Deutsche Geodätische Kommission
der Bayerischen Akademie der Wissenschaften

Reihe C Dissertationen Heft Nr. 761

Jan Martin Brockmann

On High Performance Computing in Geodesy
– Applications in Global Gravity Field Determination –

München 2015

Verlag der Bayerischen Akademie der Wissenschaften
in Kommission beim Verlag C. H. Beck

ISSN 0065-5325 ISBN 978-3-7696-5173-7

Diese Arbeit ist gleichzeitig veröffentlicht in:
Schriftenreihe des Instituts für Geodäsie und Geoinformation
der Rheinischen Friedrich-Wilhelms Universität Bonn
ISSN 1864-1113, Nr. 49, Bonn 2015

DGK Deutsche Geodätische Kommission

der Bayerischen Akademie der Wissenschaften

Reihe C Dissertationen Heft Nr. 761

On High Performance Computing in Geodesy – Applications in Global Gravity Field Determination –

Inaugural-Dissertation zur

Erlangung des Grades

Doktor-Ingenieur (Dr.-Ing.)

der Hohen Landwirtschaftlichen Fakultät

der Rheinischen Friedrich-Wilhelms Universität

zu Bonn

vorgelegt von

Dipl.-Ing. Jan Martin Brockmann

aus Nachrodt-Wiblingwerde

München 2015

Verlag der Bayerischen Akademie der Wissenschaften
in Kommission beim Verlag C. H. Beck

ISSN 0065-5325 ISBN 978-3-7696-5173-7

Diese Arbeit ist gleichzeitig veröffentlicht in:
Schriftenreihe des Instituts für Geodäsie und Geoinformation
der Rheinischen Friedrich-Wilhelms Universität Bonn
ISSN 1864-1113, Nr. 49, Bonn 2015

Adresse der Deutschen Geodätischen Kommission:

Deutsche Geodätische Kommission

Alfons-Goppel-Straße 11 ● D – 80 539 München

Telefon +49 – 89 – 23 031 1113 ● Telefax +49 – 89 – 23 031 - 1283 / - 1100
e-mail hornik@dgfi.badw.de ● http://www.dgk.badw.de

Diese Publikation ist als pdf-Dokument veröffentlicht im Internet unter den Adressen /
This volume is published in the internet

<http://dgk.badw.de> / <http://hss.ulb.uni-bonn.de/2014/3860/3860.pdf>

Prüfungskommission

Referent:	Prof. Dr. techn. Wolf-Dieter Schuh
Korreferenten:	Prof. Dr.-Ing. Jürgen Kusche
	Prof. Dr. Carsten Burstedde

Tag der mündlichen Prüfung: 21.11.2014

ISSN 0065-5325 ISBN 978-3-7696-5173-7

Summary

Autonomously working sensor platforms deliver an increasing amount of precise data sets, which are often usable in geodetic applications. Due to the volume and quality, models determined from the data can be parameterized more complex and in more detail. To derive model parameters from these observations, the solution of a high dimensional inverse data fitting problem is often required. To solve such high dimensional adjustment problems, this thesis proposes a systematical, end-to-end use of a massive parallel implementation of the geodetic data analysis, using standard concepts of massive parallel high performance computing. It is shown how these concepts can be integrated into a typical geodetic problem, which requires the solution of a high dimensional adjustment problem. Due to the proposed parallel use of the computing and memory resources of a compute cluster it is shown, how general Gauss-Markoff models become solvable, which were only solvable by means of computationally motivated simplifications and approximations before. A basic, easy-to-use framework is developed, which is able to perform all relevant operations needed to solve a typical geodetic least squares adjustment problem. It provides the interface to the standard concepts and libraries used. Examples, including different characteristics of the adjustment problem, show how the framework is used and can be adapted for specific applications. In a computational sense rigorous solutions become possible for hundreds of thousands to millions of unknown parameters, which have to be estimated from a huge number of observations. Three special problems with different characteristics, as they arise in global gravity field recovery, are chosen and massive parallel implementations of the solution processes are derived. The first application covers global gravity field determination from real data as collected by the GOCE satellite mission (comprising 440 million highly correlated observations, 80 000 parameters). Within the second application high dimensional global gravity field models are estimated from the combination of complementary data sets via the assembly and solution of full normal equations (scenarios with 520 000 parameters, 2 TB normal equations). The third application solves a comparable problem, but uses an iterative least squares solver, allowing for a parameter space of even higher dimension (now considering scenarios with two million parameters). This thesis forms the basis for a flexible massive parallel software package, which is extendable according to further current and future research topics studied in the department. Within this thesis, the main focus lies on the computational aspects.

