &S, const Mesh<T, D> &M);
IOstream& operator > (IOstream &S,       Mesh<T, D> &M);
```

Для форматированного вывода (`.dat`–файлы для `gnuplot`) предназначен метод

```
template <typename S>
void out2dat(S &&str, Ind<D> coords=Ind<D>(2), const char* prefix=nullptr) const;
```

и его оболочки для конкретных видов потоков для инстацирования в `Python`

```
void out2dat(std::ostream &str=std::cout, aiw::Ind<D> coords=aiw::Ind<D>(2),
             const char* prefix=nullptr) const;
void out2dat(aiw::IOstream &str, aiw::Ind<D> coords=aiw::Ind<D>(2),
             const char* prefix=nullptr) const;
```

Аргумент `coords` содержит режимы вывода для каждой из координатных осей, возможны следующие режимы:

0	—	выводить значения из сетки вдоль оси в одну строку через пробел;
1	—	выводить номер ячейки;
2	—	выводить координату центра ячейки;
3	—	не выводить ни номер ячейки ни координату центра ячейки;
+4	—	не выводить пустую строку при изменении номера ячейки, не влияет на режим 0.

Аргумент `prefix` задает префикс перед каждой (не пустой) строкой при выводе. Для загрузки сеток из `.dat`–файлов используется `C++` модуль `dat2mesh` (см. раздел 2.7).

Для сериализации сеток при помощи модуля `pickle` в `Python` реализованы методы

```
std::string __getstate__() const;
void __C_setstate__(const std::string &state);
```

2.5.8 Другие методы

Для получения копии сетки (с отдельной областью памяти) предназначен метод

```
Mesh copy() const;
```

Новая сетка является упорядоченной, в ее память переносятся лишь те данные, к которым обеспечивала доступ исходная сетка.

Для заполнения сетки предназначены методы

```
void fill(const T &x);
template <typename T2> void fill(const Mesh<T2, D> &M);
void fill(const Mesh &M);
void fill(aiw::IOstream &&S);
void fill(aiw::IOstream &S);
```

Метод `fill(const T &x)` заполняет все яячейки значением x.

Метод `fill(const Mesh<T2, D> &M)` копирует в сетку содержимое сетки M, при этом должен существовать оператор приведения типа T_2 к T. Если размеры (в ячейках) заполняемой сетки и сетки M не совпадают, копируются данные лишь из области пересечения сеток.

Метод `fill(aiw::IOstream)` загружает сетку из потока (при этом предполагается что тип данных совпадает с заполняемой сеткой), и вызывает метод `fill(const Mesh &M)`.

Метод `min_max(T &a, T &b, aiw::Ind<D> &pos_a, aiw::Ind<D> &pos_b)` находит минимальное a и максимальное b значения в ячейках сетки, а так же их индексы.

2.5.9 Инстацирование в Python

Для каждого набора параметров шаблон сетки должен быть инстацирован в питон при помомщи утилиты `make`. Для этого в директории библиотеки `aiwlib` надо набрать команду

```
make MeshXXX-T-D
```

где `MeshXXX` — имя инстацируемого шаблона в `Python` (оно же имя модуля содержащего инстацированный шаблон), `T` — тип данных ячейки в `C++`, `D` — размерность. Например команда

```
make MeshF3-float-3
```

создаст модуль `MeshF3` содержащий класс `MeshF3` отвечающий шаблону `Mesh<float,3>`.

2.6 Различные варинаты интерполяции — модуль `inteporlations`

2.6.1 Локальный кубический сплайн

Локальный кубический сплайн строится по четырем отсчетам интерполируемой функции $f_{-1,0,1,2}$ как

$$\widetilde{f}(x) = \sum_{i=-1}^{2} W_i(x) f_i = \sum_{j=0}^{3} S_j x^j, \qquad x \in [0,1].$$

При этом должны выполняться следующие условия

$$\widetilde{f}(0) = f_0, \qquad \widetilde{f}(1) = f_1, \qquad \widetilde{f}'(0) = \frac{f_1 - f_{-1}}{2}, \qquad \widetilde{f}'(1) = \frac{f_2 - f_0}{2},$$

что дает в итоге СЛАУ на коэффициенты S_i:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{pmatrix} \cdot \mathbf{S} = \begin{pmatrix} f_0 \\ f_1 \\ \dfrac{f_1 - f_{-1}}{2} \\ \dfrac{f_2 - f_0}{2} \end{pmatrix}, \qquad \mathbf{S} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} f_0 \\ f_1 \\ \dfrac{f_1 - f_{-1}}{2} \\ \dfrac{f_2 - f_0}{2} \end{pmatrix},$$

откуда

$$\widetilde{f} = f_0 + \frac{f_1 - f_{-1}}{2} x + \left[-3f_0 + 3f_1 + f_{-1} - f_1 + \frac{f_0 - f_2}{2} \right] x^2 + \left[2f_0 - 2f_1 + \frac{f_1 - f_{-1} + f_2 - f_0}{2} \right] x^3 =$$
$$= \left(-\frac{x}{2} + x^2 - \frac{x^3}{2} \right) f_{-1} + \left(1 - \frac{5}{2}x^2 + \frac{3}{2}x^3 \right) f_0 + \left(\frac{x}{2} + 2x^2 - \frac{3}{2}x^3 \right) f_1 + \left(-\frac{x^2}{2} + \frac{x^3}{2} \right) f_2.$$

Веса при $f_{-1\ldots2}$ рассчитываются при помощи функции `C++` модуля `interpolations`

```
inline Vec<4> interpolate_cube_weights(double x);
```

2.6.2 Кубический B–сплайн

Фрагмент кубического B-сплайна на участке $[x_j, x_{j+1}]$ определяется как

$$\widetilde{f}(x) = \sum_{i=j-1}^{j+2} f_i N_{i,4}(x), \qquad x \in [0,1],$$

где[4] на равномерной сетке по x, при $x_{i+1} - x_i = 1 \forall i$

$$N_{i,k}(x) = \frac{\left(x - x_{i-\frac{k}{2}}\right) N_{i-\frac{1}{2},k-1}(x) + \left(x_{i+\frac{k}{2}} - x\right) N_{i+\frac{1}{2},k-1}(x)}{k-1} =$$
$$= \frac{\chi_{-k} N_{i-\frac{1}{2},k-1} - \chi_k N_{i+\frac{1}{2},k-1}(x)}{k-1}, \qquad N_{i,1}(x) \equiv \Pi_i = \begin{cases} 1, & x_{i-\frac{1}{2}} \le x < x_{i+\frac{1}{2}}, \\ 0, & \text{else}, \end{cases}$$

где $\chi_k = x - x_{i+\frac{k}{2}}$. Тогда, с учетом того что $x - \left(x_{i+\frac{k}{2}} + \Delta\right) = \chi_{k+2\Delta}$, получаем

$$N_{i,2}(x) = \chi_{-2}\Pi_{i-\frac{1}{2}} - \chi_2 \Pi_{i+\frac{1}{2}},$$

$$N_{i,3}(x) = \frac{\chi_{-3} N_{i-\frac{1}{2},2} - \chi_3 N_{i+\frac{1}{2},2}}{2} = \frac{1}{2}\left\{\chi_{-3}\left[\chi_{-3}\Pi_{i-1} - \chi_1 \Pi_i\right] - \chi_3\left[\chi_{-1}\Pi_i - \chi_3 \Pi_{i+1}\right]\right\} =$$
$$= \frac{1}{2}\left\{\chi_{-3}^2 \Pi_{i-1} - \left[\chi_{-3}\chi_1 + \chi_{-1}\chi_3\right]\Pi_i + \chi_3^2 \Pi_{i+1}\right\},$$

$$N_{i,4}(x) = \frac{\chi_{-4} N_{i-\frac{1}{2},3} - \chi_4 N_{i+\frac{1}{2},3}}{3} = \frac{1}{6}\left\{\chi_{-4}^3 \Pi_{i-\frac{3}{2}} - \chi_{-4}\left[\chi_{-4}\chi_0 + \chi_{-2}\chi_2\right]\Pi_{i-\frac{1}{2}} + \chi_{-4}\chi_2^2 \Pi_{i+\frac{1}{2}} -\right.$$
$$\left. - \chi_{-2}^2 \chi_4 \Pi_{i-\frac{1}{2}} + \left[\chi_{-2}\chi_2 + \chi_0\chi_4\right]\chi_4 \Pi_{i+\frac{1}{2}} - \chi_4^3 \Pi_{i+\frac{3}{2}}\right\} =$$
$$= \frac{1}{6}\left\{\chi_{-4}^3 \Pi_{i-\frac{3}{2}} - \left[\chi_{-4}^2\chi_0 + \chi_{-4}\chi_{-2}\chi_2 + \chi_{-2}^2\chi_4\right]\Pi_{i-\frac{1}{2}} + \left[\chi_{-4}\chi_2^2 + \chi_{-2}\chi_2\chi_4 + \chi_0\chi_4^2\right]\Pi_{i+\frac{1}{2}} - \chi_4^3 \Pi_{i+\frac{3}{2}}\right\}.$$

В итоге получаем

$$\widetilde{f} = \frac{1}{6}\left\{ -\chi_2^3 f_{-1} + \left[\chi_{-4}\chi_2^2 + \chi_{-2}\chi_2\chi_4 + \chi_0\chi_4^2\right] f_0 - \left[\chi_{-2}^2\chi_2 + \chi_{-2}\chi_0\chi_4 + \chi_0^2\chi_6\right] f_1 + \chi_0^3 f_2\right\},$$

или после преобразований в `maxima`:

$\chi_{-4}\chi_2^2 + \chi_{-2}\chi_2\chi_4 + \chi_0\chi_4^2 \to$

