ПРИЛОЖЕНИЕ В

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ (C++)

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Библиотека открытого доступа DataReconcile для согласования неточных индустриальных данных за счет учета взаимосвязей между ними для нужд промышленных предприятий Индустрии 4.0 и целей Национальной технологической инициативы на языке C++

2025 год

В.1 НАЗНАЧЕНИЕ БИБЛИОТЕКИ

Библиотека предназначена для решения задач **согласования неточных данных** (Data Reconcile) в условиях, когда входные данные содержат ошибки, неточности или противоречия. Она позволяет корректировать данные таким образом, чтобы они удовлетворяли заданным ограничениям, представленным в виде систем линейных и нелинейных уравнений и неравенств.

Библиотека помогает устранить противоречия в данных, минимизируя отклонения от исходных значений при соблюдении всех заданных ограничений. Это особенно полезно в задачах, где данные поступают из разных источников и могут содержать ошибки измерений или несоответствия.

Применение в различных областях:

Промышленность: согласование данных в системах управления технологическими процессами.

Экономика и финансы: обработка данных в моделях прогнозирования и анализа.

Наука и инженерия: решение задач оптимизации и калибровки моделей.

Энергетика: балансировка данных в энергосистемах.

Гибкость и масштабируемость:

Библиотека поддерживает работу с большими объемами данных и может быть интегрирована в различные программные системы.

Она предоставляет инструменты для настройки и адаптации под конкретные задачи пользователя.

Примеры задач, где библиотека может быть полезна:

Балансировка данных в химических процессах (например, согласование материальных и энергетических балансов).

Корректировка данных в системах мониторинга (например, устранение противоречий в показаниях датчиков).

Оптимизация данных в моделях прогнозирования (например, согласование исторических данных с прогнозными значениями).

Библиотека является мощным инструментом для анализа, корректировки и оптимизации данных в условиях наличия ограничений, что делает её незаменимой в задачах, требующих высокой точности и согласованности данных.

В.2 ОСНОВНЫЕ ВОЗМОЖНОСТИ

Поддержка стационарных и динамических задач:

Стационарные задачи: работа с данными, которые не изменяются во времени (например, балансовые уравнения в статических системах).

Динамические задачи: обработка данных, которые изменяются во времени (например, временные ряды или процессы, описываемые дифференциальными уравнениями).

Работа с ограничениями:

Библиотека поддерживает ограничения в виде:

- **Линейных уравнений и неравенств** (например, $\mathbf{Ax} = \mathbf{b}$ или $\mathbf{Ax} \leq \mathbf{b}$).
- **Нелинейных уравнений и неравенств** (например, $\mathbf{F(x)} = \mathbf{0}$ или $\mathbf{g(x)} \leq \mathbf{0}$).

Библиотека разрешает использование дифференциальных и интегро-дифференциальных уравнений в качестве ограничений, но в форме разностных уравнений по выбранной пользователем вычислительной схеме.

Это позволяет учитывать физические, экономические или другие законы, которым должны соответствовать данные.

Минимизация отклонений:

Библиотека минимизирует отклонения скорректированных данных от исходных, используя различные метрики.

В.3 СИСТЕМНЫЕ ТРЕБОВАНИЯ

Для успешной компиляции и работы библиотеки, написанной на C++ и использующей библиотеку Eigen для численной оптимизации, необходимо учитывать следующие системные требования. Они зависят от объема данных, сложности задач и желаемой производительности.

Минимальные системные требования

Минимальные требования позволяют компилировать и запускать библиотеку на компьютерах с ограниченными ресурсами. Однако производительность может быть низкой при работе с большими объемами данных или сложными задачами.

- **Операционная система**:
  - Windows 7/8/10/11 (64-битная) или новее.
  - macOS 10.13 (High Sierra) или новее.
  - Linux: Ubuntu 18.04 LTS или совместимые дистрибутивы.
- **Процессор (CPU)**:

- Минимум: 2-ядерный процессор с тактовой частотой 1.8 ГГц (например, Intel Core i3 или аналогичный AMD).

- **Оперативная память (RAM)**:
  - Минимум: 4 ГБ (для небольших задач).
  - Рекомендуется: 8 ГБ для более комфортной работы.
- **Место на диске**:
  - Минимум: 500 МБ свободного места для компиляции и работы библиотеки.
  - Дополнительное место для хранения данных и результатов.
- **Компилятор C++**:
  - Поддерживаемые компиляторы:
    - Windows: Microsoft Visual Studio 2017 или новее (с поддержкой C++17).
    - macOS: Clang (устанавливается через Xcode Command Line Tools).
    - Linux: GCC 7 или новее.
- **Библиотека Eigen**:
  - Версия Eigen 3.3 или новее.
  - Eigen — это заголовочная библиотека, поэтому её не нужно компилировать отдельно. Достаточно указать путь к заголовочным файлам.
- **Дополнительные зависимости**:
  - Если библиотека использует дополнительные зависимости (например, OpenMP для параллельных вычислений), убедитесь, что они установлены.

Рекомендуемые системные требования

Рекомендуемые требования обеспечивают высокую производительность при работе с большими объемами данных, сложными моделями и динамическими задачами.

- **Операционная система**:
  - Windows 10/11 (64-битная) или новее.
- **Процессор (CPU)**:
  - Рекомендуется: 4-ядерный процессор с тактовой частотой 2.5 ГГц или выше (например, Intel Core i5/i7, AMD Ryzen 5/7).
- **Оперативная память (RAM)**:
  - Рекомендуется: 16 ГБ или больше (для работы с большими наборами данных и сложными задачами).
- **Место на диске**:

- Рекомендуется: SSD с 1 ГБ свободного места для компиляции и работы библиотеки.
- Дополнительное место для хранения данных и результатов (в зависимости от объема данных).

- **Компилятор C++**:
  - Поддерживаемые компиляторы:
    - Windows: Microsoft Visual Studio 2019 или новее (с поддержкой C++17).
- **Библиотека Eigen**:
  - Версия Eigen 3.4 или новее.

Пример компиляции с использованием CMake

Создайте файл CMakeLists.txt для сборки библиотеки (cmake):

```
cmake_minimum_required(VERSION 3.10)
project(DataReconcile)
set(CMAKE_CXX_STANDARD 17)
# Добавление Eigen
find_package(Eigen3 REQUIRED)
# Добавление исходных файлов
add_executable(DataReconcile main.cpp reconcile.cpp)
```

Добавление в проект требуемого модуля из библиотеки DataReconcile

Все реализованные алгоритмы хранятся в формате компилируемых .cpp файлов, названия которых совпадают с названиями искомых функций. Способ добавления реализованного функционала в собственный проект остается на усмотрение пользователя: допустима добавление файлов с требуемыми функциями .cpp директивой `#include`, либо создание файла заголовков .h с перечислением тех функций библиотеки DataReconcile, которые пользователь желает использовать в своем проекте.

В.4 ПОШАГОВАЯ ИНСТРУКЦИЯ ПО РАЗВЕРТЫВАНИЮ БИБЛИОТЕКИ DATARECONCILE (C++)

1. Если ранее этого не было сделано, скачайте и установите компилятор C++ версии 11 или новее.

2. Скачайте (А) файлы библиотеки DataReconcile, либо клонируйте (Б) на вычислительную машину, где планируется сборка проекта, использующего функционал библиотеки.

2.А Как скачать содержимое репозитория DataReconcile.

– Перейдите на главную страницу репозитория по ссылке https://github.com/scientific-soft/DataReconcile

– Кликните на кнопку «<>», расположенную над списком содержащихся в корневой директории фалов и папок библиотеки (рисунок В.1).

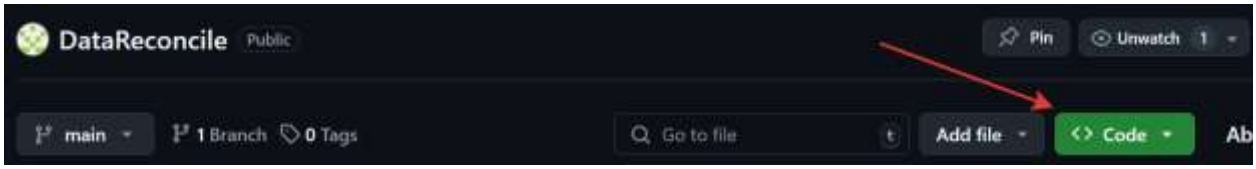

Рисунок В.1 – К развертыванию библиотеки DataReconcile

– Во всплывающем меню выберите вкладку «Local» (1), после чего кликнете на строку «Download ZIP» (2) (рисунок В.2).

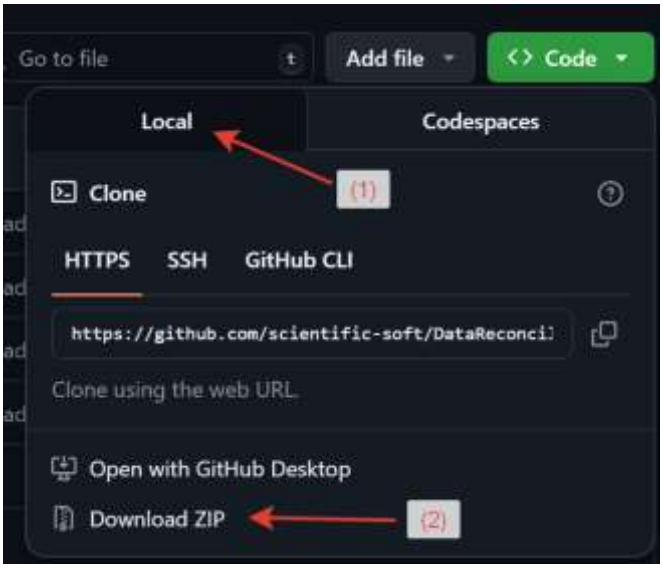

Рисунок В.2 – К развертыванию библиотеки DataReconcile

– После выполнения данных операций содержимое библиотеки начнет скачиваться в директорию загрузки на локальном компьютере в виде файла с расширением «.zip».

– После того, как скачивание завершено, скаченный архивный файл «DataReconcile-main.zip» необходимо разархивировать программой архиватором, поддерживающей формат «.zip».

– Разархивация с использованием свободно распространяемого архиватора 7-Zip (рисунок В.3).

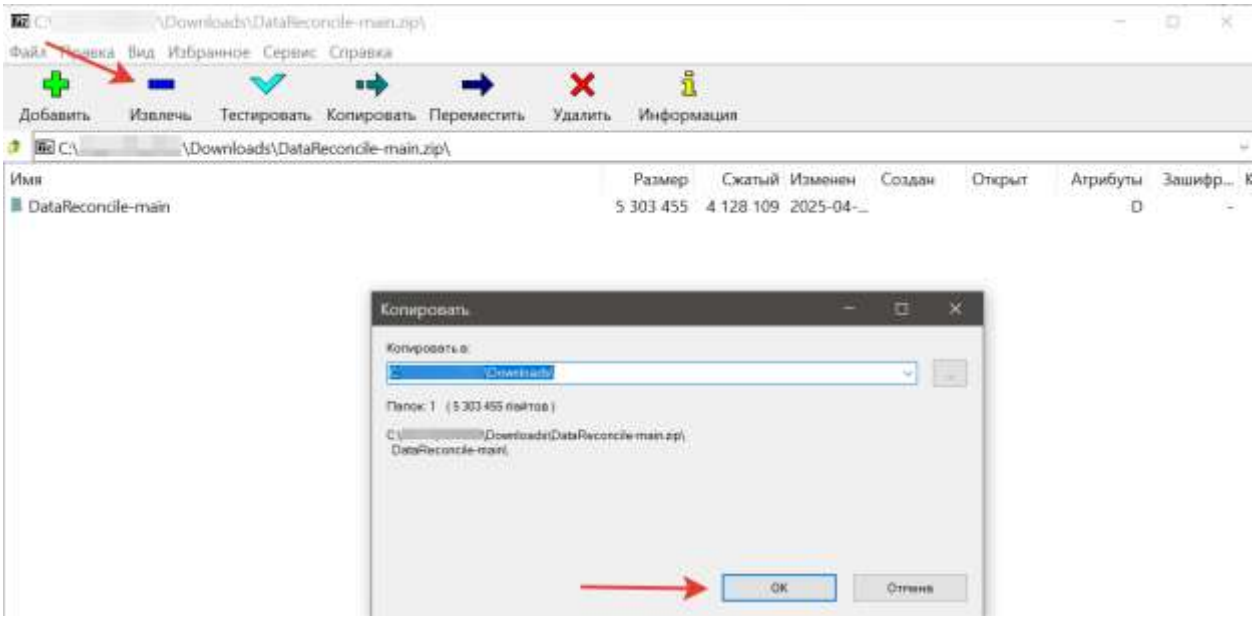

Рисунок В.3 – К развертыванию библиотеки DataReconcile

– Результат скачивания файлов библиотеки – папка с именем DataReconcile-main, содержимое которой приведено на скриншоте ниже (рисунок В.4).