Zusammenfassung

Autonom arbeitende Sensorplattformen liefern präzise geodätisch nutzbare Datensätze in größer werdendem Umfang. Deren Menge und Qualität führt dazu, dass Modelle die aus den Beobachtungen abgeleitet werden, immer komplexer und detailreicher angesetzt werden können. Zur Bestimmung von Modellparametern aus den Beobachtungen gilt es oftmals, ein hochdimensionales inverses Problem im Sinne der Ausgleichungsrechnung zu lösen. Innerhalb dieser Arbeit soll ein Beitrag dazu geleistet werden, Methoden und Konzepte aus dem Hochleistungsrechnen in der geodätischen Datenanalyse strukturiert, durchgängig und konsequent zu verwenden. Diese Arbeit zeigt, wie sich diese nutzen lassen, um geodätische Fragestellungen, die ein hochdimensionales Ausgleichungsproblem beinhalten, zu lösen. Durch die gemeinsame Nutzung der Rechen- und Speicherressourcen eines massiv parallelen Rechenclusters werden Gauss-Markoff Modelle lösbar, die ohne den Einsatz solcher Techniken vorher höchstens mit massiven Approximationen und Vereinfachungen lösbar waren. Ein entwickeltes Grundgerüst stellt die Schnittstelle zu den massiv parallelen Standards dar, die im Rahmen einer numerischen Lösung von typischen Ausgleichungsaufgaben benötigt werden. Konkrete Anwendungen mit unterschiedlichen Charakteristiken zeigen das detaillierte Vorgehen um das Grundgerüst zu verwenden und zu spezifizieren. Rechentechnisch strenge Lösungen sind so für Hunderttausende bis Millionen von unbekannten Parametern möglich, die aus einer Vielzahl von Beobachtungen geschätzt werden. Drei spezielle Anwendungen aus dem Bereich der globalen Bestimmung des Erdschwerefeldes werden vorgestellt und die Implementierungen für einen massiv parallelen Hochleistungsrechner abgeleitet. Die erste Anwendung beinhaltet die Bestimmung von Schwerefeldmodellen aus realen Beobachtungen der Satellitenmission GOCE (welche 440 Millionen korrelierte Beobachtungen umfasst, 80 000 Parameter). In der zweite Anwendung werden globale hochdimensionale Schwerefelder aus komplementären Daten über das Aufstellen und Lösen von vollen Normalgleichungen geschätzt (basierend auf Szenarien mit 520 000 Parametern, 2 TB Normalgleichungen). Die dritte Anwendung löst dasselbe Problem, jedoch über einen iterativen Löser, wodurch der Parameterraum noch einmal deutlich höher dimensional sein kann (betrachtet werden nun Szenarien mit 2 Millionen Parametern). Die Arbeit bildet die Grundlage für ein massiv paralleles Softwarepaket, welches schrittweise um Spezialisierungen, abhängig von aktuellen Forschungsprojekten in der Arbeitsgruppe, erweitert werden wird. Innerhalb dieser Arbeit liegt der Fokus rein auf den rechentechnischen Aspekten.

Contents

1. Introduction

Automatically and autonomously working sensors and sensor platforms like satellites deliver a huge amount of precise geodetic data allowing the observation of a wide range of processes within the System Earth. These sensors either deliver data with a high frequency or over long time periods like decades — or even both, leading to a significant increase of the data volume. Due to the design of the sensors, the observations are often highly correlated and sophisticated stochastic models are required to describe the correlations and to extract as much information out of the data as possible. Although such large data sets are difficult to handle, they allow the set up of increasingly complex functional models to describe for instance processes in the System Earth with enhanced temporal and/or spatial resolution. From these high quality data sets, model parameters are typically estimated in an adjustment procedure, as the resulting system of observation equations is highly overdetermined. Only if a realistic stochastic model of the observations is used, which often requires a huge numerical effort, a consistent combination of different observation types is possible, and the covariance matrix of the estimated parameters can be expected to deliver a realistic error estimate. The parameters together with the covariance matrix can be used in further analysis without loss of information.

Due to the increasing data volume, the three main components of the adjustment problem, i.e. the observations, the stochastic model of the observations and the functional model, require a tailored treatment to enable computations in a reasonable amount of time. In many geodetic applications, where such high dimensional data sets are analyzed, a wide range of simplifications and approximations (down sampling, model simplifications, interpolation to regular grids, disregarded correlations, approximate solutions, ...) are introduced on different levels of the data analysis procedure to reduce the computational requirements of the analysis. These approximations, of course, have an influence on either the estimation of the unknown parameters or on their accuracy estimates and thus on the quality of the output of the analysis. As these approximations and simplifications are very application specific, the effect cannot be generally quantified.

An alternative to the simplified modeling mentioned above is the use of concepts and methods of scientific and high performance computing (SC and HPC) to derive implementations of the analysis software which are able to solve the task with less simplifications in a reasonable amount of time. These methods either imply the use of more efficient algorithms or, as it is the focus of this thesis, the use of massive parallel implementations on high performance compute clusters. These massive parallel implementations then make the computationally motivated approximations (of the data or of the models) often decrepit or at least lead to a significant reduction of them.

This thesis represents a novel approach to comprehensively introduce the concepts of SC and HPC into geodetic data analysis. In contrast to existing approaches, where only parts of the least squares adjustment procedure are performed in a parallel way and decoupled software modules are applied as black box (e.g. for the inversion of matrices), this thesis proposes for the first time a systematical, end-to-end massive parallel implementation of geodetic data analysis using standard concepts of HPC. Therefore, a basic, easy-to-use framework is developed, which is able to perform all relevant operations needed to solve a typical geodetic least squares adjustment problem. Distributed storage of data and matrices is extensively used to achieve a best possible flexibility with respect to the dimension of the adjustment problem. The use of this framework is demonstrated for three examples arising in the field of global gravity field determination, where high dimensional adjustment problems with varying characteristics have to be solved. These examples show i) the flexibility of the framework to be specified for different applications, ii) the potential of the HPC approach with respect to the possible dimension of the adjustment problem and iii) the performance which can be achieved with such massive parallel implementations.

