Федеральное государственное бюджетное образовательное учреждение
высшего образования «Московский государственный университет имени
М. В. Ломоносова», физический факультет

На правах рукописи

Лукьяненко Дмитрий Витальевич

Математическое моделирование, численные методы и комплекс программ для решения трёхмерных обратных задач магнитометрии

Специальность 1.2.2 —
«Математическое моделирование, численные методы и комплексы программ»

Диссертация на соискание учёной степени
доктора физико-математических наук

Научный консультант:
д. ф.-м. н, профессор
Ягола Анатолий Григорьевич

Москва — 2024

Оглавление

Стр.

Стр.

Стр.

Введение

Решение прикладных трёхмерных обратных задач магнитометрии зачастую требует столь больши́х объёмов вычислений, что эти вычисления невыполнимы на персональных компьютерах за разумное время. Чаще всего эту проблему решают с помощью упрощения базовой математической модели, которая основана на решении трёхмерного интегрального уравнения Фредгольма 1-го рода. В частности, используется подход, основанный на понижении физической размерности задачи, — рассматривается двумерная модель исходной задачи, естественной постановкой которой является постановка именно в трёхмерном пространстве. Но упрощения такого типа обычно приводят к значительному ухудшению детализации восстанавливаемого решения по сравнению с истинным. Как вариант, задача решается в полной трёхмерной постановке, но для сокращения времени вычислений для нахождения численного решения используются грубые сетки, что опять же приводит к потере детализации в восстанавливаемом приближённом решении.

С развитием возможностей вычислительной техники появился другой способ решения упомянутой проблемы — стало широко распространено использование параллельных вычислений и технологий параллельного программирования, которые позволяют решать задачу в полной постановке без использования различных упрощений и допущений. Проблема длительного счёта решается распараллеливанием вычислений между различными вычислительными узлами сложной вычислительной системы (суперкомпьютера).

При использовании такого подхода исследователи руководствуются следующей логикой: чем более мощная вычислительная система доступна для расчётов, тем больше вычислений можно совершить; а чем больше вычислений можно совершить, тем более точное и детализированное решение будет получено в результате использования более густых сеток. Но при увеличении объёма вычислений может возникнуть проблема, которая связана с накапливающимися в процессе счёта ошибками машинного округления: чем больше вычислений выполняется, тем более значительная ошибка машинного округления может накопи́ться; а чем бо́льшая ошибка машинного округления накопилась, тем менее достоверное решение будет получено. Такая проблема особенно актуальна для решения трёхмерных обратных задач магнитометрии, так как их решение сво-

дится к решению больших переопределённых систем линейных алгебраических уравнений с плотно заполненной матрицей, что приводит к необходимости выполнения большого объёма вычислений.

В результате возникает вопрос о возможности аккуратного учёта накапливающихся ошибок машинного округления при вычислениях. Ответу на этот вопрос посвящено достаточно много работ, в которых рассматриваются различные алгоритмы учёта ошибок машинного округления при решении систем алгебраических уравнений. Но ни одна из существующих на данный момент работ не учитывала следующую практическую проблему, возникающую при расчётах «больших» прикладных обратных задач (требующих нахождения сотен тысяч или миллионов неизвестных при решении систем алгебраических уравнений с плотно заполненной матрицей). Эта проблема связана с вопросом о возможности эффективной программной реализации предлагаемых вычислительных алгоритмов с использованием больших суперкомпьютерных систем со множеством вычислительных узлов. Этот вопрос является чрезвычайно важным по следующей причине. Для решения прикладных задач чаще всего применяются суперкомпьютерные (кластерные) системы, которые в нулевом приближении представляют из себя многопроцессорные системы с распределённой памятью. Наиболее распространённой технологией организации взаимодействия между различными вычислительными узлами в этом случае является технология передачи сообщений MPI (Message Passing Interface). При этом на эффективность параллельной программной реализации вычислительного алгоритма влияют накладные расходы по взаимодействию вычислительных узлов посредством использования коммуникационной сети и передачи через неё сообщений, содержащих данные промежуточных расчётов. Любой алгоритм, учитывающий ошибки машинного округления, предполагает выполнение дополнительных вычислительных операций, которые могут быть достаточно небольшими по сравнению с основными вычислениями, но выполнение которых на множестве вычислительных узлов может порождать существенное увеличение накладных расходов по взаимодействию этих вычислительных узлов между собой. То есть возможно возникновение достаточно распространённой при решении прикладных задач проблемы: предлагаемый алгоритм математически обоснован и выглядит «хорошим», однако его применение для решения прикладных задач является неэффективным.

Поэтому важным является вопрос о возможности разработки не только современных методов математического моделирования решений обратных задач магнитометрии, но и численных методов решения, которые допускают достаточно эффективную параллельную программную реализацию с использованием как технологии параллельного программирования MPI (в том числе последнего стандарта MPI-4), так и с использованием гибридных технологий параллельного программирования MPI+OpenMP+CUDA в том случае когда каждый вычислительный узел содержит многоядерный процессор и/или графический ускоритель, т.е. таким образом являясь вычислительной подсистемой с общей памятью.

При этом необходимо отметить, что одновременно с развитием возможностей вычислительной техники и программных решений по её использованию не стояло на месте и развитие технических средств проведения экспериментальных измерений. Классические постановки обратных задач магнитометрии предполагают использование в качестве входной информации только данных экспериментальных измерений компонент вектора индукции магнитного поля, индуцированного исследуемым объектом. За последние десять лет получили активное развитие технические решения, которые позволяют с достаточно высокой точностью выполнять измерения градиентов компонент магнитной индукции. Математические модели, основанные на использовании экспериментальной информации такого типа, обладают существенными преимуществами перед классическими, но до сих пор широко не применялись для математического моделирования решений обратных задач магнитометрии.

Таким образом, комбинация современных методов математического моделирования с разработкой комплекса программ, основанного на специализированных численных методах и использующего современные программные решения, должна существенным образом повысить эффективность решения трёхмерных обратных задач магнитометрии.

Целью данной работы являлась разработка методов математического моделирования, численных методов и комплекса программ для эффективного решения трёхмерных обратных задач магнитометрии.

Для достижения поставленной цели необходимо было решить следующие **задачи**:

1. Разработать методы математического моделирования решений обратных задач магнитометрии, основанные на математических моделях,

которые связывают параметры намагниченности исследуемого объекта с параметрами индуцированного им магнитного поля.

2. Разработать численные методы решения больших переопределённых систем линейных алгебраических уравнений с плотно заполненной матрицей с учётом ошибок машинного округления.
3. Разработать комплекс программ для решения трёхмерных обратных задач магнитометрии при наличии входной информации об измеренных в эксперименте значениях компонент индукции и/или компонент тензора градиентов компонент индукции магнитного поля.
4. Разработать комплекс программ для решения больших переопределённых систем линейных алгебраических уравнений с плотно заполненной матрицей с учётом ошибок машинного округления для вычислений на гетерогенных вычислительных системах, узлы которых содержат как многоядерные процессоры, так и графические ускорители.

Научная новизна:

1. Впервые предложен конструктивный способ учёта ошибок машинного округления при решении переопределённых систем линейных алгебраических уравнений с плотно заполненной матрицей, учитывающий возможности суперкомпьютерных вычислительных систем с распределённой памятью при его программной реализации.
2. Впервые реализованы параллельные итерационные алгоритмы решения больших переопределённых систем линейных алгебраических уравнений с плотно заполненной матрицей с использованием современного стандарта MPI-4 технологии параллельного программирования MPI.
3. Проведено оригинальное исследование параметров намагниченности коры планет Солнечной системы, как обладающих только остаточной намагниченностью, так и магнито-гидродинамическим динамо.

Практическая значимость заключается в том, что предложенные методы могут быть использованы для решения практически любых обратных задач магнитометрии — начиная от задач по определению параметров намагниченности небольших локализованных объектов и заканчивая задачами космического масштаба по определению параметров намагниченности коры планет и других тел Солнечной системы. В случае исследования небольших объектов предложенные методы позволяют локализовывать их месторасположение в пространстве (если в них есть некоторая доля магнитных масс), что, фактиче-

ски, является задачей магнитной локации. Знание параметров намагниченности в случае решения задач магниторазведки позволяет, при определённых условиях, получать информацию о типах залегающих в недрах планет пород, в том числе полезных ископаемых. Разработанные для решения такого типа задач численные методы и их программные реализации могут быть применены также и при решении задач из совершенно других областей науки и техники, если решение соответствующих задач сводится к решению больших переопределённых систем линейных алгебраических уравнений с плотно заполненной матрицей.

Методология и методы исследования. В работе использованы следующие теоретические методы исследования: методы математической физики, функционального анализа, вычислительной математики, линейной алгебры, теории решения обратных и некорректно поставленных задач. Прикладные методы исследования включали в себя использование технологий параллельного программирования MPI (в том числе самый последний стандарт MPI-4), OpenMP, CUDA, с программной реализацией на языках программирования Python и Fortran (в том случае если необходимы вычисления с повышенной — «четверной» — точностью).

Основные положения, выносимые на защиту:

1. Использование экспериментальных данных о компонентах тензора градиентов компонент вектора магнитной индукции позволяет восстанавливать более детализированные магнитные изображения исследуемого объекта по сравнению с использованием только экспериментальных данных о компонентах вектора магнитной индукции. При этом восстановление детализированного распределения магнитной восприимчивости в исследуемом объекте возможно только при наличии достаточно точной информации о нормальном (регулярном геомагнитном) поле в области расположения исследуемого объекта.
2. Отсутствие у планеты магнитного динамо позволяет восстановить распределение намагниченности в коре планеты по данным спутниковых измерений компонент индукции магнитного поля только исходя из допущения, что все магнитные массы были «выметены» в область коры за счёт протекавших ранее геологических процессов. Наличие у планеты магнитного динамо позволяет восстановить распределение намагниченности в коре только исходя из допущения, что модель нормального (регулярного геомагнитного) поля является достаточно точной.

3. Математическое моделирование решений трёхмерных обратных задач магнитометрии допускает разработку численных методов решения, эффективно реализуемых на современных гетерогенных суперкомпьютерных системах, состоящих из многоядерных процессоров и графических ускорителей за счёт использования современных технологий параллельного программирования MPI (включая последний стандарт MPI-4), OpenMP и CUDA.
4. Учёт ошибок машинного округления в критерии прекращения итерационного процесса в методе сопряжённых градиентов для решения систем линейных алгебраических уравнений с плотно заполненной матрицей может существенно экономить вычислительные ресурсы. Для решения достаточно больших задач параллельная программная реализация этого метода с использованием технологии параллельного программирования MPI-4 не требует дополнительных времязатрат на реализацию дополнительных вычислений по реализации усовершенствованного критерия прекращения итерационного процесса и обладает хорошими свойствами сильной масштабируемости.

Выносимые на защиту положения соответствуют следующим пунктам паспорта специальности 1.2.2 «Математическое моделирование, численные методы и комплексы программ»:

1. Положение 1 соответствует пункту 1 паспорта специальности («Разработка новых математических методов моделирования объектов и явлений»).
2. Положение 2 соответствует пункту 4 паспорта специальности («Разработка новых математических методов и алгоритмов интерпретации натурного эксперимента на основе его математической модели»).
3. Положение 3 соответствует пункту 3 паспорта специальности («Реализация эффективных численных методов и алгоритмов в виде комплексов проблемно-ориентированных программ для проведения вычислительного эксперимента») и пункту 8 («Комплексные исследования научных и технических проблем с применением современной технологии математического моделирования и вычислительного эксперимента»).

4. Положение 4 соответствует пункту 2 паспорта специальности («Разработка, обоснование и тестирование эффективных вычислительных методов с применением современных компьютерных технологий»).

Достоверность полученных результатов обеспечивается математическими методами обоснования разработанных алгоритмов, проведёнными численными экспериментами, открытым кодом комплекса программ, реализующих разработанные численные алгоритмы, публикациями в рецензируемых журналах и апробацией на российских и международных конференциях. Результаты численных расчётов находятся в соответствии с результатами, полученными другими авторами.

Апробация работы. Основные результаты работы докладывались на следующих международных и всероссийских конференциях: международная конференция «Алгоритмический анализ неустойчивых задач», посвященная памяти В. К. Иванова, г. Екатеринбург, Россия, 2011 г.; международный конгресс «8th international ISAAC congress», г. Москва, Россия, 2011 г.; международная конференция «The 3th international workshop on computational inverse problems and applications», г. Нанчанг, Китай, 2013 г.; международная конференция «Inverse problems, design and optimization», г. Альби, Франция, 2013 г.; международная конференция «Inverse problems: modeling and simulation», г. Фетхие, Турция, 2014 г.; международный конгресс «8th international congress on industrial and applied mathematics», г. Пекин, Китай, 2015 г.; международная конференция «International workshop on inverse and ill-posed problems», г. Москва, Россия, 2015 г.; международная конференция «The 4th international workshop on computational inverse problems and applications», г. Цибо, Китай, 2016 г.; международная молодежная научная школа-конференция «Теория и численные методы решения обратных и некорректных задач», посвящённая 85-летию со дня рождения академика М. М. Лаврентьева, г. Новосибирск, Россия, 2017 г.; международная конференция «Марчуковские научные чтения – 2017», г. Новосибирск, Россия, 2017 г.; международная конференция «Quasilinear equations, inverse problems and their applications», г. Москва, Россия, 2017 г.; международная конференция «Марчуковские научные чтения – 2018», г. Новосибирск, Россия, 2018 г.; международная конференция «Суперкомпьютерные технологии математического моделирования», г. Москва, Россия, 2019 г.; международная конференция «Квазилинейные уравнения, обратные задачи и их приложения», г. Долгопрудный, Россия, 2019 г.; международная

конференция «The 5th international workshop on computational inverse problems and applications», г. Лунъянь, Китай, 2019 г.; всероссийская научно-практическая конференция «Обратные задачи и математические модели», г. Бирск, Россия, 2021 г.; международная конференция «Марчуковские научные чтения – 2021», г. Новосибирск, Россия, 2021 г.; международная конференция «Евразийская конференция по прикладной математике», г. Новосибирск, Россия, 2021 г.; международная конференция «The 6th international workshop on computational inverse problems and applications», г. Шенчжень, Китай, 2022 г.; конференция «Вычислительная математика и приложения», г. Сочи, Россия, 2022 г.; международная конференция «Квазилинейные уравнения, обратные задачи и их приложения», г. Сочи, Россия, 2022 г.; всероссийская научно-практическая конференция «Обратные задачи и математические модели», г. Бирск, Россия, 2022 г.; международная конференция «Современные проблемы обратных задач», г. Новосибирск, Россия, 2022 г.; международная конференция «International conference on mathematics and the applications», г. Шенчжень, Китай, 2023 г.; международная конференция «Современные проблемы обратных задач», г. Новосибирск, Россия, 2023 г.

Основные результаты диссертационного исследования докладывались и обсуждались на следующих научных семинарах:

- в Москве:
  - НИВЦ МГУ имени М. В. Ломоносова:
    - научно-методологический семинар НИВЦ МГУ под руководством члена-корреспондента РАН Вл. В. Воеводина (10 февраля 2023 г.).
    - научный семинар «Обратные задачи математической физики» под руководством профессора А. Б. Бакушинского, профессора А. В. Тихонравова и профессора А. Г. Яголы (28 сентября 2023 г.).
- в Новосибирске:
  - объединённый научный семинар ИМ СО РАН, МЦА, НГУ, ИВМиМГ СО РАН «Обратные задачи естествознания» под руководством члена-корреспондента РАН С. И. Кабанихина и профессора РАН М. А. Шишленина (11 мая 2023 г.).
  - научный семинар «Актуальные проблемы прикладной математики» под руководством академика РАН И. А. Тайманова,

члена-корреспондента РАН С. И. Кабанихина, члена-корреспондента РАН А. Е. Миронова и профессора РАН М. А. Шишленина (1 декабря 2023 г.).

– в Иркутске: межинститутский семинар «Машинное обучение, компьютерное зрение и динамические системы» под руководством профессора РАН Д. Н. Сидорова (18 мая 2023 г.).
– в Екатеринбурге: научный семинар «Методы решения некорректных задач» под руководством члена-корреспондента РАН В. В. Васина (29 мая 2023 г.).

Публикации. Основные результаты по теме диссертации изложены в 20 печатных изданиях (6 из которых изданы в журналах, рекомендованных ВАК [1–6]; 16 [3–5; 7–19] в журналах, индексируемых в базе Web of Science; 15 [3–5; 7–10; 12–19] в журналах, индексируемых в базе Scopus); глава «Using parallel computers for solving multidimensional ill-posed problems» (авторы Д. В. Лукьяненко и А. Г. Ягола) в книге «Computational methods for applied inverse problems» (Ed. by Y. Wang, A. Yagola and C. Yang) [20]; получены 2 свидетельства о государственной регистрации программ для ЭВМ [21; 22].

Личный вклад. Все представленные в диссертационном исследовании численные методы и комплекс программ разработаны и реализованы автором лично. В работах [1; 2; 7–11; 14; 15; 20], написанных в соавторстве, вклад автора диссертации в полученные результаты в части математического моделирования, разработке численных методов и их реализации в виде комплекса программ является определяющим. В работах [3; 12; 13], написанных в соавторстве, результаты исследований были получены посредством расчётов с использованием программ, написанных соавторами на языке программирования C и в основе которых лежит структура параллельных программ, разработанных автором лично на языке программирования Fortran. В совместной работе [6] частично используются методы математического моделирования, разработанные в настоящей диссертации её автором; ссылки на результаты этой работы приводятся для целостности изложения материала, но сами результаты этой работы не включены в диссертацию и на защиту не выносятся. Используемые для математического моделирования решений обратных задач магнитометрии математические модели предложены соавторами (см. работы [4; 5; 17–19]) или известны из литературы.

Объем и структура работы. Диссертация состоит из введения, 4 глав, заключения и 2 приложений. Полный объём диссертации составляет 227 страниц, включая 46 рисунков. Список литературы содержит 190 наименований.

В **первой главе** рассматриваются различные постановки обратных задач магнитометрии. Эти постановки связывают между собой данные экспериментальных измерений компонент вектора индукции и/или компонент тензора градиентов компонент индукции магнитного поля с намагниченностью или магнитной восприимчивостью исследуемого объекта. Рассматриваемые постановки отличаются степенью физической переопределённости решаемых обратных задач.

Во **второй главе** рассматриваются основные подходы к математическому моделированию решений в прикладных трёхмерных обратных задачах магнитометрии, учитывающие особенности самых распространённых на практике прикладных задач такого типа. Так, сначала рассматривается тип задач, в которых объект является «движымым», в результате чего вспомогательная задача предобработки экспериментальных данных является достаточно простой. Далее рассматриваются обратные задачи геологоразведки в недрах Земли. Для таких задач применимы отработанные в геофизике методы выделения из экспериментальных данных необходимых для расчётов параметров магнитого поля. Затем рассматриваются проблемы геологоразведки в недрах планет Солнечной системы по данным спутниковых наблюдений. Часть таких задач предполагает достаточно простую предобработку экспериментальных данных и являются лишь вычислительно сложными, а для некоторых из таких задач для выделения необходимых для расчётов параметров магнитного поля может потребоваться решить вспомогательную обратную задачу бóльшей численной размерности по сравнению с численной размерность «основной» задачи. Содержимое этой главы основано на материалах, изложенных в работах [2; 3; 7; 9; 10; 12; 13; 15; 18].

В **третьей главе** показывается, как решаемые прикладные обратные задачи магнитометрии сводятся к решению больших переопределённых систем линейных алгебраических уравнений с плотно заполненной матрицей. Затем обсуждается способ учёта ошибок машинного округления при построении критерия прекращения итерационного процесса в градиентных методах решения таких систем. Демонстрируется, что такой подход позволяет 1) улучшить качество решения прикладных задач минимизации, решение которых требует значительных объёмов вычислений и, как следствие, может быть чувствительно

к накапливающимся ошибкам машинного округления, и 2) экономить вычислительные ресурсы. Содержимое этой главы основано на материалах, изложенных в работах [2; 4–6; 8; 11; 14; 17; 19].

В **четвёртой главе** описывается программный комплекс решения обратных задач магнитометрии с использованием технологии параллельного программирования MPI. Основная часть главы посвящена вычислительному ядру программного комплекса, а именно параллельной реализации с помощью технологии MPI алгоритма решения переопределённых систем линейных алгебраических уравнений с плотно заполненной матрицей с учётом ошибок округления. Описываются различные подходы к построению параллельной реализации алгоритма и его программной реализации с учётом возможностей различных стандартов MPI (MPI-2, MPI-3 и MPI-4). Все рассмотренные в главе примеры программ, реализующие рассмотренные алгоритмы, написаны с использованием языка программирования Python, что связано в первую очередь с относительной компактностью соответствующих программных реализаций. Однако все программы построены таким образом, что они могут быть достаточно легко переписаны на языки программирования C/C++/Fortran. Также описываются программные особенности использования на вычислительных узлах многоядерных процессоров с помощью технологии OpenMP и графических процессоров с помощью технологии CUDA. Демонстрируется то, как возможности языка программирования Python позволяют достаточно просто использовать соответствующие технические решения в рамках рассматриваемого класса прикладных задач. Созданный программный комплекс имеет открытый код и подробное описание, в связи с чем он может быть применён или модифицирован для решения широкого класса прикладных обратных задач. Содержимое этой главы основано на материалах, изложенных в работах [1; 16; 20] и зарегистрированных программах для ЭВМ [21; 22] (также см. приложение A).

Глава 1. Математические модели и постановки обратных задач магнитометрии

Всякое вещество является магнетиком, то есть способно под действием магнитного поля приобретать магнитный момент (намагничиваться). Поле, под действием которого вещество приобретает магнитный момент, принято называть *нормальным полем* и обозначать его индукцию как $\boldsymbol{B}^0$. Таким полем обычно является поле генерируемое какими-либо токами, например, магнитным динамо планеты. В результате, за счёт приобретённого магнитного момента, магнетики сами создают магнитное поле, которое в этой работе будет обозначаться как $\boldsymbol{B}_{field}$. Оба поля дают в сумме результирующее поле с индукцией

$$\boldsymbol{B} = \boldsymbol{B}^0 + \boldsymbol{B}_{field}, \tag{1.1}$$

которое измеряется в эксперименте.

Магнитный момент единицы объёма принято называть *намагниченностью* и обозначать как $\boldsymbol{M}$.

Во всех рассматриваемых далее постановках обратных задач будет предполагаться, что некоторый объём V заполнен магнитными массами с интенсивностью намагниченности (вектором намагниченности) $\boldsymbol{M}(\boldsymbol{r}) \equiv \big(M_x(\boldsymbol{r})\ M_y(\boldsymbol{r})\ M_z(\boldsymbol{r})\big)^T$, $\boldsymbol{r} \in V$. Известно (см. работу М. С. Жданова [23, с. 169]), что напряжённость магнитного поля $\boldsymbol{H}_{field}(\boldsymbol{r}_s) \equiv \big(H_x(\boldsymbol{r}_s)\ H_y(\boldsymbol{r}_s)\ H_z(\boldsymbol{r}_s)\big)^T$, индуцированного этими магнитными массами, может быть представлена в виде

$$\boldsymbol{H}_{field}(\boldsymbol{r}_s) = \frac{1}{4\pi} \nabla_s \iiint\limits_V \left(\boldsymbol{M}(\boldsymbol{r}), \nabla_s \frac{1}{|\boldsymbol{r} - \boldsymbol{r}_s|} \right) dv. \tag{1.2}$$

Здесь: $\boldsymbol{H}_{field}(\boldsymbol{r}_s)$ — вектор-функция, характеризующая напряжённость магнитного поля в точке $\boldsymbol{r}_s = (x_s,y_s,z_s)$ расположения сенсора; $\boldsymbol{M}(\boldsymbol{r})$ — вектор-функция, характеризующая плотность магнитного момента элементарного объёма dv в малой окрестности точки $\boldsymbol{r} = (x,y,z)$ области V; $\nabla_s \equiv \left(\frac{\partial}{\partial x_s}, \frac{\partial}{\partial y_s}, \frac{\partial}{\partial z_s}\right)$ — оператор вычисления градиента по переменным с индексом s.

Замечание. Необходимо подчеркнуть, что, так как намагниченность определяется как плотность магнитного момента единицы объёма вещества, термины *намагниченность* и *плотность магнитного момента* являются эквивалентными.

Магнитная индукция $\boldsymbol{B}$ и напряжённость $\boldsymbol{H}$ магнитного поля связаны соотношением

$$\boldsymbol{B} = \mu\mu_0\boldsymbol{H}. \tag{1.3}$$

Здесь: μ_0 — магнитная постоянная, μ — магнитная проницаемость среды. В случае ферромагнетиков магнитная проницаемость нелинейно зависит от напряжённости $\boldsymbol{H}^0$ внешнего (нормального) магнитного поля, поэтому формула (1.3) применима для ферромагнетиков только в случае приложения к ним достаточно малых полей. Но подавляющее большинство веществ относятся либо к классу *диамагнетиков* ($\mu \lessapprox 1$), либо к классу *парамагнетиков* ($\mu \gtrapprox 1$). Поэтому в последующих формулах значение μ будет считаться равным единице и соответствующий множитель будет опускаться. Такое допущение будет применимо и для восстановления параметров остаточной намагниченности ферромагнетиков, если важно определить факт намагниченности каких-либо частей исследуемого объёма V, пренебрегая количественными характеристиками интенсивности этой намагниченности.

Учитывая (1.3), а также то, что

$$\nabla_s \frac{1}{|\boldsymbol{r} - \boldsymbol{r}_s|} = \frac{\boldsymbol{r} - \boldsymbol{r}_s}{|\boldsymbol{r} - \boldsymbol{r}_s|^3}$$

и

$$\nabla_s \left(\boldsymbol{M}(\boldsymbol{r}), \frac{\boldsymbol{r} - \boldsymbol{r}_s}{|\boldsymbol{r} - \boldsymbol{r}_s|^3} \right) = \frac{3\big(\boldsymbol{M}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(\boldsymbol{r} - \boldsymbol{r}_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} - \frac{\boldsymbol{M}(\boldsymbol{r})}{|\boldsymbol{r} - \boldsymbol{r}_s|^3},$$

уравнение (1.2) может быть переписано в виде

$$\boldsymbol{B}_{field}(\boldsymbol{r}_s) = \frac{\mu_0}{4\pi} \iiint\limits_V \left(\frac{3\big(\boldsymbol{M}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(\boldsymbol{r} - \boldsymbol{r}_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} - \frac{\boldsymbol{M}(\boldsymbol{r})}{|\boldsymbol{r} - \boldsymbol{r}_s|^3} \right) dv. \tag{1.4}$$

Уравнение (1.4) лежит в основе всех математических моделей, которые будут использоваться для математического моделирования решений обратных задач магнитометрии.

При этом необходимо отметить, что все рассматриваемые далее постановки обратных задач подразумевают, что известна именно индукция $\boldsymbol{B}_{field}$ поля, индуцированного магнитными массами, а не полная индукция, измеряемая сенсорами в соответствие с уравнением (1.1). Таким образом, при практическом использовании таких постановок будет подразумеваться, что есть способы выделения из общего измеряемого поля $\boldsymbol{B}$ его составляющей $\boldsymbol{B}_{field}$, индуцируемой именно магнитными массами, распределение которых необходимо восстановить.

Постановки обратных задач магнитометрии будут рассмотрены в следующей последовательности. В параграфе 1.1 рассматриваются постановки обратной задачи восстановления намагниченности (плотности магнитного момента) по данным экспериментальных измерений 1) компонент вектора индукции магнитного поля (подпараграф 1.1.1), 2) компонент тензора градиентов компонент магнитной индукции (подпараграф 1.1.3), 3) полных магнито-градиентных данных (подпараграф 1.1.4). Все указанных постановки обратных задач предполагают восстановление одной вектор-функции (или, что то же самое, трёх скалярных функции) либо одной скалярной функции (подпараграф 1.1.2) по результатам экспериментального наблюдения 1) трёх скалярных функций, 2) пяти скалярных функций, 3) восьми скалярных функций. Таким образом, соответствующие постановки обратных задач будут отличаться степенью физической переопределённости решаемой задачи. Как следствие, это будет влиять на качество математического моделирования решений соответствующих обратных задач. В параграфе 1.2 рассматриваются постановки обратной задачи восстановления магнитной восприимчивости по данным аналогичных экспериментальных измерений. В этом случае по тем же экспериментальным данным при наличии дополнительной информации о параметрах нормального поля в области расположения исследуемого объекта необходимо будет восстановить уже только одну скалярную функцию (распределение магнитной восприимчивости), которая может позволить определить не только наличие и распределение магнитных масс, но и распознать тип веществ, которые образуют эти магнитные массы.

1.1 Обратная задача восстановления распределения намагниченности

Возможны постановки обратных задач магнитометрии, в которых восстанавливается либо вектор-функция намагниченности, либо только одна скалярная функция, определяющая распределение в пространстве модуля вектора намагниченности. Соответствующие постановки при практическом моделировании решений соответствующих реальных прикладных задач имеют как преимущества, так и недостатки.

1.1.1 Использование в качестве входных данных измерений компонент вектора индукции магнитного поля

Уравнение (1.4) следует выписать ещё раз, выделив красным цветом вектор-функцию, которую требуется восстановить, а зелёным цветом — данные, которые известны из экспериментальных наблюдений:

$$\boldsymbol{B}_{field}(\boldsymbol{r}_s) = \frac{\mu_0}{4\pi} \iiint\limits_V \left(\frac{3\big(\boldsymbol{M}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(\boldsymbol{r} - \boldsymbol{r}_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} - \frac{\boldsymbol{M}(\boldsymbol{r})}{|\boldsymbol{r} - \boldsymbol{r}_s|^3} \right) dv. \tag{1.5}$$

Таким образом, классическая постановка обратной задачи магнитометрии будет формулироваться следующим образом.

Обратная задача состоит в определении вектор-функции $\boldsymbol{M}(\boldsymbol{r})$, $\boldsymbol{r} \in V$, из уравнения (1.5) по данным экспериментальных измерений компонент вектор-функции $\boldsymbol{B}_{field}(\boldsymbol{r}_s)$, $\boldsymbol{r}_s \notin V$ (см. рис 1.1).

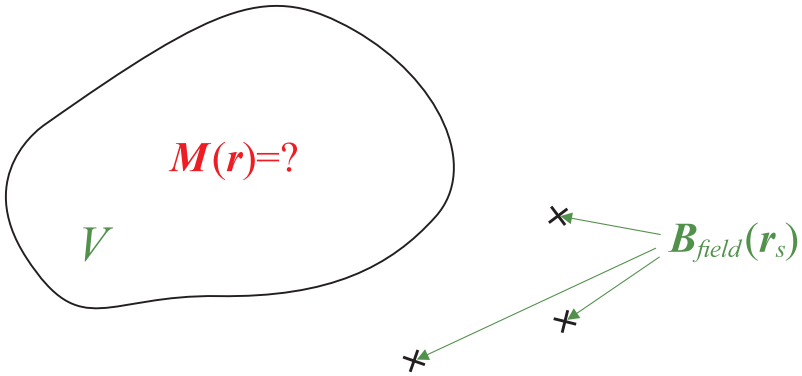


Рисунок 1.1 — Классическая постановка обратной задачи магнитометрии: необходимо восстановить вектор-функцию $\boldsymbol{M}(\boldsymbol{r})$ в области V по данным экспериментальных измерений значений компонент вектор-функции $\boldsymbol{B}_{field}(\boldsymbol{r}_s)$ вне области V.

Эта постановка является достаточно общей и не учитывает никакой априорной информации об исследуемом объекте.

С физической точки зрения такая постановка задачи является «определённой»: необходимо восстановить одну вектор-функцию по данным наблюдения тоже одной вектор-функции, либо, что то же самое, необходимо восстановить три скалярных функции (компоненты $M_x(\boldsymbol{r})$, $M_y(\boldsymbol{r})$ и $M_z(\boldsymbol{r})$ вектор-функции $\boldsymbol{M}(\boldsymbol{r})$) по данным наблюдения тоже трёх скалярных функций (компоненты $B_x(\boldsymbol{r})$, $B_y(\boldsymbol{r})$ и $B_z(\boldsymbol{r})$ вектор-функции $\boldsymbol{B}_{field}(\boldsymbol{r})$).

Уравнение (1.5) также принято записывать в более удобном для численного решения виде:

$$\boldsymbol{B}_{field}(x_s, y_s, z_s) = \frac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}_{MI}(x, y, z, x_s, y_s, z_s)\boldsymbol{M}(x, y, z)dv. \qquad (1.6)$$

Здесь $\mathbf{K}_{MI}(x, y, z, x_s, y_s, z_s)$ — матричная функция, определяющая ядро интегрального уравнения (1.6) и имеющая вид

$$\mathbf{K}_{MI}(x, y, z, x_s, y_s, z_s) = \\ = \frac{1}{\rho^5} \begin{bmatrix} 3(x-x_s)^2 - \rho^2 & 3(x-x_s)(y-y_s) & 3(x-x_s)(z-z_s) \\ 3(y-y_s)(x-x_s) & 3(y-y_s)^2 - \rho^2 & 3(y-y_s)(z-z_s) \\ 3(z-z_s)(x-x_s) & 3(z-z_s)(y-y_s) & 3(z-z_s)^2 - \rho^2 \end{bmatrix},$$

где

$$\rho = |\boldsymbol{r} - \boldsymbol{r}_s| = \sqrt{(x-x_s)^2 + (y-y_s)^2 + (z-z_s)^2}.$$

Замечание. Аббревиатура MI образована от термина «Magnetic Intensity» (интенсивность магнитного поля).

1.1.2 Использование априорной информации о параметрах нормального поля

Если известна априорная информация о нормальном поле $\boldsymbol{B}^0(\boldsymbol{r})$, $\boldsymbol{r} \in V$, в области V, то эту информацию можно использовать следующим образом.

Можно выписать соотношение

$$\boldsymbol{M}(\boldsymbol{r}) = |\boldsymbol{M}(\boldsymbol{r})|\, \boldsymbol{l}(\boldsymbol{r}), \qquad (1.7)$$

в котором $\boldsymbol{l}(\boldsymbol{r}) \equiv \big(l_x(\boldsymbol{r})\ l_y(\boldsymbol{r})\ l_z(\boldsymbol{r})\big)^T$ является единичным вектором, сонаправленным с вектором $\boldsymbol{B}^0(\boldsymbol{r})$ в точках $\boldsymbol{r} \in V$, $|\boldsymbol{M}(\boldsymbol{r})|$ — величина (модуль) вектора $\boldsymbol{M}(\boldsymbol{r})$ в точке $\boldsymbol{r}$.

Подставив (1.7) в (1.4), можно получить, что

$$\boldsymbol{B}_{field}(\boldsymbol{r}_s) = \frac{\mu_0}{4\pi} \iiint\limits_V \left(\frac{3\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(\boldsymbol{r} - \boldsymbol{r}_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} - \frac{\boldsymbol{l}(\boldsymbol{r})}{|\boldsymbol{r} - \boldsymbol{r}_s|^3} \right) \big|\boldsymbol{M}(\boldsymbol{r})\big| dv. \qquad (1.8)$$

Таким образом, можно сформулировать следующую постановку обратной задачи магнитометрии.

Обратная задача состоит в определении скалярной функции $\big|\boldsymbol{M}(\boldsymbol{r})\big|$, $\boldsymbol{r} \in V$, из уравнения (1.8) по данным экспериментальных измерений компонент вектор-функции $\boldsymbol{B}_{field}(\boldsymbol{r}_s)$, $\boldsymbol{r}_s \notin V$, и наличию информации о поле направлений $\boldsymbol{l}(\boldsymbol{r})$, $|\boldsymbol{l}| = 1$, векторов нормального магнитного поля в области V.

В этой постановке необходимо восстановить одну скалярную функцию $|\boldsymbol{M}(\boldsymbol{r})|$ по результатам экспериментальных наблюдений трёх скалярных функций — компонент вектор-функции $\boldsymbol{B}_{field}(\boldsymbol{r}_s) = \big(B_x(\boldsymbol{r}_s)\ B_y(\boldsymbol{r}_s)\ B_z(\boldsymbol{r}_s)\big)^T$. То есть полученная постановка является сильно переопределённой с физической точки зрения.

Однако существенным недостатком этой постановки является необходимость наличия информации о вектор-функции $\boldsymbol{l}(\boldsymbol{r})$. При решении практических задач соответствующие значения определяются посредством решения вспомогательных задач и, как следствие, содержат ошибку. А это означает, что при решении реальных задач рассматриваемая модель содержит ошибки, которые будут приводить к дополнительным неустойчивостям в решении соответствующей обратной задачи. Поэтому такая модель является чувствительной по отношению к точности априорной информации о вектор-функции $\boldsymbol{l}(\boldsymbol{r})$.

Уравнение (1.8) также принято записывать в более удобном для численного решения виде:

$$\boldsymbol{B}_{field}(x_s, y_s, z_s) = \frac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}^l_{MI}(x, y, z, x_s, y_s, z_s, l_x,l_y,l_z)\big|\boldsymbol{M}(x, y, z)\big|dv. \quad (1.9)$$

Здесь $\mathbf{K}^l_{MI}(x, y, z, x_s, y_s, z_s, l_x,l_y,l_z)$ — матричная функция, определяющая ядро интегрального уравнения (1.9) и имеющая вид

$$\mathbf{K}^l_{MI}(x, y, z, x_s, y_s, z_s, l_x,l_y,l_z) = \frac{1}{\rho^5} \begin{bmatrix} 3\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(x - x_s) - l_x\,\rho^2 \\ 3\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(y - y_s) - l_y\,\rho^2 \\ 3\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(z - z_s) - l_z\,\rho^2 \end{bmatrix},$$

где

$$\rho = \sqrt{(x - x_s)^2 + (y - y_s)^2 + (z - z_s)^2},$$
$$\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big) = l_x(x,y,z)\,(x - x_s) + l_y(x,y,z)\,(y - y_s) + l_z(x,y,z)\,(z - z_s).$$

1.1.3 Использование в качестве входных данных измерений компонент тензора градиентов компонент магнитной индукции

Формулу (1.4) для $\boldsymbol{B}_{field}(\boldsymbol{r}_s)$ можно преобразовать к следующему виду:

$$\begin{aligned}\boldsymbol{B}_{field}(\boldsymbol{r}_s) &= B_x(\boldsymbol{r}_s)\boldsymbol{i} + B_y(\boldsymbol{r}_s)\boldsymbol{j} + B_z(\boldsymbol{r}_s)\boldsymbol{k} = \\ &= \left(\frac{\mu_0}{4\pi}\iiint\limits_V \left(\frac{3\big(\boldsymbol{M}(\boldsymbol{r}), \boldsymbol{r}-\boldsymbol{r}_s\big)(x-x_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} - \frac{M_x(\boldsymbol{r})}{|\boldsymbol{r}-\boldsymbol{r}_s|^3}\right)dv\right)\boldsymbol{i} + \\ &+ \left(\frac{\mu_0}{4\pi}\iiint\limits_V \left(\frac{3\big(\boldsymbol{M}(\boldsymbol{r}), \boldsymbol{r}-\boldsymbol{r}_s\big)(y-y_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} - \frac{M_y(\boldsymbol{r})}{|\boldsymbol{r}-\boldsymbol{r}_s|^3}\right)dv\right)\boldsymbol{j} + \\ &+ \left(\frac{\mu_0}{4\pi}\iiint\limits_V \left(\frac{3\big(\boldsymbol{M}(\boldsymbol{r}), \boldsymbol{r}-\boldsymbol{r}_s\big)(z-z_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} - \frac{M_z(\boldsymbol{r})}{|\boldsymbol{r}-\boldsymbol{r}_s|^3}\right)dv\right)\boldsymbol{k}.\end{aligned}$$

Если использовать обозначение $i \in \{x,y,z\}$, то выражения для компонент вектор-функции $\boldsymbol{B}_{field}(\boldsymbol{r}_s)$ можно записать в следующем общем виде:

$$B_i(\boldsymbol{r}_s) = \frac{\mu_0}{4\pi}\iiint\limits_V \left(\frac{3\big(\boldsymbol{M}(\boldsymbol{r}), \boldsymbol{r}-\boldsymbol{r}_s\big)(i-i_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} - \frac{M_i(\boldsymbol{r})}{|\boldsymbol{r}-\boldsymbol{r}_s|^3}\right)dv.$$

Беря производные от $B_i(\boldsymbol{r}_s)$ по пространственным переменным $i \in \{x,y,z\}$ и $j \in \{x,y,z\} \neq i$, можно получить следующие выражения:

$$B_{ii} = \frac{\mu_0}{4\pi}\iiint\limits_V \left(\frac{6M_i(i-i_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} + \frac{3\big(\boldsymbol{M}, \boldsymbol{r}-\boldsymbol{r}_s\big)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} - \frac{15\big(\boldsymbol{M}, \boldsymbol{r}-\boldsymbol{r}_s\big)(i-i_s)^2}{|\boldsymbol{r}-\boldsymbol{r}_s|^7}\right)dv,$$

$$B_{ij} = \frac{\mu_0}{4\pi}\iiint\limits_V \left(\frac{3M_i(j-j_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} + \frac{3M_j(i-i_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} - \frac{15\big(\boldsymbol{M}, \boldsymbol{r}-\boldsymbol{r}_s\big)(i-i_s)(j-j_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^7}\right)dv.$$

Полученные формулы определяют компоненты полного тензора $\mathbf{B}_{tensor}$ градиентов компонент магнитной индукции, который в отличие от индукции магнитного поля $\boldsymbol{B}_{field}$ (которая имеет 3 компоненты) имеет 9 компонент и может быть записан в следующей матричной форме:

$$\mathbf{B}_{tensor} = [B_{ij}]\Big|_{i,j\in\overline{1,3}} \equiv \begin{bmatrix} \dfrac{\partial B_x}{\partial x} & \dfrac{\partial B_x}{\partial y} & \dfrac{\partial B_x}{\partial z} \\ \dfrac{\partial B_y}{\partial x} & \dfrac{\partial B_y}{\partial y} & \dfrac{\partial B_y}{\partial z} \\ \dfrac{\partial B_z}{\partial x} & \dfrac{\partial B_z}{\partial y} & \dfrac{\partial B_z}{\partial z} \end{bmatrix}.$$

Необходимо отметить, что матрица тензора $\mathbf{B}_{tensor}$ является симметричной, так как $\frac{\partial B_x}{\partial y} = \frac{\partial B_y}{\partial x}$, $\frac{\partial B_x}{\partial z} = \frac{\partial B_z}{\partial x}$, $\frac{\partial B_y}{\partial z} = \frac{\partial B_z}{\partial y}$. Также естественным условием является равенство нулю дивергенции вектора $\boldsymbol{B}_{field}$, так как предполагается, что в области экспериментальных измерений отсутствуют источники поля: источники сосредоточены в точках $\boldsymbol{r} \in V$, а измерения проводятся в точках $\boldsymbol{r}_s \notin V$. В результате $\frac{\partial B_x}{\partial x} + \frac{\partial B_y}{\partial y} + \frac{\partial B_z}{\partial z} = 0$. Это означает, что тензор градиентов компонент магнитной индукции содержит только 5 независимых компонент. Для определённости в качестве таких компонент будем использовать $\frac{\partial B_x}{\partial x}$, $\frac{\partial B_x}{\partial y}$, $\frac{\partial B_x}{\partial z}$, $\frac{\partial B_y}{\partial z}$, $\frac{\partial B_z}{\partial z}$.

В результате получается система

$$
\begin{cases}
\dfrac{\partial B_x}{\partial x}(\boldsymbol{r}_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \left(\dfrac{6M_x(\boldsymbol{r})(x - x_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} + \dfrac{3\big(\boldsymbol{M}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} - \right. \\
\qquad \left. - \dfrac{15\big(\boldsymbol{M}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(x - x_s)^2}{|\boldsymbol{r} - \boldsymbol{r}_s|^7} \right) dv, \\
\dfrac{\partial B_z}{\partial z}(\boldsymbol{r}_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \left(\dfrac{6M_z(\boldsymbol{r})(z - z_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} + \dfrac{3\big(\boldsymbol{M}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} - \right. \\
\qquad \left. - \dfrac{15\big(\boldsymbol{M}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(z - z_s)^2}{|\boldsymbol{r} - \boldsymbol{r}_s|^7} \right) dv, \\
\dfrac{\partial B_x}{\partial y}(\boldsymbol{r}_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \left(\dfrac{3M_x(\boldsymbol{r})(y - y_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} + \dfrac{3M_y(\boldsymbol{r})(x - x_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} - \right. \\
\qquad \left. - \dfrac{15\big(\boldsymbol{M}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(x - x_s)(y - y_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^7} \right) dv, \\
\dfrac{\partial B_x}{\partial z}(\boldsymbol{r}_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \left(\dfrac{3M_x(\boldsymbol{r})(z - z_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} + \dfrac{3M_z(\boldsymbol{r})(x - x_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} - \right. \\
\qquad \left. - \dfrac{15\big(\boldsymbol{M}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(x - x_s)(z - z_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^7} \right) dv, \\
\dfrac{\partial B_y}{\partial z}(\boldsymbol{r}_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \left(\dfrac{3M_y(\boldsymbol{r})(z - z_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} + \dfrac{3M_z(\boldsymbol{r})(y - y_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} - \right. \\
\qquad \left. - \dfrac{15\big(\boldsymbol{M}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(y - y_s)(z - z_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^7} \right) dv.
\end{cases}
\tag{1.10}
$$

Таким образом, можно сформулировать следующую постановку обратной задачи магнитометрии.

Обратная задача состоит в определении компонент вектор-функции $\boldsymbol{M}(\boldsymbol{r})$, $\boldsymbol{r} \in V$, из системы уравнений (1.10) по данным экспериментальных

измерений независимых компонент тензора градиентов компонент магнитной индукции $\mathbf{B}_{tensor}(\boldsymbol{r}_s)$, $\boldsymbol{r}_s \notin V$.

Эта постановка является достаточно общей и не учитывает никакой априорной информации об исследуемом объекте. С физической точки зрения такая постановка является «переопределённой»: необходимо восстановить 3 скалярных функции (компоненты $M_x(\boldsymbol{r})$, $M_y(\boldsymbol{r})$ и $M_z(\boldsymbol{r})$ вектор-функции $\boldsymbol{M}(\boldsymbol{r})$) по данным наблюдения 5-и скалярных функций (компоненты $\frac{\partial B_x}{\partial x}$, $\frac{\partial B_x}{\partial y}$, $\frac{\partial B_x}{\partial z}$, $\frac{\partial B_y}{\partial z}$, $\frac{\partial B_z}{\partial z}$ тензора $\mathbf{B}_{tensor}(\boldsymbol{r}_s)$).

Систему (1.10) также принято записывать в более удобном для численного решения виде:

$$\boldsymbol{B}_{tensor}(x_s, y_s, z_s) = \frac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}_{MGT}(x, y, z, x_s, y_s, z_s)\boldsymbol{M}(x, y, z)dv. \qquad (1.11)$$

Здесь $\boldsymbol{B}_{tensor} \equiv \left(\frac{\partial B_x}{\partial x} \frac{\partial B_x}{\partial y} \frac{\partial B_x}{\partial z} \frac{\partial B_y}{\partial z} \frac{\partial B_z}{\partial z}\right)^T$, $\boldsymbol{M} \equiv (M_x \quad M_y \quad M_z)^T$, а $\mathbf{K}_{MGT}(x, y, z, x_s, y_s, z_s)$ — матричная функция, определяющая ядро интегрального уравнения (1.11) и имеющая вид

$$\mathbf{K}_{MGT}(x, y, z, x_s, y_s, z_s) = \frac{3}{\rho^7} \times$$

$$\begin{bmatrix} (x-x_s)[3\rho^2 - 5(x-x_s)^2] & (y-y_s)[\rho^2 - 5(x-x_s)^2] & (z-z_s)[\rho^2 - 5(x-x_s)^2] \\ (y-y_s)[\rho^2 - 5(x-x_s)^2] & (x-x_s)[\rho^2 - 5(y-y_s)^2] & -5(x-x_s)(y-y_s)(z-z_s) \\ (z-z_s)[\rho^2 - 5(x-x_s)^2] & -5(x-x_s)(y-y_s)(z-z_s) & (x-x_s)[\rho^2 - 5(z-z_s)^2] \\ -5(x-x_s)(y-y_s)(z-z_s) & (z-z_s)[\rho^2 - 5(y-y_s)^2] & (y-y_s)[\rho^2 - 5(z-z_s)^2] \\ (x-x_s)[\rho^2 - 5(z-z_s)^2] & (y-y_s)[\rho^2 - 5(z-z_s)^2] & (z-z_s)[3\rho^2 - 5(z-z_s)^2] \end{bmatrix},$$

где

$$\rho = \sqrt{(x-x_s)^2 + (y-y_s)^2 + (z-z_s)^2}.$$

Замечание. Аббревиатура *MGT* образована от термина «Magnetic Gradient Tensor» (тензор градиентов компонент магнитного поля).

Использование априорной информации о параметрах нормального поля

По аналогии с подходом изложенном в подпараграфе 1.1.2 можно использовать априорную информацию о нормальном поле $\boldsymbol{B}^0(\boldsymbol{r})$, $\boldsymbol{r} \in V$, в области V

в виде знания векторного поля $\boldsymbol{l}(\boldsymbol{r}) \equiv \big(l_x(\boldsymbol{r})\ l_y(\boldsymbol{r})\ l_z(\boldsymbol{r})\big)^T$, которое определяет единичные вектора, сонаправленные в каждой точке $\boldsymbol{r} \in V$ с вектором $\boldsymbol{B}^0(\boldsymbol{r})$. В этом случае опять верно соотношение

$$\boldsymbol{M}(\boldsymbol{r}) = |\boldsymbol{M}(\boldsymbol{r})|\, \boldsymbol{l}(\boldsymbol{r}). \tag{1.12}$$

В частности:

$$\begin{aligned} M_x(\boldsymbol{r}) &= |\boldsymbol{M}(\boldsymbol{r})|\, l_x(\boldsymbol{r}), \\ M_y(\boldsymbol{r}) &= |\boldsymbol{M}(\boldsymbol{r})|\, l_y(\boldsymbol{r}), \\ M_z(\boldsymbol{r}) &= |\boldsymbol{M}(\boldsymbol{r})|\, l_z(\boldsymbol{r}). \end{aligned} \tag{1.13}$$

Подставив (1.12) и (1.13) в (1.10), можно получить

$$\begin{cases} \dfrac{\partial B_x}{\partial x}(\boldsymbol{r}_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \left(\dfrac{6\, l_x(\boldsymbol{r})(x - x_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} + \dfrac{3\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} - \right. \\ \qquad \left. - \dfrac{15\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(x - x_s)^2}{|\boldsymbol{r} - \boldsymbol{r}_s|^7} \right) |\boldsymbol{M}(\boldsymbol{r})|\, dv, \\ \dfrac{\partial B_z}{\partial z}(\boldsymbol{r}_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \left(\dfrac{6\, l_z(\boldsymbol{r})(z - z_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} + \dfrac{3\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} - \right. \\ \qquad \left. - \dfrac{15\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(z - z_s)^2}{|\boldsymbol{r} - \boldsymbol{r}_s|^7} \right) |\boldsymbol{M}(\boldsymbol{r})|\, dv, \\ \dfrac{\partial B_x}{\partial y}(\boldsymbol{r}_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \left(\dfrac{3\, l_x(\boldsymbol{r})(y - y_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} + \dfrac{3\, l_y(\boldsymbol{r})(x - x_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} - \right. \\ \qquad \left. - \dfrac{15\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(x - x_s)(y - y_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^7} \right) |\boldsymbol{M}(\boldsymbol{r})|\, dv, \\ \dfrac{\partial B_x}{\partial z}(\boldsymbol{r}_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \left(\dfrac{3\, l_x(\boldsymbol{r})(z - z_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} + \dfrac{3\, l_z(\boldsymbol{r})(x - x_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} - \right. \\ \qquad \left. - \dfrac{15\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(x - x_s)(z - z_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^7} \right) |\boldsymbol{M}(\boldsymbol{r})|\, dv, \\ \dfrac{\partial B_y}{\partial z}(\boldsymbol{r}_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \left(\dfrac{3\, l_y(\boldsymbol{r})(z - z_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} + \dfrac{3\, l_z(\boldsymbol{r})(y - y_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} - \right. \\ \qquad \left. - \dfrac{15\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(y - y_s)(z - z_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^7} \right) |\boldsymbol{M}(\boldsymbol{r})|\, dv. \end{cases} \tag{1.14}$$

Таким образом, можно сформулировать следующую постановку обратной задачи магнитометрии.

Обратная задача состоит в определении скалярной функции $\big|\boldsymbol{M}(\boldsymbol{r})\big|$, $\boldsymbol{r} \in V$, из системы уравнений (1.14) по данным экспериментальных измерений независимых компонент тензора градиентов компонент магнитной индукции $\mathbf{B}_{tensor}(\boldsymbol{r}_s)$, $\boldsymbol{r}_s \notin V$, и наличию информации о поле направлений $\boldsymbol{l}(\boldsymbol{r})$, $|\boldsymbol{l}| = 1$, векторов нормального магнитного поля в области V.

В этой постановке необходимо восстановить 1 скалярную функцию $|\boldsymbol{M}(\boldsymbol{r})|$ по результатам экспериментальных наблюдений 5 скалярных функций (компоненты $\frac{\partial B_x}{\partial x}$, $\frac{\partial B_x}{\partial y}$, $\frac{\partial B_x}{\partial z}$, $\frac{\partial B_y}{\partial z}$, $\frac{\partial B_z}{\partial z}$ тензора $\mathbf{B}_{tensor}$). То есть с физической точки зрения полученная постановка является сильно переопределённой.

Однако также как и в постановке обратной задачи из подпараграфа 1.1.2, существенным недостатком этой постановки является знание вектор-функции $\boldsymbol{l}(\boldsymbol{r})$. При решении практических задач соответствующие значения определяются посредством решения вспомогательных задач и, как следствие, содержат ошибку. А это означает, что при решении реальных задач модель содержит ошибки, которые будут приводить к дополнительным неустойчивостям в решении соответствующей обратной задачи.

Систему (1.14) также принято записывать в более удобном для численного решения виде:

$$\boldsymbol{B}_{tensor}(x_s, y_s, z_s) = \frac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}^l_{MGT}(x, y, z, x_s, y_s, z_s, l_x,l_y,l_z)\big|\boldsymbol{M}(x, y, z)\big|dv. \quad (1.15)$$

Здесь $\mathbf{K}^l_{MGT}(x, y, z, x_s, y_s, z_s, l_x,l_y,l_z)$ — матричная функция, определяющая ядро интегрального уравнения (1.15) и имеющая вид

$$\mathbf{K}^l_{MGT}(x, y, z, x_s, y_s, z_s, l_x,l_y,l_z) = \\ = \frac{1}{\rho^7}\begin{bmatrix} 6\rho^2 l_x(x-x_s) + 3\rho^2\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r}-\boldsymbol{r}_s\big) - 15\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r}-\boldsymbol{r}_s\big)(x-x_s)^2 \\ 6\rho^2 l_z(z-z_s) + 3\rho^2\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r}-\boldsymbol{r}_s\big) - 15\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r}-\boldsymbol{r}_s\big)(z-z_s)^2 \\ 3\rho^2 l_x(y-y_s) + 3\rho^2 l_y(x-x_s) - 15\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r}-\boldsymbol{r}_s\big)(x-x_s)(y-y_s) \\ 3\rho^2 l_x(z-z_s) + 3\rho^2 l_z(x-x_s) - 15\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r}-\boldsymbol{r}_s\big)(x-x_s)(z-z_s) \\ 3\rho^2 l_y(z-z_s) + 3\rho^2 l_z(y-y_s) - 15\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r}-\boldsymbol{r}_s\big)(y-y_s)(z-z_s) \end{bmatrix},$$

где

$$\rho = \sqrt{(x-x_s)^2 + (y-y_s)^2 + (z-z_s)^2},$$
$$\big(\boldsymbol{l}(\boldsymbol{r}), \boldsymbol{r}-\boldsymbol{r}_s\big) = l_x(x,y,z)\,(x-x_s) + l_y(x,y,z)\,(y-y_s) + l_z(x,y,z)\,(z-z_s).$$

1.1.4 Обращение полных магнито-градиентных данных

Если обе математические модели, описанные в подпараграфах 1.1 и 1.3, объединить, то это даст систему уравнений

$$\begin{cases} \boldsymbol{B}_{field}(x_s, y_s, z_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}_{MI}(x, y, z, x_s, y_s, z_s)\boldsymbol{M}(x, y, z)dv, \\ \boldsymbol{B}_{tensor}(x_s, y_s, z_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}_{MGT}(x, y, z, x_s, y_s, z_s)\boldsymbol{M}(x, y, z)dv, \end{cases}$$

для случая отсутствия априорной информации о нормальном поле $\boldsymbol{B}^0(\boldsymbol{r})$, $\boldsymbol{r} \in V$.

В случае же наличия такой информации о поле направлений $\boldsymbol{l}(\boldsymbol{r})$ векторов нормального магнитного поля $\boldsymbol{B}^0(\boldsymbol{r})$ в области V (см. подпараграф 1.2) совместная математическая модель примет вид

$$\begin{cases} \boldsymbol{B}_{field}(x_s, y_s, z_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}^l_{MI}(x, y, z, x_s, y_s, z_s, l_x, l_y, l_z)|\boldsymbol{M}(x, y, z)|dv, \\ \boldsymbol{B}_{tensor}(x_s, y_s, z_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}^l_{MGT}(x, y, z, x_s, y_s, z_s, l_x, l_y, l_z)|\boldsymbol{M}(x, y, z)|dv. \end{cases}$$

Эти системы можно формально переписать в виде одного интегрального уравнения Фредгольма 1-го рода:

$$\mathbf{A}\,\boldsymbol{M} \equiv \frac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}(x, y, z, x_s, y_s, z_s)\, \boldsymbol{M}(x,y,z)\, dxdydz = \boldsymbol{B}(x_s,y_s,z_s). \qquad (1.16)$$

Здесь возможны следующие варианты структуры матричной функции $\mathbf{K}$ и вектор-функций $\boldsymbol{M}$ и $\boldsymbol{B}$.

1. Случай обработки «магнитных» данных:
   $\boldsymbol{M} \equiv \left(M_x\, M_y\, M_z\right)^T$, $\boldsymbol{B} \equiv \left(B_x\, B_y\, B_z\right)^T$, $\mathbf{K} \equiv \mathbf{K}_{MI}$.
2. Случай обработки «градиентных» данных:
   $\boldsymbol{M} \equiv \left(M_x\, M_y\, M_z\right)^T$, $\boldsymbol{B} \equiv \left(\frac{\partial B_x}{\partial x}\, \frac{\partial B_x}{\partial y}\, \frac{\partial B_x}{\partial z}\, \frac{\partial B_y}{\partial z}\, \frac{\partial B_z}{\partial z}\right)^T$, $\mathbf{K} \equiv \mathbf{K}_{MGT}$.
3. Случай обработки «магнито-градиентных» данных:
   $\boldsymbol{M} \equiv \left(M_x\, M_y\, M_z\right)^T$, $\boldsymbol{B} \equiv \left(B_x\, B_y\, B_z\, \frac{\partial B_x}{\partial x}\, \frac{\partial B_x}{\partial y}\, \frac{\partial B_x}{\partial z}\, \frac{\partial B_y}{\partial z}\, \frac{\partial B_z}{\partial z}\right)^T$,
   $\mathbf{K} \equiv \left(\mathbf{K}_{MI}\, \mathbf{K}_{MGT}\right)^T$.

4. Случай обработки «магнитных» данных при наличии априорной информации о векторном поле $\boldsymbol{l}(\boldsymbol{r})$:
   $\boldsymbol{M} \equiv |\boldsymbol{M}|$, $\boldsymbol{B} \equiv \left(B_x\, B_y\, B_z\right)^T$, $\mathbf{K} \equiv \mathbf{K}^l_{MI}$.
5. Случай обработки «градиентных» данных при наличии априорной информации о векторном поле $\boldsymbol{l}(\boldsymbol{r})$:
   $\boldsymbol{M} \equiv |\boldsymbol{M}|$, $\boldsymbol{B} \equiv \left(\frac{\partial B_x}{\partial x}\, \frac{\partial B_x}{\partial y}\, \frac{\partial B_x}{\partial z}\, \frac{\partial B_y}{\partial z}\, \frac{\partial B_z}{\partial z}\right)^T$, $\mathbf{K} \equiv \mathbf{K}^l_{MGT}$.
6. Случай обработки «магнито-градиентных» данных при наличии априорной информации о векторном поле $\boldsymbol{l}(\boldsymbol{r})$:
   $\boldsymbol{M} \equiv |\boldsymbol{M}|$, $\boldsymbol{B} \equiv \left(B_x\, B_y\, B_z\, \frac{\partial B_x}{\partial x}\, \frac{\partial B_x}{\partial y}\, \frac{\partial B_x}{\partial z}\, \frac{\partial B_y}{\partial z}\, \frac{\partial B_z}{\partial z}\right)^T$,
   $\mathbf{K} \equiv \left(\mathbf{K}^l_{MI}\, \mathbf{K}^l_{MGT}\right)^T$.

Таким образом, все рассмотренные постановки обратных задач магнитометрии сводятся к необходимости решения интегрального уравнения Фредгольма 1-го рода вида (1.16) с достаточно произвольной областью V.

При этом постановки обратных задач для всех случаев кроме первого являются физически переопределёнными: необходимо найти некоторый набор скалярных функций по значительно бóльшему числу экспериментально наблюдаемых других скалярных функций.

1.2 Обратная задача восстановления распределения магнитной восприимчивости

Рассматриваемые в этом параграфе постановки обратных задач магнитометрии дают возможность восстановить не только распределение магнитных масс в некоторой области V, но также позволяют распознать конкретные виды веществ, из которых состоят магнитные массы. Но для этого необходимо будет использовать полную априорную информацию о нормальном поле $\boldsymbol{B}^0(\boldsymbol{r})$ в области V, то есть не только информацию о поле направлений $\boldsymbol{l}(\boldsymbol{r})$ векторов нормального магнитного поля в области V, но также и информацию о модуле $\left|\boldsymbol{B}^0(\boldsymbol{r})\right|$ в каждой точке этой области. Эту информацию можно использовать следующим образом. Можно выписать соотношение

$$\boldsymbol{M}(\boldsymbol{r}) = \chi(\boldsymbol{r})\boldsymbol{H}^0(\boldsymbol{r}) = \frac{\chi(\boldsymbol{r})}{\mu_0\mu(\boldsymbol{r})}\boldsymbol{B}^0(\boldsymbol{r}).$$

Здесь χ — магнитная восприимчивость магнитного вещества. При этом $|\chi| \ll 1$ для *диа-* и *пара*магнетиков.

Если учесть, что $\mu = 1 + \chi$, то можно получить, что

$$\boldsymbol{M}(\boldsymbol{r}) = \frac{\chi(\boldsymbol{r})}{\mu_0\big(1+\chi(\boldsymbol{r})\big)}\boldsymbol{B}^0(\boldsymbol{r}).$$

Если использовать разложение

$$\frac{\chi}{1+\chi} = \chi - \chi^2 + 2\chi^3 - 6\chi^4 + \cdots$$

и пренебречь в нём всеми членами разложения с порядками выше первого, то можно получить, что

$$\boldsymbol{M}(\boldsymbol{r}) = \frac{\chi(\boldsymbol{r})}{\mu_0}\boldsymbol{B}^0(\boldsymbol{r}). \tag{1.17}$$

Замечание. Необходимо подчеркнуть, что условие применимости формулы (1.17): $|\chi(\boldsymbol{r})| \ll 1$. То есть эта формула верная только для *диа-* и *пара*магнетиков.

Подставив (1.17) в (1.4), можно получить

$$\boldsymbol{B}_{field}(\boldsymbol{r}_s) = \frac{1}{4\pi}\iiint\limits_V \left(\frac{3\big(\boldsymbol{B}^0(\boldsymbol{r}), \boldsymbol{r}-\boldsymbol{r}_s\big)(\boldsymbol{r}-\boldsymbol{r}_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} - \frac{\boldsymbol{B}^0(\boldsymbol{r})}{|\boldsymbol{r}-\boldsymbol{r}_s|^3}\right)\chi(\boldsymbol{r})dv. \tag{1.18}$$

Таким образом, можно сформулировать следующую постановку обратной задачи магнитометрии.

Обратная задача состоит в определении скалярной-функции $\chi(\boldsymbol{r})$, $\boldsymbol{r} \in V$, из уравнения (1.18) по данным экспериментальных измерений компонент вектор-функции $\boldsymbol{B}_{field}(\boldsymbol{r}_s)$, $\boldsymbol{r}_s \notin V$, и наличию информации о нормальном магнитном поле $\boldsymbol{B}^0(\boldsymbol{r})$ в области V.

В этой постановке необходимо восстановить одну скалярную функцию $\chi(\boldsymbol{r})$ по результатам экспериментальных наблюдений трёх скалярных функций — компонент вектор-функции $\boldsymbol{B}_{field}(\boldsymbol{r}_s) = \big(B_x(\boldsymbol{r}_s)\ B_y(\boldsymbol{r}_s)\ B_z(\boldsymbol{r}_s)\big)^T$. То есть полученная постановка является сильно переопределённой с физической точки зрения.

Однако существенным недостатком этой постановки является необходимость знания вектор-функции $\boldsymbol{B}^0(\boldsymbol{r})$. При решении практических задач соответствующие значения определяются посредством решения вспомогательных задач и, как следствие, содержат ошибку. А это означает, что при решении

реальных задач модель содержит ошибки, которые будут приводить к дополнительным неустойчивостям в решении соответствующей обратной задачи.

Наравне со сформулированной постановкой задачи обработки «магнитных» данных можно сформулировать и обратную задачу по обработке «градиентных» данных. Для этого надо подставить (1.17) в (1.10). В результате будет получена система

$$
\begin{cases}
\dfrac{\partial B_x}{\partial x}(\boldsymbol{r}_s) = \dfrac{1}{4\pi}\iiint\limits_V \left(\dfrac{6B_x^0(\boldsymbol{r})(x-x_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} + \dfrac{3\big(\boldsymbol{B}^0(\boldsymbol{r}),\boldsymbol{r}-\boldsymbol{r}_s\big)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} - \right. \\
\qquad\qquad \left. - \dfrac{15\big(\boldsymbol{B}^0(\boldsymbol{r}),\boldsymbol{r}-\boldsymbol{r}_s\big)(x-x_s)^2}{|\boldsymbol{r}-\boldsymbol{r}_s|^7}\right)\chi(\boldsymbol{r})dv, \\
\dfrac{\partial B_z}{\partial z}(\boldsymbol{r}_s) = \dfrac{1}{4\pi}\iiint\limits_V \left(\dfrac{6B_z^0(\boldsymbol{r})(z-z_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} + \dfrac{3\big(\boldsymbol{B}^0(\boldsymbol{r}),\boldsymbol{r}-\boldsymbol{r}_s\big)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} - \right. \\
\qquad\qquad \left. - \dfrac{15\big(\boldsymbol{B}^0(\boldsymbol{r}),\boldsymbol{r}-\boldsymbol{r}_s\big)(z-z_s)^2}{|\boldsymbol{r}-\boldsymbol{r}_s|^7}\right)\chi(\boldsymbol{r})dv, \\
\dfrac{\partial B_x}{\partial y}(\boldsymbol{r}_s) = \dfrac{1}{4\pi}\iiint\limits_V \left(\dfrac{3B_x^0(\boldsymbol{r})(y-y_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} + \dfrac{3B_y^0(\boldsymbol{r})(x-x_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} - \right. \\
\qquad\qquad \left. - \dfrac{15\big(\boldsymbol{B}^0(\boldsymbol{r}),\boldsymbol{r}-\boldsymbol{r}_s\big)(x-x_s)(y-y_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^7}\right)\chi(\boldsymbol{r})dv, \\
\dfrac{\partial B_x}{\partial z}(\boldsymbol{r}_s) = \dfrac{1}{4\pi}\iiint\limits_V \left(\dfrac{3B_x^0(\boldsymbol{r})(z-z_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} + \dfrac{3B_z^0(\boldsymbol{r})(x-x_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} - \right. \\
\qquad\qquad \left. - \dfrac{15\big(\boldsymbol{B}^0(\boldsymbol{r}),\boldsymbol{r}-\boldsymbol{r}_s\big)(x-x_s)(z-z_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^7}\right)\chi(\boldsymbol{r})dv, \\
\dfrac{\partial B_y}{\partial z}(\boldsymbol{r}_s) = \dfrac{1}{4\pi}\iiint\limits_V \left(\dfrac{3B_y^0(\boldsymbol{r})(z-z_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} + \dfrac{3B_z^0(\boldsymbol{r})(y-y_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} - \right. \\
\qquad\qquad \left. - \dfrac{15\big(\boldsymbol{B}^0(\boldsymbol{r}),\boldsymbol{r}-\boldsymbol{r}_s\big)(y-y_s)(z-z_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^7}\right)\chi(\boldsymbol{r})dv.
\end{cases}
\tag{1.19}
$$

Таким образом, можно сформулировать следующую постановку обратной задачи магнитометрии.

Обратная задача состоит в определении скалярной функции $\chi(\boldsymbol{r})$, $\boldsymbol{r} \in V$, из системы уравнений (1.19) по данным экспериментальных измерений независимых компонент тензора градиентов компонент магнитной индукции $\mathbf{B}_{tensor}(\boldsymbol{r}_s)$, $\boldsymbol{r}_s \notin V$, и наличию информации о нормальном магнитном поле $\boldsymbol{B}^0(\boldsymbol{r})$ в области V.