```
(%i1) expand((x+2)*(x-1)^2 + (x+1)*(x-1)*(x-2)+x*(x-2)^2);
                                    3       2
(%o1)                             3 x  - 6 x  + 4
```

[4] почему то во всех источниках выражения для $N_{i,k}$ сдвинуты вперед на единицу, что приводит к ассиметрии. На самом деле центр функции $N_{i,k}$ должен совпадать с точкой x_i?

$\chi_{-2}^2\chi_2 + \chi_{-2}\chi_0\chi_4 + \chi_0^2\chi_6 \rightarrow$

```
(%i2) expand((x+1)^2*(x-1)+(x+1)*x*(x-2)+x^2*(x-3));
                                3      2
(%o2)                         3 x  - 3 x  - 3 x - 1
```

$$\widetilde{f} = \frac{1}{6}\left\{\left[-x^3+3x^2-3x+1\right]f_{-1} + \left[3x^3-6x^2+4\right]f_0 + \left[-3x^3+3x^2+3x+1\right]f_1 + x^3 f_2\right\}.$$

Веса при $f_{-1\ldots2}$ рассчитываются при помощи функции C++ модуля `interpolations`

```
inline Vec<4> interpolate_Bspline_weights(double x);
```

2.6.3 Функции модуля interpolations

Для проведения интерполяции используется параметризованная функция (frontend)

```
template<typename C>
typename C::cell_type interpolate(const C& arr,            // контейнер
                                  const Ind<C::dim> &pos,  // позиция ячейки
                                  const Vec<C::dim> &x,    // координаты в ячейке
                                  const Ind<3> &Itype);    // тип интерполяции
```

где `Itype` — битовые маски. Соответствующие некоторой оси `axe` биты в `Itype` означают: `Itype[0]&(1<<axe)==0` — интерполяция нулевого порядка (кусочно–постоянная в рамках ячейки), `Itype[0]&(1<<axe)==1`, `Itype[1]&(1<<axe)==0` — линейная интерполяция между центрами ячеек, `Itype[0]&(1<<axe)==1`, `Itype[1]&(1<<axe)==1`, `Itype[2]&(1<<axe)==0` — локальный кубический сплайн, `Itype[0]&(1<<axe)==1`, `Itype[1]&(1<<axe)==1`, `Itype[2]&(1<<axe)==1` — кубический B–сплайн.

2.7 Загрузка сеток из .dat–файлов — модуль dat2mesh

Модуль предназанчен для загрузки сеток `Mesh<T, D>` из `.dat`–файлов, содержащих данные в текстовом формате в несколько колонок. Строки, начинающиейся с символа `#` и пустые строки игнорируются.

Возможно два режима работы — заполнение готовой сетки (с известными размерами, пределами и шагами по осям и т.д. и т.п.) или создание новой сетки (при этом шаги и пределы выбираются автоматичски, необходимо лишь указать по каким осям используется логарифмический масштаб).

Для заполнения готовых сеток используются фунцкции

```
template <typename T, int D, typename S>
void dat2Mesh(S&& str, Mesh<T, D> &dst, int vcol=0, Ind<D> rcols=Ind<D>());
template <typename T, int D, typename S>
void dat2Mesh(S&& str, Mesh<T, D> &dst, std::vector<int> vcols={}, int vaxe=-1,
              Ind<D> rcols=Ind<D>());
```

и их оболочки для конкретных типов потока ввода–вывода

```
template <typename T, int D>
void dat2Mesh(std::istream& str, aiw::Mesh<T, D> &dst,
              int vcol=0, aiw::Ind<D> rcols=aiw::Ind<D>());
template <typename T, int D>
void dat2Mesh(aiw::IOstream& str, aiw::Mesh<T, D> &dst,
              int vcol=0, aiw::Ind<D> rcols=aiw::Ind<D>());
template <typename T, int D>
void dat2Mesh(std::istream& str, aiw::Mesh<T, D> &dst,
              std::vector<int> vcols={}, int vaxe=-1,
              aiw::Ind<D> rcols=aiw::Ind<D>());
template <typename T, int D>
void dat2Mesh(aiw::IOstream& str, aiw::Mesh<T, D> &dst,
              std::vector<int> vcols={}, int vaxe=-1,
              aiw::Ind<D> rcols=aiw::Ind<D>());
```

Аргумент `vcol` содержит номер столбца файла, в котором лежат значения (либо номера столбцов, если по одной из координат сетки значения записаны в строку, в этом случае соответствующая ось сетки задается при помощи аргумента `vaxe`). Аргумент `rcols` содержит номера столбцов файла, в которых лежат координаты. Допускается индексация столбцов с конца строки с минусом, как в `Python`.

Для создания новых сеток используются функции

```
template <typename T, int D, typename S>
Mesh<T,D> dat2Mesh(S &&str, T def_val=0.,    // значение по умолчанию
   int vcol=0,               // столбец из которого берутся значения
   Ind<D> rcols=Ind<D>(),    // соответствие столбцов координатам сетки,
   int logscale=0,           // логарифмический масштаб (битовая маска)
   Vec<D> eps=Vec<D>(1e-6)); // ошибка (окно кластеризации) при разборе координат
template <typename T, int D, typename S>
Mesh<T,D> dat2Mesh(S &&str, T def_val=0.,     // значение по умолчанию
   std::vector<int> vcols={}, // столбцы из которых берутся значения
   int vaxe=-1,               // координата сетки по которой значения развернуты в строку
   Ind<D> rcols=Ind<D>(),     // соответствие столбцов координатам сетки,
   int logscale=0,            // логарифмический масштаб (битовая маска)
   Vec<D> eps=Vec<D>(1e-6));  // ошибка (окно кластеризации) при разборе координат
```

и их оболочки для конкретных типов потока ввода–вывода.

При создании новой сетки необходимо явно указать битовую маску `logscale`, задающую логарифмический масштаб по отдельным осям. Для вычисления параметров осей (пределов, числа ячеек и шагов) используется алгоритм кластеризации. Для каждой оси все отсчеты (координаты) $\{x_i\}$, $i \in [1, N]$ упорядочиваются по возрастанию. Затем отсчеты объединяются в группы, расстояние между соседними отсчетами в группе не должно превышать значения аргумента `eps` (для логарифмического масштаба отношение соседних отсчетов не должно превышать `1+eps`). Для каждой группы вычисляется координата центра группы X_j, $j \in [1, M]$, $M \leq N$ как среднее арифметическое (для логарифмического масштаба как среднее геометрическое). Число ячеек по соотвествующей оси принимается равным M, пределы сетки $b_{\min}$ и

$b_{\max}$ рассчитываются как

$$b_{\min} = X_1 - \frac{\Delta}{2}, \qquad b_{\max} = X_M + \frac{\Delta}{2}, \qquad \Delta = \begin{cases} \dfrac{X_M - X_1}{M-1}, & M > 1, \\ x_N - x_1 + 2\texttt{eps}, & M = 1, \end{cases}$$

или для логарифмического масштаба

$$b_{\min} = \frac{X_1}{\sqrt{\Delta}}, \qquad b_{\max} = X_M\sqrt{\Delta}, \qquad \Delta = \begin{cases} \exp\dfrac{\log X_M/X_1}{M-1}, & M > 1, \\ \dfrac{x_N}{x_1} + 2\texttt{eps}, & M = 1. \end{cases}$$

2.8 Построение линий постоянного уровня для двумерной сетки — модуль isolines

Для построения изолиний (линий постоянного уровня) на сеточных функциях $f(x, y)$ C++ модуль `isolines` предоставляет класс

```
    class IsoLines{
    public:
        init(const aiw::Mesh<float, 2> &arr, double z0, double dz, bool logscale=false);

        int count() const; // число изолиний
        size_t size(int l) const; // число точек в изолинии l
        float level(int l) const; // значений уровня на изолинии l
        aiw::Vecf<2> point(int l, int i) const; // координаты узла изолинии l