Within the first part of the thesis, the application unspecific concepts are introduced and the general HPC concepts used within an adjustment process are summarized. In Chap. 2 and 3 the basic methods are developed to map a general dense adjustment procedure (least squares adjustment) to massive parallel compute clusters. For that purpose, standard concepts from scientific and high performance computing are used to implement an interface for the standard operations needed for linear algebra operations (cf. Chap. 2). As in adjustment theory most operations are performed using matrices, the concept of block-cyclic distributed matrices is used and consequently applied in the implemented software package (cf. Chap. 3). A general framework for the handling of huge dimensional matrices is implemented in this chapter, intensively using the available standard concepts and libraries from HPC. Chap. 4 introduces the generalized form of the adjustment problem, the solution of which should be determined by the massive parallel implementation. The implemented methodology is summarized and special concepts required for data combination within the adjustment procedure are introduced.

Within the second part, the basics are applied and refined for solving three special problems with different characteristics as they arise in global gravity field recovery. Chap. 5 is the bridge from the general formulation of the concepts to the specific applications. It introduces the specific problem and summarizes the methods and the physical theory which is common for the three tasks. Some definitions and analysis concepts are provided to define the figures and quantities shown later in the application chapters. Besides the development of the basic framework an application specific massive parallel software package is developed for three applications, which are related to current research projects of the Theoretical Geodesy Group at the Institute of Geodesy and Geoinformation (IGG) at the University of Bonn. The applications are representatives for the challenges relevant for high dimensional adjustment problems: a huge number of highly correlated observations and a large to huge number of unknown parameters.

The first application (cf. Chap. 6) is the computation of global gravity field models from data observed by the GOCE (Gravity field and steady-state Ocean Circulation Explorer) satellite mission. The main challenge in this context is the processing of a huge number of observations: 440 million observations were collected during the whole mission period. In addition to the huge data volume, the observations measured along the satellites orbit are highly correlated in time, thus a complex decorrelation approach is needed, which is intensive with respect to computing time. Due to the mission design and the attenuation of the gravity field signal at satellite altitude, the resolution of gravity field models from those observations is limited such that a relatively moderate amount of 60 000–80 000 unknowns has to be estimated. Nevertheless, the resulting normal equation matrices have memory requirements of 30 GB–50 GB. As the developed software was used for real-data GOCE analysis, results from the real-data analysis are shown and discussed as well. The group is an official processing center within the ESA's GOCE HPF (High-Level Processing Facility). The software is used in the context of the production of ESA's official GOCE models.

As a second example in Chap. 7, a simulation study for high resolution global gravity field determination from a combination of satellite and terrestrial data is set up to demonstrate a massive parallel implementation of applications where a moderate number of observations are used to estimate a large number of unknown parameters, spanning a high dimensional vector space in the range of 10 000 to 600 000 unknowns. An objective of this application is to derive an implementation which solves the adjustment procedure via the assembly and solution of full normal equations such that afterwards a full covariance matrix is available, e.g. for a possible assembly of the estimated model into further process models. The simulation performed within this thesis assembles and solves full normal equations for 520 000 unknown parameters from about 4 million observations.

For the third application (cf. Chap. 8) the dimensions of the adjustment problem are even further increased by introducing a huge dimensional parameter space that cannot be estimated by direct solution of the normal equation system. Therefore, a massive parallel implementation of an iterative solver is derived enabling the rigorous solution of adjustment problems with hundreds of thousands to millions of unknown parameters. This way rigorous, non-approximative solutions become possible for such large problems, even though a covariance matrix is not directly available. The method is first demonstrated for the 520 000 unknown parameters and about four million observations used in the second example, and afterwards the problem is expanded to solve for two million parameters from about twenty million observations.

Although the basic framework is adapted to specific applications from global gravity field recovery only, the concepts can easily be transferred to any other adjustment problem typical in geodetic data analysis (mainly implementing the specific observation equations and the required decorrelation concept). For the applications and simulations shown, the focus is not on the design of alternative processing methods or on a perfect physical modeling, but on the massive parallel implementation of the mostly well known concepts and statistical methods. In this way, this thesis fuses concepts from informatics, statistics, mathematics and geodesy to derive a massive parallel implementation which solves an inverse geodetic data fitting problem. It contributes to solve the challenges arising from the computational point of view on the way towards a more rigorous geodetic data analysis. The derived software package makes analyses possible, which have – due to computational limits – not been solvable before. Avoiding the widely used — often historical — approximations in geodetic data analysis leads to improved geodetic products due to HPC.

Including analysis software components, a software package with more then 35 000 lines of massive parallel C++ code was developed within this thesis. As only (quasi-) standard concepts and libraries were used, the software is highly portable and is able to run on every HPC compute cluster and enables the use of up to tens of thousands of compute cores.

Parts of this thesis have already been published in Brockmann et al. (2014b) and Brockmann et al. (2014c).