С физической точки зрения такая постановка является «сильно переопределённой»: необходимо восстановить 1 скалярную функцию $\chi(\boldsymbol{r})$ по данным наблюдения 5-и скалярных функций (компоненты $\frac{\partial B_x}{\partial x}$, $\frac{\partial B_x}{\partial y}$, $\frac{\partial B_x}{\partial z}$, $\frac{\partial B_y}{\partial z}$, $\frac{\partial B_z}{\partial z}$ тензора $\mathbf{B}_{tensor}(\boldsymbol{r}_s)$).

Аналогично материалу подпараграфа 1.1.3 можно поставить обратную задачу магнитометрии и по обращению полных магнито-градиентных данных.

1.3 Обзор литературы по теме диссертационного исследования

Математические модели в магнитостатике, связывающие плотность магнитного момента ограниченного тела и индуцируемое им магнитное поле, достаточно хорошо известны (см., например, работы Ю. Г. Ли [24], П. Г. Леливра [25], А. Пиньятелли [26]). Постановки прямых и обратных задач, основанных на этих моделях, принято классифицировать следующим образом (см., например, работу М. С. Жданова [27]): если известны параметры намагниченности тела и надо рассчитать параметры индуцируемого им магнитного поля, то такую задачу принято называть *прямой*; если же известны измеренные в эксперименте параметры магнитного поля и надо восстановить параметры намагниченности тела, то такую задачу принято называть *обратной*.

Необходимо отметить, что здесь специально делается акцент именно на измеряемых в эксперименте *параметрах магнитного поля*, а не просто об измеряемом в эксперименте *магнитном поле*.

Традиционными параметрами магнитного поля, измеряемыми при экспериментальных наблюдениях, являются либо общая индукция магнитного поля (то есть модуль вектора индукции — см., например, работу Р. Г. Хендерсона [28]) либо набор из трёх компонент вектора индукции магнитного поля (разложение этого вектора по некоторому базису в трёхмерном пространстве). То же относится и к *параметрам намагниченности* тела. В самой общей постановке обратных задач магнитометрии, при ограниченности данных, принято восстанавливать векторную величину — намагниченность (плотность магнитного момента) тела. Если же имеется априорная информация об индукции нормального поля (например, об индуцированном Землёй магнитном поле) в области расположения тела, обратную задачу восстановления векторной функции можно свести к об-

ратной задаче поиска скалярной функции, определяющей либо модуль вектора намагниченности либо магнитную восприимчивость. В таких постановках физическая переопределённость моделей получается более высокой, что в теории позволяет восстанавливать искомые параметры намагниченности более качественно при точно заданной априорной информации (см., например, работу Я. Вана [2]). Более того, возможность восстановить магнитную восприимчивость позволяет также распознавать магнитные материалы, из которых состоит исследуемое тело.

Использованию для восстановления параметров намагниченности экспериментальных измерений компонент индукции магнитного поля посвящено достаточно много работ. Основные классические подходы к решению соответствующих обратных задач описаны, например, в работах О. Портнягина и М. С. Жданова [29; 30]. Современные подходы в основном используют различные способы учёта априорной информации о решении с целью более качественного восстановления искомых параметров намагниченности (см., например, работы С. Вана и Р. О. Хансена [31], Я. Г. Ли и Д. В. Ольденбурга [24], А. Пиньятелли и др. [26]).

А вот использованию для восстановления параметров намагниченности экспериментальных измерений компонент тензора градиентов компонент магнитной индукции до некоторых пор было посвящено достаточно мало работ, среди которых следует отметить работы А. Кристенсена [32], П. В. Шмидта [33; 34], П. Хита [35], М. Шиффлера [36] и М. С. Жданова [37]. С разработкой высокотемпературных сверхпроводящих квантовых интерференционных устройств (SQUID — «Superconducting Quantum Interference Device») появилась возможность измерять компоненты тензора градиентов компонент индукции магнитного поля с достаточно высокой точностью (см., например, работы П. В. Шмидта [33; 34] и М. С. Жданова [37]). Измерения градиентов компонент индукции магнитного поля предоставляют как минимум ценную дополнительную информацию по сравнению с обычными измерениями индукции магнитного поля, так как чем больше информации об объекте исследования есть в наличии, тем более качественный результат можно получить в результате решения соответствующей обратной задачи. В результате возникло много обсуждений о преимуществах магнитометрических измерений с тензором градиентов компонент магнитного поля по сравнению с обычными измерениями компонент вектора магнитной индукции (здесь можно выделить работы

П. В. Шмидта [33; 34], П. Хита [35], М. Шиффлера [36] и С. С. Джи [38]). По итогу, многочисленные экспериментальные работы установили, что тензор градиентов компонент индукции магнитного поля в таких моделях обладает большей чувствительностью к тонкой структуре распределения магнитного момента, чем сама магнитная индукция. Поэтому измерения компонент тензора градиентов компонент индукции магнитного поля представляются более перспективными для интерпретации магнитных полей с помощью решения обратных задач.

Большинство указанных обратных задач магнитометрии, как было показано в подпараграфе 1.1.4 параграфа 1.1 этой главы, сводятся к операторному уравнению первого рода вида (1.16). Согласно работе А. Г. Яголы [39] все такие геофизические задачи являются некорректно поставленными по Ж. Адамару (см. его основополагающую работу [40]). Они могут как иметь неединственное решение, так и не иметь классического решения вообще. При этом эти задачи, как правило, неустойчивы по отношению к ошибкам измерения входных данных (компонент индукции и/или компонент тензора градиентов компонент индукции магнитного поля). Эти трудности преодолеваются с помощью применения специальных методов решения таких некорректно поставленных задач — регуляризирующих алгоритмов.

Академиком А. Н. Тихоновым в 60-х годах прошлого века была заложена теория решения некорректно поставленных задач, основанная на понятии регуляризирующего алгоритма (см. работы [41;42]). После основополагающих работ А. Н. Тихонова [41–46], М. М. Лаврентьева [47;48] и В. К. Иванова [49–52] теория некорректных задач была развита многими учеными в применении к разным областям науки и техники. В частности следует выделить книги В. В. Васина [53–56], С. И. Кабанихина [57–62], В. Г. Романова [63–65], В. П. Тананы [66], К. Г. Резницкой [67], Г. М. Вайникко [68; 69], А. М. Федотова [70; 71], А. Л. Бухгейма [72; 73], В. Б Гласко [74], А. В. Гончарского [75], В. Я. Арсенина [76], С. Ф. Гилязова [77], В. А. Морозова [78], О. М. Алифанова [79], А. Б. Бакушинского [80], Ч. У. Гроетча [81], И. В. Кочикова [82], А. М. Денисова [83], А. С. Леонова [84; 85], Х. В. Энгля [86], Л. Я. Савельева [87], Ю. С. Осипова [88].

Важнейшим классом регуляризирующих алгоритмов с прикладной точки зрения является семейство вариационных регуляризирующих алгоритмов. Одним из наиболее известным из них является регуляризирующий алгоритм, основанный на минимизации функционала А. Н. Тихонова. Как и любой дру-

гой регуляризирующий алгоритм, он включает в себя параметр регуляризации, необоснованный выбор которого может кардинальным образом повлиять на регуляризованное решение. При этом, согласно работе А. Б. Бакушинского [89] невозможно построить способ выбора параметра регуляризации, который не был бы согласован с погрешностью задания входных данных. К сожалению, по теме решения обратных задач геофизики (в том числе и магнитометрии) на данный момент существует достаточно большое число работ, в которых авторы используют эвристические способы выбора параметра регуляризации, которые противоречат этому «вето Бакушинского». Чаще всего авторы таких работ получают приемлемые результаты в связи с тем, что решаемые ими задачи за счёт использования различной априорной информации на самом деле являются корректно поставленными, но недостаточные теоретические исследования не выявили этого факта. Однако существуют и работы, результаты которых являются недостоверными. В работе А. Г. Яголы, А. С. Леонова и В. Н. Титаренко [90] приведён обзор таких самых популярных эвристических способов выбора параметра регуляризации и дано их опровержение в части использования при решении некорректно поставленных задач. В случае решения некорректно поставленных обратных задач одним из самых популярных обоснованных способов согласования параметра регуляризации с погрешностью задания входных данных является обобщённый принцип невязки (см. работу А. Н. Тихонова, А. В. Гончарского, В. В. Степанова и А. Г. Яголы [91]), который является обобщением принципа невязки В. А. Морозова (см. работу [92]) на случай наличия ошибок в операторе.

Необходимо также отметить, что существуют и статистические подходы к построению регуляризирующих алгоритмов для решения задач рассматриваемого класса (см., например, работу А. Тарантолы [93]). Этот подход основан на статистической теории, предполагающей, что данные и модель являются неопределёнными и подчиняются распределению Гаусса. Подходы основанные на статистической регуляризации и на теории регуляризации А. Н. Тихонова [91] являются эквивалентными при определённых условиях. Поэтому в диссертационной работе в основе всех численных подходов лежат регуляризирующие алгоритмы, основанные на минимизации функционала А. Н. Тихонова.

Следующая проблема, которой посвящено множество работ, связана с трудностями численного счёта. Многие современные постановки прикладных обратных задач магнитометрии предполагают восстановление параметров

намагниченности исследуемых объектов в пространстве. Это приводит к необходимости решения трёхмерных интегральных уравнений Фредгольма 1-го рода для векторных или скалярных функций, что невозможно сделать за разумное время с использованием обычных персональных компьютеров. Есть два пути решения этой проблемы.

Первый путь подразумевает использование различных упрощений и допущений, которые понижают физическую размерность решаемой задачи, но при этом дают ограниченную информацию об исследуемом объекте либо приводят к существенным ошибкам в восстанавливаемых магнитных параметрах исследуемых тел. Такому подходу посвящено большинство процитированных работ, в которых рассматриваются прикладные методы решения трёхмерных обратных задач магнитометрии.

Второй путь предполагает решение рассматриваемых задач в полных постановках, что требует использования для расчётов серьёзных вычислительных ресурсов, в частности использования суперкомпьютерных систем. Современное развитие суперкомпьютерных технологий в недавнее время позволило решать действительно огромные в вычислительном смысле задачи (см., например, работы Вл. В. Воеводина и др. [94–96]). В части решения рассматриваемых в диссертационной работе задач следует отметить работы Е. Н. Акимовой и др. [97–99]. Однако возможность использования мощных вычислительных систем привела к возникновению очередной проблемы. С ростом числа вычислений растёт и влияние ошибок машинного округления, которые могут кардинальным образом повлиять на точность получаемого приближённого решения. При этом с целью уменьшения ошибок конечно-разностной аппроксимации приходится увеличивать размерности сеток, в узлах которых ищется приближённое численное решение. А это в свою очередь приводит к дополнительному накоплению ошибок машинного округления за счёт увеличения числа вычислений. В связи с этим учёт ошибок машинного округления при решении многомерных обратных задач, которые приводят к необходимости решения больших переопределённых систем линейных алгебраических уравнений с плотно заполненной матрицей, стал особо актуален последнее время.

Подобным вопросом посвящено множество работ, среди которых следует выделить, например, работы В. В. Воеводина [100], Х. Возняковского [101], Э. Ф. Каашитера [102], З. Стракова [103; 104], М. А. Ариоли [105; 106], И. Но-

тайа [107], Ж. Мерана [108], Н. Н. Калиткина [109], Н. Дж. Хайэма [110], М. Крочи [111], М. П. Коннолли [112].

Все указанные работы носят преимущественно теоретический характер и не рассматривают другую распространённую прикладную проблему, которая заключается в том, что предложенный алгоритм может являться математически обоснованным и выглядеть «хорошим», однако его применение для решения реальных прикладных задач будет неэффективным. Такие ситуации на практике возникают в первую очередь по той причине, что использование для вычислений многопроцессорных систем влечёт и дополнительные времязатраты, порождаемые необходимостью взаимодействия между собой вычислительных узлов, рассчитывающих различные части одной большой задачи (см., например, работы Э. Д'Азеведо [113], Э. Де Стерлера [114], П. Р. Эллера [115]). Поэтому любой численный алгоритм должен быть «согласован» с возможностями многопроцессорных систем, которые будут использоваться для его реальной численной реализацией. До сих пор публикаций на эту тему не существовало. Диссертационное исследование, в частности, закрывает и этот пробел в литературе.

Глава 2. Математические методы моделирования в прикладных трёхмерных обратных задачах магнитометрии

В этой главе рассматриваются основные подходы к математическому моделированию решений в прикладных трёхмерных обратных задачах магнитометрии, учитывающие особенности самых распространённых на практике прикладных задач этого типа. Как уже отмечалось в предыдущей главе (глава 1) одной из важных прикладных проблем, кроме необходимости решения трёхмерных интегральных уравнений вида (1.16), является выделение из экспериментально измеренного магнитного поля с индукцией $\boldsymbol{B}$ его составляющей $\boldsymbol{B}_{field}$ (см. формулу (1.1)), которая индуцируется непосредственно магнитными массами, распределение которых в пространстве необходимо восстановить.

Поэтому структура этой главы построена следующим образом: сначала рассматриваются наиболее простые в плане предобработки экспериментальных данных прикладные задачи, а далее их сложность в соответствующем смысле возрастает.

Так, в параграфе 2.1 рассматривается тип задач, в которых объект является «движымым», в результате чего вспомогательная задача о выделении компоненты $\boldsymbol{B}_{field}$ из измеренной индукции $\boldsymbol{B}$ общего поля решается с точностью до ошибок экспериментальных измерений компонент вектора магнитной индукции $\boldsymbol{B}$. В параграфе 2.2 рассматриваются задачи геологоразведки в недрах Земли. Для таких задач применимы отработанные в геофизике методы выделения из экспериментальных данных необходимого для расчётов поля $\boldsymbol{B}_{field}$. Однако эти методы являются неточными, в результате чего в выделенную компоненту $\boldsymbol{B}_{field}$ вносится дополнительная систематическая, пусть и небольшая, ошибка. В параграфе 2.3 рассматриваются проблемы геологоразведки в недрах планет Солнечной системы по данным спутниковых наблюдений. Часть таких задач (см. подпараграф 2.3.1) предполагает достаточно точное определение компоненты $\boldsymbol{B}_{field}$ и являются лишь вычислительно сложными, а для некоторых из таких задач (см. подпараграф 2.3.2) для выделения необходимой для расчётов компоненты $\boldsymbol{B}_{field}$ может потребоваться решить вспомогательную обратную задачу бóльшей численной размерности по сравнению с численной размерность «основной» задачи, что косвенным образом приводит к дополнительным ошибкам в выделенной компоненте $\boldsymbol{B}_{field}$.

2.1 Обратная задача восстановления параметров намагниченности локализованного объекта

Как уже было упомянуто в предисловии этой главы, в этом параграфе рассматривается тип прикладных обратных задач магнитометрии, в которых объект является «движымым», в результате чего вспомогательная задача определения $\boldsymbol{B}_{field}$ решается с точностью до ошибок экспериментальных измерений компонент вектор-функции $\boldsymbol{B}$.

Типовая схема выделения необходимого поля $\boldsymbol{B}_{field}$ следующая. В отсутствие исследуемого тела проводятся измерения индукции магнитного поля $\boldsymbol{B}$, которое совпадает с нормальным полем $\boldsymbol{B}^0$. Затем, не меняя пространственную конфигурацию сенсоров, проводятся измерения индукции магнитного поля $\boldsymbol{B}$ в присутствии объекта (см. рис. 2.1). Искомая индукция $\boldsymbol{B}_{field}$ находится как разница этих двух проведённых экспериментальных измерений: $\boldsymbol{B}_{field} = \boldsymbol{B} - \boldsymbol{B}^0$. При этом погрешность определения компонент вектор-функции $\boldsymbol{B}_{field}$ совпадает с погрешностью измерения компонент вектор-функции $\boldsymbol{B}$.

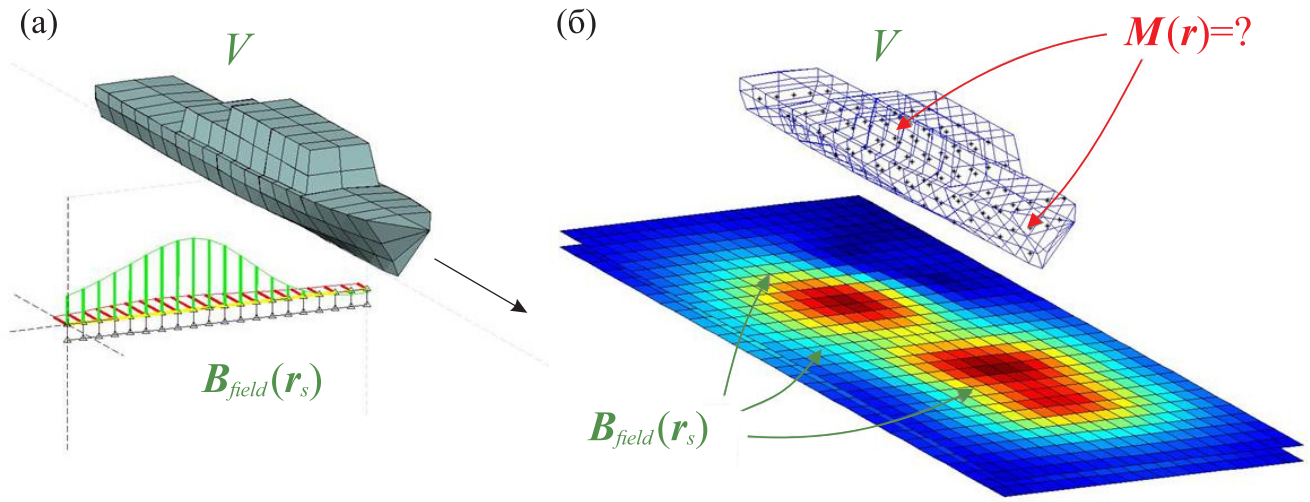


Рисунок 2.1 — Схема измерения индукции магнитного поля «движимого» тела: объект проходит над двумерным массивом сенсоров — картинка (а), что эквивалентно статичным измерениям, выполненным трёхмерным массивом сенсоров — картинка (б).

Таким образом, прикладная задача измерения значений компонент вектор-функции $\boldsymbol{B}_{field}$, которые необходимых для восстановления параметров намагниченности $\boldsymbol{M}$ не вызывает затруднений.

Необходимо отметить, что такого типа задачи чаще всего возникают, когда надо определить параметры остаточной намагниченности некоторого объекта с целью его последующего размагничивания. Классической задачей такого типа является задача размагничивания кораблей (см. рис. 2.1). В СССР в самом начале Великой Отечественной Войны проблему размагничивания кораблей для защиты от магнитных морских мин решали И. В. Курчатов вместе с А. П. Александровым (см. ссылки в книге Б. А. Ткаченко [116]). Сначала соответствующие задачи носили чисто экспериментальный характер и измерения параметров намагниченности проводились вручную с непосредственным контактом с объектом.

Позже, для уменьшения трудозатрат по определению параметров намагниченности была сформулирована соответствующая обратная задача (см. подпараграф 1.1.1 главы 1): **обратная задача** состоит в определении вектор-функции $\boldsymbol{M}(\boldsymbol{r}) \equiv \big(M_x(\boldsymbol{r})\ M_y(\boldsymbol{r})\ M_z(\boldsymbol{r})\big)^T$, $\boldsymbol{r} \equiv (x,y,z) \in V$, из уравнения

$$\boldsymbol{B}_{field}(\boldsymbol{r}_s) = \frac{\mu_0}{4\pi} \iiint\limits_V \left(\frac{3\big(\boldsymbol{M}(\boldsymbol{r}), \boldsymbol{r}-\boldsymbol{r}_s\big)(\boldsymbol{r}-\boldsymbol{r}_s)}{|\boldsymbol{r}-\boldsymbol{r}_s|^5} - \frac{\boldsymbol{M}(\boldsymbol{r})}{|\boldsymbol{r}-\boldsymbol{r}_s|^3} \right) dv$$

по данным экспериментальных измерений компонент вектор-функции $\boldsymbol{B}_{field}(\boldsymbol{r}_s) \equiv \big(B_x(\boldsymbol{r}_s)\ B_y(\boldsymbol{r}_s)\ B_z(\boldsymbol{r}_s)\big)^T$, $\boldsymbol{r}_s \equiv (x_s,y_s,z_s) \notin V$.

Как было показано в главе 1 это уравнение сводится к интегральному уравнению Фредгольма 1-го рода в виде (1.16):

$$\mathbf{A}\boldsymbol{M} \equiv \frac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}(x,y,z,x_s,y_s,z_s)\, \boldsymbol{M}(x,y,z)\, dxdydz = \boldsymbol{B}(x_s,y_s,z_s). \qquad (2.1)$$

Здесь $\boldsymbol{M} \equiv \big(M_x\ M_y\ M_z\big)^T$, $\boldsymbol{B} \equiv \boldsymbol{B}_{field} \equiv \big(B_x\ B_y\ B_z\big)^T$, $\mathbf{K} \equiv \mathbf{K}_{MI}$ (выражение для этой матричной функции выписано в подпараграфе 1.1.1 главы 1).

Замечание. Несмотря на то, что в обратных задачах такого типа бóльшая часть тела содержит существенное количество ферромагнитных материалов, здесь решается вопрос об определении параметров остаточной намагниченности в относительно слабом внешнем поле. В связи с этим эта постановка обратной задачи остаётся применимой, если целью является определение в объекте намагниченных областей. Соответствующему вопросу посвящено достаточно большое число работ, из которых можно выделить работы Ф. Дютуа [117], М. Гуамьери [118], Х. Брюнотта [119], Ф. Риу-Дамидо [120], Г. Олунда [121], Ш. Лью [122].

Важная практическая особенность этого типа задач заключается в том, что геометрические параметры исследуемого объекта обычно известны.

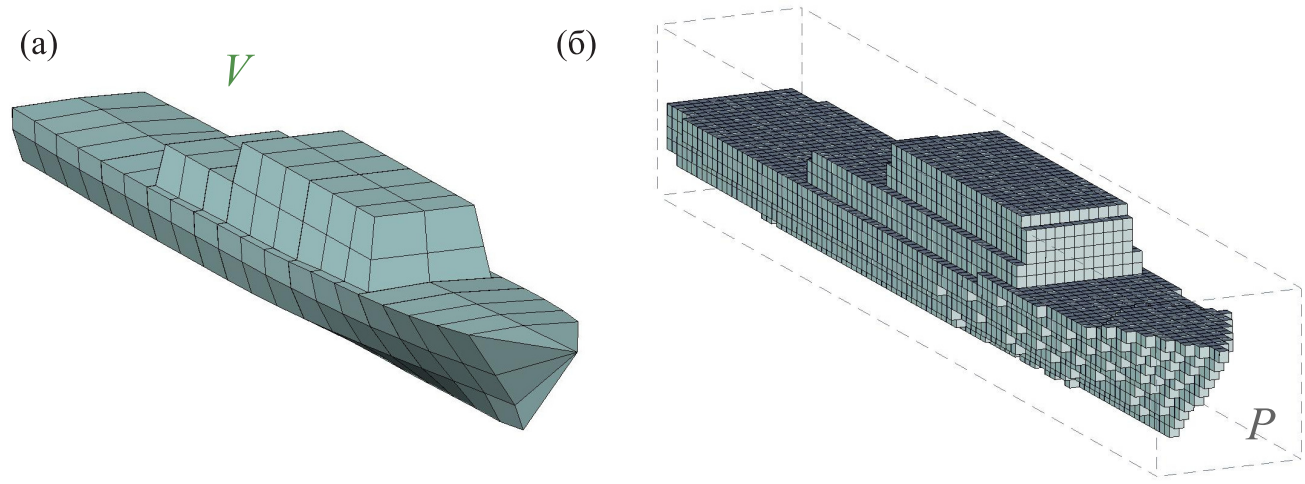


Рисунок 2.2 — Если известны геометрические параметры тела — картинка (а), то известны и позиции расположения магнитных масс, которые достаточно легко дискретизировать — картинка (б).

В результате можно предполагать, что исследуемый объект лежит в параллелепипеде P с известным расположением в пространстве и известными геометрическими параметрами (см. рис. 2.2-б):

$$V \subset P = \big\{(x,y,z):\ L_x \leqslant x \leqslant R_x, L_y \leqslant y \leqslant R_y, L_z \leqslant z \leqslant R_z\big\}.$$

Массив точек, в которых проводятся измерения сенсорами, для таких задач также обычно образует параллелепипед в трёхмерном пространстве (см. рис. 2.1-б):

$$Q = \big\{(x_s,y_s,z_s):\ L_{x_s} \leqslant x_s \leqslant R_{x_s}, L_{y_s} \leqslant y_s \leqslant R_{y_s}, L_{z_s} \leqslant z_s \leqslant R_{z_s}\big\}.$$

В результате уравнение (2.1) сводится к виду

$$\mathbf{A}\boldsymbol{M} \equiv \frac{\mu_0}{4\pi} \int\limits_{L_x}^{R_x}\int\limits_{L_y}^{R_y}\int\limits_{L_z}^{R_z} \mathbf{K}(x,y,z,x_s,y_s,z_s)\boldsymbol{M}(x,y,z)dxdydz = \boldsymbol{B}(x_s,y_s,z_s). \quad (2.2)$$

В задачах такого типа предполагается, что $\boldsymbol{M} \in W_2^2(P)$, $\boldsymbol{B} \in L_2(Q)$, и оператор $\mathbf{A}$ с ядром $\mathbf{K}$ является непрерывным и однозначным. Нормы правой части уравнения (2.2) и его решения можно ввести следующим образом:

$$\|\boldsymbol{B}\|_{L_2} = \sqrt{\|B_x\|_{L_2}^2 + \|B_y\|_{L_2}^2 + \|B_z\|_{L_2}^2},$$
$$\|\boldsymbol{M}\|_{W_2^2} = \sqrt{\|M_x\|_{W_2^2}^2 + \|M_y\|_{W_2^2}^2 + \|M_z\|_{W_2^2}^2}.$$

Пусть вместо точно известных $\boldsymbol{B}$ и оператора $\mathbf{A}$ известны их приближённые значения $\boldsymbol{B}_\delta$ и $\mathbf{A}_h$ такие, что $\|\boldsymbol{B}_\delta - \boldsymbol{B}\|_{L_2} \leqslant \delta$, $\|\mathbf{A} - \mathbf{A}_h\|_{W_2^2 \to L_2} \leqslant h$. При выписанных условиях задача определения вектор-функции $\boldsymbol{M}$ из уравнения (2.2) является некорректно поставленной и для её решения необходимо строить регуляризирующий алгоритм. Если воспользоваться регуляризирующим алгоритмом, основанным на минимизации функционала А. Н. Тихонова, то задача сводится к поиску элемента, реализующего минимум функционала

$$F^\alpha[\boldsymbol{M}] = \|\mathbf{A}_h\boldsymbol{M} - \boldsymbol{B}_\delta\|^2_{L_2} + \alpha\|\boldsymbol{M}\|^2_{W_2^2}.$$

Согласно работе А. Н. Тихонова, А. В. Гочарского, В. В. Степанова и А. Г. Яголы [91] для любого $\alpha > 0$ существует единственная экстремаль $\boldsymbol{M}^\alpha_\eta$, $\eta = \{\delta, h\}$, которая реализует минимум функционала $F^\alpha[\boldsymbol{M}]$.

Параметр регуляризации можно выбрать, например, по обобщённому принципу невязки (см. работу [91]), как корень нелинейного уравнения

$$\rho(\alpha) \equiv \left\|\mathbf{A}_h\boldsymbol{M}^\alpha_\eta - \boldsymbol{B}_\delta\right\|^2_{L_2} - \left(\delta + h\left\|\boldsymbol{M}^\alpha_\eta\right\|_{W_2^2}\right)^2 - \mu^2_\eta = 0.$$

Здесь μ_η — мера несовместности, которая определяется следующим образом:

$$\mu_\eta = \inf_{\boldsymbol{M}} \left\|\mathbf{A}_h\boldsymbol{M} - \boldsymbol{B}_\delta\right\|_{L_2}.$$

При этом $\boldsymbol{M}^\alpha_\eta$ стремиться при $\eta \to 0$ к точному решению задачу в норме W_2^2, а следовательно, и равномерно на множестве P.

В качестве метода минимизации функционала А. Н. Тихонова применяется метод сопряжённых градиентов, особенности практической реализации которого для рассматриваемого класса задач описываются в главе 3 «Численные методы» и в главе 4 «Комплекс программ».

Замечание. В случае решения уравнения вида (2.2) сглаживающий функционал $\|\boldsymbol{M}\|^2_{W_2^2}$ примет вид

$$\|\boldsymbol{M}\|^2_{W_2^2} = \int_{L_x}^{R_x}\int_{L_y}^{R_y}\int_{L_z}^{R_z}\left(\boldsymbol{M}^2 + \left(\frac{\partial^2\boldsymbol{M}}{\partial x^2}\right)^2 + \left(\frac{\partial^2\boldsymbol{M}}{\partial y^2}\right)^2 + \left(\frac{\partial^2\boldsymbol{M}}{\partial z^2}\right)^2\right)dxdydz.$$

При этом надо отметить следующие особенности решения рассматриваемого класса задач. Если известна «примерная» геометрия объекта и используется восстановление распределения магнитных масс в параллелепипеде P, то в этом

случае существенным образом можно использовать особенность ядра интегрального уравнения (2.2). Эта особенность заключается в том, что уравнение (2.2) можно классифицировать как интегральное уравнение типа свёртки (см. работу [91, с. 38]), и за счёт того, что решение ищется в многомерной прямоугольной области, можно использовать подход основанный на применении быстрого преобразования Фурье к этому уравнению (см. работу [6]: «“Быстрый” алгоритм решения некоторых трёхмерных обратных задач магнитометрии»).

В том случае, если геометрия исследуемого тела известна точно (см. рис. 2.2), то вместо задачи решения уравнения (2.2) может быть поставлена более простая в смысле численной трудоёмкости задача. Например, на рис. 2.2 изображено разбиение параллелепипеда P на $\widetilde{N} = 100 \times 15 \times 15 = 22\,500$ «элементарных» параллелепипедов объёмом $dx_i dy_i dz_i = \texttt{const}$, $i = \overline{1,\,22\,500}$, каждый. Если пронумеровать только те элементарные разбиения области P, для которых $(x_i, y_i, z_i) \in V$, то окажется, что число таких элементов $N \ll \widetilde{N} = 22\,500$.

Если перенумеровать эти элементы от 1 до N, то можно выписать следующую систему из S уравнений:

$$\frac{\mu_0}{4\pi} \sum_{i=1}^{N} \mathbf{K}(x_i, y_i, z_i, x_{sj}, y_{sj}, z_{sj}) \boldsymbol{M}(x_i,y_i,z_i) dx_i dy_i dz_i = \boldsymbol{B}(x_{sj}, y_{sj}, z_{sj}), \quad j = \overline{1,S}.$$

Здесь S — число сенсоров, измеряющих компоненты вектор-функции $\boldsymbol{B} \equiv \boldsymbol{B}_{field}$, с известными координатами (x_{sj}, y_{sj}, z_{sj}), $j = \overline{1,S}$.

При $\max\limits_i dx_i dy_i dz_i \to 0$ каждое уравнение этой системы будет эквивалентно уравнению (2.1).

Математические методы моделирования прикладных трёхмерных обратных задач магнитометрии, рассматриваемые в следующих параграфах, будут обобщать описанные здесь подходы. В связи с этим примеры результатов моделирования решений для этого типа задач здесь не приводятся, но могут быть найдены в работах автора [7; 8].

Главная прикладная проблема, которая возникает при решении таких систем уравнений — необходимость решения огромных систем линейных алгебраических уравнений с плотно заполненной матрицей. Для этого нужно использовать многопроцессорные системы и применять специальные методы решения систем линейных алгебраических уравнений. Соответствующие вопросы будут рассмотрены в главах 3 и 4.

2.2 Обратная задача восстановления параметров намагниченности полезных ископаемых

В этом параграфе рассматривается тип прикладных обратных задач магнитометрии, относящихся к задачам геологоразведки. Принципиальная особенность этих задач состоит в наличии следующих проблем. Во-первых, объект является «недвижымым», в результате чего вспомогательная задача определения $\boldsymbol{B}_{field}$ усложняется. Во-вторых, точное расположение магнитных масс неизвестно.

Первая задача решается следующим образом. Из экспериментально измеренного магнитного поля с индукцией $\boldsymbol{B}$ надо выделить необходимую для последующих расчётов составляющую $\boldsymbol{B}_{field}$, которая индуцируется магнитными массами, из которых состоят, например, полезные ископаемые. В геофизике такую составляющую магнитного поля принято называть *аномалией земного магнитного поля*. Чтобы её найти необходимо знать индукцию нормального магнитного поля $\boldsymbol{B}^0$, которое в геофизике также часто называют *регулярным геомагнитным полем*. В настоящее время считается, что магнитное поле Земли порождается токами в жидком ядре Земли. С весьма хорошим приближением регулярное геомагнитное поле можно представить как поле диполя, расположенного в центре Земли и имеющего магнитный момент, направленный в сторону Южного географического полюса, но при этом смещённого от оси вращения Земли на угол около 11°. Вблизи поверхности Земли напряжённость регулярного магнитного поля составляет примерно 40 $\mathrm{A} \cdot \mathrm{m}^{-1}$. Модель регулярного геомагнитного поля как поля диполя даёт возможность с приемлемой точностью рассчитать компоненты вектора $\boldsymbol{B}^0$ в точках измерения, расположенных недалеко от поверхности Земли (см., например, работу У. О. Кэмпбелаа [123]). Таким образом, указанная особенность не представляет принципиальных трудностей для решения реальных прикладных задач. Следует лишь отметить, что так как модель геомагнитного поля Земли как модель диполя является лишь приближённой, то это вносит дополнительные систематические ошибки в результаты определения компонент вектор-функции $\boldsymbol{B}_{field}$.

Вторая особенность и пути преодоления связанных с нею проблем будут рассмотрены далее в процессе изложения математических подходов к решению задач указанного типа.

2.2.1 Восстановление намагниченности

При решении указанного типа задач (см. работы автора [2; 10]) использовалась модель, описанная в подпараграфе 1.1.4 «Обращение полных магнито-градиентных данных» главы 1. Это означает, что **обратная задача** состоит в определении вектор-функции $\boldsymbol{M}(\boldsymbol{r}) \equiv \left(M_x(\boldsymbol{r})\ M_y(\boldsymbol{r})\ M_z(\boldsymbol{r})\right)^T$, $\boldsymbol{r} \equiv (x,y,z) \in V$, из системы уравнений

$$\begin{cases} \boldsymbol{B}_{field}(x_s,y_s,z_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}_{MI}(x,y,z,x_s,y_s,z_s)\boldsymbol{M}(x,y,z)dv, \\ \boldsymbol{B}_{tensor}(x_s,y_s,z_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}_{MGT}(x,y,z,x_s,y_s,z_s)\boldsymbol{M}(x,y,z)dv, \end{cases}$$

по данным экспериментальных измерений компонент вектор-функции $\boldsymbol{B}_{field}(\boldsymbol{r}_s) \equiv \left(B_x(\boldsymbol{r}_s)\ B_y(\boldsymbol{r}_s)\ B_z(\boldsymbol{r}_s)\right)^T$, $\boldsymbol{r}_s \equiv (x_s,y_s,z_s) \notin V$, и независимых компонент тензора градиентов компонент магнитной индукции $\boldsymbol{B}_{tensor}(\boldsymbol{r}_s) \equiv \left(\dfrac{\partial B_x}{\partial x}(\boldsymbol{r}_s)\ \dfrac{\partial B_x}{\partial y}(\boldsymbol{r}_s)\ \dfrac{\partial B_x}{\partial z}(\boldsymbol{r}_s)\ \dfrac{\partial B_y}{\partial z}(\boldsymbol{r}_s)\ \dfrac{\partial B_z}{\partial z}(\boldsymbol{r}_s)\right)^T$, $\boldsymbol{r}_s \notin V$.

Выражения для матричных функций $\mathbf{K}_{MI}$ и $\mathbf{K}_{MGT}$ выписаны в подпараграфах 1.1.1 и 1.1.3 главы 1 соответственно.

Как было показано в главе 1, выписанная система сводится к интегральному уравнению Фредгольма 1-го рода в виде (1.16):

$$\mathbf{A}\boldsymbol{M} \equiv \frac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}(x,y,z,x_s,y_s,z_s)\,\boldsymbol{M}(x,y,z)\,dxdydz = \boldsymbol{B}(x_s,y_s,z_s). \qquad (2.3)$$

Здесь $\boldsymbol{M} \equiv \left(M_x\ M_y\ M_z\right)^T$, $\boldsymbol{B} \equiv \left(B_x\ B_y\ B_z\ \frac{\partial B_x}{\partial x}\ \frac{\partial B_x}{\partial y}\ \frac{\partial B_x}{\partial z}\ \frac{\partial B_y}{\partial z}\ \frac{\partial B_z}{\partial z}\right)^T$, $\mathbf{K} \equiv \left(\mathbf{K}_{MI}\ \mathbf{K}_{MGT}\right)^T$.

Если предположить, что исследуемый объект лежит в параллелепипеде с известным расположением в пространстве и известными геометрическими параметрами

$$V \subset P = \left\{(x,y,z):\ L_x \leqslant x \leqslant R_x, L_y \leqslant y \leqslant R_y, L_z \leqslant z \leqslant R_z\right\},$$

а массив точек, в которых проводятся измерения сенсорами, содержится в некоторой произвольной области Q, то уравнение (2.3) сводится к виду

$$\mathbf{A}\boldsymbol{M} \equiv \frac{\mu_0}{4\pi} \int\limits_{L_x}^{R_x}\int\limits_{L_y}^{R_y}\int\limits_{L_z}^{R_z} \mathbf{K}(x,y,z,x_s,y_s,z_s)\boldsymbol{M}(x,y,z)dxdydz = \boldsymbol{B}(x_s,y_s,z_s). \qquad (2.4)$$

Замечание. Принципиально отличие этой математической постановки обратной задачи от постановки рассмотренной в предыдущем параграфе (параграф 2.1) кроме использования «тензорных» данных заключается в следующем. Во-первых, обычно отсутствует *априорная* информация об оптимальных геометрических параметрах параллелепипеда P (так как локализация магнитных масс не известна). Поэтому приходится изначально выбирать заведомо большие геометрические размеры области, в которой ищется решение, а затем по найденному решению сужать область восстановления решения с целью более качественного восстановления параметров намагниченности исследуемого тела. Во-вторых, в которых проводятся измерения, уже не принадлежат некоторому параллелепипеду, так как на практике зачастую точки измерения образуют в трёхмерном пространстве достаточно сложную кривую (см. рис. 2.4). А это исключает возможность применения «быстрых» алгоритмов решения таких задач, основанных на быстром преобразовании Фурье (см. работу автора [6]).

В задачах такого типа предполагается, что $\boldsymbol{M} \in W_2^2(P)$, $\boldsymbol{B} \in L_2(Q)$, и оператор $\mathbf{A}$ с ядром $\mathbf{K}$ является непрерывным и однозначным. Нормы правой части уравнения (2.4) и его решения можно ввести следующим образом:

$$\|\boldsymbol{B}\|_{L_2} = \Bigg(\|B_x\|_{L_2}^2 + \|B_y\|_{L_2}^2 + \|B_z\|_{L_2}^2 + \\ + \left\|\frac{\partial B_x}{\partial x}\right\|_{L_2}^2 + \left\|\frac{\partial B_x}{\partial y}\right\|_{L_2}^2 + \left\|\frac{\partial B_x}{\partial z}\right\|_{L_2}^2 + \left\|\frac{\partial B_x y}{\partial z}\right\|_{L_2}^2 + \left\|\frac{\partial B_z}{\partial z}\right\|_{L_2}^2\Bigg)^{\frac{1}{2}},$$

$$\|\boldsymbol{M}\|_{W_2^2} = \sqrt{\|M_x\|_{W_2^2}^2 + \|M_y\|_{W_2^2}^2 + \|M_z\|_{W_2^2}^2}.$$

Пусть вместо точно известных $\boldsymbol{B}$ и оператора $\mathbf{A}$ известны их приближённые значения $\boldsymbol{B}_\delta$ и $\mathbf{A}_h$ такие, что $\|\boldsymbol{B}_\delta - \boldsymbol{B}\|_{L_2} \leqslant \delta$, $\|\mathbf{A} - \mathbf{A}_h\|_{W_2^2 \to L_2} \leqslant h$. При выписанных условиях задача является некорректно поставленной и для её решения необходимо строить регуляризирующий алгоритм. Если воспользоваться регуляризирующим алгоритмом, основанным на минимизации функционала А. Н. Тихонова, то задача сводится к поиску элемента, реализующего минимум функционала

$$F^\alpha[\boldsymbol{M}] = \|\mathbf{A}_h\boldsymbol{M} - \boldsymbol{B}_\delta\|_{L_2}^2 + \alpha\|\boldsymbol{M}\|_{W_2^2}^2.$$

Согласно работе А. Н. Тихонова, А. В. Гочарского, В. В. Степанова и А. Г. Яголы [91] для любого $\alpha > 0$ существует единственная экстремаль $\boldsymbol{M}_\eta^\alpha$, $\eta = \{\delta, h\}$, которая реализует минимум функционала $F^\alpha[\boldsymbol{M}]$.

Параметр регуляризации можно выбрать, например, по обобщённому принципу невязки (см. работу [91]), как корень нелинейного уравнения

$$\rho(\alpha) \equiv \left\| \mathbf{A}_h \boldsymbol{M}_\eta^\alpha - \boldsymbol{B}_\delta \right\|_{L_2}^2 - \left(\delta + h \left\| \boldsymbol{M}_\eta^\alpha \right\|_{W_2^2} \right)^2 - \mu_\eta^2 = 0.$$

Здесь μ_η — мера несовместности, которая определяется как

$$\mu_\eta = \inf_{\boldsymbol{M}} \left\| \mathbf{A}_h \boldsymbol{M} - \boldsymbol{B}_\delta \right\|_{L_2}.$$

При этом $\boldsymbol{M}_\eta^\alpha$ стремиться при $\eta \to 0$ к точному решению задачу в норме W_2^2, а следовательно, и равномерно на множестве P.

В качестве метода минимизации функционала А. Н. Тихонова применяется метод сопряжённых градиентов, особенности практической реализации которого для рассматриваемого класса задач описываются в главе 3 «Численные методы» и в главе 4 «Комплекс программ».

Численный эксперимент. Сначала следует рассмотреть различные модели, которые используют только «магнитные», только «градиентные» и совместные «магнито-градиентные» данные экспериментальных измерений (см. подпараграф 1.1.4 главы 1). Для этого было выполнено моделирование экспериментальных данных с уровнем ошибки δ, эквивалентном относительной ошибке $\sim 4\%$. Для тестовых расчётов использовалась область $P = \{(x,y,z) : -5\,000 \leqslant x \leqslant 5\,000,\ -5\,000 \leqslant y \leqslant 5\,000,\ -105 \leqslant z \leqslant -95\}$. Параллелепипед P был разбит по каждому направлению на $(N_x,N_y,N_z) = (80,80,1)$ частей. Таким образом задача заключалась в восстановлении усреднённой плотности магнитного момента каждого элементарного объёма dv, полученного за счёт разбиения исходной области P на части. Область наблюдения была выбрана как $Q = \{(x_s,y_s,z_s) :\ -4\,000 \leqslant x_s \leqslant 4\,000, -4\,000 \leqslant y_s \leqslant 4\,000, z_s = 2\,000\}$. Эта область была разбита по каждому направлению на $(N_{x_s},N_{y_s},N_{y_s}) = (350,20,1)$ частей. В узлах этого разбиения области на части были симулированы как «магнитные», так и «тензорные» магнитные данные. Затем симулированные данные использовались в качестве входных при решении обратной задачи в разных постановках. На рис. 2.3 представлена нормализованная величина модуля намагниченности, как результат решения обратной задачи с симулированными экспериментальными данными. Среднеквадратичная ошибка в компонентах восстановленного вектора $\boldsymbol{M}$ составила 0.12263 для обработки только «магнитных» данных, 0.12262 для обработки совместных «магнито-градиентных»

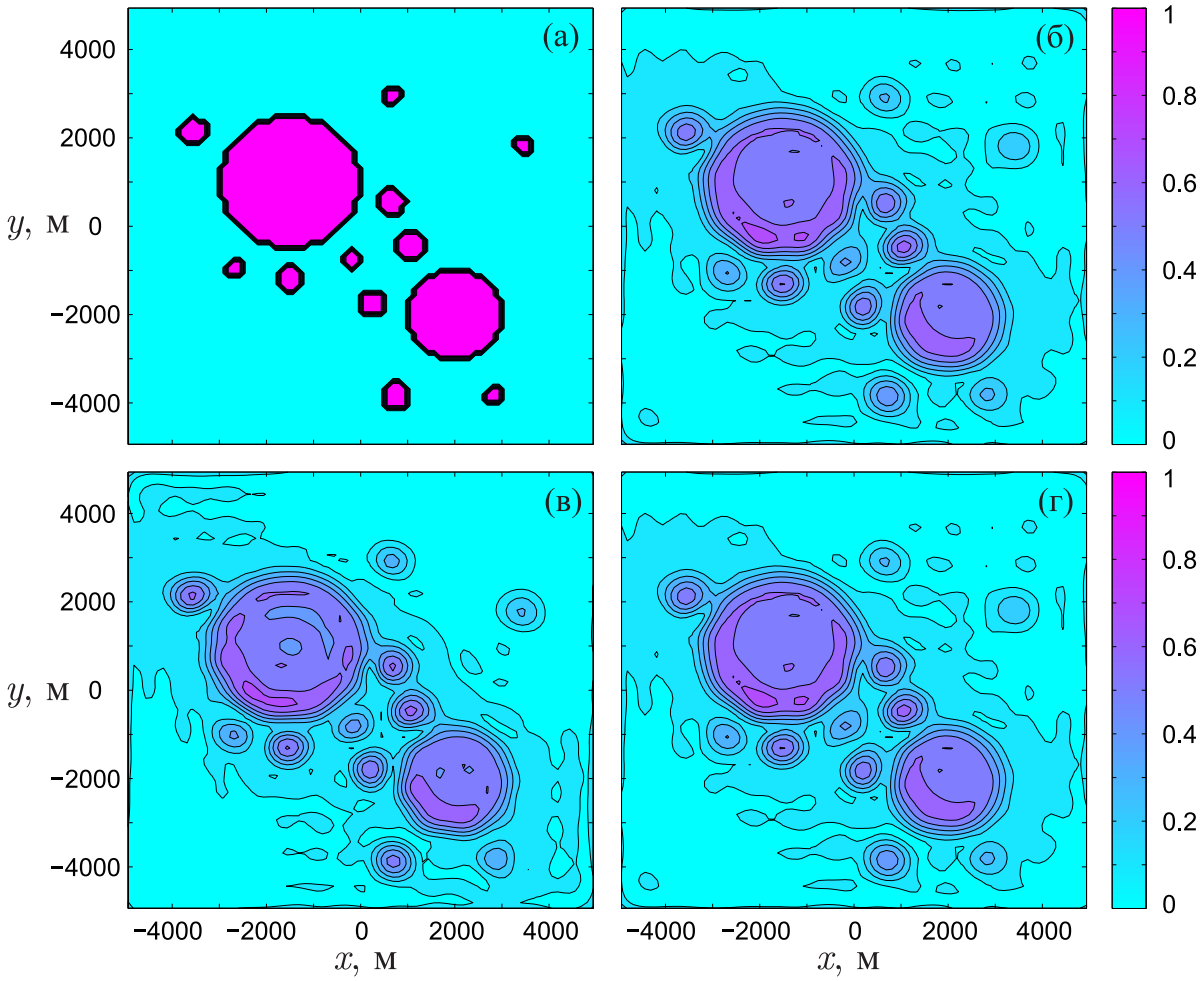


Рисунок 2.3 — Результаты численного эксперимента: (а) нормализованное модельное решение — модуль вектора намагниченности $\boldsymbol{M}$, (б) восстановленное решение при обработке только «магнитных» данных, (в) восстановленное решение при обработке только «тензорных» данных, (г) восстановленное решение при обработке совместных «магнито-градиентных» данных.

данных и 0.12527 для обработки только «градиентных» данных. Это означает, что все модели дают «равные» результаты. При уменьшении уровня ошибки δ в симулированных данных до эквивалентного уровня относительной ошибки $\sim 1\%$, которые достаточно близки к реальным полевым экспериментам, среднеквадратичная ошибка в компонентах восстановленного вектора $\boldsymbol{M}$ оказалась наименьшей для случая обработки только «тензорных» данных.

Таким образом основной вывод из тестовых численных экспериментов следующий. Обработка только «тензорных» данных позволяет восстановить более детализированное приближённое решение. Обработка совместных «магнито-градиентных» данных не даёт никаких преимуществ в детализации решения. При обработке экспериментальных данных «тензорные» данные должны ис-

пользоваться обособленно и не объединяться с «магнитными» данными. Это связано с тем, что величины, соответствующие «тензорным» данным, существенно меньше величин, соответствующих «магнитным» данным. В результате за счёт наличия ошибок машинного округления применяемые численные методы становятся нечувствительными к «тензорным» данным, если они используются вместе с «магнитными» данными. Если же «тензорные» данные используются отдельно от магнитных, то точность восстановления приближённого регуляризованного решения возрастает за счёт того, что «тензорных» данных больше, чем «магнитных» (5 компонент тензора градиентов компонент магнитной индукции против 3 компонент магнитной индукции).

Замечание. Повысить эффект от использования совместных «магнитоградиентных» данных можно за счёт масштабирования «градиентных» данных до уровня «магнитных». Такой подход эквивалентен выбору предобуславливателя, который понизит обусловленность системы линейных алгебраических уравнений, которая возникает при численном решении после дискретизации интегрального уравнения (2.4).

Пример обработки реальных экспериментальных данных Натурные экспериментальные измерения компонент тензора градиентов компонент магнитной индукции проводились с использованием низкотемпературной системы SQUID (см. рис. 2.4) в одном из районов Северного Китая, состоящем из парамагнитных и ферромагнитных материалов (см. работу автора [9]).

В качестве области, в которой восстанавливалось решение, использовалась область $P = \{(x,y,z) : \ -1\,000 \leqslant x \leqslant 1\,000, \ -1\,000 \leqslant y \leqslant 1\,000, \ -205 \leqslant z \leqslant -195\}$ m. Параллелепипед P был разбит по каждому направлению на $(N_x,N_y,N_z) = (40,40,1)$ частей. Таким образом задача заключалась в восстановлении усреднённой плотности магнитного момента каждого элементарного объёма dv, полученного за счёт разбиения исходной области P на части. Область наблюдения была выбрана как $Q = \{(x_s,y_s,z_s) : \ -215 \leqslant x_s \leqslant 215, -250 \leqslant y_s \leqslant 250, \ z_s = 20\}$ m. Эта область была разбита по каждому направлению на $(N_{x_s},N_{y_s},N_{y_s}) = (43,50,1)$ частей. Такие параметры разбиения области наблюдения соответствуют реальной размерности массивов экспериментальных данных. Здесь вертолёт пролетел над одной и той же область 43 раза с равномерным смещением траектории каждого пролёта относительно предыдущих. При этом

Рисунок 2.4 — Проведение натурных экспериментальных измерений (см. работу [9]) компонент тензора градиентов компонент магнитной индукции на разных высотах над поверхностью Земли с помощью сенсора SQUID. Траектория движения сенсора может представлять достаточно произвольную кривую.

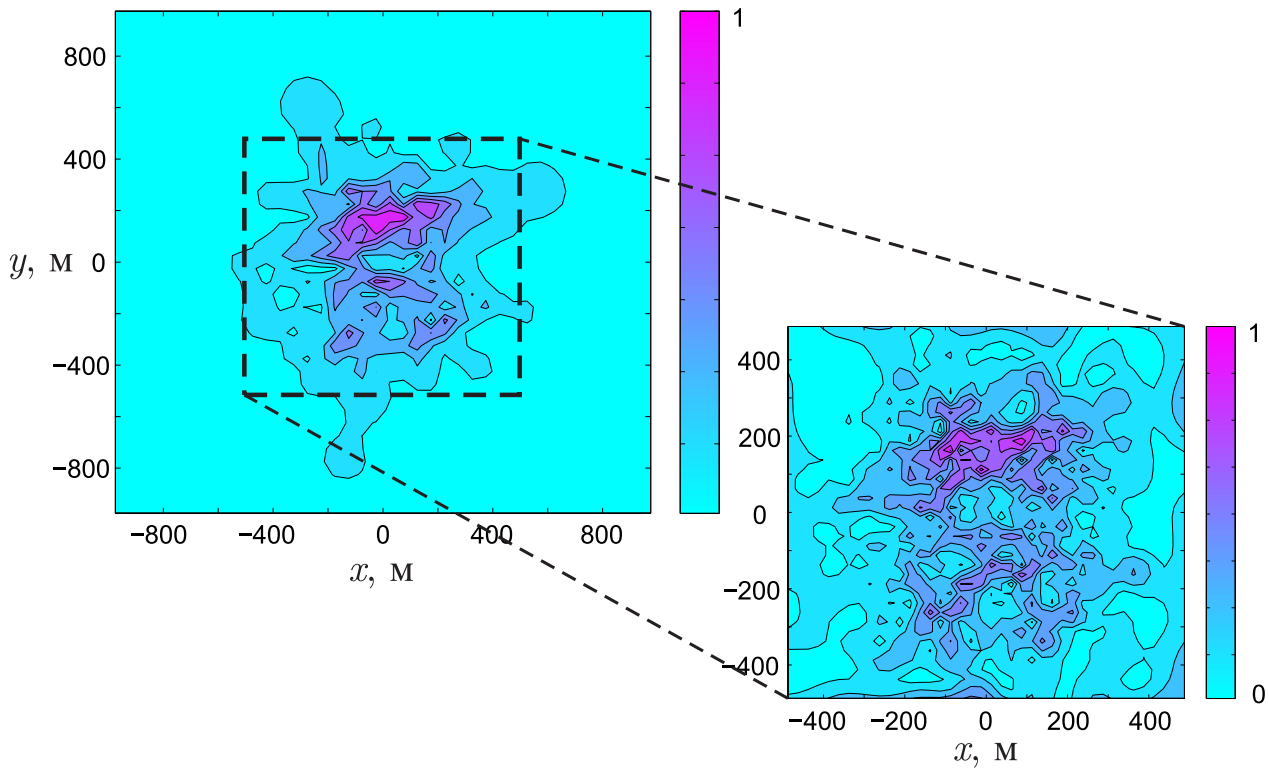


Рисунок 2.5 — Результат обработки реальных экспериментальных данных. На основной картинке изображён результат предварительного определения области локализации магнитных масс (изображено нормализованное значение модуля вектора намагниченности $\boldsymbol{M}$). Картинка-врезка соответствует результату повторной обработки экспериментальных данных после уточнения размеров области P, в которой восстанавливается эквивалентное по внешнему полю распределение магнитных масс.

вдоль каждой траектории было выбрано 50 точек, которые все вместе образовывали прямоугольную декартову сетку. Так как экспериментальные данные заданы в двумерном пространстве (на плоскости), то и восстановление искомой вектор-функции осуществлялась на плоскости на глубине с координатой $z = -200$ m. То есть необходимо восстанавливалось эквивалентное по внешнему полю распределение магнитных масс на заданной глубине.

Выбранные параметры области P позволили получить после проведения расчётов предварительную информацию о месторасположении магнитных масс (см. рис. 2.5). При этом надо отметить, что предварительные расчёты проводились без регуляризации ($\alpha = 0$). Затем, основываясь на полученной информации, размеры области P были уточнены: $P = \{(x,y,z) : \ -500 \leqslant x \leqslant 500, \ -500 \leqslant y \leqslant 500, \ -205 \leqslant z \leqslant -195\}$, и были проведены повторные рас-

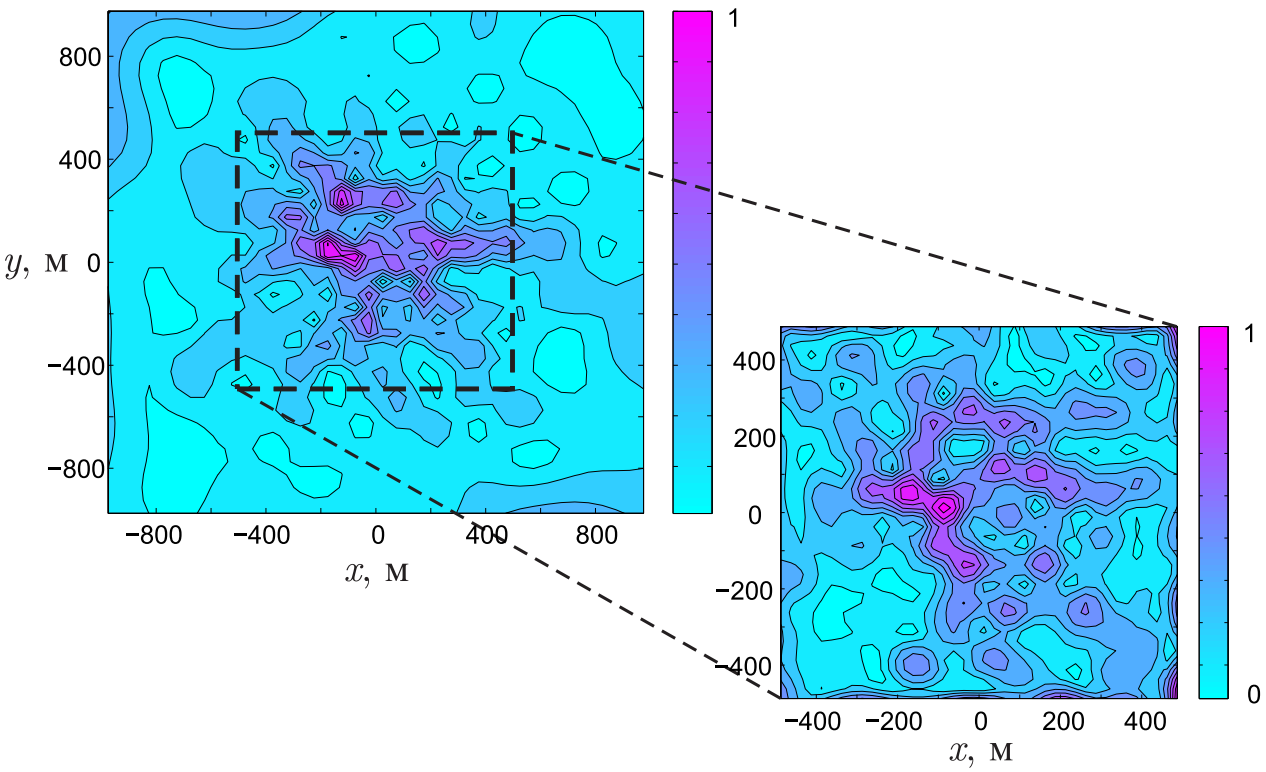


Рисунок 2.6 — Результат обработки реальных экспериментальных данных на другой глубине по сравнению. На основной картинке изображён результат предварительного определения области локализации магнитных масс (изображено нормализованное значение модуля вектора намагниченности $\boldsymbol{M}$). Картинка-врезка соответствует результату повторной обработки экспериментальных данных после уточнения размеров области P, в которой восстанавливается эквивалентное по внешнему полю распределение магнитных масс.

чёты с выбором параметра регуляризации, согласованным с уровнем ошибок экспериментальных данных по обобщённому принципу невязки.

На рис. 2.6 представлены результаты восстановления эквивалентного распределения магнитных масс на глубине $z = -500$ m.

В случае проведения экспериментальных измерений на разных высотах возможно восстановление распределения магнитных масс в трёхмерном пространстве. Однако практическую ценность в этой задаче имеет и восстановление эквивалентного по внешнему полю распределения магнитных масс на заданной фиксированной глубине (подробности могут быть найдены в работе автора [9]). Это связано с тем, что при решении некоторых задач геологоразведки важно определить не точную глубину залегания полезных ископаемых, а те точки на поверхности Земли, в которых следует производить бурение с целью точной доразведки места предполагаемого залегания полезных ископаемых.

2.2.2 Восстановление магнитной восприимчивости

При решении задач указанного типа использовалась модель (см. работу автора [3]), описанная в параграфе 1.2 главы 1. Это означает, что **обратная задача** состоит в определении функции $\chi(\boldsymbol{r})$, $\boldsymbol{r} \equiv (x,y,z) \in V$, из системы уравнений

$$\begin{cases} \boldsymbol{B}_{field}(x_s,y_s,z_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}^{B^0}_{MI}(x,y,z,x_s,y_s,z_s)\,\chi(x,y,z)\,dv, \\ \boldsymbol{B}_{tensor}(x_s,y_s,z_s) = \dfrac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}^{B^0}_{MGT}(x,y,z,x_s,y_s,z_s)\,\chi(x,y,z)\,dv, \end{cases}$$

по данным экспериментальных измерений компонент вектор-функции $\boldsymbol{B}_{field}(\boldsymbol{r}_s) \equiv \left(B_x(\boldsymbol{r}_s) \;\; B_y(\boldsymbol{r}_s) \;\; B_z(\boldsymbol{r}_s)\right)^T$, $\boldsymbol{r}_s \equiv (x_s,y_s,z_s) \notin V$, и независимых компонент тензора градиентов компонент магнитной индукции $\boldsymbol{B}_{tensor}(\boldsymbol{r}_s) \equiv \left(\dfrac{\partial B_x}{\partial x}(\boldsymbol{r}_s) \; \dfrac{\partial B_x}{\partial y}(\boldsymbol{r}_s) \; \dfrac{\partial B_x}{\partial z}(\boldsymbol{r}_s) \; \dfrac{\partial B_y}{\partial z}(\boldsymbol{r}_s) \; \dfrac{\partial B_z}{\partial z}(\boldsymbol{r}_s)\right)^T$, $\boldsymbol{r}_s \notin V$, и наличию информации о нормальном магнитном поле $\boldsymbol{B}^0(\boldsymbol{r})$ в области V.

Выражения для матричных функций $\mathbf{K}^{B^0}_{MI}$ и $\mathbf{K}^{B^0}_{MGT}$ могут быть получены за счёт умножения на $\left|\boldsymbol{B}^0(\boldsymbol{r})\right|$ матричных функций $\mathbf{K}^{l}_{MI}$ и $\mathbf{K}^{l}_{MGT}$, выписанных в подпараграфах 1.1.2 и 1.1.3.

Указанную систему можно записать в виде следующего интегрального уравнения Фредгольма 1-го рода:

$$\mathbf{A}\chi \equiv \frac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}(x,y,z,x_s,y_s,z_s)\,\chi(x,y,z)\,dxdydz = \boldsymbol{B}(x_s,y_s,z_s). \qquad (2.5)$$

Здесь $\boldsymbol{B} \equiv \left(B_x \; B_y \; B_z \; \frac{\partial B_x}{\partial x} \; \frac{\partial B_x}{\partial y} \; \frac{\partial B_x}{\partial z} \; \frac{\partial B_y}{\partial z} \; \frac{\partial B_z}{\partial z}\right)^T$, $\mathbf{K} \equiv \left(\mathbf{K}^{B^0}_{MI} \; \mathbf{K}^{B^0}_{MGT}\right)^T$.

Подход к построению и реализации регуляризирующего алгоритма решения этого уравнения полностью повторяет методику, описанную в предыдущих двух подпараграфах. Поэтому соответствующие пояснения здесь опускаются (подробности см. в работе автора [3]).

Параметры тестового примера, демонстрирующего преимущество использования «тензорных» данных, полностью повторяют параметры численного эксперимента, описанного в предыдущем подпараграфе. Единственное отличие заключается в том, что теперь модельным решением является не векторная

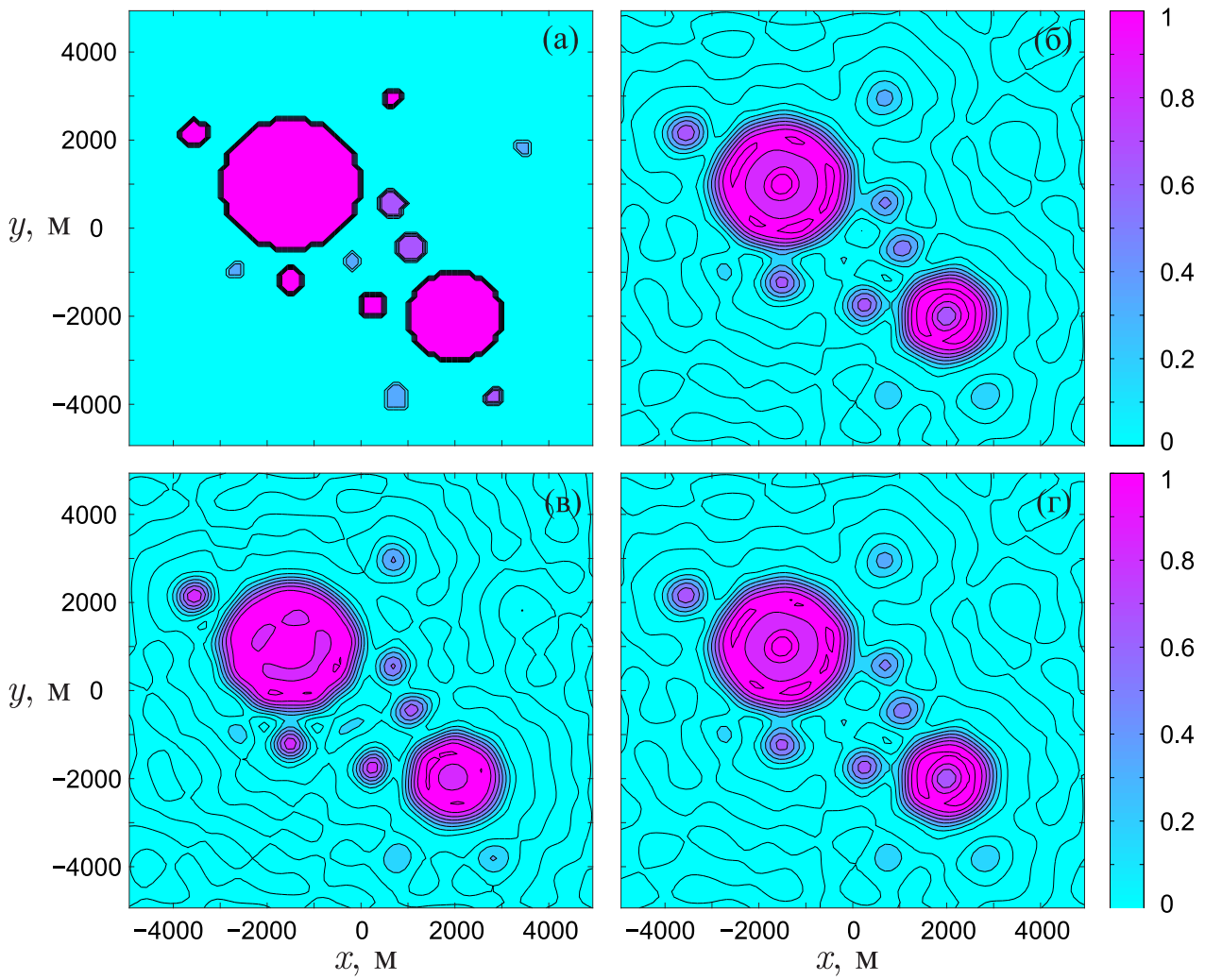


Рисунок 2.7 — Результаты численного эксперимента: (а) нормализованное модельное решение — магнитная восприимчивость χ, (б) восстановленное решение при обработке только «магнитных» данных, (в) восстановленное решение при обработке только «тензорных» данных, (г) восстановленное решение при обработке совместных «магнито-градиентных» данных.

функция, а скалярная. На рис. 2.7 представлены результаты решения обратной задачи с симулированными экспериментальными данными.

Основной вывод из тестовых численных экспериментов является аналогичным. Обработка только «тензорных» данных позволяет восстановить более детализированное приближённое решение. Обработка совместных «магнитно-градиентных» данных не даёт никаких преимуществ в детализации решения. При обработке экспериментальных данных «тензорные» данные должны использоваться обособленно и не объединяться с «магнитными» данными.

Замечание. Обработка совместных «магнито-градиентных» данных имеет смысл только в случае масштабирования «градиентных» данных до уровня

«магнитных». Такой подход эквивалентен выбору предобуславливателя, который понизит обусловленность системы линейных алгебраических уравнений, которая возникает при численном решении после дискретизации интегрального уравнения (2.5).

Необходимо подчеркнуть, что принципиальное отличие этой модели от использованной в предыдущем подпараграфе заключается в том, что с физической точки зрения решается более перепределённая задача (восстанавливается одна скалярная функция по данным наблюдения значений пяти скалярных функций), что даёт надежду (подтверждённую в том числе экспериментально) на получение более детализированного решения. Также этот подход даёт возможность, восстановив функцию $\chi(\boldsymbol{r})$, $\boldsymbol{r} \in V$, сделать вывод о виде конкретных материалов (полезных ископаемых), которые залегают в недрах Земли.