#ifndef SWIG
        template <typename T> void out2dat(T &&S) const;
#endif //SWIG
        void out2dat(aiw::IOstream &S) const;
        void out2dat(std::ostream &S) const;
        void out2dat(const char *path) const;
};
```

Реализация методов класса `IsoLines` находится в файле `src/isolines.cpp`

Метод `init` принимает сетку `arr`, опорное значение функции `z0` (которому точно соответствует изолиния), шаг изолиний `dz` и флаг `logscale` задающий логарифмический масштаб по z (в этом лучаем параметр `dz` трактуется как отношение значения на двух соседних изолиниях, `dz>0`), и строит изолинии.

При построении считается, что функция `arr` билинейно интерполируется внутри ячеек. Каждая изолиния представляет ломанную линию, узлы которой попадают на границы ячеек сеточной функции.

Изолинии могут быть прочитаны из экземпляра класса `IsoLines` при помощи методов `count`, `size`, `level` и `point`, либо выведены в файловый поток вывода (или файла) при помощи методов `out2dat`.

2.9 Многомерный кубический массив на основе Z–кривой Мортона — модуль zcube

Традиционные многомернеы массивы, реализованные в модуле `mesh` (раздел 2.5), не всегда эффективны с точки зрения локальности данных, в этом смысле многомерные массивы, основанные на Z–кривой МОртона, оказываются предпочтительнее. Кроме того, такие массивы оказываются удобнее для проведения ряда операций по рекурсивному разбиению сеток.

Модуль `zcube` предоставляет следующие функции для преобразования смещения от начала массива `f` в позицию ячейки `f` для массива размерности `D` и ранга `R` и обратно:

```
inline uint64_t interleave_bits_any(int D, int32_t x, int R);
inline int32_t de_interleave_bits_any(int D, uint64_t f, int R);
inline uint64_t interleave_bits(int D, int32_t x, int R);
inline int32_t de_interleave_bits(int D, uint64_t f, int R);
```

Функции с постфиксом `_any` реализуют преобразование для массива любой размерности на основе `R` смещений, условий и побитовых операций и/или. Функции без постфикса `_any` оптимизированы для массивов размерности `D=2,3` и вызывают функции с постфиксом `_any` для массивов других размерностей.

Класс `ZCube<T, D>` реализует контейнер с типом ячейки `T` размнерности `D`

```
template <typename T, int D> class ZCube{
public:
    int rank() const;         // ранг массива
    uint64_t size() const;    // число ячекк в массиве
    aiw::Ind<D> bbox() const; // размеры массива по всем осям

    void init(int R_); // инициализация массива
    ZCube(int R_=0);   // конструктор (вызывает метод init)

    // возвращает объект, обспечивающий расчет свдигов к соседям ячейки f
    inline ZCubeNb<D> get_nb(uint64_t f, int periodic=0) const;

    // преобразование позиции ячекйи в сдвиг и обратно
    inline uint64_t pos2offset(const aiw::Ind<D> &pos) const;
    inline aiw::Ind<D> offset2pos(uint64_t f) const;

    // методы для организации обхода массива
    inline aiw::Ind<D> first() const;
    inline bool next(aiw::Ind<D> &pos) const;

    // доступ к ячейкам
    inline       T& operator [](uint64_t f);
    inline const T& operator [](uint64_t f) const;
    inline       T& operator [](const Ind<D> &pos);
    inline const T& operator [](const Ind<D> &pos) const;
```

```
    // проверка принадлежности ячейки к границе массива
    inline bool is_bound(uint64_t f, int axes=0xFF) const;
    inline bool is_bound_up(uint64_t f, int axes=0xFF) const;
    inline bool is_bound_down(uint64_t f, int axes=0xFF) const;

    operator Mesh<T, D>() const; // приведение к типу традиционной сетки Mesh<T, D>
};
```

Доступ к ячейке массива возможен как по D–мерной позиции `pos` так и по смещению от начала массива `f`, оптимизированы методы доступа и обхода по смещению.

Для обхода массива `zarr` можно использовать либо традиционный цикл

```
for(size_t i=0; i<zarr.size(); ++i){ ... zarr[i] ... }
```

либо цикл по позиции, аналогичный циклу для традиционных сеток `Mesh`

```
Ind<D> pos=zarr.first();
while(zarr.next(pos)){ ... zarr[pos] ... }
```

порядок обхода ячеек у обоих вариантов совпадает, но первый вариант требует меньше (практически не требует) накладных расходов.

Для обхода соседей ячейки со смещением `f` используется метод

```
inline ZCubeNb<D> get_nb(uint64_t f, int periodic=0) const;
```

где битовая маска `periodic` задает периодические граничные условия по нужным осям. метод возвращает экземпялр класса

```
template <int D> struct ZCubeNb{
    inline int64_t operator [](const Ind<D> &nb) const;
};
```

который в свою очередь позволяет получать смещения к соседям ячейки при помощи перегруженной операции `[]`, принимающей аргумент `nb` компоненты которого могут содержать числа 0 (сосед слева), 1 (центр) либо 2 (сосед справа), фактически `nb` содержит позицию соседа внутри куба размерами `3x3x...`, в котором исходная ячейка имеет координаты `(1,1...)`. Перегруженная операция возвращает ненулевое значение если соседняя ячейка находится внутри массива и не является исходной ячейкой. Например, для численной схемы с шаблоном «крест», фрагмент кода с обходом соседей будет иметь вид

```
for(size_t f=0; f<zarr.size(); ++f){ // цикл по ячейкам
    ...
    auto znb = zarr.get_nb(f);
    for(int k=0; k<D; k++) // цикл по осям массива
        Ind<D> nb(1); // центр куба
        for(int nb[k]=0; nb[k]<=2; nb[k]+=2){ // левый-правый сосед
            int64_t df = znb[nb];
```

```
                if(df){ // если соседняя ячейка находится внутри области
                    ... zarr[f+df] ... // доступ к соседней ячейке
                }
            }
    }
```

Операции с ZCubeNb оптимизированы настолько, насколько это возможно, в частности никаких преобразований к позиции ячейки и обратно не производится.

В настоящий момент контейнер ZCube реализован достаиочно примитивно, для хранения данных используется std::vector<T>. В дальнейшем планируется перейти на управляени памятью через std::shared_ptr<T> (аналогично контейнеру Mesh), добавить операции copy, crop, slice, flip и transpose, интерполяцию и т.д. — в общем провести унифицикацию с Mesh насколько это возможно.

В настоящий момент контейнер ZCube не инстацирйется в Python.

2.10 Чтение и запись сейсмических данных в формате seg-Y — модуль segy

В C++ модуле segy (файлы include/aiwlib/segy и src/segy.cpp) определены функции и классы для работы с сейсмическими данными в формате Seg-Y.

Метод

```
int segy_raw_read(IOstream &S, std::list<std::vector<float> > &data,
                  std::vector<Vecf<8> > &heads, size_t count, bool read_data);
```

читает все трассы из входного потока S (считается что заголовок файла длиной 3600 байт уже прочитан), записывает их в список трасс data и возвращает число прочитанных трасс. Информация из заголовков трасс записывается в вектор heads, для каждой трассы сохраняется восемь чисел — координаты источника (PVx, PVy, PVz), координаты приемника (PPx, PPy, PPz), шаг дискретизациии dt, число отсчетов в трассе trace_sz. Максимальное число трасс для чтения задается параметром count (-1 без ограничений, до конца файла), если параметр read_data=false производится чтение только заголовков трасс.

Методы

```
#ifndef SWIG
    Mesh<float, 2> segy_read_geometry(IOstream &&S, bool read_file_head=true,
                                      size_t count=-1);
    Mesh<float, 2> segy_read(IOstream &&S, Mesh<float, 2> &data,
                             bool read_file_head=true, size_t count=-1);
    Mesh<float, 3> segy_read(IOstream &&S, Mesh<float, 3> &data);
#endif //SWIG
    Mesh<float, 2> segy_read_geometry(IOstream &S, bool read_file_head=true,
                                      size_t count=-1);
    Mesh<float, 2> segy_read(IOstream &S, Mesh<float, 2> &data, bool read_file_head=true,
                             size_t count=-1);
    Mesh<float, 3> segy_read(IOstream &S, Mesh<float, 3> &data);
```

возвращают геометрию — сетку размерами
[8={PVx,PVy,PVz,PPx,PPy,PPz,dt,trace_sz}][Nx][опицонально Ny].

Параметр read_file_head указывает на необходимость чтения заголовка файла, параметр count задает максимальное число трасс для чтения. Данные записываются в сетку data, при этом ось времени в сейсмограммах всегда отвечает оси номер 0, шаги отвечают шагам сеток.

Методы