Part I

Basic Framework for a Massive Parallel Solution of Adjustment Problems

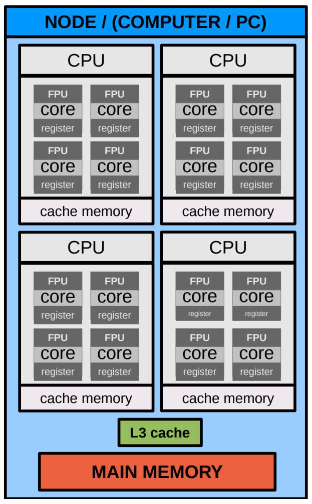
2. Standard Concepts of High Performance and Scientific Computing

The goal of this chapter is to give an introduction into high performance and scientific computing (HPC and SC) concepts. Whereas SC in general typically covers the numeric solution of scientific problems, HPC covers the massive parallel implementations of the SC processing concepts and numerical experiments on distributed memory compute clusters. What is called "just" implementation here, covers the conversion of sequential specialized algorithms for the concept of parallel computing and their efficient implementation in a HPC environment. Some definitions of terms from the area of scientific and high performance computing are provided. As this terms are not uniquely defined across different scientific areas, definitions as used in this thesis are given. Afterwards, basic concepts of SC, especially matrix related concepts, are introduced. As the rigorous computation of applications from the field of adjustment theory is the focus of this thesis, the computational concepts introduced are mainly matrix and matrix-based operations from linear algebra. The concepts mentioned and all associated libraries are (quasi-) standards in scientific and high performance computing.

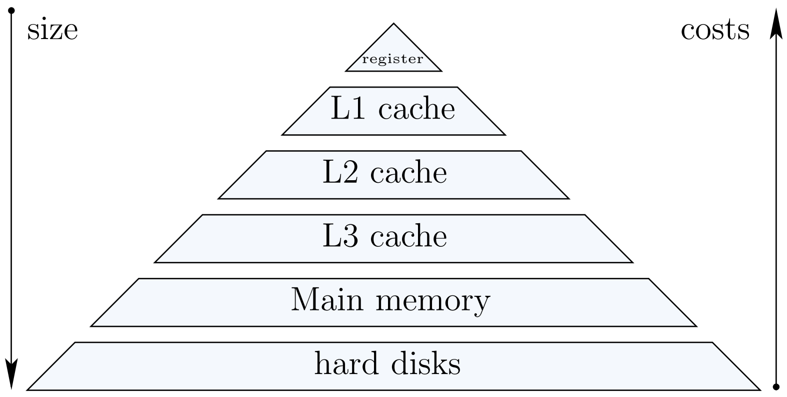
Within this thesis, many algorithmic descriptions of the implemented steps are provided. They do not claim to be complete, but should show the general process of the implemented software and should be read as a summary. Parts of the algorithms, which are from an implementational point of view complex and would require many details, are often hidden in a descriptive symbol to avoid details, which would make the algorithms unreadable. Nevertheless, within the text, the details are explained. In addition to the algorithms, some header files are provided, which should give an overview of some basis classes implemented. These header files often only show excerpts from the actual header file, as they are often to long. Special symbols and the syntax used within this thesis is not explicitly introduced, but a descriptive list is provided in Appendix A.

2.1 Introduction, Terms and Definitions

A single computer is an autonomous working unit with the (in this context) important components as depicted in Fig. 2.1(a). This computer, which is often called *compute node* in context of HPC, has a certain number of *processors* (i.e. Central Processing Units, CPUs). Each CPU consists of a certain number of *compute cores*. Each of the cores can perform instructions independently from the other cores. Every compute node has a certain amount of *main memory*, which can be addressed by every processor and by every individual core within the node. As the access to the main memory is slow compared to the floating point operations performed by the CPU, the memory access is the limiting factor for numerical computations (Von Neumann bottleneck, e.g. Bauke and Mertens, 2006, p. 7). To circumvent the bottleneck, a smaller but faster memory is integrated into the processors to cache the effect of the slow main memory. Compared to the main memory, this so called *cache memory* is faster but significantly smaller (e.g. cf. Rauber and Rünger, 2013, Sect. 2.3.3, Sect. 2.7). Even in shared memory multi-processor systems, as the cache memory is integrated into the processor, the cache memory can only be accessed by its own processor and its cores. The memory is hierarchical organized as demonstrated in Fig. 2.1(b). Typically two levels of cache memory are integrated into a processor, i.e. the very fast level 1 cache and the larger but slower level 2 cache. Especially in shared memory systems a level 3 cache is mounted outside the processors, between the main memory and the processors. This is mainly used for a fast exchange of data between multiple processors. More detailed descriptions can be found for instance in Bauke and Mertens (2006, Chap. 1), Karniadakis and Kirby (2003, Sect. 2.2.6), Rauber and Rünger (2013, Chap. 2) and Dowd and Severance (1998, Chap. 3).



(a) Parts of a compute node.



(b) Memory hierarchies, modified from Karniadakis and Kirby (2003).

Figure 2.1: Important components of a compute node.

The main memory itself can be seen as a one-dimensional linear addressable vector (Karniadakis and Kirby, 2003, p. 41), a sequence of bits, where 8 bit are grouped as one byte (1 B). Each byte can be uniquely accessed via an address (typically an hexadecimal number). Using one dimensional fields (i.e. arrays) in programming languages like C++ guarantees that the array elements are stored consecutively without gaps in the memory. Consequently, all elements of the array can be accessed via the address (i.e. a pointer variable which can store addresses in C++) of the field's first element in the main memory and the length information of the field (i.e. the number of entries).