Однако существенным недостатком использования рассматриваемой модели является необходимость знания точных параметров нормального магнитного поля $\boldsymbol{B}^0(\boldsymbol{r})$ в области V. Как уже упоминалось ранее, значения нормального поля определяются исследователями из некоторых вспомогательных упрощённых моделей (в частности с помощью моделирования магнитного поля Земли с помощью модели диполя). Как следствие, это приводит к тому, что в оператор $\mathbf{A}$ уравнения (2.5) вносится дополнительная ошибка. Учёт этой ошибки при выборе параметра регуляризации по обобщённому принципу невязки, в свою очередь, приводит к увеличению параметра регуляризации, и, как следствие, к более сглаженному (а значит, менее детализированному) приближённому решению. Таким образом, решение, найденное с помощью указанной модели, является достаточно чувствительным к точности определения параметров нормального магнитного поля Земли.

2.3 Обратная задача восстановления параметров намагниченности коры планет Солнечной системы по данным спутниковых измерений

Исследование магнитных полей планет и решение соответствующих обратных задач является одним из способов получения информации о внутренней структуре планет и их эволюции. Доступ к информации о магнитных полях

планет стал возможен благодаря появлению автоматических межпланетных станций (АМС). На данный момент больше всего экспериментальных данных о магнитных полях собрано для Марса и Меркурия.

Первые результаты измерения магнитного поля Марса были получены в 1965 году американской АМС Маринер-4, который обнаружил отсутствие глобальной магнитосферы Марса. В 1970-х годах советские АМС Марс-2, -3, -5 и позже Фобос-2 обнаружили достаточно малое магнитное поле величиной примерно 60 nT в окрестности экватора и 120 nT в окрестности полюса (см. работу В. Ридлера [124]). В 1996 году к Марсу был отправлена американская АМС «Марс Глобал Сервейор» с магнетометром-рефлектометром на борту (см. работу Джона Э. П. Коннерни [125]). Благодаря этому аппарату были получены данные о магнитном поле Марса на различных высотах над поверхностью Красной планеты. Эти данные позволили решать обратные задачи по восстановлению параметров намагниченности (см. работы О. Портнягина и М. С. Жданова [29; 30]). Было обнаружено, что кора Марса местами достаточно сильно намагничена. Из этого был сделан вывод, что, хотя Марс сейчас и не имеет глобального магнитного поля, возможно он имел активное магнитное динамо ранее (см. работу М. Х. Акуны [126]). В связи с этим моделирование остаточного магнитного поля является важной задачей для изучения глубинного строения Марса и для проверки моделей магнитного динамо Марса в прошлом. В последнем случае это даёт возможность создателям моделей магнитного динамо верифицировать свои модели и проверить что будет после того, как оно исчезнет, — совпадёт ли предсказанное этими моделями распределение остаточной намагниченности в коре Марса с наблюдаемыми в наши дни значениями. Экспериментальные данные АМС «Марс Глобал Сервейор» позволили получить первичную оценку глобального распределения источников магнитного поля в коре планеты (см. работы М. Х. Акуны [127; 128]).

Самые свежие исследования базируются на данных экспериментальных наблюдений [129] АМС MAVEN-1 («Mars Atmosphere and Volatile EvolutioN») [130], которая приступила к работе в 2014 году. Множество методов, разработанных для исследования магнитного поля Земли, могут быть применены и для исследования магнитного поля Марса (см., например, работы О. Портнягина и М. С. Жданова [29; 30]). Широко распространённые ранее подходы к моделированию локальных (сильно намагниченные южные высокогорья с сильными магнитными аномалиями, достигающими 1500 nT

на высоте 200 km, — см. работы М. Х. Акуны [131], К. Ф. Шпренке [132] и Д. М. Юрди [133]) и глобальных областей распределения магнитного поля, индуцированных остаточным магнетизмом Марсианской коры, разделялись на модели «разложения по сферически гармоникам» (см. работы Дж. Аркани-Хамеда [134] и Дж. С. Кейна [135]) и модели «эквивалентных источников» (см. работы М. Пурукера [136] и Б. Лангле [137]). Современные модели магнитного поля коры (см. работы А. Миттельхольца [138] и Б. Лангле [139]) учитывают данные магнитометра АМС MAVEN-1 на высоте около 135 km (см., например, работу Джона Э. П. Коннерни [125]). Это связано с тем, что данные полученные уже упомянутым выше магнитометром АМС «Марс Глобал Сервейор» в основном распределены на орбите высотой 370–430 km, некоторые данные доступны на высоте 90–170 km. Поэтому в указанных работах использовалась только часть данных АМС MAVEN-1 с целью сопоставления результатов, полученных на основе данных магнитометра АМС MAVEN-1, с результатами, полученными ранее на основе данных магнитометра АМС «Марс Глобал Сервейор».

В данном диссертационном исследовании используется полный набор данных АМС MAVEN-1. При этом будет использоваться модель для восстановления эквивалентного распределения параметров намагниченности в приповерхностном слое (коре) Марса. Этот подход схож с предложенной в работе Д. Зидарова [140] методикой «выметания» источников полей из многомерной области пространства на её границу. На примере исследования гравитационного поля Марса было показано (см. работы Т. В. Гудковой, И. Э. Степановой и А. В. Батова [141; 142]), что распределение двумерных носителей на плоскости под поверхностью Марса напоминает очертаниями само поле. Поэтому рассматриваемый подход к восстановлению эквивалентного распределения источников магнитного поля является обнадёживающим. Предложенный алгоритм применяется для обработки магнитных данных АМС MAVEN-1 на всех высотах. Полученные результаты демонстрируют эффективность предложенного подхода.

Что касается исследования магнитных полей Меркурия, то за последние 10-20 лет объём информации о физических полях Меркурия увеличился на несколько порядков. Благодаря различным автоматическим межпланетным станциям (АМС), таким как АМС серии «Ма́ринер» и «Мессенджер» (MESSENGER ≡ MErcury Surface, Space ENvironment, GEochemistry and Ranging) исследователи смогли получить высококачественные данные спутни-

ковых наблюдений (см., например, работы И. И. Алекссева [143], Б. Дж. Андерсона [144–147], М А Мэйхъю [148], Н. Ф. Несса [149; 150], С. К. Соломона [151], Л. К. Филпотта [152], Дж. Вихта [153]). Качество данных было обусловлено в первую очередь тем, что специфика орбит космических аппаратов обеспечила получение экспериментальных данных, распределённых в трёхмерном пространстве. Это позволило решать достаточно сложные задачи по определению различных физических параметров планеты.

Как уже было отмечено выше, результаты обработки данных дистанционного зондирования небесных тел используются при уточнении моделей внутреннего строения планет Солнечной системы. Запущенный в рамках совместной японо-европейской программы космический аппарат «БепиКоломбо» должен достигнуть Меркурия в 2025 году, и тогда знания о ближайшей к Солнцу планете будут дополнены качественно новой информацией (см. работу А. Милилло [154]). В наше время исследователи считают, что Меркурий «устроен» приблизительно так, как это описано, например, в работах С. Плагеманна [155] и Д. Э. Смита [156]. Однако новые космические миссии могут подтолкнуть учёных к пересмотру устоявшихся моделей относительно внутренней структуры этой планеты. Это связано с тем, что модели физических полей корректируются при появлении новых массивов данных после запуска каждой очередной межпланетной миссии (см. работы автора [18;19]). На данный момент основной акцент в исследованиях делается на различных подходах к интерпретации основных составляющих полей и их высокочастотных компонент.

Магнитное поле Меркурия не является стационарным. Наличие у планеты магнитного динамо (как это было установлено благодаря данным АМС «Мессенджер» (см. работу С. Топфера [157]) обуславливает весьма сложный характер взаимодействия внутреннего магнитного поля с солнечным ветром, представляющим собой потоки заряженных частиц. При построении аналитических моделей магнитного поля планет, в том числе и Меркурия, применяются как «изолированные» разложения сигналов в ряд по сферическим гармоникам (см., например, работы Б. Лангле [137] и Дж. С. Оливейры [158], так и комбинированные представления компонент магнитного поля в виде суммы полоидальной и тороидальной составляющих — так называемое представление Ми (см. работу С. Топфера [157]). Для построения уточнённых моделей магнитного поля планет необходимо выделить так называемую «коровую» составляющую, которая обусловлена наличием близко залегающих к поверхности

плотностных магнитных неоднородностей (см. работы Т. В. Гудковой [142], Л. Пана [159], К. Л. Джонсона [160], И. Э. Степановой [161–163], А. М. Сальникова [164], Б. Лангле [137], Ф. Дж. Лоуса [165], Л. Л. Худа [166]). Таким образом восстановление приповерхностного магнитного изображения Меркурия является актуальной задачей. При этом отдельно следует отметить важность разделения полей, создаваемых носителями, залегающими на разных глубинах. Вдали от поверхности планеты «коровая» составляющая магнитного поля выглядит как помеха, имеющая малую амплитуду по сравнению с зависящей от времени трендовой компонентой.

2.3.1 Случай планет с наличием только остаточного магнетизма

Случай исследования параметров намагниченности планет, лишённых магнитного динамо, таких как Марс, является более простым, так как восстановлению подлежит лишь остаточная намагниченность магнитных масс, которые индуцируют измеряемое спутниками магнитное поле с индукцией $\boldsymbol{B}$. Так как нормальное поле $\boldsymbol{B}^0$ в связи с отсутствием магнитного динамо равно нулю, то согласно формуле (1.1): $\boldsymbol{B} \equiv \boldsymbol{B}_{field}$. Таким образом, никаких вспомогательных действий по предобработке экспериментальных данных для выделения поля $\boldsymbol{B}_{field}$, индуцируемого магнитными массами в коре планеты, совершать не надо.

Поэтому, для решения задачи определения расположения магнитных масс в приповерхностном слое (коре) Марса можно сразу использовать модель, описанную в подпараграфе 1.1.1 главы 1. Это означает, что **обратная задача** состоит в определении вектор-функции $\boldsymbol{M}(\boldsymbol{r}) \equiv \big(M_x(\boldsymbol{r})\ M_y(\boldsymbol{r})\ M_z(\boldsymbol{r})\big)^T$, $\boldsymbol{r} \equiv (x,y,z) \in V$, из уравнения

$$\boldsymbol{B}_{field}(\boldsymbol{r}_s) = \frac{\mu_0}{4\pi} \iiint\limits_V \left(\frac{3\big(\boldsymbol{M}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\big)(\boldsymbol{r} - \boldsymbol{r}_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} - \frac{\boldsymbol{M}(\boldsymbol{r})}{|\boldsymbol{r} - \boldsymbol{r}_s|^3} \right) dv$$

по данным экспериментальных измерений компонент вектор-функции $\boldsymbol{B}_{field}(\boldsymbol{r}_s) \equiv \big(B_x(\boldsymbol{r}_s)\ B_y(\boldsymbol{r}_s)\ B_z(\boldsymbol{r}_s)\big)^T$ в точках спутниковых наблюдений $\boldsymbol{r}_s \equiv (x_s,y_s,z_s)$, $s = \overline{1,S}$ (на рис. 2.8 приведён пример расположения относительно Марса точек измерения магнитного поля).

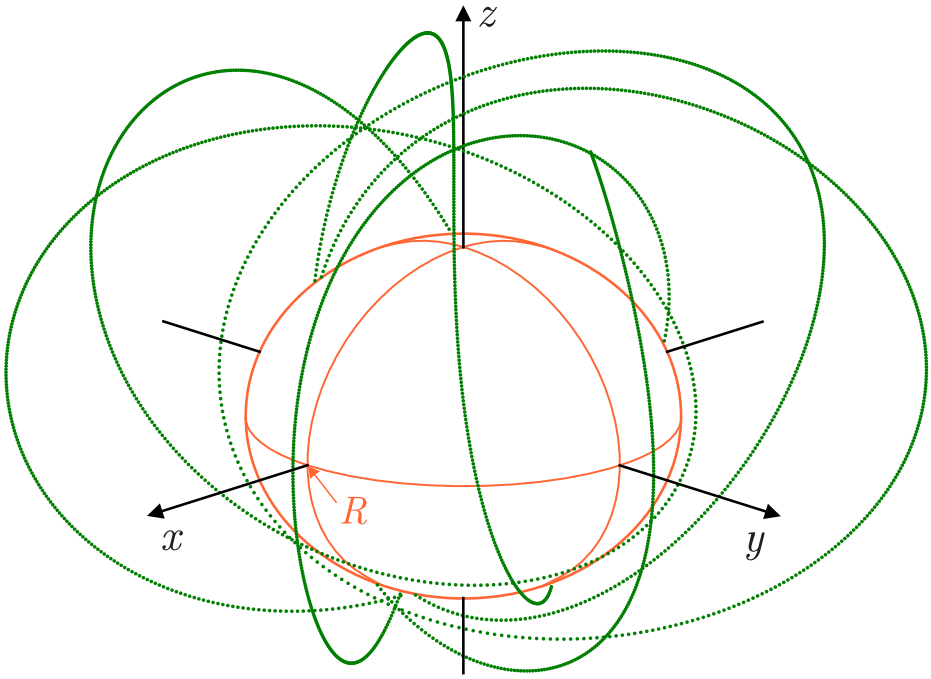


Рисунок 2.8 — Расположение относительно Марса точек измерения магнитного поля автоматической межпланетной станцией MAVEN-1, которые использовались при решении обратной задачи.

Как было показано в подпараграфе 1.1.1 главы 1 это уравнение может быть записано в виде (1.6):

$$\boldsymbol{B}_{field}(x_s, y_s, z_s) = \frac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}_{MI}(x, y, z, x_s, y_s, z_s) \boldsymbol{M}(x, y, z) dv, \qquad (2.6)$$

где

$$\mathbf{K}_{MI}(x, y, z, x_s, y_s, z_s) = \\ = \frac{1}{\rho^5} \begin{bmatrix} 3(x-x_s)^2 - \rho^2 & 3(x-x_s)(y-y_s) & 3(x-x_s)(z-z_s) \\ 3(y-y_s)(x-x_s) & 3(y-y_s)^2 - \rho^2 & 3(y-y_s)(z-z_s) \\ 3(z-z_s)(x-x_s) & 3(z-z_s)(y-y_s) & 3(z-z_s)^2 - \rho^2 \end{bmatrix},$$

и

$$\rho = |\boldsymbol{r} - \boldsymbol{r}_s| = \sqrt{(x-x_s)^2 + (y-y_s)^2 + (z-z_s)^2}.$$

В дополнение к используемой по умолчанию декартовой системе координат удобно использовать и сферическую систему координат (см. рис. 2.9):

$$\begin{aligned} &x = \rho\cos\varphi\sin\theta, \quad y = \rho\sin\varphi\sin\theta, \quad z = \rho\cos\theta, \\ &\rho \in [0, R],\ \theta \in [0, \pi],\ \varphi \in [0, 2\pi). \end{aligned} \qquad (2.7)$$

Здесь R — средний радиус Марса.

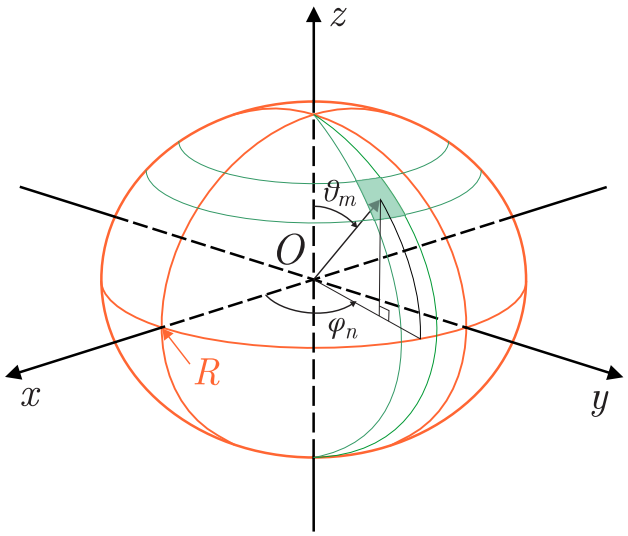


Рисунок 2.9 — Используемая при решении задачи планетарная система координат.

Исходя из предположения, что область V является приповерхностным шаровым слоем Марса глубины h, уравнение (2.6) с учётом (2.7) может быть переписано в виде

$$\begin{aligned}\boldsymbol{B}_{field}(x_s,y_s,z_s) = \\ = \frac{\mu_0}{4\pi}\int\limits_{R-h}^{R}\int\limits_{0}^{2\pi}\int\limits_{0}^{\pi}\mathbf{K}(\rho\cos\varphi\sin\theta,\ \rho\sin\varphi\sin\theta,\ \rho\cos\theta,\ x_s,\ y_s,\ z_s)\cdot \\ \cdot\boldsymbol{M}(\rho\cos\varphi\sin\theta,\ \rho\sin\varphi\sin\theta,\ \rho\cos\theta)\cdot\rho^2\sin\theta\cdot d\rho\,d\varphi\,d\theta.\end{aligned} \quad (2.8)$$

Замечание. Здесь $\boldsymbol{B}\equiv\boldsymbol{B}_{field}$ и $\mathbf{K}\equiv\mathbf{K}_{MI}$. Введение дополнительных обозначения связано с тем, что матричная функция $\mathbf{K}$ (и соответственно вектор-функция $\boldsymbol{B}$) может быть расширена при учёте дополнительной физической информации в постановке задачи. Например, если надо восстановить только модуль намагниченности или доступна информация о значениях компонент тензора градиентов компоент индукции магнитного поля в точках измерения (см. работу автора [12]).

Если ввести сетки $\Phi_{N_\varphi}=\{\varphi_n, 0\leqslant n\leqslant N_\varphi : \varphi_n = h_\varphi\, n,\ h_\varphi=\frac{2\pi}{N_\varphi}\}$ и $\Theta_{N_\theta}=\{\theta_m, 0\leqslant m\leqslant N_\theta : \theta_m = h_\theta\, m,\ h_\theta=\frac{\pi}{N_\theta}\}$, а также, исходя из предположения, что шаровой слой достаточно тонкий, по переменной ρ ввести сетку только с одним узлом $\rho_h = R-\frac{h}{2}$, то, аппроксимируя интегралы в (2.8) по переменным φ и θ по формуле трапеций, а интеграл по переменной ρ по формуле средних, можно получить

$$\boldsymbol{B}(x_s,y_s,z_s) = \frac{\mu_0}{4\pi}\,\rho_h^2\,h \times \sum_{n=1}^{N_\varphi} \Bigg\{$$

$$\sum_{m=1}^{N_\theta} \frac{1}{2}\Big[\mathbf{K}(\rho_h\cos\varphi_{n-1}\sin\theta_{m-1},\ \rho_h\sin\varphi_{n-1}\sin\theta_{m-1},\ \rho_h\cos\theta_{m-1},\ x_s,\ y_s,\ z_s)\cdot$$

$$\cdot\,\boldsymbol{M}(\rho_h\cos\varphi_{n-1}\sin\theta_{m-1},\ \rho_h\sin\varphi_{n-1}\sin\theta_{m-1},\ \rho_h\cos\theta_{m-1})\cdot\sin\theta_{m-1} +$$

$$+\,\mathbf{K}(\rho_h\cos\varphi_{n-1}\sin\theta_m,\ \rho_h\sin\varphi_{n-1}\sin\theta_m,\ \rho_h\cos\theta_m,\ x_s,\ y_s,\ z_s)\cdot$$

$$\cdot\,\boldsymbol{M}(\rho_h\cos\varphi_{n-1}\sin\theta_m,\ \rho_h\sin\varphi_{n-1}\sin\theta_m,\ \rho_h\cos\theta_m)\cdot\sin\theta_m\Big]h_\theta +$$

$$+\sum_{m=1}^{N_\theta} \frac{1}{2}\Big[\mathbf{K}(\rho_h\cos\varphi_n\sin\theta_{m-1},\ \rho_h\sin\varphi_n\sin\theta_{m-1},\ \rho_h\cos\theta_{m-1},\ x_s,\ y_s,\ z_s)\cdot$$

$$\cdot\,\boldsymbol{M}(\rho_h\cos\varphi_n\sin\theta_{m-1},\ \rho_h\sin\varphi_n\sin\theta_{m-1},\ \rho_h\cos\theta_{m-1})\cdot\sin\theta_{m-1} +$$

$$+\,\mathbf{K}(\rho_h\cos\varphi_n\sin\theta_m,\ \rho_h\sin\varphi_n\sin\theta_m,\ \rho_h\cos\theta_m,\ x_s,\ y_s,\ z_s)\cdot$$

$$\cdot\,\boldsymbol{M}(\rho_h\cos\varphi_n\sin\theta_m,\ \rho_h\sin\varphi_n\sin\theta_m,\ \rho_h\cos\theta_m)\cdot\sin\theta_m\Big]h_\theta$$

$$\Bigg\}h_\varphi.$$

Если учесть, что измерения сделаны для всех $s = \overline{1,S}$, а $\boldsymbol{B}$ и $\boldsymbol{M}$ являются векторными функциями, то получается система с $3\times S$ уравнениями (которые соответствуют измерению трёх компонент вектор-функции $\boldsymbol{B}$ в S точках) с $3\times(N_\varphi+1)\times(N_\theta+1)$ неизвестными (которые соответствуют сеточным значениям трёх компонент вектор-функции $\boldsymbol{M}$ на введённой сетке $\Phi_{N_\varphi}\times\Theta_{N_\theta}$).

К этим уравнениям следует добавить следующие естественные физические условия.

1. Условие сшивки вдоль одного из меридианов: магнитное изображение должно быть 2π-периодическим по переменной φ:

$$\boldsymbol{M}(\rho_h\cos\varphi_0\sin\theta_m,\,\rho_h\sin\varphi_0\sin\theta_m,\,\rho_h\cos\theta_m) =$$
$$= \boldsymbol{M}(\rho_h\cos\varphi_{N_\varphi}\sin\theta_m,\,\rho_h\sin\varphi_{N_\varphi}\sin\theta_m,\,\rho_h\cos\theta_m),\quad m=\overline{1,N_\theta-1}.$$

   Эти условия дают дополнительные $3\times(N_\theta-1)$ уравнений.

2. Условие сшивки на южном полюсе: все сеточные значения компонент вектор-функции $\boldsymbol{M}$ должны совпадать при $\theta=\theta_{N_\theta}$:

$$\boldsymbol{M}(\rho_h\cos\varphi_n\sin\theta_{N_\theta},\,\rho_h\sin\varphi_n\sin\theta_{N_\theta},\,\rho_h\cos\theta_{N_\theta}) =$$
$$= \boldsymbol{M}(\rho_h\cos\varphi_{n+1}\sin\theta_{N_\theta},\,\rho_h\sin\varphi_{n+1}\sin\theta_{N_\theta},\,\rho_h\cos\theta_{N_\theta}),\quad n=\overline{0,N_\varphi}.$$

Эти условия дают дополнительные $3 \times (N_\varphi + 1)$ уравнений.

3. Условие сшивки на северном полюсе:

$$\boldsymbol{M}(\rho_h \cos\varphi_n \sin\theta_1, \rho_h \sin\varphi_n \sin\theta_1, \rho_h \cos\theta_1) = \\ = \boldsymbol{M}(\rho_h \cos\varphi_{n+1} \sin\theta_1, \rho_h \sin\varphi_{n+1} \sin\theta_1, \rho_h \cos\theta_1), \quad n = \overline{0,N_\varphi + 1}.$$

Эти условия дают дополнительные $3 \times (N_\varphi + 1)$ уравнений.

Таким образом, с учётом дополнительных физических условий получается система линейных алгебраических уравнений, состоящую из $3S + 3N_\theta + 6N_\varphi + 3$ уравнений, каждое из которых содержит $3 \times (N_\varphi + 1) \times (N_\theta + 1)$ неизвестную (сеточные значения компонент вектор-функции $\boldsymbol{M}$).

Важно отметить, что полученная система при достаточно большом S является переопределённой лишь с численной точки зрения. С физической точки зрения полученная система по-прежнему является определённой, так как надо восстановить одну вектор-функцию по результатам наблюдения одной другой вектор-функции.

Эта система линейных алгебраических уравнений может быть записана в матричном виде

$$AM = B. \tag{2.9}$$

Здесь вектор M содержит сеточные значение трёх компонент неизвестной вектор-функции $\boldsymbol{M}$, первые $3S$ компоненты вектора правой части B содержат результаты экспериментальных измерений трёх компонент вектор-функции $\boldsymbol{B}$, последующие компоненты вектора B являются нулями.

При обработке экспериментальных данных вместо точно известных вектора B и матрицы A обычно известны их приближенные значения B_δ и A_h, такие, что $\|B_\delta - B\|_E \leqslant \delta$, $\|A - A_h\|_{E \to E} \leqslant h$. Это связано как с ошибками измерений индукции магнитного поля (они вносят ошибку в значения компонент вектора B), так и с ошибками в точности определения позиционирования спутника относительно Марса (они вносят ошибку в значения элементы матрицы A). При выписанных условиях задача является некорректной, для её решения необходимо построить регуляризующий алгоритм. Если воспользоваться регуляризирующим алгоритмом, основанным на минимизации функционала А. Н. Тихонова, то задача сводится к поиску элемента, реализующего минимум функционала

$$F^\alpha[M] = \|A_h M - B_\delta\|_E^2 + \alpha\|M\|_E^2.$$

Согласно работе А. Н. Тихонова, А. В. Гочарского, В. В. Степанова и А. Г. Яголы [91] для любого $\alpha > 0$ существует единственная экстремаль M_η^α, $\eta = \{\delta, h\}$, которая реализует минимум функционала $F^\alpha[M]$.

Параметр регуляризации можно выбрать по обобщённому принципу невязки (см. работу [91]), как корень нелинейного уравнения

$$\rho(\alpha) \equiv \|A_h M_\eta^\alpha - B_\delta\|_E^2 - \left(\delta + h\|M_\eta^\alpha\|_E\right)^2 - \mu_\eta^2 = 0.$$

Здесь μ_η — мера несовместности, которая определяется следующим образом:

$$\mu_\eta = \inf_M \left\|A_h M - B_\delta\right\|_E.$$

При этом M_η^α стремиться при $\eta \to 0$ к точному решению задачи.

В качестве метода минимизации функционала А. Н. Тихонова применяется метод сопряжённых градиентов, особенности практической реализации которого для рассматриваемого класса задач описываются в главе 3 «Численные методы» и в главе 4 «Комплекс программ».

Результат обработки экспериментальных данных В качестве экспериментальных данных были взяты данные [129] наблюдений автоматической межпланетной станции MAVEN-1 [130]. При расчётах использовались значения для среднего радиуса Марса $R = 3\,389\,500$ m и толщины приповерхностного слоя $h = 1000$ m. Подробное описание численных параметров приведено в работе автора [12]. Результаты расчётов представлены на рис. 2.10 в полярной системе координат и на рис. 2.11 в равновеликой псевдоцилиндрической проекции Мольвейде-Бабине.

Важно отметить, что изменения значения h (которое было выбрано из экспертных оценок) приведут к пропорциональному изменению значений компонент вектора $\boldsymbol{M}$, однако картина его нормированного значения, представленная на рис. 2.8, останется без изменений.

Также необходимо отметить, что результаты расчётов полностью согласуются с результатами, полученными другими независимыми исследованиями. В частности, в отдельных точках поверхности Марса известны результаты локального измерения магнитного поля различными марсоходами. Таким образом, можно предполагать, что результаты расчётов дают достоверную картину приповерхностного «магнитного изображения» Марса.

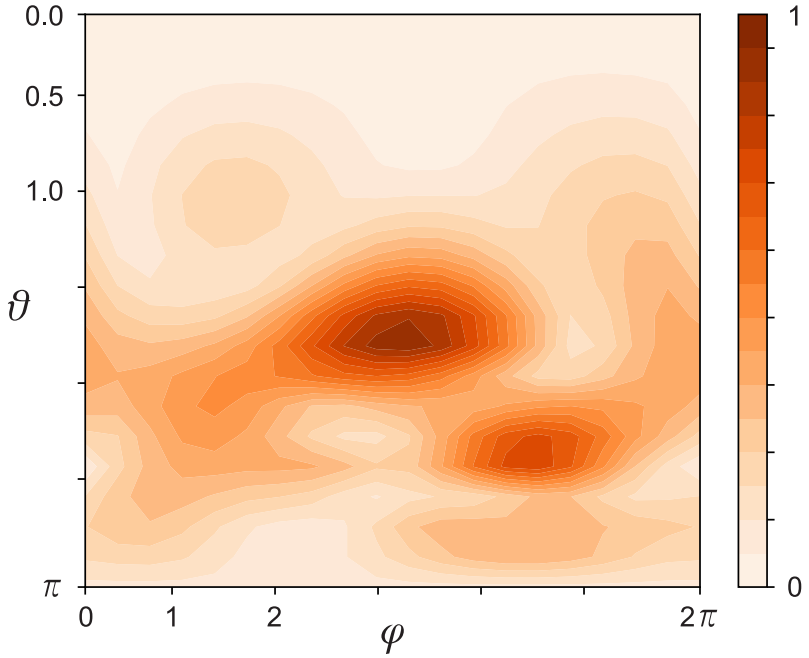


Рисунок 2.10 — Нормализованное значение модуля вектора намагниченности $\boldsymbol{M}$ в полярных координатах.

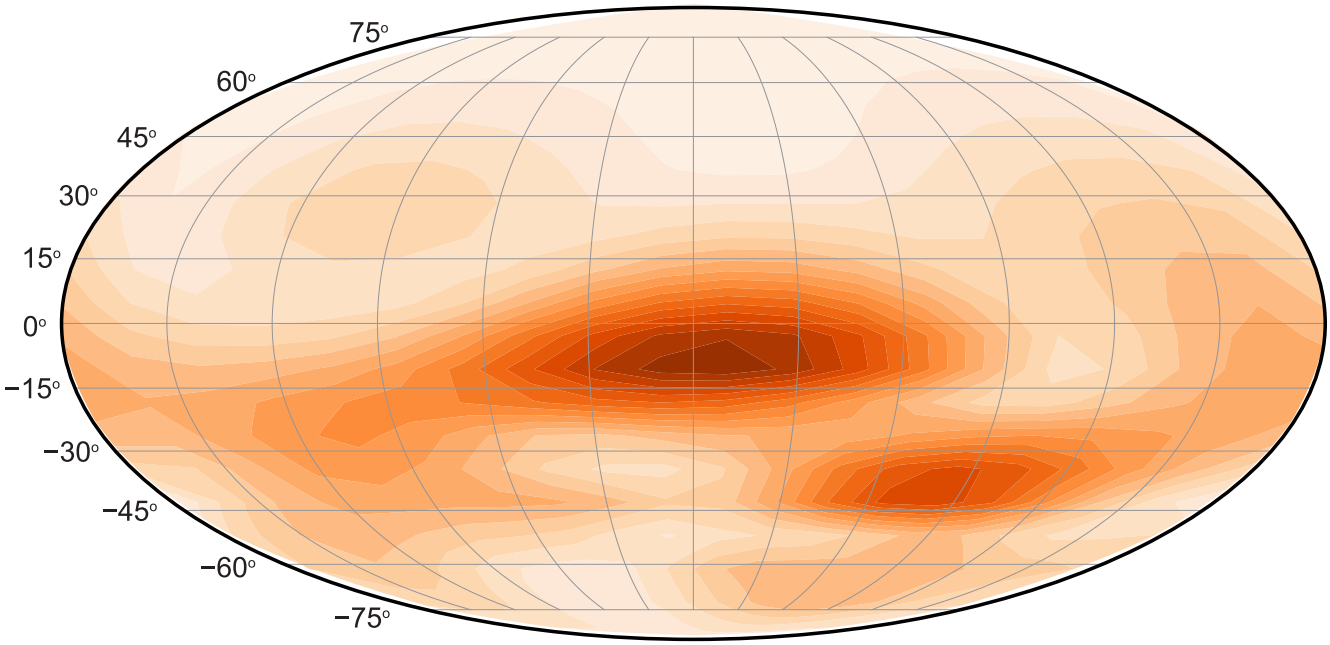


Рисунок 2.11 — Восстановленное приповерхностное «магнитное изображение» Марса (перерисовка рис. 2.10 в проекции Мольвейде-Бабине).

2.3.2 Случай планет с наличием магнитного динамо

Как уже было пояснено, случай исследования параметров намагниченности планет, которые обладают действующим магнитным динамо, таких как Меркурий, является достаточно сложным. Это связано с тем, что измеряемое спутником магнитное поле с индукцией $\boldsymbol{B}$ является суперпозицией (см. формулу (1.1)) неизвестного нормального поля $\boldsymbol{B}^0$ и аномального поля $\boldsymbol{B}_{field}$, которое индуцируется магнитными массами в коре планеты. Таким образом, нужно будет совершить множество вспомогательных действий по предобработке экспериментальных данных с целью выделения требуемой для расчётов составляющей $\boldsymbol{B}_{field}$. При этом вычислительная сложность таких вспомогательных действий может превышать сложность решения самой обратной задачи.

Если из $\boldsymbol{B}$ удалось выделить составляющую $\boldsymbol{B}_{field}$, то все действия по восстановлению приповерхностного «магнитного изображения» Меркурия будут в точности такими же, как и те, что описаны в предыдущем подпараграфе (подпараграф 2.3.1).

Схема предобработки данных экспериментальных измерений магнитного поля с индукцией $\boldsymbol{B}$ следующая [15; 167].

1. Сначала по схеме, описанной в предыдущем подпараграфе (подпараграф 2.3.1), по вектор-функции $\boldsymbol{B}$ восстанавливается вектор-функция $\boldsymbol{M}$ по всему объёму Меркурия. То есть решается полноценная трёхмерная обратная задача восстановления эквивалентной по внешнему полю намагниченности. Найденное распределение намагниченности содержит полную информацию как о «коровой» составляющей, так и о составляющей ядра.
2. По найденному трёхмерному распределению намагниченности перечитывается индуцированное магнитное поля в некотором окружающий Меркурий шаровом слое (см. рис. 2.12-б). Фактически, выполняется «продолжение» экспериментально наблюдаемого в относительно произвольных точках магнитного поля в удобную для дальнейшей обработки область (тонкий шаровой слой на некотором удалении от поверхности Меркурия).
3. С помощью представления Ми выделяется «тонкая структура» магнитного поля (вектор-функция $\boldsymbol{B}_{field}$), соответствующая магнитным

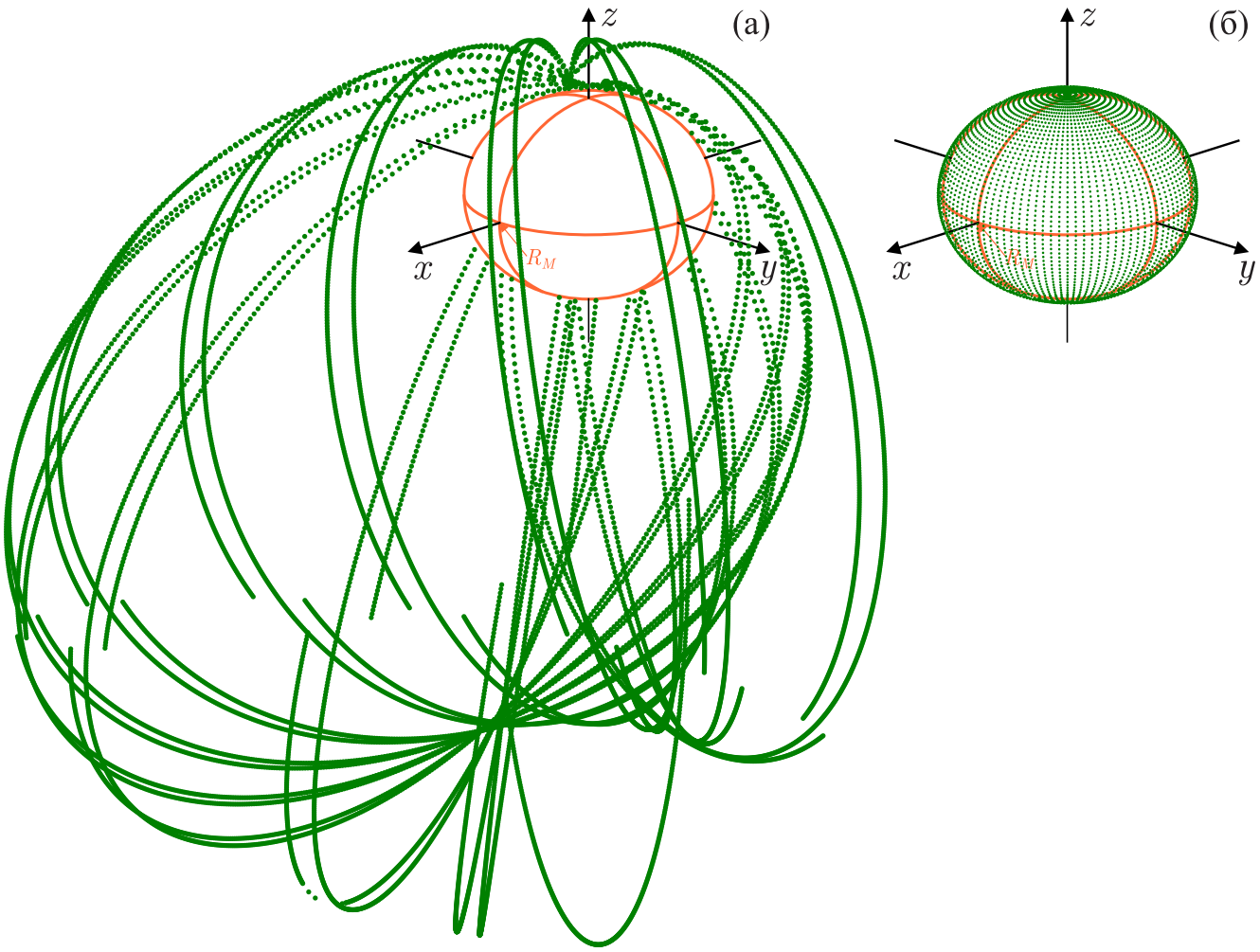


Рисунок 2.12 — Картинка (а): расположение относительно Меркурия точек измерения магнитного поля автоматической межпланетной станцией «Мессенджер»; картинка (б): расположение точек в тонком шаровом слое, в которых было пересчитано магнитное поле.

массам в коре Меркурия. Именно для того, чтобы была возможность выполнить соответствующие аналитические преобразования, и нужно «продолжение» экспериментально измеренного поля в более удобную для аналитических преобразований область.

Первые два пункта этой схемы могут быть реализованы по рассмотренным ранее алгоритмам. При этом надо отметить, что для поиска эквивалентной по внешнему полю намагниченности по всему объёму Меркурия необходимо решить трёхмерную задачу, а после выделения искомого поля $\boldsymbol{B}_{field}$ восстановление искомой вектор-функции будет осуществляться в двумерном приповерхностном слое (коре) планеты. Как следствие, вычислительная сложность предобработки экспериментальных данных будет выше, чем вычислительная сложность решения основной части обратной задачи.

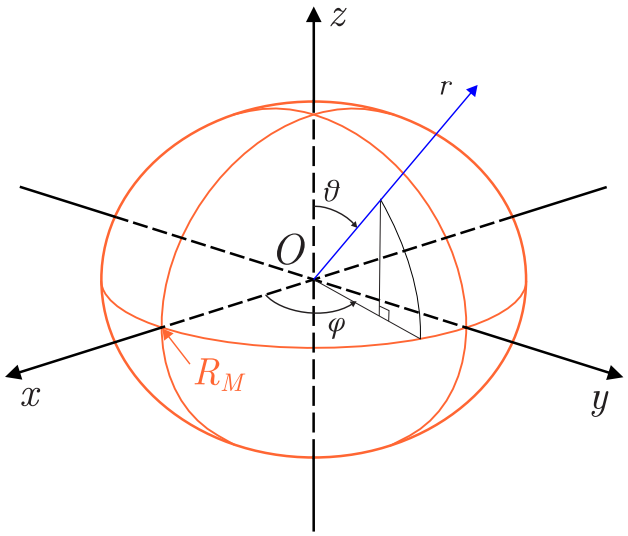


Рисунок 2.13 — Используемая при решении задачи планетарная система координат.

Предположим, что магнитное поле уже пересчитано для окружающего Меркурий шарового слоя $S(a,c) = \{r : a < r < c\}$. Внутренний радиус этого шарового слоя равен a, внешний c. Разложение Гаусса-Ми внутри этого шарового слоя, согласно принципу суперпозиции (см. работы С. Топфера [157; 168]), имеет вид

$$\boldsymbol{B} = \boldsymbol{B}^{int} + \boldsymbol{B}^{ext} + \boldsymbol{B}^{sh}_T + \boldsymbol{B}^{sh}_P. \tag{2.10}$$

Здесь $\boldsymbol{B}^{sh}_T + \boldsymbol{B}^{sh}_P$ — разложение Мия на тороидальную и полоидальную компоненту (T — «toroidal», P — «poloidal», sh — «shell»); $\boldsymbol{B}^{int} = -\nabla\Phi^i$ — внутреннее поле (int — internal), порождённое токами в области $r < a$; $\boldsymbol{B}^{ext} = -\nabla\Phi^{ext}$ — внешнее поле (ext — «external»), порождённое токами в области $r > c$; $\boldsymbol{B}^{sh}_T = [\nabla, \boldsymbol{r}\Psi^{sh}_T]$ — поле в области $a < r < c$, порождённое полоидальными токами; $\boldsymbol{B}^{sh}_P = [\nabla, [\nabla, \boldsymbol{r}\Psi^{sh}_P]]$ — поле в области $a < r < c$, порождённое тороидальными токами; $\boldsymbol{r} = r\,\boldsymbol{i}_r$ ($\boldsymbol{i}_r$ — единичный вектор в сферической системе координат, см. рис. 2.9); функции Φ^{int}, Φ^{ext}, Ψ^{sh}_T, Ψ^{sh}_P — скалярные потенциалы.

Для выделения «тонких структур» необходимо из поля $\boldsymbol{B}^{int}$ выделить высокочастотную составляющую $\boldsymbol{B}^{int}_{high}$. Как это сделать, детально будет изложено далее. И именно эта высокочастотная составляющая $\boldsymbol{B}^{int}_{high}$ является тем самым полем, которое в диссертационной работе обозначается как $\boldsymbol{B}_{field}$.

Для выделения высокочастотной составляющей внутреннего магнитного поля Меркурия необходимо параметризовать поле (2.10) через коэффициенты разложения по сферическим гармоникам потенциалов соответствующих полей в разложении Гаусса-Ми. Для этого будет использоваться планетарная система

координат, изображенная на рис. 2.13 с $r \in [R_M, +\infty)$, $\varphi \in [0, 2\pi)$ и $\theta \in [0, \pi]$. Здесь R_M — средний радиус Меркурия. В этой системе координат скалярные потенциалы, согласно работам С. Топфера [157; 168], можно представить следующим образом:

$$\Phi^{int}(r,\varphi,\theta) = R_M \sum_{l=1}^{\infty}\sum_{m=0}^{l}\left(\frac{R_M}{r}\right)^{l+1}[g_l^m\cos(m\varphi)+h_l^m\sin(m\varphi)]P_l^m(\cos(\theta)),$$

$$\Phi^{ext}(r,\varphi,\theta) = R_M \sum_{l=1}^{\infty}\sum_{m=0}^{l}\left(\frac{r}{R_M}\right)^{l}[q_l^m\cos(m\varphi)+s_l^m\sin(m\varphi)]P_l^m(\cos(\theta)),$$

$$\begin{aligned}\Psi_T^{sh}(r,\varphi,\theta) = \left(\frac{R_M}{r}\right)\sum_{l=1}^{\infty}\sum_{m=0}^{l}&[a_l^m\cos(m\varphi)+b_l^m\sin(m\varphi)+\\ &+\frac{r-b}{R_M}\bar{a}_l^m\cos(m\varphi)+\frac{r-b}{R_M}\bar{b}_l^m\sin(m\varphi)]P_l^m(\cos(\theta)),\end{aligned}$$

$$\begin{aligned}\Psi_P^{sh}(r,\varphi,\theta) = \left(\frac{R_M^2}{r}\right)\sum_{l=1}^{\infty}\sum_{m=0}^{l}&[c_l^m\cos(m\varphi)+d_l^m\sin(m\varphi)+\\ &+\frac{r-b}{R_M}\bar{c}_l^m\cos(m\varphi)+\frac{r-b}{R_M}\bar{d}_l^m\sin(m\varphi)]P_l^m(\cos(\theta)).\end{aligned}$$

Здесь $b = \frac{a+c}{2}$, а $g_l^m, h_l^m, q_l^m, s_l^m, a_l^m, b_l^m, \bar{a}_l^m, \bar{b}_l^m$ — искомые коэффициенты, $P_l^m(\cos(\theta))$ — присоединённые полиномы Лежандра (с нормировкой по Шмидту) степени l и порядка m.

Уравнение (2.10) можно записать через потенциалы. Для этого необходимо отметить, что в рассматриваемой задаче толщина шарового слоя $S(a,c) = \{r : a < r < c\}$ считается достаточно малой, чтобы можно было использовать приближение тонкой оболочки, т.е. не учитывать $\boldsymbol{B}_P^{sh}$ [157; 168]. Поэтому уравнение (2.10) примет вид

$$\boldsymbol{B} = -\nabla\Phi^{int} - \nabla\Phi^{ext} + [\nabla, \boldsymbol{r}\Psi_T^{sh}].$$

Для вычисления градиентов и векторных произведений будут использоваться следующие обозначения:

1. $K_l^m(\varphi; g_l^m, h_l^m) := [g_l^m\cos(m\varphi)+h_l^m\sin(m\varphi)]$.
2. $R_1^l(r) := \left(\frac{R_M}{r}\right)^{l+1}, \quad R_2^l(r) := \left(\frac{r}{R_M}\right)^{l}$.
3. $\nabla(\cdot) := \frac{\partial(\cdot)}{\partial r}\boldsymbol{i}_r + \frac{1}{r}\frac{\partial(\cdot)}{\partial\theta}\boldsymbol{i}_\theta + \frac{1}{r\sin(\theta)}\frac{\partial(\cdot)}{\partial\varphi}\boldsymbol{i}_\varphi$.

Используя введённые обозначения, выражения для потенциалов могут быть переписаны следующим образом:

$$\Phi^{int}(r,\varphi,\theta) = R_M \sum_{l=1}^{\infty}\sum_{m=0}^{l} R_1^l(r) K_l^m(\varphi; g_l^m, h_l^m) P_l^m(\cos(\theta)),$$

$$\Phi^{ext}(r,\varphi,\theta) = R_M \sum_{l=1}^{\infty}\sum_{m=0}^{l} R_2^l(r) K_l^m(\varphi; q_l^m, s_l^m) P_l^m(\cos(\theta)),$$

$$\Psi_T^{sh}(r,\varphi,\theta) = \sum_{l=1}^{\infty}\sum_{m=0}^{l} \frac{R_M}{r} K_l^m(\varphi; a_l^m, b_l^m) P_l^m(\cos(\theta)) + \\ + \sum_{l=1}^{\infty}\sum_{m=0}^{l} \frac{r-b}{r} K_l^m(\varphi; \bar{a}_l^m, \bar{b}_l^m) P_l^m(\cos(\theta)).$$

Замечание. В рассматриваемой задаче для потенциалов внутреннего и внешнего поля $l = 3$, для потенциала тороидального поля $l = 2$.

Согласно работам С. Топфера [157; 168] тороидальное магнитное поле в шаровом слое $S(a,c) = \{r : a < r < c\}$ может быть переписано как

$$\boldsymbol{B}_T^{sh} = [\nabla, \boldsymbol{r}\Psi_T^{sh}] = [\nabla, \Psi_T^{sh} r \boldsymbol{i}_r] = \frac{1}{r^2\sin(\theta)} \begin{vmatrix} \boldsymbol{i}_r & r\boldsymbol{i}_\theta & r\sin(\theta)\boldsymbol{i}_\varphi \\ \frac{\partial}{\partial r} & \frac{\partial}{\partial \theta} & \frac{\partial}{\partial \varphi} \\ r\Psi_T^{sh} & 0 & 0 \end{vmatrix} = \\ = 0\boldsymbol{i}_r + \frac{1}{r\sin(\theta)}\frac{\partial \Psi_T^{sh}}{\partial \varphi}\boldsymbol{i}_\theta - \frac{\partial \Psi_T^{sh}}{\partial \theta}\boldsymbol{i}_\varphi.$$

Таким образом, в сферической системе координат в каждой фиксированной точке (r,φ,θ) внутри шарового слоя $S(a,c) = \{r : a < r < c\}$ поле будет иметь следующий вид:

$$B_r(r,\varphi,\theta) = -R_M \sum_{l=1}^{3}\sum_{m=0}^{l} \frac{\partial R_1^l(r)}{\partial r} K_l^m(\varphi; g_l^m, h_l^m) P_l^m(\cos(\theta)) - \\ - R_M \sum_{l=1}^{3}\sum_{m=0}^{l} \frac{\partial R_2^l(r)}{\partial r} K_l^m(\varphi; q_l^m, s_l^m) P_l^m(\cos(\theta)),$$

$$B_\varphi(r,\varphi,\theta) = -\frac{R_M}{r\sin(\theta)} \sum_{l=1}^{3}\sum_{m=0}^{l} R_1^l(r) \frac{\partial K_l^m(\varphi; g_l^m, h_l^m)}{\partial \varphi} P_l^m(\cos(\theta)) - \\ - \frac{R_M}{r\sin(\theta)} \sum_{l=1}^{3}\sum_{m=0}^{l} R_2^l(r) \frac{\partial K_l^m(\varphi; q_l^m, s_l^m)}{\partial \varphi} P_l^m(\cos(\theta)) - \\ - \sum_{l=1}^{2}\sum_{m=0}^{l} \frac{R_M}{r} K_l^m(\varphi; a_l^m, b_l^m) \frac{\partial P_l^m(\cos(\theta))}{\partial \theta} -$$

$$
\begin{aligned}
&- \sum_{l=1}^{2}\sum_{m=0}^{l}\frac{r-b}{r}K_l^m(\varphi;\bar{a}_l^m,\bar{b}_l^m)\frac{\partial P_l^m(\cos(\theta))}{\partial\theta},\\
B_\theta(r,\varphi,\theta) = &-\frac{R_M}{r}\sum_{l=1}^{3}\sum_{m=0}^{l}R_1^l(r)K_l^m(\varphi;g_l^m,h_l^m)\frac{\partial P_l^m(\cos(\theta))}{\partial\theta}-\\
&-\frac{R_M}{r}\sum_{l=1}^{3}\sum_{m=0}^{l}R_2^l(r)K_l^m(\varphi;q_l^m,s_l^m)\frac{\partial P_l^m(\cos(\theta))}{\partial\theta}-\\
&-\frac{1}{r\sin(\theta)}\sum_{l=1}^{2}\sum_{m=0}^{l}\frac{R_M}{r}\frac{K_l^m(\varphi;a_l^m,b_l^m)}{\partial\varphi}P_l^m(\cos(\theta))-\\
&-\frac{1}{r\sin(\theta)}\sum_{l=1}^{2}\sum_{m=0}^{l}\frac{r-b}{r}\frac{K_l^m(\varphi;\bar{a}_l^m,\bar{b}_l^m)}{\partial\varphi}P_l^m(\cos(\theta)).
\end{aligned}
\tag{2.11}
$$

То есть полное поле $\boldsymbol{B}$ параметризуется коэффициентами внутреннего Гауссова диполя $[g_1^0\ \ h_1^0\ \ g_1^1\ \ h_1^1]^T$, коэффициентами внутреннего Гауссова квадруполя $[g_2^0\ \ h_2^0\ \ g_2^1\ \ h_2^1\ \ g_2^2\ \ h_2^2]^T$, коэффициентами внутреннего Гауссова октуполя $[g_3^0\ \ h_3^0\ \ g_3^1\ \ h_3^1\ \ g_3^2\ \ h_3^2\ \ g_3^3\ \ h_3^3]^T$, коэффициентами внешнего Гауссова диполя $[q_1^0\ \ s_1^0\ \ q_1^1\ \ s_1^1]^T$, коэффициентами внешнего Гауссова квадруполя $[q_2^0\ \ s_2^0\ \ q_2^1\ \ s_2^1\ \ q_2^2\ \ s_2^2]^T$, коэффициентами внешнего Гауссова октуполя $[q_3^0\ \ s_3^0\ \ q_3^1\ \ s_3^1\ \ q_3^2\ \ s_3^2\ \ q_3^3\ \ s_3^3]^T$ и тороидальными коэффициентами $[a_1^0\ \ b_1^0\ \ a_1^1\ \ b_1^1\ \ a_2^0\ \ b_2^0\ \ a_2^1\ \ b_2^1\ \ a_2^2\ \ b_2^2]^T$.

Все эти параметры записываются в один вектор коэффициентов разложения Гаусса-Ми:

$$
\begin{aligned}
\hat{g} = \Big[&g_1^0\ h_1^0\ g_1^1\ h_1^1\ g_2^0\ h_2^0\ g_2^1\ h_2^1\ g_2^2\ h_2^2\ g_3^0\ h_3^0\ g_3^1\ h_3^1\ g_3^2\ h_3^2\ g_3^3\ h_3^3\\
&q_1^0\ s_1^0\ q_1^1\ s_1^1\ q_2^0\ s_2^0\ q_2^1\ s_2^1\ q_2^2\ s_2^2\ q_3^0\ s_3^0\ q_3^1\ s_3^1\ q_3^2\ s_3^2\ q_3^3\ s_3^3\\
&a_1^0\ b_1^0\ a_1^1\ b_1^1\ a_2^0\ b_2^0\ a_2^1\ b_2^1\ a_2^2\ b_2^2\Big]^T.
\end{aligned}
$$

Необходимо отметить, что предполагается, что все точки измерения магнитного поля $\boldsymbol{B}$ располагаются на сфере радиуса b и их количество равно S, поэтому в системе (2.11) в выражениях для B_θ и B_φ можно не рассматривать последнюю группу слагаемых.

Для перехода от (2.11) к полной системе линейных алгебраических уравнений необходимо ввести сетки $\Phi_{N_\varphi^s} = \{\varphi_n, 1 \leqslant n \leqslant N_\varphi^s : \varphi_n = \frac{h_\varphi^s}{2} + h_\varphi^s\,(n-1),\ h_\varphi^s = \frac{2\pi}{N_\varphi^s}\}$, $\Theta_{N_\theta^s} = \{\theta_m^s, 1 \leqslant m \leqslant N_\theta^s : \theta_m^s = \frac{h_\theta^s}{2} + h_\theta^s\,(m-1),\ h_\theta^s = \frac{\pi}{N_\theta^s}\}$ и $R_{N_r^s} = \{r_k, 1 \leqslant k \leqslant N_r^s : r_k = a + \frac{h_r^s}{2} + h_r^s\,(k-1),\ h_r^s = \frac{c-a}{N_r^s}\}$ и записать

конечно-разностные аппроксимации уравнений системы (2.11). Таким образом, для выделения внутренней составляющей магнитной индукции $\boldsymbol{B}^{int}$ получена система из $3S = 3N_r^s N_\theta^s N_\varphi^s$ линейных алгебраических уравнений:

$$A\hat{g} = B. \tag{2.12}$$

Здесь вектор правой части B содержит измеренные в узлах сетки компоненты магнитного поля (левые части системы (2.11), а матрица A состоит из элементов, определяемых конечно-разностной аппроксимацией правых частей системы (2.11).

Для поиска коэффициентов разложения Гаусса-Ми $\hat{g}$ необходимо найти решение системы линейных алгебраических уравнений (2.12). Это может быть сделано с помощью метода наименьших квадратов. Таким образом (псевдо)решение системы может быть найдено по формуле

$$\hat{g} = (A^T A)^{-1} A^T B.$$

Внутреннему магнитному полю соответствует вектор

$$\hat{g}_{int} = [g_1^0\ h_1^0\ g_1^1\ h_1^1\ g_2^0\ h_2^0\ g_2^1\ h_2^1\ g_2^2\ h_2^2\ g_3^0\ h_3^0\ g_3^1\ h_3^1\ g_3^2\ h_3^2\ g_3^3\ h_3^3]^T.$$

Затем из поля $\boldsymbol{B}^{int}$ необходимо выделить высокочастотную составляющую $\boldsymbol{B}_{high}^{int}$, которая отвечает за магнитные массы в приповерхностном слое Меркурия и позволяет локализовать «тонкие структуры».

Для получения высокочастотной составляющей из вектора $\hat{g}_{int}$ необходимо взять компоненты разложения, начиная с шестого. Таким образом, «отбрасывается» дипольная часть внутреннего магнитного поля, которая может заглушить все прочие компоненты поля, и появляется возможность локализовать «тонкие структуры». Поэтому коэффициенты Гаусса высокочастотной составляющей магнитного поля определяются следующим образом:

$$\hat{g}_{int}^{high} := [g_1^0 = 0\ h_1^0 = 0\ g_1^1 = 0\ h_1^1 = 0\ g_2^0 = 0\ h_2^0 \\ g_2^1\ h_2^1\ g_2^2\ h_2^2\ g_3^0\ h_3^0\ g_3^1\ h_3^1\ g_3^2\ h_3^2\ g_3^3\ h_3^3]^T.$$

Эти коэффициенты используются для вычисления высокочастотной составляющей $\boldsymbol{B}_{high}^{int}$, компоненты которой определяются по следующим формулам:

$$B_r(r,\varphi,\theta) = -R_M \sum_{l=1}^{3}\sum_{m=0}^{l} \frac{\partial R_1^l(r)}{\partial r} K_l^m(\varphi; g_l^m, h_l^m) P_l^m(\cos(\theta)),$$

$$B_\varphi(r,\varphi,\theta) = -\frac{R_M}{r\sin(\theta)} \sum_{l=1}^{3}\sum_{m=0}^{l} R_1^l(r) \frac{\partial K_l^m(\varphi; g_l^m, h_l^m)}{\partial \varphi} P_l^m(\cos(\theta)),$$

$$B_\theta(r,\varphi,\theta) = -\frac{R_M}{r} \sum_{l=1}^{3}\sum_{m=0}^{l} R_1^l(r) K_l^m(\varphi; g_l^m, h_l^m) \frac{\partial P_l^m(\cos(\theta))}{\partial \theta}.$$

Переход к декартовым координатам осуществляется стандартным преобразованием поворота. Полученное магнитное поле $\boldsymbol{B}_{high}^{int}$ совпадает с полем, для которого в диссертационном исследовании используется обозначение $\boldsymbol{B}_{field}$, и используется при решении обратной задачи для локализации «тонких структур» в приповерхностном слое (коре) Меркурия.

Как уже было отмечено ранее, алгоритм поиска $\boldsymbol{M}(\boldsymbol{r})$, $\boldsymbol{r} \in V$ по известным значениям $\boldsymbol{B}_{field} \equiv \boldsymbol{B}_{high}^{int}(\boldsymbol{r}_s)$, в точности повторяет алгоритм, подробно описанный в предыдущем подпараграфе, поэтому его описание здесь опускается.

Результаты обработки экспериментальных данных Для выделения высокочастотной компоненты внутреннего магнитного поля метод был применён для трёх шаровых слоёв, окружающих Меркурий (см. рис. 2.13). Для этого была предварительно решена обратная задача по восстановлению эквивалентного по внешнему полю распределения вектора намагниченности по всему объёму Меркурия на основе результатов статьи автора [13]. Затем по полученным данным была решена прямая задача по «продолжению» магнитного поля на выбранные шаровые слои.

Подробное описание численных параметров приведено в работе автора [15]. Результаты расчётов представлены на рис. 2.14 в полярной системе координат и на рис. 2.15 в равновеликой псевдоцилиндрической проекции Мольвейде-Бабине.

Полученные результаты согласуются с результатами, полученными другими учёными. Таким образом, можно предполагать, что результаты расчётов дают достоверную картину приповерхностного «магнитного изображения» Меркурия.

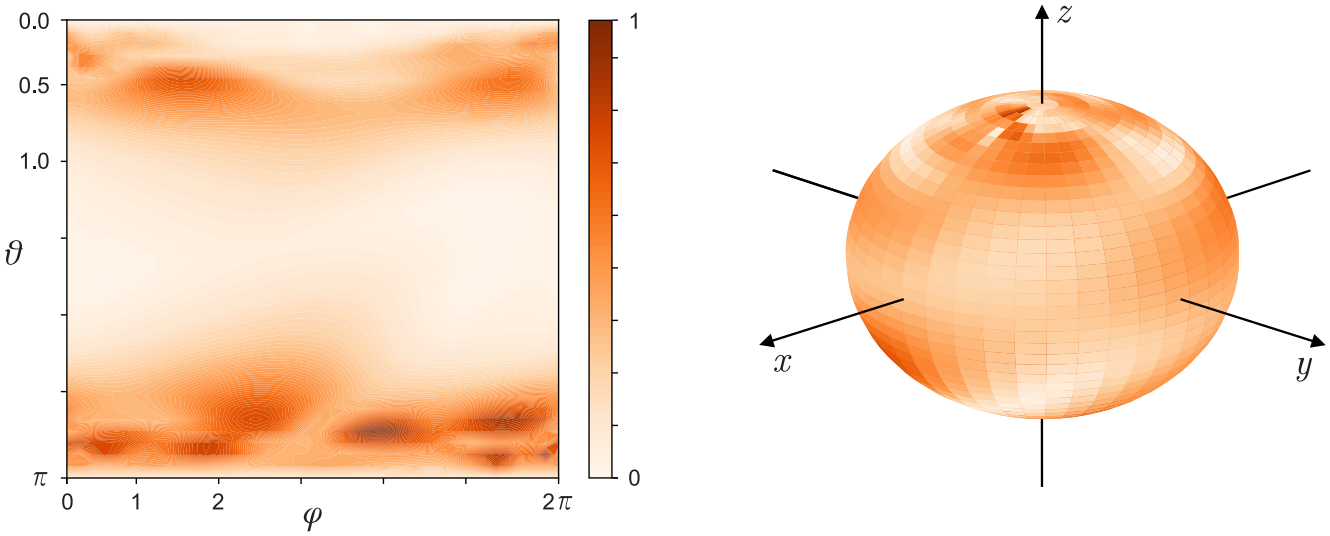


Рисунок 2.14 — Нормализованное значение модуля вектора намагниченности $\boldsymbol{M}$.

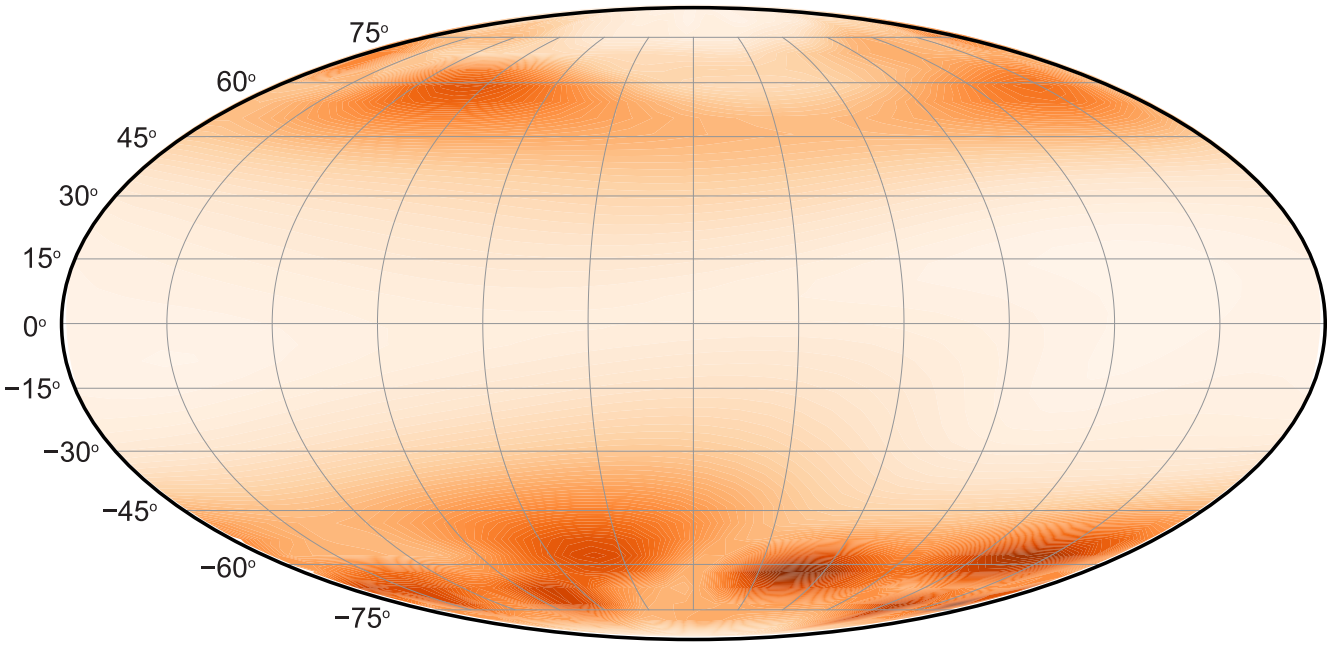


Рисунок 2.15 — Восстановленное приповерхностное «магнитное изображение» Меркурия (перерисовка рис. 2.14 в проекции Мольвейде-Бабине).

2.3.3 Обсуждение результатов математического моделирования

Необходимо подчеркнуть, что вне источников поля (т.е. вне внешнего жидкого ядра Меркурия и других космических объектов, обладающих внутренним генератором магнитного поля) методика разложения сигналов в ряд по сферическим гармоникам, представления в виде интегралов Фурье, вейвлетов и т.п. (см. работы П. Г. Фрика [169] и С. Г. Казанцева [170]) вполне допустима и может давать хорошие результаты при решении задач анализа и синтеза. Но адекватное реальности моделирование магнитного поля Меркурия невозможно получить без учёта магнито-гидродинамических соотношений (см. работы М. Ю. Решетняка [171; 172]). Пренебрежение этими соотношениями вносит дополнительную ошибку в модель и, как следствие, в оператор соответствующего уравнения. Так как решаемые задачи являются некорректно поставленными, это может приводить к неадекватному математическому моделированию решений соответствующих обратных задач. Поэтому все математические модели физических полей должны проходить должную апробацию: они должны гарантировать высокую точность построения линейных трансформант поля, к которым относятся аналитическое продолжение в сторону источников, высшие производные потенциала и т.п. В противном случае, можно получить модели, не соответствующие требованию адекватности реальным физическим данным.

При моделировании магнитного поля планет Солнечной системы другими авторами применялись различные способы решения обратных задач (см., например, работы Х. Уно [173] и К. А. Уэйлера [174]). В последней работе была построена модель непрерывно намагниченного Марса. Однако, как указывалось выше, восстановление источников физических полей по данным дистанционного зондирования является некорректно поставленной задачей, поэтому допустимы самые разные интерпретации спутниковой информации. Выбор наиболее адекватной из существующих аналитических моделей магнитного поля может быть сделан только при всестороннем анализе данных различных геофизических исследований.

Отдельно следует отметить важность разделения полей, создаваемых носителями, залегающими на разных глубинах. Вдали от поверхности планеты «коровая» составляющая магнитного поля выглядит как помеха, имеющая малую амплитуду по сравнению с зависящей от времени трендовой компонентой.

Поэтому столь актуальными представляются новые эффективные регуляризирующие алгоритмы, которые позволяют получать устойчивое к случайным помехам во входных данных приближённое решение обратной задачи магниторазведки.

Предложенная в диссертационном исследовании оригинальная методика восстановления приповерхностного магнитного изображения Меркурия по спутниковым данным позволяет выделить высокочастотную составляющую из огромного массива «сырых» экспериментальных данных о магнитном поле планет Солнечной системы. Важно подчеркнуть, что разработанный алгоритм фильтрации «дипольной» составляющей глобального магнитного поля Меркурия может быть полезен при исследованиях других физических полей планет Солнечной системы, в частности, гравитационного поля. Как правило, на практике очень трудно отделить слабые сигналы от маломощных и глубоко залегающих источников от случайных помех. Использование регуляризирующих алгоритмов позволяет осуществлять «тонкую настройку» параметра регуляризации, при которой получаются физически осмысленные решения некорректных поставленных обратных задач. Сами по себе математические процедуры фильтрации разнородных данных, основанные на алгоритмах подавления помехи или полезного сигнала (в зависимости от контекста), не могут считаться в полной мере реализованными до тех пор, пока не будут приведены результаты решения конкретных задач по обработке больших и сверхбольших массивов данных о физических, химических, тепловых и т.п. свойствах природных объектов. В особенности это касается методики интерпретации спутниковых данных о физических полях и топографии Земли и планет. Поэтому в диссертационной работе и делается акцент на надёжные вычислительные алгоритмы для решения задач инверсии магнитного поля.

Глава 3. Численные методы

В главах 1 и 2 было показано, что все рассматриваемые в диссертационном исследовании обратные задачи магнитометрии сводятся к решению интегрального уравнения Фредгольма 1-го рода следующего вида:

$$\mathbf{A}\boldsymbol{X} \equiv \frac{\mu_0}{4\pi} \iiint\limits_V \mathbf{K}(x,y,z,x_s,y_s,z_s)\,\boldsymbol{X}(x,y,z)\,dxdydz = \boldsymbol{B}(x_s,y_s,z_s). \qquad (3.1)$$

Здесь $\boldsymbol{X} \equiv \boldsymbol{M}$, если целью решения уравнения (3.1) является поиск распределения намагниченности по объёму V; $\boldsymbol{X} \equiv \chi$, если восстанавливается распределение магнитной восприимчивости.

В задачах рассматриваемого типа предполагается, что $\boldsymbol{X} \in \square(V)$, $\boldsymbol{B} \in L_2(Q)$ (Q — область расположения измерительных датчиков), и оператор $\mathbf{A}$ ($\mathbf{A} : \square \to L_2$) с ядром $\mathbf{K}$ является непрерывным и однозначным. Символ $\square$ используется для подстановки вместо него одного из пространств, которому, согласно априорной информации о решении, принадлежит решение обратной задачи. В зависимости от наличия или отсутствия априорной информации о решении $\square \equiv W_2^2$ либо $\square \equiv L_2$.