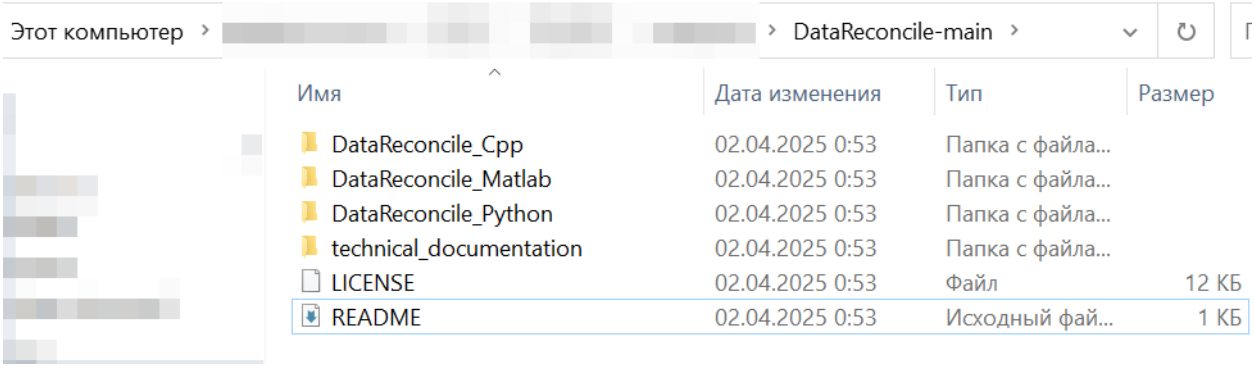

Рисунок В.4 – К развертыванию библиотеки DataReconcile

2.Б Как клонировать содержимое репозитория DataReconcile.

– Перейдите на главную страницу репозитория по ссылке https://github.com/scientific-soft/DataReconcile

– Кликните на кнопку «<>», расположенную над списком содержащихся в корневой директории фалов и папок библиотеки (рисунок В.5).

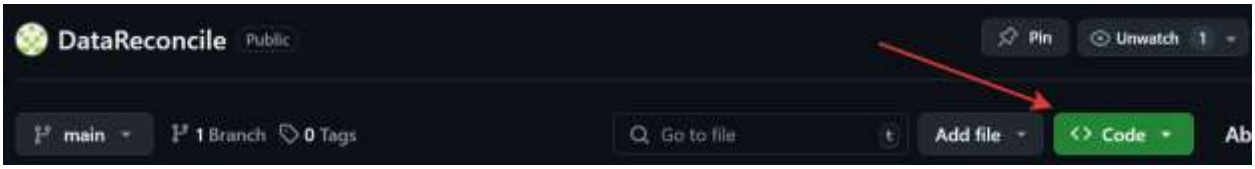

Рисунок В.5 – К развертыванию библиотеки DataReconcile

– Скопируйте URL репозитория в буфер обмена:

1) Чтобы клонировать репозиторий с помощью HTTPS, нажмите кнопку копирования в разделе «HTTPS».

2) Чтобы клонировать репозиторий с помощью SSH ключа, включая сертификат, выданный центром сертификации SSH вашей организации, нажмите кнопку копирования в разделе SSH.

3) Чтобы клонировать репозиторий с помощью GitHub CLI, нажмите кнопку копирования в разделе GitHub CLI (рисунок В.6).

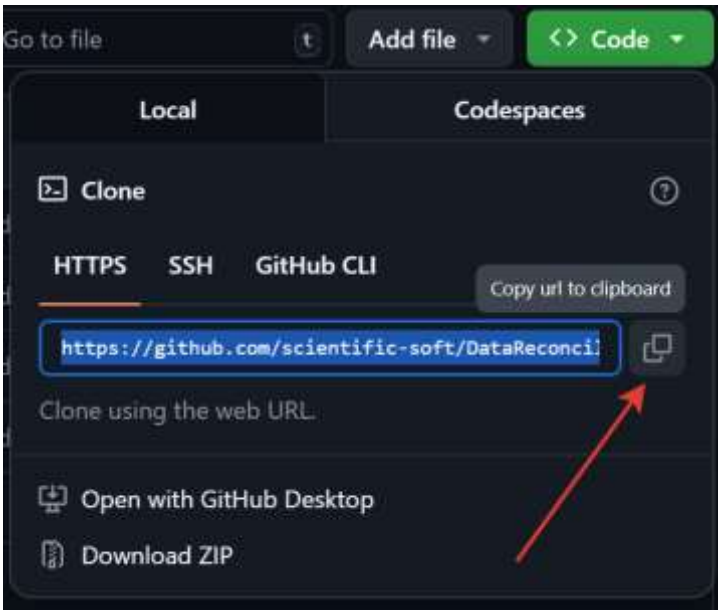

Рисунок В.6 – К развертыванию библиотеки DataReconcile

– Откройте консоль Git Bash.

– Измените текущий рабочий каталог на место, куда вы хотите клонировать содержимое DataReconcile.

– Введите git clone, а затем вставьте URL-адрес, скопированный ранее в формате

git clone https://github.com/scientific-soft/DataReconcile.git

– Нажмите Enter, чтобы создать локальный клон библиотеки.

3. Установите зависимую библиотеку (Eigen) (в консоли bash)

vcpkg install eigen3

4. Рекомендуется установить среду разработки, поддерживающую компилятор C++.

5. Интегрируйте библиотеку в свой проект.

Вариант А: Прямое включение

Скопируйте файлы .cpp и .h в свой проект.

Пример структуры:

your_project/

├── DataReconcileFiles/ # Скопированные файлы .cpp/.h

├── main.cpp # Ваш код

└── CMakeLists.txt # Конфигурация сборки

Примечание: DataReconcileFiles – условное обозначение файлов библиотеки, добавляемых в проект.

Включите в свой код (например, main.cpp):

```
#include "DataReconcile/DRnonparamEq.h"  // Example header
#include <Eigen/Dense>

int main() {
    Eigen::MatrixXd data(3, 10);
    // ... use DataReconcile functions ...
    return 0;
}
```

5.3 Скомпилируйте вручную (задайте свой путь при необходимости):

(консоль bash, пример)

```
g++ -std=c++11 -I/path/to/eigen -I./DataReconcile main.cpp DataReconcile/*.cpp -o my_app
```

Вариант B: CMake

Создайте CMakeLists.txt (cmake):

```
cmake_minimum_required(VERSION 3.10)
project(MyProject)
# Find Eigen (replace path if Eigen is not in default locations)
find_package(Eigen3 REQUIRED)
# Add DataReconcile sources
file(GLOB RECONCILE_SOURCES " DataReconcileFile/*.cpp")
# Build your executable
add_executable(my_app main.cpp ${RECONCILE_SOURCES})
target_link_libraries(my_app Eigen3::Eigen)
target_include_directories(my_app PRIVATE "DataReconcile")
```

Примечание: DataReconcileFile – условное обозначение файла библиотеки, добавляемых в проект.

Сборка и запуск:

(консоль bash, пример)

```
mkdir build && cd build
```

cmake .. && make

./my_app

6. Все доступные функции для согласования данных, описанные в технической документации, по названию совпадают с названием файлов с расширением «.cpp». Если возникает ошибка вызова функции из библиотеки, проверьте, действительно в директории по указанному пути присутствует файл с названием функции.

В.5 ОСНОВНЫЕ ФУНКЦИИ И API

В.5.1 Описание ключевых модулей

Решаемая в рамках программной библиотеки задача относится к задачам повышения точности результатов совместных измерений, выполняемых на сложном объекте измерений, для которого предоставлена или получена в ходе эмпирических исследований математическая модель (возможно, не вполне точная). Снижение неопределенности получаемых результатов достигается за счет учета функциональных зависимостей между измеряемыми величинами, формализованных в рамках упомянутой математической модели. В качестве последней могут выступать как различные уравнения (алгебраические – линейные и нелинейные, дифференциальные и интегро-дифференциальные), а также наборы ограничений, вытекающие из условий решаемой измерительной задачи и условий функционирования и эксплуатации наблюдаемого объекта измерений (системы неравенств – односторонних и двусторонних). Решение задачи производится также при разном объеме известной информации, ключевое место среди которой занимают сведения о распределении погрешностей выполняемых измерений – как известно из математической статистики, при разных законах распределения случайных погрешностей разные числовые оценки для одного и того же параметра распределений будут являться эффективными (то есть будут обеспечивать наименьший статистический разброс от выборки к выборке). Данное обстоятельство указывает на обязательную необходимость привлечения сведений о распределении погрешностей (в том объеме, в котором она доступна). Решение задачи повышения точности (за счет согласования индустриальных данных) под перечисленными ограничениями представляет собой комплекс задач, решения которых имеет как общие черты, так и разнится в зависимости от типа ограничений. Эффективные вычислительные решения для разных значимых для измерительной практики ситуаций представляет собой основу настоящей библиотеки. На первом этапе основным типом рассматриваемых ограничений выступают математические модели, составленные из алгебраических уравнений (как линейных, так и нелинейных) при условии, что закон распределения

случайных погрешностей отличается от нормального (возможно, несильно). Данный тип задачи должен включать в себя традиционные решения (в предположении гауссовости распределения погрешностей), так и новые, до сего момента не представленные в научной литературе по вопросу методы и подходы. Другим важным моментом является необходимость учета режима выполняемых измерений – статического (когда изменениями во времени значений измеряемых величин можно пренебречь без значимого снижения точности) или динамического (когда изменения сигналов измерительной информации во времени приводят к возникновению значимых составляющих погрешностей, отсутствие учета которых приводит к существенному снижению достоверности конечных результатов). Данная постановка задачи крайне важна для измерительной практики и предложена впервые в контексте задачи согласования индустриальных данных.

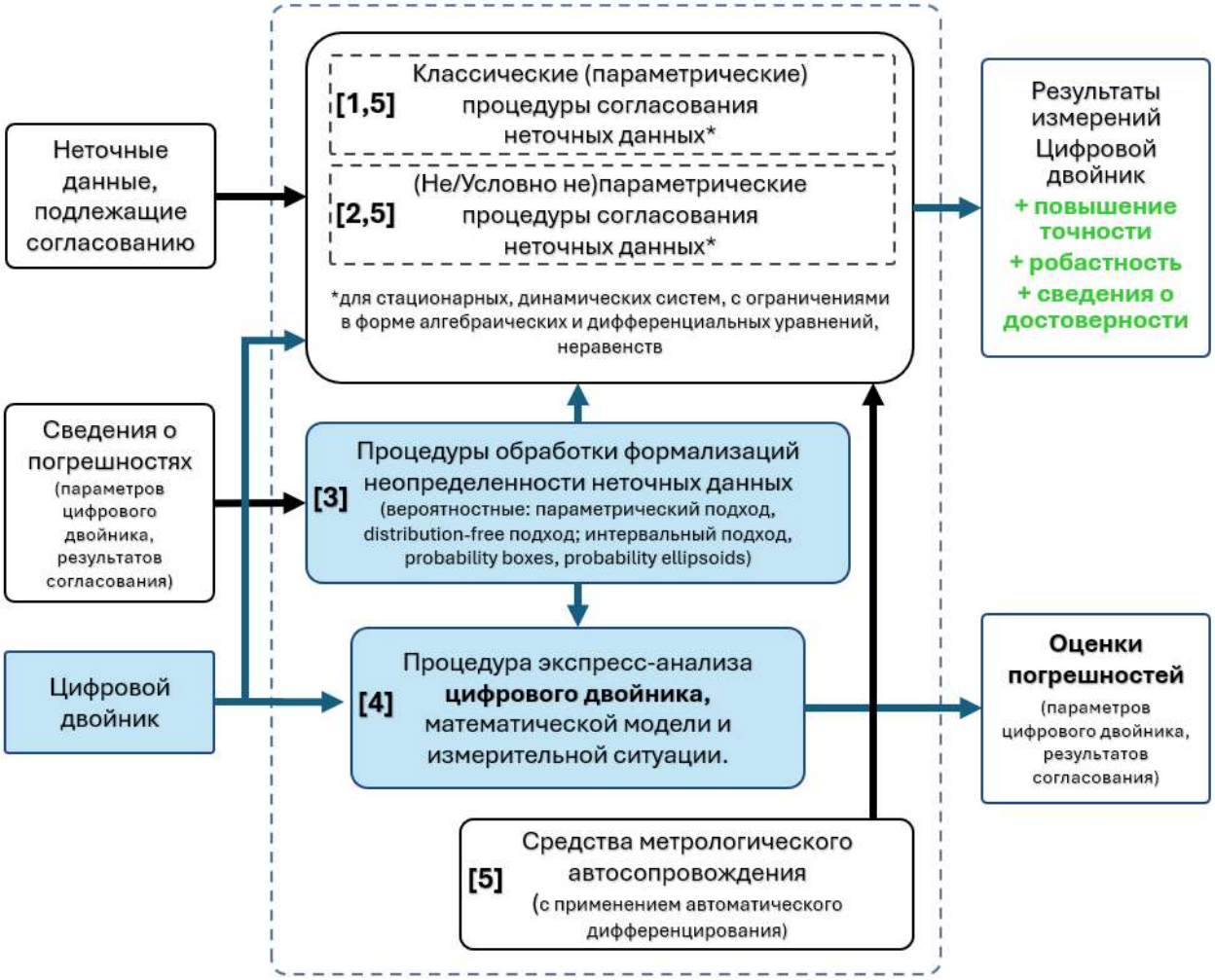

Рисунок В.7 – Структура библиотеки DataReconcile

Рассматривая задачу согласования индустриальных данных целостно, следует отметить, что задача так или иначе сводится к условной оптимизационной задаче некоторой размерности и с тем или иным набором условий. Данное обстоятельство позволяет

построить эффективную модель решения задачи согласования данных, выбрав для того или иного набора ограничений наиболее подходящий алгоритм оптимизации. Программная библиотека, реализующей разные методы согласования неточных данных друг с другом, также содержит в себе ряд автоматизированных инструментов.

На рисунке В.7 ниже представлена визуализация структуры библиотеки, а также входные и выходные данные.