```
#ifndef SWIG
    void segy_write(IOstream &&S, const Mesh<float, 1> &data, double z_pow,
                    Vec<2> PV, Vec<3> PP);
    void segy_write(IOstream &&S, const Mesh<float, 2> &data, double z_pow,
                    Vec<2> PV, Vec<3> PP0, double rotate=0.,
                    bool write_file_head=true);
    void segy_write(IOstream &&S, const Mesh<float, 3> &data, double z_pow,
                    Vec<2> PV, Vec<3> PP0, double rotate=0.,
                    bool write_file_head=true);
#endif //SWIG
    void segy_write(IOstream &S, const Mesh<float, 1> &data, double z_pow,
                    Vec<2> PV, Vec<3> PP);
    void segy_write(IOstream &S, const Mesh<float, 2> &data, double z_pow,
                    Vec<2> PV, Vec<3> PP0, double rotate=0.,
                    bool write_file_head=true);
    void segy_write(IOstream &S, const Mesh<float, 3> &data, double z_pow,
                    Vec<2> PV, Vec<3> PP0, double rotate=0.,
                    bool write_file_head=true);
```

записывают сейсмические данные из data в поток S. Параметр z_pow задает степень в шкалирующем амплитуду множителе $z^{\texttt{z\_pow}}$, параметр rotate задает поворот системы координат в плоскости xy относительно точки PP0. Ось времени в сейсмограммах всегда отвечает оси номер 0, шаги по времени и по латерали отвечают шагам сеток.

Для более низкоуровневой работы предназначены класс заголовка файла и класс заголовка трассы.

```
    class SegyFileHead{
char head[3600];
    public:
        double dt;      // шаг по времени, в секундах
        int trace_sz;   // число отсчетов в трассе
        int profile_sz; // число трасс в профиле (магнитограмме)

        SegyFileHead();

// pos --- позиция в байтах
        void set_int16(int pos, int value);
        void set_int32(int pos, int value);
        int get_int16(int pos) const;
```

название поля	размер	позиция	комментарий
SegyFileHead::dt	2B	16	в мкс
SegyFileHead::trace_sz	2B	20, 22	
SegyFileHead::profile_sz	2B	12	
SegyTraceHead::dt	2B	116	в мкс
SegyTraceHead::trace_sz	2B	114	
SegyTraceHead::PV[0]	4B	72	
SegyTraceHead::PV[1]	4B	76	
SegyTraceHead::PP[0]	4B	80	
SegyTraceHead::PP[1]	4B	84	
SegyTraceHead::PP[2]	4B	40	рельеф ПП

Таблица 2.3: разумеры и позиции в заголовке стандартных полей формата seg-Y

```
    int get_int32(int pos) const;

    void dump(aiw::IOstream&);
    void load(aiw::IOstream&);
};
class SegyTraceHead{
    char head[240];
public:
    double dt;          // шаг по времени, в секундах
    int trace_sz;       // число отсчетов в трассе
    aiw::Vec<3> PV, PP; // координаты ПВ и ПП

    SegyTraceHead();

    // pos --- позиция в байтах
    void set_int16(int pos, int value);
    void set_int32(int pos, int value);
    int get_int16(int pos) const;
    int get_int32(int pos) const;

    void dump(aiw::IOstream&);
    bool load(aiw::IOstream&);

    void write(aiw::IOstream &S, float *data); // запись трассы
    aiw::Mesh<float, 1> read(aiw::IOstream&);  // чтение трассы
};
```

Методы get/set_int16/32(pos, value) предназначены для чтения/записи данных (чисел) в произвольные места заголовка. Поля dt, trace_sz, profile_sz, PP и PV автоматически записываются/читаются при вызове методов dump/load на позиции указанные в таблице 2.3.

Методы `SegyTraceHead::write/read` записывают/читают трассу вместе с заголовком (вызывают методы `dump/load`).

Все числа в с плавающей точкой сохранются в офрмате `IEEE`, поэтому в заголовок файла на позицию 24 пишется двухбайтовое число 5 (да, это магия). В планах ввести конфретрех из/в формата `IBM`, но пока это не сделано.

2.11 Контрольные точки для остановки и последующего восстановления расчета — модуль `checkpoint`

Модуль `checkpoint` состоит из файлов `aiwlib/checkpoint` и `src/checkpoint.cpp` и предоставляет макрос `CHECKPOINT(ARGS...)`, обеспечивающий сохранение в контрольных точках и последующее восстановление расчета. Макрос работает через глобальный экземпляр класса `CheckPoint`

```
class CheckPoint{
public:
    void init(const char *path, int mode=0); // 0 - auto, 1 - read, other - write
#ifndef SWIG
    template <typename ... Args>
    bool operator()(const char* fname, int line,
                    const char *argnames, Args& ... args);
#endif //SWIG
};
extern CheckPoint checkpoint;
#define CHECKPOINT(ARGS...) aiw::checkpoint(__FILE__, __LINE__, #ARGS, ARGS)
```

Метод `init` задает путь к файлу, содержащему состояние расчета, и режим работы — по умолчанию автоматический режим (восстановление состояния если файл существует и сохранение состояния если файла нет, `mode=0`), предварительное восстановление состояния расчета (`wmode=1`) или сохранение состояния расчета (`mode!=0,1`). . Без вызова метода

```
checkpoint.init(...)
```

все макросы `CHECKPOINT` всегда возвращают `true`.

В режиме записи состояния расчета, вызовы макросов `CHECKPOINT` всегда возвращают `true`, при этом каждый вызов приводит к записи аргументов макроса в файл, указанный при вызове метода `checkpoint.init`. Запись производится в бинарном формате, для аргументов макроса должны быть перегружены операции

```
aiw::IOstream& operator < (aiw::IOstream&, const T&);
aiw::IOstream& operator > (aiw::IOstream&, T&);
```

см. раздел 2.18 (этому требованию удовлевотряют практически все контейнеры `aiwlib`, встроенные типы, типы `std::string`, `std::vector`, `std::list` и `std::map`).

В режиме восстановления состояния расчета вызовы макроса CHECKPOINT возвращают false до тех пор, пока очередь не дойдет до последнего вызова в режиме записи при предыдущем запуске расчета[5] — при этом производится восстановление значения аргументов макроса, макрос возвращает true, а глобальный экземпляр класса checkpoint переводится в режим записи и может быть снова использован для сохранения состояния расчета с последующим восстановлением.

Расстановка вызовов макроса CHECKPOINT вдоль трассы выполнения программы (в условиях, циклах и т.д.) позволяет при восстановлении расчета пропускать уже пройденные ранее участки трассы выполнения. Недостатком такого подхода является некоторое замедление работы — при кжадом вызове макроса в режиме записи происходит сохранение данных на диск.

Пример использования

```
...
int main(int argc, const char **argv){
    ...
    checkpoint.init(argv[1], !(argc>2 && std::string(argv[2])=="-load"));
    aiw::Mesh<float, 3> data; // сетка определяющая состояние расчета
    ...
    for(int i=0; CHECKPOINT(i, data) && i<100; i++){
        // основной расчетный цикл
        ...
    }
    ...
}
```

2.12 Сериализация данных в формате pickle языка Python — модуль pickle

Модуль pickle позволяет выгружать сериализовыванные данные из C++ в формате pickle, что бывает полезно при подготовке данных для последующей обработки в Python.

Для сериализации предоставляются следующие структуры и функции

```
struct NoneType{};
const NoneType None;

struct Pickle{
    // основные операторы для вывода объектов
    Pickle& operator << (const Pickle& other);
    Pickle& operator << (NoneType);
    Pickle& operator << (bool x);
    Pickle& operator << (int x);
    Pickle& operator << (long x);
```

[5]Определение соответствующего вызова производится на основе имени файла с исходным кодом, номера строки в файле и набора имен аргументов

```
        Pickle& operator << (double x);
        Pickle& operator << (const std::string& x);
        Pickle& operator << (const char* x);

        template <typename T> Pickle& operator << (const std::complex<T> &x);

        // для сериализации записи в словаре вида k:v
        template <typename T1, typename T2>
        Pickle& operator ()(const T1& k, const T2& v);

        // для модуля objconf
        template <typename T> void get(const std::string& k, const T& v);
        template <typename T> void set(const std::string& k, T& v);
    };

    template <typename ... Args> Pickle pickle_tuple(const Args& ... args);
    template <typename ... Args> Pickle pickle_list(const Args& ... args);
    template <typename ... Args> Pickle pickle_set(const Args& ... args);
    inline Pickle pickle_dict();
    inline Pickle pickle_class(const char* module, const char *name, bool dict=true);

    inline std::ostream& operator << (std::ostream& S, const Pickle &P);
```

Для сериализации рекомендуется использовать функции вида `pickle_`XXX.

Сериализованы могут быть любые данные типа `T`, для которых перегружена операция