Пара вектор-функций $\{\boldsymbol{X}, \boldsymbol{B}\}$ определяют тип решаемой обратной задачи магнитометрии. Ядро интегрального уравнения — матричная функция $\mathbf{K}$ — выбирается согласованной с этой парой векторных функций следующим образом.

1. $\boldsymbol{X} = \boldsymbol{M} \equiv \left(M_x\ M_y\ M_z\right)^T$ и $\boldsymbol{B} \equiv \left(B_x\ B_y\ B_z\right)^T$ определяют обратную задачу восстановления вектор-функции намагниченности по данным экспериментальных измерений компонент вектора индукции магнитного поля. В этом случае $\mathbf{K} \equiv \mathbf{K}_{MI}$.
2. $\boldsymbol{X} = \boldsymbol{M} \equiv \left(M_x\ M_y\ M_z\right)^T$ и $\boldsymbol{B} \equiv \left(\frac{\partial B_x}{\partial x}\ \frac{\partial B_x}{\partial y}\ \frac{\partial B_x}{\partial z}\ \frac{\partial B_y}{\partial z}\ \frac{\partial B_z}{\partial z}\right)^T$ определяют обратную задачу восстановления вектор-функции намагниченности по данным экспериментальных измерений независимых компонент тензора градиентов компонент вектора индукции магнитного поля. В этом случае $\mathbf{K} \equiv \mathbf{K}_{MGT}$.
3. $\boldsymbol{X} = \boldsymbol{M} \equiv \left(M_x\,M_y\,M_z\right)^T$ и $\boldsymbol{B} \equiv \left(B_x\,B_y\,B_z\,\frac{\partial B_x}{\partial x}\,\frac{\partial B_x}{\partial y}\,\frac{\partial B_x}{\partial z}\,\frac{\partial B_y}{\partial z}\,\frac{\partial B_z}{\partial z}\right)^T$ определяют обратную задачу восстановления вектор-функции намагниченности по данным экспериментальных измерений компонент вектора индук-

ции и независимых компонент тензора градиентов компонент вектора индукции магнитного поля. В этом случае $\mathbf{K} \equiv \left(\mathbf{K}_{MI}\, \mathbf{K}_{MGT}\right)^T$.

4. $\boldsymbol{X} = \chi$ и $\boldsymbol{B} \equiv \left(B_x B_y B_z\right)^T$ определяют обратную задачу восстановления скалярной функции магнитной восприимчивости по данным экспериментальных измерений компонент вектора индукции магнитного поля и дополнительной информации о векторной-функции $\boldsymbol{B}^0$, определяющей нормальное поле в области восстановления решения. В этом случае $\mathbf{K} \equiv \mathbf{K}_{MI}^{B^0}$.
5. $\boldsymbol{X} = \chi$ и $\boldsymbol{B} \equiv \left(\frac{\partial B_x}{\partial x} \frac{\partial B_x}{\partial y} \frac{\partial B_x}{\partial z} \frac{\partial B_y}{\partial z} \frac{\partial B_z}{\partial z}\right)^T$ определяют обратную задачу восстановления скалярной функции магнитной восприимчивости по данным экспериментальных измерений компонент вектора индукции и независимых компонент тензора градиентов компонент вектора индукции магнитного поля и дополнительной информации о векторной-функции $\boldsymbol{B}^0$, определяющей нормальное поле в области восстановления решения. В этом случае $\mathbf{K} \equiv \mathbf{K}_{MGT}^{B^0}$.
6. $\boldsymbol{X} = \chi$ и $\boldsymbol{B} \equiv \left(B_x\, B_y\, B_z\, \frac{\partial B_x}{\partial x} \frac{\partial B_x}{\partial y} \frac{\partial B_x}{\partial z} \frac{\partial B_y}{\partial z} \frac{\partial B_z}{\partial z}\right)^T$ определяют обратную задачу восстановления скалярной функции магнитной восприимчивости по данным экспериментальных измерений компонент вектора индукции и независимых компонент тензора градиентов компонент вектора индукции магнитного поля и дополнительной информации о векторной-функции $\boldsymbol{B}^0$, определяющей нормальное поле в области восстановления решения. В этом случае $\mathbf{K} \equiv \left(\mathbf{K}_{MI}^{B^0}\, \mathbf{K}_{MGT}^{B^0}\right)^T$.

Выражения для матричных функций $\mathbf{K}_{MI}$ и $\mathbf{K}_{MGT}$ выписаны в подпараграфах 1.1.1 и 1.1.3 главы 1 соответственно. Выражения для матричных функций $\mathbf{K}_{MI}^{B^0}$ и $\mathbf{K}_{MGT}^{B^0}$ могут быть получены за счёт умножения на $\left|\boldsymbol{B}^0\right|$ матричных функций $\mathbf{K}_{MI}^{l}$ и $\mathbf{K}_{MGT}^{l}$, выписанных в подпараграфах 1.1.2 и 1.1.3 главы 1.

Замечание. Необходимо отметить, что выше перечислены те постановки обратных задач магнитометрии, которые могут быть наиболее востребованы на практике. Сознательно опущены постановки, в которых надо восстановить скалярную функцию — модуль $|\boldsymbol{M}|$ вектор-функции $\boldsymbol{M}$ при наличии дополнительной информации о векторном поле $\boldsymbol{l}$, определяющем направления нормального поля $\boldsymbol{B}^0$ (см. подпараграфы 1.1.2 и 1.1.3 главы 1). Как уже отмечалось ранее, такие постановки содержат в три раза меньше неизвестных (так как восстанавливается одна скалярная функция, а не три компоненты векторной функции), но они менее надёжны в связи с тем, что векторное поле $\boldsymbol{l}$ на

практике известно не точно, а значит, в оператор $\mathbf{A}$ уравнения (3.1) вносится дополнительная ошибка. Учёт этой ошибки при выборе параметра регуляризации по обобщённому принципу невязки, в свою очередь, приводит к увеличению параметра регуляризации, и, как следствие, к более сглаженному (а значит, менее детализированному) приближённому решению.

Далее, изложение этой главы будет построено следующим образом. Сначала в параграфе 3.2 показывается, как решаемые в диссертационном исследовании прикладные обратные задачи магнитометрии сводятся к решению больших переопределённых систем линейных алгебраических уравнений с плотно заполненной матрицей. В параграфе 3.2 обсуждается способ учёта ошибок машинного округления при построении критериев прекращения итерационного процесса в градиентных методах решения таких систем. Основной мотивацией работы являлись 1) разработка методов улучшения качества решения реальных прикладных задач минимизации, решение которых требует значительных объёмов вычислений и, как следствие, может быть чувствительно к накапливающимся ошибкам машинного округления, 2) разработка методов, позволяющих по возможности экономить вычислительные ресурсы. В параграфе 3.3 обсуждается вопрос однозначной разрешимости систем линейных алгебраических уравнений, возникающих в результате дискретизации решаемых задач.

3.1 Классические методы решения, основанные на минимизации сглаживающего функционала и приводящие к необходимости решения систем линейных алгебраических уравнений

При решении практических задач вместо точно известных $\boldsymbol{B}$ и оператора $\mathbf{A}$ известны их приближённые значения $\boldsymbol{B}_\delta$ и $\mathbf{A}_h$ такие, что $\|\boldsymbol{B}_\delta - \boldsymbol{B}\|_{L_2} \leqslant \delta$, $\|\mathbf{A} - \mathbf{A}_h\|_{\square\to L_2} \leqslant h$. При выписанных условиях задача определения функции $\boldsymbol{X}$ из уравнения (3.1) является некорректно поставленной и для её решения необходимо строить регуляризирующий алгоритм. Если воспользоваться регуляризирующим алгоритмом, основанным на минимизации функционала А. Н. Тихонова, то задача сводится к поиску элемента, реализующего минимум функционала

$$F^\alpha[\boldsymbol{M}] = \|\mathbf{A}_h\boldsymbol{X} - \boldsymbol{B}_\delta\|^2_{L_2} + \alpha\|\boldsymbol{X}\|^2_{\square}. \tag{3.2}$$

Согласно работе А. Н. Тихонова, А. В. Гочарского, В. В. Степанова и А. Г. Яголы [91] для любого $\alpha > 0$ существует единственная экстремаль $\boldsymbol{X}_\eta^\alpha$, $\eta = \{\delta,h\}$, которая реализует минимум функционала $F^\alpha[\boldsymbol{X}]$.

Параметр регуляризации можно выбрать, например, по обобщённому принципу невязки (см. работу [91]), как корень нелинейного уравнения

$$\rho(\alpha) \equiv \left\|\mathbf{A}_h \boldsymbol{X}_\eta^\alpha - \boldsymbol{B}_\delta\right\|_{L_2}^2 - \left(\delta + h\left\|\boldsymbol{X}_\eta^\alpha\right\|_\square\right)^2 - \mu_\eta^2 = 0.$$

Здесь μ_η — мера несовместности, которая определяется следующим образом:

$$\mu_\eta = \inf_{\boldsymbol{X}} \left\|\mathbf{A}_h \boldsymbol{X} - \boldsymbol{B}_\delta\right\|_{L_2}.$$

При этом $\boldsymbol{X}_\eta^\alpha$ стремиться при $\eta \to 0$ к точному решению задачи.

Элемент $\boldsymbol{X}_\eta^\alpha$, реализующий минимум функционала (3.2), может быть найден по следующей формуле (см., например, работу автора [11]):

$$\boldsymbol{X}_\eta^\alpha = \left(\mathbf{A}_h^* \mathbf{A}_h + \alpha \mathbf{R}^* \mathbf{R}\right)^{-1} \mathbf{A}_h^* \boldsymbol{B}_\delta. \tag{3.3}$$

Здесь оператор $\mathbf{R}$ определяется априорной информацией о решении (принадлежность его пространству $\square$) следующим образом:

$$\left\|\boldsymbol{X}\right\|_\square = \left\|\mathbf{R}\boldsymbol{X}\right\|_{L_2}.$$

Для численного поиска элемента $\boldsymbol{X}_\eta^\alpha$, определяемого уравнением (3.3), необходимо выполнить дискретизацию уравнения (3.1). Общий подход заключается в следующем. Объём V, в котором ищется решение, разбивается на N подобластей, каждой из которых сопоставляются её индекс $\{i\}$, $i = \overline{1,N}$, координаты (x_i, y_i, z_i) расположения этой подобласти и её объём dv_i. В результате уравнение (3.1) может быть переписано в следующем виде:

$$\frac{\mu_0}{4\pi} \sum_{i=1}^{N} \mathbf{K}(x_i, y_i, z_i, x_s, y_s, z_s)\, \boldsymbol{X}(x_i,y_i,z_i)\, dv_i = \boldsymbol{B}(x_s,y_s,z_s).$$

Замечание. Здесь и далее будет предполагаться, что в качестве вектор-функции $\boldsymbol{B}(x_s,y_s,z_s)$ используется вектор-функция, измеренная в эксперименте с ошибкой δ.

Необходимо напомнить, что набор координат (x_s, y_s, z_s) определяет месторасположение в пространстве точки, в которой измеряется индуцированное телом магнитное поле $\boldsymbol{B}(x_s,y_s,z_s)$. Так как измерения проводятся обычно в

большом количестве точек (их число обозначается как S) с известными координатами $(x_{s_j},y_{s_j},z_{s_j})$, $j = \overline{1,S}$, то в результате получается следующая система линейных алгебраических уравнений:

$$\frac{\mu_0}{4\pi}\sum_{i=1}^{N}\mathbf{K}(x_i,y_i,z_i,x_{s_j},y_{s_j},z_{s_j})\,\boldsymbol{X}(x_i,y_i,z_i)\,dv_i = \boldsymbol{B}(x_{s_j},y_{s_j},z_{s_j}),\quad j=\overline{1,S}. \tag{3.4}$$

Систему (3.4) можно переписать в матричной форме записи

$$A\,x = b. \tag{3.5}$$

Матрица A и вектора x и b имеют блочную структуру следующего вида:

$$A = \frac{\mu_0}{4\pi}\begin{bmatrix} \mathbf{K}_{11}\,dv_1 & \mathbf{K}_{12}\,dv_2 & \dots & \mathbf{K}_{1N}\,dv_N \\ \mathbf{K}_{21}\,dv_1 & \mathbf{K}_{22}\,dv_2 & \dots & \mathbf{K}_{2N}\,dv_N \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{K}_{S1}\,dv_1 & \mathbf{K}_{S2}\,dv_2 & \dots & \mathbf{K}_{SN}\,dv_N \end{bmatrix},\quad x = \begin{pmatrix} \boldsymbol{X}_1 \\ \boldsymbol{X}_2 \\ \vdots \\ \boldsymbol{X}_N \end{pmatrix},\quad b = \begin{pmatrix} \boldsymbol{B}_1 \\ \boldsymbol{B}_2 \\ \vdots \\ \boldsymbol{B}_S \end{pmatrix}.$$

Здесь блок $\mathbf{K}_{ji}$ вычисляется для пары точек (x_i,y_i,z_i) и $(x_{s_j},y_{s_j},z_{s_j})$; блок $\boldsymbol{X}_i$ вектора x содержит набор сеточных значений компонент вектор-функции $\boldsymbol{X}$ в точке (x_i,y_i,z_i); блок $\boldsymbol{B}_j$ вектора b содержит набор измеренных в эксперименте в точке $(x_{s_j},y_{s_j},z_{s_j})$ компонент вектор-функции $\boldsymbol{B}$.

Следует привести несколько примеров конкретных формул, определяющих такие блоки.

Пример 1. При решении обратной задачи восстановления вектор-функции намагниченности $\boldsymbol{M}$ по данным экспериментальных измерений компонент вектора индукции магнитного поля $\boldsymbol{B}$ соответствующие блоки будут иметь следующий вид:

$$\mathbf{K}_{ji} = \frac{1}{\rho_{ji}^5}\begin{bmatrix} 3(x_i-x_{s_j})^2-\rho_{ji}^2 & 3(x_i-x_{s_j})(y-y_{s_j}) & 3(x_i-x_{s_j})(z_i-z_{s_j}) \\ 3(y_i-y_{s_j})(x_i-x_{s_j}) & 3(y_i-y_{s_j})^2-\rho_{ji}^2 & 3(y_i-y_{s_j})(z_i-z_{s_j}) \\ 3(z_i-z_{s_j})(x_i-x_{s_j}) & 3(z_i-z_{s_j})(y-y_{s_j}) & 3(z_i-z_{s_j})^2-\rho_{ji}^2 \end{bmatrix},$$

где

$$\rho_{ji} = \sqrt{(x_i-x_{s_j})^2+(y_i-y_{s_j})^2+(z_i-z_{s_j})^2};$$

$$\boldsymbol{X}_i = \begin{bmatrix} M_x(x_i,y_i,z_i) \\ M_y(x_i,y_i,z_i) \\ M_z(x_i,y_i,z_i) \end{bmatrix},\quad \boldsymbol{B}_j = \begin{bmatrix} B_x(x_{s_j},y_{s_j},z_{s_j}) \\ B_y(x_{s_j},y_{s_j},z_{s_j}) \\ B_z(x_{s_j},y_{s_j},z_{s_j}) \end{bmatrix}.$$

Пример 2. При решении обратной задачи восстановления скалярной функции магнитной восприимчивости χ по данным экспериментальных измерений компонент вектора индукции $\boldsymbol{B}$ и независимых компонент тензора градиентов компонент вектора индукции магнитного поля $\boldsymbol{B}_{tensor}$ и дополнительной информации о векторной-функции $\boldsymbol{B}^0 \equiv \left(B_x^0 \;\; B_y^0 \;\; B_z^0\right)^T$, определяющей нормальное поле в области восстановления решения, соответствующие блоки будут иметь следующий вид:

$$\mathbf{K}_{ji} \equiv \frac{1}{\rho_{ji}^7} \left[\begin{array}{c} 3\rho_{ji}^2 c_{ji}(x_i - x_{s_j}) - B_x^0\,\rho_{ji}^4 \\ 3\rho_{ji}^2 c_{ji}(y_i - y_{s_j}) - B_y^0\,\rho_{ji}^4 \\ 3\rho_{ji}^2 c_{ji}(z_i - z_{s_j}) - B_z^0\,\rho_{ji}^4 \\ \hline 6\rho_{ji}^2 B_x^0(x_i - x_{s_j}) + 3\rho_{ji}^2 c_{ji} - 15 c_{ji}(x_i - x_{s_j})^2 \\ 6\rho_{ji}^2 B_z^0(z_i - z_{s_j}) + 3\rho_{ji}^2 c_{ji} - 15 c_{ji}(z_i - z_{s_j})^2 \\ 3\rho_{ji}^2 B_x^0(y_i - y_{s_j}) + 3\rho_{ji}^2 B_y^0(x_i - x_{s_j}) - 15 c_{ji}(x_i - x_{s_j})(y_i - y_{s_j}) \\ 3\rho_{ji}^2 B_x^0(z_i - z_{s_j}) + 3\rho_{ji}^2 B_z^0(x_i - x_{s_j}) - 15 c_{ji}(x_i - x_{s_j})(z_i - z_{s_j}) \\ 3\rho_{ji}^2 B_y^0(z_i - z_{s_j}) + 3\rho_{ji}^2 B_z^0(y_i - y_{s_j}) - 15 c_{ji}(y_i - y_{s_j})(z_i - z_{s_j}) \end{array}\right],$$

где

$$\rho_{ji} = \sqrt{(x_i - x_{s_j})^2 + (y_i - y_{s_j})^2 + (z_i - z_{s_j})^2},$$

$$c_{ji} = B_x^0(x_i,y_i,z_i)\,(x_i - x_{s_j}) + B_y^0(x_i,y_i,z_i)\,(y_i - y_{s_j}) + B_z^0(x_i,y_i,z_i)\,(z_i - z_{s_j});$$

$$\boldsymbol{X}_i = \Big[\chi(x_i, y_i, z_i)\Big], \quad \boldsymbol{B}_j = \left[\begin{array}{c} B_x(x_{s_j},y_{s_j},z_{s_j}) \\ B_y(x_{s_j},y_{s_j},z_{s_j}) \\ B_z(x_{s_j},y_{s_j},z_{s_j}) \\ \frac{\partial B_x}{\partial x}(x_{s_j},y_{s_j},z_{s_j}) \\ \frac{\partial B_x}{\partial y}(x_{s_j},y_{s_j},z_{s_j}) \\ \frac{\partial B_x}{\partial z}(x_{s_j},y_{s_j},z_{s_j}) \\ \frac{\partial B_y}{\partial z}(x_{s_j},y_{s_j},z_{s_j}) \\ \frac{\partial B_z}{\partial z}(x_{s_j},y_{s_j},z_{s_j}) \end{array}\right].$$

В результате численное приближение x элемента $\boldsymbol{X}_\eta^\alpha$, определяемого уравнением (3.3), будет вычисляться формуле:

$$x = \left(A^T A + \alpha R^T R\right)^{-1} A^T b.$$

Матрица R будет являться конечномерной аппроксимацией оператора $\mathbf{R}$, определяемого формулой $\left\|\boldsymbol{X}\right\|_\square = \left\|\mathbf{R}\boldsymbol{X}\right\|_{L_2}$. В случае $\square \equiv L_2$, то есть в отсутствии априорной информации о решении, R будет единичной матрицей. В

других случаях структура этой матрицы усложняется (см. работу [91]), но она всегда будет сильно разреженной матрицей.

Используемые далее обозначения. В текущем изложении, для сокращения записи формул, для обозначения полного числа сенсоров использовалось обозначение S, а для обозначения числа точек, в которых восстанавливаются сеточные значения x вектор-функции $\boldsymbol{X}$, использовалось обозначение N. Далее, для удобства, вместо S будет использоваться обозначение $S_{sensors}$, а вместо N будет использоваться $N_{sources}$. Переменная M будет зарезервирована за числом строк матрицы A, а переменная N будет зарезервировано за числом столбцов матрицы A.

Матрица системы (3.6) будет иметь размеры $m \cdot S_{sensors} \times n \cdot N_{sources}$. Параметр m определяется тем, какого типа обрабатываются магнитные данные: $m = 3$ в случае обработки данных измерения компонент индукции магнитного поля, $m = 5$ в случае обработки данных измерения компонент тензора градиентов компонент индукции магнитного поля, $m = 8$ в случае обработки полных магнито-градиентных данных. Параметр n определяется тем какого типа ищется решение: $n = 3$ в случае восстановления вектор-функции намагниченности, $m = 1$ в случае восстановления скалярной функции магнитной восприимчивости.

Таким образом число строк матрицы системы равно $M = m \cdot S_{sensors}$, а число столбцов — $N = n \cdot N_{sources}$.

3.2 Учёт ошибок машинного округления при решении больших переопределённых систем линейных алгебраических уравнений с плотно заполненной матрицей

Как было показано в предыдущем параграфе, численные методы решения рассматриваемого в диссертации класса задач сводятся к необходимости решения систем линейных алгебраических уравнений вида

$$A\,x = b. \tag{3.6}$$

Здесь, в общем случае, A — прямоугольная плотно заполненная матрица размерности $M \times N$ ($M \geqslant N$), b — вектор-столбец с M компонентами. Целью решения матричного уравнения (3.6) является поиск вектора x с N компонентами.

Для целостности изложения этого параграфа необходимо повторить, что при решении реальных прикладных задач компоненты вектора b, стоящего в правой части матричного уравнения (3.6), обычно измеряются в эксперименте. Поэтому из-за наличия экспериментальных ошибок у этой системы может не существовать классического решения и, более того, решение (псевдорешение, если оно ищется с помощью метода наименьших квадратов) может быть неустойчиво по отношению к ошибкам задания входных данных. В связи с этим для поиска решения необходимо использовать какой-либо регуляризирующий алгоритм. Как было пояснено а предыдущем параграфе, в случае использования регуляризирующего алгоритма, основанного на минимизации функционала А. Н. Тихонова [91], регуляризованное решение системы (3.6) может быть найдено следующим образом:

$$x = \underset{x \in \mathbb{R}^N}{\operatorname{argmin}} f(x), \quad \text{где} \quad f(x) = \|Ax - b\|_2^2 + \alpha \|Rx\|_2^2. \tag{3.7}$$

Здесь α — параметр регуляризации [91], а матрица R содержит информацию об априорных ограничениях на искомое решение x.

Элемент, реализующий минимум функционала (3.7), может быть найден посредством решения системы нормальных уравнений

$$(A^T A + \alpha R^T R)\, x = A^T b. \tag{3.8}$$

Система (3.8) может быть решена с помощью прямых методов решения систем линейных алгебраических уравнений с квадратной матрицей (см., например, работу Дж. В. Деммеля [175]). Но надо отметить, что матрица системы (3.8) имеет размерность $N \times N$. Как следствие, реализация прямых методов решения в случае произвольной матрицы A требует $O(MN^2)$ операций: $(2M-1)N^2 + O(N^1)$ операций, чтобы определить матрицу системы, $(2M-1)N$ операций, чтобы определить правую часть, и $O(N^3)$ операций, чтобы реализовать прямой метод решения системы. Поэтому при больших значениях M и N на качество решения могут критическим образом влиять накапливающиеся в процессе выполнения этих $\sim MN^2$ операций ошибки машинного округления.

В связи с этим, на практике часто ищут элемент, реализующий минимум функционала (3.7), не с помощью решения системы нормальных

уравнений (3.8), а с помощью градиентных методов минимизации самого функционала (3.7). Эти методы являются итерационными: они стартуют с произвольного начального приближения для решения и на каждой итерации находят очередное более хорошее приближение для искомого решения. При вычислениях без ошибок за бесконечное число итераций такие методы сходятся к элементу x, который реализует минимум функционала (3.7).

С учётом того, что задача (3.6) является линейной, наиболее эффективным градиентным методом её решения является метод сопряжённых градиентов (см., например, работы М. Р. Хестенеса [176], Э. Гринбаума [177], С. И. Кабанихина [60]). Этот метод в случае точных вычислений способен найти элемент, реализующий минимум функционала (3.7), не более чем за N итераций, на каждой из которых необходимо выполнить $O(MN)$ операций. Однако с учётом того, что при решении практических задач все вычисления выполняются лишь приближённо (за счёт наличия ошибок машинного округления), утверждение о возможности минимизации функционала ровно за N итераций оказывается неверным. На практике возможны следующие три ситуации.

1. Начиная с какой-то итерации, номер которой меньше N, значение минимизируемого функционала становится сопоставимым с фоном ошибок машинного округления. Это означает, что все последующие итерации бессмысленны и итерационный процесс можно прекратить, так как на последующих итерациях значение функционала не будет убывать. Знание такого номера итерации даёт возможность прервать вычисления и сэкономить вычислительные ресурсы, найдя приближённое решение посредством совершения гораздо меньшего числа операций по сравнению с прямыми методами решения.
2. В первом случае также возможна ситуация, при которой выполнение полного набора из N итераций приводит к «разрушению» численного решения. Таким образом возможность досрочно прервать итерационный процесс может быть особенно полезной при решении прикладных задач. Реализация такой возможности не только экономит на практике вычислительные ресурсы, но ещё и позволяет найти адекватное приближённое решение.
3. Из-за влияния ошибок машинного округления на точность определения направлений минимизации и шагов вдоль них после совершения N итераций значение минимизируемого функционала остаётся достаточно

большим. Это означает, что найденное через N итераций приближённое решение может быть уточнено ещё, если итерационный процесс продолжить. В этом случае итерационный процесс необходимо продолжать до тех пор, пока значение функционала не выйдет на фон ошибок машинного округления.

Другими словами, если на практике используется «классический» критерий остановки итерационного процесса в методе сопряжённых градиентов (по фиксированному числу итераций равному N), то: в первом случае приближённое решение будет найдено, но на его поиск будут затрачены излишние вычислительные ресурсы; во втором случае приближённое решение не будет найдено вовсе; в третьем случае будет найдено очень грубое приближённое решение, которое ещё может быть уточнено.

При решении многих реальных прикладных обратных задач магнитометрии (как двумерных так и трёхмерных) автор регулярно сталкивался с подобными проблемами. Сначала для решения этой проблемы приходилось использовать чисто эмпирические подходы, определяя оптимальное число итераций экспериментально. Но при решении сложных реальных задач такой подход требовал больших вычислительных ресурсов. В результате возникла потребность в разработке метода автоматического определения числа итераций, при котором значение минимизируемого функционала становится сопоставимо с фоном ошибок машинного округления. Поэтому вопрос учёта ошибок машинного округления при выборе критерия прекращения итерационного процесса является актуальным и востребованным на практике.

Таким образом, при решении многих прикладных задач чрезвычайно важным является вопрос о возможности разработки такого критерия прекращения итерационного процесса в методе сопряжённых градиентов, который был бы способен учитывать накапливающиеся в процессе вычислений ошибки машинного округления. Подобным вопросам посвящено множество работ (см., например, работы Х. Возняковски [101], Э. Ф. Каашитера [102], З. Стракова [103; 104], М. Арлиоли [105;106;178], И. Нотайа [107], О. Аксельсона [179], Г. Мерана [108], С.-В. Чанга [180], П. Йиранкка [181], Г. Ланди [182], К. Рао [183], Э. Карсона [184], З. Кулса [185], Э. Гринбаум [186], Б. Т. Поляка [187]). В работе автора [14] был дан достаточно обстоятельный ответ именно на поставленный выше вопрос. В этой работе был сформулирован алгоритм, который на практике позволяет успешно решать все три указанные выше проблемные ситуации.

В этом параграфе будет продемонстрирована методология вывода соответствующих формул на примере применения метода сопряжённых градиентов для решения переопределённой системы линейных алгебраических уравнений с плотно заполненной матрицей. При желании, аналогичные формулы могут быть построены и для решения нелинейных систем. В этом случае можно не ограничиваться методом сопряжённых градиентов, а использовать любой градиентный метод, для которого будет получен свой критерий прекращения итерационного процесса по выходу на фон ошибок машинного округления. Выбор же в качестве примера метода сопряжённых градиентов обоснован тем, что для этого метода есть «классический» критерий прекращения итерационного процесса, с которым можно провести сравнение рассматриваемого подхода.

Отдельно необходимо обратить внимание на то, что учёт ошибок машинного округления особенно актуален сейчас, когда многим исследователям для расчёта реальных «больших» задач доступны суперкомпьютерные системы. Это связано с тем, что использование высокопроизводительных вычислительных систем предоставляет возможность проводить огромные объёмы вычислений. Однако, чем больше вычислений выполняется, тем большая ошибка машинного округления может накопи́ться в течение счёта. А чем большая ошибка накопилась, тем более недостоверный результат можно получить в случае применения критериев остановки итерационного процесса, не учитывающих эту ошибку. Однако, несмотря на это, в этом параграфе будет продемонстрировано, что эти формулы могут давать хороший результат и в случае решения «маленьких» задач.

Структура этого параграфа следующая. В подпараграфе 3.2.1 приводятся формулы одной из возможных реализаций метода сопряжённых градиентов решения системы (3.8). В подпараграфе 3.2.2 демонстрируется вывод формул для критерия прекращения итерационного процесса, учитывающего выход значения минимизируемого функционала на фон ошибок машинного округления. В подпараграфе 3.2.3 обсуждается вычислительная сложность предложенного алгоритма. В подпараграфе 3.2.4 формализуется усовершенствованный итерационный алгоритм, основанный на формулах метода сопряжённых градиентов. В подпараграфе 3.2.5 приводятся примеры расчётов, демонстрирующие эффективность предлагаемого подхода. В подпараграфе 3.2.6 даются рекомендации касательно особенностей работы предложенного подхода при решении «больших» задач.

3.2.1 Метод сопряжённых градиентов

Здесь будет рассмотрена одна из возможных реализаций метода сопряжённых градиентов для решения системы линейных алгебраических уравнений с плотно заполненной матрицей (3.8). При этом необходимо отметить, что способов поиска вектора x существует достаточно много, но был выбран именно тот, который наиболее показателен в рамках данной работы.

Итак, вектор x с N компонентами, который является решением системы (3.8), или, что то же самое, регуляризованным решением системы (3.6), может быть найден с помощью следующего итерационного алгоритма, который строит последовательность $x^{(s)}$. Эта последовательность за N шагов сходится к искомому решению системы (3.8), исходя из предположения, что все вычисления делаются точно.

Сначала необходимо задать $p^{(0)} = 0$, $s = 1$ и произвольное начальное приближение $x^{(1)}$, а затем многократно выполнить следующую последовательность действий:

$$
\begin{aligned}
r^{(s)} &= \begin{cases} A^T\left(A\,x^{(1)} - b\right) + \alpha\, R^T(R\,x^{(1)}), & \text{если} \quad s = 1, \\ r^{(s-1)} - \dfrac{q^{(s-1)}}{\left(p^{(s-1)},q^{(s-1)}\right)}, & \text{если} \quad s \geqslant 2, \end{cases} \\
p^{(s)} &= p^{(s-1)} + \frac{r^{(s)}}{\left(r^{(s)},r^{(s)}\right)}, \\
q^{(s)} &= A^T\left(A\,p^{(s)}\right) + \alpha\, R^T(R\,p^{(s)}), \\
x^{(s+1)} &= x^{(s)} - \frac{p^{(s)}}{\left(p^{(s)},q^{(s)}\right)}, \\
s &= s + 1.
\end{aligned}
$$

Необходимо подчеркнуть, что в данном случае «классический» критерий остановки итерационного процесса формулируется следующим образом: вычисления проводятся до тех пор, пока $s \leqslant N$.

В результате, через N шагов вектор $x^{res} = x^{(N+1)}$ будет расцениваться решением системы (3.8).

Формально, этот итерационный алгоритм с «классическим» критерием прекращения итерационного процесса может быть сформулирован следующим образом:

1. Определить $p^{(0)} = 0$, $s := 1$ и произвольное начальное приближение $x^{(1)}$.
2. Вычислить $r^{(s)} = A^T\big(A\,x^{(s)} - b\big) + \alpha\,R^T\big(R\,x^{(s)}\big)$ и перейти к шагу 4.
3. Вычислить $r^{(s)} = r^{(s-1)} - \dfrac{q^{(s-1)}}{\big(p^{(s-1)},q^{(s-1)}\big)}$.
4. Вычислить $p^{(s)} = p^{(s-1)} + \dfrac{r^{(s)}}{\big(r^{(s)},r^{(s)}\big)}$.
5. Вычислить $q^{(s)} = A^T\big(A\,p^{(s)}\big) + \alpha\,R^T\big(R\,p^{(s)}\big)$.
6. Вычислить $x^{(s+1)} = x^{(s)} - \dfrac{p^{(s)}}{\big(p^{(s)},q^{(s)}\big)}$.
7. Если $s = N$, то остановить итерационный процесс и положить $x^{(s+1)}$ в качестве решения системы (3.8).
8. Переопределить $s := s + 1$ и перейти к шагу 3.

Замечание. Если не использовать рекуррентную форму записи и на каждой итерации вычислять невязку $r^{(s)}$ так же, как и на первой итерации (шаг 2 алгоритма), то количество арифметических операций, требуемых для завершения итерационного процесса, возрастёт в два раза. При решении «больших» задач это является мотивацией использования формы метода сопряжённых градиентов в рекуррентной форме записи.

3.2.2 Подходы к выводу формул усовершенствованного критерия остановки итерационного процесса

Итерационный процесс будет останавливаться на той итерации, на которой норма невязки $r^{(s)}$ перестаёт превышать набегающие при её вычислении ошибки машинного округления. Иными словами, итерационный процесс имеет смысл продолжать, пока верно неравенство

$$\|r^{(s)}\|^2 \geqslant \sigma_s^2\Delta^2.$$

Здесь σ_s^2 — дисперсия ошибки нормы невязки на s-ой итерации в единицах Δ^2, Δ — относительная ошибка машинного округления (при вычислениях с половинной точностью $\Delta = 10^{-3.3}$, при вычислениях с одинарной точностью $\Delta = 10^{-7.6}$, при вычислениях с двойной точностью $\Delta = 10^{-16.3}$, при вычисле-

ниях с четверной точностью $\Delta = 10^{-34.0}$). Далее описывается способ оценки значения σ_s^2.

Сначала выполняется оценка дисперсии ошибки каждого из элементов вектора $r^{(s)}$ на первой итерации ($s = 1$):

$$r_n^{(s)} = \sum_{k=1}^{M} A_{kn} \left(\sum_{l=1}^{N} A_{kl} x_l^{(s)} - b_k \right) + \alpha \sum_{k=1}^{N} R_{kn} \left(\sum_{l=1}^{N} R_{kl} x_l^{(s)} \right).$$

Ошибки от сложения будут оцениваться по правилам статистики, то есть $\sigma_{a+b}^2 = \sigma_a^2 + \sigma_b^2$. Ошибки от умножения достаточно малы по сравнению с ошибками от сложения, поэтому они учитываться не будут.

Тогда

$$\sigma^2_{\sum\limits_l A_{kl} x_l - b_k} = \sum_{l=1}^{N} \left(A_{kl} x_l^{(s)} \right)^2 + b_k^2.$$

При последующем умножении полученные дисперсии ошибок также увеличиваются на соответствующий множитель в квадрате:

$$\sigma^2_{A_{kn}\left(\sum\limits_l A_{kl} x_l - b_k\right)} = A_{kn}^2 \left(\sum_{l=1}^{N} \left(A_{kl} x_l^{(s)} \right)^2 + b_k^2 \right).$$

Затем

$$\sigma^2_{A_{kn}\left(\sum\limits_l A_{kl} x_l - b_k\right) + \alpha R_{kn}\left(\sum\limits_l R_{kl} x_l\right)} = A_{kn}^2 \left(\sum_{l=1}^{N} \left(A_{kl} x_l^{(s)} \right)^2 + b_k^2 \right) + \alpha^2 R_{kn}^2 \sum_{l=1}^{N} \left(R_{kl} x_l^{(s)} \right)^2.$$

Наконец, при суммировании по k получается

$$\sigma^2_{r_n^{(s)}} = \sum_{k=1}^{M} A_{kn}^2 \left(\sum_{l=1}^{N} \left(A_{kl} x_l^{(s)} \right)^2 + b_k^2 \right) + \alpha^2 \sum_{l=1}^{N} R_{kn}^2 \sum_{l=1}^{N} \left(R_{kl} x_l^{(s)} \right)^2.$$

Теперь надо рассмотреть вычисление ошибкок для компонент вектора $r^{(s)}$ на итерациях $s \geqslant 2$. На этих итерациях рекуррентную формулу для вычисления невязки $r^{(s)}$ можно записать в виде

$$r_n^{(s)} = r_n^{(s-1)} - \frac{q_n^{(s-1)}}{\sum\limits_{l=1}^{N} p_l^{(s-1)} q_l^{(s-1)}}.$$

Дисперсию ошибки можно вычислить, используя частные производные по независимым компонентам. Однако учёт всех возможных величин на предыдущих итерациях приведёт к переоценке ошибки. Чтобы этого избежать, учтём

особенность градиентных методов, заключающуюся в том, что градиентные методы являются отчасти устойчивыми к ошибкам машинного округления, потому что на каждой итерации часть ошибок компенсируется. Это связано с тем, что каждую итерацию градиентного метода можно воспринимать как способ поиска очередного более хорошего приближения для искомого решения.

Первый способ вычисления $\sigma^2_{r_n^{(s)}}$. Будет предполагаться, что при подсчёте вектора $p^{(s)}$, который является направлением минимизации, не имеет смысла учитывать все предыдущие ошибки. Таким образом остаются лишь ошибки машинного округления, возникающие при использовании значения $p^{(s)}$ в дальнейших вычислениях. С учётом этого можно получить, что

$$\sigma^2_{r_n^{(s)}} = \sigma^2_{r_n^{(s-1)}} + \sum_{n'=1}^{N} \left(\frac{\partial r_n^{(s)}}{\partial q_{n'}^{(s-1)}} \right)^2 \sigma^2_{q_{n'}^{(s-1)}} =$$

$$= \sigma^2_{r_n^{(s-1)}} + \left(\frac{1}{\sum\limits_{l=1}^{N} p_l^{(s-1)} q_l^{(s-1)}} - \frac{p_n^{(s-1)} q_n^{(s-1)}}{\left(\sum\limits_{l=1}^{N} p_l^{(s-1)} q_l^{(s-1)} \right)^2} \right)^2 \sigma^2_{q_n^{(s-1)}} +$$

$$+ \sum_{n' \neq n} \left(\frac{p_{n'}^{(s-1)} q_{n'}^{(s-1)}}{\left(\sum\limits_{l=1}^{N} p_l^{(s-1)} q_l^{(s-1)} \right)^2} \right)^2 \sigma^2_{q_{n'}^{(s-1)}},$$

Здесь выражения для дисперсий ошибок вычисления компонент вектора $q^{(s-1)}$ считаются уже по известным формулам:

$$\sigma^2_{q_n^{(s-1)}} = \sum_{k=1}^{M} A_{kn}^2 \sum_{l=1}^{N} \left(A_{kl} p_l^{(s-1)} \right)^2.$$

Второй способ вычисления $\sigma^2_{r_n^{(s)}}$. Чтобы значительно упростить (в смысле числа арифметических операций) подсчёт дисперсий ошибок на каждой итерации градиентного метода, можно предполагать, что не имеет смысла учитывать все предыдущие ошибки не только при вычислении вектора $p^{(s)}$, но и при вычислении вектора $q^{(s)}$. С учётом этого получим

$$\sigma^2_{r_n^{(s)}} = \sigma^2_{r_n^{(s-1)}} + \left(\frac{q_n^{(s-1)}}{\sum\limits_{k=1}^{N} p_k^{(s-1)} q_k^{(s-1)}} \right)^2.$$

После вычисления $\sigma^2_{r_n^{(s)}}$ любым из способов дисперсия ошибки нормы невязки σ_s^2 вычисляется как сумма дисперсий компонент, так как они между собой независимы:

$$\sigma_s^2 = \sum_{n=1}^{N} \sigma^2_{r_n^{(s)}}.$$

При выполнении условия

$$\frac{\sigma_s^2 \Delta^2}{\|r^{(s)}\|^2} \geqslant 1$$

итерационный процесс прерывается, а вектор $x^{res} = x^{(s)}$ выбирается в качестве решения системы (3.8).

Это и есть усовершенствованный критерий прекращения итерационного процесса.

3.2.3 Об увеличении вычислительной сложности

Необходимо отметить, что в рассматриваемом градиентном методе на каждой итерации, начиная с $s = 2$, самой вычислительно ёмкой операцией является операция вычисления $q^{(s)}$. Она требует $M(2N-1)$ арифметических операций для вычисления вектора $A\,p^{(s)}$ и $N(2M-1)$ арифметических операций для умножения полученного значения на A^T. Вычислительная сложность вычисления второго слагаемого в $q^{(s)}$ пренебрежимо мала, так как при решении реальных задач матрица R является разреженной, а значит соответствующее число дополнительных арифметических операций составляет $O(N)$. Таким образом суммарное число арифметических операций для вычисления $q^{(s)}$ можно оценить как $4MN = O(MN)$. Остальные операции на каждой итерации градиентного метода вносят тоже вносят пренебрежимо малый вклад в общее число арифметических операций, поэтому вычислительная сложность каждой итерации градиентного метода оценивается как $O(MN)$.

Надо ещё раз подчеркнуть преимущество рекуррентного вычисления невязки $r^{(s)}$ на всех итерациях, начиная с $s \geqslant 2$. Если не использовать рекуррентную форму записи и на каждой итерации вычислять невязку $r^{(s)}$ так же,

как и на первой итерации, то количество арифметических операций, требуемых для завершения итерационного процесса, возрастёт примерно два раза.

Возникает важный вопрос о том, увеличивается ли вычислительная сложность алгоритма при использовании нового критерия.

Первый способ вычисления $\sigma^2_{r_n^{(s)}}$ требует порядка $O(MN)$ дополнительных арифметических операций на каждой итерации рассматриваемого градиентного метода. Можно аккуратно подсчитать все арифметические операции, для этого нужно рассмотреть самую вычислительно ёмкую часть — вычисление компонент $\sigma^2_{q_n^{(s-1)}}$. Эта операция требует $M(3N-1)$ арифметических операций при вычислении элементов внутренней суммы, а затем ещё $N(3M-1)$ арифметических операций при вычислении внешней суммы. Суммарная оценка получается $6MN = O(MN)$ операций, что увеличивает вычислительную трудоёмкость рассматриваемого градиентного метода примерно в 2,5 раза.

Второй способ вычисления $\sigma^2_{r_n^{(s)}}$ требует уже только порядка $O(N)$ дополнительных арифметических операций на каждой итерации. А это означает, что вычислительная трудоёмкость рассматриваемого градиентного метода не изменится.

Применение обоих подходов при решении большого числа прикладных задач показало, что оба способа дают приблизительно одинаковые результаты. В связи с этим, стал использоваться только второй способ, как наиболее экономичный в вычислительном смысле.

Более того, в главе 4 «Комплекс программ» будет показано, что выбранный способ учёта ошибок машинного округления позволяет организовать вычисления на многопроцессорных системах с распределённой памятью таким образом, что все дополнительные операции по реализации усовершенствованного критерия прекращения итерационного процесса будут скрыты на фоне операций основного алгоритма.

Замечание. Формулы, эквивалентные предложенным, можно вывести достаточно большим числом способов. Вполне возможно, что некоторые варианты этих формул будут работать лучше для некоторых конкретных прикладных задач, а для каких-то наоборот хуже. Но при этом важно, чтобы предлагаемые варианты формул не сильно увеличивали вычислительную сложность алгоритма. Это критично для построения эффективных параллельных алгоритмов, что будет рассмотрено в параграфе 4.2 главы 4.

3.2.4 Усовершенствованный итерационный алгоритм

Таким образом, итерационный алгоритм решения системы (3.8) с усовершенствованным критерием остановки итерационного процесса примет следующую форму (красным цветом выделены дополнительные по сравнению с «классическим» итерационным алгоритмом действия).

1. Определить $p^{(0)} = 0$, $s = 1$ и произвольное начальное приближение $x^{(1)}$.
2. Вычислить $r^{(s)} = A^T\left(A\,x^{(s)} - b\right) + \alpha\,R^T(R\,x^{(s)})$.
3. Вычислить $\sigma^2_{r_n^{(s)}} = \sum\limits_{k=1}^{M} A_{kn}^2 \left(\sum\limits_{l=1}^{N}\left(A_{kl}x_l^{(s)}\right)^2 + b_k^2\right) + \alpha^2 \sum\limits_{l=1}^{N} R_{kn}^2 \sum\limits_{l=1}^{N}\left(R_{kl}x_l^{(s)}\right)^2$ для каждого $n \in \overline{1,N}$ и перейти к шагу 6.
4. Вычислить $r^{(s)} = r^{(s-1)} - \dfrac{q^{(s-1)}}{\left(p^{(s-1)},q^{(s-1)}\right)}$.
5. Вычислить $\sigma^2_{r_n^{(s)}} = \sigma^2_{r_n^{(s-1)}} + \left(\dfrac{q_n^{(s-1)}}{\left(p^{(s-1)},\,q^{(s-1)}\right)}\right)^2$ для каждого $n \in \overline{1,N}$.
6. Если $\dfrac{\Delta^2 \sum\limits_{n=1}^{N} \sigma^2_{r_n^{(s)}}}{\|r^{(s)}\|^2} \geqslant 1$, то остановить итерационный процесс и положить $x^{(s)}$ в качестве решения системы (3.8).
7. Вычислить $p^{(s)} = p^{(s-1)} + \dfrac{r^{(s)}}{\left(r^{(s)},r^{(s)}\right)}$.
8. Вычислить $q^{(s)} = A^T\left(A\,p^{(s)}\right) + \alpha\,R^T(R\,p^{(s)})$.
9. Вычислить $x^{(s+1)} = x^{(s)} - \dfrac{p^{(s)}}{\left(p^{(s)},q^{(s)}\right)}$.
10. Переопределить $s := s + 1$ и перейти к шагу 4.

Для удобства последующей программной реализации этого алгоритма, которая будет подробно описана в параграфе 4.2 главы 4 этот алгоритм следует представить в виде псевдокода (см. алг. 3.1). При этом для упрощения формы записи некоторых формул используется обозначение $^{\circ 2}$ — «степень Адамара», то есть поэлементное возведение вектора/матрицы во вторую степень. В этом алгоритме красным цветом выделены строки, которые соответствуют действиям необходимым для реализации усовершенствованного критерия прекращения итерационного процесса. Если эти строки убрать и изменить условие $True$ на $s \leqslant N$, то получится «классическая» реализация метода сопряжённых градиентов для решения системы (3.8).

Алгоритм 3.1: Псевдокод для усовершенствованного итерационного алгоритма.

Входные данные: A, b, $x \equiv x^{(1)}$, α, R

Результат: x

$s \leftarrow 1$

$p \leftarrow 0$

while *True* **do**

  if $s = 1$ **then**

    $r \leftarrow A^T(Ax - b) + \alpha R^T(Rx)$

    $\sigma_r^2 \leftarrow \left(A^T\right)^{\circ 2}\left(A^{\circ 2}x^{\circ 2} + b^{\circ 2}\right) + \alpha^2\left(R^T\right)^{\circ 2}\left(R^{\circ 2}x^{\circ 2}\right)$

  else

    $r \leftarrow r - \dfrac{q}{(p,q)}$

    $\sigma_r^2 \leftarrow \sigma_r^2 + \dfrac{q^{\circ 2}}{(p,q)^2}$

  end

  if $\dfrac{\Delta^2 \sum\limits_n (\sigma_r^2)_n}{(r,r)} \geqslant 1$ **then**

    return x

  end

  $p \leftarrow p + \dfrac{r}{(r,r)}$

  $q \leftarrow A^T(Ap) + \alpha R^T(Rp)$

  $x \leftarrow x - \dfrac{p}{(p,q)}$

  $s \leftarrow s + 1$

end

3.2.5 Примеры численных экспериментов

Для демонстрации возможностей предложенного алгоритма были использованы 1) матрица A размерности $M \times N$ с элементами, сгенерированными как случайные величины с равномерным распределением в диапазоне $[0, 1]$, 2) модельное решение x^{model} — вектор-столбец размерности N, элементы которого

соответствуют значениям синуса на интервале $[0, 2\pi]$:

$$x_n^{model} = \sin \frac{2\pi(n-1)}{N-1}, \quad n \in \overline{1,N}.$$

Для матрицы A и модельного решения x^{model} была вычислена правая часть b: $b = Ax^{model}$.

Для решения системы (3.8) с этими матрицей и правой частью применялась рассматриваемая модификация метода сопряжённых градиентов с усовершенствованным критерием прекращения итерационного процесса. Все вычисления выполнялись с отсутствием регуляризации ($\alpha = 0$).

Замечание. Необходимо отметить, что все последующие результаты могут незначительно отличаться в деталях при их воспроизведении, так как матрица A задаётся случайным образом.

Пример 1. Расчёты проводились для $M = 32$, $N = 30$ и с двойной точностью (т.е. $\Delta \sim 10^{-16}$). На рисунке 3.1 изображен график зависимости значения $\dfrac{\sigma_s^2 \Delta^2}{\|r^{(s)}\|^2}$ от номера итерации s. «Классический» критерий прекращения итерационного сработал бы на итерации с номером $s = N \equiv 30$ (отмечено на рисунке красной пунктирной вертикальной линией), а вот усовершенствованный критерий прекратил итерационный процесс только на итерации с номером $s = 48$, то есть значительно позже. На рисунке 3.2 изображён график зависимости $\|x^{(s)} - x^{model}\|$ от номера итерации s. Из этого графика можно сделать вывод, что «классический» критерий прекращения итерационного процесса даёт решение, достаточно сильно отличающегося от модельного по норме. Таким образом, возникает вопрос о том, как выглядят приближённые решения в случае использования разных критериев прекращения итерационного процесса. На рисунке 3.3 изображено приближённое решение — вектор x^{res}. Прекрасно видно, что приближённое решение найденное «классическим» методом (отмечено на графике красным) достаточно сильно отличается от точного (синуса) даже зрительно. При этом решение, найденное с использование усовершенствованного критерия прекращения итерационного процесса, уже зрительно не отличается от точного решения.

Таким образом, на таком простом примере удалось продемонстрировать работоспособность предложенного алгоритма, хотя при его построении предполагалось, что метод будет работать только при решении «больших» задач.

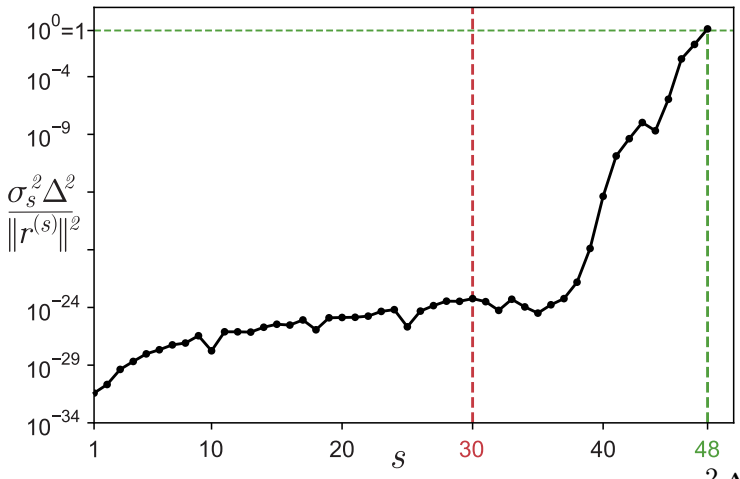


Рисунок 3.1 — Пример 1: график зависимости значения $\dfrac{\sigma_s^2\Delta^2}{\|r^{(s)}\|^2}$ от номера итерации s.

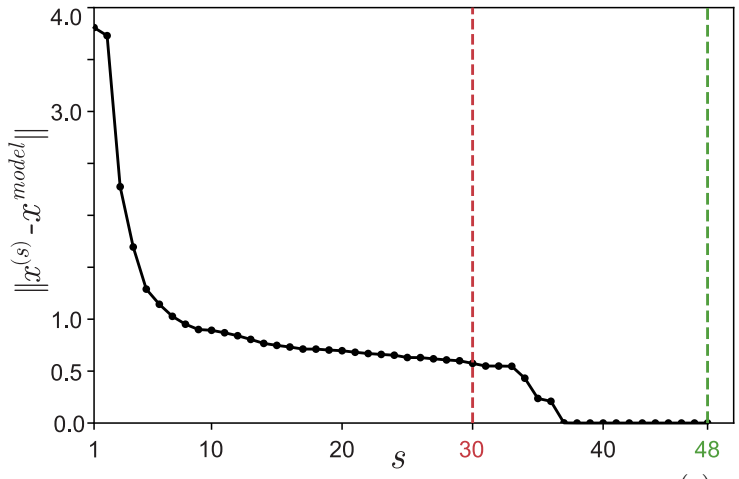


Рисунок 3.2 — Пример 1: график зависимости значения $\|x^{(s)} - x^{model}\|$ от номера итерации s.

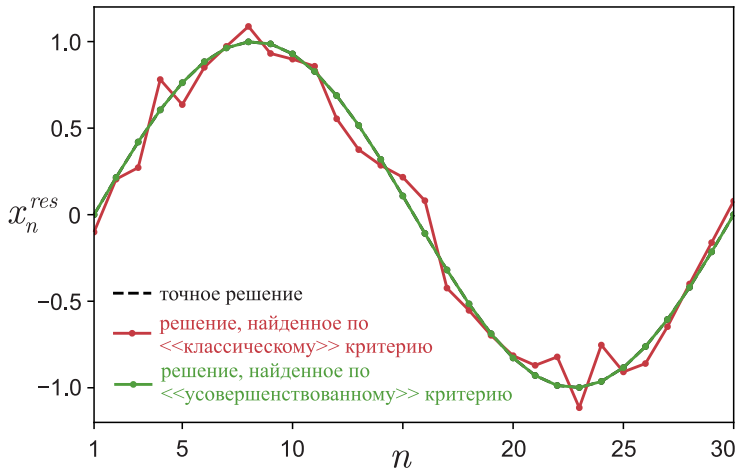


Рисунок 3.3 — Пример 1: график зависимости x_n^{res} от номера компоненты n.

Пример 2. Расчёты проводились для $M = 900$, $N = 30$ и с двойной точностью (т.е. $\Delta \sim 10^{-16}$). Из рисунка 3.4 видно, что «классический» критерий прекращения итерационного сработал бы на итерации с номером $s = N \equiv 30$, а вот усовершенствованный критерий прекратил итерационный процесс раньше, а именно на итерации с номером $s = 23$. Рисунок 3.5 подтверждает, что «классический» критерий прекращения итерационного процесса срабатывает слишком поздно, и много итераций делается впустую. Более того, приближённое решение найденное «классическим» методом не отличимо от решения найденного с помощью усовершенствованного критерия прекращения итерационного процесса — графики наложились друг на друга и не отличимы зрительно (см. рисунок 3.6), что является обоснованием целесообразности досрочного прекращения итерационного процесса.

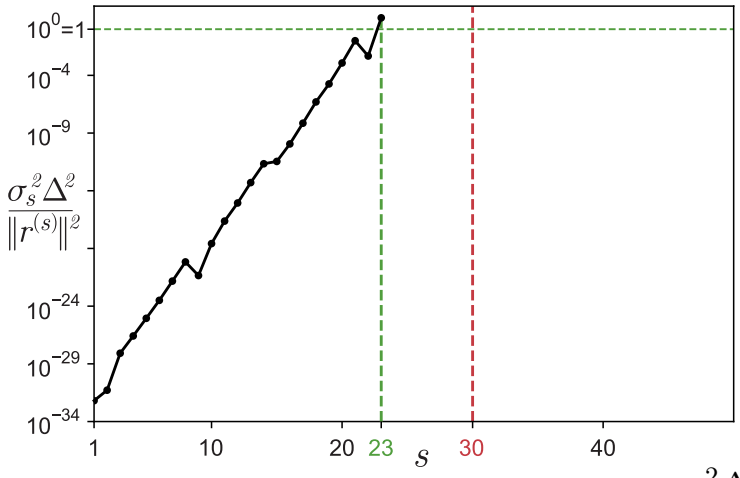


Рисунок 3.4 — Пример 2: график зависимости значения $\dfrac{\sigma_s^2\Delta^2}{\|r^{(s)}\|^2}$ от номера итерации s.

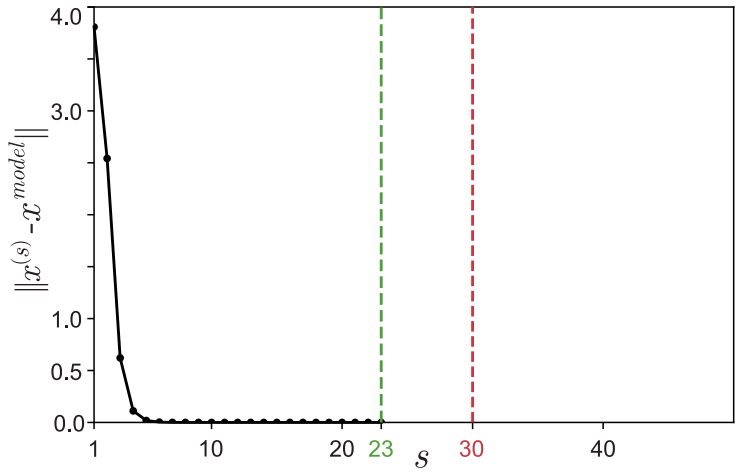


Рисунок 3.5 — Пример 2: график зависимости значения $\|x^{(s)} - x^{model}\|$ от номера итерации s.

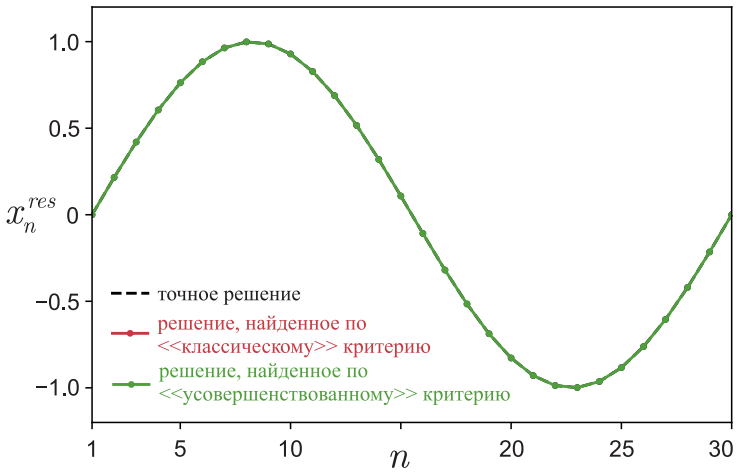


Рисунок 3.6 — Пример 2: график зависимости x_n^{res} от номера компоненты n. Решения для разных критериев прекращения итерационных процессов зрительно не отличимы друг от друга (графики наложились).

Пример 3. Формулы, представленные в этой работе, получены из статистических соображений. Поэтому надёжность предложенного алгоритма является достаточно высокой при решении «больших» задач с высокой точностью вычислений. Однако существуют практические задачи, в которых активно используются вычисления с низкой точностью. Например, в современных реализациях машинного обучения популярны вычисления с пониженной точностью, а именно, вычисления с половинной точностью, для которых $\Delta \sim 10^{-3}$). Такие вычисления преследуют цель обрабатывать большие объёмы данных несмотря на ограниченную видеопамять видеокарт, которые используются для вычислений. Результаты применения предложенного алгоритма для вычислений с такой низкой точностью могут обладать следующими особенностями. С одной стороны, по-прежнему возможны ситуации, в которых предложенный алгоритм даёт хороший результат. Например, на рисунках 3.7 и 3.8, представлены результаты расчётов для $M = 12$, $N = 10$ и половинной точностью вычислений. С другой стороны, низкая точность вычислений приводит к тому, что ошибка округлений становится сопоставимой с невязкой достаточно рано. В результате, это может приводить к срабатыванию критерия прекращения итерационного процесса раньше, чем нужно. Например, на рисунках 3.9 и 3.10 представлены такие результаты расчётов для $M = 120$ и $N = 100$. Невыполнение статистических соображений прекрасно подтверждается немонотонностью кривой определяющей зависимость значения $\dfrac{\sigma_s^2 \Delta^2}{\|r^{(s)}\|^2}$ от номера итерации s (см. рисунок 3.9). Но, с учётом того, что вычисления с низкой точностью используются в приложениях,

где высокая точность не критична, представленный на рисунке 3.10 результат может быть вполне адекватным и отвечать практическим потребностям. При этом необходимо отметить, что решение по «классическому» алгоритму не найдено (на рисунке 3.10 отсутствует соответствующая кривая), в связи с тем, что возник эффект численного переполнения. Использование же усовершенствованного критерия позволило избежать этого эффекта за счёт досрочного прекращения итерационного процесса.

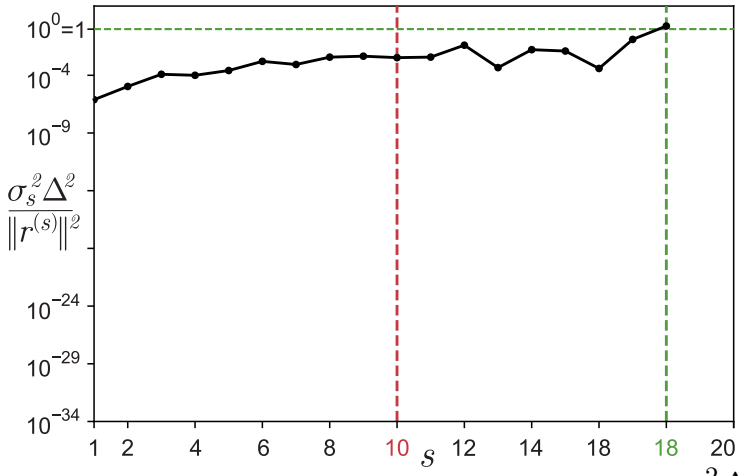


Рисунок 3.7 — Пример 3: график зависимости значения $\dfrac{\sigma_s^2\Delta^2}{\|r^{(s)}\|^2}$ от номера итерации s.

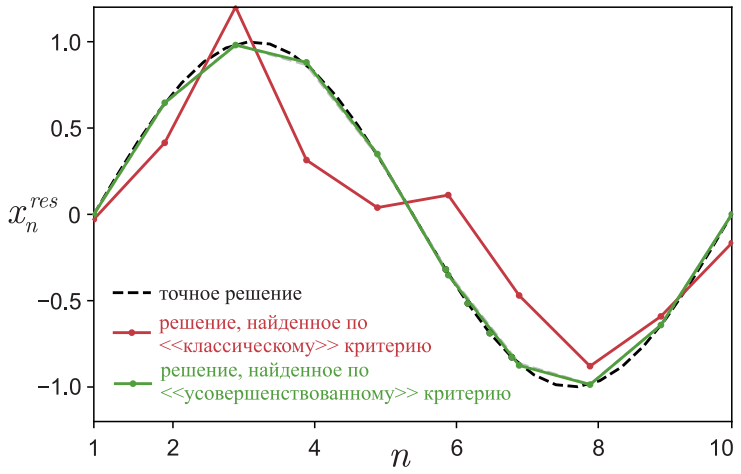


Рисунок 3.8 — Пример 3: график зависимости x_n^{res} от номера компоненты n. Решения для разных критериев прекращения итерационных процессов зрительно не отличимы друг от друга (графики наложились).

Пример 4. Предыдущие примеры демонстрировали работоспособность предложенного алгоритма на предельно маленьких задачах. Сейчас предлагается рассмотреть более репрезентативный пример для $N = 1\,000$. Большее

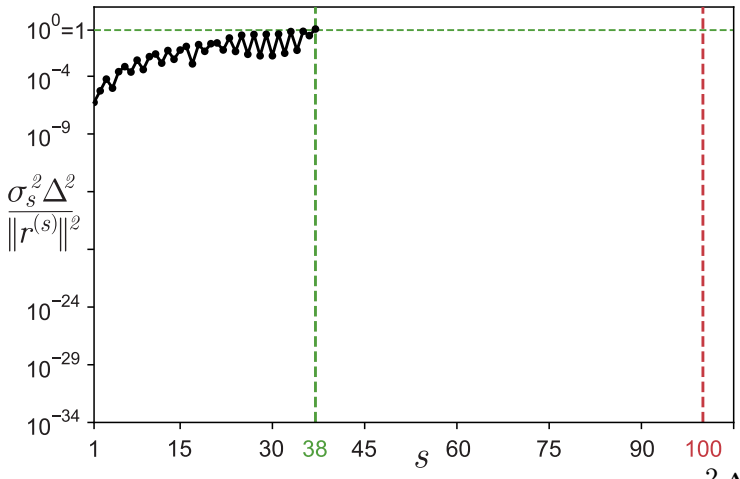


Рисунок 3.9 — Пример 3: график зависимости значения $\dfrac{\sigma_s^2\Delta^2}{\|r^{(s)}\|^2}$ от номера итерации s.

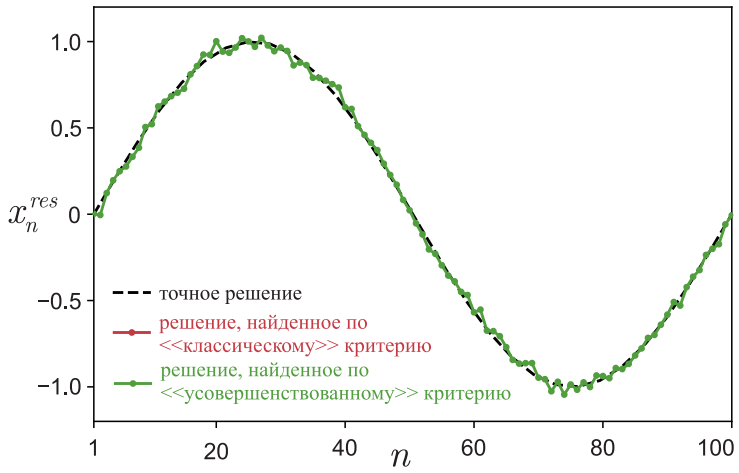


Рисунок 3.10 — Пример 3: график зависимости x_n^{res} от номера компоненты n. Решения для разных критериев прекращения итерационных процессов зрительно не отличимы друг от друга (графики наложились).

число брать для демонстрационных расчётов бесмысленно, так как на графике соседние точки зрительно будут сливаться. На рисунке 3.11 изображено приближённое решение — вектор x^{res} для $M = 1\,000$, $N = 1\,000$. Алгоритму потребовалось совершить $s = 2\,476$ итераций, что существенно больше $s = N \equiv 1\,000$ в случае использования «классической» реализации метода сопряжённых градиентов для решения системы (3.8). Однако, прекрасно видно, что приближённое решение, найденное «классическим» методом (отмечено на графике красным цветом) достаточно сильно отличается от точного (синуса) даже зрительно. При этом решение, найденное с использованием усовершенствованного алгоритма, зрительно не отличается от точного решения.

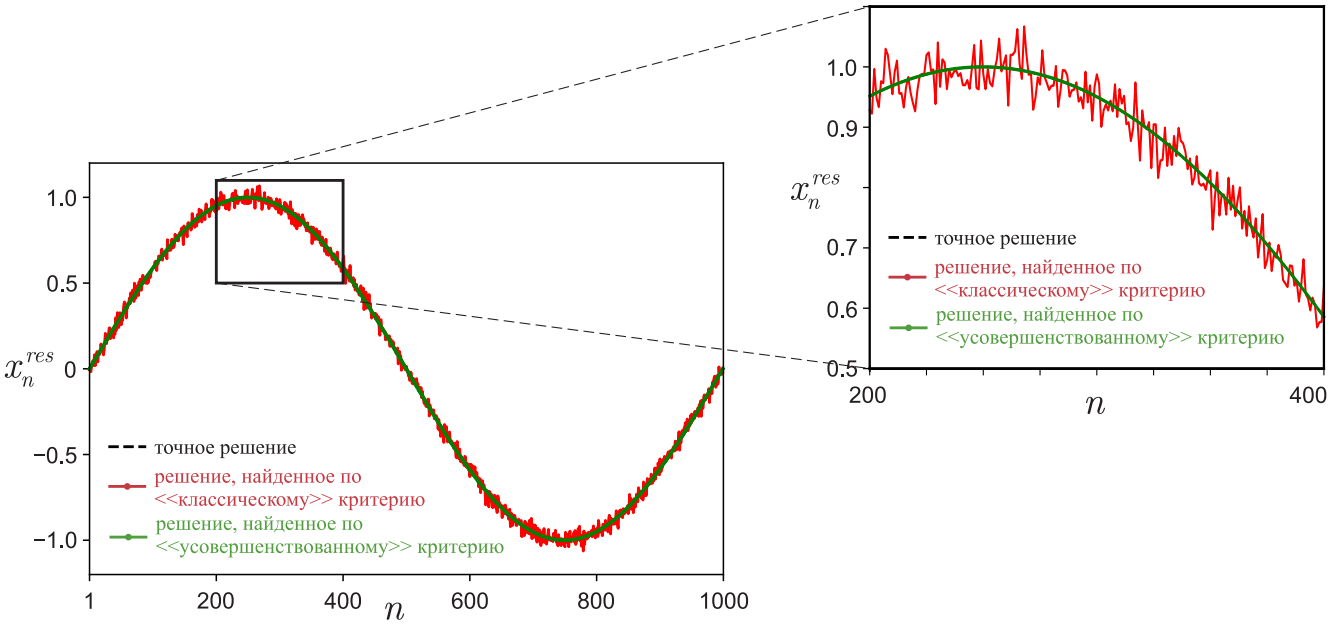


Рисунок 3.11 — Пример 4: график зависимости x_n^{res} от номера компонеты n.