В соответствии с приведенной на рисунке нумерацией, библиотека содержит:

1) программные средства, реализующие традиционные методы согласования индустриальных данных, которые содержатся и применяются в зарубежных коммерческих продуктах,

2) программные средства, реализующие непараметрические и условно непараметрические методы согласования индустриальных данных, позволяющие учитывать в рамках согласования существенные или малые отклонения действительного закона распределения погрешностей от нормального, применимые как в статическом режиме измерения, так и в динамическом,

3) программные представления и преобразования современных формализмов описания погрешности результатов измерений, таковых, чтобы результат преобразования/представления мог быть эффективно использован в качестве мера неопределенности неточных индустриальных данных при их согласовании по известной модели.

4) программные предварительной экспресс-оценки потенциального уточнения, достигаемого при математической обработке результатов выполняемых измерений за счет выполнения их согласования на основе известной априорной информации,

5) программные средства для обеспечения метрологического автосопровождения производимых вычислений [2-4], поскольку без него нет возможности оценить точность конечных результатов согласования, а также – при необходимости – произвести согласование моментов остановки выполняемых итерационных процедур с точностью исходных данных (то есть результатов выполненных измерений).

В.5.2 Доступные функции, их параметры и возвращаемые значения

Презентуемая библиотека оформлена в виде независимых cpp-файлов, для работы с которым требуется компилятор языка программирования C++ версии не старше 16.0,

включающий библиотеки `vector, complex, cmath, algorithm, numeric, functional, stdexcept, stdexcept, numeric, functional, algorithm, limits, stdexcept, Eigen`. Ниже приведено описание ключевых процедур для не/полу/параметрического согласования данных и выполнения метрологического анализа потенциальных результатов согласования.

Для каждого параметра `msrd_data` функции каждой функции справедливо следующее: в случае согласования результатов однократного совестного измерения нескольких взаимосвязанных величин, согласуемые результаты нужно передавать в формате вектор-столбца формата `MatrixXd` класса библиотеки Eigen; если согласованию подлежит массив многократных совместных измерений, согласуемые значения нужно передавать в виде матрицы `msrd_data`, где каждый столбец – набор однократного совместного измерения всех согласуемых величин (таким образом в *j*-ой строке предаваемой матрицы `msrd_data` располагаются все результаты измерений *j*-ой измеряемой физической величины).

Презентуемая библиотека оформлена в виде независимых cpp-файлов, для работы с которым требуется компилятор языка программирования C++ версии 18.0 или новее, включающий, помимо стандартных, библиотеку Eigen. Ниже приведено описание ключевых процедур для не/полу/параметрического согласования данных и выполнения метрологического анализа потенциальных результатов согласования.

Для каждого параметра `msrd_data` функции каждой функции справедливо следующее: в случае согласования результатов однократного совестного измерения нескольких взаимосвязанных величин, согласуемые результаты нужно передавать в формате вектор-столбца; если согласованию подлежит массив многократных совместных измерений, согласуемые значения нужно передавать в виде матрицы `msrd_data`, где каждый столбец – набор однократного совместного измерения всех согласуемых величин (таким образом в *j*-ой строке предаваемой матрицы `msrd_data` располагаются все результаты измерений *j*-ой измеряемой физической величины).

Процедура оценки степени потенциального уточнения совместных измерений за счет согласования

`estAccuracyIncreaseByDR`

Вызов

```
auto result = estAccuracyIncreaseByDR(func, ChosenMode,
vals_for_reconc, no_error_params, errors)
```

Описание	Функция выполняет приближенную оценку потенциального уточнения совместных измерений, достигаемого за счет учета известных функциональных взаимосвязей между измеряемыми величинами, на основе локальной линеаризации модели и метода декомпозиции алгоритма условной оптимизации.
Аргументы	`func` – указатель на функцию (`const function<VectorXd(const VectorXd&, const VectorXd&)>&`), возвращающую значения уравнений, формализующих взаимосвязь между измеряемыми величинами. `ChosenMode` – строка (`const string&`), позволяющей выбрать в каком режиме работает функция: по методу наименьших квадратов `'LS'` или по обобщенному методу наименьших квадратов `'WLS'`, `vals_for_reconc` – одномерный массив (`const VectorXd&`), содержащий результаты совместных измерений, подлежащих уточнению; `no_error_params` – одномерный массив (`const VectorXd&`) значений параметров уравнений, описывающих зависимости между измеряемыми величинами. `errors` – массив (`const VectorXd&`), содержащий меры погрешностей согласуемых результатов измерений и параметров модели, используемой для согласования.
Возвращаемое значение	`result` – пара одномерных массивов [одномерный массив оценок степени повышения точности, которое может быть достигнуто за счет процедуры согласования, одномерный массив оценок дисперсий результатов согласования].

Функции согласования, учитывающего зависимости между взаимосвязанными величинами, выраженные в виде систем уравнений

Классическая процедура согласования совместных измерений в предположении, что случайные погрешности распределены нормально

`DRparamEq`

Вызов

```
VectorXd reconciled_data = GaussDataReconcil(
            depend_func,
            msrd_data, error_params, params);
```

Описание	Функция выполняет параметрическое согласование совместно измеренных величин, закон распределения случайных погрешностей которых соответствует нормальному распределению вероятностей.

Аргументы

`depend_func` – указатель на функцию (`DependFunc`), возвращающую результат вычисления левых частей системы уравнений вида $\mathbf{f}_M(\mathbf{x}, \mathbf{a}) = \mathbf{0}$, описывающую взаимосвязи между согласуемыми величинами,
`msrd_data` – массив (`const MatrixXd&`), содержащий согласуемые результаты измерений,
`error_params` – одномерный массив (`VectorXd&`), содержащий меры погрешности независимо полученных результатов измерений, подлежащих согласованию,
`params` – одномерный массив (`const VectorXd&`) параметров модели взаимосвязей.

Возвращаемое значение

`reconciled_data` – одномерный массив (`VectorXd`) результатов параметрического согласования совместных измерений, выполненного при следующих допущениях: случайные погрешности результатов совместных измерений распределены нормально; согласуемые результаты измерений получены независимо, и, следовательно, корреляция между случайными погрешностями отсутствует

Процедура семи-непараметрического согласования с представлением неизвестного закона распределения погрешностей рядом Грама-Шарлье

`DRsemiparamEq`

Вызов

```
VectorXd reconciled_data = DRparamEq(depend_func,
      msrd_data, error_params, alpha_params, params)
```

Описание

Файл содержит одноименную вызываемую функцию, которая выполняет непараметрическое согласование совместно измеренных величин, закон распределения которых оценивается проекционным методом с применением в качестве модели усеченного ряда Грамма-Шарлье.

Аргументы

`depend_func` – указатель на функцию (`const function<VectorXd(const VectorXd&, const VectorXd&)>&`), возвращающую значения левой части уравнений взаимосвязи между измеряемыми величинами;
`msrd_data` – массив (`const MatrixXd&`), значения результатов совместного измерения величин, подлежащих согласованию;
`error_params` – одномерный массив (`vector<double>&`), содержащий априорно заданные или оцененных меры погрешностей результатов измерений;
`alpha_params` – двухмерный массив (`const MatrixXd&`), содержащий оценки среднеквадратического отклонения результатов измерений, коэффициентов асимметрии, коэффициентов и эксцесса случайных отклонений результатов измерений,
`params` – одномерный массив (`const vector<double>&`), содержащий априорно заданные параметры модели, описывающей зависимости между согласуемыми величинами.

Возвращаемое значение	`reconciled_data` – одномерный массив (`VectorXd`) результатов семи-непараметрического согласования совместных измерений.

Процедура непараметрического согласования с ядерной аппроксимацией распределения случайных погрешностей

`DRnonparamEq`

Вызов	`VectorXd reconciled_data = DRnonparamEq(`
	`depend_func,`
	`msrd_data, model_params, prior_vars, bandwidths);`

Описание	Функция выполняет непараметрическое согласование совместно измеренных величин, закон распределения которых оценивается методом ядерной аппроксимации с использованием ядра Гаусса.
Аргументы	`func` – указатель на функцию (`const function<VectorXd(const VectorXd&, const VectorXd&)>&`), возвращающую значения левой части уравнений вида $\mathbf{f}_M(\mathbf{x}, \mathbf{a}) = \mathbf{0}$, описывающих зависимости между измеряемыми величинами **x** и параметрами **a** модели; `msrd_data` – одномерный массив (`const MatrixXd&`), значения результатов однократного совместного измерения величин, подлежащих согласованию; `model_params` – одномерный массив (`const VectorXd&`) параметров модели, описывающих зависимости между согласуемыми величинами, заданных точно; `prior_params` – одномерный массив (`const VectorXd&`), содержащий априорно заданные или оцененных дисперсии результатов измерений; `bandwidth` – одномерный массив (`const VectorXd&`), содержащий регуляризирующее условие на результат ядерной аппроксимации неизвестного распределения случайных погрешностей согласуемых измерений. Данные значения являются ширинами окон ядерной аппроксимации. В качестве аппроксимирующего ядра использовано ядро Гаусса.
Возвращаемое значение	`reconciled_data` – одномерный массив (`VectorXd`) результатов непараметрического согласования результатов измерений взаимосвязанных величин. Предполагается, что измерения выполнялись независимо, и корреляция между случайными погрешностями отсутствует. В качестве процедуры идентификации неизвестного распределения случайных погрешностей согласуемых измерений применен метод ядерной аппроксимации ядром Гаусса.

Классическая процедура робастного согласования совместных измерений в предположении, что случайные погрешности распределены нормально

`DRparamEqRobust`

Вызов	`VectorXd reconciled_data = DRparamEqRobust(depend_func, msrd_data, error_params, params)`
Описание	Функция выполняет параметрическое согласование совместно измеренных величин, закон распределения случайных погрешностей которых соответствует нормальному распределению вероятностей.
Аргументы	`depend_func` – указатель на функцию (`DependFunc`), возвращающую результат вычисления левых частей системы уравнений вида $\mathbf{f}_M(\mathbf{x}, \mathbf{a}) = \mathbf{0}$, описывающую взаимосвязи между согласуемыми величинами, `msrd_data` – массив (`const MatrixXd&`), содержащий согласуемые результаты измерений, `error_params` – одномерный массив (`VectorXd&`), содержащий меры погрешности независимо полученных результатов измерений, подлежащих согласованию, `params` – одномерный массив (`const VectorXd&`) параметров модели взаимосвязей.
Возвращаемое значение	`reconciled_data` – одномерный массив (`VectorXd`) результатов параметрического согласования совместных измерений, выполненного при следующих допущениях: случайные погрешности результатов совместных измерений распределены нормально; согласуемые результаты измерений получены независимо, и, следовательно, корреляция между случайными погрешностями отсутствует

Процедура семи-непараметрического робастного согласования с представлением неизвестного закона распределения погрешностей рядом Грама-Шарлье

`DRsemiparamEqRobust`

Вызов	`VectorXd reconciled_data = DRsemiparamEqRobust(`
	`depend_func,`
	`msrd_data, error_params, alpha_params, params);`

Описание	Файл содержит одноименную вызываемую функцию, которая выполняет непараметрическое согласование совместно измеренных величин, закон распределения которых оценивается проекционным методом с применением в качестве модели усеченного ряда Грамма-Шарлье.
Аргументы	`depend_func` – указатель на функцию (`const function<VectorXd(const VectorXd&, const VectorXd&)>&`), возвращающую значения левой части уравнений взаимосвязи между измеряемыми величинами; `msrd_data` – массив (`const MatrixXd&`), значения результатов совместного измерения величин, подлежащих согласованию;

`error_params` – одномерный массив (`vector<double>&`), содержащий априорно заданные или оцененных меры погрешностей результатов измерений;
`alpha_params` – двухмерный массив (`const MatrixXd&`), содержащий оценки среднеквадратического отклонения результатов измерений, коэффициентов асимметрии, коэффициентов и эксцесса случайных отклонений результатов измерений,
`params` – одномерный массив (`VectorXd`), содержащий априорно заданные параметры модели, описывающей зависимости между согласуемыми величинами.

Возвращаемое значение — `reconciled_data` – одномерный массив (`ndarray`) результатов семи-непараметрического согласования совместных измерений.

Процедура непараметрического робастного согласования с ядерной аппроксимацией распределения случайных погрешностей

`DRnonparamEqRobust`

Вызов

```
VectorXd reconciled_data = DRnonparamEqRobust(
     depend_func,
     msrd_data, model_params, prior_vars, bandwidths)
```

Описание — Функция выполняет непараметрическое согласование совместно измеренных величин, закон распределения которых оценивается методом ядерной аппроксимации с использованием ядра Гаусса.

Аргументы

`depend_func` – указатель на функцию (`const function<VectorXd(const VectorXd&, const VectorXd&)>&`), возвращающую значения левой части уравнений вида $\mathbf{f}_M(\mathbf{x}, \mathbf{a}) = \mathbf{0}$, описывающих зависимости между измеряемыми величинами **x** и параметрами **a** модели;
`msrd_data` – одномерный массив (`const MatrixXd&`), значения результатов однократного совместного измерения величин, подлежащих согласованию;
`model_params` – одномерный массив (`const VectorXd&`) параметров модели, описывающих зависимости между согласуемыми величинами, заданных точно;
`prior_params` – одномерный массив (`VectorXd&`), содержащий априорно заданные или оцененных дисперсии результатов измерений;
`bandwidth` – одномерный массив (`const VectorXd&`), содержащий регуляризирующее условие на результат ядерной аппроксимации неизвестного распределения случайных погрешностей согласуемых измерений. Данные значения являются ширинами окон ядерной аппроксимации. В качестве аппроксимирующего ядра использовано ядро Гаусса.