```
Pickle& operator << (Pickle &P, const T &x);
```

допускается перегрузка операций для своих типов данных, например код

```
struct A{ int x; double y; bool z; };
Pickle& operator << (Pickle &P, const A &a){ P<<pickle_tuple(a.x, a.y, a.z); }
```

будет сериализовывать экземпляры структуры `A` как кортежи.

Сформированный в итоге экземпляр структуры `Pickle` может быть выведен в поток `std::ostream` (при этом обязательно нужно вывести финальный символ «точка») и далее загружен в `Python`, например код

```
std::cout<<pickle_tuple(
    pickle_list(vec(1.,2,3), ind(2,3), vecf(3,4,5,6)),
    std::complex<double>(1,2),
    pickle_dict()(ind(1), "qwe"),
    None)
 <<'.';
```

выдаст в итоге сериализованную структуру данных `Python`

```
([Vec(1.0,2.0,3.0), Ind(2,3), Vecf(3.0,4.0,5.0,6.0)],
 (1+2j),
 {Ind(1): 'qwe'},
 None)
```

2.13 Дискретизация разбиения Вороного на равномерной сетке в D–мерном пространстве — модуль `voronoy`

Модуль `voronoy` предоставляет функцию

```
template <int D, typename T1, typename T2>
void voronoy(const std::vector<Vec<D, T1> > &src, aiw::Mesh<T2, D> dst);
```

дискретизирующую разбиение Вороного в D–мерном пространстве. Вектор `src` содержит множество исходных точек, после вызова функции в ячейках сетки `dst` будут прописаны номера ближайших к центрам ячеек сетки `dst` из множества `src` (считая с нуля до `src.size()-1` включительно).

При дискретизации сетка `dst` разбивается на 2^D одинаковых D–мерных параллелепипедов, затем для каждрого угла перебором находится ближайшая точка из множества `src`. Если все углы параллелепипеда относятся к одной и той же точке, весь параллелепипед относится к этой точке. В противном случае параллелепипед снова разбивается на 2^D частей, и т.д. — вплоть до того момента как элементом разбиения будет одна ячейка сетки `dst`.

2.14 Описание пользовательских типов — модуль `typeinfo`

Модуль `typeinfo` предоставляет механизм для описания пользовательских типов, обеспечивающий некоторые возможности для интроспекции. В частности, описание пользовательского типа может быть сохранено (например как описание типа ячейки сетки), и использовано в дальнейшем при анализе данных, построении графиков и т.д.

На уровне конечного пользователя работа с модулем `typeinfo` выглядит как вызов макросов

```
#define TYPEINFO(ARGS...)
#define TYPEINFOX(T, ARGS...)
```

перегружающих для пользовательского типа `T` операции

```
aiw::TypeInfo operator ^ (aiw::TypeInfoObj& tio) const;
aiw::TypeInfo operator ^ (const T& X, aiw::TypeInfoObj& tio);
```

Макрос `TYPEINFO` перегружает операцию внутри пользовательского типа, макрос `TYPEINFOX` перегружает операцию снаружи. Например

```
struct A{
    int x;
    double y[10];
```

```
    bool f;
    TYPEINFO(x, y, f);
};
struct B{
    A a[2][3];
    Vec<3> v;
};
TYPEINFOX(B, X.a, X.v);
```

Предполагается, что использование этого механизма позволит эффективно обрабатывать результаты расчетов, сохраненных в виде сеток из польщовательских типов. Находится на стадии реализации.

2.15 Некоторые элементы аналитической геометрии — модуль angem

Модуль `angem` предоставляет несколько функицй из аналитической геометрии.

```
template <int D, typename T> Vec<D, T>
point2plane(const Vec<D, T> &r, const Vec<D, T> &p, const Vec<D, T> &n);
```

проекция точки $\mathbf{r}$ на плоскость проходящую через точку $\mathbf{p}$ с нормалью $\mathbf{n}$, $n = 1$, вычисляется как $\mathbf{r} - (\mathbf{r} - \mathbf{p}) \cdot \mathbf{n} \cdot \mathbf{n}$.

```
template <int D, typename T>
bool cross_plane(Vec<D, T> &res, const Vec<D, T> &r, const Vec<D, T> &g,
                 const Vec<D, T> &p, const Vec<D, T> &n, bool ray=false);
```

расчет точки пересечения прямой (или луча) проходящей через точку $\mathbf{r}$ в направлении $\mathbf{g}$, $g = 1$ и плоскости проходящей через точку $\mathbf{p}$ с нормалью $\mathbf{n}$, $n = 1$ вычисляется как

$$\texttt{res} = \mathbf{r} - \mathbf{r}\frac{(\mathbf{r} - \mathbf{p}) \cdot \mathbf{n}}{\mathbf{n} \cdot \mathbf{g}}.$$

Функция возвращает `true` если проекция существует — если луч перескает плоскость, должно выполняться условие

$$(\mathbf{r} - \mathbf{p}) \cdot \mathbf{n} \cdot \mathbf{n} \cdot \mathbf{g} < 0,$$

кроме того для существования проекции необходимо что бы $|\mathbf{n} \cdot \mathbf{g}| > 10^{-8}$. В случае если проекция сущсетвует, функция возвращает `true`, результат записывается в `res`.

Упрощенным (оптимизированным) вариантом этой функции является

```
template <int D, typename T>
bool cross_plane(Vec<D, T> &res, const Vec<D, T> &r, const Vec<D, T> &g,
 const Vec<D, T> &p, int axe, bool ray=false);
```

где ориентация плоскости задается ортогональной ей осью координат `axe`.

```
template <int D, typename T> Vec<D, T>
shoot_box_out(const Vec<D, T> &r, const Vec<D, T> &g,
              const Vec<D, T> &a, const Vec<D, T> &b);
```

расчитывает точку пересечения луча выходящего из точки $\mathbf{r}$ в направлении $\mathbf{g}$, $g = 1$ и параллепипеда заданного точками $\mathbf{a}$, $\mathbf{b}$, $\mathbf{a} \leq \mathbf{r}$, $\mathbf{r} \leq \mathbf{b}$.

2.16 Решение СЛАУ с трехдиагональными матрицами (методы прогонки) — модуль 3diag

Модуль `3diag` предоставляет методы прогонки (обычной и циклической) для решения СЛАУ с трехдиагональными матрицами

$$M_{i-1,i}X_{i-1} + M_{i,i}X_i + M_{i+1,i}X_{i+1} = R_i,$$

```
template <typename T>
void shuttle_alg(const Mesh<T, 2> &M, const Mesh<T, 1> &R, Mesh<T, 1> &X);
template <typename T>
void cyclic_shuttle_alg(const Mesh<T, 2> &M, const Mesh<T, 1> &R, Mesh<T, 1> &X);
```

где `M` — тридиагональная матрица, сетка размерами `3xN`, главная диагональ расположена в ячейках `(1,0...N-1)`, для обычной прогонки ячейки `(0,0)` и `(2,N-1)` игнорируются; `R` — правая часть, `X` — решение. Сетки `R` и `N` должны иметь размер `N` ячеек.

Алгоритм для обычной прогонки взят из английской Википедии[6], алгоритм для циклической прогонки взят у Самарского[7] с изменением знаков у главной диагонали и правой части.

Проверка на диагональное преобладание **не** проводится.

2.17 Источник случайных числе и специфические операции в трехмерном декартовом пространстве — модуль gauss

Модуль `gauss` предоставляет ряд функций для генерации случайных чисел[8], а так же специфические операции в трехмерном декратовом пространстве — операции поворота и методы генерации вектора, ортогонального данному. Макрос `MINGW` определяется при кросс–компиляции под `OS Windows`.