Launching a standard compiled *program* (e.g. implemented in C++), a *process* of the program is started and the instructions are executed by a single core of one of the processors. It does not matter how many cores and processors are available on the compute node. Only when special multi-threading concepts are used within the implementation of a program, or other special parallel programming concepts are used (as introduced later), the program runs on more than a single core and thus uses the computing power of the additional cores and/or processors.

2.2 Matrices, Computers and Main Memory

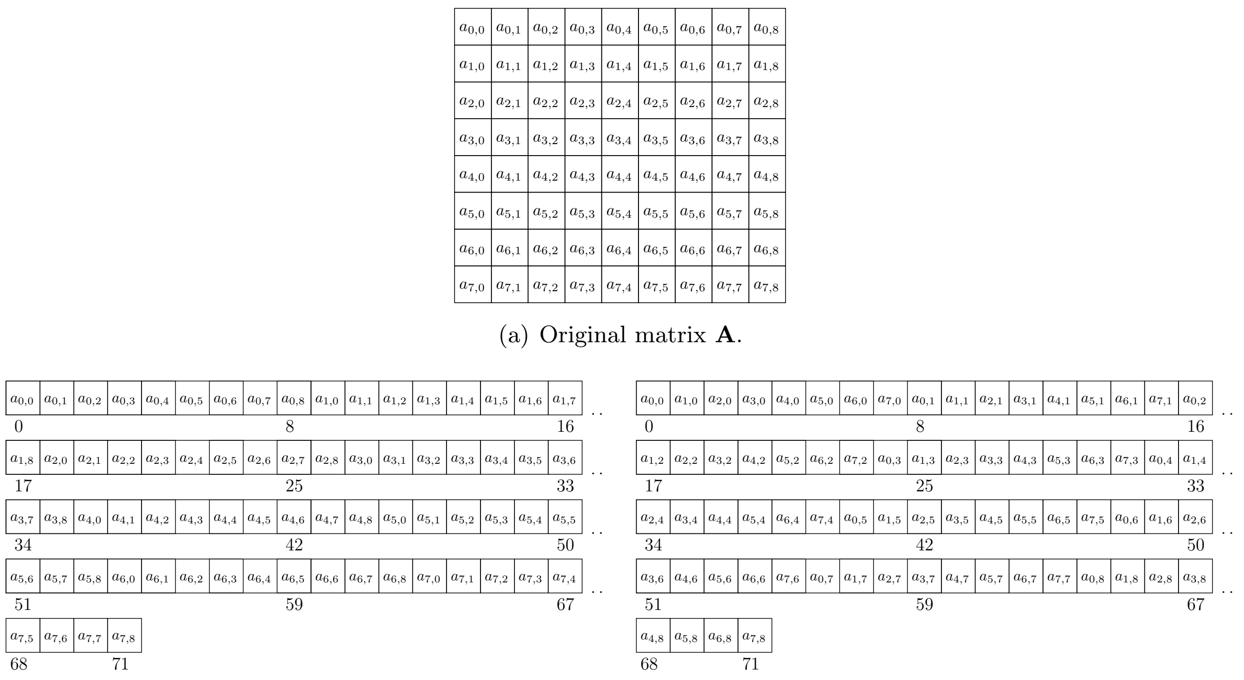
A matrix is a two-dimensional field whose entries are described by two coordinates, e.g. (r, c). Within this work, we will use the indexing of the matrix entries starting with zero[1]. A matrix of dimension $R \times C$ with $N = RC$ elements is written as

$$\mathbf{A} = \begin{bmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,C-1} \\ a_{1,0} & a_{1,1} & \dots & a_{1,C-1} \\ \vdots & \vdots & \vdots & \vdots \\ a_{R-1,0} & a_{R-1,1} & \dots & a_{R-1,C-1} \end{bmatrix} = \begin{bmatrix} \mathbf{A}(0,0) & \mathbf{A}(0,1) & \dots & \mathbf{A}(0,C-1) \\ \mathbf{A}(1,0) & \mathbf{A}(1,1) & \dots & \mathbf{A}(1,C-1) \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{A}(R-1,0) & \mathbf{A}(R-1,1) & \dots & \mathbf{A}(R-1,C-1) \end{bmatrix}. \tag{2.1}$$

2.2.1 Linear Mapping of a Matrix to the Main Memory

Performing standard computations involving matrices or performing linear algebra operations on matrices in a lower-level programming language, the matrix – the two-dimensional field – has to

[1]It is standard for indices in some programming languages, e.g. in C++ (e.g. Stroustrup, 2000, p. 28), which is used within this thesis.



(a) Original matrix **A**.

(b) **A** mapped in RMO to **a**.

(c) **A** mapped in CMO to **a**.

Figure 2.2: Example Matrix **A**, and **A** mapped to a linear vector **a** using RMO and CMO.

be mapped to the one-dimensional memory of the compute node. This is usually done via the mapping of a matrix into a one dimensional field, which might be an array or in C++ the advanced dynamic `std::vector<double>` class as provided by the Standard Template Library (STL, e.g. Kuhlins and Schader, 2005). In lower-level programming languages, these one-dimensional fields have the property, that their entries are stored continuous in the main memory. The matrix is mapped to a vector (one-dimensional field), which is called **a** in the following. This vector can then be linearly mapped to the computers main memory. There are two different concepts for mapping (general) matrices to the computers memory, i.e. column major order (CMO) or row major order (RMO) (e.g. Karniadakis and Kirby, 2003, p. 41). All entries of the matrix are accessible, if the address of the first element in memory and the dimension ($R \times C$) of the matrix is known.