Возвращаемое значение — `reconciled_data` – одномерный массив (`VectorXd`) результатов непараметрического согласования результатов измерений

взаимосвязанных величин. Предполагается, что измерения выполнялись независимо, и корреляция между случайными погрешностями отсутствует. В качестве процедуры идентификации неизвестного распределения случайных погрешностей согласуемых измерений применен метод ядерной аппроксимации ядром Гаусса.

Функции согласования, учитывающего зависимости между взаимосвязанными величинами, выраженные в виде систем уравнений и неравенств

Классическая процедура согласования совместных измерений в предположении, что случайные погрешности распределены нормально

`DRparamEqIneq`

Вызов	`VectorXd reconciled_data = DRparamEqIneq(eqies_model, ineqies_model, msrd_data, error_params, params)`
Описание	Функция выполняет параметрическое согласование совместно измеренных величин, закон распределения случайных погрешностей которых соответствует нормальному распределению вероятностей. Подлежат учету зависимости в виде равенств и неравенств.
Аргументы	`eqies_model` – функция (`const std::function<VectorXc(const VectorXc&, const VectorXd&)>&`), возвращающую результат вычисления левых частей системы уравнений вида $\mathbf{f}_M(\mathbf{x}, \mathbf{a}) = \mathbf{0}$, описывающую взаимосвязи между согласуемыми величинами (размерность возвращаемого вектора соответствует размерности вектор-функции $\mathbf{f}_M$), `ineqies_model` – функция (`const std::function<VectorXc(const VectorXc&, const VectorXd&)>&`), возвращающую результат вычисления левых частей системы неравенств вида $\mathbf{g}_M(\mathbf{x}, \mathbf{a}) \geq \mathbf{0}$, представленных в формате равенств $\mathbf{g}_M(\mathbf{x}, \mathbf{a}) = \mathbf{0}$ (в соответствии с методом множителей Лагранжа), описывающую взаимосвязи (ограничения) между согласуемыми величинами, `msrd_data` – массив (`const MatrixXd&`), содержащий согласуемые результаты измерений, `error_params` – одномерный массив (`const VectorXd&`), содержащий меры погрешности независимо полученных результатов измерений, подлежащих согласованию, `params` – одномерный массив (`const VectorXd&`) параметров модели взаимосвязей.
Возвращаемое значение	`reconciled_data` – одномерный массив (`VectorXd`) результатов параметрического согласования совместных измерений, выполненного при следующих допущениях: случайные погрешности результатов совместных измерений распределены нормально; согласуемые результаты измерений получены независимо, и, следовательно, корреляция между случайными погрешностями отсутствует

Процедура семи-непараметрического согласования с представлением неизвестного закона распределения погрешностей рядом Грама-Шарлье

`DRsemiparamEqIneq`

Вызов	`VectorXd reconciled_data = DRsemiparamEqIneq(`
	`  eqies_model, ineqies_model,`
	`  msrd_data, error_params, alpha_params, params_)`

Описание	Файл содержит одноименную вызываемую функцию, которая выполняет непараметрическое согласование совместно измеренных величин, закон распределения которых оценивается проекционным методом с применением в качестве модели усеченного ряда Грамма-Шарлье. Подлежат учету зависимости в виде равенств и неравенств.
Аргументы	`eqies_model` – функция (`const std::function<VectorXc(const VectorXc&, const VectorXd&)>&`), возвращающую результат вычисления левых частей системы уравнений вида $\mathbf{f}_M(\mathbf{x}, \mathbf{a}) = \mathbf{0}$, описывающую взаимосвязи между согласуемыми величинами (размерность возвращаемого вектора соответствует размерности вектор-функции $\mathbf{f}_M$), `ineqies_model` – функция (`const std::function<VectorXc(const VectorXc&, const VectorXd&)>&`), возвращающую результат вычисления левых частей системы неравенств вида $\mathbf{g}_M(\mathbf{x}, \mathbf{a}) \geq \mathbf{0}$, представленных в формате равенств $\mathbf{g}_M(\mathbf{x}, \mathbf{a}) = \mathbf{0}$ (в соответствии с методом множителей Лагранжа), описывающую взаимосвязи (ограничения) между согласуемыми величинами, `msrd_data` – массив (`const MatrixXd&`), значения результатов совместного измерения величин, подлежащих согласованию; `error_params` – одномерный массив (`const VectorXd&`), содержащий априорно заданные или оцененных меры погрешностей результатов измерений; `alpha_params` – двухмерный массив (`const VectorXd&`), содержащий оценки среднеквадратического отклонения результатов измерений, коэффициентов асимметрии, коэффициентов и эксцесса случайных отклонений результатов измерений, `params` – одномерный массив (`const VectorXd&`), содержащий априорно заданные параметры модели, описывающей зависимости между согласуемыми величинами.
Возвращаемое значение	`reconciled_data` – одномерный массив (`VectorXd`) результатов семи-непараметрического согласования совместных измерений.

Процедура непараметрического согласования с ядерной аппроксимацией распределения случайных погрешностей

`DRnonparamEqIneq`

Вызов	`VectorXd reconciled_data = DRnonparamEqIneq(`
	`  eqies_model, ineqies_model,`
	`  msrd_data, model_params, error_params, bandwidths)`

Описание	Функция выполняет непараметрическое согласование совместно измеренных величин, закон распределения которых оценивается методом ядерной аппроксимации с использованием ядра Гаусса. Подлежат учету зависимости в виде равенств и неравенств.
Аргументы	`eqies_model` – функция (`const VectorXc&, const VectorXd&)>&`), возвращающую результат вычисления левых частей системы уравнений вида $\mathbf{f}_M(\mathbf{x}, \mathbf{a}) = \mathbf{0}$, описывающую взаимосвязи между согласуемыми величинами (размерность возвращаемого вектора соответствует размерности вектор-функции $\mathbf{f}_M$), `ineqies_model` – функция (`const VectorXc&, const VectorXd&)>&`), возвращающую результат вычисления левых частей системы неравенств вида $\mathbf{g}_M(\mathbf{x}, \mathbf{a}) \geq \mathbf{0}$, представленных в формате равенств $\mathbf{g}_M(\mathbf{x}, \mathbf{a}) = \mathbf{0}$ (в соответствии с методом множителей Лагранжа), описывающую взаимосвязи (ограничения) между согласуемыми величинами; `msrd_data` – одномерный массив (`const MatrixXd&`), значения результатов однократного совместного измерения величин, подлежащих согласованию; `model_params` – одномерный массив (`const VectorXd&`) параметров модели, описывающих зависимости между согласуемыми величинами, заданных точно; `prior_params` – одномерный массив (`const VectorXd&`), содержащий априорно заданные или оцененных дисперсии результатов измерений; `bandwidth` – одномерный массив (`const VectorXd&`), содержащий регуляризирующее условие на результат ядерной аппроксимации неизвестного распределения случайных погрешностей согласуемых измерений. Данные значения являются ширинами окон ядерной аппроксимации. В качестве аппроксимирующего ядра использовано ядро Гаусса.
Возвращаемое значение	`reconciled_data` – одномерный массив (`VectorXd`) результатов непараметрического согласования результатов измерений взаимосвязанных величин. Предполагается, что измерения выполнялись независимо, и корреляция между случайными погрешностями отсутствует. В качестве процедуры идентификации неизвестного распределения случайных погрешностей согласуемых измерений применен метод ядерной аппроксимации ядром Гаусса.

Классическая процедура робастного согласования совместных измерений в предположении, что случайные погрешности распределены нормально

`DRparamEqIneqRobust`

Вызов

```
VectorXd reconciled_data = DRparamEqIneqRobust(
    eqies_model, ineqies_model,
    msrd_data, error_params, params)
```

Описание	Функция выполняет параметрическое согласование совместно измеренных величин, закон распределения случайных погрешностей которых соответствует нормальному распределению вероятностей. Подлежат учету зависимости в виде равенств и неравенств.
Аргументы	`eqies_model` – функция (`const VectorXc&, const VectorXd&)>&`), возвращающую результат вычисления левых частей системы уравнений вида $\mathbf{f}_M(\mathbf{x}, \mathbf{a}) = \mathbf{0}$, описывающую взаимосвязи между согласуемыми величинами (размерность возвращаемого вектора соответствует размерности вектор-функции $\mathbf{f}_M$), `ineqies_model` – функция (`const VectorXc&, const VectorXd&)>&`), возвращающую результат вычисления левых частей системы неравенств вида $\mathbf{g}_M(\mathbf{x}, \mathbf{a}) \geq \mathbf{0}$, представленных в формате равенств $\mathbf{g}_M(\mathbf{x}, \mathbf{a}) = \mathbf{0}$ (в соответствии с методом множителей Лагранжа), описывающую взаимосвязи (ограничения) между согласуемыми величинами; `msrd_data` – массив (`const MatrixXd&`), содержащий согласуемые результаты измерений, `error_params` – одномерный массив (`const VectorXd&`), содержащий меры погрешности независимо полученных результатов измерений, подлежащих согласованию, `params` – одномерный массив (`const VectorXd&`) параметров модели взаимосвязей.
Возвращаемое значение	`reconciled_data` – одномерный массив (`VectorXd`) результатов параметрического согласования совместных измерений, выполненного при следующих допущениях: случайные погрешности результатов совместных измерений распределены нормально; согласуемые результаты измерений получены независимо, и, следовательно, корреляция между случайными погрешностями отсутствует

Процедура семи-непараметрического робастного согласования с представлением неизвестного закона распределения погрешностей рядом Грама-Шарлье

`DRsemiparamEqIneqRobust`

Вызов

```
VectorXd reconciled_data = DRsemiparamEqIneqRobust(
    eqies_model, ineqies_model,
    msrd_data, error_params, alpha_params, params)
```

Описание

Файл содержит одноименную вызываемую функцию, которая выполняет непараметрическое согласование совместно измеренных величин, закон распределения которых оценивается проекционным методом с применением в качестве модели усеченного ряда Грамма-Шарлье. Подлежат учету зависимости в виде равенств и неравенств.

Аргументы	`eqies_model` – функция (`const std::function<VectorXc(const VectorXc&, const VectorXd&)>&`), возвращающую результат вычисления левых частей системы уравнений вида $\mathbf{f}_M(\mathbf{x}, \mathbf{a}) = \mathbf{0}$, описывающую взаимосвязи между согласуемыми величинами (размерность возвращаемого вектора соответствует размерности вектор-функции $\mathbf{f}_M$), `ineqies_model` – функция (`const std::function<VectorXc(const VectorXc&, const VectorXd&)>&`), возвращающую результат вычисления левых частей системы неравенств вида $\mathbf{g}_M(\mathbf{x}, \mathbf{a}) \geq \mathbf{0}$, представленных в формате равенств $\mathbf{g}_M(\mathbf{x}, \mathbf{a}) = \mathbf{0}$ (в соответствии с методом множителей Лагранжа), описывающую взаимосвязи (ограничения) между согласуемыми величинами; `msrd_data` – массив (`const MatrixXd&`), значения результатов совместного измерения величин, подлежащих согласованию; `error_params` – одномерный массив (`const VectorXd&`), содержащий априорно заданные или оцененных меры погрешностей результатов измерений; `alpha_params` – двухмерный массив (`const VectorXd&`), содержащий оценки среднеквадратического отклонения результатов измерений, коэффициентов асимметрии, коэффициентов и эксцесса случайных отклонений результатов измерений, `params` – одномерный массив (`const VectorXd&`), содержащий априорно заданные параметры модели, описывающей зависимости между согласуемыми величинами.
Возвращаемое значение	`reconciled_data` – одномерный массив (`VectorXd`) результатов семи-непараметрического согласования совместных измерений.

Процедура непараметрического робастного согласования с ядерной аппроксимацией распределения случайных погрешностей

`DRnonparamEqIneqRobust`

Вызов	`VectorXd reconciled_data = DRnonparamEqIneqRobust(`
	`  eqies_model, ineqies_model,`
	`  msrd_data, model_params, prior_vars, bandwidths);`

Описание	Функция выполняет непараметрическое согласование совместно измеренных величин, закон распределения которых оценивается методом ядерной аппроксимации с использованием ядра Гаусса. Подлежат учету зависимости в виде равенств и неравенств.
Аргументы	`eqies_model` – функция (`const VectorXc&, const VectorXd&)>&`), возвращающую результат вычисления левых частей системы уравнений вида $\mathbf{f}_M(\mathbf{x}, \mathbf{a}) = \mathbf{0}$, описывающую взаимосвязи между согласуемыми величинами (размерность

возвращаемого вектора соответствует размерности вектор-функции $\mathbf{f}_M$),

`ineqies_model` – функция (`const VectorXc&, const VectorXd&)>&`), возвращающую результат вычисления левых частей системы неравенств вида $\mathbf{g}_M(\mathbf{x}, \mathbf{a}) \geq \mathbf{0}$, представленных в формате равенств $\mathbf{g}_M(\mathbf{x}, \mathbf{a}) = \mathbf{0}$ (в соответствии с методом множителей Лагранжа), описывающую взаимосвязи (ограничения) между согласуемыми величинами;

`msrd_data` – одномерный массив (`const MatrixXd&`), значения результатов однократного совместного измерения величин, подлежащих согласованию;

`model_params` – одномерный массив (`const VectorXd&`) параметров модели, описывающих зависимости между согласуемыми величинами, заданных точно;

`prior_params` – одномерный массив (`const VectorXd&`), содержащий априорно заданные или оцененных дисперсии результатов измерений;

`bandwidth` – одномерный массив (`const VectorXd&`), содержащий регуляризирующее условие на результат ядерной аппроксимации неизвестного распределения случайных погрешностей согласуемых измерений. Данные значения являются ширинами окон ядерной аппроксимации. В качестве аппроксимирующего ядра использовано ядро Гаусса.