```
#ifndef MINGW
    const double rand_alpha = 1./(1.+RAND_MAX), rand_alpha2PI = 2*M_PI/(1.+RAND_MAX);
    inline void rand_init();    // иницализация генератора случайных чисел
```

[6] `https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm`

[7] Самарский А.А. «Введение в теорию разностных схем». М.: Наука, 1971. — 553 с. `http://info.alnam.ru/book_sub.php?id=91`

[8] Планируется сделать потокобезопасный вариант таких источников

```
    // случайное число с нормальным распределением,
    //     нулевым матожиданием и единичной дисперсией
    inline double rand_gauss();
    // вектор случайных чисел с нормальным распределением,
    //     нулевым матожиданием и единичной дисперсией
    template <int D, class T> inline Vec<D, T> rand_gaussV();
#else //MINGW
    inline void sincos(double phi, double *s, double *c);
#endif //MINGW
    // оператор поворота вектора a вокруг вектора b
    template <typename T> inline aiw::Vec<3, T>
         rotate(const aiw::Vec<3, T>& a, const aiw::Vec<3, T>& b);
    // оператор поворота вектора a вокруг вектора b в виде ряда длины R
    template <int R, typename T> inline aiw::Vec<3, T>
         rotate(const aiw::Vec<3, T>& a, const aiw::Vec<3, T> &b);
    // перевод из полярных координат в декартовы
inline aiw::Vec<3, double> polar(double theta, double phi);
    // перевод из декартовых координат в полярные, возвращает theta, phi
    template <typename T>
    inline aiw::Vec<2, double> polar(const aiw::Vec<3, T> &n);
    // генерация вектора перпендикулярного вектору a в 2D, длины a
    template <typename T>
    inline aiw::Vec<2, T> perp(const aiw::Vec<2, T> &a);
    // генерация вектора перпендикулярного вектору a в 3D, единичной длины
    template <typename T>
    inline aiw::Vec<3, T> perp(const aiw::Vec<3, T> &a);
#ifndef MINGW
    // источник шума, отклоняющий вектор a с сохранением его длины, с дисперсией g
    template <typename T>
    aiw::Vec<3, T> gauss_rotate(const aiw::Vec<3, T>& a, double g);
#endif //MINGW
```

Для реализации оператора поворота использовано выражение

$$\mathbf{a} \circlearrowleft \mathbf{b} = \mathbf{a}\cos b - \mathbf{a} \times \mathbf{b}\frac{\sin b}{b} + \mathbf{b}(\mathbf{a} \cdot \mathbf{b})\frac{1 - \cos b}{b^2}.$$

2.17.1 Определение поворота

Введем матрицу вращения $\Xi(\mathbf{n}, \delta)$ вокруг произвольного единичного вектора $\mathbf{n}$ на произвольный угол δ против часовой стрелки в правой системе координат, если смотреть против направ-

ления $\mathbf{n}$:

$$\Xi(\mathbf{n},\delta) = \begin{pmatrix} \left(n_x^2-n_y^2-n_z^2\right)\sin^2\frac{\delta}{2}+\cos^2\frac{\delta}{2} & 2\sin\frac{\delta}{2}\left(n_xn_y\sin\frac{\delta}{2}-n_z\cos\frac{\delta}{2}\right) & 2\sin\frac{\delta}{2}\left(n_xn_z\sin\frac{\delta}{2}+n_y\cos\frac{\delta}{2}\right) \\ 2\sin\frac{\delta}{2}\left(n_xn_y\sin\frac{\delta}{2}+n_z\cos\frac{\delta}{2}\right) & \left(n_y^2-n_x^2-n_z^2\right)\sin^2\frac{\delta}{2}+\cos^2\frac{\delta}{2} & 2\sin\frac{\delta}{2}\left(n_yn_z\sin\frac{\delta}{2}-n_x\cos\frac{\delta}{2}\right) \\ 2\sin\frac{\delta}{2}\left(n_xn_z\sin\frac{\delta}{2}-n_y\cos\frac{\delta}{2}\right) & 2\sin\frac{\delta}{2}\left(n_yn_z\sin\frac{\delta}{2}+n_x\cos\frac{\delta}{2}\right) & \left(n_z^2-n_x^2-n_y^2\right)\sin^2\frac{\delta}{2}+\cos^2\frac{\delta}{2} \end{pmatrix} =$$

$$= \begin{pmatrix} n_x^2(1-\cos\delta)+\cos\delta & n_xn_y(1-\cos\delta)-n_z\sin\delta & n_xn_z(1-\cos\delta)+n_y\sin\delta \\ n_xn_y(1-\cos\delta)+n_z\sin\delta & n_y^2(1-\cos\delta)+\cos\delta & n_yn_z(1-\cos\delta)-n_x\sin\delta \\ n_xn_z(1-\cos\delta)-n_y\sin\delta & n_yn_z(1-\cos\delta)+n_x\sin\delta & n_z^2(1-\cos\delta)+\cos\delta \end{pmatrix}.$$

Введем бинарный оператор поворота $\circlearrowleft$, поворот некоторого вектора $\mathbf{a}$ вокруг $\mathbf{n}$ на угол δ будем обозначать как $\mathbf{a} \circlearrowleft \delta\mathbf{n}$

$$\mathbf{a} \circlearrowleft \delta\mathbf{n} \equiv \Xi(\mathbf{n},\delta)\cdot\mathbf{a}.$$

Очевидно оператор $\circlearrowleft$ некоммутативен

$$\mathbf{a} \circlearrowleft \mathbf{b} \neq \mathbf{b} \circlearrowleft \mathbf{a},$$

неассоциативен

$$(\mathbf{a} \circlearrowleft \mathbf{b}) \circlearrowleft \mathbf{c} \neq \mathbf{a} \circlearrowleft (\mathbf{b} \circlearrowleft \mathbf{c}),$$

и недистрибутивен слева

$$\mathbf{a} \circlearrowleft (\mathbf{b}+\mathbf{c}) \neq (\mathbf{a} \circlearrowleft \mathbf{b}) + (\mathbf{a} \circlearrowleft \mathbf{c}).$$

Рассмотрим некоторые его свойства:

$$\begin{aligned} (\mathbf{a}+\mathbf{b}) \circlearrowleft \mathbf{c} &= (\mathbf{a} \circlearrowleft \mathbf{c}) + (\mathbf{b} \circlearrowleft \mathbf{c}), \\ (\alpha\mathbf{a}) \circlearrowleft \mathbf{b} &= \alpha(\mathbf{a} \circlearrowleft \mathbf{b}), \\ \mathbf{a} \circlearrowleft \alpha\mathbf{a} &= \mathbf{a}, \\ \mathbf{a} \circlearrowleft \mathbf{b} \circlearrowleft (-\mathbf{b}) &= \mathbf{a}, \\ \mathbf{a} \circlearrowleft \alpha\mathbf{b} \circlearrowleft \beta\mathbf{b} &= \mathbf{a} \circlearrowleft (\alpha+\beta)\mathbf{b}, \\ (\mathbf{a} \circlearrowleft \mathbf{b}) \circlearrowleft \mathbf{c} &= (\mathbf{a} \circlearrowleft \mathbf{c}) \circlearrowleft (\mathbf{b} \circlearrowleft \mathbf{c}), \\ (\mathbf{a} \circlearrowleft \mathbf{b}) \circlearrowleft \alpha\mathbf{a} &= \mathbf{a} \circlearrowleft (\mathbf{b} \circlearrowleft \alpha\mathbf{a}), \\ \mathbf{a} \circlearrowleft (\mathbf{b} \circlearrowleft \mathbf{c}) &= \mathbf{a} \circlearrowleft (-\mathbf{c}) \circlearrowleft \mathbf{c} \circlearrowleft (\mathbf{b} \circlearrowleft \mathbf{c}) = \mathbf{a} \circlearrowleft (-\mathbf{c}) \circlearrowleft \mathbf{b} \circlearrowleft \mathbf{c}. \end{aligned}$$

2.17.2 Разложение поворота в ряд

Не теряя общности рассуждений, рассмотрим поворот вектора $(0, r_y, r_z)$ вокруг вертикальной оси на угол α, и разложим $\sin\alpha$ и $\cos\alpha$ в ряд в окрестностях точки $\alpha = 0$:

$$\begin{pmatrix} 0 \\ r_y \\ r_z \end{pmatrix} \circlearrowleft \begin{pmatrix} 0 \\ 0 \\ \alpha \end{pmatrix} = \begin{pmatrix} -r_y\sin\alpha \\ r_y\cos\alpha \\ r_z \end{pmatrix} = \begin{pmatrix} -r_y\sum\limits_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)!}\alpha^{2n+1} \\ r_y\sum\limits_{n=0}^{\infty}\frac{(-1)^n}{(2n)!}\alpha^{2n} \\ r_z \end{pmatrix} = \begin{pmatrix} -r_y\sum\limits_{n=0}^{\infty}\frac{1}{n!}\alpha^n\sin\frac{\pi n}{2} \\ r_y\sum\limits_{n=0}^{\infty}\frac{1}{n!}\alpha^n\cos\frac{\pi n}{2} \\ r_z \end{pmatrix}.$$

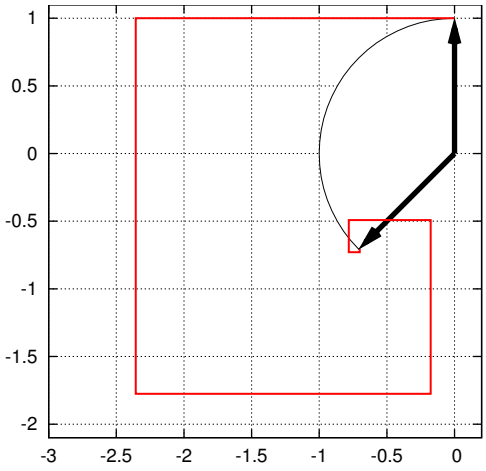

Рис. 2.1: Пример поворота вектора $(0, 1, 0)$ на угол $3\pi/4$

Пусть $\mathbf{a} \times^n \mathbf{b} \equiv [...[\underbrace{\mathbf{a} \times \mathbf{b}] \times ...\mathbf{b}}_{n}]$ и $\mathbf{a} \times^0 \mathbf{b} \equiv \mathbf{a}$. Рассмотрим последовательность векторных произведений:

$$\begin{pmatrix} 0 \\ r_y \\ r_z \end{pmatrix} \times \begin{pmatrix} 0 \\ 0 \\ \alpha \end{pmatrix} = \begin{pmatrix} \alpha r_y \\ 0 \\ 0 \end{pmatrix}, \quad \begin{pmatrix} 0 \\ r_y \\ r_z \end{pmatrix} \times^2 \begin{pmatrix} 0 \\ 0 \\ \alpha \end{pmatrix} = \begin{pmatrix} 0 \\ -\alpha^2 r_y \\ 0 \end{pmatrix}, \quad \begin{pmatrix} 0 \\ r_y \\ r_z \end{pmatrix} \times^3 \begin{pmatrix} 0 \\ 0 \\ \alpha \end{pmatrix} = \begin{pmatrix} -\alpha^3 r_y \\ 0 \\ 0 \end{pmatrix},$$
$$\begin{pmatrix} 0 \\ r_y \\ r_z \end{pmatrix} \times^4 \begin{pmatrix} 0 \\ 0 \\ \alpha \end{pmatrix} = \begin{pmatrix} 0 \\ \alpha^4 r_y \\ 0 \end{pmatrix}, \quad ..., \quad \begin{pmatrix} 0 \\ r_y \\ r_z \end{pmatrix} \times^n \begin{pmatrix} 0 \\ 0 \\ \alpha \end{pmatrix} = \begin{pmatrix} \alpha^n r_y \sin \frac{\pi n}{2} \\ \alpha^n r_y \cos \frac{\pi n}{2} \\ 0 \end{pmatrix}.$$

Тогда, с учетом четности $\cos\alpha$ и нечетности $\sin\alpha$, поворот можно представить как

$$\mathbf{a} \circlearrowleft \mathbf{b} = \sum_{n=0}^{\infty} \frac{1}{n!} \mathbf{a} \times^n (-\mathbf{b}) = \sum_{n=0}^{\infty} \frac{(-1)^n}{n!} \mathbf{a} \times^n \mathbf{b}.$$

Этот же результат можно получить другим способом. Рассмотрим уравнение вида

$$\dot{\mathbf{a}} = -\left[\mathbf{a} \times \begin{pmatrix} 0 \\ 0 \\ \omega \end{pmatrix}\right] \tag{2.1}$$

описывающее очевидно вращение вектора $\mathbf{a}$ вокруг вертикальной оси с угловой скоростью ω против часовой стрелки (если смотреть сверху). Дифференцируя обе части уравнения (2.1) по времени получаем рекуррентное соотношение для произвольной производной вектора $\mathbf{a}$ по времени:

$$\frac{\partial^n \mathbf{a}}{\partial t^n} = -\left[\frac{\partial^{n-1} \mathbf{a}}{\partial t^{n-1}} \times \begin{pmatrix} 0 \\ 0 \\ \omega \end{pmatrix}\right] = ... = (-1)^n \left[\mathbf{a} \times^n \begin{pmatrix} 0 \\ 0 \\ \omega \end{pmatrix}\right].$$

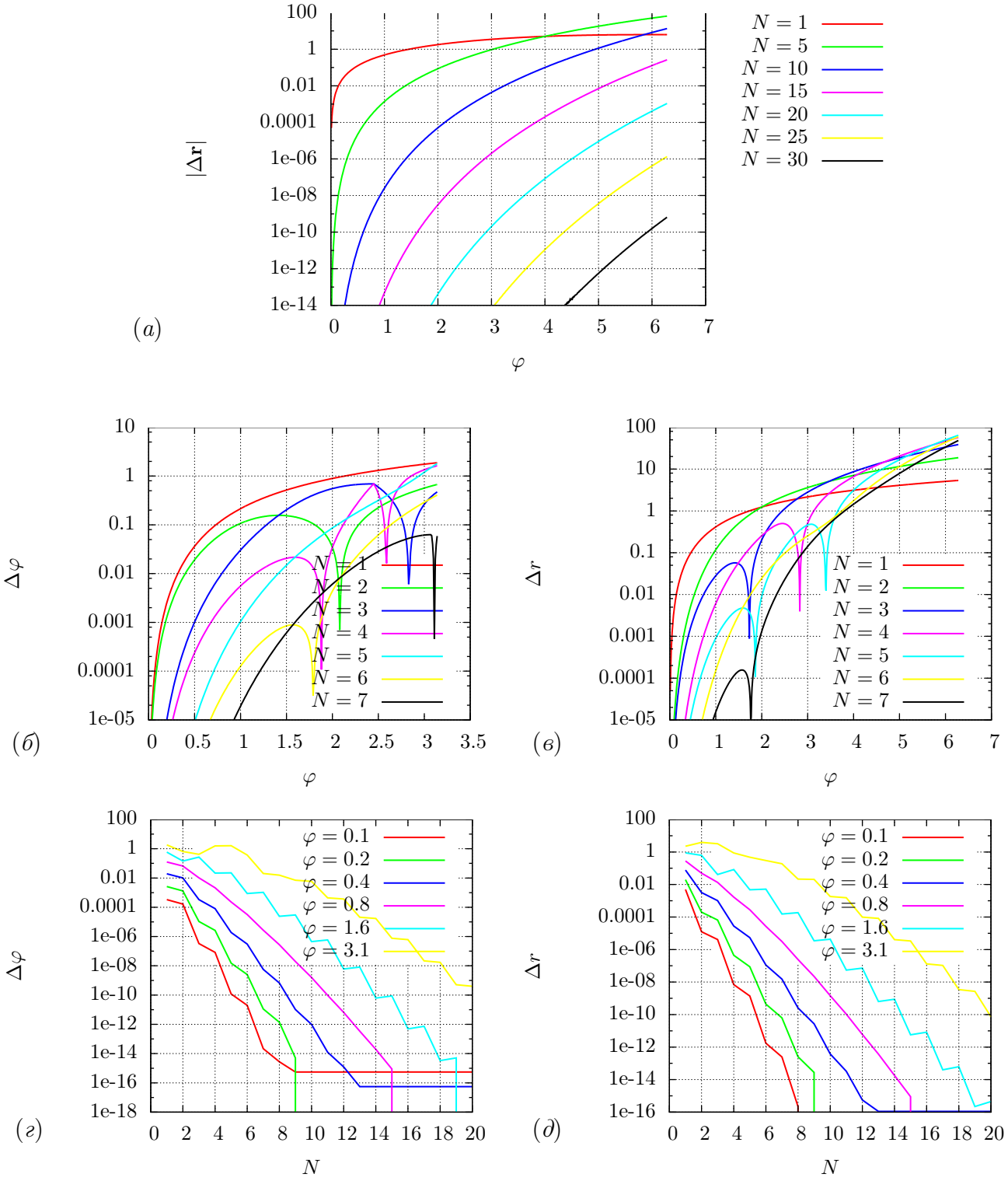

Рис. 2.2: Зависимости модуля ошибки $\vec{r}$ от угла поворота φ единичного вектора $(0, 1, 0)$ для различного числа членов разложения N (*а*). Зависимости абсолютной ошибки по углу $\Delta\varphi$ (*б*) и модулю радиуса Δr (*в*) от угла поворота φ при использовании конечного числа членов разложения N. Зависимости абсолютной ошибки по углу $\Delta\varphi$ (*г*) и радиусу Δn (*д*) от числа членов разложения N для различных углов поворота φ

Пусть начальное значение вектора $\mathbf{a}|_{t=0} = \mathbf{a}_0$, тогда

$$\mathbf{a}_0 \circlearrowleft \begin{pmatrix} 0 \\ 0 \\ \omega t \end{pmatrix} = \mathbf{a}(t).$$

Разложим вектор $\mathbf{a}$ в ряд Тейлора в окрестности точки $t = 0$:

$$\mathbf{a}_0 \circlearrowleft \begin{pmatrix} 0 \\ 0 \\ \omega t \end{pmatrix} = \sum_{n=0}^{\infty} \frac{t^n}{n!} \frac{\partial^n \mathbf{a}}{\partial t^n} = \sum_{n=0}^{\infty} \frac{(-1)^n}{n!} \left[\mathbf{a} \times^n \begin{pmatrix} 0 \\ 0 \\ \omega t \end{pmatrix} \right].$$

Пример поворота на угол $3\pi/4$ приведен на рис. 2.1. Зависимости ошибок разложения приведены на рис. 2.2.

Разложение оператора поворота в ряд может существенно ускорить численные схемы, использующие подобные конструкции. Так, поворот через матрицу (функция `rotate` из модуля `derart` библиотеки `aivlib`) занимает на процессоре Intel(R) Core(TM)2 CPU U7500 около 170 тактов, а поворот через ряд (параметризованная по числу членов разложения R функция `rotop<R>` из того же модуля) требует примерно 10 тактов на каждый из членов разложения.

Поскольку

$$\mathbf{a} \times^{n+2} \mathbf{b} = -b^2 \mathbf{a} \times^n \mathbf{b}$$

можно представить оператор поворота в виде суммы трех векторов

$$\begin{aligned} \mathbf{a} \circlearrowleft \mathbf{b} = \sum_{n=0}^{\infty} \frac{(-1)^n}{n!} \mathbf{a} \times^n \mathbf{b} = \mathbf{a} - [\mathbf{a} \times \mathbf{b}] \sum_{n=0}^{\infty} \frac{(-1)^n b^{2n}}{(2n+1)!} + \\ + \big[[\mathbf{a} \times \mathbf{b}] \times \mathbf{b}\big] \sum_{n=0}^{\infty} \frac{(-1)^n b^{2n}}{(2n+2)!} \equiv \mathbf{a} - [\mathbf{a} \times \mathbf{b}]\, \beta_1(b) - \big[\mathbf{b} \times [\mathbf{a} \times \mathbf{b}]\big]\, \beta_2(b) = \\ = \mathbf{a}\big(1 - b^2 \beta_2(b)\big) - [\mathbf{a} \times \mathbf{b}]\, \beta_1(b) + \mathbf{b}\, (\mathbf{a} \cdot \mathbf{b})\, \beta_2(b), \quad (2.2) \end{aligned}$$

где

$$\beta_1(b) = \sum_{n=0}^{\infty} \frac{(-1)^n b^{2n}}{(2n+1)!} = \frac{\sin b}{b}, \qquad \beta_2(b) = \sum_{n=0}^{\infty} \frac{(-1)^n b^{2n}}{(2n+2)!} = \frac{1 - \cos b}{b^2},$$

или

$$\mathbf{a} \circlearrowleft \mathbf{b} = \mathbf{a} \cos b - [\mathbf{a} \times \mathbf{n}_b] \sin b + \mathbf{n}_b (\mathbf{a} \cdot \mathbf{n}_b)(1 - \cos b).$$

Рассмотрим выражение $\partial(\mathbf{a} \circlearrowleft \varphi\mathbf{b})/\partial\varphi$, где $\mathbf{b}$ вектор задающий ось вращения.

$$\begin{aligned} \frac{\partial(\mathbf{a} \circlearrowleft \varphi\mathbf{b})}{\partial\varphi} = \frac{\partial}{\partial\varphi} \sum_{n=0}^{\infty} \frac{(-1)^n}{n!} \mathbf{a} \times^n \varphi\mathbf{b} = \sum_{n=1}^{\infty} \frac{(-1)^n \varphi^{n-1}}{(n-1)!} \mathbf{a} \times^n \mathbf{b} = \\ = -\left[\sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{(n-1)!} \mathbf{a} \times^n \varphi\mathbf{b} \right] \times \mathbf{b} = -(\mathbf{a} \circlearrowleft \varphi\mathbf{b}) \times \mathbf{b} = \mathbf{b} \times (\mathbf{a} \circlearrowleft \varphi\mathbf{b}). \end{aligned}$$

2.18 Операции бинарного потокового ввода/вывода — модуль binaryio

Модуль binaryio перегружает операции

