ПРИЛОЖЕНИЕ В

**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**

**С++**

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, **Ax**= **b** или **Ax**≤ **b**).
    - **Нелинейных уравнений и неравенств** (например, **F**(**x**) = **0** или **g**(**x**) ≤ **0**).

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. ОСНОВНЫЕ ФУНКЦИИ И API**

**В.4.1. 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.

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 ниже представлена визуализация структуры библиотеки, а также входные и выходные данные.

Изображение выглядит как текст, снимок экрана, Шрифт, Параллельный

Автоматически созданное описание

Рис. – Структура библиотеки DataReconcile

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.4.2. 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 **f***M*(**x**, **a**) = **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 **f***M*(**x**, **a**) = **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 **f***M*(**x**, **a**) = **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 **f***M*(**x**, **a**) = **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 **f***M*(**x**, **a**) = **0**, описывающую взаимосвязи между согласуемыми величинами (размерность возвращаемого вектора соответствует размерности вектор-функции **f***M*),  ineqies\_model – функция (const std::function<VectorXc(const VectorXc&, const VectorXd&)>&), возвращающую результат вычисления левых частей системы неравенств вида **g***M*(**x**, **a**) ≥ **0**, представленных в формате равенств **g***M*(**x**, **a**) = **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 **f***M*(**x**, **a**) = **0**, описывающую взаимосвязи между согласуемыми величинами (размерность возвращаемого вектора соответствует размерности вектор-функции **f***M*),  ineqies\_model – функция (const std::function<VectorXc(const VectorXc&, const VectorXd&)>&), возвращающую результат вычисления левых частей системы неравенств вида **g***M*(**x**, **a**) ≥ **0**, представленных в формате равенств **g***M*(**x**, **a**) = **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 **f***M*(**x**, **a**) = **0**, описывающую взаимосвязи между согласуемыми величинами (размерность возвращаемого вектора соответствует размерности вектор-функции **f***M*),  ineqies\_model – функция (const VectorXc&, const VectorXd&)>&), возвращающую результат вычисления левых частей системы неравенств вида **g***M*(**x**, **a**) ≥ **0**, представленных в формате равенств **g***M*(**x**, **a**) = **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 **f***M*(**x**, **a**) = **0**, описывающую взаимосвязи между согласуемыми величинами (размерность возвращаемого вектора соответствует размерности вектор-функции **f***M*),  ineqies\_model – функция (const VectorXc&, const VectorXd&)>&), возвращающую результат вычисления левых частей системы неравенств вида **g***M*(**x**, **a**) ≥ **0**, представленных в формате равенств **g***M*(**x**, **a**) = **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 **f***M*(**x**, **a**) = **0**, описывающую взаимосвязи между согласуемыми величинами (размерность возвращаемого вектора соответствует размерности вектор-функции **f***M*),  ineqies\_model – функция (const std::function<VectorXc(const VectorXc&, const VectorXd&)>&), возвращающую результат вычисления левых частей системы неравенств вида **g***M*(**x**, **a**) ≥ **0**, представленных в формате равенств **g***M*(**x**, **a**) = **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 **f***M*(**x**, **a**) = **0**, описывающую взаимосвязи между согласуемыми величинами (размерность возвращаемого вектора соответствует размерности вектор-функции **f***M*),  ineqies\_model – функция (const VectorXc&, const VectorXd&)>&), возвращающую результат вычисления левых частей системы неравенств вида **g***M*(**x**, **a**) ≥ **0**, представленных в формате равенств **g***M*(**x**, **a**) = **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. | |

**В.4.3. Примеры и особенности**

Приведённая в листинге 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.

|  |
| --- |
| #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;  } |

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

**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.

|  |
| --- |
| #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;  } |

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

**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.5. 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. |