Возвращаемое значение

`reconciled_data` – одномерный массив (`VectorXd`) результатов непараметрического согласования результатов измерений взаимосвязанных величин. Предполагается, что измерения выполнялись независимо, и корреляция между случайными погрешностями отсутствует. В качестве процедуры идентификации неизвестного распределения случайных погрешностей согласуемых измерений применен метод ядерной аппроксимации ядром Гаусса.

Конструктор объектов типа PBox, служащих для отображения множества возможных значений согласуемых величин и/или их неопределенности в форме области возможных значений функции распределения

PBox

Вызов

```
PBox(const std::vector<double>& x, const std::vector<double>& lowerCDF, const std::vector<double>& upperCDF)
```

Описание

Функция представляет собой конструктор класса типа PBox, служащего для отображения возможных значений и/или неопределенности согласуемых величин в виде области возможных значений функции распределения.

Аргументы

`x` – набор значений, отражающих узлы сетки дискретизации значений аргумента кумулятивной функции распределения (cdf);

	`lowerCDF` – набор значений нижней границы области допустимых значений функции распределения; `upperCDF` – набор значений верхней границы области допустимых значений функции распределения.
Возвращаемое значение	`this` – созданный объект типа PBox.

Конструктор объектов типа Hist, служащих для отображения множества возможных значений согласуемых величин и/или их неопределенности в форме интервальной гистограммы по Берлинту

`Hist`

Вызов	`Hist(const std::vector<double>& x, const std::vector<double>& lowerPDF, const std::vector<double>& upperPDF)`
Описание	Функция представляет собой конструктор класса типа Hist, служащего для отображения возможных значений и/или неопределенности согласуемых величин в виде гистограммы функции плотности распределения с интервальным заданием возможных значений высоты ее полос.
Аргументы	`x` – массив значений, последовательно содержащий границы полос гистограммы (pdf); `lowerPDF` – массив значений, отображающих нижние границы множества допустимых значений высоты полос гистограммы; `upperPDF` – массив значений, отображающих нижние границы множества допустимых значений высоты полос гистограммы.
Возвращаемое значение	`this` – созданный объект типа Hist.

Конструктор объектов типа DermpsterShafer, служащих для отображения множества возможных значений согласуемых величин и/или их неопределенности в форме структуры Демпстера-Шафера

`DempsterShafer`

Вызов	`DempsterShafer(const MatrixXd& intervals, const VectorXd& masses)`
Описание	Функция представляет собой конструктор класса типа DempsterShafer, служащего для отображения возможных значений и/или неопределенности согласуемых величин в виде структуры Демпстера-Шафера.

Аргументы	`intervals` – массив интервалов для фокальных элементов; `masses` – соответствующие им массы.
Возвращаемое значение	`this` – созданный объект типа DempsterShafer.

Конструктор объектов типа Fuzzy, служащих для отображения множества возможных значений согласуемых величин и/или их неопределенности в форме нечеткой переменной по Заде

`Fuzzy`

Вызов	`Fuzzy(const VectorXd& universe, const VectorXd& membership)`
Описание	Функция представляет собой конструктор класса типа Fuzzy, служащего для отображения возможных значений и/или неопределенности согласуемых величин в виде нечеткой переменной.
Аргументы	`universe` – значения носителя нечеткой переменной; `membership` – значения функции принадлежности (от 0 до 1).
Возвращаемое значение	`this` – созданный объект типа Fuzzy.

Конструктор объектов типа FuzzyInterval, служащих для отображения множества возможных значений согласуемых величин и/или их неопределенности в форме нечеткого интервала

`FuzzyInterval`

Вызов	`FuzzyInterval(const VectorXd& alphaLevels, const MatrixXd& intervals)`
Описание	Функция представляет собой конструктор класса типа FuzzyInterval, служащего для отображения возможных значений и/или неопределенности согласуемых величин в виде нечеткого интервала (включает в себя как предельный случай классический интервал, а исчисление нечетких интервалов – классическую интервальную арифметику по Муру).
Аргументы	`alphaLevels` – так называемые значения α-cut (уровни значений функции принадлежности); `intervals` – соответствующие вложенные интервалы (nested intervals).
Возвращаемое значение	`this` – созданный объект типа FuzzyInterval.

Конструктор объектов типа Sample, служащих для отображения выборки значений результатов многократных измерений

Sample

Вызов	`Sample(const std::vector<double>& x)`
Описание	Функция представляет собой конструктор класса типа Sample, служащего для отображения выборки значений результатов многократных измерений одной и той же отображаемой величины.
Аргументы	`x` – массив значений, образующих выборку значений результатов многократных измерений отображаемой величины.
Возвращаемое значение	`this` – созданный объект типа Sample.

Функция преобразования переменной типа PBox в переменную типа Hist с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

PBox2Hist

Вызов	`Hist PBox2Hist(size_t numBins)`
Описание	Функция представляет собой процедуру преобразования переменной типа PBox в переменную типа Hist с минимизацией потерь содержащейся в PBox информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса PBox; `numBins`– количество полос в создаваемом экземпляре класса Hist
Возвращаемое значение	`Hist` – созданный объект типа Hist.

Функция преобразования переменной типа PBox в переменную типа DempsterShafer с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

PBox2DempsterShafer

Вызов	`DempsterShafer PBox2DempsterShafer(size_t numFocal)`
Описание	Функция представляет собой процедуру преобразования переменной типа PBox в переменную типа DempsterShafer с минимизацией потерь содержащейся в PBox информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса PBox;

	`numFocal` – количество фокальных элементов в создаваемом экземпляре класса DempsterShafer
Возвращаемое значение	`DempsterShafer` – созданный объект типа DempsterShafer.

Функция преобразования переменной типа PBox в переменную типа Fuzzy с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

`PBox2Fuzzy`

Вызов	`Fuzzy PBox2Fuzzy(size_t numPoints)`
Описание	Функция представляет собой процедуру преобразования переменной типа PBox в переменную типа Fuzzy с минимизацией потерь содержащейся в PBox информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса PBox; `numPoints` – количество значений универсального множества для создаваемой нечеткой переменной (экземпляре класса Fuzzy)
Возвращаемое значение	`Fuzzy` – созданный объект типа Fuzzy.

Функция преобразования переменной типа PBox в переменную типа FuzzyInterval с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

`PBox2FuzzyInterval`

Вызов	`FuzzyInterval PBox2FuzzyInterval(size_t numAlpha)`
Описание	Функция представляет собой процедуру преобразования переменной типа PBox в переменную типа FuzzyInterval с минимизацией потерь содержащейся в PBox информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса PBox; `numAlpha` – количество значений уровней значений функции принадлежности для множества вложенных интервалов, образующих создаваемый экземпляр класса FuzzyInterval.
Возвращаемое значение	`FuzzyInterval` – созданный объект типа FuzzyInterval.

Функция преобразования переменной типа Hist в переменную типа PBox с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

Hist2PBox

Вызов	`PBox Hist2PBox()`
Описание	Функция представляет собой процедуру преобразования переменной типа Hist в переменную типа PBox с минимизацией потерь содержащейся в Hist информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса Hist.
Возвращаемое значение	`PBox`– созданный объект типа PBox.

Функция преобразования переменной типа Hist в переменную типа DempsterShafer с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

Hist2DempsterShafer

Вызов	`DempsterShafer Hist2DempsterShafer()`
Описание	Функция представляет собой процедуру преобразования переменной типа Hist в переменную типа DempsterShafer с минимизацией потерь содержащейся в Hist информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса Hist.
Возвращаемое значение	`DempsterShafer` – созданный объект типа DempsterShafer.

Функция преобразования переменной типа Hist в переменную типа Fuzzy с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

Hist2Fuzzy

Вызов	`Fuzzy Hist2Fuzzy(size_t numUniverse)`
Описание	Функция представляет собой процедуру преобразования переменной типа Hist в переменную типа Fuzzy с минимизацией потерь содержащейся в Hist информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса Hist; `numUniverse` – количество значений универсального множества для создаваемой нечеткой переменной (экземпляре класса Fuzzy)
Возвращаемое значение	`Fuzzy` – созданный объект типа Fuzzy.

Функция преобразования переменной типа Hist в переменную типа FuzzyInterval с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

`Hist2FuzzyInterval`

Вызов	`FuzzyInterval Hist2FuzzyInterval(size_t numAlpha)`
Описание	Функция представляет собой процедуру преобразования переменной типа Hist в переменную типа FuzzyInterval с минимизацией потерь содержащейся в Hist информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса Hist; `numAlpha` – количество значений уровней значений функции принадлежности для множества вложенных интервалов, образующих создаваемый экземпляр класса FuzzyInterval.
Возвращаемое значение	`FuzzyInterval` – созданный объект типа FuzzyInterval.

Функция преобразования переменной типа DempsterShafer в переменную типа PBox с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

`DempsterShafer2PBox`

Вызов	`PBox DempsterShafer2PBox(int numPoints)`
Описание	Функция представляет собой процедуру преобразования переменной типа DempsterShafer в переменную типа PBox с минимизацией потерь содержащейся в DempsterShafer информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса DempsterShafer; `numPoints` – количество значений в сетке дискретизации границ области возможных значений функции распределения
Возвращаемое значение	`PBox` – созданный объект типа PBox.

Функция преобразования переменной типа DempsterShafer в переменную типа Hist с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

`DempsterShafer2Hist`

Вызов	`Hist DempsterShafer2Hist(int numBins)`

Описание	Функция представляет собой процедуру преобразования переменной типа DempsterShafer в переменную типа Hist с минимизацией потерь содержащейся в DempsterShafer информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса DempsterShafer; `numBins` – количество полос в создаваемом экземпляре класса Hist (интервальнозначеной гистограмме)
Возвращаемое значение	`Histogram` – созданный объект типа Hist.

Функция преобразования переменной типа DempsterShafer в переменную типа Fuzzy с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

`DempsterShafer2Fuzzy`

Вызов	`Fuzzy DempsterShafer2Fuzzy(int numPoints)`
Описание	Функция представляет собой процедуру преобразования переменной типа DempsterShafer в переменную типа Fuzzy с минимизацией потерь содержащейся в DempsterShafer информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса DempsterShafer; `numPoints` – количество значений универсального множества для создаваемой нечеткой переменной (экземпляре класса Fuzzy)
Возвращаемое значение	`Fuzzy` – созданный объект типа Fuzzy.

Функция преобразования переменной типа DempsterShafer в переменную типа FuzzyInterval с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

`DempsterShafer2FuzzyInterval`

Вызов	`FuzzyInterval DempsterShafer2FuzzyInterval()`
Описание	Функция представляет собой процедуру преобразования переменной типа DempsterShafer в переменную типа FuzzyInterval с минимизацией потерь содержащейся в DempsterShafer информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса DempsterShafer.
Возвращаемое значение	`FuzzyInterval` – созданный объект типа FuzzyInterval.

Функция преобразования переменной типа Fuzzy в переменную типа PBox с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

Fuzzy2PBox

Вызов	`PBox Fuzzy2PBox(int numPoints)`
Описание	Функция представляет собой процедуру преобразования переменной типа Fuzzy в переменную типа PBox с минимизацией потерь содержащейся в Fuzzy информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса Fuzzy; `numPoints` – количество значений в сетке дискретизации границ области возможных значений функции распределения
Возвращаемое значение	`PBox` – созданный объект типа PBox.

Функция преобразования переменной типа Fuzzy в переменную типа Hist с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

Fuzzy2Hist

Вызов	`Hist Fuzzy2Hist(int numBins)`
Описание	Функция представляет собой процедуру преобразования переменной типа Fuzzy в переменную типа Hist с минимизацией потерь содержащейся в Fuzzy информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса Fuzzy; `numBins`– количество полос в создаваемом экземпляре класса Hist
Возвращаемое значение	`Hist` – созданный объект типа Hist.

Функция преобразования переменной типа Fuzzy в переменную типа DempsterShafer с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

Fuzzy2DempsterShafer

Вызов	`DempsterShafer Fuzzy2DempsterShafer(int numAlpha)`
Описание	Функция представляет собой процедуру преобразования переменной типа Fuzzy в переменную типа DempsterShafer с

	минимизацией потерь содержащейся в Fuzzy информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса Fuzzy; `numFocal` – количество фокальных элементов в создаваемом экземпляре класса DempsterShafer
Возвращаемое значение	`DempsterShafer` – созданный объект типа DempsterShafer.

Функция преобразования переменной типа Fuzzy в переменную типа FuzzyInterval с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

`Fuzzy2FuzzyInterval`

Вызов	`FuzzyInterval Fuzzy2FuzzyInterval(int numAlpha)`
Описание	Функция представляет собой процедуру преобразования переменной типа Fuzzy в переменную типа FuzzyInterval с минимизацией потерь содержащейся в Fuzzy информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса Fuzzy; `numAlpha` – количество значений уровней значений функции принадлежности для множества вложенных интервалов, образующих создаваемый экземпляр класса FuzzyInterval.
Возвращаемое значение	`FuzzyInterval` – созданный объект типа FuzzyInterval.