Column Major Order – CMO Within CMO, the matrix is stored column by column in the vector **a** of length $R \cdot C = N$. The example matrix shown in Fig. 2.2(a) results in the vector shown in Fig. 2.2(c). To reference a matrix element (r, c) in the vector, two quantities, which are step sizes in the vector, are introduced. There are always two steps in the vector or in the linear memory, respectively: Firstly the large step in the vector, the so called *leading dimension* (l_d), which must be covered going from element i of column c to element i of column $c+1$. In CMO this step equals exactly the number of rows of the matrix **A**, i.e. $l_d = R$. The small step in the vector is the step which is covered going from one row element r of column c to the next element in the same column, i.e. $r+1$, the so called *increment* (i_c) which is equal to $i_c = 1$ in the CMO.

Thus, we can use

$$i\,(r, c) = cl_d + ri_c = cR + r. \tag{2.2}$$

to determine the index i of matrix element $\mathbf{A}\,(r, c)$ in the vector **a** so that $\mathbf{a}(i) = \mathbf{A}\,(r, c)$. This index describes the position in memory relative to the first element. The address can thus be determined via

$$\&\mathbf{A}\,(r, c) = \&\mathbf{a}\,(0) + i\,(r, c)\,, \tag{2.3}$$

using the C++ address operator &. Or vice versa, the element at position i corresponds to the element with the row and column index

$$r = i\%R, \quad c = i \div R, \tag{2.4}$$

where the symbol % is the modulo operator, which returns the remainder of the integer division and ÷ is used for the integer division. With (2.2) and (2.4) the mapping between **a** and **A** is uniquely defined. The vector **a** is directly mapped to the linear memory of the computer, using a one dimensional field. All elements of the vector can be accessed via the address of the first element, &**a**(0), the number of rows and columns of the matrix and the resulting index of an element (r, c) can be determined using (2.2).

Row Major Order – RMO Instead of grouping the matrix column-wise into a one-dimensional vector, one can decide to group the matrix row-wise into a vector. This results in RMO. The example matrix from Fig. 2.2(a) would result in an vector as shown in Fig. 2.2(b). Obviously, the resulting vector **a** is of same dimension $R \cdot C \times 1 = N \times 1$ as for the CMO.

However, the access step sizes in memory change. The large step in memory, i.e. the step going from one row r to the next row $r+1$ is $l_d = C$, the small step in memory, accessing the next element of the same row is $i_c = 1$. Thus,

$$i\,(r,c) = c i_c + r l_d = c + rC. \tag{2.5}$$

is used to determine the index i of matrix element $\mathbf{A}\,(r,c)$ in the vector **a**. The address in memory of the element follows from

$$\&\mathbf{A}\,(r,c) = \&\mathbf{a}\,(0) + i\,(r,c)\,. \tag{2.6}$$

The inverse operations

$$r = i \div C, \quad c = i\%C. \tag{2.7}$$

can be used to determine r and c from a given vector index i.

Comparing both Mapping Schemes Both methods of mapping a matrix to the linear computer vector are equivalent. None of both schemes is generally superior to the other with respect to performance, if the algorithms are properly adapted. Depending on the algorithm which operates on a matrix, one of them may be more efficient with respect to performance (for a performance analysis and a comparison to alternatives see e.g. Thiyagalingam, 2005, Thiyagalingam et al., 2006, Chap. 5). As none of both schemes is perfect, it is useful to decide for one within a certain project. Within this work, the column major storage scheme was chosen, and will be used in the following.

2.2.2 File Formats for Matrices

As needed later on as well, a simple but flexible binary file format for matrices is summarized here. The same format is going to be used for parallel Input/Output (I/O) operations of matrices which are stored distributed over several compute nodes. As it will be of importance within this work, the same idea of a one-dimensional view on a matrix as for the mapping into main memory is used to save the matrix within a binary file. Comparable to the main memory, a binary file can be seen as a one-dimensional field (of single bytes) as well.

First of all, a binary header of fixed size (in Bytes) is written to the file, containing at least the metadata of the matrix, i.e. the dimension (R and C, 2 integer numbers, 8 B). Additional metadata (special matrix properties like symmetry,...) can be stored as well, as long as the size (i.e. the number of Bytes) of the header is known. The header is followed by the $R \cdot C$ matrix entries, stored as $R \cdot C$ `double` numbers in CMO or RMO. Each `double` number requires 8 B. Thus, the bytes can be continuously written from memory into a file. Of course the same mapping as for the memory should be chosen for the file. As sequential binary files allow for high performance I/O operations (file size as well as reading time), the focus is on binary files only.