Если же произвести расчёты для $M = 3\,000$, $N = 1\,000$, то приближённое решение, зрительно не отличающееся от точного модельного решения, будет найдено всего-лишь за $s = 75$ итераций, что меньше $1\,000$ итераций, которые потребовались бы, чтобы найти решение с помощью классического алгоритма.

Таким образом этот пример наглядно демонстрирует, что предложенный в работе [14] алгоритм в зависимости от ситуации позволяет как прекратить итерационный процесс досрочно (таким образом сэкономив вычислительные ресурсы), так и продолжить итерационный процесс (таким образом получив более качественное приближённое решение).

3.2.6 Некоторые замечания про расчёты «больших» задач

1. Чем больше численные размеры задачи (M и N), тем лучше работает усовершенствованный критерий прекращения итерационного процесса. Это связано с тем, что соответствующие формулы выводились из статистических соображений, которые лучше соответствуют реальности при обработке больших массивов данных.
2. При использовании регуляризации число итераций, необходимых для выхода на уровень ошибок машинного округления, уменьшается с ростом параметра регуляризации α. Так как параметр регуляризации

принято выбирать по обобщённому принципу невязки [91], согласуя его с погрешностью задания входных данных, то с ростом ошибок задания входных данных увеличивается согласованное с ними значение параметра регуляризации. А это, в свою очередь, приводит к более раннему прекращению итерационного процесса и, как следствие, к потери детализации решения. Этот эффект является вполне логичным, так как с ростом ошибки задания входных данных естественным образом уменьшается и предельно возможная точность восстановления приближённого решения (в случае, если не используется никакая априорная информация о решении).

3. Эффект немонотонности графика зависимости значения $\dfrac{\sigma_s^2\Delta^2}{\|r^{(s)}\|^2}$ от номера итерации s (см., например, рис. 3.9) говорит о нарушении условий применимости подхода: либо численные размеры задачи слишком маленькие, либо недостаточна точность вычислений. На практике следует, по-возможности, проводить вычисления с максимально возможной точностью (т.е. с четверной точностью). При этом с увеличением точности вычислений будет возрастать и значение номера итерации, на которой срабатывает усовершенствованный критерий прекращения итерационного процесса.
4. С увеличением M (при постоянном N) увеличивается и оценка дисперсии невязки. Это, как следствие, приводит к уменьшению значение номера итерации, на котором срабатывает усовершенствованный критерий прекращения итерационного процесса.

3.3 Об однозначной разрешимости СЛАУ, возникающих при решении обратных задач магнитометрии

Как было показано, при решении обратных задач рассматриваемого класса непрерывные постановки дискретизируются и сводятся к системам линейных алгебраических уравнений (СЛАУ) с приближённо заданными как правой частью, так и матрицей. Возникает вопрос разрешимости таких СЛАУ. При этом надо отметить, что разрешимость СЛАУ не гарантирует разрешимости зада-

чи в исходной непрерывной интегральной постановке. В общем случае этот вопрос не решается. Однако можно сделать некоторые допущения, исходя из которых сформулировать условия, при которых в некоторых частных случаях такие СЛАУ будут однозначно разрешимыми. В частности, это удобно сделать из следующего допущения: рассмотреть задачу в дискретной постановке, вместо непрерывной. Таким образом, можно пойти следующим путём.

На практике часто востребована формула для выражения магнитного поля, создаваемого одним магнитным диполем с магнитным моментом $\boldsymbol{m} = (m_x,m_y,m_z)^T$, который расположен в точке с радиус-вектором $\boldsymbol{r}_d = (x_d,y_d,z_d)$. В этом случае выражение для магнитного поля имеет вид

$$\boldsymbol{B}_{field}(\boldsymbol{r}_s) = \frac{\mu_0}{4\pi}\left(\frac{3\big(\boldsymbol{m}(\boldsymbol{r}_d),\boldsymbol{r}_d - \boldsymbol{r}_s\big)(\boldsymbol{r}_d - \boldsymbol{r}_s)}{|\boldsymbol{r}_d - \boldsymbol{r}_s|^5} - \frac{\boldsymbol{m}(\boldsymbol{r}_d)}{|\boldsymbol{r}_d - \boldsymbol{r}_s|^3}\right). \tag{3.9}$$

Здесь: $\boldsymbol{B}_{field}(\boldsymbol{r}_s)$ — вектор-функция, характеризующая индукцию магнитного поля в точке с радиус-вектором $\boldsymbol{r}_s = (x_s,y_s,z_s)$; $\boldsymbol{M}(\boldsymbol{r})$ — вектор-функция, характеризующая плотность магнитного момента элементарного объёма dv в малой окрестности точки $\boldsymbol{r} = (x,y,z)$ области V; μ_0 — магнитная постоянная

Предполагается, что имеется набор условных «сенсоров» s_j, $j = \overline{1,S}$, каждый из которых расположен в точке с координатой $\boldsymbol{r}_{s_j} = (x_{s_j},y_{s_j},z_{s_j})$ и измеряет в этой точке индукцию магнитного поля $\boldsymbol{B}_{field}(\boldsymbol{r}_{s_j}) = \big(B_x^{(s_j)}\ B_y^{(s_j)}\ B_z^{(s_j)}\big)^T$. В каждой точке $\boldsymbol{r}_{s_j}$ поле индуцируется набором магнитных диполей d_i, $i = \overline{1,N}$, каждый из которых расположен в точке с координатой $\boldsymbol{r}_{d_i} = (x_{d_i},y_{d_i},z_{d_i})$ и имеет магнитный момент $\boldsymbol{m}(\boldsymbol{r}_{d_i}) = \big(m_x^{(d_i)}\ m_y^{(d_i)}\ m_z^{(d_i)}\big)^T$.

Для удобства представления последующих формул наравне с указанной выше индексацией будет использоваться следующая (при условиях, что она не будет приводить к противоречиям в обозначениях):

$$\{s_j\}\Big|_{j=\overline{1,S}} \equiv \{s_1, s_2, \ldots, s_S\} \quad \leftrightarrow \quad \{s\}\Big|_{s=\overline{1,S}} \equiv \{1, 2, \ldots, S\},$$
$$\{d_i\}\Big|_{i=\overline{1,N}} \equiv \{d_1, d_2, \ldots, d_N\} \quad \leftrightarrow \quad \{d\}\Big|_{d=\overline{1,N}} \equiv \{1, 2, \ldots, N\}.$$

Таким образом, в точках наблюдения $\boldsymbol{r}_s = (x_s,y_s,z_s)$, $s = \overline{1,S}$, проведены измерения магнитного поля $\boldsymbol{B}_{field}(\boldsymbol{r}_s) \equiv \boldsymbol{B}_{field}^{(s)} = \big(B_x^{(s)}\ B_y^{(s)}\ B_z^{(s)}\big)^T$, $s = \overline{1,S}$, которое индуцировано магнитными диполями $\boldsymbol{m}(\boldsymbol{r}_d) \equiv \boldsymbol{m}^{(d)} = \big(m_x^{(d)}\ m_y^{(d)}\ m_z^{(d)}\big)^T$, $d = \overline{1,N}$, расположенными в точках с координатами $\boldsymbol{r}_d = (x_d,y_d,z_d)$, $d = \overline{1,N}$. В результате возникает вопрос об однозначной разрешимости следующей системы

линейных алгебраических уравнений с целью определения значений магнитных моментов $\boldsymbol{m}(\boldsymbol{r}_d) \equiv \boldsymbol{m}^{(d)} = \left(m_x^{(d)}\ m_y^{(d)}\ m_z^{(d)}\right)^T$, $d = \overline{1,N}$:

$$\begin{cases} \sum\limits_{d=1}^{N} \left(\dfrac{3\left(m_x^{(d)} x_{sd} + m_y^{(d)} y_{sd} + m_z^{(d)} z_{sd}\right) x_{sd}}{r_{sd}^5} - \dfrac{m_x^{(s)}}{r_{sd}^3} \right) = B_x^{(s)}, \quad s = \overline{1,S}, \\ \sum\limits_{d=1}^{N} \left(\dfrac{3\left(m_x^{(d)} x_{sd} + m_y^{(d)} y_{sd} + m_z^{(d)} z_{sd}\right) y_{sd}}{r_{sd}^5} - \dfrac{m_y^{(s)}}{r_{sd}^3} \right) = B_y^{(s)}, \quad s = \overline{1,S}, \\ \sum\limits_{d=1}^{N} \left(\dfrac{3\left(m_x^{(d)} x_{sd} + m_y^{(d)} y_{sd} + m_z^{(d)} z_{sd}\right) z_{sd}}{r_{sd}^5} - \dfrac{m_z^{(s)}}{r_{sd}^3} \right) = B_z^{(s)}, \quad s = \overline{1,S}. \end{cases} \tag{3.10}$$

Здесь через r_{sd} обозначено расстояние между s-ым сенсором и d-ым диполем:

$$r_{sd} = \sqrt{(x_d - x_s)^2 + (y_d - y_s)^2 + (z_d - z_s)^2}.$$

Аналогичный смысл имеют и обозначения x_{sd}, y_{sd} и z_{sd}:

$$x_{sd} = x_d - x_s, \quad y_{sd} = y_d - y_s, \quad z_{sd} = z_d - z_s.$$

Замечание. Необходимо отметить, что система (3.10) возникает также при дискретизации задачи (1.4), которая является интегральным аналогам уравнения (3.9):

$$\boldsymbol{B}_{field}(\boldsymbol{r}_s) = \frac{\mu_0}{4\pi} \iiint\limits_V \left(\frac{3\left(\boldsymbol{M}(\boldsymbol{r}), \boldsymbol{r} - \boldsymbol{r}_s\right)(\boldsymbol{r} - \boldsymbol{r}_s)}{|\boldsymbol{r} - \boldsymbol{r}_s|^5} - \frac{\boldsymbol{M}(\boldsymbol{r})}{|\boldsymbol{r} - \boldsymbol{r}_s|^3} \right) dv.$$

В этом случае $\boldsymbol{m} = \boldsymbol{M}\, dv$, и при дискретизации элементарный объём dv области V заменяется объёмом фиксированной величины, которому сопоставляется соответствующий магнитный диполь.

Систему (3.10) можно переписать в блочной форме записи

$$\begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} & \dots & \mathbf{K}_{1N} \\ \mathbf{K}_{21} & \mathbf{K}_{22} & \dots & \mathbf{K}_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{K}_{S1} & \mathbf{K}_{S2} & \dots & \mathbf{K}_{SN} \end{bmatrix} \times \begin{pmatrix} \boldsymbol{m}^{(1)} \\ \boldsymbol{m}^{(2)} \\ \vdots \\ \boldsymbol{m}^{(N)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{B}^{(1)} \\ \boldsymbol{B}^{(2)} \\ \vdots \\ \boldsymbol{B}^{(S)} \end{pmatrix}. \tag{3.11}$$

Здесь блок $\mathbf{K}_{sd}$ имеет вид

$$\mathbf{K}_{sd} = \frac{1}{r_{sd}^5} \begin{bmatrix} 2x_{sd}^2 - y_{sd}^2 - z_{sd}^2 & 3x_{sd}\, y_{sd} & 3x_{sd}\, z_{sd} \\ 3x_{sd}\, y_{sd} & 2y_{sd}^2 - x_{sd}^2 - z_{sd}^2 & 3y_{sd}\, z_{sd} \\ 3x_{sd}\, z_{sd} & 3y_{sd}\, z_{sd} & 2z_{sd}^2 - x_{sd}^2 - y_{sd}^2 \end{bmatrix}.$$

Таким образом возникает вопрос о формулировке условий, при которых разрешима система линейных алгебраических уравнений вида (3.10) или (3.11).

Системы линейных алгебраических уравнений, к которым редуцируются геофизические задачи, как правило, имеют большую и сверхбольшую размерность (десятки и сотни тысяч, и даже миллионы неизвестных) (см., например, работу А. М. Сальникова [164]). Но в настоящее время, в связи с бурным ростом вычислительных мощностей и развитием суперкомпьютерных технологий, широкое распространение получили методы решения блочных СЛАУ. Поэтому так важно изучение свойств отдельных блоков, из которых состоят огромные матрицы, несущие в себе информацию об источниках поля. Невырожденность блока составной матрицы позволяет повысить устойчивость приближённого решения обратной задачи к ошибкам задания входных данных. В работе А. М. Сальникова [164] было показано, что блочные методы решения СЛАУ, основанные на регуляризации метода разложения матрицы по Холецкому, эффективны именно в случае невырожденности отдельных (в основном, диагональных) блоков. Что касается системы в целом, то пока вопрос о необходимых и достаточных условиях её невырожденности в зависимости от геометрии источников поля остаётся открытым: элементы матриц СЛАУ, возникающих в рамках метода интегральных уравнений не являются ни симметрическими, ни кососимметрическими в общем случае. Если же рассматриваются постановки в рамках метода линейных интегральных представлений, то у матриц начинают проявляться специфические свойства — они являются обязательно симметрическими и положительно полуопределенными и могут быть приведены к виду дважды стохастических матриц (когда суммы всех элементов каждого столбца и каждой строки равны 1) (см. работы автора [4; 17]). Но даже в этом случае весьма сложно при произвольной размерности матрицы судить о её ранге. Поэтому возникает необходимость хотя бы частично сформулировать подходы к частичному решению указанной проблемы.

Далее будут сформулированы теоремы единственность как для частных случаев, так и для достаточно общего случая. Сначала будет рассмотрен простейший случай двух сенсоров и двух магнитных диполей.

Два диполя и два сенсора на одной прямой. Пусть сенсоры s_1, s_2 и диполи d_1, d_2 располагаются на одной прямой (она будет принята за ось Ox неко-

торой декартовой системы координат). В этом случае система (3.11) примет вид

$$\begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix} \times \begin{pmatrix} \boldsymbol{m}^{(1)} \\ \boldsymbol{m}^{(2)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{B}^{(1)} \\ \boldsymbol{B}^{(2)} \end{pmatrix}. \tag{3.12}$$

Выражения для блоков матрицы системы (3.12) выглядят следующим образом:

$$\mathbf{K}_{11} = \begin{bmatrix} \frac{2}{r_{11}^3} & 0 & 0 \\ 0 & -\frac{1}{r_{11}^3} & 0 \\ 0 & 0 & -\frac{1}{r_{11}^3} \end{bmatrix}, \quad \mathbf{K}_{12} = \begin{bmatrix} \frac{2}{r_{12}^3} & 0 & 0 \\ 0 & -\frac{1}{r_{12}^3} & 0 \\ 0 & 0 & -\frac{1}{r_{12}^3} \end{bmatrix},$$

$$\mathbf{K}_{21} = \begin{bmatrix} \frac{2}{r_{21}^3} & 0 & 0 \\ 0 & -\frac{1}{r_{21}^3} & 0 \\ 0 & 0 & -\frac{1}{r_{21}^3} \end{bmatrix}, \quad \mathbf{K}_{22} = \begin{bmatrix} \frac{2}{r_{22}^3} & 0 & 0 \\ 0 & -\frac{1}{r_{22}^3} & 0 \\ 0 & 0 & -\frac{1}{r_{21}^3} \end{bmatrix}.$$

Для удобства следует переобозначить элементы в матрице $\mathbf{K}$ системы (3.12):

$$\begin{bmatrix} a_{11} & 0 & 0 & a_{12} & 0 & 0 \\ 0 & b_{11} & 0 & 0 & b_{12} & 0 \\ 0 & 0 & b_{11} & 0 & 0 & b_{12} \\ a_{21} & 0 & 0 & a_{22} & 0 & 0 \\ 0 & b_{21} & 0 & 0 & b_{22} & 0 \\ 0 & 0 & b_{21} & 0 & 0 & b_{22} \end{bmatrix}. \tag{3.13}$$

Для того чтобы детерминант матрицы (3.13) был равен нулю необходимо и достаточно, чтобы координаты сенсоров совпадали (при различных координатах диполей):

$$\det A = 0 \Rightarrow a_{11}a_{22} - a_{12}a_{21} = (b_{11}b_{22} - b_{12}b_{21})^2 = 0 \Rightarrow$$
$$\det A = 0 \Leftrightarrow b_{11}b_{22} - b_{12}b_{21} = 0 \Leftrightarrow x_{11}x_{22} = x_{12}x_{21} \Leftrightarrow$$
$$(x_{s_1} - x_{d_1})(x_{s_2} - x_{d_2}) = (x_{s_1} - x_{d_2})(x_{s_2} - x_{d_1}) \Rightarrow$$
$$x_{s_1}(x_{d_1} - x_{d_2}) = x_{s_2}(x_{d_1} - x_{d_2}) \Rightarrow (x_{s_1} - x_{s_2})(x_{d_1} - x_{d_2}) = 0.$$

Таким образом, доказана

Теорема 1. Решение системы (3.12) единственно, если два сенсора и два диполя расположены в различных точках на одной и той же прямой и выполняется условие $(x_{s_1} - x_{d_1})(x_{s_2} - x_{d_2}) = (x_{s_1} - x_{d_2})(x_{s_2} - x_{d_1})$, что соответствует

случаю «неразделенных» диполей (см. рис. 3.12-а); если имеет место соотношение $(x_{s_1} - x_{d_1})(x_{s_2} - x_{d_2}) = (x_{s_1} - x_{d_2})(x_{d_1} - x_{s_2})$, то матрица система уравнений может быть вырожденной и компоненты двух диполей определить однозначно нельзя (диполи и сенсоры «разделяют» друг друга, см. рис. 3.12-б).

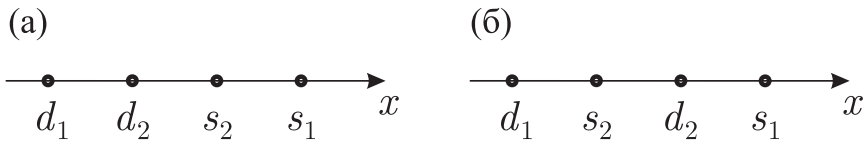


Рисунок 3.12 — Диполи и сенсоры «не разделяют» друг друга — картинка (а); диполи и сенсоры «разделяют» друг друга — картинка (б).

Два диполя и два сенсора, расположенные в одной плоскости. Если через два диполя провести прямую и расположить два сенсора s_1, s_2 на какой-нибудь прямой в плоскости, перпендикулярной прямой, содержащей диполи d_1, d_2 и проходящей через середину соединяющего диполи отрезка (см. рис. 3.13), то система (3.11) примет вид

$$\begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix} \times \begin{pmatrix} \boldsymbol{m}^{(1)} \\ \boldsymbol{m}^{(2)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{B}^{(1)} \\ \boldsymbol{B}^{(2)} \end{pmatrix}. \tag{3.14}$$

Выражения для блоков матрицы (3.14) выглядят следующим образом:

$$\begin{aligned}
\mathbf{K}_{11} &= \frac{1}{r_{11}^5} \begin{bmatrix} 2x_{11}^2 - z_{11}^2 & 0 & 3x_{11}\,z_{11} \\ 0 & -r_{11}^2 & 0 \\ 3x_{11}\,z_{11} & 0 & 2z_{11}^2 - x_{11}^2 \end{bmatrix}, \\
\mathbf{K}_{12} &= \frac{1}{r_{sd}^5} \begin{bmatrix} 2x_{12}^2 - z_{12}^2 & 0 & 3x_{12}\,z_{12} \\ 0 & -r_{11}^2 & 0 \\ 3x_{12}\,z_{12} & 0 & 2z_{12}^2 - x_{12}^2 \end{bmatrix}, \\
\mathbf{K}_{21} &= \frac{1}{r_{21}^5} \begin{bmatrix} 2x_{21}^2 - z_{21}^2 & 0 & 3x_{21}\,z_{21} \\ 0 & -r_{11}^2 & 0 \\ 3x_{21}\,z_{21} & 0 & 2z_{21}^2 - x_{21}^2 \end{bmatrix}, \\
\mathbf{K}_{22} &= \frac{1}{r_{22}^5} \begin{bmatrix} 2x_{22}^2 - z_{22}^2 & 0 & 3x_{22}\,z_{22} \\ 0 & -r_{11}^2 & 0 \\ 3x_{22}\,z_{22} & 0 & 2z_{22}^2 - x_{22}^2 \end{bmatrix}.
\end{aligned} \tag{3.15}$$

Расстояния от расположенного таким образом сенсора до каждого из двух диполей будут одинаковыми.

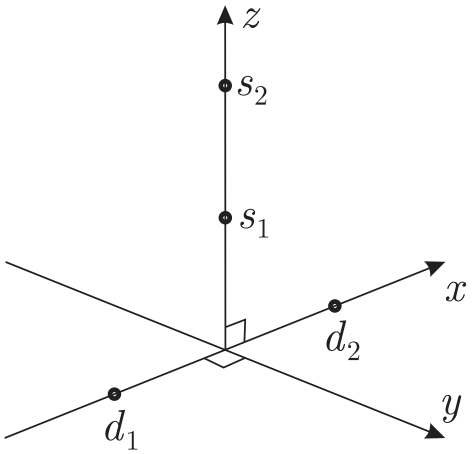


Рисунок 3.13 — Второй случай взаимного расположения сенсоров и диполей в трёхмерном пространстве.

С учётом того, что $y_{1d} = y_{2d} = 0$, $d = \overline{1,2}$, и $r_{s1} = r_{s2}$, $s = \overline{1,2}$, получается, что в матрице системы (3.14) две строки являются пропорциональными:

$$\begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix} \equiv \begin{bmatrix} \square & \square & \square & \square & \square & \square \\ 0 & -\frac{1}{r_{11}^3} & 0 & 0 & -\frac{1}{r_{12}^3} & 0 \\ \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square \\ 0 & -\frac{1}{r_{21}^3} & 0 & 0 & -\frac{1}{r_{22}^3} & 0 \\ \square & \square & \square & \square & \square & \square \end{bmatrix}.$$

Таким образом однозначной разрешимости системы (3.14) нет.

Аналогичный результат можно получить, если расположить два сенсора на одном расстоянии от прямой, соединяющей два диполя (при этом не требуется, чтобы расстояния от сенсора до каждого из двух диполей были одинаковыми). Сенсоры s_1, s_2 и диполи d_1, d_2 на рис. 3.13 при этом меняются местами. Квазирешение в этом случае также определяется неоднозначно.

Таким образом, верна

Теорема 2. Псевдорешение системы (3.14) определяется неоднозначно, если имеют место следующие два случая взаимного расположения диполей и сенсоров: а) два диполя и N сенсоров расположены в одной плоскости, при этом сенсоры лежат на прямой, проходящей через середину отрезка, связывающего диполи, и перпендикулярной им; б) два сенсора и N диполей лежат в одной плоскости, причем диполи располагаются на прямой, перпендикулярной отрезку, соединяющему два сенсора, и проходящей через центр этого отрезка.

Два диполя и N сенсоров, расположенных на прямой, перпендикулярной отрезку, соединяющему диполи и проходящей через его середину Если предположить, что измерения компонент магнитной индукции выполняются в точках прямой, проходящей через середину отрезка, соединяющего два диполя, и перпендикулярной этому отрезку. Тогда система (3.10) примет вид

$$\begin{cases} \sum\limits_{d=1}^{2} \left(\dfrac{3\left(m_x^{(d)} x_{sd} + m_y^{(d)} y_{sd} + m_z^{(d)} z_{sd}\right) x_{sd}}{r_{sd}^5} - \dfrac{m_x^{(s)}}{r_{sd}^3} \right) = B_x^{(s)}, \quad s = \overline{1,N}, \\ \sum\limits_{d=1}^{2} \left(\dfrac{3\left(m_x^{(d)} x_{sd} + m_y^{(d)} y_{sd} + m_z^{(d)} z_{sd}\right) y_{sd}}{r_{sd}^5} - \dfrac{m_y^{(s)}}{r_{sd}^3} \right) = B_y^{(s)}, \quad s = \overline{1,N}, \\ \sum\limits_{d=1}^{2} \left(\dfrac{3\left(m_x^{(d)} x_{sd} + m_y^{(d)} y_{sd} + m_z^{(d)} z_{sd}\right) z_{sd}}{r_{sd}^5} - \dfrac{m_z^{(s)}}{r_{sd}^3} \right) = B_z^{(s)}, \quad s = \overline{1,N}. \end{cases} \tag{3.16}$$

Очевидно, что система (3.16) является переопределённой системой линейных алгебраических уравнений, которая, в общем случае, несовместна. Как известно (см. работы автора [8;12]), в конечномерном случае всегда существует псевдорешение системы, которое может быть получено путём решения нормальной системы уравнений $\mathbf{K}^T\mathbf{K}\,\boldsymbol{m} = \mathbf{K}^T\boldsymbol{B}$ (здесь $\mathbf{K}$ — матрица исходной системы, а $\boldsymbol{B}$ — правая часть исходной системы). Поскольку ранг $\mathbf{K}^T\mathbf{K}$ равен рангу исходной матрицы $\mathbf{K}$, то при описанном выше расположении диполей и сенсоров будет получена система линейных алгебраических уравнений с пропорциональными друг другу строками в каждом блоке (с учётом того, что $r_{s1} = r_{s2}$ для всех $s = \overline{1,N}$). Такие строки имеют вид

$$\left[0 \quad -\tfrac{1}{r_{s1}^3} \quad 0 \quad 0 \quad -\tfrac{1}{r_{s2}^3} \quad 0\right].$$

Таким образом, доказана

Теорема 3. Если все сенсоры расположены на прямой, лежащей в плоскости симметрии двух неизвестных диполей d_1, d_2, то квазирешение системы (3.16) определяется неоднозначно. Однако нормальное псевдорешение будет единственно.

Два диполя и N сенсоров, расположенных на одной прямой Для данного случая верен аналог **Теоремы 1**:

Теорема 4. Если два диполя и N сенсоров расположены на одной прямой, координаты двух каких-либо сенсоров не совпадают и эти два сенсора

“не разделяют” выделенные два диполя, то квазирешение системы (3.10) единственно, независимо от того, какие значения принимают координаты остальных $N-2$ сенсоров.

Доказательство. Система уравнений, из которой в описанном случае определяются компоненты двух диполей, лежащих на одной прямой с сенсорами, имеет вид (3.16), первые две пары блоков которой выражаются с помощью формулы (3.13). Ранг такой системы не может быть больше 6.

Рассмотрим два диполя и два сенсора, которые оба расположены либо правее, либо левее обоих диполей. Согласно Теореме 1, компоненты диполей определяются однозначно, и, следовательно, остальные сенсоры не добавят информации о магнитном поле.

N диполей и N сенсоров в трёхмерном пространстве Рассмотрим теперь случай, в котором требуется определить компоненты N магнитных диполей d_i, $i = \overline{1,N}$ по измеренным в N произвольных точках трёхмерного пространства трём компонентам вектора магнитной индукции. Исследование, аналогичное проведённому в предыдущих подразделах, позволяет сделать вывод о том, что решение системы (3.10) будет неединственным, если для каких-либо двух индексов s_1 и s_2 справедливы следующие соотношения:

$$\begin{aligned} y_{s_1 d} = y_{s_2 d} = 0, &\quad d = \overline{1,N}, \\ r_{s_1 d} = r_{s_2 d}, &\quad d = \overline{1,N}. \end{aligned} \tag{3.17}$$

Тогда две строки в (3.11) будут совпадать: в s_1-ом блоке

$$\left[0 \quad -\frac{1}{r_{s_1 1}^3} \quad 0 \quad 0 \quad -\frac{1}{r_{s_1 2}^3} \quad 0 \quad \dots \quad 0 \quad -\frac{1}{r_{s_1 N}^3} \quad 0\right]$$

и в s_2-ом блоке

$$\left[0 \quad -\frac{1}{r_{s_2 1}^3} \quad 0 \quad 0 \quad -\frac{1}{r_{s_2 2}^3} \quad 0 \quad \dots \quad 0 \quad -\frac{1}{r_{s_2 N}^3} \quad 0\right].$$

Таким образом, доказана

Теорема 5. При выполнении условий (3.17) решение системы (3.10) определяется неоднозначно.

Глава 4. Комплекс программ

При решении рассматриваемого в диссертации класса прикладных задач возникает необходимость в решении систем линейных алгебраических уравнений вида

$$A\,x = b. \tag{4.1}$$

Здесь, в общем случае, A — прямоугольная плотно заполненная матрица размерности $M \times N$ ($M \geqslant N$), b — вектор-столбец с M компонентами. Целью решения матричного уравнения (4.1) является поиск вектора x с N компонентами.

При решении реальных прикладных задач компоненты вектора b, стоящего в правой части матричного уравнения (4.1), обычно измеряются в эксперименте. Поэтому из-за наличия экспериментальных ошибок у этой системы может не существовать классического решения и, более того, решение (псевдорешение, если оно ищется с помощью метода наименьших квадратов) может быть неустойчиво по отношению к ошибкам задания входных данных. В связи с этим для поиска решения необходимо использовать какой-либо регуляризирующий алгоритм.

Будет рассматриваться алгоритм, основанный на минимизации функционала А. Н. Тихонова [91]. Регуляризованное решение системы (4.1) может быть найдено как элемент, реализующий минимум функционала А. Н. Тихонова [91]:

$$x = \underset{x \in \mathbb{R}^N}{\operatorname{argmin}} f(x), \quad \text{где} \quad f(x) = \|Ax - b\|_2^2 + \alpha \|Rx\|_2^2. \tag{4.2}$$

Здесь α — параметр регуляризации [91], а матрица R содержит информацию об априорных ограничениях на искомое решение x.

Для простоты изложения материала в этой главе будет полагаться, что $R \equiv E$, где E — единичная матрица. В этом случае элемент, реализующий минимум функционала (4.2), может быть найден посредством решения системы нормальных уравнений

$$(A^T A + \alpha E)\, x = A^T b. \tag{4.3}$$

Система (4.3) может быть решена с помощью прямых методов решения систем линейных алгебраических уравнений с квадратной матрицей, но,

как было отмечено в параграфе 3.2 главы 3, более конструктивным подходом для решения задач рассматриваемого в диссертации класса является использование градиентных методов минимизации функционала (4.2), так как они зачастую позволяют значительным образом сэкономить вычислительные ресурсы по сравнению с прямыми методами решения. В частности, в параграфе 3.2 главы 3 была рассмотрена усовершенствованная версия метода сопряжённых градиентов, в которой был реализован критерий прекращения итерационного процесса, учитывающий накапливающие в процессе вычислений ошибки машинного округления. Соответствующий метод был сформулирован в виде алгоритма 3.1.

Теперь осталось рассмотреть вопрос о путях эффективной программной реализации предложенного алгоритма. Этот вопрос является чрезвычайно важным. Это связано с тем, что для решения многих реальных прикладных задач необходимо применять суперкомпьютерные (кластерные) системы со множеством вычислительных узлов. Если такая суперкомпьютерная система представляет из себя многопроцессорную систему с распределённой памятью, то для организации взаимодействия различных вычислительных узлов обычно используют технологию передачи сообщений MPI (Message Passing Interface). При этом на эффективность параллельной программной реализации влияют накладные расходы по взаимодействию вычислительных узлов посредством использования коммуникационной сети и передачи сообщений через неё. В рассмотренном в параграфе 3.2 алгоритме по сравнению с «классическим» алгоритмом добавлено множество дополнительных относительно мелких вычислений, выполнение которых на множестве вычислительных узлов может порождать существенное увеличение накладных расходов по взаимодействию этих вычислительных узлов между собой [113–115]. То есть возможно возникновение достаточно распространённой при решении реальных прикладных задач проблемы: алгоритм обоснован и выглядит «хорошим», однако его применение для решения реальных прикладных задач является неэффективным. Однако предложенный алгоритм был создан таким образом, чтобы он позволял достаточно эффективную параллельную программную реализацию. Поэтому в работе [16] автором было продемонстрировано то, как предложенный в работе [14] алгоритм может быть эффективно распараллелен с учётом использования технологии параллельного программирования MPI и с использованием технологии гибридного параллельного программирования

MPI+OpenMP+CUDA. Таким образом, новизна этой части работы автора заключается в разработке эффективной параллельной программной реализации алгоритма решения больших переопределённых систем линейных алгебраических уравнений с плотно заполненной матрицей с использованием преимуществ современных технологий параллельного программирования (в частности, стандарта MPI-4.0 2021-го года).

Более того, указанные подходы необходимо применить для построения эффективной программной реализации численных методов восстановления параметров намагниченности объекта по данным измерения компонент вектора индукции или тензора градиентов компонент индукции магнитного поля с помощью языков программирования Python и Fortran и технологий параллельного программирования MPI, OpenMP и CUDA. Особенностям построения соответствующего программного комплекса, способного быть использованным на сложных вычислительных системах, посвящена эта глава.

Структура этой главы следующая. В параграфе 4.1 описываются особенности подготовки на вычислительных узлах многопроцессорной системы данных для расчёта обратной задачи магнитометрии с использованием технологии параллельного программирования MPI. В частности, в подпараграфе 4.1.1 подробно рассматривается вопрос первичной обработки входных данных обратной задачи и подготовке соответствующих данных на вычислительных узлах с помощью технологии MPI. Затем в подпараграфе 4.1.2 описываются особенности параллельной подготовки на каждом вычислительном узле своей части матрицы решаемой системы. Параграф 4.2 посвящён вычислительному ядру программного комплекса, а именно параллельной реализации с помощью технологии MPI алгоритма решения переопределённых систем линейных алгебраических уравнений с плотно заполненной матрицей с учётом ошибок округления. Для полноты изложения соответствующего материала, сначала в подпараграфе 4.2.1 приводится описание последовательного алгоритма и его программная реализация. В подпараграфе 4.2.2 описываются параллельные алгоритмы реализации некоторых базовых операций линейной алгебры, которые будут использоваться для построения параллельной реализации рассматриваемого алгоритма. В подпараграфе 4.2.3 описываются различные подходы к построению параллельной реализации алгоритма и его программной реализации с учётом возможностей различных стандартов MPI (MPI-2, MPI-3 и MPI-4). В подпараграфе 4.2.4 проводится исследование предложенных программных

реализаций на наличие сильной и слабой масштабируемости, а также даются рекомендации по их использованию при вычислениях на больших параллельных системах. Все рассмотренные в параграфе 4.2 примеры программ, реализующих рассмотренные алгоритмы, написаны с использованием языка программирования Python, что связано в первую очередь с относительной компактностью соответствующих программных реализаций. Однако все программы построены таким образом, что они могут быть достаточно легко переписаны на языки программирования C/C++/Fortran. В связи с этим в параграфе 4.3 даются комментарии касательно переносимости предложенных программных реализаций на языки программирования C/C++/Fortran. В параграфе 4.4 описываются программные особенности использования на вычислительных узлах многоядерных процессоров с помощью технологии OpenMP (подпараграф 4.4.2) и графических процессоров с помощью технологии параллельного программирования CUDA (подпараграф 4.4.3). Демонстрируется то, как возможности языка программирования Python позволяют достаточно просто использовать соответствующие технические решения в рамках рассматриваемого класса прикладных задач. Для этого в подпараграфе 4.4.1 предварительно рассматривается упрощённый (но конструктивный) пример программной реализации для некоторых задач рассматриваемого класса. В параграфе 4.5 даются рекомендации касательно учёта априорной информации о решении, которая может быть заложена посредством задания неединичной матрицы R в (4.2). Наконец, в параграфе 4.6 подытоживаются преимущества и недостатки программных реализаций с использованием языков программирования Python и Fortran, которые использовались автором при создании программного комплекса решения обратных задач магнитометрии.

4.1 Программные особенности подготовки на вычислительных узлах данных для расчёта обратной задачи магнитометрии с использованием технологии параллельного программирования MPI

Предполагается, что программная реализация разрабатываемого параллельного алгоритма решения СЛАУ (4.3) будет использовать технологию параллельного программирования MPI (Message Passing Interface) и запускать-

ся на числе MPI-процессов равном `numprocs`. Все эти процессы будут входить в коммуникатор `comm` $\equiv$ `MPI.COMM_WORLD`, внутри которого будут иметь свой номер/идентификатор `rank` $\in \overline{0, \texttt{numprocs} - 1}$.

Все элементы матрицы A должны быть распределены по блокам между участвующими в вычислениях процессами, используя двумерное деление матрицы на блоки (см. рис. 4.1).

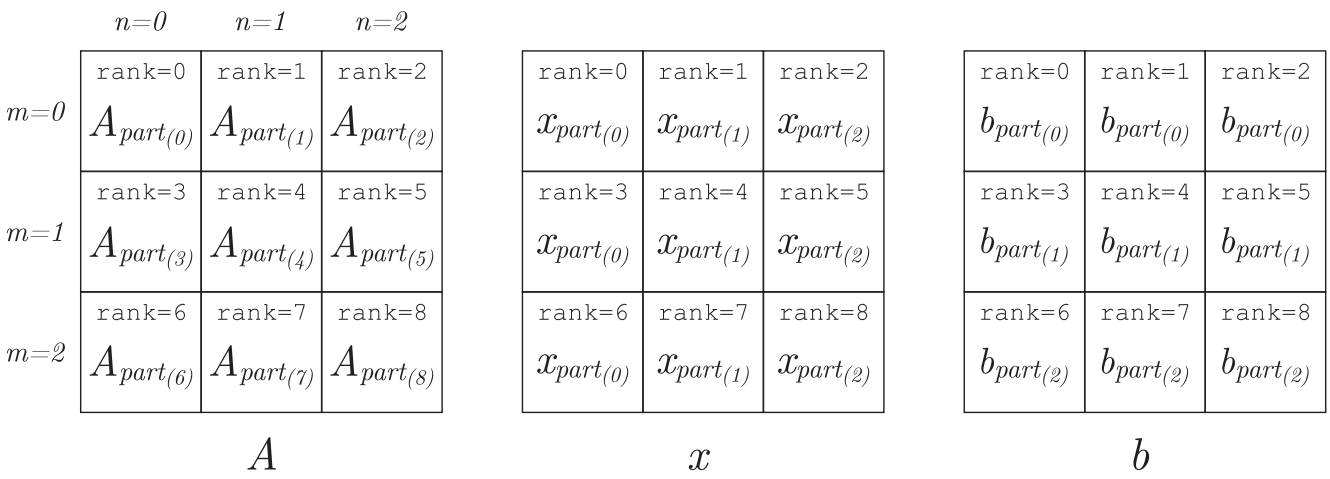


Рисунок 4.1 — Пример распределения данных по 9 MPI-процессам, которые образуют сетку 3×3.

При этом число разбиений вдоль вертикали будет обозначаться как `num_row` (сокращение от *number of rows*), а число разбиений вдоль горизонтали — `num_col` (сокращение от *number of columns*). В результате матрица A размерности $M \times N$ будет разбита на части $A_{part_{(\texttt{rank})}}$ размерности $M_{part_{(m)}} \times N_{part_{(n)}}$. Здесь номер процесса `rank` связан с индексами m и n следующим образом:

$$m = \left\lfloor \frac{\texttt{rank}}{\texttt{num\_col}} \right\rfloor, \quad n = \texttt{rank} - \left\lfloor \frac{\texttt{rank}}{\texttt{num\_col}} \right\rfloor \cdot \texttt{num\_col}.$$

Замечание. Необходимо обратить внимание на то, что выбранный способ индексации предполагает, что в вычислениях будут принимать участие все процессы, то есть `num_row` $\cdot$ `num_col` $\equiv$ `numprocs`.

При этом

$$\sum_{m=0}^{\texttt{num\_row}-1} M_{part_{(m)}} = M, \qquad \sum_{n=0}^{\texttt{num\_col}-1} N_{part_{(n)}} = N.$$

Таким образом, предполагается, что на каждом MPI-процессе должен быть сформирован массив `A_part`, который содержит одну из частей $A_{part_{(\cdot)}}$ матрицы A. Также на каждом процессе должны содержатся массивы `x_part` и

`b_part`, которые содержат одну из частей $x_{part_{(\cdot)}}$ и $b_{part_{(\cdot)}}$ векторов x и b соответственно (здесь x — начальное приближение для итерационного алгоритма). Структура хранения векторов x и b на различных процессах также показана на рис. 4.1. Видно, что часть вектора $x_{part_{(n)}}$ для фиксированного индекса n будет хранится во всех ячейках столбца сетки процессов с индексом n. Аналогично, часть вектора $b_{part_{(m)}}$ для фиксированного индекса m будет хранится во всех ячейках строки сетки процессов с индексом m. Такая структура хранения векторов x и b на различных процессах обусловлена особенностью программной реализации параллельного алгоритма решения СЛАУ (4.3), который будет рассмотрен в подпараграфе 4.2.2.

4.1.1 Подготовка на вычислительных узлах вспомогательных данных

В этом подпараграфе описывается программная реализация всех вспомогательных действий, которые помогут сформировать на каждом MPI-процессе необходимые для вычислений данные (части матрицы A и векторов x и b).

Программа начинается стандартным образом: импортируются необходимые библиотеки и функции, а затем определяется количество процессов `numprocs` в коммуникаторе `comm` $\equiv$ `MPI.COMM_WORLD`, на которых запущена программа.

```
1 from mpi4py import MPI
2 from numpy import empty, array, zeros, int32, float64, size, sqrt,
     fromfile, reshape, pi
3 #from module import *
4
5 comm = MPI.COMM_WORLD
6 numprocs = comm.Get_size()
```

Необходимо обратить внимание на закомментированную строку 3. В этой программе будут использоваться две свои собственные функции. Одна будет подготавливать вспомогательные массивы, а другая будет реализовывать метод сопряжённых градиентов. Эти функции можно, по желанию, включить как в основной листинг программы, так и вынести их в отдельный модуль. В последнем случае необходимо раскомментировать указанную строку, предварительно

создав файл module.py и поместив в него необходимые функции. При этом здесь не определяется идентификатор (номер/ранг) процесса `rank` в коммуникаторе `comm`. Он будет не нужен, так как все операции будут совершаться с идентификатором MPI-процесса в новом коммуникаторе `comm_cart`, который будет создан далее.

Затем задаются значения `num_row` и `num_col`, которые определяют число строк и число столбцов сетки процессов.

```
7 num_row = num_col = int32(sqrt(numprocs))
```

Выбранный способ определения значений `num_row` и `num_col` предполагает, что программа может запускаться только на числе процессов `numprocs`, которое является квадратом натурального числа (1, 4, 9, 16, 25, 36, 49 и т.д.). Эта особенность не ограничивает общность дальнейшей программной реализации — способ определения значений `num_row` и `num_col` может быть произвольным при условии, что $\texttt{num\_row} \cdot \texttt{num\_col} \equiv \texttt{numprocs}$.

Затем необходимо создать виртуальную топологию `comm_cart` типа двумерного тора.

```
8  comm_cart = comm.Create_cart(dims=(num_row, num_col),
9                               periods=(True, True), reorder=True)
10
11 rank_cart = comm_cart.Get_rank()
12
13 my_row, my_col = comm_cart.Get_coords(rank_cart)
```

Здесь в строке 11 определяется номер `rank_cart` MPI-процесса в новом коммуникаторе `comm_cart`, а в строке 13 определяются декартовы координаты (`my_row`, `my_col`) MPI-процесса по его номеру `rank_cart` в новом коммуникаторе `comm_cart` с декартовой топологией.

Затем создаются вспомогательные коммуникаторы.

```
14 comm_col = comm_cart.Split(rank_cart % num_col, rank_cart)
15 comm_row = comm_cart.Split(rank_cart // num_col, rank_cart)
```

Здесь, например, в строке 14 порождаются коммуникаторы `comm_col`. Надо обратить внимание на то, что порождаются именно коммуникаторы (во множественном числе), а не один коммуникатор. Приведём поясняющий пример для случая 9-ти MPI-процессов (то есть `numprocs = 9`). Первым аргументом функции `Split()` является значение `color`, определённое как `rank_cart % num_col`, которое для процессов с `rank_cart = 0, 1, 2, 3, 4, 5, 6, 7, 8`

примет значения `0, 1, 2, 0, 1, 2, 0, 1, 2`. Таким образом, будет создано три коммуникатора: в первый войдут процессы со значением `color = 0`, во второй — со значением `color = 1`, в третий — со значением `color = 2`. Можно дать наглядную интерпретацию такой группировки процессов — см. рис. 4.1: если процессы коммуникатора `comm_cart` образуют двумерную сетку, то процессы коммуникаторов `comm_col` являются столбцами этой двумерной сетки процессов.

При этом объект `comm_col` на каждом MPI-процессе будет содержать информацию о разных процессах. Например, на процессах с `rank_cart = 1, 4` коммуникатора `comm_cart` объект `comm_col` будет содержать информацию о процессах с `rank_cart = 1, 4, 7` коммуникатора `comm_cart`. А на процессе с `rank_cart = 2` коммуникатора `comm_cart` объект `comm_col` будет содержать информацию о процессах с `rank_cart = 2, 5, 8` коммуникатора `comm_cart`.

Аналогично, если процессы коммуникатора `comm_cart` образуют двумерную сетку, то процессы коммуникаторов `comm_row` являются строками этой двумерной сетки процессов.

Принципиальной особенностью такой программной реализации является то, что дополнительные коммуникаторы создаются на основе коммуникатора `comm_cart` с декартовой топологией типа двумерного тора. Таким образом, каждый дополнительный коммуникатор, введённый указанным образом, обладает топологией типа кольца, так как является одномерным срезом двумерного тора. Напомним, что, используя виртуальную топологию с аргументом `reorder`≡`True`, предполагается, что системе удаётся удачно отобразить виртуальную топологию `comm_cart` на реальную физическую топологию вычислительной системы. Таким образом, можно надеяться на то, что вспомогательные коммуникаторы также удачным образом отображены на реальную вычислительную систему, в результате чего MPI-процессы, входящие в каждый из таких вспомогательных коммуникаторов, будут выполняться на вычислительных узлах, образующих кольцо в топологическом смысле.

Теперь организовывается считывание координат сенсоров и источников поля из бинарных файлов coords_of_sensors.bin и coords_of_sources.bin.

Структура бинарного файла coords_of_sensors.bin следующая: первые 32 бита (4 байта) должны содержать целочисленное значение типа `int32`, определяющее число сенсоров $S_{sensors}$, а последующие $3 \times S_{sensors} \times 64$ бита данных должны содержать последовательность значений координат

$(x_1,y_1,z_1,\ x_2,y_2,z_3,\ \ldots,\ x_{S_{sensors}},y_{S_{sensors}},z_{S_{sensors}})$, каждое из которых занимает 64 бита (8 байт) памяти и имеет тип данных `float64` (`double`). Структура бинарного файла coords_of_cources.bin аналогична.

Данные из соответствующих файлов считываются процессом с `rank_cart = 0` коммуникатора `comm_cart`.

```
16 if rank_cart == 0 :
17
18     file_1 = "coords_of_sources.bin"
19     N_sources = array(fromfile(file_1, dtype=int32,
20                                count=1, offset=0)[0], dtype=int32)
21     coords_of_sources = fromfile(file_1, dtype=float64,
22                                  count=3*N_sources, offset=4)
23
24     file_2 = "coords_of_sensors.bin"
25     S_sensors = array(fromfile(file_2, dtype=int32,
26                                count=1,offset=0)[0], dtype=int32)
27     coords_of_sensors = fromfile(file_2, dtype=float64,
28                                  count=3*S_sensors, offset=4)
29
30 else :
31     N_sources = array(0, dtype=int32)
32     S_sensors = array(0, dtype=int32)
33     coords_of_sources = None
34     coords_of_sensors = None
35
36 comm_cart.Bcast([N_sources, 1, MPI.INT], root=0)
37 comm_cart.Bcast([S_sensors, 1, MPI.INT], root=0)
```

Необходимо отметить, что матрица A имеет в общем случае размеры $m \cdot S_{sensors} \times n \cdot N_{sources}$. Параметр m определяется тем, какого типа обрабатываются магнитные данные: $m = 3$ в случае обработки данных измерения компонент индукции магнитного поля, $m = 5$ в случае обработки данных измерения компонент тензора градиентов компонент индукции магнитного поля, $m = 8$ в случае обработки полных магнито-градиентных данных. Параметр n определяется тем какого типа ищется решение: $n = 3$ в случае восстановления вектор-функции намагниченности, $m = 1$ в случае восстановления скалярной функции магнитной восприимчивости. Здесь рассматривается частный случай $n = 3$, но рассматриваемая программная реализация легок модифицируется на случай $n = 1$. Таким образом, $m \cdot S_{sensors} \equiv M$ и $3 \cdot N_{sources} \equiv N$.

Значение $N_{sources}$, содержащееся в массиве `N_sources`, нужно будет иметь на всех процессах коммуникатора `comm_cart`. Поэтому в строке 36 оно рассылается с помощью MPI-функции коллективного взаимодействия процессов `Bcast()` с процесса с `rank_cart = 0` коммуникатора `comm_cart` по всем процессам этого коммуникатора. Необходимо напомнить, что MPI-функции из пакета `mpi4py` работают только с `numpy`-массивами. Именно по этой причине в строке 19 считанное значение $N_{sources}$ процессом с `rank_cart = 0` сразу преобразуется в `numpy`-массив `N_sources`, а на остальных процессах (строка 31) выделяется место в памяти под `numpy`-массив, который в последствие будет содержать значение $N_{sources}$. Для единообразия аналогично надо поступить и со значением $S_{sensors}$, которое будет содержащееся в массиве `S_sensors`.

Таким образом, на процессе с `rank_cart = 0` коммуникатора `comm_cart` будут содержаться массивы `coords_of_sources` и `coords_of_sensors`. Далее нужно будет распределить эти массивы по частям среди всех процессов коммуникатора `comm_cart`. Таким образом, на остальных процессах массивы `coords_of_sources` и `coords_of_sensors` будут несущественны. Но с учётом того, что эти массивы в дальнейшем будут являться аргументами MPI-функции `Scatterv()` (см. далее строку 44 программного кода), которая вызывается на всех процессах некоторых коммуникаторов, то соответствующие объекты Python формально должны быть инициализированы на всех процессах соответствующих коммуникаторов. Для этого в строках 33–34 выполняется соответствующая инициализация для всех процессов с `rank_cart >= 1` коммуникатора `comm_cart`.

Но сначала надо подготовить вспомогательные массивы `rcounts` и `displs`, которые помогут распределять данные (либо собирать их) между всеми процессами некоторого коммуникатора. В MPI принято, что `rcounts` — целочисленный массив, содержащий информацию о количестве элементов, передаваемых каждому процессу (принимаемых от каждого процесса); при этом индекс элемента массива равен идентификатору принимающего (посылающего) процесса, а размер массива равен числу процессов в соответствующем коммуникаторе. В свою очередь, `displs` — целочисленный массив, содержащий информацию о смещении относительно начала массива из которого соответствующие данные распределяются (собираются); при этом индекс равен идентификатору принимающего (посылающего) процесса, а размер массива равен числу процессов в соответствующем коммуникаторе.

Так как таких пар массивов будет две, алгоритм подготовки этих массивов выделяется в отдельную функцию, которую следует включить либо в отдельный модуль (и в этом случае необходимо раскомментировать строку 3), либо добавить эту функцию в программный код этой программы после строки 37.

```
def auxiliary_arrays(M, num) :
    ave, res = divmod(M, num)
    rcounts = empty(num, dtype=int32)
    displs = empty(num, dtype=int32)
    for k in range(0, num) :
        if k < res :
            rcounts[k] = ave + 1
        else :
            rcounts[k] = ave
        if k == 0 :
            displs[k] = 0
        else :
            displs[k] = displs[k-1] + rcounts[k-1]
    return rcounts, displs
```

Массив `rcounts` заполняется так, что число элементов M распределяется среди числа процессов `num` равномерно таким образом, чтобы максимальное отличие в числе элементов достигало только 1.

Далее, на каждом процессе формируются две пары вспомогательных массивов: `rcounts_sources` + `displs_sources` и `rcounts_sensors` + `displs_sensors`. Постфикс `_sensors` обозначает, что соответствующие массивы содержат информацию о том, как число элементов $S_{sensors}$ распределяется между блоками вдоль столбца сетки процессов. А постфикс `_sources` обозначает, что соответствующие массивы содержат информацию о том, как число элементов $N_{sources}$ распределяется между блоками вдоль строки сетки процессов.

```
38 rcounts_sources,displs_sources=auxiliary_arrays(N_sources,num_col)
39 rcounts_sensors,displs_sensors=auxiliary_arrays(S_sensors,num_row)
```

MPI-процесс знает свои декартовы координаты (`my_row`, `my_col`) в коммуникаторе `comm_cart` с декартовой топологией. Поэтому он может определить свои значения $S_{part_{(.)}}$ и $N_{part_{(.)}}$, определяющие размерность $m \cdot S_{part_{(.)}} \times 3 \cdot N_{part_{(.)}}$ части матрицы $A_{part_{(.)}}$, за операции с которой будет отвечать этот MPI-процесс. Соответствующие значения $S_{part_{(.)}}$ и $N_{part_{(.)}}$ на каждом MPI-процессе будут содержаться в `S_part` и `N_part`.

```
40 N_part = rcounts_sources[my_col]
41 S_part = rcounts_sensors[my_row]
```

При этом

$$\sum_{m=0}^{\texttt{num\_row}-1} S_{part_{(m)}} = S_{sensors}, \qquad \sum_{n=0}^{\texttt{num\_col}-1} N_{part_{(n)}} = N_{sources}.$$

Стоит ещё раз напомнить, что $m \cdot S_{sensors} \equiv M$ и $3 \cdot N_{sources} \equiv N$, где M и N — размерности матрицы системы в постановке задачи (4.1).

Теперь можно распределить элементы массивов `coords_of_sources` и `coords_of_sensors` среди всех процессов коммуникатора `comm_cart`. Сначала будет рассмотрен алгоритм распределения элементов массива `coords_of_sources`.

```
42 coords_of_sources_part = empty(3*N_part, dtype=float64)
43 if rank_cart in range(num_col) :
44     comm_row.Scatterv([coords_of_sources, 3*rcounts_sources,
45                        3*displs_sources, MPI.DOUBLE],
46                       [coords_of_sources_part,
47                        3*N_part, MPI.DOUBLE],
48                       root=0)
49 comm_col.Bcast([coords_of_sources_part, 3*N_part, MPI.DOUBLE],
50                root=0)
51 coords_of_sources_part = reshape(coords_of_sources_part,
52                                  (3, N_part), order='F')
```

В строке 42 выделяется место в памяти под массив `coords_of_sources_part`, который будет содержать только ту часть координат источников, которая будет необходима для формирования части $A_{part_{(.)}}$ матрицы A, за операции с которой будет отвечать этот MPI-процесс.

В строках 43–48 осуществляется распределение координат источников поля, содержащихся в массиве `coords_of_sources` на процессе с `rank_cart = 0` коммуникатора `comm_cart`, по частям, которые будут содержаться в массивах `coords_of_sources_part` каждого процесса в строке сетки процессов с индексом 0, или, другими словами, на процессах с `my_row = 0` (см. рис. 4.2).

Замечание. Если строку 43 (то есть условие) убрать, то действие в сроках 44–48 будут пытаться выполнить все процессы коммуникатора `comm_cart`. Но с учётом того, что в данной программной реализации функция `Scattevr()` реализует коллективное взаимодействие процессов только в коммуникаторе `comm_row`, то на процессах с `rank_cart > num_col-1` будет происходить

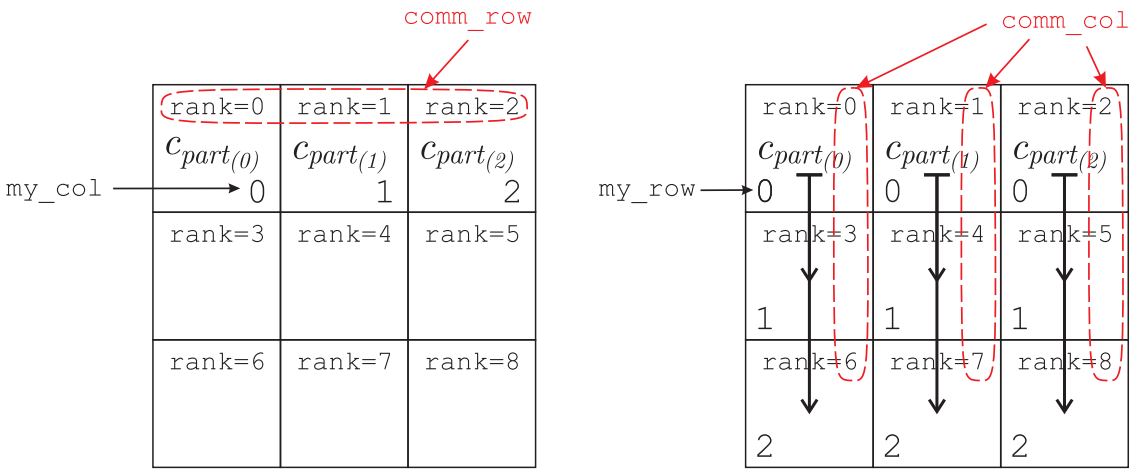


Рисунок 4.2 — Распределение данных по сетке процессов: (а) в результате выполнения строк 43–48 программного кода и (б) в результате выполнения строк 49–50 программного кода. Для удобства отображения, вектора $c_{part_{(\cdot)}}$ выполняют роль содержимого массивов `coords_of_sources_part` на каждом из MPI-процессов, а `rank` роль `rank_cart`.

следующее. Процессы с `rank_cart = num_col,...,2*num_col-1` образуют коммуникатор `comm_row`, который содержит строку сетки процессов с индексом 1. Функция `Scatterv()` будет пытаться разослать массив `rcounts_N` с нулевого процесса коммуникатора `comm_row`, по всем процессам этого коммуникатора. Но нулевой процесс этого коммуникатора также является процессом с `rank_cart = num_col` коммуникатора `comm_cart`, а в адресном пространстве (оперативной памяти) этого процесса массив `coords_of_sources` отсутствует. В результате возникнет ошибка. Аналогичные ошибки возникнут и на всех остальных процессах коммуникатора `comm_cart`.

В строке 49–50 осуществляется пересылка каждого массива `coords_of_sources_part` с нулевого процесса коммуникатора `comm_col` по всем процессам соответствующего коммуникатора (результат представлен на рис. 4.2).

Особенность этой программной реализации следующая. Этот программный код запускается на некотором числе MPI-процессов. И все MPI-процессы выполняют строки 49-50 программного кода. Но эти MPI-процессы принадлежат разным коммуникаторам `comm_col`. Поэтому каждый MPI-процесс реализует операции пересылки сообщений в своём коммуникаторе. Поэтому в реальности рассылка содержимого массива `coords_of_sources_part` по процессам коммуникатора `comm_col` осуществляется независимо от рассылки содержимого массива `coords_of_sources_part` на процессах других коммуни-

каторов с таким же именем `comm_col`. То есть пересылка данных вдоль каждого столбца сетки процессов осуществляется параллельно с пересылкой аналогичных данных вдоль других столбцов сетки процессов! А это, как следствие, сокращает накладные временные расходы по приёму/передаче сообщений между различными процессами.

Замечание. Распределение координат источников по процессам из первой (с индексом 0) строки сетки процессов, выполняющееся с помощью функции `Scatterv()` в строках 43–47, может быть выполнено и следующим эквивалентным (в функциональном смысле) образом:

```
if rank_cart == 0 :
    coords_of_sources_part[:] = coords_of_sources[0:3*N_part]
    for k in range(1, num_col) :
        comm_row.Send([coords_of_sources[3*displs_sources[k]:],
                       3*rcounts_sources[k], MPI.DOUBLE],
                      dest=k, tag=0)
else :
    comm_row.Recv([coords_of_sources_part, 3*N_part, MPI.DOUBLE],
                  source=0, tag=0, status=None)
```

Однако в этом случае время обмена сообщениями внутри коммуникатора `comm_row` будет пропорционально `num_col` $- 1$ по сравнению с $\log_2$ `num_col` в случае использования для коммуникаций функции `Scatterv()`.

В строках 51–52 с помощью функции `reshape()` происходит преобразование одномерного массива `coords_of_sources_part` в двумерный, в первой строке (с индексом 0) которого будут содержаться x-координаты источников поля, во второй строке — y-координаты, а в третьей — z-координаты. Таким образом, число столбцов полученного массива равно числу источников поля (содержится в массиве `N_part`), чьи координаты будут использоваться MPI-процессом при формировании своей части $A_{part_{(.)}}$ матрицы A.

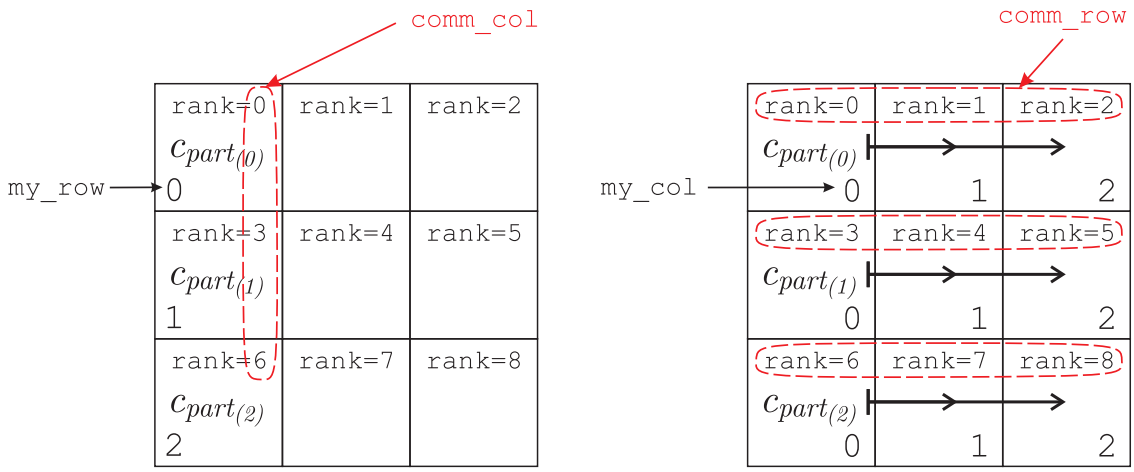
Элементы массива `coords_of_sensors` распределяются среди всех процессов коммуникатора `comm_cart` аналогичным образом.

```
53 coords_of_sensors_part = empty(3*S_part, dtype=float64)
54 if rank_cart in range(0, numprocs, num_col) :
55     comm_col.Scatterv([coords_of_sensors, 3*rcounts_sensors,
56                        3*displs_sensors, MPI.DOUBLE],
57                       [coords_of_sensors_part,
58                        3*S_part, MPI.DOUBLE],
59                       root=0)
60 comm_row.Bcast([coords_of_sensors_part,3*S_part,MPI.DOUBLE],
```

```
61                  root=0)
62  coords_of_sensors_part = reshape(coords_of_sensors_part,
63                                   (3, S_part), order='F')
```

В строке 53 выделяется место в памяти под массив `coords_of_sensors_part`, который будет содержать только часть координат сенсоров, которые будут необходимы для формирования части $A_{part_{(.)}}$ матрицы A, за операции с которой будет отвечать этот MPI-процесс.

В строках 54–59 осуществляется распределение координат сенсоров, содержащихся в массиве `coords_of_sensors` на процессе с `rank_cart = 0` коммуникатора `comm_cart`, по частям, которые будут содержаться в массивах `coords_of_sensors_part` каждого процесса в столбце сетки процессов с индексом 0, или, другими словами, на процессах с `my_col = 0` (см. рис. 4.3).



(а) (б)

Рисунок 4.3 — Распределение данных по сетке процессов: (а) в результате выполнения строк 54–59 программного кода и (б) в результате выполнения строк 60–61 программного кода. Для удобства отображения, вектора $c_{part_{(.)}}$ выполняют роль содержимого массивов `coords_of_sensors_part` на каждом из MPI-процессов, а `rank` роль `rank_cart`.

В строках 60–61 осуществляется пересылка каждого массива `coords_of_sensors_part` с нулевого процесса коммуникатора `comm_row` по всем процессам соответствующего коммуникатора (результат представлен на рис. 4.3).

В строках 62–63 с помощью функции `reshape()` происходит преобразование одномерного массива `coords_of_sensors_part` в двумерный, в первой строке (с индексом 0) которого содержатся x-координаты сенсоров, во второй строке — y-координаты, а в третьей — z-координаты. Таким образом, число столбцов полученного массива равно числу сенсоров (содержится в массиве

S_part), чьи координаты будут использоваться MPI-процессом при формировании своей части $A_{part_{(\cdot)}}$ матрицы A.

Аналогичным образом на всех процессах коммуникатора comm_cart можно подготовить части правой части решаемой системы в соответствии со схемой изображённой на рис. 4.1.

Сначала процесс с rank_cart = 0 коммуникатора comm_cart считывает из бинарного файла b.bin правую часть b системы (4.1). Этот бинарный файл содержит $3 \times S_{sensors} \times 64$ бита данных, которые представляют из себя последовательность значений компонент индукции магнитного поля (B_{x1},B_{y1},B_{z1}, B_{x2},B_{y2},B_{z3}, ..., $B_{xS_{sensors}}$,$B_{yS_{sensors}}$,$B_{zS_{sensors}}$), каждое из которых занимает 64 бита (8 байт) памяти и имеет тип данных float64 (double). Как вариант, этот бинарный файл может содержать $5 \times S_{sensors} \times 64$ бита данных в случае обработки данных измерения компонент тензора градиентов компонент магнитной индукции поля, либо $8 \times S_{sensors} \times 64$ бита данных в случае обработки полных магнито-градиентных данных. Размеры b будут учитываться при формировании матрицы системы (4.1).

```
64 if rank_cart == 0 :
65     file_3 = "b.bin"
66     b = fromfile(file_3, dtype=float64, count=-1, offset=0)
67 else :
68     b = None
```

Затем вектор b, содержащийся в массиве b на процессе с rank_cart = 0, распределяется по всем процессам коммуникатора comm_cart в соответствии со схемой изображённой на рис. 4.1.

```
69 b_part = empty(3*S_part, dtype=float64)
70 if rank_cart in range(0, numprocs, num_col) :
71     comm_col.Scatterv([b, 3*rcounts_sensors,
72                        3*displs_sensors, MPI.DOUBLE],
73                       [b_part, 3*S_part, MPI.DOUBLE],
74                       root=0)
75 comm_row.Bcast([b_part, 3*S_part, MPI.DOUBLE], root=0)
```

Схематично результаты работы команд в строках 70–75 изображены на рис. 4.4.

В результате каждый процесс с rank_cart = $(\cdot)$ коммуникатора comm_cart будет содержать следующие данные.

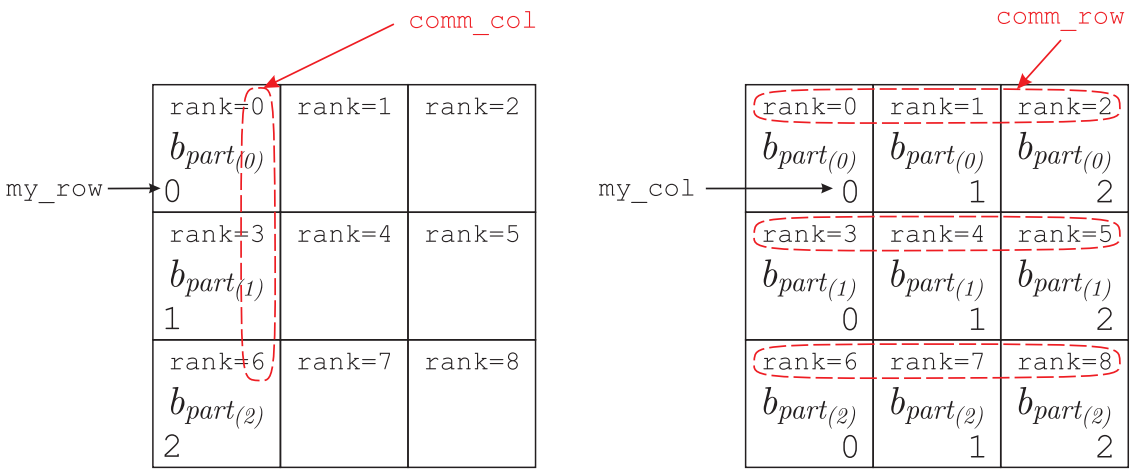


Рисунок 4.4 — Распределение данных из массива `b` по сетке процессов: (а) в результате выполнения строк 70–74 программного кода и (б) в результате выполнения строки 75 программного кода. Для удобства отображения, роль `rank_cart` выполняет `rank`.

1. В массивах `S_part` и `N_part` будут содержатся значения $S_{part_{(.)}}$ и $N_{part_{(.)}}$, которые определяют размерность $m \cdot S_{part_{(.)}} \times n \cdot N_{part_{(.)}}$ части матрицы $A_{part_{(.)}}$, за операции с которой будет отвечать этот MPI-процесс. Напомним, что параметр m определяется тем, какого типа обрабатываются магнитные данные: $m = 3$ в случае обработки данных измерения компонент индукции магнитного поля, $m = 5$ в случае обработки данных измерения компонент тензора градиентов компонент индукции магнитного поля, $m = 8$ в случае обработки полных магнито-градиентных данных. Параметр n определяется тем какого типа ищется решение: $n = 3$ в случае восстановления вектор-функции намагниченности, $m = 1$ в случае восстановления скалярной функции магнитной восприимчивости. Здесь рассматривается частный случай $n = 3$, но рассматриваемая программная реализация легко модифицируется на случай $n = 1$.
2. В массивах `coords_of_sensors_part` и `coords_of_sources_part` будут содержаться те значения координат сенсоров и источников, которые будут использоваться MPI-процессом при формировании своей части $A_{part_{(.)}}$ матрицы A.