Функция преобразования переменной типа FuzzyInterval в переменную типа PBox с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

`FuzzyInterval2PBox`

Вызов	`PBox FuzzyInterval2PBox(int numPoints)`
Описание	Функция представляет собой процедуру преобразования переменной типа FuzzyInterval в переменную типа PBox с минимизацией потерь содержащейся в FuzzyInterval информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса FuzzyInterval; `numPoints` – количество значений в сетке дискретизации границ области возможных значений функции распределения
Возвращаемое значение	`PBox` – созданный объект типа PBox.

Функция преобразования переменной типа FuzzyInterval в переменную типа Hist с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

FuzzyInterval2Hist

Вызов	`Hist FuzzyInterval2Hist(int numBins)`
Описание	Функция представляет собой процедуру преобразования переменной типа FuzzyInterval в переменную типа Hist с минимизацией потерь содержащейся в FuzzyInterval информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса FuzzyInterval; `numBins`– количество полос в создаваемом экземпляре класса Hist
Возвращаемое значение	`Hist` – созданный объект типа Hist.

Функция преобразования переменной типа FuzzyInterval в переменную типа DempsterShafer с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

FuzzyInterval 2DempsterShafer

Вызов	`DempsterShafer FuzzyInterval2DempsterShafer(const string& method)`
Описание	Функция представляет собой процедуру преобразования переменной типа FuzzyInterval в переменную типа DempsterShafer с минимизацией потерь содержащейся в FuzzyInterval информации о возможных значениях отображаемой величины.
Аргументы	`this` – преобразуемый экземпляр класса FuzzyInterval; `method` – метод преобразования: ‘fractional’ или ‘nested’.
Возвращаемое значение	`DempsterShafer` – созданный объект типа DempsterShafer.

Функция преобразования переменной типа FuzzyInterval в переменную типа Fuzzy с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование)

FuzzyInterval2Fuzzy

Вызов	`Fuzzy FuzzyInterval2Fuzzy(int numPoints)`
Описание	Функция представляет собой процедуру преобразования переменной типа FuzzyInterval в переменную типа Fuzzy с минимизацией потерь содержащейся в FuzzyInterval информации о возможных значениях отображаемой величины.

Аргументы	`this` – преобразуемый экземпляр класса FuzzyInterval; `numPoints` – количество значений универсального множества для создаваемой нечеткой переменной (экземпляре класса Fuzzy)
Возвращаемое значение	`Fuzzy` – созданный объект типа Fuzzy.

Функция преобразования переменной типа Sample в переменную типа PBox (построение области возможных значений функции распределения по выборке результатов многократных измерений согласуемой величины)

`Sample2PBox`

Вызов	`PBox Sample2PBox()`
Описание	Функция представляет собой процедуру преобразования переменной типа Sample в переменную типа PBox.
Аргументы	`this` – преобразуемый экземпляр класса Sample.
Возвращаемое значение	`PBox` – созданный объект типа PBox.

Функция преобразования переменной типа Sample в переменную типа FuzzyInterval (построение нечеткого интервала по выборке результатов многократных измерений согласуемой величины)

`Sample2FuzzyInterval`

Вызов	`FuzzyInterval Sample2FuzzyInterval(size_t numAlpha)`
Описание	Функция представляет собой процедуру преобразования переменной типа Sample в переменную типа FuzzyInterval.
Аргументы	`this` – преобразуемый экземпляр класса Sample; `numAlpha` – количество значений уровней значений функции принадлежности для множества вложенных интервалов, образующих создаваемый экземпляр класса FuzzyInterval.
Возвращаемое значение	`FuzzyInterval` – созданный объект типа FuzzyInterval.

Функция построения интервала возможных значений (доверительного интервала) среднеквадратического отклонения того множества возможных значений или неопределенности согласуемой величины, что отражается экземплярами классов PBox, Hist, DempsterShafer, Fuzzy, FuzzyInterval, Sample

`getStd`

Вызов	`std::pair<double, double> getStd(const std::string& method)`
Описание	Функция оценки возможных значений среднеквадратического отклонения неточной величины, представленной в одном из форматов представления неопределенности (метод классов PBox, Hist, DempsterShafer, Fuzzy, FuzzyInterval, Sample).
Аргументы	`this` – экземпляр одного из классов представления информации о неопределенности: PBox, Hist, DempsterShafer, Fuzzy, FuzzyInterval, Sample; `method` – метод оценки множества возможных значений среднеквадратического отклонения: ‘approximate’ – приближенная, но быстрая оценка; ‘accurate’ – точная, но более продолжительная по времени оценка.
Возвращаемое значение	`stdCint` – границы интервала возможных значений среднеквадратического отклонения (доверительного интервала), большее из значений которых используется для дальнейшего выполнения согласования индустриальных данных.

Процедура выполнения согласования неточных величин в рамках аналитической модели с решением задачи Каруша-Куна-Таккера с применением теоремы Ефремова-Козлова

`AnalyticalModel`

Вызов	`VectorXd AnalyticalModel(const VectorXd& x, const MatrixXd& A_y, const VectorXd& b_y, const VectorXd& c_y, double r2_y)`
Описание	Процедура выполнения согласования неточных величин в рамках аналитической модели с решением задачи Каруша-Куна-Таккера с применением теоремы Ефремова-Козлова.
Аргументы	`x` – вектор значений согласуемых величин (результаты совместных измерений на объекте, чья математическая модель описывает ограничения на возможные значения согласуемых значений); `A_y` – матрица линейных ограничений, накладываемых на согласуемые значения, вида $A\_y \cdot y = b\_y$; `b_y` – вектор правых частей в линейных ограничений, накладываемых на согласуемые значения, вида $A\_y \cdot y = b\_y$; `c_y` – вектор, соответствующий координатам центра эллипсоида в пространстве значений вектора `x`, накладывающего групповое условие типа неравенства на возможную область допустимых значений вектора `y`, вида $(y-c\_y)^T \cdot (y-c\_y) = r2\_y$; `r2_y` – величина группового ограничения на возможную область допустимых значений вектора `y`, вида $(y-c\_y)^T \cdot (y-c\_y) = r2\_y$.
Возвращаемое значение	`y` – результаты согласования значений по аналитической модели процедуры Data Reconciliation, основанной на теореме Ефремова-Козлова.

Процедура выполнения оценки неопределенности результата согласования по аналитической модели AnalyticalModel при представлении неопределенности исходных данных в одном из перечисленных форматов: PBox, Hist, DempsterShafer, Fuzzy, FuzzyInterval, Sample

GetUncertainty

Вызов	`VectorXd GetUncertainty(const VectorXd& x, const VectorXd& dx, const MatrixXd& A_y, const VectorXd& b_y, const VectorXd& c_y, double r2_y)`
Описание	Процедура выполнения оценки неопределенности результата согласования по аналитической модели AnalyticalModel при представлении неопределенности исходных данных в одном из перечисленных форматов: PBox, Hist, DempsterShafer, Fuzzy, FuzzyInterval, Sample.
Аргументы	`x` – вектор значений согласуемых величин (результаты совместных измерений на объекте, чья математическая модель описывает ограничения на возможные значения согласуемых значений); `dx` – массив экземпляров одного из классов выражения неопределенности согласуемых величин `x` (на выбор пользователя – PBox, Hist, DempsterShafer, Fuzzy, FuzzyInterval, Sample); `A_y` – матрица линейных ограничений, накладываемых на согласуемые значения, вида $A\_y \cdot y = b\_y$; `b_y` – вектор правых частей в линейных ограничений, накладываемых на согласуемые значения, вида $A\_y \cdot y = b\_y$; `c_y` – вектор, соответствующий координатам центра эллипсоида в пространстве значений вектора `x`, накладывающего групповое условие типа неравенства на возможную область допустимых значений вектора `y`, вида $(y-c\_y)^T \cdot (y-c\_y) = r2\_y$; `r2_y` – величина группового ограничения на возможную область допустимых значений вектора `y`, вида $(y-c\_y)^T \cdot (y-c\_y) = r2\_y$.
Возвращаемое значение	`dy` – массив оценок неопределенности результатов согласования значений по аналитической модели процедуры Data Reconciliation, основанной на теореме Ефремова-Козлова; результаты возвращаются в виде экземпляра того же класса, что и `dx`.

B.5.3. Примеры и особенности

Приведённая в листинге B.1 функция `DRparamEq()` предназначена для согласования измеренных данных с использованием метода, основанного на численном решении системы уравнений. Она использует модель зависимости между измеренными величинами, заданную функцией `depend_func`, и априорные ошибки измерений для корректировки данных. Решается задача оптимальной оценки значений физических величин, известных с погрешностью, при этом априорно заданные взаимосвязи меду величинами формализованы в виде системы уравнений.

Листинг В.1 – Функция DRparamEq() для согласования данных с использованием метода, основанного на численной оптимизации

```
#include <iostream>
#include <vector>
#include <complex>
#include <stdexcept>
#include <numeric>
#include <cmath>
#include <Eigen/Dense>
#include <unsupported/Eigen/NonLinearOptimization>

using namespace std;
using namespace Eigen;

// Function signature for the dependency model function
typedef VectorXd(*DependFunc)(const VectorXd&, const VectorXd&);

// Function to solve the Alpha system
VectorXd alpha_system_to_solve(const VectorXd& mu_and_l,
    DependFunc depend_func,
    const MatrixXd& msrd_data,
    const VectorXd& vars_,
    const VectorXd& params_) {
    int xNum = msrd_data.rows();
    int n = msrd_data.cols();

    if (xNum == 0 || n == 0) {
        throw invalid_argument("Measured data must be a non-empty 2D array.");
    }

    VectorXd mustbezeros(xNum);
    mustbezeros.setZero();

    for (int j = 0; j < xNum; ++j) {
        if (vars_(j) == 0) {
            throw invalid_argument("None of the error parameters should be 0.");
        }
        else {
         //  // Imaginary perturbation for numerical derivative
         //   VectorXcd imagx = VectorXcd::Zero(xNum);
```

```
        //   imagx(j) = complex<double>(0.0, mu_and_l(j) * pow(10.0, -100) + pow(10.0,
-101));


        //   // Create perturbed input for numerical derivative
        //   VectorXd mu_and_l_perturbed = mu_and_l;
        //   mu_and_l_perturbed += imagx.real();



            // ---------------
            // Numerical derivative using small imaginary perturbation
            VectorXcd imagx = VectorXcd::Zero(xNum);
            imagx[j] = std::complex<double>(0.0, mu_and_l[j] * pow(10.0, -100) +
pow(10.0, -101));


            VectorXcd                          mu_and_l_complex                          =
mu_and_l.head(xNum).cast<std::complex<double>>() + imagx;
            VectorXd mu_and_l_real = mu_and_l_complex.real().cast<double>();
            VectorXd df_result = depend_func(mu_and_l_real, params_);


            VectorXd df_dx = df_result.cast<std::complex<double>>().array().imag() /
imagx.array().imag()[j];


            // Update mustbezeros
            VectorXd        elem_wize_multy_res        =        df_dx.array()        *
mu_and_l.tail(mu_and_l.size() - xNum).array();
            // ---------------



            //// Numerical derivative approximation using imaginary perturbation
            //VectorXd df_dx_result = depend_func(mu_and_l_perturbed, params_);
            //complex<double> df_dx = df_dx_result(j).imag() / imagx(j).imag();


            // Update mustbezeros
            double mean_msrd_data_j = msrd_data.row(j).mean();
            mustbezeros(j) = mu_and_l(j) / vars_(j)
                - mean_msrd_data_j / vars_(j) + elem_wize_multy_res.sum() / n;
        }
    }


    // Evaluate the model residuals
    VectorXd model_res = depend_func(mu_and_l.head(xNum), params_);
```

```
    mustbezeros.conservativeResize(mustbezeros.size() + model_res.size());
    mustbezeros.tail(model_res.size()) = model_res;

    return mustbezeros;
}

// Functor for the system of equations
struct AlphaSystemFunctor {
    DependFunc depend_func;
    const MatrixXd& msrd_data;
    const VectorXd& vars_;
    const VectorXd& params_;
    int xNum;

    AlphaSystemFunctor(DependFunc df, const MatrixXd& data, const VectorXd& vars, const
VectorXd& params)
        :    depend_func(df),    msrd_data(data),    vars_(vars),    params_(params),
xNum(data.rows()) {}

    // Compute the residuals
    int operator()(const VectorXd& mu_and_l, VectorXd& fvec) const {
        VectorXd residuals = alpha_system_to_solve(mu_and_l, depend_func, msrd_data,
vars_, params_);
        fvec = residuals;
        return 0;
    }

    // Compute the Jacobian (optional but recommended for performance)
    int df(const VectorXd& mu_and_l, MatrixXd& fjac) const {
        const double epsilon = 1e-8; // Small perturbation for numerical differentiation
        VectorXd fvec(values());
        VectorXd mu_and_l_perturbed = mu_and_l;

        // Compute the Jacobian using finite differences
        for (int j = 0; j < mu_and_l.size(); ++j) {
            double temp = mu_and_l(j);
            mu_and_l_perturbed(j) = temp + epsilon;
            (*this)(mu_and_l_perturbed, fvec);
            VectorXd fvec_plus = fvec;

            mu_and_l_perturbed(j) = temp - epsilon;
```

```
            (*this)(mu_and_l_perturbed, fvec);
            VectorXd fvec_minus = fvec;

            fjac.col(j) = (fvec_plus - fvec_minus) / (2 * epsilon); // Central difference
            mu_and_l_perturbed(j) = temp; // Restore the original value
        }
        return 0;
    }

    int inputs() const { return xNum + depend_func(VectorXd::Zero(xNum), params_).size(); }
    int values() const { return xNum + depend_func(VectorXd::Zero(xNum), params_).size(); }
};

// Function to reconcile measured data
VectorXd DRparamEq(DependFunc depend_func,
    const MatrixXd& msrd_data,
    VectorXd& error_params,
    const VectorXd& params_) {
    int er_par_num = error_params.size();
    int xNum = msrd_data.rows();
    int n = msrd_data.cols();

    // ----------------------------------------
    error_params = error_params * 10.0;
    // ----------------------------------------

    if (xNum == 0 || n == 0) {
        throw invalid_argument("Measured data must be a non-empty 2D array.");
    }

    // Step 1: Compute l (a zero vector of the same length as depend_func's output)
    VectorXd l = depend_func(VectorXd::Zero(xNum), params_);

    // Step 2: Extend the initial guess (mu_and_l_start) by appending l to msrd_data
    VectorXd mu_and_l_start(xNum + l.size());
    mu_and_l_start.head(xNum) = msrd_data.col(0); // Assuming first column as initial guess
    mu_and_l_start.tail(l.size()) = l;
```

```
    // Step 3: Solve the system of equations using Eigen's LevenbergMarquardt solver
    AlphaSystemFunctor functor(depend_func, msrd_data, error_params, params_);
    LevenbergMarquardt<AlphaSystemFunctor> lm(functor);
    lm.minimize(mu_and_l_start);

    // Step 4: Extract the reconciled measured data (excluding 'l' values)
    VectorXd reconciled = mu_and_l_start.head(xNum);

    return reconciled;
}

// Example dependency function
VectorXd example_depend_func(const VectorXd& mu, const VectorXd& params) {
    VectorXd result(1);
    result(0) = mu(0) - 2.0 * mu(1) - params(0);
    return result;
}

int main() {
    try {
        // Example usage
        MatrixXd msrd_data(2, 5); // Example (measured_var_n x n_of_measurements) matrix
        msrd_data << 10.0, 11.0, 9.0, 9.5, 10.5,
            10.0, 9.0, 11.0, 9.5, 10.5;

        VectorXd error_params(2); // length is (measured_var_n)
        error_params << 2.0, 2.0;
        VectorXd params(1);
        params << 0;

        VectorXd    reconciled_data    =    DRparamEq(example_depend_func,    msrd_data,
error_params, params);
        cout << "Reconciled Data:" << endl;
        cout << reconciled_data.transpose() << endl;
    }
    catch (const exception& e) {
        cerr << "Error: " << e.what() << endl;
    }
    return 0;
}
```