2.3 Standard Concepts for Matrix Computations and Linear Algebra

Efficient matrix computations and linear algebra operations are standard operations in SC and HPC (e.g. Dongarra et al., 1990a, Karniadakis and Kirby, 2003, Chap. 2.2.7). Starting with early initiatives (esp. by Lawson et al., 1979, Dongarra et al., 1988, 1990a) standard libraries for basic vector-vector (level 1, L1), matrix-vector (level 2, L2) and matrix-matrix (level 3, L3, highest optimization potential) operations were established in the fields of SC and HPC. The library denoted as “Basic Linear Algebra Subprograms” (BLAS, Dongarra et al., 1990a) became a standard library in SC covering numerical analysis. Tailored implementations of the basic subroutines for matrix and vector operations, which are organized in three levels, are available as optimized versions for special processor architectures (i.e. hardware). Due to the standard, programs using the BLAS routines can be efficiently used on various computers, just linking BLAS routines optimized for that architecture. The basic optimization concepts refer to block-algorithms for the matrix computations, which efficiently exploit the hierarchical organized cache memory. Detailed descriptions of the optimization concepts can be found in several dedicated papers. As a starting point see Lawson et al. (1979), Dongarra et al. (1988, 1990a). A nice overview of the concepts used is given by Karniadakis and Kirby (2003, Chap. 2.2.7).

In contrast to hardware optimized BLAS routines, the ATLAS-Project (Automatically Tuned Linear Algebra Software, Whaley et al., 2000, Whaley and Dongarra, 1997) automatically tunes the parameters of the BLAS routines with respect to the specific hardware, where the ATLAS library is compiled. A priori hardware information and empirical runtime measurements are used to derive the hardware dependent parameters. Close to optimal BLAS routines can be compiled on nearly every platform, thus, in addition to performance, programs using the BLAS are highly portable without loss of performance.

As an extension to the vector-vector, matrix-vector and matrix-matrix operations contained in the BLAS, the Linear Algebra PACKage (LAPACK, Anderson et al., 1999, 1990) provides the most common linear algebra routines used in SC and HPC. For instance, matrix factorizations, eigenvalue computations, solvers for linear systems and matrix inversions are contained in the LAPACK library which provides all in all several hundred of routines (Anderson et al., 1990). As the basic computations within LAPACK again extensively use the BLAS routines, LAPACK can obtain a great performance on nearly every hardware, again just linking a tailored BLAS library.

Both the BLAS and the LAPACK library use a simple interface to matrices. A matrix is passed to the routines via a pointer to the first matrix element in memory and the dimension information of the matrix (number of rows and columns as well as the increment and the leading dimension to operate on sub-matrices). The routine can either operate on matrices stored in CMO or RMO. This is the reason why only the standard concepts of CMO or RMO were addressed in Sect. 2.2.1.

2.4 Implementation of a Matrix as a C++ Class

As it is a basis in the main part, i.e. the development of a class for distributed matrices in the next chapter, some details of an implementation of a class for a matrix in C++ are given. Without going into detail, Listing 2.1 shows an excerpt of a possible C++ class implementation in form of a header file. The basic features and functions are provided in the header, but not all member functions implemented are mentioned. For all computations possible, BLAS or LAPACK routines are used inside the member functions to efficiently perform the serial computations. The interface to the BLAS and LAPACK library is thus hidden in the class implementation. The main features of the class are:

- The two-dimensional data is mapped to the one-dimensional field (`std::vector<double>`).
- Data access via several member functions (e.g., `double operator()( size_t r, size_t c) const`).
- Data manipulation via member functions (e.g., `double & operator()( size_t r, size_t c)`).
- Data manipulation and access via pointers (e.g., `double * data()`).
- Column-wise access via pointers (e.g., `double * data( size_t col )`).
- BLAS and LAPACK functionality added in member functions for computing routines (e.g., `void chol()`).
- ASCII and binary based file I/O (read and write).

This basic class for matrices serves as a basis for the parallel computations and the block-cyclic distributed matrices introduced in the next chapter.

Listing 2.1: Simple header file defining the main features of the class `Matrix`.

```cpp
#ifndef MATRIX_H
#define MATRIX_H

#include <vector>
#include <iostream>
#include <math.h>
#include <algorithm>
#include <iterator>
#include "blas.h"
#include "lapack.h"
#include "mpi.h"

using namespace std;
class Matrix
{
    public:
        // constructors
        Matrix( );
        Matrix( size_t r, size_t c );
        Matrix( const Matrix & A );
        // destructor
        ~Matrix( );
        // access to data and meta data
        size_t rows( ) const;
        size_t cols( ) const;
        size_t ld( ) const;
        size_t inc( ) const;
        double operator()( size_t i, size_t j ) const;
        const double* data() const;
        const double* data( size_t col ) const;
        // manipulate matrix
        Matrix & operator=( const Matrix & A );
        void resize( size_t nrows, size_t ncols );
        double* data();
        double* data( size_t col );
        double & operator()( size_t i, size_t j );
        // Computing functions interface to BLAS/LAPACK (not listed completely)
        Matrix operator*( const Matrix & B );
        Matrix operator+( const Matrix & B );
        Matrix & operator*=( const Matrix & B );
        Matrix & operator+=( const Matrix & B );
        void plusProductOf( const Matrix & A, const Matrix & B, double w, char transA, char transB );
        void plusATAOf( const Matrix & A, char trans);
        void chol();
        void invert();
        void eigenvals( Matrix & evals );
        void eigenvecs( Matrix & evals, Matrix & evecs );
        ...
        // Matrix I/O
        void binarySave( string filename ) const;
        void binaryRead( string filename );
        // communicating matrices with MPI
        void Send( int toRank );
        void Recv( int fromRank );
        void Bcast( int fromRank );
        void Reduce( int onRank, const MPI::Op& operation );
        void Scatterv( int fromRank ); // distribute rows only
        void Gatherv( int fromRank );  // collect rows only
    private:
        // Dimension of matrix
        size_t _R, _C;
        // entries of matrix as vector
        std::vector<double> _data;
};
#endif // MATRIX_H
```

3. Standard Concepts for Parallel Distributed High Performance Computing

Handling huge adjustment problems requires on the one hand a lot of computing power for the computations and on the other hand the treatment of large and in some cases huge matrices. Especially within the modeling of physical processes within the System Earth, huge data sets are analyzed and more and more refined models are set up, whose parameters are often estimated within an adjustment procedure from the data. These adjustment procedures often produce huge dense systems. Thus, to analyze available data sets and to adjust huge dimensional parameters of complex models the computational requirements significantly increase with autonomous sensors like e.g., instruments carried on satellite platforms. The computational requirements can often not be handled by a single compute node, especially if a rigorous modeling without significant computational approximations is aimed for.