Таким образом, на каждом процессе остаётся вызвать функцию `A_part_preparation()`, которая на всех процессах коммуникатора `comm_cart`

подготавливает свою часть $A_{part_{(.)}}$ матрицы A (эта функция будет подробно рассмотрена в следующем подпараграфе 4.1.2).

```
76 A_part = A_part_preparation(coords_of_sources_part,
77                              coords_of_sensors_part,
78                              N_part, S_part)
```

Далее необходимо подготовить нулевое начальное приближение для решения x системы (4.1). В роли x выступает вектор

$$M = (M_{x1}\, M_{y1}\, M_{z1},\ M_{x2}\, M_{y2}\, M_{z3}\ \ldots\ M_{xN_{sources}}\, M_{yN_{sources}}\, M_{zN_{sources}})^T,$$

определяющий значения компонент магнитных диполей. На каждом процессе определяется $M_{part_{(.)}} \equiv 0$, что эквивалентно заданию нулевого начального приближения $\equiv 0$.

```
79 M_part = zeros(3*N_part, dtype=float64)
```

Также задаётся параметр регуляризации α.

```
80 alpha = 3.5*10**(-14)
```

И вызывается функция `conjugate_gradient_method()`, которая реализует поиск регуляризованного решения системы (4.1) (является решением системы (4.3)).

```
81 M_part, s, M_classic_part = conjugate_gradient_method(
82                                A_part, b_part, M_part, alpha,
83                                comm_row, comm_col, 3*N_sources)
```

Эта функция представляет в плане параллельных вычислений основной интерес, поэтому она будет подробно разобрано отдельно в параграфе 4.2.

Результатом работы этой функции на каждом процессе должен быть `numpy`-массив `M_part`, содержащий часть решения M (см. рис. 4.1). Все `numpy`-массивы `M_part` могут быть собраны в единый `numpy`-массив `M`, содержащий решение M, например на процессе с `rank_cart = 0` коммуникатора `comm_cart`, который также является нулевым процессом одного из коммуникаторов `comm_row`:

```
84 if rank_cart == 0 :
85     M = zeros(3*N_sources, dtype=float64)
86 elif rank_cart in range(1, num_col) :
87     M = None
88
89 if rank_cart in range(num_col) :
```

```
90      comm_row.Gatherv([M_part, 3*N_part, MPI.DOUBLE],
91                       [M, 3*rcounts_sources,
92                        3*displs_sources, MPI.DOUBLE],
93                       root=0)
94
95  if M_classic_part is not None :
96      if rank_cart == 0 :
97          M_classic = zeros(3*N_sources, dtype=float64)
98      elif rank_cart in range(1, num_col) :
99          M_classic = None
100
101     if rank_cart in range(num_col) :
102         comm_row.Gatherv([M_classic_part, 3*N_part, MPI.DOUBLE],
103                          [M_classic, 3*rcounts_sources,
104                           displs_sources, MPI.DOUBLE],
105                          root=0)
```

Если программа запускается удалённо на многопроцессорной системе, то результаты расчётов необходимо будет сохранить в файл, скопировать на свой компьютер и построить график с помощью другой вспомогательной программы. Если для тестирования используется свой персональных компьютер, то график решения можно сразу вывести на экран. Способы графической интерпретации полученных результатов расчётов не относятся к теме работы и здесь не приводятся, предоставляя читателю возможность самим выбрать наиболее привычные способы графического отображения результатов расчётов.

4.1.2 Подготовка на вычислительных узлах частей матрицы СЛАУ

В этом подпараграфе рассматривается функция `A_part_preparation()`, которая на MPI-процессе, запускающем эту функцию, формирует свою часть $A_{part_{(.)}}$ матрицы A. Таким образом, каждый MPI-процесс сможет формировать свою часть матрицы параллельно с формированием других частей матрицы на других MPI-процессах при одновременном вызове этой функции всеми MPI-процессами.

Необходимо ещё раз напомнить, что матрица A имеет размеры $m \cdot S_{sensors} \times n \cdot N_{sources}$. Параметр m определяется тем, какого типа обрабатываются магнитные данные: $m = 3$ в случае обработки данных измерения компонент индукции

магнитного поля, $m = 5$ в случае обработки данных измерения компонент тензора градиентов компонент индукции магнитного поля, $m = 8$ в случае обработки полных магнито-градиентных данных. Параметр n определяется тем какого типа ищется решение: $n = 3$ в случае восстановления вектор-функции намагниченности, $m = 1$ в случае восстановления скалярной функции магнитной восприимчивости.

Часть $A_{part_{(\cdot)}}$ матрицы A имеет размеры $m \cdot S_{part_{(\cdot)}} \times nN_{part_{(\cdot)}}$.

Для избежания излишней громоздкости изложения материала приводится пример соответствующей функции в случае обработки данных измерений только магнитного поля ($m = 3$) с целью восстановления намагниченности ($n = 3$). Для случаев $m = 5$, $m = 8$ и $n = 1$ функция будет иметь аналогичную структуру.

```
def A_part_preparation(coords_of_sources,
                       coords_of_sensors,
                       N_sources, S_sensors) :

    x = coords_of_sources[0,:]
    y = coords_of_sources[1,:]
    z = coords_of_sources[2,:]

    x_obs = coords_of_sensors[0,:]
    y_obs = coords_of_sensors[1,:]
    z_obs = coords_of_sensors[2,:]

    A = empty((3*S_sensors, 3*N_sources), dtype=float64)

    for j in range(S_sensors) :
        for i in range(N_sources) :
            rho = sqrt((x[i] - x_obs[j])**2 +
                       (y[i] - y_obs[j])**2 +
                       (z[i] - z_obs[j])**2)
            A[3*j+0, 3*i+0] = 1/(rho**5)*\
                              (3*(x[i] - x_obs[j])**2 - rho**2)
            A[3*j+0, 3*i+1] = 1/(rho**5)*\
                              (3*(x[i] - x_obs[j])*(y[i] - y_obs[j]))
            A[3*j+0, 3*i+2] = 1/(rho**5)*\
                              (3*(x[i] - x_obs[j])*(z[i] - z_obs[j]))
            A[3*j+1, 3*i+0] = 1/(rho**5)*\
                              (3*(x[i] - x_obs[j])*(y[i] - y_obs[j]))
            A[3*j+1, 3*i+1] = 1/(rho**5)*\
```

```
29                                  (3*(y[i] - y_obs[j])**2 - rho**2)
30              A[3*j+1, 3*i+2] = 1/(rho**5)*\
31                                  (3*(y[i] - y_obs[j])*(z[i] - z_obs[j]))
32              A[3*j+2, 3*i+0] = 1/(rho**5)*\
33                                  (3*(x[i] - x_obs[j])*(z[i] - z_obs[j]))
34              A[3*j+2, 3*i+1] = 1/(rho**5)*\
35                                  (3*(y[i] - y_obs[j])*(z[i] - z_obs[j]))
36              A[3*j+2, 3*i+2] = 1/(rho**5)*\
37                                  (3*(z[i] - z_obs[j])**2 - rho**2)
38
39      mu_0 = 4.*pi*10**(-7.)
40      weight = 1. # параметр определяется геометрией задачи
41
42      A = mu_0/(4*pi)*A*weight
43
44      return A
```

При вызове этой функции в качестве входных аргументов передаются массивы `coords_of_sensors_part`, `coords_of_sources_part` и `S_part`, `N_part`. Первая пара массивов содержит значения координат сенсоров и источников, которые будут использоваться MPI-процессом, вызвавшим эту функцию, при формировании своей части $A_{part_{(.)}}$ матрицы A. Вторая пара массивов содержит значения $S_{part_{(.)}}$ и $N_{part_{(.)}}$, которые определяют размерность $3 \cdot S_{part_{(.)}} \times 3 \cdot N_{part_{(.)}}$ формируемой части $A_{part_{(.)}}$ матрицы A.

Однако в самой функции в качестве соответствующих входных параметров выступают массивы `coords_of_sensors`, `coords_of_sources`, `S_sensors` и `N_sources`. Это связано с тем, что этой функции не важно, что она работает только с частью координат. Какой набор координат в эту функцию передан, с тем набором эта функция и работает, воспринимая его полным набором координат.

В строках 5–7 и 9–11 из полученных массивов выделяются x-, y- и z-координаты источников магнитного поля и сенсоров. В строках 15–42 выполняется формирование соответствующего блока матрицы A.

Необходимо обратить внимание на параметр `weight = 1.` в строке 40. Его текущее значение соответствует случаю, когда восстанавливается эквивалентное по внешнему полю распределение магнитных диполей в заданных точках. Если же необходимо восстановить, например, распределение объёмной плотности магнитного момента, то этот параметр должен быть переопределён (он

будет соответствовать элементу разбиения объёма). Поэтому, в случае сложной геометрии задачи он может принимать различные значения в различных точках и, соответственно, его нужно внести внутрь двойного цикла.

Оптимизация работы функции формирования блоков матрицы

Основные вычисления при работе функции `A_part_preparation()` приходятся на двойной цикл. Хорошо известно, что Python будет выполнять соответствующие операции существенно медленнее (в десятки и сотни раз) по сравнению с реализацией соответствующих вычислений, написанных с помощью языков программирования C/C++/Fortran.

Поэтому есть два пути решения проблемы медленной работы этой функции.

В первом случае, можно реализовать соответствующие вычисления в виде отдельной функции, например, на Fortran, и далее вызывать эту функцию из Python.

Во втором случае, можно использовать jit-компиляцию («Just-in-Time») функции с помощью C++.

Для этого необходимо подключить пакет `numba`

```
from numba import jit, prange
```

и добавить к функции декоратор `@jit()` с указанными значениями аргументов:

```
@jit(nopython=True, parallel=True)
def A_part_preparation(coords_of_sources,
                       coords_of_sensors,
                       N_sources, S_sensors) :
    .
    .
    .
    for j in prange(S_sensors) :
        for i in range(N_sources) :
    .
    .
    .
    return A
```

В результате эта функция при запуске программы будет скомпилирована с помощью C++, в результате чего при её вызовах соответствующая часть кода будет работать со скоростью, сопоставимой с C/C++/Fortran.

Более того, необходимо обратить внимание на замену `range` на `prange` во внешнем цикле. С учётом значения аргумента `parallel=True` в декораторе `@jit()`, по возможности будет выполнено распараллеливание вычислений с помощью технологии параллельного программирования OpenMP. Поэтому функция `prange` помогает компилятору понять по какому набору индексов надо осуществлять распараллеливание по OpenMP-нитям.

Таким образом, такие мелкие изменения позволяют существенным образом ускорить работу функции `A_part_preparation()`, в том числе с использованием всех ядер многоядерного процессора, если выполнение MPI-процесса привязывается к вычислительному узлу, содержащему многоядерный процессор.

Замечание. Необходимо отметить, что данные рекомендации являются конструктивными только в том случае, если решаемая задача допускает вычисления с двойной точностью. Если необходимо проводить вычисления с четверной точностью, что требуют многие рассмотренные постановки обратных задач магнитометрии, использование языка программирования Fortran является практически безальтернативным. При этом это утверждение верно только на момент написание диссертационной работы. Это связано с тем, что последние годы в научном сообществе активно продвигается запрос на поддержку пакетом `numpy` языка программирования Python вычислений с четверной точностью. Но возможности реализации вычислений с четверной точностью упираются в то, что большинство функций линейной алгебры пакета `numpy` используют функции программных библиотек линейной алгебры Intel MKL и OpenBLAS (в том числе и те, которые поддерживают использование многоядерных процессоров). До тех пор, пока соответствующие библиотеки не станут полноценно поддерживать вычисления с четверной точностью, Python, как зависимый от них посредством пакета `numpy` язык программирования, будет тоже ограничен в возможностях. Но в связи с бурным развитием вычислительных технологий в последние годы, автор считает необходимым отметить этот момент, так как с каждым годом язык программирования Python сокращает отставание от языка программирования Fortran в части высокопроизводительных научных расчётов.

4.2 Программная реализация с использованием MPI параллельного алгоритма решения переопределённой СЛАУ с плотно заполненной матрицей с учётом ошибок машинного округления

Предыдущий параграф был посвящён описанию действий, необходимых для подготовки на каждом MPI-процессе данных, требующихся для вычислений. В результате этих действий каждый процесс содержит в своей памяти (адресном пространстве) `numpy`-массивы `A_part`, `b_part`, `x_part` (его роль в параграфе 4.1 выполняет `M_part`), константу `alpha` (параметр регуляризации α), объекты `comm_row` и `comm_col` (содержат информацию о коммуникаторах, в которые входит MPI-процесс) и `N` (его роль в параграфе 4.1 выполняет `N_sources`, которое связано с числом неизвестных системы (4.1)).

Теперь можно реализовать параллельную версию функции `conjugate_gradient_method()`, которая реализует модификацию метода сопряжённых градиентов, рассмотренную в главе 3, для решения системы (4.3). Но сначала в подпараграфе 4.2.1 будет рассмотрен последовательный алгоритм, являющийся основой этой функции, и его программная реализация. Затем в подпараграфе 4.2.3 будут рассмотрены различные программные реализации параллельной версии функции `conjugate_gradient_method()`, учитывающие возможности различных стандартов MPI. Будет показано, что использование современных стандартов MPI (MPI-3 и MPI-4) позволяет все дополнительные действия по учёту ошибок машинного округления скрыть на фоне основных операций итерационного алгоритма.

4.2.1 Последовательный алгоритм и его программная реализация

Для целостности изложения материала сначала будет рассмотрена одна из классических реализаций метода сопряжённых градиентов для решения системы (4.3). Этот вариант метода сопряжённых градиентов был взят за основу усовершенствованного алгоритма (см. работу автора [14]), параллельная версия которого является одним из главных предметов обсуждения этой главы.

Классическая реализация метода сопряжённых градиентов

Вектор x с N компонентами, который реализует минимум функционала (4.2), может быть найден с помощью следующего итерационного алгоритма, который строит последовательность $x^{(s)}$. Эта последовательность не более, чем за N шагов, сходится к искомому регуляризованному решению системы (4.1), либо, что то же самое, к решению системы (4.3), исходя из предположения, что все вычисления делаются точно.

Сначала необходимо задать $p^{(0)} = 0$, $s = 1$ и произвольное начальное приближение $x^{(1)}$, а затем выполнить N раз следующую последовательность действий:

$$
\begin{aligned}
r^{(s)} &= \begin{cases} A^T\left(A\,x^{(1)} - b\right) + \alpha R^T(Rx^{(1)}), & \text{если} \quad s = 1, \\ r^{(s-1)} - \dfrac{q^{(s-1)}}{\left(p^{(s-1)},q^{(s-1)}\right)}, & \text{если} \quad s \geqslant 2, \end{cases} \\
p^{(s)} &= p^{(s-1)} + \frac{r^{(s)}}{\left(r^{(s)},r^{(s)}\right)}, \\
q^{(s)} &= A^T\left(A\,p^{(s)}\right) + \alpha R^T(Rp^{(s)}), \\
x^{(s+1)} &= x^{(s)} - \frac{p^{(s)}}{\left(p^{(s)},q^{(s)}\right)}, \\
s &= s + 1.
\end{aligned}
$$

В результате, через N шагов вектор $x^{res} = x^{(N+1)}$ будет являться решением системы (4.3).

Замечание. Если не использовать рекуррентную форму записи и на каждой итерации вычислять невязку $r^{(s)}$ так же, как и на первой итерации, то количество арифметических операций, требуемых для завершения итерационного процесса, возрастёт в два раза. При решении «больших» задач это является мотивацией использования формы метода сопряжённых градиентов в рекуррентной форме записи.

Замечание. Необходимо напомнить про допущение, сделанное в начале этой главы, о том, что для простоты изложения материала в этой главе будет предполагаться, что $R \equiv E$, где E — единичная матрица.

С учётом этого указанный метод поиска регуляризованного решения системы 4 может быть записан в виде псевдокода, представленного в алгоритме 4.1.

Алгоритм 4.1: Псевдокод для последовательной версии «классического» алгоритма.

Входные данные: A, b, $x \equiv x^{(1)}$, α

Результат: x

$s \leftarrow 1$

$p \leftarrow 0$

while $s \leqslant N$ **do**

  if $s = 1$ **then**

    $r \leftarrow A^T(Ax - b) + \alpha x$

  else

    $r \leftarrow r - \dfrac{q}{(p,q)}$

  end

  $p \leftarrow p + \dfrac{r}{(r,r)}$

  $q \leftarrow A^T(Ap) + \alpha p$

  $x \leftarrow x - \dfrac{p}{(p,q)}$

  $s \leftarrow s + 1$

end

Python-код для функции, которая реализует указанный метод сопряжённых градиентов для поиска регуляризованного решения системы (4.1), будет иметь достаточно компактный вид, который представлен в листинге 4.1.

Листинг 4.1: Python-код для последовательной версии «классического» алгоритма.

```
def conjugate_gradient_method(A, b, x, alpha) :
    s = 1
    p = zeros(size(x))
    while s <= N :
        if s == 1 :
            r = dot(A.T, dot(A,x) - b) + alpha*x
        else :
            r = r - q/dot(p, q)
        p = p + r/dot(r, r)
        q = dot(A.T, dot(A, p)) + alpha*p
        x = x - p/dot(p,q)
        s = s + 1
    return x
```

Предполагается, что для вычислений используется стандартный пакет `numpy`. Как следствие, эта вроде бы последовательная программная реализация алгоритма на самом деле уже содержит параллельные вычисления. Это связано с особенностью используемой для основных вычислений функции `dot()` из пакета `numpy`. Эта функция автоматически использует многопоточность, если программа запущена на многоядерном процессоре — все ядра процессора задействуются за счёт использования технологии параллельного программирования OpenMP (реализация на Fortran через пакет Intel MKL).

Усовершенствованная реализация метода сопряжённых градиентов

В параграфе 3.2 главы 3, основываясь на результатах работы автора [14], была описана модификация указанного выше метода сопряжённых градиентов, которая учитывает накапливающиеся в процессе вычислений ошибки машинного округления при принятии решения о прекращении итерационного процесса.

Сначала необходимо задать $p^{(0)} = 0$, $s = 1$ и произвольное начальное приближение $x^{(1)}$, а затем многократно выполнить следующую последовательность действий:

$$r^{(s)} = \begin{cases} A^T\bigl(A\,x^{(1)} - b\bigr) + \alpha x^{(1)}, & \text{если} \quad s = 1, \\ r^{(s-1)} - \dfrac{q^{(s-1)}}{\bigl(p^{(s-1)}, q^{(s-1)}\bigr)}, & \text{если} \quad s \geqslant 2, \end{cases}$$

$$\sigma^2_{r^{(s)}} = \begin{cases} \bigl(A^T\bigr)^{\circ 2}\Bigl(A^{\circ 2}\bigl(x^{(1)}\bigr)^{\circ 2} + b^{\circ 2}\Bigr) + \alpha\bigl(x^{(1)}\bigr)^{\circ 2}, & \text{если} \quad s = 1, \\ \sigma^2_{r^{(s-1)}} + \dfrac{\bigl(q^{(s-1)}\bigr)^{\circ 2}}{\bigl(p^{(s-1)}, q^{(s-1)}\bigr)^2}, & \text{если} \quad s \geqslant 2, \end{cases}$$

если $\dfrac{\Delta^2 \sum\limits_{n=1}^{N} \bigl(\sigma^2_{r^{(s)}}\bigr)_n}{\|r^{(s)}\|_2^2} \geqslant 1$, то итерационный процесс прерывается и $x^{(s)}$ полагается решением системы (4.1),

$$p^{(s)} = p^{(s-1)} + \frac{r^{(s)}}{\bigl(r^{(s)}, r^{(s)}\bigr)}, \quad q^{(s)} = A^T\bigl(A\,p^{(s)}\bigr) + \alpha\,p^{(s)},$$

$$x^{(s+1)} = x^{(s)} - \frac{p^{(s)}}{\bigl(p^{(s)}, q^{(s)}\bigr)}, \quad s = s + 1.$$

Здесь $^{\circ 2}$ — «степень Адамара» (Hadamard power), то есть поэлементное возведение вектора/матрицы во вторую степень, Δ — относительная ошибка машинного округления:

– $\Delta = 10^{-3.3}$ при вычислениях с половинной точностью (`float16`);
– $\Delta = 10^{-7.6}$ при вычислениях с одинарной точностью (`float32`);
– $\Delta = 10^{-16.3}$ при вычислениях с двойной точностью (`float64`);
– $\Delta = 10^{-34.0}$ при вычислениях с четверной точностью (`float128`).

Замечание. Необходимо подчеркнуть ещё раз, что возможная относительная ошибка машинного округления Δ оценивается сверху, что в отдельных случаях может приводить к переоценке общей ошибки.

Данный алгоритм может быть записан в виде псевдокода, оформленного как алгоритм 4.2.

В этом алгоритме красным цветом выделены строки, которые соответствуют действиям по реализации усовершенствованного критерия прекращения итерационного процесса. Если эти строки убрать и изменить условие $True$ на $s \leqslant N$, то получится «классическая» реализация метода сопряжённых градиентов для поиска регуляризованного решения системы (4.1).

Коричневым цветом выделены слагаемые, которые соответствуют регуляризирующей добавке в функционале А. Н. Тихонова (3.2) для случая $\square \equiv L_2$, что эквивалентно $R \equiv E$ в (4.2) в результате дискретизации функционала. Комментарии касательно учёта априорной информации другого вида будут даны в параграфе 4.5. Но при этом сразу необходимо отметить, что при решении практических задач учёт наиболее распространённых на практике ограничений даст сильно разреженную матрицу R, в результате чего вычислительная сложность алгоритма останется неизменной.

Замечание. Алгоритм является полностью эквивалентным алгоритму 3.1. Изменения в форме записи по сравнению с оригинальным алгоритмом 3.1 сделаны с целью полного соответствия алгоритма 4.2 его программной реализации, которая будет приведена далее.

Python-код для функции, которая реализует метод сопряжённых градиентов для решения системы (4.3) с учётом усовершенствованного критерия прекращения итерационного процесса, представлен в листинге 4.2. Стилистика оформления алгоритма и стилистика написания программного кода опять полностью согласованы между собой, чтобы между ними было полное соответствие.

Алгоритм 4.2: Псевдокод для последовательной версии усовершенствованного алгоритма. Красным цветом выделены строки, которые соответствуют действиям по реализации усовершенствованного критерия прекращения итерационного процесса. Алгоритм полностью эквивалентен алгоритму 3.1 (изменения по сравнению с оригинальным алгоритмом 3.1 сделаны с целью полного соответствия алгоритма его программной реализации, представленной в листинге 4.2). Коричневым цветом выделены регуляризирующие добавки для случая $\square \equiv L_2$.

Входные данные: A, b, $x \equiv x^{(1)}$, α

Результат: x

$s \leftarrow 1$

$p \leftarrow 0$

while *True* **do**

  if $s = 1$ **then**

    $r \leftarrow A^T(Ax - b) + \alpha x$

    $\sigma_r^2 \leftarrow \left(A^T\right)^{\circ 2}\left(A^{\circ 2}x^{\circ 2} + b^{\circ 2}\right) + \alpha x^{\circ 2}$

  else

    $r \leftarrow r - \dfrac{q}{\text{scalar\_product\_pq}}$

    $\sigma_r^2 \leftarrow \sigma_r^2 + \dfrac{q^{\circ 2}}{(\text{scalar\_product\_pq})^2}$

  end

  $\text{scalar\_product\_rr} \leftarrow (r,r)$

  $\text{criterion} \leftarrow \dfrac{\Delta^2 \sum\limits_n (\sigma_r^2)_n}{\text{scalar\_product\_rr}}$

  if criterion $\geqslant 1$ **then**

    return x

  end

  $p \leftarrow p + \dfrac{r}{\text{scalar\_product\_rr}}$

  $q \leftarrow A^T(Ap) + \alpha p$

  $\text{scalar\_product\_pq} \leftarrow (p,q)$

  $x \leftarrow x - \dfrac{p}{\text{scalar\_product\_pq}}$

  $s \leftarrow s + 1$

end

Листинг 4.2: Python-код для последовательной версии усовершенствованного алгоритма.

```
def conjugate_gradient_method(A, b, x, alpha) :
    x_classic = None
    s = 1
    p = zeros(size(x))
    while True :
        if s == 1 :
            r = dot(A.T, dot(A,x) - b) + alpha*x
            sigma2_r = dot(A.T**2, dot(A**2, x**2) + b**2)
        else :
            r = r - q/scalar_product_pq
            sigma2_r = sigma2_r + q**2/scalar_product_pq**2
        scalar_product_rr = dot(r, r)
        delta = finfo(r.dtype).eps
        criterion = delta**2 * sum(sigma2_r)/scalar_product_rr
        if criterion >= 1 :
            return x, s, x_classic
        p = p + r/scalar_product_rr
        q = dot(A.T, dot(A, p)) + alpha*p
        scalar_product_pq = dot(p, q)
        x = x - p/scalar_product_pq
        s = s + 1
        if s == size(x) + 1 :
            x_classic = x.copy()
```

Данная программная реализация содержит следующие особенности.

1. Функция `conjugate_gradient_method()` возвращает: 1) массив `x`, который содержит решение системы (4.3), найденное по усовершенствованному алгоритму 4.2; 2) число итераций s, которое потребовалось сделать алгоритму, чтобы найти приближённое решение; 3) массив `x_classic`, который содержит решение, найденное с помощью «классического» алгоритма 4.1 (этот массив содержит такое решение только в том случае, когда число выполненных алгоритмом итераций $s \geqslant N+1$).
2. В этой программной реализации (и в соответствующем псевдокоде) учтено, что результат вычисления каждого скалярного произведения используется по два раза, поэтому результат вычисления скалярного произведения хранится в отдельной переменной.
3. На первой итерации алгоритма ($s = 1$) значение σ_r^2 вычисляется с помощью команды

```
sigma2_r = dot(A.T**2, dot(A**2, x**2) + b**2)
```

Особенность Python состоит в том, что под массивы `A**2` и `A.T**2`, которые содержат матрицы $A^{\circ 2}$ и $\left(A^T\right)^{\circ 2}$, выделяется отдельное место в памяти. Это очень плохо, потому что подразумевается, что решается задача (4.1) с огромной матрицей A. Поэтому при решении реальных «больших» задач массив, содержащий матрицу A, может занимать бóльшую часть памяти. Как следствие, на дополнительные массивы места в памяти может не хватить.

Эту проблему можно обойти несколькими путями.

В первом случае, можно оформить указанную команду в виде отдельной функции на языке программирования C/C++/Fortran, используя для вычислений только элементы массива, в котором хранится матрица A.

Во втором случае, можно использовать специально написанную функцию

```
sigma2_r = dot_special(A.T,dot_special(A,x**2,M,N)+b**2,N,M)
```

которая использует jit-компиляцию с помощью пакета `numba`:

```
from numba import jit, prange

@jit(nopython=True, parallel=True)
def dot_special(A, x, M, N):
    b = empty(M)
    for m in prange(M):
        b[m] = 0.
        for n in range(N):
            b[m] = b[m] + A[m, n]**2*x[n]
    return b
```

Напомним, что такой подход к решению проблемы позволяет не только получить скорость работы функции сопоставимую с реализацией на C/C++/Fortran, но и реализовать использование всех ядер многоядерного процессора. Это связано с заменой `range` на `prange` во внешнем цикле и указанием аргумента `parallel=True` в декораторе `@jit()`, за счёт которого, по возможности, будет выполнено распараллеливание

вычислений с помощью технологии параллельного программирования OpenMP.

Однако, чтобы не перегружать программную реализацию алгоритма такими техническими особенностями, выберем третий путь.

В третьем случае, можно пренебречь учётом накапливающейся ошибки машинного округления на первой итерации и положить на первой итерации $(s = 1)$ $\sigma_r^2 = 0$:

```
sigma2_r = zeros(N)
```

Далее, во всех программных реализациях будет выбираться третий путь, так как многочисленные численные эксперименты продемонстрировали, что при решении реальных прикладных задач учётом накапливающейся ошибки машинного округления на первой итерации $(s = 1)$ можно пренебречь. Однако, при желании, соответствующие программы могут быть легко усовершенствованы с целью учёта ошибки машинного округления и на первой итерации.

4.2.2 Подходы к построению параллельного алгоритма и его программной реализации

В данном подпараграфе будут описаны основные подходы, которые будут использованы для построения параллельного варианта рассматриваемого алгоритма и его последующей программной реализации. При этом будут учитываться возможности различных стандартов технологии параллельного программирования MPI.

Распараллеливаемые операции

В рассматриваемой реализация метода сопряжённых градиентов для решения задачи (4.3) все вычисления приходятся на следующие четыре операции.

1. Скалярное произведение двух векторов размерности N.

Эта операция требует N умножений и $N-1$ сложение — в сумме $2N-1$ арифметических операций. Принято обозначать $2N-1 = O(N^1)$, что говорит о линейной вычислительной сложности операции скалярного произведения.

2. Умножение матрицы размерности $M \times N$ на вектор размерности N.
   Для получения каждого элемента итогового вектора необходимо скалярно умножить соответствующую строку матрицы A на умножаемый вектор. Такие вычисления надо провести для каждого элемента вектора, являющегося результатом умножения. Таким образом, в сумме надо совершить $M \cdot (2N-1)$ арифметических операций. В случае квадратной матрицы ($M = N$) таких операций будет $O(N^2)$, что говорит о квадратичной вычислительной сложности операции умножения матрицы на вектор.
3. Умножение транспонированной матрицы размерности $N \times M$ на вектор размерности M.
   Вычислительная сложность этой операции эквивалентна операции умножения матрицы на вектор.
4. Сложение двух векторов размерности N.
   Эта операция требует N сложений. То есть вычислительная сложность этой операции является линейной.

Далее рассматриваются используемые для распараллеливание этих операций алгоритмы, которые будут применены для построения параллельной версии алгоритма 4.2 и его программной реализации.

Распределение данных по участвующим в вычислениях MPI-процессам

Также, как и в параграфе 4.1, предполагается, что программная реализация разрабатываемого параллельного алгоритма в своей основе использует технологию параллельного программирования MPI и запускается на числе MPI-процессов равном `numprocs`. Все эти процессы входят в коммуникатор `comm` $\equiv$ `MPI.COMM_WORLD`, внутри которого они имеют свой номер/идентификатор `rank` $\in \overline{0,\texttt{numprocs}-1}$.

Предполагается, что в результате действий, описанных в параграфе 4.1, при запуске параллельной версии функции `conjugate_gradient_method()` каждый MPI-процесс содержит в своей памяти (адресном пространстве) следующие данные.

- `numpy`-массив `A_part` содержит часть $A_{part_{(\cdot)}}$ матрицы A (размерность $A_{part_{(\cdot)}} : M_{part_{(\cdot)}} \times N_{part_{(\cdot)}}$).
- `numpy`-массив `b_part` содержит часть $b_{part_{(\cdot)}}$ вектора b (число элементов $b_{part_{(\cdot)}}$: $M_{part_{(\cdot)}}$).
- `numpy`-массив `x_part` содержит часть начального приближения $x_{part_{(\cdot)}}$ вектора x (число элементов $x_{part_{(\cdot)}}$: $N_{part_{(\cdot)}}$).
- Константа `alpha` содержит параметр регуляризации α.
- Объекты `comm_row` и `comm_col` содержат информацию о дополнительных коммуникаторах, в которые входит MPI-процесс.
- Константа `N` содержит число компонент N искомого вектора x.

Распределение элементов матрицы A по блокам между участвующими в вычислениях процессами использует двумерное деление матрицы на блоки и представлена на рис. 4.1.

Необходимо напомнить обозначения, которые были введены в начале параграфа 4.1. Они потребуются для описания соответствующих параллельных алгоритмов.

Число разбиений матрицы A вдоль вертикали обозначается как `num_row` (сокращение от *number of rows*), а число разбиений вдоль горизонтали — `num_col` (сокращение от *number of columns*). В результате матрица A размерности $M \times N$ будет разбита на части $A_{part_{(\texttt{rank})}}$ размерности $M_{part_{(m)}} \times N_{part_{(n)}}$. Здесь номер процесса `rank` связан с индексами m и n следующим образом (см. также рис. 4.5):

$$m = \left\lfloor \frac{\texttt{rank}}{\texttt{num\_col}} \right\rfloor, \quad n = \texttt{rank} - \left\lfloor \frac{\texttt{rank}}{\texttt{num\_col}} \right\rfloor \cdot \texttt{num\_col}.$$

Замечание. Выбранный способ индексации предполагает, что в вычислениях принимают участие все процессы, то есть $\texttt{num\_row} \cdot \texttt{num\_col} \equiv \texttt{numprocs}$.

Структура хранения векторов x и b на различных процессах также показана на рис. 4.1. Видно, что часть $x_{part_{(n)}}$ вектора x для фиксированного индекса n хранится во всех ячейках столбца сетки процессов с индексом n. Аналогично, часть $b_{part_{(m)}}$ вектора b для фиксированного индекса m хранится во всех ячейках строки сетки процессов с индексом m.

Необходимо напомнить, что

$$\sum_{m=0}^{\texttt{num\_row}-1} M_{part_{(m)}} = M, \qquad \sum_{n=0}^{\texttt{num\_col}-1} N_{part_{(n)}} = N.$$

Каждый дополнительный коммуникатор `comm_row` и `comm_col` содержит информацию о группе MPI-процессов, образующих одну из строк или один из столбцов сетки процессов, в который входит MPI-процесс, в памяти которого содержатся объекты `comm_row` и `comm_col`. Графическое пояснение структуры дополнительных коммуникаторов `comm_row` и `comm_col` представлено на рис. 4.5.

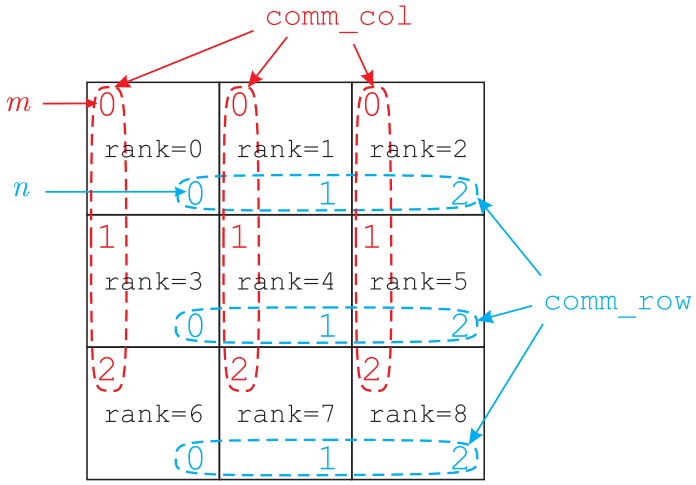


Рисунок 4.5 — Пример распределения 9 MPI-процессов, образующих двумерную сетку 3×3, по дополнительным коммуникаторам `comm_row` и `comm_col`. Каждый такой дополнительный коммуникатор содержит информацию о группе MPI-процессов, образующих одну из строк или один из столбцов сетки процессов.

Параллельный алгоритм умножения матрицы на вектор в случае двумерного деления матрицы на блоки

Слагаемые, которые возникают при вычислении произведения матрицы на вектор, можно разбить на группы следующим образом (ниже приводится пример, соответствующий рис. 4.6, для случая, когда в вычислениях участвует только 9 MPI-процессов, образующих двумерную сетку процессов 3×3):

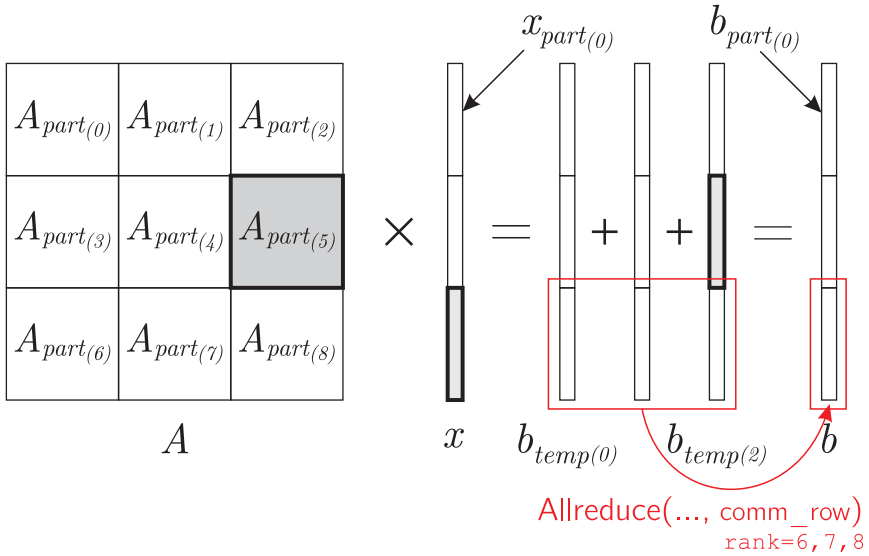


Рисунок 4.6 — Параллельный алгоритм умножения матрицы на вектор в случае двумерного деления матрицы на блоки.

$$Ax = \begin{bmatrix} A_{part(0)} & A_{part(1)} & A_{part(2)} \\ A_{part(3)} & A_{part(4)} & A_{part(5)} \\ A_{part(6)} & A_{part(7)} & A_{part(8)} \end{bmatrix} \begin{bmatrix} x_{part(0)} \\ x_{part(1)} \\ x_{part(2)} \end{bmatrix} =$$

$$= \begin{bmatrix} A_{part(0)}x_{part(0)} + A_{part(1)}x_{part(1)} + A_{part(2)}x_{part(2)} \\ A_{part(3)}x_{part(0)} + A_{part(4)}x_{part(1)} + A_{part(5)}x_{part(2)} \\ A_{part(6)}x_{part(0)} + A_{part(7)}x_{part(1)} + A_{part(8)}x_{part(2)} \end{bmatrix} =$$

$$= \begin{bmatrix} A_{part(0)}x_{part(0)} \\ A_{part(3)}x_{part(0)} \\ A_{part(6)}x_{part(0)} \end{bmatrix} + \begin{bmatrix} A_{part(1)}x_{part(1)} \\ A_{part(4)}x_{part(1)} \\ A_{part(7)}x_{part(1)} \end{bmatrix} + \begin{bmatrix} A_{part(2)}x_{part(2)} \\ A_{part(5)}x_{part(2)} \\ A_{part(8)}x_{part(2)} \end{bmatrix} =$$

$$= \begin{bmatrix} b_{part\_temp(0,0)} \\ b_{part\_temp(1,0)} \\ b_{part\_temp(2,0)} \end{bmatrix} + \begin{bmatrix} b_{part\_temp(0,1)} \\ b_{part\_temp(1,1)} \\ b_{part\_temp(2,1)} \end{bmatrix} + \begin{bmatrix} b_{part\_temp(0,2)} \\ b_{part\_temp(1,2)} \\ b_{part\_temp(2,2)} \end{bmatrix} =$$

$$= b_{temp(0)} + b_{temp(1)} + b_{temp(2)} = \begin{bmatrix} b_{part(0)} \\ b_{part(1)} \\ b_{part(2)} \end{bmatrix} = b.$$

Такая форма представления умножения матрицы на вектор выбрана в связи с тем, чтобы наглядно показать, что вектор x (предполагается, что он состоит из N элементов) нет необходимости хранить целиком на каждом процессе. На каждом процессе нужна лишь своя часть этого вектора $x_{part(n)}$ с $N_{part(n)}$ элементами, где $n \in \overline{0,\texttt{num\_col} - 1}$.

Тогда каждый участвующий в вычислениях процесс будет выполнять операцию умножения своей части $A_{part_{(\texttt{rank})}}$ матрицы A на свою часть $x_{part_{(n)}}$ вектора x. В результате этой операции будет вычисляться вектор $b_{part\_temp_{(m,n)}}$, который будет являться одним из слагаемых части $b_{part_{(m)}}$ итогового вектора b. Каждый участвующий в вычислениях процесс может выполнять это действие независимо от других процессов, при этом оперируя только с данными, которые расположены в памяти этого процесса. Эти вычисления можно реализовать параллельно. Затем, соответствующие слагаемые $b_{part\_temp_{(m,n)}}$ вдоль каждой строки сетки процессов (т.е. индекс m фиксирован) надо собрать (за счёт обмена сообщениями между процессами) по крайней мере на одном процессе и просуммировать.

Таким образом, этот параллельный алгоритм (учитывающий двумерное деление матрицы A на блоки) содержит следующие особенности.

1. На каждом процессе надо хранить не весь вектор x, а лишь часть $x_{part_{(n)}}$ этого вектора. Вектор $x_{part_{(n)}}$ придётся рассылать не по всем процессам коммуникатора `comm`, а только по части процессов этого коммуникатора — по процессам столбца с индексом n сетки процессов. При этом передача данных среди процессов каждого столбца сетки процессов может быть организована параллельно с передачей данных среди процессов других столбцов сетки процессов, что даст выигрыш по времени.
2. При вычислении части $b_{part_{(m)}}$ итогового вектора b необходимо обмениваться сообщениями не всем процессам коммуникатора `comm`, а лишь части процессов этого коммуникатора — процессам строки с индексом m введённой сетки процессов. При этом передача данных вдоль каждой строки сетки процессов может быть организована параллельно с передачей данных среди процессов других строк сетки процессов, что также даст выигрыш по времени.

Опишем теперь некоторые особенности программной реализации этого алгоритма с учётом возможностей технологии параллельного программирования MPI.

Как известно, обмен сообщениями между процессами, организованный с помощью MPI-функций `Send()` и `Recv()`, возможен, но неэффективен. Гораздо эффективнее использовать MPI-функции коллективного взаимодействия процессов. Но такие функции должны быть вызваны на всех процессах коммуникатора. Если работать только с коммуникатором `comm`, который кроме тех

процессов, которые необходимо задействовать (процессы из отдельного столбца или строки сетки процессов), содержит и другие процессы, которые между собой взаимодействовать не будут. Таким образом, возникает необходимость использования дополнительных коммуникаторов `comm_row` и `comm_col`, которые содержат только те группы процессов, внутри которых организовывается взаимодействие по обмену данными с помощью функций коллективного взаимодействия процессов.

Замечание. Процесс создания вспомогательных коммуникаторов `comm_row` и `comm_col` был подробно описан в подпараграфе 4.1.1.

Необходимо напомнить, что если процессы исходного коммуникатора `comm` образуют двумерную сетку, то процессы коммуникаторов `comm_row` являются строками этой двумерной сетки процессов (см. рис. 4.5).

Таким образом, параллельная программная реализация умножения матрицы на вектор может быть оформлена в следующем виде.

```
b_part_temp = dot(A_part, x_part)
b_part = empty(M_part, dtype=float64)
comm_row.Allreduce([b_part_temp, M_part, MPI.DOUBLE],
                   [b_part, M_part, MPI.DOUBLE], op=MPI.SUM)
```

Результат такого умножения (вектор b) будет хранится на всех процессах по частям: часть вектора $b_{part_{(m)}}$ для фиксированного индекса m будет хранится во всех ячейках строки сетки процессов с индексом m.

Замечание. Необходимо отметить, что время обмена сообщениями пропорционально $\log_2 \texttt{numprocs}$, но при этом объём передаваемых сообщений пропорционален $\texttt{numprocs}^{-1/2}$ в случае двумерного деления матрицы на блоки и постоянен в случае одномерного деления матрицы на блоки [175; 188].

Параллельный алгоритм умножения транспонированной матрицы на вектор в случае двумерного деления матрицы на блоки

Параллельный алгоритм умножения транспонированной матрицы на вектор в случае двумерного деления матрицы на блоки достаточно похож на алгоритм, рассмотренный в предыдущем разделе, и в каком-то смысле обладает симметрией относительно него. Так как умножение транспонированной

матрицы на вектор представляет интерес в контексте метода сопряжённых градиентов, предполагается, что матрица системы A хранится по процессам сетки процессов так, как указано на рис. 4.1. Это связано с тем, что, необходимо построить алгоритм умножения матрицы A^T на вектор таким образом, чтобы избежать создания в памяти дополнительных массивов под транспонированную матрицу. Ведь при решении реальных прикладных задач суммарная доступная память на всех процессах ограничена. А необходимо построить алгоритмы для решения очень больших задач!

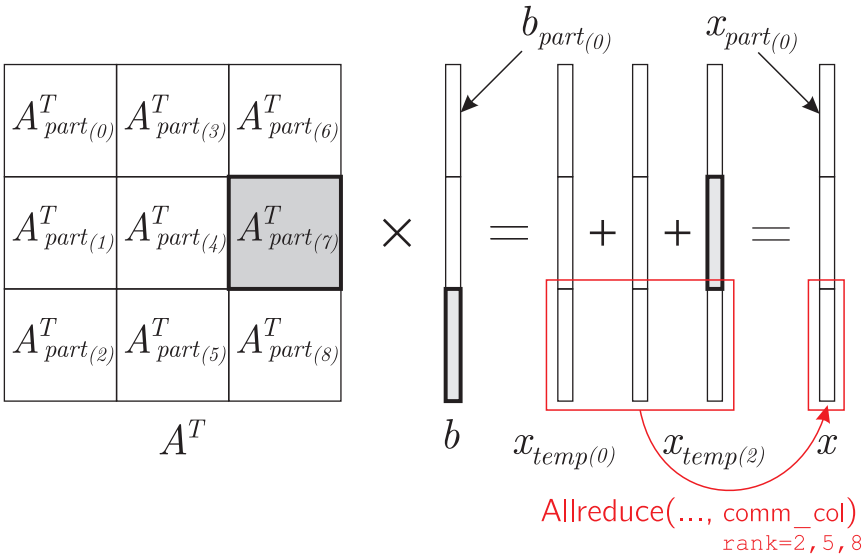


Рисунок 4.7 — Параллельный алгоритм умножения транспонированной матрицы на вектор в случае двумерного деления матрицы на блоки.

Итак, слагаемые, которые возникают при вычислении произведения транспонированной матрицы на вектор, можно разбить на группы следующим образом (также, как и в предыдущем разделе, приведём пример, в котором в вычислениях участвует только 9 MPI-процессов, которые образуют двумерную сетку процессов 3×3, см. рис. 4.7):

$$A^T b = \begin{bmatrix} A^T_{part(0)} & A^T_{part(3)} & A^T_{part(6)} \\ A^T_{part(1)} & A^T_{part(4)} & A^T_{part(7)} \\ A^T_{part(2)} & A^T_{part(5)} & A^T_{part(8)} \end{bmatrix} \begin{bmatrix} b_{part(0)} \\ b_{part(1)} \\ b_{part(2)} \end{bmatrix} =$$

$$= \begin{bmatrix} A^T_{part(0)} b_{part(0)} + A^T_{part(3)} b_{part(1)} + A^T_{part(6)} b_{part(2)} \\ A^T_{part(1)} b_{part(0)} + A^T_{part(4)} b_{part(1)} + A^T_{part(7)} b_{part(2)} \\ A^T_{part(2)} b_{part(0)} + A^T_{part(5)} b_{part(1)} + A^T_{part(8)} b_{part(2)} \end{bmatrix} =$$

$$= \begin{bmatrix} A^T_{part_{(0)}} b_{part_{(0)}} \\ A^T_{part_{(1)}} b_{part_{(0)}} \\ A^T_{part_{(2)}} b_{part_{(0)}} \end{bmatrix} + \begin{bmatrix} A^T_{part_{(3)}} b_{part_{(1)}} \\ A^T_{part_{(4)}} b_{part_{(1)}} \\ A^T_{part_{(5)}} b_{part_{(1)}} \end{bmatrix} + \begin{bmatrix} A^T_{part_{(6)}} b_{part_{(2)}} \\ A^T_{part_{(7)}} b_{part_{(2)}} \\ A^T_{part_{(8)}} b_{part_{(2)}} \end{bmatrix} =$$

$$= \begin{bmatrix} x_{part\_temp_{(0,0)}} \\ x_{part\_temp_{(0,1)}} \\ x_{part\_temp_{(0,2)}} \end{bmatrix} + \begin{bmatrix} x_{part\_temp_{(1,0)}} \\ x_{part\_temp_{(1,1)}} \\ x_{part\_temp_{(1,2)}} \end{bmatrix} + \begin{bmatrix} x_{part\_temp_{(2,0)}} \\ x_{part\_temp_{(2,1)}} \\ x_{part\_temp_{(2,2)}} \end{bmatrix} =$$

$$= x_{temp_{(0)}} + x_{temp_{(1)}} + x_{temp_{(2)}} = \begin{bmatrix} x_{part_{(0)}} \\ x_{part_{(1)}} \\ x_{part_{(2)}} \end{bmatrix} = x.$$

Тогда каждый участвующий в вычислениях процесс будет выполнять операцию умножения своей части $A^T_{part_{(\texttt{rank})}}$ матрицы A^T на свою часть вектора $b_{part_{(m)}}$. В результате этой операции будет вычисляться вектор $x_{part\_temp_{(m,n)}}$, который будет являться одним из слагаемых части $x_{part_{(n)}}$ итогового вектора x. Каждый участвующий в вычислениях процесс может выполнять это действие независимо от других процессов, при этом оперируя только с данными, которые расположены в памяти этого процесса. Эти вычисления можно реализовать параллельно. Затем, соответствующие слагаемые $x_{part\_temp_{(m,n)}}$ вдоль каждого столбца сетки процессов (т.е. индекс n фиксирован) надо собрать (за счёт обмена сообщениями между процессами) по крайней мере на одном процессе и просуммировать их.

Таким образом, этот параллельный алгоритм (учитывающий двумерное деление матрицы A на блоки) содержит следующие особенности.

1. На каждом процессе надо хранить не весь вектор b, а лишь часть $b_{part_{(m)}}$ этого вектора. Вектор $b_{part_{(m)}}$ придётся рассылать не по всем процессам коммуникатора `comm`, а только по части процессов этого коммуникатора — по процессам строки с индексом m сетки процессов. При этом передача данных среди процессов каждой строки сетки процессов может быть организована параллельно с передачей данных среди процессов других строк сетки процессов, что даст выигрыш по времени.
2. При вычислении части $x_{part_{(n)}}$ итогового вектора x необходимо обмениваться сообщениями не всем процессам коммуникатора `comm`, а лишь части процессов этого коммуникатора — процессам столбца с индексом n введённой сетки процессов. При этом передача данных вдоль каж-

дого столбца сетки процессов может быть организована параллельно с передачей данных среди процессов других столбцов сетки процессов, что также даст выигрыш по времени.

Как видно, параллельный алгоритм умножения транспонированной матрицы на вектор в случае двумерного деления матрицы на блоки обладает симметрией относительно параллельного алгоритма умножения матрицы на вектор, рассмотренного в предыдущем разделе. Этот алгоритм получается из рассмотренного ранее за счёт замен $m \leftrightarrow n$ и «строки» $\leftrightarrow$ «столбцы». При этом все действия совершаются не с частями матрицы $A_{part_{(\texttt{rank})}}$, а с их транспонированными частями $A^T_{part_{(\texttt{rank})}}$. Таким образом, программная реализация этого алгоритма будет мало отличаться от программной реализации параллельного алгоритма умножения матрицы на вектор (в том числе будет необходима работа с дополнительными коммуникаторами, которые будут содержать только те группы процессов, среди которых необходимо организовывать взаимодействия по обмену данными с помощью функций коллективного взаимодействия процессов).

Таким образом, параллельная программная реализация умножения транспонированной матрицы на вектор (с учётом того, что операции осуществляются с элементами исходной нетранспонированной матрицы) может быть оформлена в следующем виде.

```
x_part_temp = dot(A_part.T, b_part)
x_part = empty(N_part, dtype=float64)
comm_col.Allreduce([x_part_temp, N_part, MPI.DOUBLE],
                   [x_part, N_part, MPI.DOUBLE], op=MPI.SUM)
```

Результат такого умножения (вектор x) будет храниться на всех процессах по частям: часть вектора $x_{part_{(n)}}$ для фиксированного индекса n будет храниться во всех ячейках столбцах сетки процессов с индексом n.

Параллельный алгоритм скалярного умножения векторов

Предполагается, что структура хранения элементов векторов a и b (которые содержат по N элементов) такая же, как и у вектора x (см. рис. 4.1). В этом случае части векторов, хранящиеся на различных процессах, будут обо-

значаться как $a_{part_{(n)}}$ и $b_{part_{(n)}}$. Каждая такая часть будет состоять из $N_{part_{(n)}}$ элементов. Для простоты изложения материала рассматривается индексация для векторов, хранящихся в первой строке сетки процессов (с индексом $m = 0$). Таким образом, $\mathtt{n} \in \overline{0,\mathtt{num\_col}-1}$.

Слагаемые, которые возникают при вычислении скалярного произведения векторов, можно разбить следующим образом (также приведём пример, в котором в вычислениях участвует только 9 MPI-процессов):

$$
\begin{aligned}
(a,b) &= \left(\begin{bmatrix} a_{part_{(0)}} & a_{part_{(1)}} & a_{part_{(2)}} \end{bmatrix}, \begin{bmatrix} b_{part_{(0)}} & b_{part_{(1)}} & b_{part_{(2)}} \end{bmatrix} \right) = \\
&= \left(a_{part_{(0)}}, b_{part_{(0)}}\right) + \left(a_{part_{(1)}}, b_{part_{(1)}}\right) + \left(a_{part_{(2)}}, b_{part_{(2)}}\right) = \\
&= \mathtt{scalar\_product}_{temp_{(0)}} + \mathtt{scalar\_product}_{temp_{(1)}} + \mathtt{scalar\_product}_{temp_{(2)}} = \\
&= \mathtt{scalar\_product}.
\end{aligned}
$$

Тогда каждый участвующий в вычислениях процесс будет выполнять операцию $(a_{part_{(\mathtt{rank})}}, b_{part_{(\mathtt{rank})}})$. В результате этой операции будет вычисляться одно из слагаемых, из которых состоит итоговое скалярное произведение. Каждый участвующий в вычислениях процесс может выполнять это действие независимо от других процессов, при этом оперируя только с данными, которые расположены в памяти этого процесса. Эти вычисления можно реализовать параллельно. Затем, соответствующие слагаемые вдоль каждой строки сетки процессов надо собрать (за счёт обмена сообщениями между процессами) по крайней мере на одном процессе и просуммировать.

Таким образом, параллельная программная реализация скалярного умножения векторов может быть оформлена в следующем виде.

```
scalar_product_temp = array(dot(r_part, r_part), dtype=float64)
scalar_product = array(0, dtype=float64)
comm_row.Allreduce([scalar_product_temp, 1, MPI.DOUBLE],
                   [scalar_product, 1, MPI.DOUBLE], op=MPI.SUM)
```

Результат такой программной реализации скалярного умножения будет хранится на всех процессах коммуникатора `comm`.

Замечание. Минусом такого алгоритма является то, что различные группы процессов, каждая из которых образует свой коммуникатор `comm_row`, выполняют одни и те же действия. Но этот минус является несущественным в контексте параллельной реализации метода сопряжённых градиентов,

в которой на данный алгоритм приходится пренебрежительно малая часть вычислений в случае решения «больших» задач. Более формально этот минус можно интерпретировать как то, что в параллельных вычислениях скалярного произведения векторов будут принимать участие не все `numprocs` процесса, а только $\sqrt{\texttt{numprocs}}$. Перестройка алгоритма таким образом, чтобы в вычислениях принимали участие все `numprocs` процесса, приведёт к тому, что очень сильно вырастут времязатраты на накладные расходы по взаимодействию процессов друг с другом посредством передачи сообщений с результатами промежуточных вычислений через коммуникационную сеть. В связи с этим, указанный минус является приемлемым при решении прикладных задач.

4.2.3 Параллельная версия усовершенствованного алгоритма и его программная реализация

В этом подпараграфе рассматриваются различные подходы к построению параллельной версии алгоритма 4.2 и его программной реализации. Эти подходы учитывают возможности различных стандартов технологии параллельного программирования MPI. Сначала описывается «наивный» подход к распараллеливанию, который основан на использовании стандарта MPI-2. В этом подходе на все дополнительные операции, связанные с учётом накапливающихся в процессе вычислений ошибок машинного округления, затрачивается дополнительное время. В том числе появляются дополнительные временные расходы на приём/передачу дополнительных сообщений между процессами, которые требуются для организации дополнительных вычислений. Затем описывается подход к распараллеливанию, основанный на использовании стандарта MPI-3. Этот подход за счёт использования неблокирующих (асинхронных) операций обмена сообщениями между процессами позволят скрыть все дополнительные вычисления на фоне обмена сообщениями между процессами в основном алгоритме, а дополнительные пересылки сообщений скрыть на фоне вычислений основного алгоритма. За счёт этого все дополнительные операции (вычисления и пересылка сообщений между MPI-процессами) не требуют дополнительных времязатрат. Таким образом, фиксированное число итераций в параллельной версии усовершенствованного алгоритма может выполняться столько же вре-

мени, как такое же число итераций в параллельной версии «классического» алгоритма (для последовательных реализаций усовершенствованного и «классического» алгоритмов это утверждение, очевидно, не верно). В результате эффективность программной реализации усовершенствованного алгоритма не отличается от эффективности программной реализации классического алгоритма. Затем даются комментарии касательно использования стандарта MPI-4, который позволяет за счёт использования отложенных запросов на взаимодействие существенно повысить эффективность параллельной программной реализации.

Учёт возможностей стандрата MPI-2

«Наивный» параллельный вариант алгоритма 4.2, реализующего усовершенствованную версию метода сопряжённых градиентов для решения системы (4.3), может быть записан в виде псевдокода, оформленного как алгоритм 4.3.

В этом алгоритме красным цветом выделены строки, которые соответствуют действиям по реализации усовершенствованного критерия прекращения итерационного процесса. Если эти строки убрать и изменить условие $True$ на $s \leqslant N$, то получится классическая реализация метода сопряжённых градиентов для решения системы (4.3).

Коричневым цветом выделены операции, которые соответствуют учёту регуляризирующей добавки в функционале А. Н. Тихонова (3.2) для случая $\square \equiv L_2$, что эквивалентно $R \equiv E$ в (4.2) в результате дискретизации функционала. Комментарии касательно учёта априорной информации другого вида будут даны в параграфе 4.5. Но важно отметить, что при решении практических задач матрица R будет разреженной, в результате чего вычислительная сложность алгоритма останется неизменной.

Особенность этого алгоритма заключается в том, что этапы счёта и этапы пересылки сообщений, содержащих результаты промежуточных вычислений, чётко разделены. Поэтому накладные расходы по обмену сообщениями между участвующими в вычислениях процессами будут негативно влиять на эффективность параллельной программной реализации.

Алгоритм 4.3: Псевдокод («наивный») для параллельной версии усовершенствованного алгоритма.

Входные данные: A_{part}, b_{part}, $x_{part} \equiv x^{(1)}_{part}$, α, `comm_row`, `comm_col`, N

Результат: x_{part}

$s \leftarrow 1$; $p_{part} \leftarrow 0$

while *True* **do**

  if $s = 1$ **then**

    $Ax_{part\_temp} \leftarrow A_{part} x_{part}$

    $Ax_{part} \leftarrow$ `comm_row.Allreduce`(Ax_{part_temp})

    $b_{part} \leftarrow Ax_{part} - b_{part}$

    $r_{part\_temp} \leftarrow A^T_{part} b_{part}$

    $r_{part} \leftarrow$ `comm_col.Allreduce`(r_{part_temp}); $r_{part} \leftarrow r_{part} + \alpha x_{part}$

    $\sigma^2_{r\,part} \leftarrow 0$

  else

    $r_{part} \leftarrow r_{part} - \dfrac{q_{part}}{\text{scalar\_product\_pq}}$

    $\sigma^2_{r\,part} \leftarrow \sigma^2_{r\,part} + \dfrac{{q_{part}}^{\circ 2}}{(\text{scalar\_product\_pq})^2}$

  end

  $\text{scalar\_product}_{temp} \leftarrow (r_{part}, r_{part})$

  $\text{scalar\_product\_rr} \leftarrow$ `comm_row.Allreduce`$(\text{scalar\_product}_{temp})$

  $\text{criterion}_{temp} \leftarrow \dfrac{\Delta^2 \sum\limits_n (\sigma^2_{r\,part})_n}{\text{scalar\_product\_rr}}$

  $\text{criterion} \leftarrow$ `comm_row.Allreduce`$(\text{criterion}_{temp})$

  if $\text{criterion} \geqslant 1$ **then**

    return x_{part}

  end

  $p_{part} \leftarrow p_{part} + \dfrac{r_{part}}{\text{scalar\_product\_rr}}$

  $Ap_{part\_temp} \leftarrow A_{part} p_{part}$

  $Ap_{part} \leftarrow$ `comm_row.Allreduce`(Ap_{part_temp})

  $q_{part\_temp} \leftarrow A^T_{part}(Ap_{part})$

  $q_{part} \leftarrow$ `comm_col.Allreduce`(q_{part_temp}); $q_{part} \leftarrow q_{part} + \alpha p_{part}$

  $\text{scalar\_product}_{temp} \leftarrow (p_{part}, q_{part})$

  $\text{scalar\_product\_pq} \leftarrow$ `comm_row.Allreduce`$(\text{scalar\_product}_{temp})$

  $x_{part} \leftarrow x_{part} - \dfrac{p_{part}}{\text{scalar\_product\_pq}}$; $s \leftarrow s + 1$

end

Программная реализация функции, которая реализует алгоритм 4.3 для решения системы (4.3), представлена в листинге 4.3.

Листинг 4.3: Python-код («наивный») для параллельной версии усовершенствованного алгоритма.

```
def conjugate_gradient_method(A_part, b_part, x_part, alpha,
                              comm_row, comm_col, N) :

    N_part = size(x_part);  M_part = size(b_part)

    x_classic_part = None
    delta = finfo(float64).eps

    s = 1
    p_part = zeros(N_part, dtype=float64)

    while True :

        if s == 1 :
            Ax_part_temp = dot(A_part, x_part)
            Ax_part = empty(M_part, dtype=float64)
            comm_row.Allreduce([Ax_part_temp, M_part, MPI.DOUBLE],
                               [Ax_part, M_part, MPI.DOUBLE],
                               op=MPI.SUM)
            b_part = Ax_part - b_part
            r_part_temp = dot(A_part.T, b_part)
            r_part = empty(N_part, dtype=float64)
            comm_col.Allreduce([r_part_temp, N_part, MPI.DOUBLE],
                               [r_part, N_part, MPI.DOUBLE],
                               op=MPI.SUM)
            r_part = r_part + alpha*x_part

            sigma2_r_part = zeros(N_part, dtype=float64)
        else :
            r_part = r_part - q_part/scalar_product_pq

            sigma2_r_part = sigma2_r_part + \
                            q_part**2/scalar_product_pq**2

        scalar_product_temp = array(dot(r_part, r_part),
                                    dtype=float64)
        scalar_product_rr = array(0, dtype=float64)
        comm_row.Allreduce([scalar_product_temp, 1, MPI.DOUBLE],
```

```
                            [scalar_product_rr, 1, MPI.DOUBLE],
                            op=MPI.SUM)

        criterion_temp = array(delta**2*sum(sigma2_r_part)/
                               scalar_product_rr, dtype=float64)
        criterion = array(0, dtype=float64)
        comm_row.Allreduce([criterion_temp, 1, MPI.DOUBLE],
                            [criterion, 1, MPI.DOUBLE],
                            op=MPI.SUM)
        if criterion >= 1 :
            return x_part, s, x_classic_part

        p_part = p_part + r_part/scalar_product_rr

        Ap_part_temp = dot(A_part, p_part)
        Ap_part = empty(M_part, dtype=float64)
        comm_row.Allreduce([Ap_part_temp, M_part, MPI.DOUBLE],
                            [Ap_part, M_part, MPI.DOUBLE],
                            op=MPI.SUM)
        q_part_temp = dot(A_part.T, Ap_part)
        q_part = empty(N_part, dtype=float64)
        comm_col.Allreduce([q_part_temp, N_part, MPI.DOUBLE],
                            [q_part, N_part, MPI.DOUBLE],
                            op=MPI.SUM)
        q_part = q_part + alpha*p_part

        scalar_product_temp = array(dot(p_part, q_part),
                                    dtype=float64)
        scalar_product_pq = array(0, dtype=float64)
        comm_row.Allreduce([scalar_product_temp, 1, MPI.DOUBLE],
                            [scalar_product_pq, 1, MPI.DOUBLE],
                            op=MPI.SUM)
        x_part = x_part - p_part/scalar_product_pq

        s = s + 1

        if s == N + 1 :
            x_classic_part = x_part.copy()
```

Аналогично последовательной версии программы по итогу работы этой функции каждый MPI-процесс возвращает 1) массив `x_part`, который содержит часть решения системы (4.3), найденное по усовершенствованному алгоритму, и который при объединении с аналогичными данными с остальных процессов

даст массив `x`, содержащий полное решение системы (4.3), 2) число итераций s, которое потребовалось сделать алгоритму чтобы найти приближённое решение, 3) массив `x_classic_part`, который содержит часть решения, найденное с помощью «классического» алгоритма (этот массив содержит такие данные только в том случае, когда число выполненных алгоритмом итераций $s \geqslant N + 1$).

Ниже приводятся подробные пояснения программной реализации этой функции.

В строке 4 определяются размеры массивов `x_part` и `b_part`, в которых хранятся части x_{part} и b_{part} векторов x и b. Эти размеры определяют также и размеры части $A_{part_{(\cdot)}}$ матрицы A, которая хранится в массиве `A_part`.

В строке 6 формально инициализируется объект `x_classic_part`, который будет содержать часть решения, найденное с помощью «классического» алгоритма в том случае, если число выполненных алгоритмом итераций превысит $s = N$).

В строке 7 определяется Δ — относительная ошибка машинного округления. При этом здесь учитывается, что предельно возможная точность при вычислениях на Python — двойная (`float64`).

В строке 9 устанавливается начальное значение счётчика итераций метода сопряжённых градиентов.

В строке 10 массив `p_part` заполняется нулями согласно алгоритму метода сопряжённых градиентов.

В строке 12 начинается основной цикл метода сопряжённых градиентов.

В строках 15–26 реализуется параллельная версия операции $A^T(Ax - b) + \alpha x$. Строка 15 содержит именно те вычисления, которые осуществляются параллельно на всех процессах исходного коммуникатора `comm` (или, что в данном контексте то же самое, `comm_cart`). Операцию `dot(A_part, x_part)` каждый процесс выполняет со своей частью матрицы $A_{part_{(\cdot)}}$ и своей частью вектора $x_{part_{(\cdot)}}$. В результате этих действий на каждом процессе получается массив `Ax_part_temp`, элементы которого образуют одно из слагаемых части вектора, который является результатом операции Ax. Если сложить все эти массивы вдоль одной из строк сетки процессов, то получится массив `Ax_part`, который будет содержать часть соответствующего вектора. Этот массив будет нужен на всех процессах соответствующей строки сетки процессов (то есть на всех процессах соответствующего коммуникатора `comm_row`). Поэтому в строке 16 под этот массив выделяется место в памяти. Для сложения значений

массивов `Ax_part_temp` вдоль каждой строки сетки процессов используется функцию `Allreduce()` (см. строки 17–19). При этом эта функция коллективного взаимодействия процессов на каждой группе процессов, входящих в свой коммуникатор `comm_row`, работает независимо от аналогичных функций, работающих в других коммуникаторах `comm_row`. То есть пересылка данных вдоль каждой строки сетки процессов осуществляется параллельно с пересылкой аналогичных данных вдоль других строк сетки процессов! Результатом работы этой функции на каждой строке сетки процессов будет один и тот же массив `Ax_part`, содержащийся на каждом процессе соответствующего коммуникатора `comm_row`. Далее в строке 20 из полученного массива вычитается массив `b_part`. Эта операция реализуется параллельно, потому что она каждым процессом совершается над своими данными. Именно по этой причине массив `b` был распределён по частям по всем процессам способом, указанным на рис. 4.1. Наконец, полученный на каждом процессе массив `b_part` содержит часть вектора, который надо умножить слева на соответствующую часть транспонированной матрицы. Эти действия каждым процессом также осуществляются независимо от действий других процессов, то есть параллельно. В результате этих действий в строке 21 на каждом процессе получается массив `r_part_temp`, элементы которого образуют одно из слагаемых части вектора, который является результатом операции $A^T \cdot (A \cdot x - b)$. Если сложить все эти массивы вдоль одного из столбцов сетки процессов, то получится массив `r_part`, который будет содержать часть соответствующего вектора. Этот массив будет нужен на всех процессах соответствующего столбца сетки процессов (то есть на всех процессах соответствующего коммуникатора `comm_col`). Для сложения значений массивов `r_part_temp` вдоль каждого столбца сетки процессов используется функция `Allreduce()` (см. строки 23–25). При этом эта функция коллективного взаимодействия процессов на каждой группе процессов, входящих в свой коммуникатор `comm_col`, работает независимо от аналогичных функций, работающих в других коммуникаторах `comm_col`. То есть пересылка данных вдоль каждого столбца сетки процессов осуществляется параллельно с пересылкой аналогичных данных вдоль других столбцов сетки процессов! Результатом работы этой функции будет один и тот же массив `r_part`, содержащийся на каждом процессе коммуникатора `comm_col`. Затем в строке 26 к полученному массиву прибавляется массив `alpha*x_part`. Эта операция реализуется параллельно, потому что она каждым процессом совершается над своими данными. То есть параллель-

ная версия операции $A^T(Ax - b) + \alpha x$ реализована. Результирующий вектор r хранится по частям $r_{part_{(.)}}$ на всех процессах коммуникатора `comm` в массиве `r_part` (распределение частей $r_{part_{(.)}}$ вектора r по сетке процессов аналогично распределению частей вектора x и соответствует рис. 4.1).