Пошаговое объяснение работы функции

1. Инициализация начальных значений:

Начальные значения для согласованных данных (`mu`) берутся из первого столбца измеренных данных.

Начальные значения для множителей Лагранжа (`l`) устанавливаются в нули.

2. Решение системы уравнений:

Функция `alpha_system_to_solve` формирует систему уравнений, которая включает:

Уравнения для согласования данных с учетом априорных ошибок.

Уравнения, заданные моделью зависимости.

Для решения системы используется метод Левенберга-Марквардта.

3. Извлечение результата:

После решения системы извлекаются согласованные данные (первые `xNum` элементов результата).

Пример использования функции `DRparamEq()`

Рассмотрим пример, где у нас есть набор измеренных данных, которые мы хотим согласовать с использованием модели зависимости.

1. Определение модели зависимости

Модель зависимости может быть, например, линейной функцией:

```
VectorXd example_depend_func(const VectorXd& mu, const VectorXd& params) {
    VectorXd result(1);
    result(0) = mu(0) - 2.0 * mu(1) - params(0);
    return result;
}
```

2. Подготовка данных

Измеренные данные: 2 измерения, каждое измерение повторено 5 раз.

Априорные ошибки: заданы для каждого измерения.

Параметры модели: `params = [0]`.

```
MatrixXd msrd_data(2, 5); // Пример (measured_var_n x n_of_measurements) матрицы
msrd_data << 10.0, 11.0, 9.0, 9.5, 10.5,
             10.0, 9.0, 11.0, 9.5, 10.5;

VectorXd error_params(2); // Длина равна количеству измерений (measured_var_n)
error_params << 2.0, 2.0;

VectorXd params(1);
```

```
params << 0;
```

Вызов функции `DRparamEq()`

```
VectorXd reconciled_data = DRparamEq(example_depend_func, msrd_data, error_params, params);


cout << "Reconciled Data:" << endl;
cout << reconciled_data.transpose() << endl;
```

Результат

Функция вернет вектор согласованных данных, которые лучше соответствуют модели зависимости и учитывают априорные ошибки измерений.

Приведённая в листинге 2 функция `DRnonparamEqIneqRobust()` предназначена для согласования измеренных данных с использованием метода, основанного на численном решении системы уравнений. Она применяется для корректировки измеренных данных с учетом заданных моделей зависимостей (уравнений и неравенств) и априорных ошибок измерений. В основе функции лежит использование библиотеки Eigen с модулем нелинейной оптимизации Eigen/NonLinearOptimization.

Листинг В.2 – Функция `DRnonparamEqIneqRobust()` для согласования данных с использованием метода, основанного на численной оптимизации

```
#include <Eigen/Dense>
#include <unsupported/Eigen/NonLinearOptimization>
#include <iostream>
#include <cmath>
#include <vector>
#include <algorithm>
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

using namespace Eigen;

typedef std::complex<double> Complex;
typedef Matrix<Complex, Dynamic, 1> VectorXc;
typedef Matrix<double, Dynamic, 1> VectorXd;
typedef Matrix<Complex, Dynamic, Dynamic> MatrixXc;

// Function to compute the Gaussian kernel
double kernel(double x) {
```

```
    return std::exp(-x * x / 2.0) / std::sqrt(2.0 * M_PI);
}

// Function to compute the median absolute deviation (MAD)
double median_abs_deviation(const VectorXd& data) {
    double median = data.size() % 2 == 0 ?
        (data(data.size() / 2 - 1) + data(data.size() / 2)) / 2.0 :
        data(data.size() / 2);
    VectorXd abs_dev = (data.array() - median).abs();
    std::sort(abs_dev.data(), abs_dev.data() + abs_dev.size());
    double mad = abs_dev.size() % 2 == 0 ?
        (abs_dev(abs_dev.size() / 2 - 1) + abs_dev(abs_dev.size() / 2)) / 2.0 :
        abs_dev(abs_dev.size() / 2);
    return mad * 1.483; // Scale factor for consistency with standard deviation
}

// Function to solve the system of equations using kernel-based estimation
VectorXd kernels_system_to_solve(const VectorXd& mu_and_l, const
std::function<VectorXc(const VectorXc&, const VectorXd&)>& eqies_model,
    const std::function<VectorXc(const VectorXc&, const VectorXd&)>& ineqies_model,
const MatrixXd& msrd_data,
    const VectorXd& model_params, const VectorXd& bandwidths, const VectorXd& msrd_vars,
int xNum, int n, int eqNum, int ineqNum) {
    VectorXc mustbezeros = VectorXc::Zero(xNum);

    // Handle scalar bandwidth case
    VectorXd bandwidths_;
    if (bandwidths.size() == 1) {
        bandwidths_ = VectorXd::Constant(xNum, bandwidths(0));
    }
    else {
        bandwidths_ = bandwidths;
    }

    for (int j = 0; j < xNum; ++j) {
        double mean_msrd = msrd_data.row(j).mean();

        // Compute median and scaled MAD
        VectorXd row = msrd_data.row(j);
        std::sort(row.data(), row.data() + row.size());
        double median = row.size() % 2 == 0 ?
```

```
            (row(row.size() / 2 - 1) + row(row.size() / 2)) / 2.0 :
            row(row.size() / 2);
        double mad_value = median_abs_deviation(row);
        double s_msrd = mad_value * 1.483;

        VectorXd res = VectorXd::Zero(n);

        for (int i = 0; i < n; ++i) {
            // Compute kernel weights
            VectorXd denumerator = (msrd_data.row(j).array() - mean_msrd) / s_msrd -
(msrd_data(j, i) - mu_and_l(j)) / s_msrd;
            denumerator = denumerator.unaryExpr([&](double x) { return kernel(x /
bandwidths_(j)); });

            double numerator = (msrd_data.row(j).array() * denumerator.array()).sum();
            if (denumerator.sum() == 0) {
                res(i) = 0;
            }
            else {
                res(i) = numerator / denumerator.sum();
            }
        }

        double mean_via_kerns = res.mean();

        // Numerical derivative using small imaginary perturbation
        VectorXc imagx = VectorXc::Zero(xNum);
        imagx(j) = Complex(0, 1) * (mu_and_l(j) * 1e-100 + 1e-101);

        // Numerical derivative approximation using imaginary perturbation
        VectorXc df_dx_eqies = eqies_model(mu_and_l.head(xNum).cast<Complex>() + imagx,
model_params).imag() / imagx(j).imag();
        VectorXc df_dx_ineqies = ineqies_model(mu_and_l.head(xNum).cast<Complex>() +
imagx, model_params).imag() / imagx(j).imag();

        VectorXc df_dx(df_dx_eqies.size() + df_dx_ineqies.size());
        df_dx << df_dx_eqies, df_dx_ineqies;

        // Update mustbezeros
        mustbezeros(j) = (mean_msrd - mean_msrd - mean_via_kerns + mu_and_l(j) +
```

```
            bandwidths_(j) * bandwidths_(j) * (1.0 / n) * s_msrd * s_msrd *
df_dx.dot(mu_and_l.segment(xNum, eqNum + ineqNum).cast<Complex>()));

    }


    // Evaluate the model residuals

    VectorXc   eqies_model_res   =   eqies_model(mu_and_l.head(xNum).cast<Complex>(),
model_params);

    VectorXc  ineqies_model_res  =  ineqies_model(mu_and_l.head(xNum).cast<Complex>(),
model_params);


    VectorXd mustbezeros_real = mustbezeros.real();

    VectorXd eqies_model_res_real = eqies_model_res.real();

    VectorXd ineqies_model_res_real = ineqies_model_res.real();


    VectorXd    result(mustbezeros_real.size()    +    eqies_model_res_real.size()    +
ineqies_model_res_real.size());

    result << mustbezeros_real, eqies_model_res_real, ineqies_model_res_real;


    VectorXd ineqies_mul = ineqies_model_res_real.cwiseProduct(mu_and_l.segment(xNum +
eqNum, ineqNum));

    result.conservativeResize(result.size() + ineqies_mul.size());

    result.tail(ineqies_mul.size()) = ineqies_mul;


    return result;

}


// Functor for the Levenberg-Marquardt algorithm

struct KernelsSystemFunctor {

    KernelsSystemFunctor(const     std::function<VectorXc(const     VectorXc&,     const
VectorXd&)>& eqies_model,

        const std::function<VectorXc(const VectorXc&, const VectorXd&)>& ineqies_model,

        const  MatrixXd&  msrd_data,  const  VectorXd&  model_params,  const  VectorXd&
bandwidths,

        const VectorXd& msrd_vars, int xNum, int n, int eqNum, int ineqNum)

        : eqies_model(eqies_model), ineqies_model(ineqies_model), msrd_data(msrd_data),

        model_params(model_params), bandwidths(bandwidths), msrd_vars(msrd_vars),

        xNum(xNum), n(n), eqNum(eqNum), ineqNum(ineqNum) {}


    // Operator to compute the residuals

    int operator()(const VectorXd& mu_and_l, VectorXd& fvec) const {

        fvec = kernels_system_to_solve(mu_and_l, eqies_model, ineqies_model, msrd_data,
model_params, bandwidths, msrd_vars, xNum, n, eqNum, ineqNum);
```

```
        return 0;
    }

    // Method to compute the Jacobian
    int df(const VectorXd& mu_and_l, MatrixXd& fjac) const {
        int m = fvec.size(); // Number of equations
        int p = mu_and_l.size(); // Number of variables

        fjac.resize(m, p);
        fjac.setZero();

        // Compute the Jacobian numerically using finite differences
        const double eps = 1e-8; // Small perturbation for finite differences
        VectorXd mu_and_l_perturbed = mu_and_l;

        for (int j = 0; j < p; ++j) {
            mu_and_l_perturbed(j) += eps;
            VectorXd fvec_perturbed = kernels_system_to_solve(mu_and_l_perturbed,
eqies_model, ineqies_model, msrd_data, model_params, bandwidths, msrd_vars, xNum, n,
eqNum, ineqNum);
            fjac.col(j) = (fvec_perturbed - fvec) / eps;
            mu_and_l_perturbed(j) = mu_and_l(j); // Reset the perturbation
        }

        return 0;
    }

    int inputs() const { return xNum + eqNum + ineqNum; }
    int values() const { return xNum + eqNum + ineqNum; }

    std::function<VectorXc(const VectorXc&, const VectorXd&)> eqies_model;
    std::function<VectorXc(const VectorXc&, const VectorXd&)> ineqies_model;
    MatrixXd msrd_data;
    VectorXd model_params;
    VectorXd bandwidths;
    VectorXd msrd_vars;
    int xNum, n, eqNum, ineqNum;
    mutable VectorXd fvec; // Store the residuals for use in df
};
```

```
// Function to reconcile measured data using kernel-based estimation
VectorXd DRnonparamEqIneqRobust(const std::function<VectorXc(const VectorXc&, const VectorXd&)>& eqies_model,
    const std::function<VectorXc(const VectorXc&, const VectorXd&)>& ineqies_model,
    const MatrixXd& msrd_data, const VectorXd& model_params, const VectorXd& prior_vars, const VectorXd& bandwidths) {
    int xNum = msrd_data.rows();
    int n = msrd_data.cols();

    VectorXd data_init_points = msrd_data.rowwise().mean();
    int eqNum = eqies_model(data_init_points.cast<Complex>(), model_params).size();
    int ineqNum = ineqies_model(data_init_points.cast<Complex>(), model_params).size();

    // Step 1: Compute l (a zero vector of the same length as eqies_model's output)
    VectorXd l_eqies = VectorXd::Zero(eqNum);
    VectorXd l_ineqies = VectorXd::Zero(ineqNum);
    VectorXd l_ineqies_mul_by_l = VectorXd::Zero(ineqNum);

    VectorXd l(eqNum + ineqNum + ineqNum);
    l << l_eqies, l_ineqies, l_ineqies_mul_by_l;

    // Step 2: Initialize mu_and_l_start with the mean of msrd_data
    VectorXd mu_and_l_start(xNum + eqNum + ineqNum + ineqNum);
    mu_and_l_start << data_init_points, l;

    // Step 3: Solve the system of equations using Levenberg-Marquardt
    KernelsSystemFunctor functor(eqies_model, ineqies_model, msrd_data, model_params, bandwidths, prior_vars, xNum, n, eqNum, ineqNum);
    LevenbergMarquardt<KernelsSystemFunctor> lm(functor);

    // Set the initial guess for the optimizer
    lm.parameters.maxfev = 1000; // Maximum number of function evaluations
    lm.parameters.xtol = 1.0e-6; // Tolerance for the solution

    // Minimize the function
    int status = lm.minimize(mu_and_l_start);

    //if (status != LevenbergMarquardtSpace::Status::RelativeReductionTooSmall && status != LevenbergMarquardtSpace::Status::CosinusTooSmall) {
    //    throw std::runtime_error("Optimization failed to converge.");
    //}
```

```
    // Step 4: Extract the reconciled measured data (excluding 'l' values)
    VectorXd reconciled_with_kernels = mu_and_l_start.head(xNum);

    return reconciled_with_kernels;
}

// Define a dependency model function
VectorXc eqies_model(const VectorXc& mu, const VectorXd& params) {
    // Example: A simple linear model
    VectorXc res(2);
    res << mu(0) * params(0) - mu(1),
        mu(0) - mu(1) / params(0);
    return res;
}

// Define a dependency model function
VectorXc ineqies_model(const VectorXc& mu, const VectorXd& params) {
    // Example: A simple linear model
    VectorXc res(2);
    res << mu(0) * params(0) - mu(1),
        mu(0) - mu(1) / params(0);
    return res;
}

int main() {
    // Define measured data (msrd_data)
    MatrixXd msrd_data(2, 3);
    msrd_data << 1.0, 1.1, 0.9,
        1.0, 1.1, 0.9;

    // Define error parameters (error_params)
    VectorXd prior_vars(2);
    prior_vars << 0.1, 0.1;

    // Define constant model parameters (params_)
    VectorXd model_params(1);
    model_params << 2.0;

    // Define bandwidths
```

```
    VectorXd bandwidths(2);
    bandwidths << 0.5, 0.5;

    // Call the kernel_data_reconcil function to reconcile the measured data
    VectorXd reconciled_data = DRnonparamEqIneqRobust(eqies_model, ineqies_model, msrd_data, model_params, prior_vars, bandwidths);

    // Print the reconciled data
    std::cout << "Reconciled Data:\n" << reconciled_data << std::endl;

    return 0;
}
```