```
template<typename T> inline IOstream& operator < (IOstream& S, const T& X);
template<typename T> inline IOstream& operator > (IOstream& S,       T& X);
```

как операции бинарного ввода/вывода в поток aiw::IOstream для большинства актуальных типов T. Операции перегружены для встроенных типов int8_t, uint8_t, int16_t, uint16_t, int32_t, uint32_t, int64_t, uint64_t, char, bool, float, double, типов std::complex<T>, std::string, T[D], std::vector<T>, std::list<T>, std::map<T1,T2>, aiw::Vec<D,T>, и типов для которых определены публичные методы

```
void dump(aiw::IOstream &S) const;
void load(aiw::IOstream &S);
```

Кроме того модуль предоставляет макрос

```
#define BINARYIO4POD                                                          \
    inline void dump(aiw::IOstream &S) const { S.write(this, sizeof(*this)); } \
    inline void load(aiw::IOstream &S)       { S.read(this, sizeof(*this)); }
```

вызов которого в POD–типе объявляет методы dump/load и автоматически перегружает операции < и >.

2.19 Настройка пользовательских классов — модуль objconf

Модуль objconf предоставляет макрос

```
#define CONFIGURATE(ARGS...)
```

который при вызове внутри пользовательского класса

```
struct A{
    int x; double y; bool z;
    CONFIGURATE(x, y, z);
};
```

создает в классе метод

```
template<typename ConfT>
void configurate(ConfT &conf, bool wmode, const char *prefix="");
```

который в зависимости от значения аргумента wmode вызывает методы

```
conf.set(prefix+name, parametr); // wmode==true
conf.get(prefix+name, parametr); // wmode==false
```

для каждого из полей класса, перечисленных в аргументах макроса CONFIGURATE. Аргумент prefix позволяет добавить префикс к именам всех полей при чтении/записи.

Созданный метод configurate(...) может использоваться при чтении/записи конфигурационных файлов (раздел 2.20), записи состояния объекта в формате pickle (раздел 2.12) и т.д.

2.20 Чтение и запись конфигурационных файлов — модуль configfile

Модуль configfile состоит из заголовчного файла configfile и файла src/configfile.cpp, и предоставляет средства для чтения/записи конфигурационных файлов в текстовом формате

```
# комментарий
имя_параметра=значение параметра
```

для каждого параметра может использоваться **только одна** строка.

Модуль определяет функции чтения/записи для различных типов из потоков ввода/вывода std::iostream

```
template <typename T> void printf_obj(std::ostream &S, const T &X){ S<<X; }
template <typename T> void scanf_obj(std::istream &S, T &X){ S>>X; }
```

которые могут быть перегружены специальным образом для отдельных типов

```
void printf_obj(std::ostream &S, bool X);
void scanf_obj(std::istream &S, bool &X);

template <typename T> void printf_obj(std::ostream &S, const std::complex<T> &X);
template <typename T> void scanf_obj(std::istream &S, std::complex<T> &X);

void scanf_obj(std::istream &S, std::string &X);
```

для типа bool допустимыми являются значения в конфигурацинном файле

```
Y y YES Yes yes ON  On  on  TRUE  True  true  V v 1
N n NO  No  no  OFF Off off FALSE False false X x 0
```

Для типа std::complex используется формат Python $x \pm yj$.

Для типа std::string при чтении из конфигурационного файла используется строка после знака = с отброшенными с конца и начала символами пробела, \t и \r (аналог вызывова функции str.strip() языка Python).

Непосредственно для чтения/записи конфигурационного файла используется класс

```
class ConfigFile{
public:
    int no_key_act = 2; // 0 - ignore, 1 - warning, 2 - exception
#ifndef SWIG
    // получает значение параметра par с именем key из другого объекта
    //        для записи в конфигурационный файл
    template <typename T>
    void get(const std::string &key, const T &par);
    // устанавливает значение параметра par с именем key в другом объекта
    //        на основе конфигурационного файла
    template <typename T>
```

```
        void set(const std::string &key, T &par);

        void load(std::istream &&fin);
        void dump(std::ostream &&fout) const;
        void load(std::istream &fin);
        void dump(std::ostream &fout);
    #endif //SWIG
        void load(const char *path);
        void dump(const char *path) const;
        void clear();
    };
```

При чтении конфигурациооного файла для экземпляра класса `ConfigFile` вызывается один из методов `load`, затем экземпляр класса передается первым аргументов в методы `configurate` настраиваемых пользовательских объектов, см. раздел 2.19. Поле `no_key_act` управляет реакцией на отсутствие какого либо параметра в конфигурационном файле: `0` — игнорировать, `1` – выдать предупреждение, `2` — сгенерировать исключение.

При записи конфигурациооного файла экземпляра класса `ConfigFile` передается первым аргументов в методы `configurate` настраиваемых пользовательских объектов, см. раздел 2.19, затем для него необходимов вызвать один из методов `dump`.