В строке 28 реализуется формула $\sigma_r^2 = 0$. Формально, эта операция реализуется параллельно на всех процессах, так как на каждом процессе содержится своя часть $\sigma^2_{rpart_{(.)}}$ вектора σ_r^2. Соответствующие значения сохраняются в массиве `sigma2_r_part`.

В строке 30 реализуется параллельная версия операции $r - q/(p,q)$. Так как это действие выполняется начиная со второй операции, то предполагается, что значение (p,q), содержащееся в массиве `scalar_product_pq`, вычислено на предыдущей итерации и содержится на всех процессах коммуникатора `comm_cart`. Деление вектора на число и вычитание векторов реализуется на каждом процессе независимо от действий других процессов, то есть параллельно. В результате на каждом процессе коммуникатора `comm_cart` в массиве `r_part` будет содержаться своё пересчитанное значение массива `r_part` (распределение частей $r_{part_{(.)}}$ вектора r по сетке процессов аналогично распределению частей вектора x и соответствует рис. 4.1).

В строках 32–33 аналогичным образом реализуется параллельная версия операции $\sigma_r^2 = \sigma_r^2 + q^{\circ 2}/(p,q)^2$.

В строках 35–40 реализуется параллельное вычисление скалярного произведения (r,r). Строки 35–36 содержат именно те вычисления, которые осуществляются параллельно на всех процессах исходного коммуникатора `comm_cart`. Операцию `dot(r_part, r_part)` каждый процесс выполняет со своими частями вектора $r_{part_{(.)}}$. Если сложить все полученные массивы `scalar_product_temp` вдоль одной из строк сетки процессов, то получится массив `scalar_product_rr`, который будет содержать результат операции (r,r). Поэтому в строке 37 под этот массив выделяется место в памяти. Для сложения значений массивов `scalar_product_temp` вдоль каждой строки сетки процессов используется функция `Allreduce()` (см. строки 38–40). При этом эта функция коллективного взаимодействия процессов на каждой группе процессов, входящих в свой коммуникатор `comm_row`, работает независимо от аналогичных функций, работающих в других коммуникаторах `comm_row`. В результате на каждом процессе коммуникатора `comm_cart` в массиве `scalar_product_rr` будет содержаться одно и тоже число, являющееся результатом операции (r,r).

Замечание. Отметим, что указанным способом результат скалярного произведения (r, r) можно получить, оперируя с данными любой строки сетки процессов. В предлагаемой реализации, процессы каждой строки сетки процессов дублируют вычислительную работу, которую уже делают процессы остальных строк сетки процессов. При этом, используя функцию `Reduce()` среди процессов произвольного коммуникатора `comm_row` можно было бы получить результат вычисления скалярного произведения, например, на нулевом процессе соответствующего коммуникатора `comm_row`. Затем с помощью функции `Bcast()` это значение могло бы быть разослано с соответствующего процесса по всем процессам коммуникатора `comm_cart`. В этом случае большинство процессов не выполняло бы лишнюю работу. Но это не давало бы повышения эффективности программной реализации, так как эти процессы сначала простаивали бы, а затем тратили бы дополнительное время на реализацию функции `Bcast()` среди процессов коммуникатора `comm_cart`.

В строках 42–47 реализуется параллельное вычисление $\Delta^2 \sum\limits_{n=1}^{N} \left(\sigma_r^2\right)_n \Big/ (r,r)^2$. Операцию `delta**2*sum(sigma2_r_part)/scalar_product_rr` каждый процесс выполняет со своими частями $\sigma^2_{part_{(\cdot)}}$ вектора σ^2. Если сложить все полученные массивы `criterion_temp` вдоль одной из строк сетки процессов, то получится массив `criterion`, который будет содержать итоговый результат. Поэтому в строке 44 под этот массив выделяется место в памяти. Для сложения значений массивов `criterion_temp` вдоль каждой строки сетки процессов используется функция `Allreduce()` (см. строки 45–47). При этом эта функция коллективного взаимодействия процессов на каждой группе процессов, входящих в свой коммуникатор `comm_row`, работает независимо от аналогичных функций, работающих в других коммуникаторах `comm_row`. В результате на каждом процессе коммуникатора `comm_cart` в массиве `criterion` будет содержаться одно и тоже число, являющееся результатом указанной в начале абзаца операции.

В строках 48–49 осуществляется проверка критерия прекращения итерационного процесса.

В случае его завершения функция `conjugate_gradient_method` прекращает работу и возвращает (строка 49) массивы, содержащие искомое решение, число затраченных итераций и, для сравнения, решение найденное с помощью «классического» критерия прекращения итерационного процесса.

В строке 51 реализуется параллельная версия операции $p + r/(r, r)$. Все действия аналогичны действиям в строке 30. В результате на каждом процессе коммуникатора `comm_cart` в массиве `p_part` будет содержаться своё пересчитанное значение вектора $p_{part_{(.)}}$ (распределение частей $p_{part_{(.)}}$ вектора p по сетке процессов аналогично распределению частей вектора p и соответствует рис. 4.1).

В строках 53–63 реализуется параллельная версия операции $A^T(A\,p) + \alpha\,p$. Все действия в этих строках аналогичны действиям в строках 15–26. В результате на каждом процессе коммуникатора `comm_cart` в массиве `q_part` будет содержаться своё пересчитанное значение вектора $q_{part_{(.)}}$ (распределение частей $q_{part_{(.)}}$ вектора q по сетке процессов аналогично распределению частей вектора x и соответствует рис. 4.1).

В строках 65–70 реализуется параллельное вычисление скалярного произведения (p,q). Все действия в этих строках аналогичны действиям в строках 35–40. В результате на каждом процессе коммуникатора `comm_cart` в массиве `scalar_product_pq` будет содержаться одно и тоже число, являющееся результатом операции (p, q).

В строке 71 реализуется параллельная версия операции $x - p/(p, q)$. Все действия в этой строке аналогичны действиям в строке 26. В результате на каждом процессе коммуникатора `comm_cart` будет содержаться своё пересчитанное значение массива `x_part` (см. рис. 4.1).

В строке 73 счётчик цикла увеличивается на единицу.

В строках 75–76 сохраняется решение, найденное по «классическому» критерию

Затем все описанные действия повторяются.

Таким образом, в предложенной программной реализации распараллелены все вычислительные операции. При этом следует отметить ещё раз, что именно параллельные вычисления содержатся в строках 15, 20, 21, 26, 30, 32, 35, 42, 51, 53, 58, 63, 65 и 71. Остальные строки кода организуют обмен сообщениями между различными MPI-процессами.

Необходимо ещё раз подчеркнуть, что особенность этой программной реализации заключается в том, что этапы счёта и этапы пересылки сообщений, содержащих результаты промежуточных вычислений, чётко разделены. Поэтому накладные расходы по обмену сообщениями между участвующими в вычислениях процессами будут негативно влиять на эффективность параллельной программной реализации.

Учёт возможностей стандарта MPI-3

Более эффективный параллельный вариант алгоритма 4.2, реализующего усовершенствованную версию метода сопряжённых градиентов для решения системы (4.3), может быть записан в виде псевдокода, оформленного как алгоритм 4.4.

Данный алгоритм использует возможности стандарта MPI-3: за счёт использования неблокирующих (асинхронных) операций (выделены в алгоритме 4.4 синим цветом) появляется возможность производить дополнительные вычисления, связанные с учётом накапливающихся ошибок машинного округления, одновременно с передачей самых больших сообщений в основном алгоритме, а также пересылать дополнительные сообщения на фоне вычислений основного алгоритма.

В этом алгоритме красным цветом, как и ранее, выделены строки, которые соответствуют действиям по реализации усовершенствованного критерия прекращения итерационного процесса. Если эти строки убрать и изменить условие $True$ на $s \leqslant N$, то получится классическая реализация метода сопряжённых градиентов для решения системы (4.3).

Коричневым цветом выделены операции, которые соответствуют учёту регуляризирующей добавки в функционале А. Н. Тихонова (3.2) для случая $\square \equiv L_2$, что эквивалентно $R \equiv E$ в (4.2) в результате дискретизации функционала. Комментарии касательно учёта априорной информации другого вида будут даны в параграфе 4.5.

Принципиальное отличие этого алгоритма от алгоритма 4.3 заключается в том, что часть этапов счёта и часть этапов пересылки сообщений, содержащих результаты промежуточных вычислений, совмещены. В частности, 1) этапы вычислений, связанные с учётом ошибок машинного округления, совмещены с этапами пересылки сообщений в основном алгоритме, и 2) этапы пересылки сообщений, содержащие результаты промежуточных вычислений, связанных с учётом ошибок машинного округления, совмещены с этапам вычислений в основном алгоритме. Поэтому накладные расходы по обмену сообщениями между участвующими в вычислениях процессами будут значительно меньше влиять на эффективность параллельной программной реализации по сравнению с алгоритмом 4.3.

Алгоритм 4.4: Псевдокод для эффективной параллельной версии усовершенствованного алгоритма.

Входные данные: A_{part}, b_{part}, $x_{part} \equiv x^{(1)}_{part}$, α, `comm_row`, `comm_col`, N

Результат: x_{part}

$s \leftarrow 1$; $p_{part} \leftarrow 0$

while *True* **do**

  if $s = 1$ **then**

    $Ax_{part\_temp} \leftarrow A_{part}x_{part}$

    $Ax_{part} \leftarrow$ `comm_row.Allreduce`(Ax_{part_temp})

    $b_{part} \leftarrow Ax_{part} - b_{part}$

    $r_{part\_temp} \leftarrow A^T_{part}b_{part}$

    $r_{part} \leftarrow$ `comm_col.Allreduce`(r_{part_temp}); $r_{part} \leftarrow r_{part} + \alpha\, x_{part}$

    $\sigma^2_{r\,part} \leftarrow 0$

  else

    $r_{part} \leftarrow r_{part} - \dfrac{q_{part}}{\text{scalar\_product\_pq}}$

  end

  $\text{scalar\_product}_{temp} \leftarrow (r_{part}, r_{part})$

  $\text{scalar\_product\_rr} \leftarrow$ `comm_row.Allreduce`$(\text{scalar\_product}_{temp})$

  $p_{part} \leftarrow p_{part} + \dfrac{r_{part}}{\text{scalar\_product\_rr}}$

  $Ap_{part\_temp} \leftarrow A_{part}p_{part}$

  $Ap_{part} \leftarrow$ `comm_row.Iallreduce`(Ap_{part_temp})

  if $s \geqslant 2$ **then**

    $\sigma^2_{r\,part} \leftarrow \sigma^2_{r\,part} + \dfrac{{q_{part}}^{\circ 2}}{(\text{scalar\_product\_pq})^2}$

  end

  $\text{criterion}_{temp} \leftarrow \dfrac{\Delta^2 \sum\limits_n \left(\sigma^2_{r\,part}\right)_n}{\text{scalar\_product\_rr}}$

  $\text{criterion} \leftarrow$ `comm_row.Iallreduce`$(\text{criterion}_{temp})$

  `Wait`(Ap_{part}) ; $q_{part\_temp} \leftarrow A^T_{part}(Ap_{part})$

  $q_{part} \leftarrow$ `comm_col.Allreduce`(q_{part_temp}); $q_{part} \leftarrow q_{part} + \alpha\, p_{part}$

  $\text{scalar\_product}_{temp} \leftarrow (p_{part}, q_{part})$

  $\text{scalar\_product\_pq} \leftarrow$ `comm_row.Allreduce`$(\text{scalar\_product}_{temp})$

  `Wait`(criterion)

  if criterion $\geqslant 1$ **then return** x_{part}

  $x_{part} \leftarrow x_{part} - \dfrac{p_{part}}{\text{scalar\_product\_pq}}$; $s \leftarrow s + 1$

end

Программная реализация функции, которая реализует алгоритм 4.4 для решения системы (4.3), представлена на листинге 4.4.

Листинг 4.4: Python-код для эффективной параллельной версии усовершенствованного алгоритма.

```
def conjugate_gradient_method(A_part, b_part, x_part, alpha,
                              comm_row, comm_col, N) :

    N_part = size(x_part);  M_part = size(b_part)

    x_classic_part = None
    delta = finfo(float64).eps

    requests = [MPI.Request() for i in range(2)]

    s = 1
    p_part = zeros(N_part, dtype=float64)

    while True :

        if s == 1 :
            Ax_part_temp = dot(A_part, x_part)
            Ax_part = empty(M_part, dtype=float64)
            comm_row.Allreduce([Ax_part_temp, M_part, MPI.DOUBLE],
                               [Ax_part, M_part, MPI.DOUBLE],
                               op=MPI.SUM)
            b_part = Ax_part - b_part
            r_part_temp = dot(A_part.T, b_part)
            r_part = empty(N_part, dtype=float64)
            comm_col.Allreduce([r_part_temp, N_part, MPI.DOUBLE],
                               [r_part, N_part, MPI.DOUBLE],
                               op=MPI.SUM)
            r_part = r_part + alpha*x_part

            sigma2_r_part = zeros(N_part, dtype=float64)
        else :
            r_part = r_part - q_part/scalar_product_pq

        scalar_product_temp = array(dot(r_part, r_part),
                                    dtype=float64)
        scalar_product_rr = array(0, dtype=float64)
        comm_row.Allreduce([scalar_product_temp, 1, MPI.DOUBLE],
                           [scalar_product_rr, 1, MPI.DOUBLE],
```

```
                            op=MPI.SUM)
        p_part = p_part + r_part/scalar_product_rr

        Ap_part_temp = dot(A_part, p_part)
        Ap_part = empty(M_part, dtype=float64)
        requests[0] = comm_row.Iallreduce(
                            [Ap_part_temp, M_part, MPI.DOUBLE],
                            [Ap_part, M_part, MPI.DOUBLE],
                            op=MPI.SUM)
        if s >= 2 :
            sigma2_r_part =  sigma2_r_part + \
                           q_part**2/scalar_product_pq**2
        criterion_temp = array(delta**2*sum(sigma2_r_part)/
                            scalar_product_rr, dtype=float64)
        criterion = array(0, dtype=float64)
        requests[1] = comm_row.Iallreduce(
                            [criterion_temp, 1, MPI.DOUBLE],
                            [criterion, 1, MPI.DOUBLE],
                            op=MPI.SUM)
        MPI.Request.Wait(requests[0], status=None)
        q_part_temp = dot(A_part.T, Ap_part)
        q_part = empty(N_part, dtype=float64)
        comm_col.Allreduce([q_part_temp, N_part, MPI.DOUBLE],
                           [q_part, N_part, MPI.DOUBLE],
                           op=MPI.SUM)
        q_part = q_part + alpha*p_part

        scalar_product_temp = array(dot(p_part, q_part),
                                    dtype=float64)
        scalar_product_pq = array(0, dtype=float64)
        comm_row.Allreduce([scalar_product_temp, 1, MPI.DOUBLE],
                           [scalar_product_pq, 1, MPI.DOUBLE],
                           op=MPI.SUM)

        MPI.Request.Wait(requests[1], status=None)
        if criterion >= 1 :
            return x_part, s, x_classic_part

        x_part = x_part - p_part/scalar_product_pq

        s = s + 1

        if s == N + 1 :
```

```
82                x_classic_part = x_part.copy()
```

Этот листинг содержит следующие принципиальные отличия от листинга 4.3.

В строке 9 добавлено создание списка объектов типа `request` по количеству вызовов неблокирующих функций по приёму/передаче сообщений, которые будут вызываться MPI-процессом.

В строках 43–44 функция `Allreduce()` заменена на её неблокирующий (асинхронный) аналог `Iallreduce()`, ассоциированный со значением `request[0]`. В строке 58 с помощью команды `Wait(request[0])` реализуется ожидание завершения этой асинхронной операции, чтобы быть уверенными, что к моменту выполнения строки 59, в которой используется для вычислений массив `Ap_part`, массив `Ap_part` уже содержит требуемые данные после обмена информацией (передачи сообщений) между процессами.

В строках 54–57 функция `Allreduce()` заменена на её неблокирующий (асинхронный) аналог `Iallreduce()`, ассоциированный со значением `request[1]`. В строке 73 с помощью функции `Wait(request[1])`реализуется ожидание завершения этой асинхронной операции, чтобы быть уверенными, что к моменту выполнения строки 74, в которой используется значение, содержащееся в массиве `criterion`), массив `criterion` уже содержит требуемые данные после передачи сообщений.

Таким образом, получилось: 1) производить дополнительные вычисления (строки 49–52), связанные с учётом накапливающихся ошибок машинного округления, одновременно с передачей одного из самых больших сообщений в основном алгоритме (строки 44-47), и 2) пересылать дополнительные сообщения (строки 54–57) на фоне вычислений основного алгоритма (строки 59–60 и 64–67).

А это означает, что по сравнению с «классическим» алгоритмом, нету никакого увеличения времени работы программы даже несмотря на добавление дополнительных вычислений и дополнительных операций по приёму/передаче сообщений. При этом необходимо отметить, что явно этот эффект будет сказывать при расчётах на суперкомпьютерных системах с достаточно большим числом вычислительных узлов. При этом эти узлы должны обладать хорошей локализацией в коммуникационной сети, что будет пояснено позже при обсуждении эффективности параллельных программных реализаций алгоритмов.

Учёт возможностей стандарта MPI-4

Программная реализация алгоритма 4.4 содержит коллективные операции взаимодействия между процессами с одинаковым набором аргументов, которые выполняются в цикле. Это означает что для повышения эффективности программной реализации можно использовать отложенные запросы на взаимодействие. Они дают возможность оптимизировать коммуникации, привязав список аргументов коммуникации к постоянному запросу коммуникации один раз, а затем повторно использовать сформированный запрос для инициирования и завершения обмена сообщениями.

Изменения в программной реализации 4.4 будут достаточно простыми. Необходимо выполнить следующую последовательность изменений.

1. До основного цикла `while` необходимо сформировать отложенные запросы на взаимодействия с помощью MPI-функций вида `request[] = Allreduce_init()` для всех функций коллективного взаимодействия процессов `Allreduce()` и `Iallreduce()`, которые многократно вызываются внутри цикла.

   Аргументами этих функций являются `numpy`-массивы, а именно фиксированные области памяти, ассоциированные с этими массивами. При инициализации сформированных запросов на взаимодействие все данные будут браться/записываться в эти области памяти, которые фиксируются один раз при формировании соответствующего отложенного запроса на взаимодействие. Поэтому также необходимо предварительно выделить место в памяти под все массивы, которые являются аргументами этих функций; а при последующих вычислениях следить за тем, чтобы соответствующие результаты вычислений сохранялись в правильных областях памяти.
2. Внутри цикла `while` заменить MPI-функции `Allreduce()` на последовательность MPI-функций `Start(request[])` «+» `Wait(request[])`.
3. Внутри цикла `while` заменить MPI-функцию `Iallreduce()` на MPI-функцию `Start(request[])`.

Соответствующая программная реализация функции, которая реализует алгоритм 4.4 для решения системы (4.3), представлена в листинге 4.5.

Листинг 4.5: Python-код для эффективной параллельной версии усовершенствованного алгоритма, использующий стандарт MPI-4.0.

```
def conjugate_gradient_method(A_part, b_part, x_part, alpha,
                              comm_row, comm_col, N) :

    N_part = size(x_part);  M_part = size(b_part)

    x_classic_part = None
    delta = finfo(float64).eps

    requests = [MPI.Request() for i in range(5)]

    scalar_product_temp = empty(1, dtype=float64)
    scalar_product_rr = array(0, dtype=float64)
    scalar_product_pq = array(0, dtype=float64)

    Ap_part_temp = empty(M_part, dtype=float64)
    Ap_part = empty(M_part, dtype=float64)

    q_part_temp = empty(N_part, dtype=float64)
    q_part = empty(N_part, dtype=float64)

    criterion_temp = empty(1, dtype=float64)
    criterion = array(0, dtype=float64)

    requests[0] = comm_row.Allreduce_init(
                            [scalar_product_temp, 1, MPI.DOUBLE],
                            [scalar_product_rr, 1, MPI.DOUBLE],
                            op=MPI.SUM)
    requests[1] = comm_row.Allreduce_init(
                            [Ap_part_temp, M_part, MPI.DOUBLE],
                            [Ap_part, M_part, MPI.DOUBLE],
                            op=MPI.SUM)
    requests[2] = comm_row.Allreduce_init(
                            [criterion_temp, 1, MPI.DOUBLE],
                            [criterion, 1, MPI.DOUBLE],
                            op=MPI.SUM)
    requests[3] = comm_col.Allreduce_init(
                            [q_part_temp, N_part, MPI.DOUBLE],
                            [q_part, N_part, MPI.DOUBLE],
                            op=MPI.SUM)
    requests[4] = comm_row.Allreduce_init(
                            [scalar_product_temp, 1, MPI.DOUBLE],
```

```
                        [scalar_product_pq, 1, MPI.DOUBLE],
                        op=MPI.SUM)

    s = 1
    p_part = zeros(N_part, dtype=float64)

    while True :

        if s == 1 :
            Ax_part_temp = dot(A_part, x_part)
            Ax_part = empty(M_part, dtype=float64)
            comm_row.Allreduce([Ax_part_temp, M_part, MPI.DOUBLE],
                               [Ax_part, M_part, MPI.DOUBLE],
                               op=MPI.SUM)
            b_part = Ax_part - b_part
            r_part_temp = dot(A_part.T, b_part)
            r_part = empty(N_part, dtype=float64)
            comm_col.Allreduce([r_part_temp, N_part, MPI.DOUBLE],
                               [r_part, N_part, MPI.DOUBLE],
                               op=MPI.SUM)
            r_part = r_part + alpha*x_part

            sigma2_r_part = zeros(N_part, dtype=float64)
        else :
            r_part = r_part - q_part/scalar_product_pq

        scalar_product_temp[:] = array(dot(r_part, r_part),
                                       dtype=float64)
        MPI.Prequest.Start(requests[0])
        MPI.Request.Wait(requests[0], status=None)
        p_part = p_part + r_part/scalar_product_rr

        Ap_part_temp[:] = dot(A_part, p_part)
        MPI.Prequest.Start(requests[1])
        if s >= 2 :
            sigma2_r_part =  sigma2_r_part + \
                            q_part**2/scalar_product_pq**2
        criterion_temp[:] = array(delta**2*sum(sigma2_r_part)/
                                  scalar_product_rr, dtype=float64)
        MPI.Prequest.Start(requests[2])
        MPI.Request.Wait(requests[1], status=None)
        q_part_temp[:] = dot(A_part.T, Ap_part)
        MPI.Prequest.Start(requests[3])
```

```
        MPI.Request.Wait(requests[3], status=None)
        q_part[:] = q_part[:] + alpha*p_part

        scalar_product_temp[:] = array(dot(p_part, q_part),
                                       dtype=float64)
        MPI.Prequest.Start(requests[4])
        MPI.Request.Wait(requests[4], status=None)

        MPI.Request.Wait(requests[2], status=None)
        if criterion >= 1 :
            return x_part, s, x_classic_part

        x_part = x_part - p_part/scalar_product_pq

        s = s + 1

        if s == N + 1 :
            x_classic_part = x_part.copy()
```

4.2.4 Оценка эффективности и масштабируемости программной реализации параллельного алгоритма

В этом подпараграфе обсуждается эффективность и масштабируемость предложенных программных реализаций параллельного алгоритма. Сначала будет приведено описание тестового примера и использованной для тестовых вычислений многопроцессорной системы, а затем будут прокомментированы результаты исследования предложенных программных реализаций на предмет наличия сильной и слабой масштабируемости.

Описание тестового примера

Для тестирования эффективности программных реализаций параллельного алгоритма использовался вычислительный раздел «compute» суперкомпьютера «Ломоносов-2» [96] научно-исследовательского вычисли-

тельного центра МГУ имени М. В. Ломоносова. Каждый вычислительный узел раздела «compute» содержит 14-ядерный процессор Intel Xeon E5-2697 v3 2.60GHz с 64 GB оперативной памяти (4.5 GB на ядро) и видеокарту Tesla K40s с объёмом видеопамяти 11.56 GB.

В качестве тестовой задачи была выбрана задача с $M = 90\,000$, $N = 70\,000$. Выбор таких параметров обусловлен тем, что для весомости численных экспериментов необходимо провести расчёты для системы (4.1) с как можно большей матрицей A, но при этом чтобы эта матрица помещалась в оперативную память одного вычислительного узла (в противном случае не получится засечь время работы T_1 последовательной версии программы). В случае выбранных параметров для хранения матрицы системы A необходимо $N \times M \times 8$ байт $= 46.93$ GB. Для тестовых расчётов использовалось ограничение в 400 итераций метода сопряжённых градиентов. В этом случае время работы T_1 последовательной версии функции `conjugate_gradient_method()` на 1 вычислительном узле составило $\sim$755 секунд (в случае выполнения N итераций при использовании «классического» критерия прекращения итерационного процесса это время увеличилось бы до $\sim$1.5 дней).

Параллельная версия программы запускалась с привязкой каждого MPI-процесса ровно к одному вычислительному узлу. При этом необходимо напомнить особенность функции `dot()` из пакета `numpy`, которая выполняет основной объём вычислений при реализации алгоритма: эта функция реализована на языке программирования Fortran через пакет Intel MKL и при расчётах автоматически использует многопоточность, если программа запущена на многоядерном процессоре — все ядра процессора задействуются за счёт использования технологии параллельного программирования OpenMP. Таким образом, привязывая каждый MPI-процесс к отдельному вычислительному узлу, автоматически реализовывалась технология гибридного параллельного программирования MPI+OpenMP. В

Использовались пакеты 1) `numpy` версии 1.24.3, 2) `mpi4py` версии 4.0.0.dev0, 3) `mpich` версии 4.1.1.

Программа запускалась на числе процессов $n \equiv$ `numprocs`, которое является квадратом натурального числа (1, 4, 9, 16, 25, 36, 49 и т.д.). Это связано с тем, что в тестовых программах использовалась виртуальная топология процессов с двумерной декартовой сеткой с параметрами `num_row` $=$ `num_col` $= \sqrt{n}$. Было засечено время работы T_n параллельной версии функции

`conjugate_gradient_method()` для каждого значения n из указанного числа процессов. Используя оценку времени работы T_1 последовательной версии функции `conjugate_gradient_method()`, по формуле $S_n = \dfrac{T_1}{T_n}$ было вычислено ускорение S_n, а затем эффективность $E_n = \dfrac{S_n}{n}$ программной реализации. На рис. 4.8 представлены графики эффективности распараллеливания E_n в зависимости от числа вычислительных узлов n, участвовавших в вычислениях. Представлены графики таких зависимостей для трёх программных реализаций алгоритма, использующих различные стандарты MPI. Графики полностью соответствуют заявленным ранее ожиданиям. При этом видно неоспоримое преимущество использования стандарта MPI-4.

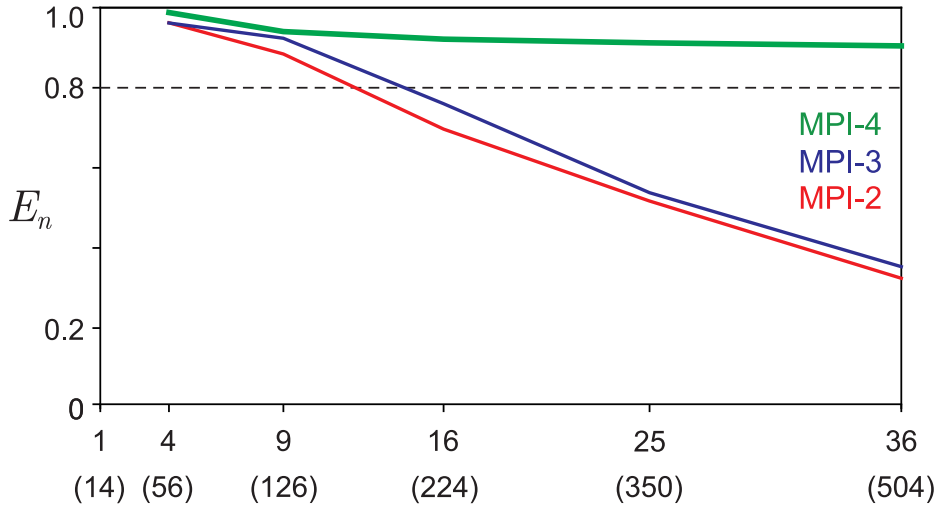


Рисунок 4.8 — Графики эффективности распараллеливания в зависимости от числа вычислительных узлов для различных программных реализаций параллельного алгоритма. Сильная масштабируемость присутствует только для программной реализации, использующей стандарт MPI-4.

При этом важно отметить следующее. Дизайн параллельного алгоритма предполагает, что MPI-процессы, участвующие в вычислениях, образуют двумерную декартову сетку (см. рис. 4.2, 4.3 и 4.5). MPI-функция `Allreduce()` (либо её неблокирующий аналог `Iallreduce()`/`Allreduce_init()`), использованная в параллельной программной реализации, будет работать эффективнее всего, если эта двумерная декартова сетка процессов будет периодической по обоим направлениям. Поэтому для тестовых расчётов использовалась соответствующая виртуальная топология процессов типа «двумерный тор» `comm` := `comm_cart`, где `comm_cart`:

```
comm_cart = comm.Create_cart(dims=(num_row, num_col),
                             periods=(True, True),
                             reorder=True)
```

Однако не всегда виртуальная топология MPI-процессов может быть эффективно отображена на реальную физическую коммуникационную сеть, связывающую вычислительные узлы суперкомпьютерной системы. Поэтому возможны ситуации, в которых соседние по виртуальной топологии MPI-процессы работают на далеко расположенных друг от друга в коммуникационной среде вычислительных узлах. Это будет приводить к существенному увеличению времени передачи сообщений между некоторыми MPI-процессами (см. рис. 4.9), в том числе и из-за возможной перегрузки коммуникационной сети. А это, как следствие, может приводить к катастрофическому падению эффективности.

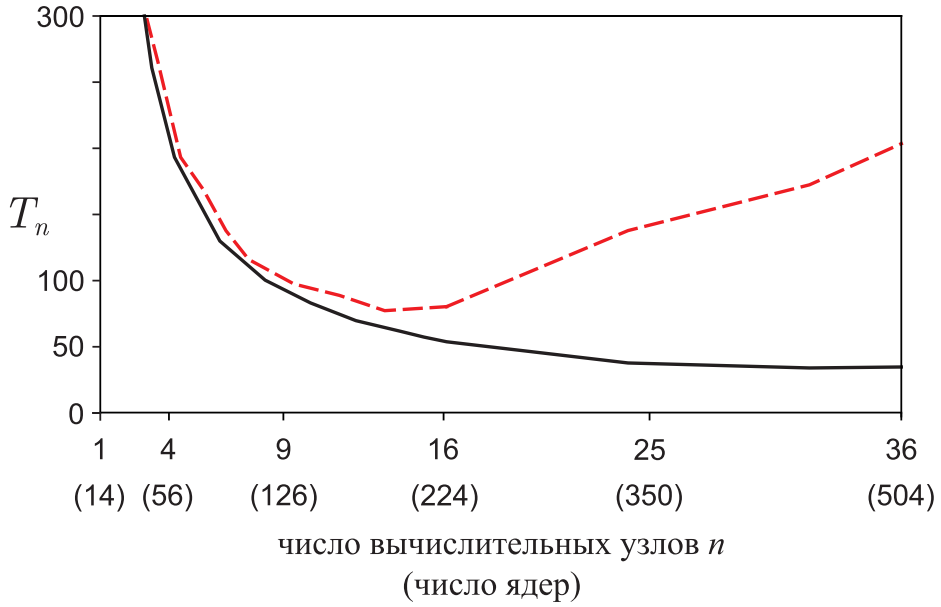


Рисунок 4.9 — Графики зависимости времени работы программы в случае хорошей локализации вычислительных узлов в коммуникационной сети (сплошная линия) и плохой локализации (красная пунктирная линия).

Отметим, что результаты, приведённые на рис. 4.8 соответствуют усреднённым значениям по серии запусков программ на произвольных распределениях MPI-процессов по вычислительным узлам суперкомпьютера «Ломоносов-2». В случае использования для вычислений узлов только с хорошей локализацией в коммуникационной среде могут быть получены более хорошие результаты для эффективности программной реализации по сравнению с результатами, представленными на рис. 4.8.

Сильная и слабая масштабируемость программной реализации

Хорошо известно, что слабым местом параллельных алгоритмов решения систем линейных алгебраических уравнений с плотно заполненной матрицей является относительно плохая сильная масштабируемость многих возможных программных реализаций (что видно из рисунка 4.8).

Это можно пояснить следующим образом на примере параллельного алгоритма умножения матрицы на вектор (см. рис. 4.2), который является основной операцией в рассмотренных алгоритмах 4.3 и 4.4.

В параллельных вычислениях принимают участие все MPI-процессы, и, как следствие, все вычисления ускоряются в n раз ($n \equiv$ `numprocs`). При этом объём передаваемых сообщений пропорционален $\dfrac{1}{\sqrt{n}}$, а значит, он уменьшается с увеличением числа участвующих в вычислениях процессов. Но при увеличении числа процессов, участвующих в вычислениях, увеличивается число обменов сообщениями между процессами вспомогательных коммуникаторов, которое пропорционально $\log_2 \sqrt{n}$.

Получается, что с ростом числа n участвующих в вычислениях процессов время счёта пропорционально $\dfrac{1}{n}$, а время на обмен сообщениями между процессами пропорционально $\dfrac{\log_2 \sqrt{n}}{\sqrt{n}}$.

Таким образом, начиная с какого-то конкретного числа процессов (это число зависит от вычислительных размеров задачи и от конфигурации многопроцессорной системы) доля накладных расходов по обмену сообщениями между процессами начнёт увеличиваться. А это, как следствие, неизбежно будет приводить к ухудшению эффективности программной реализации с увеличением числа участвующих в вычислениях процессов.

Поэтому были проведены дополнительные расчёты для тестирования предложенных программных реализаций на наличие слабой масштабируемости. Наличие слабой масштабируемости подразумевает сохранение эффективности с ростом числа участвующих в вычислениях процессов при сохранении объёма вычислительной работы, которую выполняет каждый процесс.

Таким образом, тест на слабую масштабируемость выполнялся путём одновременного увеличения размера задачи ($M := \sqrt[3]{n}\,M$, $N := \sqrt[3]{n}\,N$ и $s := \sqrt[3]{n}\,s$) и числа n участвующих в вычислениях процессов. На рис. 4.10 представлены

графики эффективности распараллеливания $E_n = \dfrac{T_1}{T_n}$ в зависимости от количества вычислительных узлов n, задействованных в вычислениях. Как видно из представленных графиков, предложенные программные реализации алгоритма обладают свойством слабой масштабируемости. При этом опять видно преимущество использования стандарта MPI-4.

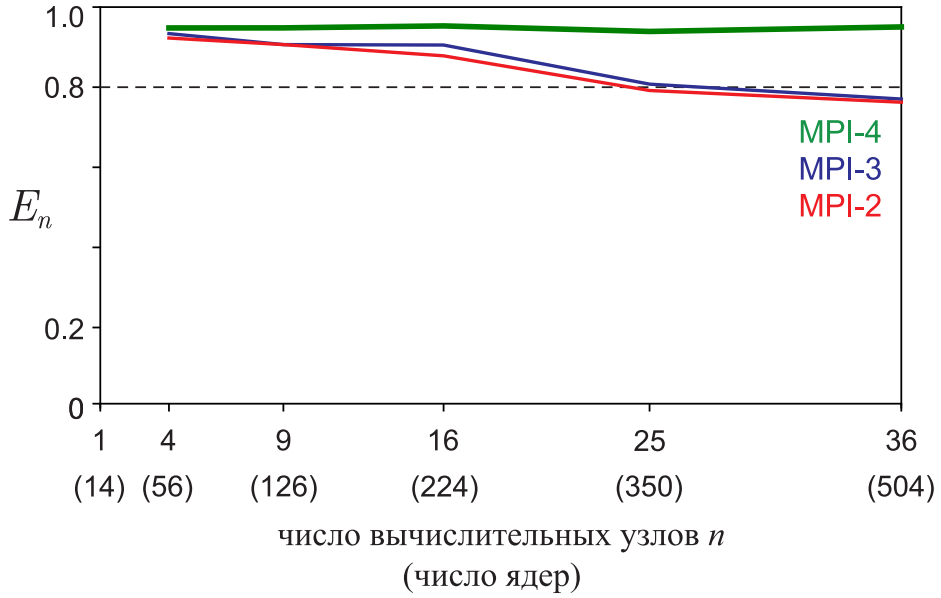


Рисунок 4.10 — График демонстрирующий наличие слабой масштабируемости у предложенных программных реализаций.

4.3 О модификации программ с использованием языков программирования C/C++/Fortran

Все рассмотренные в предыдущих параграфах примеры программ построены таким образом, чтобы их можно было легко переписать с использованием языков программирования C/C++/Fortran. Это связано с тем, что все вычислительные операции сводятся к базовым матричным операциям (умножение матриц, скалярное произведение векторов, сложение векторов), которые легко переносятся на указанные языки программирования, а все MPI-функции, используемые в программных реализациях Python, используют синтаксис, эквивалентный синтаксису соответствующих MPI-функций в языках программирования C/C++/Fortran.

Пример такой «эквивалентности» синтаксиса можно продемонстрировать на примере использования базовых функций передачи/приёма сообщений `Send()` и `Recv()`, которые реализую функциональный аналог (пусть и неэффективный) функции `Bcast()`.

Python-код имеет вид:

```
if rank == 0 :
    for k in range(1, numprocs) :
        comm.Send([N, 1, MPI.INT], dest=k, tag=0)
else :
    comm.Recv([N, 1, MPI.INT], source=0, tag=0, status=None)
```

Python позволяет не перечислять описание передаваемого/принимаемого сообщения в виде `[buf, count, datatype]`, ограничившись только первым аргументом списка `buf`, который может являться только `numpy`-массивом. В этом случае Python автоматически определяет количество элементов этого массива и его тип данных. С учётом этих упрощений и наличию необязательных аргументов разбираемый сейчас кусочек программного кода можно записать и в такой эквивалентной форме:

```
if rank == 0 :
    for k in range(1, numprocs) :
        comm.Send([N, 1, MPI.INT], k, 0)
else :
    comm.Recv([N, 1, MPI.INT], 0, 0, status=None)
```

и даже в такой:

```
if rank == 0 :
    for k in range(1, numprocs) :
        comm.Send(N, k)
else :
    comm.Recv(N)
```

Однако в предыдущих параграфах специально не использовались никакие упрощения и не опускались необязательные аргументы. Именно при перечислении всех аргументов программная реализация на Python наиболее похожа на реализации на C/C++/Fortran. И в этом случае переписать соответствующую параллельную программу на C/C++/Fortran будет проще всего. Например, эквивалентный код на C:

```
if (rank == 0) {
    for (k = 1; i < numprocs; i++){
        MPI_Send(N, 1, MPI_INTEGER, k, 0, comm);
```

```
    }
}
else {
    MPI_Recv(N, 1, MPI_INTEGER, 0, 0, comm, &status);
}
```

Эквивалентный код на Fortran:

```
if (rank == 0) then
    do k=1,numprocs-1
        call MPI_Send(N, 1, MPI_INTEGER, k, 0, comm, ierr)
    enddo
else
    call MPI_Recv(N, 1, MPI_INTEGER, 0, 0, comm, status, ierr)
endif
```

Далее, в листинге 4.6 приводится переписанный на Fortran код программы 4.4, которая реализует параллельную версию функции `conjugate_gradient_method()`. Принципиальные отличия в реализации соответствующего алгоритма на Fortran заключаются только в том, что в начале этой программной реализации (строки 7–26) присутствует традиционное для Fortran объявление переменных и подключение библиотеки `mpi` (строка 5). Структура программы в строках 25–94 полностью соответствует действиям, реализованным в строках 6–82 Python-когда в листинге 4.4. Присутствуют только мелкие синтаксические отличия.

Листинг 4.6: Fortran-код для эффективной параллельной версии усовершенствованного алгоритма (эквивалентен листингу 4.4, написанному на Python).

```
1  subroutine conjugate_gradient_method(A_part,b_part,x_part,alpha, &
2                                       x_classic_part, &
3                                       M_part, N_part, &
4                                       comm_row, comm_col, N)
5      use mpi
6
7      implicit none
8
9      integer  :: M_part, N_part, N, s
10     real(16) :: A_part(M_part,N_part),x_part(N_part),b_part(M_part)
11     real(16) :: alpha
12     real(16) :: p_part(N_part)
13     real(16) :: Ax_part(M_part), Ax_part_temp(M_part)
14     real(16) :: Ap_part(M_part), Ap_part_temp(M_part)
15     real(16) :: r_part(N_part), r_part_temp(N_part)
```

```
    real(16) :: q_part(N_part), q_part_temp(N_part)
    real(16) :: sigma2_r_part(N_part)
    real(16) :: scalar_product_rr, scalar_product_pq
    real(16) :: scalar_product_temp
    real(16) :: criterion, criterion_temp
    integer  :: comm_row, comm_col
    integer  :: status(MPI_STATUS_SIZE)
    integer  :: ierr

    real(16) :: x_classic_part(N_part)
    real(16), parameter :: delta = epsilon(0q0)

    integer requests(0:1)

    s = 1
    p_part = 0.q0

    do while (.TRUE.)

        if (s == 1) then
            Ax_part_temp = matmul(A_part, x_part)
            call MPI_Allreduce(Ax_part_temp, Ax_part, &
                               M_part, MPI_REAL16, &
                               MPI_SUM, comm_row, ierr)
            b_part = Ax_part - b_part
            r_part_temp = matmul(transpose(A_part), b_part)
            call MPI_Allreduce(r_part_temp, r_part, &
                               N_part, MPI_REAL16, &
                               MPI_SUM, comm_col, ierr)
            r_part = r_part + alpha*x_part

            sigma2_r_part = 0.q0
        else
            r_part = r_part - q_part/scalar_product_pq
        end if

        scalar_product_temp = dot_product(r_part, r_part)
        call MPI_Allreduce(scalar_product_temp, scalar_product_rr,&
                           1, MPI_REAL16, &
                           MPI_SUM, comm_row, ierr)
        p_part = p_part + r_part/scalar_product_rr

        Ap_part_temp = matmul(A_part, p_part)
```

```
59          call MPI_Iallreduce(Ap_part_temp, Ap_part, &
60                              M_part, MPI_REAL16, &
61                              MPI_SUM, comm_row, requests(0), ierr)
62          if (s >= 2) then
63              sigma2_r_part = sigma2_r_part + &
64                              q_part**2/scalar_product_pq**2
65          end if
66          criterion_temp = delta**2*sum(sigma2_r_part)/ &
67                           scalar_product_rr
68          call MPI_Iallreduce(criterion_temp, criterion, &
69                              1, MPI_REAL16, &
70                              MPI_SUM, comm_row, requests(1), ierr)
71          call MPI_Wait(requests(0), status, ierr)
72          q_part_temp = matmul(transpose(A_part), Ap_part)
73          call MPI_Allreduce(q_part_temp, q_part, &
74                              N_part, MPI_REAL16, &
75                              MPI_SUM, comm_col, ierr)
76          q_part = q_part + alpha*p_part
77
78          scalar_product_temp = dot_product(p_part, q_part)
79          call MPI_Allreduce(scalar_product_temp, scalar_product_pq,&
80                              1, MPI_REAL16, &
81                              MPI_SUM, comm_row, ierr)
82
83          call MPI_Wait(requests(1), status, ierr)
84          if (criterion >= 1) then
85              return
86          end if
87
88          x_part = x_part - p_part/scalar_product_pq
89
90          s = s + 1
91
92          if (s == N + 1) then
93              x_classic_part = x_part
94          end if
95
96      end do
97
98  end subroutine
```

В листинге 4.7 приводится переписанный на Fortran код программы 4.5. Все отличия аналогичны паре Fortran–Python-листингов 4.4 и 4.6.

Замечание. Важно отметить, что листинг 4.7 здесь является лишь поясняющим примером, демонстрирующим, что Python-код и Fortran-код будут эквивалентны по своей структуре. Программа из листинга 4.7 не тестировалась. Это связано с тем, что MPI-функции, которые способны передавать массивы с четверной точностью (тип данных `real(16)`), на момент проведения диссертационного исследования были корректно реализованы только в компиляторе Intel Fortran, однако стандарт MPI-4(.1) в Intel MPI к началу 2023 года реализован ещё не был.

Листинг 4.7: Fortrn-код для эффективной параллельной версии усовершенствованного алгоритма, использующий стандарт MPI-4.0 (эквивалентен листингу 4.5, написанному на Python).

```
subroutine conjugate_gradient_method(A_part,b_part,x_part,alpha, &
                                     x_classic_part, &
                                     M_part, N_part, &
                                     comm_row, comm_col, N)

    use mpi

    implicit none

    integer  :: M_part, N_part, N, s
    real(16) :: A_part(M_part,N_part),x_part(N_part),b_part(M_part)
    real(16) :: alpha
    real(16) :: p_part(N_part)
    real(16) :: Ax_part(M_part), Ax_part_temp(M_part)
    real(16) :: Ap_part(M_part), Ap_part_temp(M_part)
    real(16) :: r_part(N_part), r_part_temp(N_part)
    real(16) :: q_part(N_part), q_part_temp(N_part)
    real(16) :: sigma2_r_part(N_part)
    real(16) :: scalar_product_rr, scalar_product_pq
    real(16) :: scalar_product_temp
    real(16) :: criterion, criterion_temp
    integer  :: comm_row, comm_col
    integer  :: status(MPI_STATUS_SIZE)
    integer  :: ierr

    real(16) :: x_classic_part(N_part)
    real(16), parameter :: delta = epsilon(0q0)

    integer :: requests(0:4)
    integer :: info
```

```
31
32      call MPI_Info_create(info, ierr)
33
34      call MPI_Allreduce_init(scalar_product_temp,scalar_product_rr,&
35                              1, MPI_REAL16, &
36                              MPI_SUM, comm_row, &
37                              info, requests(0), ierr)
38      call MPI_Allreduce_init(Ap_part_temp, Ap_part, &
39                              M_part, MPI_REAL16, &
40                              MPI_SUM, comm_row, &
41                              info, requests(1), ierr)
42      call MPI_Allreduce_init(criterion_temp, criterion, &
43                              1, MPI_REAL16, &
44                              MPI_SUM, comm_row, &
45                              info, requests(2), ierr)
46      call MPI_Allreduce_init(q_part_temp, q_part, &
47                              N_part, MPI_REAL16, &
48                              MPI_SUM, comm_col, &
49                              info, requests(3), ierr)
50      call MPI_Allreduce_init(scalar_product_temp,scalar_product_pq,&
51                              1, MPI_REAL16, &
52                              MPI_SUM, comm_row, &
53                              info, requests(4), ierr)
54
55      s = 1
56      p_part = 0.q0
57
58      do while (.TRUE.)
59
60          if (s == 1) then
61              Ax_part_temp = matmul(A_part, x_part)
62              call MPI_Allreduce(Ax_part_temp, Ax_part, &
63                                 M_part, MPI_REAL16, &
64                                 MPI_SUM, comm_row, ierr)
65              b_part = Ax_part - b_part
66              r_part_temp = matmul(transpose(A_part), b_part)
67              call MPI_Allreduce(r_part_temp, r_part, &
68                                 N_part, MPI_REAL16, &
69                                 MPI_SUM, comm_col, ierr)
70              r_part = r_part + alpha*x_part
71
72              sigma2_r_part = 0.q0
73          else
```

```
74                 r_part = r_part - q_part/scalar_product_pq
75         end if
76
77         scalar_product_temp = dot_product(r_part, r_part)
78         call MPI_Start(requests(0), ierr)
79         call MPI_Wait(requests(0), status, ierr)
80         p_part = p_part + r_part/scalar_product_rr
81
82         Ap_part_temp = matmul(A_part, p_part)
83         call MPI_Start(requests(1), ierr)
84         if (s >= 2) then
85             sigma2_r_part = sigma2_r_part + &
86                             q_part**2/scalar_product_pq**2
87         end if
88         criterion_temp = delta**2*sum(sigma2_r_part)/ &
89                          scalar_product_rr
90         call MPI_Start(requests(2), ierr)
91         call MPI_Wait(requests(1), status, ierr)
92         q_part_temp = matmul(transpose(A_part), Ap_part)
93         call MPI_Start(requests(3), ierr)
94         call MPI_Wait(requests(3), status, ierr)
95         q_part = q_part + alpha*p_part
96
97         scalar_product_temp = dot_product(p_part, q_part)
98         call MPI_Start(requests(4), ierr)
99         call MPI_Wait(requests(4), status, ierr)
100
101        call MPI_Wait(requests(3), status, ierr)
102        if (criterion >= 1) then
103            return
104        end if
105
106        x_part = x_part - p_part/scalar_product_pq
107
108        s = s + 1
109
110        if (s == N + 1) then
111            x_classic_part = x_part
112        end if
113
114    end do
115
116 end subroutine
```

4.4 Программные особенности использования при расчётах технологий параллельного программирования OpenMP и CUDA

До сих пор при рассмотрении программных реализаций предложенных алгоритмов не учитывалось то, какими техническими особенностями обладают многопроцессорные системы, которые используются для параллельных вычислений. Предполагалось только, что каждый вычислительный MPI-процесс запускается на своём вычислительном узле, и эти процессы взаимодействуют друг с другом посредством передачи сообщений с помощью технологии MPI (см. рис. 4.11). При этом предполагалось, что каждый вычислительный узел содер-

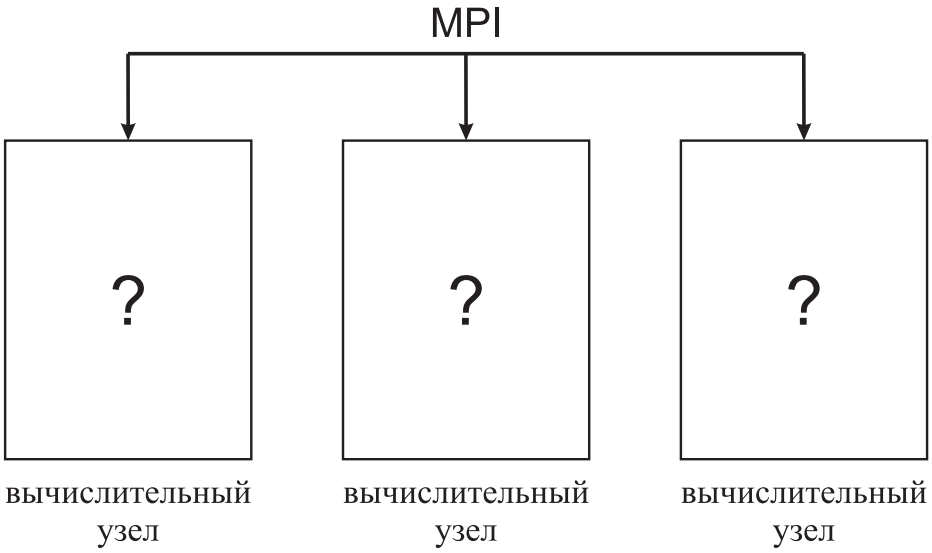


Рисунок 4.11 — Простейшая схема взаимодействия вычислительных узлов посредством технологии MPI.

жит центральный процессор и оперативную память. Таким образом, весь набор узлов представлял из себя систему с распределённой памятью (см. рис. 4.12). На практике всё обстоит гораздо сложнее. В нынешние времена каждый центральный процессор является скорее всего многоядерной системой, которая состоит из относительно большого числа вычислительных ядер (см. рис. 4.13). Таким образом, каждый такой вычислительный узел можно рассматривать как локальную вычислительную систему с общей памятью. Более того, каждый вычислительный узел может содержать ещё и графическую видеокарту, которая состоит из достаточно большого числа графический ядер и содержит общую для этих ядер видеопамять. Таким образом, локально видеокарта может восприниматься тоже как многопроцессорная система с общей памятью (см. рис. 4.14).

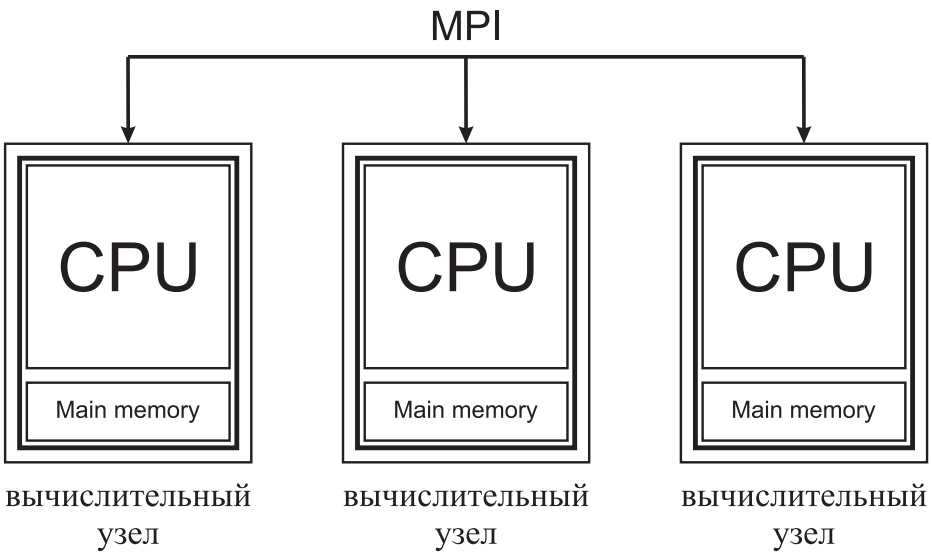


Рисунок 4.12 — Простейшая структура вычислительной системы с распределённой памятью.

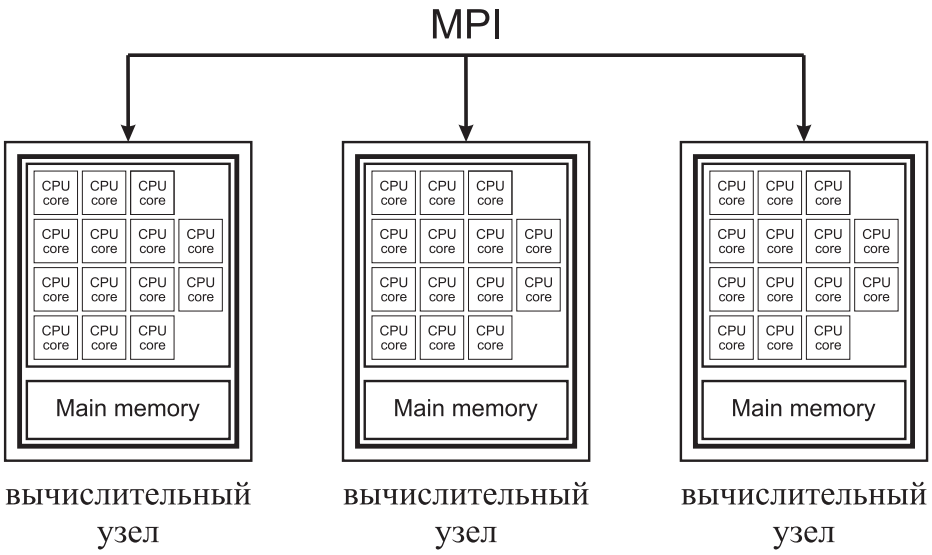


Рисунок 4.13 — Структура вычислительной системы с распределённой памятью в случае наличия многоядерных процессоров (на примере CPU вычислительных узлов суперкомпьютера «Ломоносов-2»).

Примером такой сложной вычислительной системы является суперкомпьютер «Ломоносов-2» [96], который использовался для большого числа расчётов, результаты которых представлены в этой работе. Основной вычислительный раздел суперкомпьютера «Ломоносов-2» содержит 1504 узла типа K40, на каждом из которых установлен 14-ядерный процессор Intel Xeon E5-2697 v3 2.60GHz с 64 GB оперативной памяти, а также видеокарта Tesla K40s с объёмом видеопамяти 11.56 GB.

Возникает вопрос о том, как эффективно использовать все эти вычислительные ресурсы. Для этого понадобятся следующие технологии.

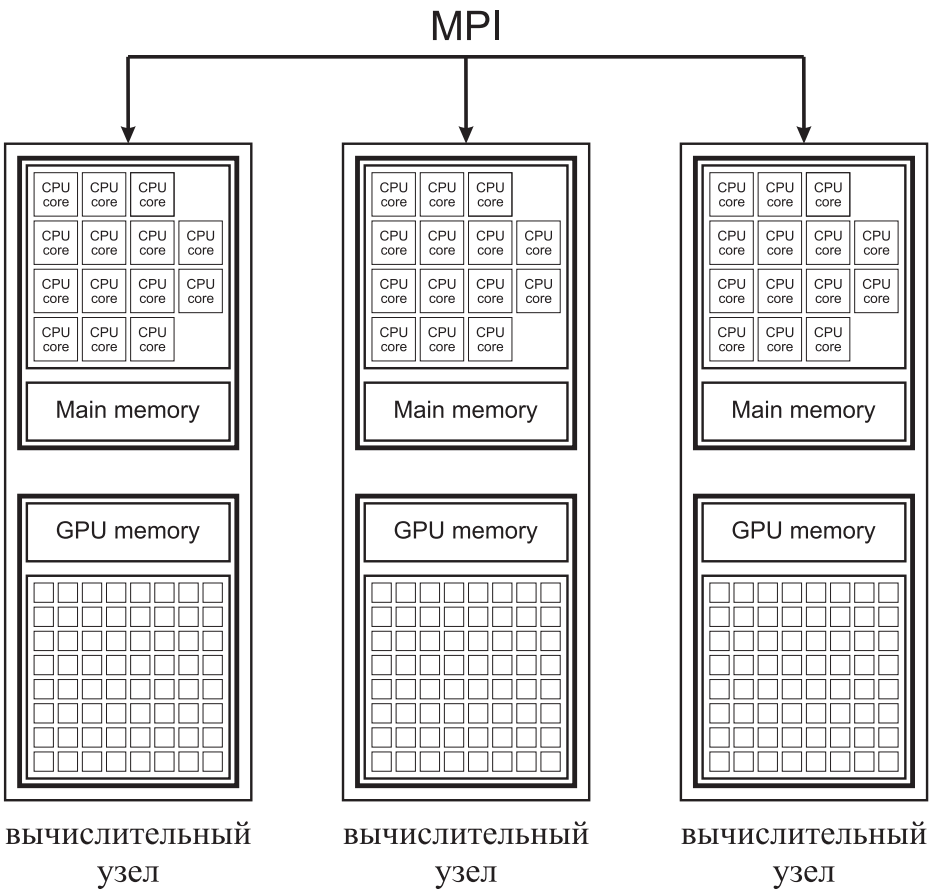


Рисунок 4.14 — Простейшая структура вычислительной системы с распределённой памятью.

- MPI ≡ Message Passing Interface — технология параллельного программирования для систем с распределённой памятью.
- OpenMP ≡ Open Multi-Processing — технология параллельного программинрования для систем с общей памятью, которая используется для написания параллельных программ под многоядерные процессоры.
- CUDA ≡ Compute Unified Device Architecture — технология программирования для сисстем с общей памятью, которая используется для написания параллельных программ под вычисления на видеокартах Nvidia (системы с общей памятью).

Если для расчётов доступен только один вычислительный узел, который содержит многоядерный процессор и видеокарту, то можно ограничиться использованием только технологии OpenMP, задействуя для вычислений все ядра многоядерного процессора, и/или технологией CUDA, задействуя все вычислительные ядра видеокарты. Если же таких узлов много, то нужно организовать взаимодействие между этими вычислительными узлами и сделать это можно с помощью технологии MPI. Таким образом, возможно писать программы,

которые задействуют все доступные вычислительные ресурсы. Но для этого надо использовать гибридные технологии параллельного программирования MPI+CUDA+OpenMP, которые позволяют для этих целей использовать все указанные технологии (MPI, CUDA, OpenMP). В результате вычисления будут распараллеливаться между вычислительными узлами с помощью технологии MPI, а на каждом узле своя часть вычислений будет опять же распараллеливаться между ядрами центрального процессора с помощью технологии OpenMP и/или между ядрами видеокарты с помощью технологии CUDA.

Достаточно часто при проведении реальных расчётов каждый MPI-процесс привязывается к отдельному ядру центрального процессора. Минус такого подхода связан с тем, что MPI является технологией программирования для систем с распределённой памятью. Поэтому если для вычислений используется, например, персональный компьютер, на котором имеется один многоядрерный процессор, то хоть ускорение работы программы по сравнению с последовательной версией и будет присутствовать, но по сравнению с использованием технологии OpenMP технология MPI будет в каком-то смысле «тяжеловесной». При её использовании будут порождаться отдельные MPI-процессы, каждый из которых будет привязываться к отдельному ядру, и для каждого из которых будет выделяться какая-то часть общей оперативной памяти. При взаимодействии процессов будут возникать «паразитные» накладные расходы, связанные с копированием данных из одного места общей памяти (которая выделена под какой-то конкретный MPI-процесс) в другое (которая выделена под другой MPI-процесс). То есть накладных расходов будет больше по сравнению с технологией OpenMP, которая порождает множество потоков, каждый из которых привязывается к своему вычислительному ядру, но на взаимодействие потоков практически не тратится никакого времени, потому что они берут данные из общей памяти.

В этом разделе будет подробно обсуждаться, как используя особенности Python и некоторые его пакеты (а именно пакет `numpy` и `cupy`), можно достаточно просто использовать гибридные технологии параллельного программирования, в том числе как в программе учесть то, что на каждом вычислительном узле есть многоядерный процессор и/или видеокарта. При этом необходимо отметить, что эффективное использование указанных технологий будет возможно именно для задач рассматриваемого в диссертационном исследовании класса.

4.4.1 Упрощённый пример параллельной реализации алгоритма решения переопределённой СЛАУ с плотно заполненной матрицей в случае одномерного деления матрицы СЛАУ на блоки (с использованием MPI)

В качестве примера будет рассматриваться упрощённая параллельная реализация метода сопряжённых градиентов для решения переопределённой системы линейных алгебраических уравнений $Ax = b$ с плотно заполненной матрицей. Особенность рассматриваемой ниже параллельной программы заключается в том, что используется параллельный алгоритм, основанный на одномерном делении матрицы на блоки, и распараллеливается только часть вычислений с целью уменьшения накладных расходов. При этом соответствующий программный код является достаточно компактным, что позволит обсудить основную тему этого раздела, не уделяя лишнего времени повторению достаточно громоздкого материала. Рассматриваемые в этом параграфе программные приёмы могут быть легко обобщены на рассмотренные ране полные программные реализации (пример может быть найден в приложении Б).

В листинге 4.8 представлен Python-код такой упрощённой параллельной версии «классического» алгоритма, использующий стандарт MPI-2.0.

Листинг 4.8: Python-код упрощённой параллельной версии «классического» алгоритма, использующий стандарт MPI-2.0.

```
def conjugate_gradient_method(A_part, b_part, x, alpha, comm, N) :

    s = 1
    p = zeros(N, dtype=float64)

    while True :

        if s == 1 :
            r_temp = dot(A_part.T, dot(A_part, x) - b_part)
            r = empty(N, dtype=float64)
            comm.Allreduce([r_temp, N, MPI.DOUBLE],
                           [r, N, MPI.DOUBLE], op=MPI.SUM)
            r = r + alpha*x
        else :
            r = r - q/scalar_product_pq
```

```
        scalar_product_rr = dot(r, r)
        p = p + r/scalar_product_rr

        q_temp = dot(A_part.T, dot(A_part, p))
        q = empty(N, dtype=float64)
        comm.Allreduce([q_temp, N, MPI.DOUBLE],
                       [q, N, MPI.DOUBLE], op=MPI.SUM)
        q = q + alpha*p

        scalar_product_pq = dot(p, q)
        x = x - p/scalar_product_pq

        s = s + 1

        if s == N + 1 :
            x_classic = x.copy()
            return x_classic
```

На вход этой функции, вызываемой каждым MPI-процессом, подаётся: 1) массив `A_part`, который содержит свою часть $A_{part_{(.)}}$ матрицы A (размерность $A_{part_{(.)}} : M_{part_{(.)}} \times N$); 2) массив `b_part`, который содержит свою часть $b_{part_{(.)}}$ вектора b (число элементов $b_{part_{(.)}} : M_{part_{(.)}}$); 3) массив `x`, который содержит начальное приближение для x полной длины N; 4) константа `alpha`, которая содержит параметр регуляризации α; 4) коммуникатор `comm` $\equiv$ `MPI.COMM_WORLD`, содержащий информацию о всех процессах, которые участвуют в вычислениях; 5) константа `N`, которая содержит число компонент N искомого вектора x По итогу работы этой функции на каждом процессе возвращается массив `x`, который содержит решение системы $A\,x = b$. Пример распределения данных по процессам приведён на рис. 4.15).

Для подготовки на вычислительных узлах вспомогательных данных для работы этой функции можно использовать последовательность действий, описанную в параграфе 4.1, положив: 1) `num_row` := `numprocs` и `num_col` := `1`; 2) `comm_row` в качестве входного аргумента `comm` функции при её вызове.

Главный минус такой программной реализации, которая основана на одномерном делении матрицы на блоки, заключается в том, что с ростом числа n участвующих в вычислениях процессов время счёта пропорционально $\frac{1}{n}$, а время на обмен сообщениями между процессами пропорционально $\log_2 n$ (в случае двумерного деления матрицы на блоки: $\frac{\log_2 \sqrt{n}}{\sqrt{n}}$). Таким образом, с увеличе-

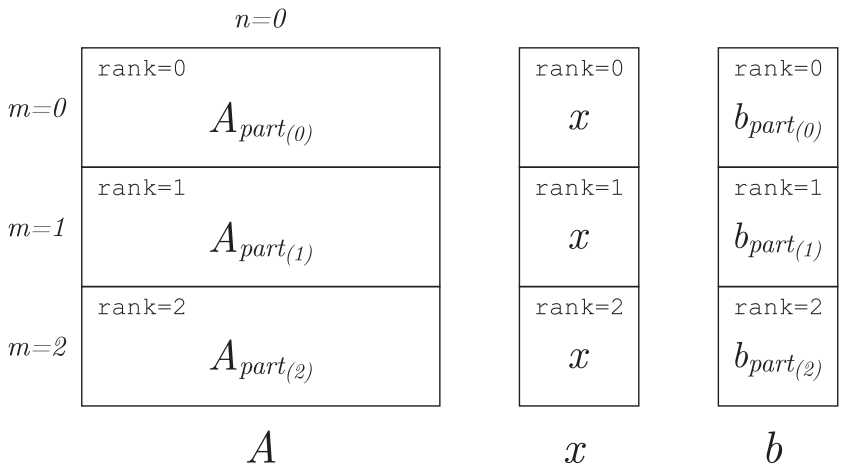


Рисунок 4.15 — Пример распределения данных по 3 MPI-процессам, которые формально образуют сетку 3×1 в случае одномерного деления матрицы на блоки.

нием числа участвующих в вычислениях процессов доля накладных расходов по приёму/передаче сообщений между MPI-процессами только увеличивается. Как следствие, это приводит к плохой сильной маштабируемости программной реализации.

Также минусом этой программной реализации является то, что не все вычисления распараллелены. Однако число последовательных (нераспараллеленых) арифметических операций равно $9N^2 - 6N + 1$, а число распараллеленных арифметических операций равно $4MN^2 + 3MN - N^2 - N$. Для достаточно больших N, M и $n \ll M$ доля нераспараллеленых операций будет близка к 0. Таким образом, этот минус не является существенными при указанных условиях применимости.

В листинге 4.9 представлен Python-код упрощённой параллельной версии усовершенствованного алгоритма, использующий стандарт MPI-4.0.

Листинг 4.9: Python-код упрощённой параллельной версии усовершенствованного алгоритма, использующий стандарт MPI-4.0.

```
def conjugate_gradient_method(A_part, b_part, x, alpha, comm, N):

    x_classic = None
    delta = finfo(float64).eps

    q_temp = empty(N, dtype=float64)
    q = empty(N, dtype=float64)

    request = comm.Allreduce_init([q_temp, N, MPI.DOUBLE],
                                  [q, N, MPI.DOUBLE], op=MPI.SUM)
```

```
    s = 1
    p = zeros(N, dtype=float64)

    while True :

        if s == 1 :
            r_temp = dot(A_part.T, dot(A_part, x) - b_part)
            r = empty(N, dtype=float64)
            comm.Allreduce([r_temp, N, MPI.DOUBLE],
                           [r, N, MPI.DOUBLE], op=MPI.SUM)
            r = r + alpha*x
            sigma2_r = zeros(N, dtype=float64)
        else :
            r = r - q/scalar_product_pq
            sigma2_r = sigma2_r + q**2/scalar_product_pq**2

        scalar_product_rr = dot(r, r)
        p = p + r/scalar_product_rr

        q_temp[:] = dot(A_part.T, dot(A_part, p))
        MPI.Prequest.Start(request)

        criterion = delta**2*sum(sigma2_r)/scalar_product_rr
        MPI.Request.Wait(request, status=None)

        if criterion >= 1 :
            return x, s, x_classic

        q[:] = q[:] + alpha*p

        scalar_product_pq = dot(p, q)
        x = x - p/scalar_product_pq

        s = s + 1

        if s == N + 1 :
            x_classic = x.copy()
```

Далее, для упрощения изложения материала, будут рассматриваться модификации листинга программы 4.8. Аналогичные рассмотренным далее модификации могут быть достаточно легко применены как к листингу 4.9, так и к основному листингу 4.5 из раздела 4.2.3.