Особенности функции:

1. Робастность:

Функция использует методы ядерного сглаживания (kernel-based estimation) для устойчивой обработки данных, содержащих выбросы или шумы.

Вместо стандартных методов (например, метода наименьших квадратов) применяется медианная абсолютная девиация (MAD), что делает функцию устойчивой к аномалиям в данных.

2. Учет уравнений и неравенств:

Функция поддерживает как уравнения (eqies_model), так и неравенства (ineqies_model), что позволяет учитывать физические или логические ограничения на данные.

3. Использование ядерных функций:

Для сглаживания данных используются ядерные функции (например, гауссово ядро), что позволяет учитывать локальные особенности данных.

4. Численная оптимизация:

Для решения системы уравнений применяется метод Левенберга-Марквардта, реализованный в библиотеке Eigen. Этот метод эффективен для решения нелинейных задач оптимизации.

Пример использования функции `DRnonparamEqIneqRobust()`

Рассмотрим пример, где у нас есть набор измеренных данных, которые мы хотим согласовать с использованием модели зависимости.

1. Определение моделей уравнений и неравенств

Модель зависимости может быть, например, линейной функцией:

```
VectorXc eqies_model(const VectorXc& mu, const VectorXd& params) {
    // Пример: Простая линейная модель
```

```
    VectorXc res(2);
    res << mu(0) * params(0) - mu(1),
           mu(0) - mu(1) / params(0);
    return res;
}

VectorXc ineqies_model(const VectorXc& mu, const VectorXd& params) {
    // Пример: Простая линейная модель
    VectorXc res(2);
    res << mu(0) * params(0) - mu(1),
           mu(0) - mu(1) / params(0);
    return res;
}
```

2. Подготовка данных

- Измеренные данные: 2 измерения, каждое измерение повторено 3 раза.
- Априорные ошибки: заданы для каждого измерения.
- Параметры модели: `params = [2.0]`.
- Параметры сглаживания: `bandwidths = [0.5, 0.5]`.

```
MatrixXd msrd_data(2, 3); // Пример (measured_var_n x n_of_measurements) матрицы
msrd_data << 1.0, 1.1, 0.9,
             1.0, 1.1, 0.9;

VectorXd prior_vars(2); // Длина равна количеству измерений (measured_var_n)
prior_vars << 0.1, 0.1;

VectorXd model_params(1);
model_params << 2.0;

VectorXd bandwidths(2);
bandwidths << 0.5, 0.5;
```

3. Вызов функции `DRnonparamEqIneqRobust()`

```
VectorXd reconciled_data = DRnonparamEqIneqRobust(eqies_model, ineqies_model, msrd_data, model_params, prior_vars, bandwidths);

std::cout << "Reconciled Data:\n" << reconciled_data << std::endl;
```

4. Результат

Функция вернет вектор согласованных данных, которые лучше соответствуют уравнениям и неравенствам, а также учитывают априорные ошибки измерений.

В.6 СПИСОК ФУНКЦИЙ И МЕТОДОВ

estAccuracyIncreaseByDR()	Функция выполняет приближенную оценку потенциального уточнения совместных измерений, достигаемого за счет учета известных функциональных взаимосвязей между измеряемыми величинами, на основе локальной линеаризации модели и метода декомпозиции алгоритма условной оптимизации.
DRparamEq()	Функция выполняет параметрическое согласование совместно измеренных величин, закон распределения случайных погрешностей которых соответствует нормальному распределению вероятностей. Подлежат учету зависимости в виде равенств и неравенств.
DRsemiparamEq()	Файл содержит одноименную вызываемую функцию, которая выполняет непараметрическое согласование совместно измеренных величин, закон распределения которых оценивается проекционным методом с применением в качестве модели усеченного ряда Грамма-Шарлье. Подлежат учету зависимости в виде равенств и неравенств.
DRnonparamEq()	Функция выполняет непараметрическое согласование совместно измеренных величин, закон распределения которых оценивается методом ядерной аппроксимации с использованием ядра Гаусса. Подлежат учету зависимости в виде равенств и неравенств.
DRparamEqRobust()	Функция выполняет робастное параметрическое согласование совместно измеренных величин, закон распределения случайных погрешностей которых соответствует нормальному распределению вероятностей.
DRsemiparamEqRobust()	Файл содержит одноименную вызываемую функцию, которая выполняет робастное полу-непараметрическое согласование совместно измеренных величин, закон распределения которых оценивается проекционным методом с применением в качестве модели усеченного ряда Грамма-Шарлье.
DRnonparamEqRobust()	Функция выполняет робастное непараметрическое согласование совместно измеренных величин, закон распределения которых оценивается методом ядерной аппроксимации с использованием ядра Гаусса.
DRparamEqIneq()	Функция выполняет параметрическое согласование совместно измеренных величин, закон распределения случайных погрешностей которых соответствует нормальному распределению вероятностей. Подлежат учету зависимости в виде равенств и неравенств.
DRsemiparamEqIneq()	Файл содержит одноименную вызываемую функцию, которая выполняет непараметрическое согласование совместно измеренных величин, закон распределения которых

	оценивается проекционным методом с применением в качестве модели усеченного ряда Грамма-Шарлье. Подлежат учету зависимости в виде равенств и неравенств.
DRnonparamEqIneq()	Функция выполняет непараметрическое согласование совместно измеренных величин, закон распределения которых оценивается методом ядерной аппроксимации с использованием ядра Гаусса. Подлежат учету зависимости в виде равенств и неравенств.
DRparamEqIneqRobust()	Функция выполняет робастное параметрическое согласование совместно измеренных величин, закон распределения случайных погрешностей которых соответствует нормальному распределению вероятностей. Подлежат учету зависимости в виде равенств и неравенств.
DRsemiparamEqIneqRobust()	Файл содержит одноименную вызываемую функцию, которая выполняет робастное полу-непараметрическое согласование совместно измеренных величин, закон распределения которых оценивается проекционным методом с применением в качестве модели усеченного ряда Грамма-Шарлье. Подлежат учету зависимости в виде равенств и неравенств.
DRnonparamEqIneqRobust()	Функция выполняет робастное непараметрическое согласование совместно измеренных величин, закон распределения которых оценивается методом ядерной аппроксимации с использованием ядра Гаусса. Подлежат учету зависимости в виде равенств и неравенств.
PBox()	Конструктор класса PBox для канонического представления неопределенности входных данных в форме области возможных значений функции распределения. Границы представляемой области дискретизированы и представлены кусочно-постоянными функциями.
Hist()	Конструктор класса Hist для представления неопределенности входных данных в форме интервальной гистограммы по Берлинту, высота полос которой представлена интервалом возможных значений.
DempsterShafer()	Конструктор класса DempsterShafer для представления неопределенности входных данных в форме структуры Демпстера-Шафера.
Fuzzy()	Конструктор класса Fuzzy для представления неопределенности входных данных в форме нечеткой переменной по Заде.
FuzzyInterval()	Конструктор класса FuzzyInterval для представления неопределенности входных данных в форме нечеткого интервала (допускающего в том числе представление классического интервала по арифметике Мура как частный вырожденный случай).
Sample()	Конструктор класса Sample для представления информации о входных данных в виде выборки значений результатов многократных измерений.
PBox2Hist()	Функция преобразования переменной типа PBox в переменную типа Hist с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).

PBox2DempsterShafer()	Функция преобразования переменной типа PBox в переменную типа DempsterShafer с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
PBox2Fuzzy()	Функция преобразования переменной типа PBox в переменную типа Fuzzy с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
PBox2FuzzyInterval()	Функция преобразования переменной типа PBox в переменную типа FuzzyInterval с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
Hist2PBox()	Функция преобразования переменной типа Hist в переменную типа PBox с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
Hist2DempsterShafer()	Функция преобразования переменной типа Hist в переменную типа DempsterShafer с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
Hist2Fuzzy()	Функция преобразования переменной типа Hist в переменную типа Fuzzy с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
Hist2FuzzyInterval()	Функция преобразования переменной типа Hist в переменную типа FuzzyInterval с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
DempsterShafer2PBox()	Функция преобразования переменной типа DempsterShafer в переменную типа PBox с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
DempsterShafer2Hist()	Функция преобразования переменной типа DempsterShafer в переменную типа Hist с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
DempsterShafer2Fuzzy()	Функция преобразования переменной типа DempsterShafer в переменную типа Fuzzy с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
DempsterShafer2 FuzzyInterval()	Функция преобразования переменной типа DempsterShafer в переменную типа FuzzyInterval с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
Fuzzy2PBox()	Функция преобразования переменной типа Fuzzy в переменную типа PBox с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
Fuzzy2Hist()	Функция преобразования переменной типа Fuzzy в переменную типа Hist с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
Fuzzy2DempsterShafer()	Функция преобразования переменной типа Fuzzy в переменную типа DempsterShafer с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
Fuzzy2FuzzyInterval()	Функция преобразования переменной типа Fuzzy в переменную типа FuzzyInterval с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).

FuzzyInterval2PBox()	Функция преобразования переменной типа FuzzyInterval в переменную типа PBox с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
FuzzyInterval2Hist()	Функция преобразования переменной типа FuzzyInterval в переменную типа Hist с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
FuzzyInterval2 DempsterShafer()	Функция преобразования переменной типа FuzzyInterval в переменную типа DempsterShafer с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
FuzzyInterval2Fuzzy()	Функция преобразования переменной типа FuzzyInterval в переменную типа Fuzzy с минимизацией потерь информации о неопределенности (квазиэквивалентное преобразование).
Sample2PBox()	Функция построения переменной типа PBox по выборке Sample значений одной из согласуемых величин.
Sample2FuzzyInterval()	Функция построения переменной типа FuzzyInterval по выборке Sample значений одной из согласуемых величин.
getStd()	Функция оценки возможных значений среднеквадратического отклонения неточной величины, представленной в одном из форматов представления неопределенности (метод классов PBox, Hist, DempsterShafer, Fuzzy, FuzzyInterval, Sample).
AnalyticalModel()	Процедура выполнения согласования неточных величин в рамках аналитической модели с решением задачи Каруша-Куна-Таккера с применением теоремы Ефремова-Козлова.
GetUncertainty()	Процедура выполнения оценки неопределенности результата согласования по аналитической модели AnalyticalModel при представлении неопределенности исходных данных в одном из перечисленных форматов: PBox, Hist, DempsterShafer, Fuzzy, FuzzyInterval, Sample.