4.4.2 Модификация программной реализации алгоритма с использованием OpenMP для вычислений на многоядерных процессорах

Сначала необходимо отметить одну особенность использования для тестов персонального компьютера с многоядерным процессором. Если засечь время работы функции `conjugate_gradient_method()`, то внезапно окажется, что время её работы практически не зависит от числа используемых для вычислений MPI-процессов при условии, что число используемых при запуске MPI-процессов не превышает число ядер многоядерного процессора.

Это связано с особенностью функции `dot()` из пакета `numpy`, которая используется для всех самых вычислительно ёмких операций в рассматриваемой сейчас версии функции `conjugate_gradient_method()`. Эта функция при расчётах автоматически использует многопоточность (используя технологию OpenMP), задействуя в вычислениях все ядра многоядерного процессора. Таким образом, при недостаточно большом числе MPI-процессов отличий во времени работы последовательной версии программы от её параллельной реализации заметно практически не будет. Если же число MPI-процессов сильно превышает число доступных для вычислений ядер, то время работы параллельной программы может даже превосходить время работы последовательной версии программы. Это связано с тем, что кроме непосредственных вычислений будет тратиться время и на взаимодействие MPI-процессов друг с другом. Таким образом, для тестирования эффективности рассматриваемой функции персональный компьютер с одним многоядерным процессором использовать не конструктивно, и будет подразумеваться, что соответствующие программы запускаются на сложных вычислительных системах, структура которых была описана выше.

При запуске программы на многопроцессорных вычислительных системах с многоядерными процессорами всегда есть возможность привязать каждый MPI-процесс как к одному вычислительному узлу (которой чаще всего содержит один процессор), так и к одному ядру. Например, скрипт-файл, который использовался при запуске рассматриваемых параллельных программ на суперкомпьютере «Ломоносов-2», содержит следующие команды:

```
#!/bin/bash --login
```

```
#SBATCH --partition=test
#SBATCH --time=0-00:15:00
#SBATCH --nodes=15
#SBATCH --ntasks-per-node=14

srun mpi=pmi2 python ./program.py
```

При этом строка

```
#SBATCH --ntasks-per-node=14
```

учитывает то, что каждый узел суперкомпьютера «Ломоносов-2» содержит многоядерный процессор с 14-ю ядрами, а значит каждый MPI-процесс привяжется к отдельному ядру. Такой же результат можно получить, если заменить эту строку на

```
#SBATCH --cpus-per-task=1
```

Если же необходимо, чтобы один MPI-процесс привязывался только к одному вычислительному узлу, то надо задать

```
#SBATCH --ntasks-per-node=1
```

В результате, каждый MPI-процесс, выполняющий функцию `conjugate_gradient_method()`, при выполнении матричных операций с помощью функции `dot()` из пакета `numpy` автоматически будет задействовать технологию OpenMP. Это связано с тем, что при выполнении функции `dot()` внутри неё автоматически будет определяться, что программа выполняется на многоядерном процессоре, и будет задействоваться версия этой функции, использующая технологию OpenMP. В результате будут полностью задействоваться вычислительные ресурсы многоядерного процессора.

То есть всё настолько просто!

Замечание. Многие функции из пакета `numpy`, использующиеся для выполнения матричных операций, при своём запуске делают проверку, запущены ли они на многоядерном процессоре, и в случае положительного ответа запускают свою многопоточную версию, написанную на C/C++/Fortran с использование технологии OpenMP.

Замечание. Если уже готовой функции, использующей технологию OpenMP, в каком-либо доступном пакете Python нет, то можно реализовать соответствующую функцию на C/C++/Fortran самостоятельно и использовать её при вызовах из Python (см., например, учебник А. С. Антонова [189]).

4.4.3 Модификация программной реализации алгоритма с использованием CUDA для вычислений на видеокартах

Есть несколько популярных пакетов для Python, которые позволяют использовать технологию CUDA, например, `numba` и `cupy`. Пакет `numba` позволяет писать программы на низком уровне, то есть в таком же стиле, в каком здесь рассматривались программные реализации алгоритмов с использованием технологии MPI. В рамках этой работы использование технологии CUDA рассматривается лишь обзорно, поэтому будет рассматриваться использование пакетных решений, а именно пакета `cupy`, который содержит множество функций, использующих видеокарты для выполнения матричных операций. Подключить этот пакет можно с помощью команды

```
import cupy as cp
```

Терминология, которая используется при написании программ для вычислений на видеокартах, следующая.

- «Хост» (host) — центральный процессор (CPU) и его оперативная память (либо, её ещё называют системной памятью).
- «Девайс» (device) — видекарта, то есть графические процессоры (GPU), и её видеопамять.

Все получаемые узлом данные, например при считывании из файла либо при получении сообщений от других вычислительных узлов, хранятся в оперативной памяти «хоста». Для вычислений на видеокарте необходимо перенести эти данные в память видеокарты, т.е. «девайса». Для этого в пакете `cupy` есть функция `cp.asarray()`. Пример её использования:

```
x_d = cp.asarray(x)
```

Эта функций по системной шине передаёт массив `x` из оперативной памяти «хоста» в видеопамять «девайса». В результате создаётся массив `x_d`, расположенный в памяти видеокарты. Для удобства чтения программного кода в именах переменных, которые ассоциированы с объектами, расположенными в видеопамяти «девайса», принято ставить постфикс `_d` (от слова «device»).

В пакте `cupy` есть аналог функции `dot()`. Её вызов осуществляется командой `cp.dot()`, которая внутри этой функции использует для вычислений технологию параллельного программирования CUDA. При этом аргументами этой функции должны быть массивы, содержащиеся в памяти видеокарты!

Результатом вычислений является массив, также хранящийся в памяти видеокарты («девайса»). Если его нужно перенести в память «хоста», то это можно сделать с помощью функции `cp.asnumpy()`. Например, так:

```
x = cp.asnumpy(x_d)
```

Либо, это же можно сделать с помощью Python-метода `get()`:

```
x = x_d.get()
```

То есть всегда надо следить за тем, чтобы обрабатываемые видеокартой данные находились в видеопамяти «девайса», для обработки этих данных использовать функции из пакета `cupy`, и не забывать возвращать соответствующие данные в память «хоста», если нужны вычисления на «хосте» либо необходим обмен данными между различными вычислительными узлами посредством технологии MPI.

Теперь, используя эти знания, листинг 4.8 может быть изменён следующим образом.

Листинг 4.10: Python-код упрощённой параллельной версии «классического» алгоритма, использующий технологии MPI-2.0, OpenMP и CUDA.

```
1  def conjugate_gradient_method(A_part, b_part, x, alpha, comm, N) :
2
3      import cupy as cp
4
5      A_part_d = cp.asarray(A_part)
6
7      s = 1
8      p = zeros(N, dtype=float64)
9
10     while True :
11
12         if s == 1 :
13             x_d = cp.asarray(x)
14             b_part_d = cp.asarray(b_part)
15             r_temp = cp.dot(A_part_d.T,
16                             cp.dot(A_part_d, x_d) -
17                             b_part_d).get()
18             r = empty(N, dtype=float64)
19             comm.Allreduce([r_temp, N, MPI.DOUBLE],
20                            [r, N, MPI.DOUBLE], op=MPI.SUM)
21             r = r + alpha*x
22         else :
23             r = r - q/scalar_product_pq
```

```
24
25          scalar_product_rr = dot(r, r)
26          p = p + r/scalar_product_rr
27
28          p_d = cp.asarray(p)
29          q_temp = cp.dot(A_part_d.T, cp.dot(A_part_d, p_d)).get()
30          q = empty(N, dtype=float64)
31          comm.Allreduce([q_temp, N, MPI.DOUBLE],
32                         [q, N, MPI.DOUBLE], op=MPI.SUM)
33          q = q + alpha*p
34
35          scalar_product_pq = dot(p, q)
36          x = x - p/scalar_product_pq
37
38          s = s + 1
39
40          if s == N + 1 :
41              x_classic = x.copy()
42              return x_classic
```

Здесь в строке 3 подключается пакет `cupy` и сразу же, в строке 5, массив `A_part` копируется из памяти «хоста» в память «девайса». Необходимо отметить, что этот массив, являющийся самым большим среди использующихся в вычислениях, переносится в память «хоста» только один раз. Далее в основном цикле обмен данными между «хостом» и «девайсом» будет осуществляться посредством передачи значительно более маленьких по объёму массивов.

В строке 13–14 массивы `x` и `b_part` копируются из памяти «хоста» в память «двайса». В памяти «девайся» соответствующие данные будут храниться в массивах `x_d` и `b_part_d`.

В строках 15–17 выполняется операция $A_{part}^T \cdot \left(A_{part}\, x - b_{part}\right)$. Эти действия выполняются на видеокарте («девайса»), используя функцию `cp.dot()` из пакета `cupy`. При этом результат этой операции (массив `r_temp_d`) должен быть сложен с аналогичными массивами, полученными на других вычислительных узлах, с помощью функции `Allreduce()`. Но функция `Allreduce()` может взаимодействовать только с памятью «хоста». Поэтому этот массив с помощью Python-метода `get()` переносится в память «хоста», где сохраняются в массиве `r_temp`.

Одно из «узких мест» использования технологии CUDA заключается в следующем: передача данных между «хостом» и «девайсом» может быть

достаточно времязатратной операцией. По возможности, надо избегать приёма/передачи данных с «хоста» на «девайс» и наоборот.

Именно по этой причине в строке 21 переопределение вектора r (содержащегося в массиве `r`) реализуется на «хосте». Это связано с тем, что скалярное произведение векторов, умножение вектора на число и сложение векторов реализуется достаточно быстро, поэтому на центральном процессоре («хост») это может занять меньше времени, чем пересылка данных на видеокарту и проведение соответствующих расчётов на ней. Кроме того, так как для вычисления скалярного произведения используется функция `dot()` из пакета `numpy`, то внутри этой функции также используется технология OpenMP.

Аналогичные изменения внесены и строки 28–38.

Таким образом, этот пример является показательным с той точки зрения, что в нём использованы гибридные технологии параллельного программирования MPI+CUDA+OpenMP.

Замечание. Если уже готовой функции, использующей технологию CUDA, в пакете `cupy` нет, то можно написать соответствующую функцию с помощью пакета `numba` или CUDA Fortran [190] самостоятельно и использовать её при вызовах из Python.

4.4.4 Оценка эффективности и масштабируемости предложенных программных реализаций

Для тестирования эффективности соответствующих программных реализаций использовался вычислительный раздел «test» суперкомпьютера «Ломоносов-2» [96] научно-исследовательского вычислительного центра МГУ имени М. В. Ломоносова. Каждый вычислительный узел раздела «test» содержит 14-ядерный процессор Intel Xeon E5-2697 v3 2.60GHz с 64 GB оперативной памяти (4.5 GB на ядро) и видеокарту Tesla K40s с объёмом видеопамяти 11.56 GB.

В качестве тестовой задачи была выбрана задача $N = 10\,000$, $M = 25\,000$. В этом случае, для хранения матрицы системы A необходимо $N \times M \times 8$ байт = 1.86 GB. Тестовые параметры N и M отличаются от тех, которые использовались в тестовой задаче в подпараграфе 4.2.4, в связи с тем, что объём видеопамяти на каждом узле меньше объёма системной оперативной памя-

ти. Поэтому при тех же тестовых параметрах, что и в подпараграфе 4.2.4, не удалось бы уместить матрицу системы для тестового примера в видеопамяти «девайса».

Для указанных параметров OpenMP-версия функции `conjugate_gradient_method()` (см. листинг 4.8) на 1 вычислительном узле работает ∼769 секунд, а CUDA+OpenMP-версия работает ∼245 секунд. Таким образом, сразу же виден значительный вклад вычислительной видеокарты в прирост скорости вычислений. Параллельная версия программы запускалась с привязкой каждого MPI-процесса ровно к одному вычислительному узлу.

На рис. 4.16 представлены графики эффективности распараллеливания в зависимости от числа использованных для вычисления узлов. Видно, что масштабируемость программной реализации, использующей CUDA + OpenMP (в дополнение к MPI) хуже, чем масштабируемость программной реализации, использующей только OpenMP (в дополнение к MPI).

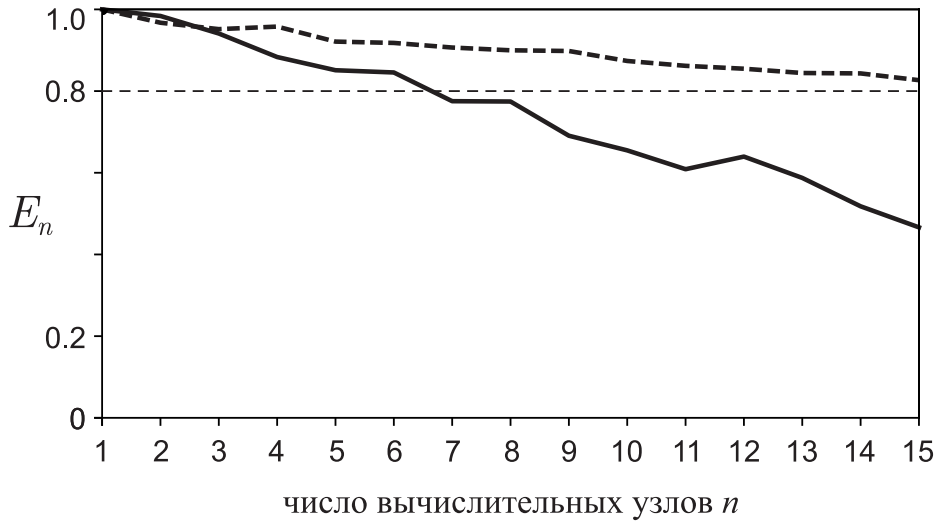


Рисунок 4.16 — Графики эффективности программной реализации функции `conjugate_gradient_method()` в зависимости от числа использованных для вычислений узлов. Сплошная линия соответствует графику на основе программной реализации использующей технологию CUDA + OpenMP (в дополнение к MPI); пунктирная линия соответствуют использованию только технологии OpenMP (в дополнение к MPI).

Такой результат связан с наличием дополнительных накладных расходов на приём/передачу данных внутри вычислительно узла между «хостом» и «девайсом» в случае использования технологии CUDA. Это приводит к дополнительным накладным расходам, доля которых растёт с увеличением участвующих в вычислениях узлов. В случае рассматриваемой в этом разделе

упрощённой версии функции `conjugate_gradient_method()` объём передаваемых по системной шине данных не меняется и пропорционален N, а объём вычислений на каждом узле уменьшается пропорционально $\frac{M\,N}{n}$. В случае же двумерного деления матрицы на блоки объём передаваемых по системной шине данных уменьшался бы пропорционально $\frac{N}{\sqrt{n}}$ (или $\frac{M}{\sqrt{n}}$), а объём вычислений уменьшался пропорционально $\frac{M\,N}{n}$. То есть дополнительные накладные расходы, связанные с использованием технологии CUDA, можно уменьшить, но их доля всё равно будет расти с ростом используемых для вычислений узлов. Как следствие, масштабируемость программных реализаций, будет хуже по сравнению с теми реализациям, что не используют технологию CUDA. Но при этом эти программные реализации во многих случаях всё равно будут быстрее. График времени работы функции `conjugate_gradient_method()` в зависимости от использованных для вычислений узлов представлен на рис. 4.17 и явно демонстрирует вклад в вычисления видеокарты: время работы существенно уменьшается (примерно в 2-3 раза в случае использованной для тестов вычислительной системы).

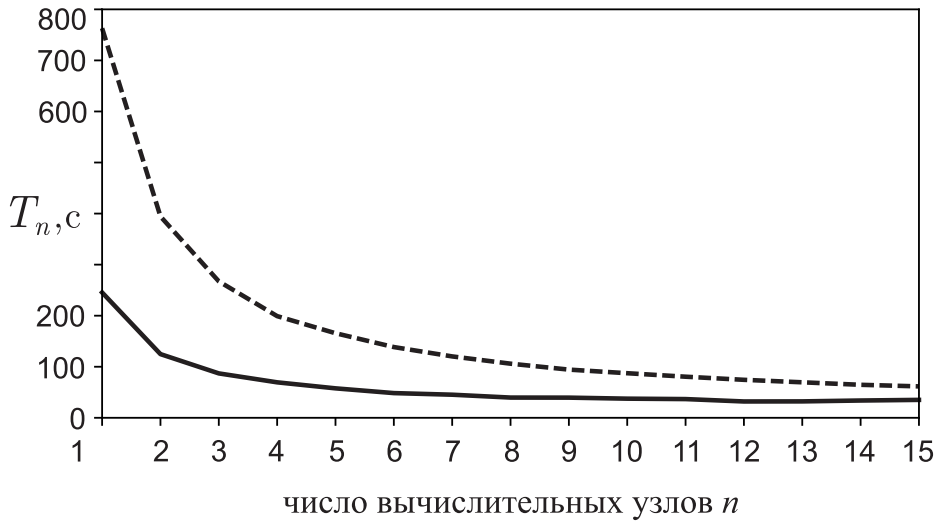


Рисунок 4.17 — Графики времени работы функции `conjugate_gradient_method()` в зависимости от числа использованных для вычислений узлов. Сплошная линия соответствует графикам на основе программной реализации использующей технологию CUDA + OpenMP (в дополнение к MPI); пунктирные линии соответствуют использованию только технологии OpenMP (в дополнение к MPI).

Замечание. Необходимо ещё раз отметить, что особенность параллельных алгоритмов, использующихся для решения рассматриваемого класса обратных

задач, такова, что в случае двумерного деления матрицы на блоки время счёта пропорционально $\frac{1}{n}$, а время на обмен сообщениями между процессами пропорционально $\frac{\log_2 \sqrt{n}}{\sqrt{n}}$ (n — число MPI-процессов). Таким образом, начиная с какого-то конкретного числа процессов (это число зависит от вычислительных размеров задачи и от конфигурации многопроцессорной системы) доля накладных расходов по обмену сообщениями между процессами начнёт существенно увеличиваться. А это, как следствие, неизбежно будет приводить к ухудшению эффективности программной реализации с увеличением числа участвующих в вычислениях процессов. Поэтому главной целью является такая программная реализация, которая позволяет существенным образом уменьшить скорость убывания графика эффективности распараллеливания. Но, как было показано в подпараграфе 4.2.4 (см., например, рис. 4.8), программные реализации, использующие для организации взаимодействия между вычислительными узлами MPI-функции из стандарта параллельного программирования MPI-4, позволяют существенным образом уменьшить скорость убывания графика эффективности распараллеливания.

4.5 Учёт априорной информации о решении

Как было отмечено в самом начале этой главы, регуляризованное решение системы (4.1) может быть найдено как элемент, реализующий минимум функционала А. Н. Тихонова (4.2), в котором присутствует матрица R, содержащая априорную информацию об искомом решении. Для упрощения изложения материала до сих пор использовалось, что $R = E$, где E — единичная матрица. Таким образом, в алгоритме упрощались слагаемые вида $\alpha R^T(Rp^{(s)})$ (см. раздел «Классическая реализация метода сопряжённых градиентов» в подпараграфе 4.2.1). В результате в алгоритмах 4.1, 4.2, 4.3 и 4.4 слагаемые вида $\alpha R^T(Rp)$ заменялись на слагаемые вида αp, что эквивалентно регуляризации по норме решения в пространстве L_2, то есть при отсутствии априорной информации о решении. В программных реализациях параллельных алгоритмов соответствующие слагаемые имели вид αp_{part}, так как на каждом MPI-процессе необходимо было иметь только часть αp_{part} вектора αp.

Если же матрица R отлична от единичной, то есть два варианта действий.

Первый вариант предполагает предварительное вычисление $\widetilde{R} := R^T R$ при подготовке данных для вычислений. Принципиальное отличие операции $R^T R$ от операции $A^T A$ заключается в том, что матрица R обычно является разреженной. Поэтому предварительное вычисление $R^T R$ потребует $O(N^2)$ арифметических операций (N — число элементов вектора решения x) по сравнению с $O(N^3)$ арифметических операций в случае вычисления $A^T A$. Таким образом, эта операция не является вычислительно затратной. Более того, в связи с тем что результат вычисления $R^T R$ также будет являться разреженной матрицей, хранение этой матрицы в памяти будет требовать только $O(N^1)$ условных ячеек памяти (в отличие от $O(N^2)$ в случае хранения в памяти результата операции $A^T A$).

Далее в программной реализации нужно будет выполнить операцию $\alpha\widetilde{R}p$. Так как вектор p хранится по частям $p_{part_{(\cdot)}}$ на различных MPI-процессах аналогично схеме хранения частей $x_{part_{(\cdot)}}$ вектора x изображённой на рис. 4.1, и результат этой операции тоже должен будет храниться на различных MPI-процессах в соответствие с этой схемой, то можно разделить матрицу $\widetilde{R}$ одномерным образом на блоки $\widetilde{R_{part_{(n)}}}$, $n = \overline{0,\texttt{num\_col} - 1}$ (по аналогии с изображённым на рис. 4.15), и в каждой ячейке столбца с индексом n сетки процессов хранить блок $\widetilde{R_{part_{(n)}}}$ (см. рис. 4.1) в разреженном формате хранения. Необходимо отметить, что деление матрицы $\widetilde{R}$ на блоки осуществляется построчно, но соответствующие блоки $\widetilde{R_{part_{(n)}}}$ хранятся по столбцам сетки процессов.

Для вычисления на каждом MPI-процессе своей части вектора $\alpha R^T(Rp)$ в соответствии со схемой изображённой на рис. 4.1 MPI-процесс должен умножить хранящуюся на неём часть $\widetilde{R_{part_{(\cdot)}}}$ матрицы $\widetilde{R}$ на весь вектор p. Для этого на каждом процессе нужно собрать вектор p из частей $p_{part_{(\cdot)}}$ этого вектора, хранящихся на всех ячейках любой из строк сетки процессов. Для этого необходимо предварительно использовать коллективную операцию взаимодействия процессов `Gatherv()` среди процессов каждого вспомогательного коммуникатора `comm_row`, что будет приводить к необходимости обмена сообщениями длины N, что может приводить к повышенным накладным расходам на общем сообщениями между MPI-процессами.

Поэтому второй вариант действий заключается в том, чтобы разбить матрицу $\widetilde{R}$ на блоки двумерным образом, на каждом процессе хранить только свой блок $\widetilde{R_{part_{(\cdot)}}}$ в разреженном формате хранения, и на каждой итерации метода

сопряжённых градиентов выполнять параллельную версию операции $R^T Rp$ по схеме аналогичной параллельной версии операции $A^T Ap$. В этом случае при реализации коллективных операций обмена сообщениями между процессами, длина каждого сообщения будет пропорциональна $\frac{N}{\sqrt{n}}$ (здесь n — число участвующих в вычислениях MPI-процессов). Значит накладных расходов будет меньше. Как следствие, эффективность программной реализации будет выше.

4.6 Выбор между Python и Fortran: «за» и «против»

В диссертационном исследовании было показано, что использование гибридных технологий параллельного программирования MPI+OpenMP+CUDA является достаточно простым при написании программ на языке программирования Python только в том случае, когда операции, которые следует выполнить на многоядерном процессоре или вычислительной видеокарте, уже реализованы в виде отдельной функции соответствующего пакета (`numpy`, `cupy`, `numba` и т.д.). Если необходимые операции не реализованы в виде доступных функций, то преимущества языка программирования Python нивелируются по сравнению с языком программирования Fortran.

Более того, даже при наличии соответствующих готовых функций, выбор Python является конструктивным только в том случае, если решаемая задача допускает эффективные вычисления с двойной точностью. Если необходимо проводить вычисления с четверной точностью, что требуют многие рассмотренные постановки обратных задач магнитометрии, использование языка программирования Fortran является практически безальтернативным. При этом это утверждение верно только на момент написание диссертационной работы. Это связано с тем, что последние годы в научном сообществе активно продвигается запрос на поддержку пакетом `numpy` языка программирования Python вычислений с четверной точностью. Но возможности реализации вычислений с четверной точностью упираются в то, что большинство функций линейной алгебры пакета `numpy` используют функции программных библиотек линейной алгебры Intel MKL и OpenBLAS (в том числе и те, которые поддерживают использование многоядерных процессоров). До тех пор, пока со-

ответствующие библиотеки не станут полноценно поддерживать вычисления с четверной точностью, Python, как зависимый от них посредством пакета `numpy` язык программирования, будет тоже ограничен в возможностях. Но в связи с бурным развитием вычислительных технологий в последние годы, автор считает необходимым отметить этот момент, так как с каждым годом язык программирования Python сокращает отставание от языка программирования Fortran в части высокопроизводительных научных расчётов.

Заключение

Основные результаты работы заключаются в следующем.

1. Математическое моделирование показало возможность решения широкого класса обратных задач магнитометрии в постановках, учитывающих различную информацию о параметрах измеряемого в эксперименте магнитного поля и априорную информацию о параметрах нормального поля в области расположения исследуемого объекта.
2. Численные исследования показали, что наиболее перспективными постановками обратных задач магнитометрии являются те, в которых в качестве входных данных используется экспериментальная информация о компонентах тензора градиентов компонент индукции магнитного поля. Математическое моделирование решений обратных задач магнитометрии на основе таких постановок позволяет получать более детализированные магнитные изображения исследуемых объектов.
3. Для выполнения поставленных задач был создан программный комплекс решения трёхмерных обратных задач магнитометрии с использованием технологий параллельного программирования MPI, OpenMP и CUDA для проведения расчётов на гетерогенных вычислительных системах, каждый узел которых содержит многоядерные процессоры и/или графические ускорители. Вычислительным ядром этого программного комплекса является параллельная программная реализация алгоритма решения больших переопределённых систем линейных алгебраических уравнений с плотно заполненной матрицей с учётом ошибок машинного округления. Созданный программный комплекс имеет открытый код и подробное описание, в связи с чем он может быть применён или модифицирован для решения задач и из совершенно других областей науки и техники, если они сводятся к решению линейных систем такого типа.

Список литературы

1. *Лукьяненко Д. В., Ягола А. Г.* Использование многопроцессорных систем для решения обратных задач, сводящихся к интегральным уравнениям Фредгольма 1-го рода // *Труды Института математики и механики УрО РАН.* — 2012. — Т. 18, № 1. — С. 222–234.

2. *Ван Я., Лукьяненко Д. В., Ягола А. Г.* Регуляризированное обращение полных тензорных магнитно-градиентных данных // *Вычислительные методы и программирование.* — 2016. — Т. 17, № 1. — С. 13–20.

3. Восстановление магнитной восприимчивости с использованием полных магнито-градиентных данных / Я. Ван, И. И. Колотов, Д. В. Лукьяненко, А. Г. Ягола // *Журнал вычислительной математики и математической физики.* — 2020. — Т. 60, № 6. — С. 1027–1034.

4. О единственности решения систем линейных алгебраических уравнений, к которым редуцируются обратные линейные задачи гравиметрии и магнитометрии: локальный случай / И. И. Колотов, Д. В. Лукьяненко, И. Э. Степанова, А. Г. Ягола // *Журнал вычислительной математики и математической физики.* — 2023. — Т. 63, № 8. — С. 1317–1331.

5. О единственности решения систем линейных алгебраических уравнений, к которым редуцируются обратные задачи гравиметрии и магнитометрии: региональный вариант / И. И. Колотов, Д. В. Лукьяненко, И. Э. Степанова и др. // *Журнал вычислительной математики и математической физики.* — 2023. — Т. 63, № 9. — С. 1446–1457.

6. *Леонов А. С., Лукьяненко Д. В., Ягола А. Г.* «Быстрый» алгоритм решения некоторых трехмерных обратных задач магнитометрии // *Математическое моделирование.* — 2024. — Т. 36, № 1. — С. 41–58.

7. *Lukyanenko D. V., Yagola A. G., Evdokimova N. A.* Application of inversion methods in solving ill-posed problems for magnetic parameter identification of steel hull vessel // *Journal of Inverse and Ill-Posed Problems.* — 2011. — Vol. 18, no. 9. — Pp. 1013–1029.

8. *Lukyanenko D. V., Yagola A. G.* Some methods for solving of 3D inverse problem of magnetometry // *Eurasian Journal of Mathematical and Computer Applications.* — 2016. — Vol. 4, no. 3. — Pp. 4–14.

9. Magnetic susceptibility inversion method with full tensor gradient data using low-temperature SQUIDs / Y. Wang, L. Rong, L. Qiu et al. // *Petroleum Science.* — 2019. — Vol. 16. — Pp. 794–807.

10. *Wang Y., Lukyanenko D., Yagola A.* Magnetic parameters inversion method with full tensor gradient data // *Inverse Problems and Imaging.* — 2019. — Vol. 13, no. 4. — Pp. 745–754.

11. General Tikhonov regularization with applications in geoscience / Y. Wang, A. S. Leonov, D. V. Lukyanenko, A. G. Yagola // *CSIAM Transaction on Applied Mathematics.* — 2020. — Vol. 1, no. 1. — Pp. 53–85.

12. Recovering the magnetic image of Mars from satellite observations / I. Kolotov, D. Lukyanenko, I. Stepanova et al. // *Journal of Imaging.* — 2021. — Vol. 7, no. 11. — P. 234.

13. Recovering the magnetic properties of Mercury from satellite observations / I. I. Kolotov, D. V. Lukyanenko, I. E. Stepanova et al. // *Eurasian Journal of Mathematical and Computer Applications.* — 2022. — Vol. 10, no. 2. — Pp. 26–41.

14. *Lukyanenko D., Shinkarev V., Yagola A.* Accounting for round-off errors when using gradient minimization methods // *Algorithms.* — 2022. — Vol. 15, no. 9. — P. 324.

15. Recovering the near-surface magnetic image of Mercury from satellite observations / I. Kolotov, D. Lukyanenko, I. Stepanova et al. // *Remote Sensing.* — 2023. — Vol. 15, no. 8. — P. 2023.

16. *Lukyanenko D.* Parallel algorithm for solving overdetermined systems of linear equations, taking into account round-off errors // *Algorithms.* — 2023. — Vol. 16, no. 5. — P. 242.

17. On the unique solvability of inverse problems of magnetometry and gravimetry / I. Stepanova, D. Lukyanenko, I. Kolotov et al. // *Mathematics.* — 2023. — Vol. 11, no. 14. — P. 3230.

18. О построении аналитических моделей магнитного поля Меркурия по спутниковым данным / И. Э. Степанова, А. Г. Ягола, Д. В. Лукьяненко, И. И. Колотов // *Физика Земли.* — 2023. — № 6. — С. 175–189.

19. The uniqueness of the inverse coefficient problem when building analytical models of Mercury's magnetic field / I. E. Stepanova, I. I. Kolotov, D. V. Lukyanenko, A. V. Shchepetilov // *Doklady Earth Sciences.* — 2023.

20. *Lukyanenko D.V., Yagola A.G.* Using parallel computers for solving multidimensional ill-posed problems // Computational Methods for Applied Inverse Problems / Ed. by Yanfei Wang, Anatoly G. Yagola, Changchun Yang. — Berlin, Boston: De Gruyter, 2012. — Vol. 56 of *Inverse and Ill-Posed Problems Series.* — Pp. 49–64.

21. *Лукьяненко Д. В.* Программа для восстановления параметров намагниченности объекта по данным измерений компонент вектора индукции или тензора градиентов компонент индукции магнитного поля. — Свидетельство о гос. регистрации программы для ЭВМ; № 2023665314; заявл. 2023-07-17; опубл. 2023-07-28, 2023666291 (Рос. Федерация).

22. *Лукьяненко Д. В.* Программа для решения больших переопределённых систем линейных алгебраических уравнений с плотно заполненной матрицей с учётом ошибок машинного округления. — Свидетельство о гос. регистрации программы для ЭВМ; № 2023666561; заявл. 2023-08-02; опубл. 2023-08-14, 2023667251 (Рос. Федерация).

23. *Жданов М. С.* Аналоги интеграла типа Коши в теории геофизических полей. — Наука, Москва, 1984.

24. *Li Y.G., Oldenburg D.W.* 3-D inversion of magnetic data // *Geophysics.* — 1996. — Vol. 61. — Pp. 394–408.

25. *Lelievre P. G., Oldenburg D. W.* Magnetic forward modelling and inversion for high susceptibility // *Geophysical Journal International.* — 2006. — Vol. 166. — Pp. 76–90.

26. *Pignatelli A., Nicolosi I., Chiappini M.* An alternative 3D inversion method for magnetic anomalies with depth resolution // *Annals of Geophysics.* — 2006. — Vol. 49. — P. Annals of Geophysics.

27. *Жданов М.С.* Теория обратных задач и регуляризации в геофизики. — М.: Научный мир, 2012.

28. *Henderson R. G., Zietz I.* Analysis of total magnetic-intensity anomalies produced by point and line sources // *Geophysics*. — 1948. — Vol. 13, no. 3. — Pp. 428–436.

29. *Portniaguine O., Zhdanov M.S.* Focusing geophysical inversion images // *Geophysics*. — 1999. — Vol. 64. — Pp. 874–887.

30. *Portniaguine O., Zhdanov M.S.* 3-D magnetic inversion with data compression and image focusing // *Geophysics*. — 2002. — Vol. 67. — Pp. 1532–1541.

31. *Wang X., Hansen R.O.* Inversion for magnetic anomalies of arbitrary three-dimensional bodies // *Geophysics*. — 1990. — Vol. 55. — Pp. 1321–1326.

32. *Christensen A., Rajagopalan S.* The magnetic vector and gradient tensor in mineral and oil exploration // *Preview*. — 2000. — Vol. 84. — P. 77.

33. *Schmidt P. W., Clark D. A.* Advantages of measuring the magnetic gradient tensor // *Preview*. — 2000. — Vol. 85. — Pp. 26–30.

34. GETMAG-a SQUID magnetic tensor gradiometer for mineral and oil exploration / P. W. Schmidt, D. A. Clark, K. E. Leslie et al. // *Exploration Geophysics*. — 2004. — Vol. 35. — Pp. 297–305.

35. *Heath P., Heinson G., Greenhalgh S.* Some comments on potential field tensor data // *Exploration Geophysics*. — 2003. — Vol. 34. — Pp. 57–62.

36. Calibration of SQUID vector magnetometers in full tensor gradiometry systems / M. Schiffler, M. Queitsch, R. Stolz et al. // *Geophysical Journal International*. — 2014. — Vol. 198. — Pp. 954–964.

37. *Zhdanov M. S., Cai H. Z., Wilson G. A.* 3D inversion of SQUID magnetic tensor data // *Geology and Geosciences*. — 2012. — Vol. 1. — Pp. 1–5.

38. *Ji S.X., Wang Y.F., Zou A.Q.* Regularizing inversion of susceptibility with projection onto convex set using full tensor magnetic gradient data // *Inverse Problems in Science and Engineering*. — 2017. — Vol. 25. — Pp. 202–217.

39. Обратные задачи и методы их решения. Приложения к геофизике / А. Г. Ягола, И. Э. Степанова, В. Н. Титаренко, Я. Ван. — Бином Москва, 2014.

40. *Hadamard J.* Le probleme de Cauchy et les equations aux derivers particlee linearies hyperbolique. — Paris: Hermann, 1932.

41. *Тихонов А. Н.* О решении некорректно поставленных задач и методе регуляризации // *Доклады Академии наук СССР.* — 1963. — Т. 151, № 3. — С. 501–502.

42. *Тихонов А. Н.* О регуляризации некорректно поставленных задач // *Доклады Академии наук СССР.* — 1963. — Т. 153, № 1. — С. 49–52.

43. *Тихонов А. Н.* Об устойчивости обратных задач // *Доклады Академии наук СССР.* — 1943. — Т. 39, № 5. — С. 195–198.

44. *Тихонов А. Н.* О решении нелинейных интегральных уравнений первого рода // *Доклады Академии наук СССР.* — 1964. — Т. 156, № 6. — С. 1296–1299.

45. *Тихонов А. Н.* О нелинейных уравнениях первого рода // *Доклады Академии наук СССР.* — 1965. — Т. 161, № 5. — С. 1023–1026.

46. *Тихонов А. Н.* О методах регуляризации задач оптимального управления // *Доклады Академии наук СССР.* — 1965. — Т. 162, № 4. — С. 763–765.

47. *Лаврентьев М. М.* Об интегральных уравнениях первого рода // *Доклады Академии наук СССР.* — 1959. — Т. 127, № 1. — С. 31–33.

48. *Лаврентьев М. М.* О некоторых некорректных задачах математической физики. — Новосибирск: Изд-во СО АН СССР, 1962.

49. *Иванов В. К.* О линейных некорректных задачах // *Доклады Академии наук СССР.* — 1962. — Т. 145, № 2. — С. 270–272.

50. *Иванов В. К.* О некорректно поставленных задачах // *Математический сборник.* — 1963. — Т. 61, № 2. — С. 211–223.

51. *Иванов В. К.* О приближенном решении операторных уравнений первого рода // *Журнал вычислительной математики и математической физики.* — 1966. — Т. 6, № 6. — С. 1089–1094.

52. *Иванов В. К.* Некорректные задачи в топологических пространствах // *Сибирский математический журнал.* — 1969. — Т. 10, № 5. — С. 1065–1074.

53. *Иванов В. К., Васин В. В., Танана В. П.* Теория линейных некорректных задач и ее приложения. — М.: Наука, 1978.

54. *Васин В. В., Агеев А. Л.* Некорректные задачи с априорной информацией. — Екатеринбург: Наука, 1978.

55. *Васин В. В., Еремин И. И.* Операторы и итерационные процессы Фейеровского типа. Теория и приложения. — Москва-Ижевск: ИКИ, НИЦ РХД, 2005.

56. *Васин В. В.* Основы теории некорректных задач. — Новосибирск: Издательство СО РАН, 2020.

57. *Кабанихин С. И.* Проекционно-разностные методы определения коэффициентов гиперболических уравнений. — Новосибирск: Наука, 1988.

58. *Кабанихин С. И., Бектемесов М. А., Аяпбергенова А. Т.* Итерационные методы решения обратных и некорректных задач. — Алматы: Наука, 2004.

59. *Кабанихин С. И.* Обратные и некорректные задачи. — Новосибирск: Сибирское научное издательство, 2009.

60. *Kabanikhin S. I.* Inverse and ill-posed problems: theory and applications. — Walter de Gruyter, 2011.

61. Алгоритмы и численные методы решения обратных и некорректных задач / С. И. Кабанихин, К. Т. Искаков, М. А. Бектемесов, М.А. Шишленин. — Астана: КазНПУ, 2012.

62. *Кабанихин С. И., Бектемесов М. А., Шишленин М.А.* Методы решения некорректных задач линейной алгебры. — Астана: КазНПУ, 2012.

63. *Лаврентьев М. М., Романов В. Г., Шишатский С. П.* Некорректные задачи математической физики и анализа. — М.: Наука, 1980.

64. *Романов В. Г.* Обратные задачи математической физики. — М.: Наука, 1984.

65. *Романов В. Г., Кабанихин С. И.* Обратные задачи геоэлектрики. — М.: Наука, 1991.

66. *Танана В. П.* Методы решения операторных уравнений. — М.: Наука, 1981.

67. *Лаврентьев М. М., Резницкая К. Г., Яхно В. Г.* Одномерные обратные задачи математической физики. — Новосибирск: Наука, 1982.

68. *Вайникко Г. М.* Методы решения линейных некорректно поставленных задач в гильбертовых пространствах. — Тарту: Изд-во Тарт. гос. ун-та, 1982.

69. *Вайникко Г. М., Веретенников А. Ю.* Итерационные процедуры в некорректных задачах. — М.: Наука, 19824.

70. *Федотов А. М.* Линейные некорректные задачи со случайными ошибками в данных. — Новосибирск: Наука, 1982.

71. *Федотов А. М.* Некорректные задачи со случайными ошибками в данных. — Новосибирск: Наука, 1990.

72. *Бухгейм А. Л.* Операторные уравнения Вольтерра. — Новосибирск: Наука, 1983.

73. *Бухгейм А. Л.* Введение в теорию обратных задач. — Новосибирск: Наука, 1983.

74. *Гласко В. Б.* Обратные задачи математической физики. — М.: Изд-во МГУ, 1984.

75. *Гончарский А.В., Черепащук А. М., Ягола А. Г.* Некорректные задачи астрофизики. — М.: Наука, 1986.

76. *Тихонов А. Н., Арсенин В. Я.* Методы решения некорректных задач. — М.: Наука, 1986.

77. *Гилязов С. Ф.* Методы решения линейных некорректных задач. — М.: Изд-во МГУ, 1987.

78. *Морозов В. А.* Регулярные методы решения некорректно поставленных задач. — М.: Наука, 1987.

79. *Алифанов О. М., Артюхин Е. А., Румянцев С.В.* Экстремальные методы решения некорректных задач. — М.: Наука, 1988.

80. *Бакушинский А. Б., Гончарский А.В.* Некорректные задачи: численные методы и приложения. — М.: Изд-во МГУ, 1989.

81. *Groetsch C. W.* Inverse problems in the mathematical sciences. — Braunschweig: Vieweg, 1993.

82. Обратные задачи колебательной спектроскопии / И. В. Кочиков, Г. М. Курамшина, Ю. А. Пентин, А. Г. Ягола. — М.: Изд-во МГУ, 1993.

83. *Денисов А. М.* Введение в теорию обратных задач. — М.: Изд-во МГУ, 1994.

84. *Тихонов А. Н., Леонов А. С., Ягола А. Г.* Нелинейные некорректные задачи. — М.: Наука, 1995.

85. *Леонов А. С.* Решение некорректно поставленных обратных задач: очерк теории, практические алгоритмы и демонстрации в МАТЛАБ. — М.: Либроком, 2013.

86. *Engl H. W., Hanke M., Neubauer A.* Regularization of inverse problems. — Dordrecht: Kluwer, 1996.

87. *Лаврентьев М. М., Савельев Л. Я.* Теория операторов и некорректные задачи. — Новосибирск: Издательство Института математики, 1999.

88. *Осипов Ю. С., Васильев Ф. П., Потапов М. М.* Основы метода динамической регуляризации. — М.: Изд-во МГУ, 1999.

89. *Бакушинский А. Б.* Замечания о выборе параметра регуляризации по критерию квазиоптимальности и отношения // *Журнал вычислительной математики и математической физики.* — 1984. — Т. 24, № 8. — С. 1258–1259.

90. *Yagola A.G., Leonov A.S., Titarenko V.N.* Data Errors and an Error Estimation for Ill-Posed Problems // *Inverse Problems in Engineering.* — 2002. — Vol. 10, no. 2. — Pp. 117–129.

91. Численные методы решения некорректных задач / А. Н. Тихонов, А. В. Гончарский, В. В. Степанов, А. Г. Ягола. — М.: Наука, 1990.

92. *Морозов В. А.* О регуляризации некорректно поставленных задач и выборе параметра регуляризации // *Журнал вычислительной математики и математической физики.* — 1966. — Т. 6, № 1. — С. 170–175.

93. *Tarantola A.* Inverse problem theory and methods for model parameter estimation. — *SIAM*, Philadelphia, 2005.

94. *Воеводин В. В., Воеводин Вл. В.* Параллельные вычисления. — СПб.: БХВ-Петербург, 2002.

95. *Воеводин Вл. В., Жуматий С. А.* Вычислительное дело и кластерные системы. — М.: Изд-во МГУ, 2007.

96. Supercomputer Lomonosov-2: large scale, deep monitoring and fine analytics for the user community / Vl. Voevodin, A. Antonov, D. Nikitenko et al. // *Supercomputing Frontiers and Innovations.* — 2019. — Vol. 6, no. 2. — Pp. 4–11.

97. *Akimova E. N., Vasin V. V.* Stable parallel algorithms for solving the inverse gravimetry and magnetometry problems // *Журнал вычислительной математики и математической физики.* — 2004. — Vol. 17, no. 1–2. — Pp. 13–19.

98. *Акимова Е. Н.* Параллельные алгоритмы решения обратных задач гравиметрии и магнитометрии на МВС-1000 // *Вестник ННГУ.* — 2009. — № 4. — С. 181–189.

99. Градиентные методы решения структурных обратных задач гравиметриии и магнитометрии на суперкомпьютере «Уран» / Е. Н. Акимова, В. Е. Мисилов, А. Ф. Скурыдина, А. И. Третьяков // *Вычислительные методы и программирование.* — 2015. — № 16. — С. 155–164.

100. *Воеводин В. В.* Об асимптотическом распределении ошибок округления при разложении матрицы на множители и решении систем уравнений //

Журнал вычислительной математики и математической физики. — 1969. — Т. 9, № 4. — С. 261–264.

101. *Woźniakowski H.* Roundoff-error analysis of a new class of conjugate-gradient algorithms // *Linear Algebra and its Applications.* — 1980. — Vol. 29. — Pp. 507–529.

102. *Kaasschieter E. F.* A practical termination criterion for the conjugate gradient method // *BIT Numerical Mathematics.* — 1988. — Vol. 28, no. 2. — Pp. 308–322.

103. *Strakoš Z.* On the real convergence rate of the conjugate gradient method // *Linear algebra and its applications.* — 1991. — Vol. 154. — Pp. 535–549.

104. *Strakoš Z., Tichý P.* On error estimation in the conjugate gradient method and why it works in finite precision computations // *ETNA. Electronic Transactions on Numerical Analysis [electronic only].* — 2002. — Vol. 13. — Pp. 56–80.

105. *Arioli M., Duff I., Ruiz D.* Stopping criteria for iterative solvers // *SIAM Journal on Matrix Analysis and Applications.* — 1992. — Vol. 13, no. 1. — Pp. 138–144.

106. *Arioli M. A.* A stopping criterion for the conjugate gradient algorithm in a finite element method framework // *Numerische Mathematik.* — 2004. — Vol. 97, no. 1. — Pp. 1–24.

107. *Notay Y.* On the convergence rate of the conjugate gradients in presence of rounding errors // *Numerische Mathematik.* — 1993. — Vol. 65, no. 1. — Pp. 301–317.

108. *Meurant G.* The Lanczos and conjugate gradient algorithms: from theory to finite precision computations. — SIAM, 2006.

109. *Калиткин Н. Н., Кузьмина Л. В.* Улучшенные формы итерационных методов для систем линейных алгебраических уравнений // *Доклады академии наук.* — 2013. — Т. 451, № 3. — С. 264–270.

110. *Higham N. J., Mary T.* A new approach to probabilistic rounding error analysis // *SIAM Journal on Scientific Computing.* — 2019. — Vol. 41, no. 5. — Pp. A2815–A2835.

111. Stochastic rounding: implementation, error analysis and applications / M. Croci, M. Fasi, N. J. Higham et al. // *Royal Society Open Science*. — 2022. — Vol. 9, no. 3. — P. 211631.

112. *Connolly M. P., Higham N. J., Mary T.* Stochastic Rounding and Its Probabilistic Backward Error Analysis // *SIAM Journal on Scientific Computing*. — 2021. — Vol. 43, no. 1. — Pp. A566–A585.

113. *D'Azevedo E., Romine C.* — Reducing communication costs in the conjugate gradient algorithm on distributed memory multiprocessors, 1992. — 09.

114. *De Sturler E., Van Der Vorst H. A.* Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers // *Applied Numerical Mathematics*. — 1995. — Vol. 18, no. 4. — Pp. 441–459.

115. *Eller P. R., Gropp W.* Scalable non-blocking preconditioned conjugate gradient methods // SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. — 2016. — Pp. 204–215.

116. *Ткаченко Б. А.* История размагничивания кораблей Советского Военно-морского флота. — Ленинград : Наука, 1981.

117. *Duthoit F., Krahenbuhl L., Nicolas A.* The boundary integral equation method for the extrapolation of field measurement // *IEEE Transactions on Magnetics*. — 1985. — Vol. 21, no. 6. — Pp. 2439–2442.

118. *Guamieri M., Stella A., Trevisan F.* A methodological analysis of different formulations for solving inverse electromagnetic problems // *IEEE Transactions on Magnetics*. — 1990. — Vol. 26, no. 2. — Pp. 622–625.

119. *Brunotte X., Meunier G.* Line element for efficient computation of the magnetic field created by thin iron plates // *IEEE Transactions on Magnetics*. — 1990. — Vol. 26, no. 5. — Pp. 2196–2198.

120. *Rioux-Damidau F., Bandelier B., Penven P.* A fast and precise determination of the static magnetic field in the presence of thin iron shells // *IEEE Transactions on Magnetics*. — 1995. — Vol. 31, no. 6. — Pp. 3491–3493.

121. *Ohlund G.* Design of submarines for stealth and survivability, Conference, Undersea defence technology // Undersea defence technology. — Nexus Media Limited, 1997. — Pp. 114–118.

122. Three-dimensional inversion of magnetic data with simultaneous remanent magnetization and self-demagnetization / Shuang Liu, Maurizio Fedi, Xiangyun Hu, Yang Ou // International Workshop on Gravity, Electrical & Magnetic Methods and Their Applications, Xi'an, China, 19–22 May 2019. — 2019. — Pp. 456–459.

123. *Campbell W. H.* Introduction to geomagnetic fields. — 2 edition. — Cambridge University Press, 2003.

124. Magnetic fields near Mars: first results / W. Riedler, D. Mohlmann, V.N. Oraevsky et al. // *Nature*. — 1989. — Vol. 341. — Pp. 604–607.

125. *Connerney J.E.P.* Planetary magnetism // *Treatise on Geophysics, 2nd Edition*. — 2015. — Vol. 10. — Pp. 195–237.

126. Magnetic field and plasma observations at Mars: Initial results of the Mars Global Surveyor Mission / M. H. Acuna, J. E. P. Connerney, P. Wasilewski et al. // *Science*. — 1998. — Vol. 279. — Pp. 1676–1680.

127. Global distribution of crustal magnetism discovered by the Mars Global SurveyorMAG/ER Experiment / M. H. Acuna, J. E. Connerney, N. F. Ness et al. // *Science*. — 1999. — Vol. 284. — Pp. 790–793.

128. Tectonic implications of Mars crustal magnetism / J. E. P. Connerney, M. H. Acuna, N. F. Ness et al. // *Proceedings of the National Academy of Sciences*. — 2005. — Vol. 102, no. 42. — Pp. 14970–14975.

129. MARS MAVEN Mission: Magnetometer (MAG) Instrument. — URL: https://pds-ppi.igpp.ucla.edu/search/?sc=MAVEN&i=MAG.

130. MARS MAVEN Mission. — URL: https://mars.nasa.gov/maven/.

131. Magnetic lineations in the ancient crust of Mars / J. E. P. Connerney, M. H. Acuna, P. J. Wasilewski et al. // *Science*. — 1999. — Vol. 284. — Pp. 794–798.

132. *Sprenke K. F., Baker L. L.* Magnetization, paleomagnetic poles, and polar wander on Mars // *Icarus*. — 2000. — Vol. 147. — Pp. 26–34.

133. *Jurdy D. M., Stefanick M.* Vertical extrapolation of Mars magnetic potentials // *Journal of Geophysical Research*. — 2004. — Vol. 109. — P. E10005.

134. *Arkani-Hamed J.* An improved 50-degree spherical harmonic model of the magnetic field of Mars derived from both high-altitude and low-altitude data // *Journal of Geophysical Research*. — 2002. — Vol. 107, no. E5.

135. *Cain J.C., Ferguson B., Mozzoni D.* An n = 90 internal potential function of the Martian crustal magnetic field // *Journal of Geophysical Research*. — 2003. — Vol. 108, no. E2. — P. 5008.

136. An altitude-normalized magnetic map of Mars and its interpretation / M. Purucker, D. Ravat, H. Frey et al. // *Geophysical Research Letters*. — 2000. — Vol. 27, no. 16. — Pp. 2449–2452.

137. *Langlais B., Purucker M. E., Mandea M.* Crustal magnetic field of Mars // *Journal of Geophysical Research – Planets*. — 2004. — Vol. 109, no. E2. — P. E02008.

138. *Mittelholz A., Johnson C. L., Morschhauser A.* A new magnetic field activity proxy for Mars from MAVEN data // *Geophysical Research Letters*. — 2018. — Vol. 45. — Pp. 5899–5907.

139. A new model of the crustal magnetic field of Mars using MGS and MAVEN / B. Langlais, E. Thébault, A. Houliez, M. E. Purucker // *Journal of Geophysical Research – Planets*. — 2019. — Vol. 124. — Pp. 1542–1569.

140. *Zidarov D.* On the solution of some inverse problems in the scope of potential fields and its application in geophysics. — Sofia: BAN, 1968.

141. Modified method S- and R-approximations in solving the problems of Mars's morphology / T. V. Gudkova, I. E. Stepanova, A. V. Batov, A. V. Shchepetilov // *Inverse Problems in Science and Engineering*. — 2021. — Vol. 29, no. 6. — Pp. 790–804.

142. *Gudkova T.V., Stepanova I.E., Batov A.V.* Density anomalies in subsurface layers of Mars: model estimates for the site of the InSight mission seismometer // *Solar System Research volume.* — 2020. — Vol. 54. — Pp. 15–19.

143. Mercury's magnetospheric magnetic field after the first two MESSENGER flybys / I. I. Alexeev, E. S. Belenkaya, J. A Slavin et al. // *Icarus.* — 2010. — Vol. 209. — Pp. 23–39.

144. The structure of Mercury's magnetic field from MESSENGER's first flyby / B. J. Anderson, M. H. Acuna, H. Korth et al. // *Science.* — 2008. — Vol. 321. — Pp. 82–85.

145. The magnetic field of Mercury / B. J. Anderson, M. H. Acuna, H. Korth et al. // *Space Science Reviews.* — 2010. — Vol. 152, no. 1–4. — Pp. 307–339.

146. The global magnetic field of Mercury from MESSENGER orbital observations / B. J Anderson, C. L. Johnson, H. Korth et al. // *Science.* — 2011. — Vol. 333. — Pp. 1859–1862.

147. Low-degree structure in Mercury's planetary magnetic field / B. J. Anderson, C. L. Johnson, H. Korth et al. // *Journal of Geophysical Research.* — 2012. — Vol. 117.

148. *Mayhew M. A.* Inversion of satellite magnetic anomaly data // *Journal Of Geophysics.* — 1979. — Vol. 45. — Pp. 119–128.

149. Magnetic field observations near Mercury: preliminary results from Mariner 10 / N. F. Ness, K. W. Behannon, R. P. Lepping et al. // *Science.* — 1974. — Vol. 185, no. 2. — Pp. 151–160.

150. The magnetic field of Mercury, 1 / N. F. Ness, K. W. Behannon, R. P. Lepping, Y. C. Whang // *Journal of Geophysical Research.* — 1975. — Vol. 80, no. 19. — Pp. 2708–2716.

151. The MESSENGER mission to Mercury: scientific objectives and implementation / S. C. Solomon, R. L. McNutt, R. E. Gold et al. // *Planetary and Space Science.* — 2001. — Vol. 49, no. 14–15. — Pp. 1445–1465.

152. Constraints on the secular variation of Mercury's magnetic field from the combined analysis of MESSENGER and Mariner 10 data / L. C. Philpott,

C. L. Johnson, R. M. Winslow et al. // *Geophysical Research Letters.* — 2014. — Vol. 41, no. 19. — Pp. 6627–6634.

153. *Wicht J., Heyner D.* Mercury's magnetic field in the MESSENGER era // Planetary geodesy and remote sensing. — CRC Press, 2014.

154. Investigating Mercury's environment with the two-spacecraft BepiColombo mission / A. Milillo, M. Fujimoto, G. Murakami et al. // *Earth and Planetary Science Letters.* — 2020. — Vol. 216, no. 5.

155. *Plagemann S.* Model of the internal constitution and temperature of the planet Mercury // *Journal of Geophysical Research.* — 1965. — Vol. 70, no. 4. — Pp. 985–993.

156. Gravity field and internal structure of Mercury from MESSENGER / D. E. Smith, M. T. Zuber, R. J. Phillips et al. // *Science.* — 2012. — Vol. 336, no. 6078. — Pp. 214–217.

157. The Mie representation for Mercury's magnetic field / S. Toepfer, Y. Narita, K.-H. Glassmeier et al. // *Earth Planets and Space.* — 2021. — Vol. 73, no. 1.

158. A modified equivalent source dipole method to model partially distributed magnetic field measurements, with application to Mercury / J. S. Oliveira, B. Langlais, M. A. Pais, H. Amit // *Journal of Geophysical Research Planets.* — 2015. — Vol. 120. — Pp. 1075–1094.

159. Crust heterogeneities and structure at the dichotomy boundary in western Elysium Planitia and Implications for InSight lander / L. Pan, C. Quantin, B. Tauzin et al. // *Icarus.* — 2020. — Vol. 338.

160. Crustal and time-varying magnetic fields at the InSight landing site on Mars / C. L. Johnson, A. Mittelholz, B. Langlais et al. // *Nature Geoscience.* — 2020. — Vol. 13, no. 3. — Pp. 199–204.

161. *Stepanova I. E.* On the S-approximation of the Earth's gravity field. Regional version // *Inverse Problems in Science and Engineering.* — 2009. — Vol. 16, no. 5. — Pp. 1095–1111.

162. *Stepanova I. E., Shchepetilov A. V., Mikhailov P. S.* Analytical models of the physical fields of the Earth in regional version with ellipticity // *Izvestiya, Physics of the Solid Earth.* — 2022. — Vol. 58, no. 3. — Pp. 406–419.

163. Improving the methods for processing large data in geophysics and geomorphology based on the modified S- and F-approximations / I. E. Stepanova, I. A. Kerimov, D. N. Raevskiy, A. V. Shchepetilov // *Izvestiya. Physics of the Solid Earth.* — 2020. — Vol. 16, no. 5. — Pp. 1095–1111.

164. Analytical modeling of the magnetic field of Mars from satellite data using modified S-approximations / A. M. Salnikov, I. E. Stepanova, T. V. Gudkova, A. V. Batov // *Doklady Earth Sciences.* — 2021. — Vol. 499. — P. 575–579.

165. *Lowes F. J., Duka B.* Magnetic multipole moments (Gauss coefficients) and vector potential given by an arbitrary current distribution // *Earth, Planets and Space.* — 2011. — Vol. 63, no. 1–4.

166. Investigating sources of Mercury's crustal magnetic field: further mapping of Messenger magnetometer data / L. L. Hood, J. S. Oliveira, V. Galluzzi, D. A. Rothery // *JGR Planets.* — 2018. — Vol. 123. — Pp. 2647–2666.

167. *Колотов И. И.* Регуляризирующие алгоритмы восстановления магнитных полей по экспериментальным данным. — дис. ... канд. физ.-мат. наук 1.3.3 / Колотов И. И. - МГУ имени М. В. Ломоносова, Москва, 2023. - 118 с.

168. Reconstruction of Mercury's internal magnetic field beyond the octupole / S. Toepfer, I. Oertel, V. Schiron et al. // *Annales Geophysicae.* — 2022. — Vol. 40. — Pp. 91–105.

169. *Frick P. G., Sokoloff D. D., Stepanov R. A.* Wavelets for the space-time structure analysis of physical fields // *Physics Uspekhi.* — 2020. — Vol. 65, no. 1.

170. *Kazantsev S. G., Kardakov V. B.* Poloidal-Toroidal Decomposition of Solenoidal Vector Fields in the Ball // *Journal of Applied and Industrial Mathematics.* — 2019. — Vol. 13. — Pp. 480–499.

171. *Reshetnyak M. Yu.* Spatial Spectra of the geomagnetic Field in the Observations and Geodynamo Models // *Izvestiya, Physics of the Solid Earth.* — 2015. — Vol. 51, no. 3. — Pp. 354–361.

172. *Reshetnyak M. Yu.* Inverse problem in Parker's dynamo // *Russian Journal of Earth Sciences.* — 2015. — Vol. 15.

173. Modeling Mercury's internal magnetic field with smooth inversions / H. Uno, B. J. Anderson, H. Korth et al. // *Earth and Planetary Science Letters.* — 2009. — Vol. 285, no. 3–4. — Pp. 328–339.

174. *Whaler K. A., Purucker M. E.* A spatially continuous magnetization model for Mars // *Earth and Planetary Science Letters.* — 2005. — Vol. 110, no. E9.

175. *Demmel J. W., Heath M. T., Van Der Vorst H. A.* Parallel numerical linear algebra // *Acta numerica.* — 1993. — Vol. 2. — Pp. 111–197.

176. *Hestenes M. R., Stiefel E.* Methods of conjugate gradients for solving linear systems // *Journal of Research of the National Bureau of Standards.* — 1952. — Vol. 49, no. 6. — P. 409.

177. *Greenbaum A.* Iterative methods for solving linear systems. — SIAM, 1997.

178. *Arioli M., Noulard E., A. Russo.* Stopping criteria for iterative methods: applications to PDE's // *Calcolo.* — 2001. — Vol. 38, no. 2. — Pp. 97–112.

179. *Axelsson O., Kaporin I.* Error norm estimation and stopping criteria in preconditioned conjugate gradient iterations // *Numerical Linear Algebra with Applications.* — 2001. — Vol. 8, no. 4. — Pp. 265–286.

180. *Chang X.-W., Paige C. C., Titley-Peloquin D.* Stopping criteria for the iterative solution of linear least squares problems // *SIAM Journal on Matrix Analysis and Applications.* — 2009. — Vol. 31, no. 2. — Pp. 831–852.

181. *Jiránek P., Strakoš Z., Vohralík M.* A posteriori error estimates including algebraic error and stopping criteria for iterative solvers // *SIAM Journal on Scientific Computing.* — 2010. — Vol. 32, no. 3. — Pp. 1567–1590.

182. *Landi G., Loli Piccolomini E., Tomba I.* A stopping criterion for iterative regularization methods // *Applied Numerical Mathematics.* — 2016. — Vol. 106. — Pp. 53–68.

183. *Rao K., Malan P., Perot J. B.* A stopping criterion for the iterative solution of partial differential equations // *Journal of Computational Physics.* — 2018. — Vol. 352. — Pp. 265–284.

184. The numerical stability analysis of pipelined conjugate gradient methods: historical context and methodology / E. C. Carson, M. Rozložník, Z. Strakoš et al. // *SIAM Journal on Scientific Computing*. — 2018. — Vol. 40, no. 5. — Pp. A3549–A3580.

185. Analyzing the effect of local rounding error propagation on the maximal attainable accuracy of the pipelined conjugate gradient method / S. Cools, E. F. Yetkin, E. Agullo et al. // *SIAM Journal on Matrix Analysis and Applications*. — 2018. — Vol. 39, no. 1. — Pp. 426–450.

186. *Greenbaum A., Liu H., Chen T.* On the convergence rate of variants of the conjugate gradient algorithm in finite precision arithmetic // *SIAM Journal on Scientific Computing*. — 2021. — Vol. 43, no. 5. — Pp. S496–S515.

187. *Polyak B.T., Kuruzov I.A., Stonyakin F.S.* Stopping rules for gradient methods for non-convex problems with additive noise in gradient // *arXiv*. — 2022.

188. Numerical linear algebra for high-performance computers / J. J. Dongarra, I. S. Duff, D. C. Sorensen, H. A. Van der Vorst. — SIAM, 1998.

189. *Антонов А. С.* Параллельное программирование с использованием технологии OpenMP: Учебное пособие. — Изд-во МГУ, 2009.

190. *Рутм Г., Фатика М.* CUDA Fortran для инженеров и научных работников. — ДМК-Пресс, 2014.

Приложение А

Свидетельства о государственной регистрации программ для ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2023666291

Программа для восстановления параметров намагниченности объекта по данным измерений компонент вектора индукции или тензора градиентов компонент индукции магнитного поля

Правообладатель: ***Лукьяненко Дмитрий Витальевич (RU)***

Автор(ы): ***Лукьяненко Дмитрий Витальевич (RU)***

Заявка № **2023665314**
Дата поступления **17 июля 2023 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***28 июля 2023 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ
Сертификат 429b6a0fe3853164baf96f83b73b4aa7
Владелец **Зубов Юрий Сергеевич**
Действителен с 10.05.2023 по 02.08.2024

Ю.С. Зубов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2023667251

Программа для решения больших переопределённых систем линейных алгебраических уравнений с плотно заполненной матрицей с учётом ошибок машинного округления

Правообладатель: ***Лукьяненко Дмитрий Витальевич (RU)***

Автор(ы): ***Лукьяненко Дмитрий Витальевич (RU)***

Заявка № **2023666561**
Дата поступления **02 августа 2023 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***14 августа 2023 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ
Сертификат 429b6a0fe3853164baf96f83b73b4aa7
Владелец **Зубов Юрий Сергеевич**
Действителен с 10.05.2023 по 02.08.2024

Ю.С. Зубов

Приложение Б

Листинг вычислительного ядра программного комплекса, использующий технологии гибридного параллельного программирования MPI+OpenMP+CUDA

Нижеприведённый листинг содержит программную реализацию функции, реализующей параллельный алгоритм 4.4 поиска регулряризованного решения переопределённой системы $Ax = b$ с плотно заполненной матрицей с учётом ошибок машинного округления. В программном коде используются технологии параллельного программирования MPI (стандарт MPI-4), OpenMP и CUDA. Программный код реализован на языке программирования Python, вычисления производятся с двойной точностью.

```
def conjugate_gradient_method(A_part, b_part, x_part, alpha,
                              comm_row, comm_col, N) :

    import cupy as cp

    A_part_d = cp.asarray(A_part)

    N_part = size(x_part);  M_part = size(b_part)

    x_classic_part = None
    delta = finfo(float64).eps

    requests = [MPI.Request() for i in range(5)]

    scalar_product_temp = empty(1, dtype=float64)
    scalar_product_rr = array(0, dtype=float64)
    scalar_product_pq = array(0, dtype=float64)

    Ap_part_temp = empty(M_part, dtype=float64)
    Ap_part = empty(M_part, dtype=float64)

    q_part_temp = empty(N_part, dtype=float64)
    q_part = empty(N_part, dtype=float64)

    criterion_temp = empty(1, dtype=float64)
    criterion = array(0, dtype=float64)
```

```
27
28      requests[0] = comm_row.Allreduce_init(
29                                [scalar_product_temp, 1, MPI.DOUBLE],
30                                [scalar_product_rr, 1, MPI.DOUBLE],
31                                op=MPI.SUM)
32      requests[1] = comm_row.Allreduce_init(
33                                [Ap_part_temp, M_part, MPI.DOUBLE],
34                                [Ap_part, M_part, MPI.DOUBLE],
35                                op=MPI.SUM)
36      requests[2] = comm_row.Allreduce_init(
37                                [criterion_temp, 1, MPI.DOUBLE],
38                                [criterion, 1, MPI.DOUBLE],
39                                op=MPI.SUM)
40      requests[3] = comm_col.Allreduce_init(
41                                [q_part_temp, N_part, MPI.DOUBLE],
42                                [q_part, N_part, MPI.DOUBLE],
43                                op=MPI.SUM)
44      requests[4] = comm_row.Allreduce_init(
45                                [scalar_product_temp, 1, MPI.DOUBLE],
46                                [scalar_product_pq, 1, MPI.DOUBLE],
47                                op=MPI.SUM)
48
49      s = 1
50      p_part = zeros(N_part, dtype=float64)
51
52      while True :
53
54          if s == 1 :
55              x_part_d = cp.asarray(x_part)
56              Ax_part_temp = cp.dot(A_part_d, x_part_d).get()
57              Ax_part = empty(M_part, dtype=float64)
58              comm_row.Allreduce([Ax_part_temp, M_part, MPI.DOUBLE],
59                                 [Ax_part, M_part, MPI.DOUBLE],
60                                 op=MPI.SUM)
61              b_part = Ax_part - b_part
62              b_part_d = cp.asarray(b_part)
63              r_part_temp = cp.dot(A_part_d.T, b_part_d).get()
64              r_part = empty(N_part, dtype=float64)
65              comm_col.Allreduce([r_part_temp, N_part, MPI.DOUBLE],
66                                 [r_part, N_part, MPI.DOUBLE],
67                                 op=MPI.SUM)
68              r_part = r_part + alpha*x_part
69
```

```
70              sigma2_r_part = zeros(N_part, dtype=float64)
71          else :
72              r_part = r_part - q_part/scalar_product_pq
73
74          scalar_product_temp[:] = array(dot(r_part, r_part),
75                                         dtype=float64)
76          MPI.Prequest.Start(requests[0])
77          MPI.Request.Wait(requests[0], status=None)
78          p_part = p_part + r_part/scalar_product_rr
79
80          p_part_d = cp.asarray(p_part)
81          Ap_part_temp[:] = cp.dot(A_part_d, p_part_d).get()
82          MPI.Prequest.Start(requests[1])
83          if s >= 2 :
84              sigma2_r_part =  sigma2_r_part + \
85                               q_part**2/scalar_product_pq**2
86          criterion_temp[:] = array(delta**2*sum(sigma2_r_part)/
87                                    scalar_product_rr,dtype=float64)
88          MPI.Prequest.Start(requests[2])
89          MPI.Request.Wait(requests[1], status=None)
90          Ap_part_d = cp.asarray(Ap_part)
91          q_part_temp[:] = cp.dot(A_part_d.T, Ap_part_d).get()
92          MPI.Prequest.Start(requests[3])
93          MPI.Request.Wait(requests[3], status=None)
94          q_part[:] = q_part[:] + alpha*p_part
95
96          scalar_product_temp[:] = array(dot(p_part, q_part),
97                                         dtype=float64)
98          MPI.Prequest.Start(requests[4])
99          MPI.Request.Wait(requests[4], status=None)
100
101         MPI.Request.Wait(requests[2], status=None)
102         if criterion >= 1 :
103             return x_part, s, x_classic_part
104
105         x_part = x_part - p_part/scalar_product_pq
106
107         s = s + 1
108
109         if s == N + 1 :
110             x_classic_part = x_part.copy()
```