To handle the computational burden, parallel implementations are used to perform the computations in a reasonable amount of time and to operate with large models and huge dimensional adjustment processes. Thus, the joint computing power and the memory of a set of compute nodes can be used to solve the computational tasks. For tasks requiring linear algebra on huge matrices, a concept for the distribution of a matrix along the set of compute nodes is required to make use of the joint memory. These distributed matrices are then used for rigorous computations in the algorithms avoiding approximations which reduce the computational and memory requirements. Within this thesis, it is generally assumed, that full systems and thus dense matrices are needed for a rigorous modeling. If not explicitly stated, the matrices are not sparse and thus the operations are needed for dense matrices.

The goal of this section is to summarize the standards from parallel distributed HPC which are used later on to solve the tasks which are summarized in the application Chapters 6–8. The basic concepts to be used there, are introduced and reviewed in this chapter and the basic implementation, the specific application implementations are based on, is summarized here.

3.1 Definitions in the Context of Parallel and Distributed HPC

Before going into details of the main topic, i.e. parallel distributed HPC, some important terms need to be defined. The main goal within this thesis is to derive implementations of algorithms which are able to operate on many compute nodes and are thus able to make use of the joint resources of the nodes. This resources are the computing power (of all cores) as well as the distributed main memory. The idea is to have a set of individual (stand-alone) compute nodes which are connected via a network switch. This might be a collection of standard PCs or dedicated compute nodes installed for that purpose only. In spite of dedicated hardware and thus performance of the final program, there is no difference between network connected standard PCs and connected dedicated compute servers from a conceptual and implementational point of view. Thus, the general term *compute cluster* is used for the network connected ensemble of compute nodes (which might be standard PCs, workstations or dedicated compute servers). Of course, nowadays, these individual nodes are multi-processor and multi-core nodes. Fig. 3.1 gives an overview of a compute cluster sketching the main components.

Although in Fig. 3.1 all nodes are depicted with the same hardware, there is no need for that (e.g. number of processors or amount of main memory may vary), even the performance characteristics of the nodes may vary. Each node has a certain number of processors, where again each processor

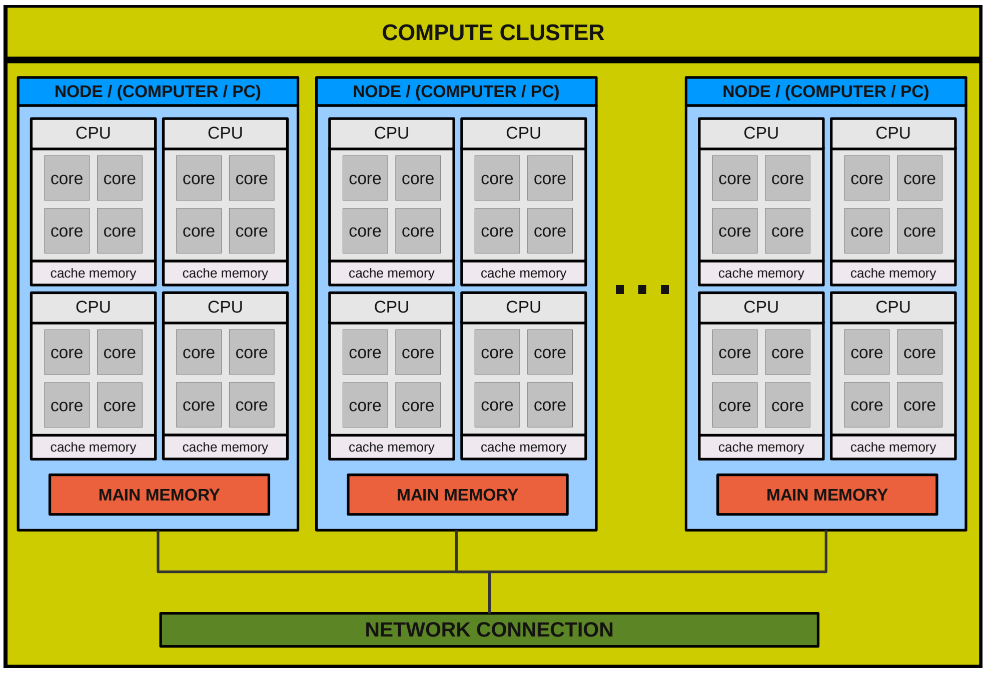


Figure 3.1: Components of a compute cluster.

has a certain number of compute cores. Within this general context, the number of cores N of the cluster is much more important than the number of processors or nodes. Only in special scenarios, where distributed parallel concepts (as focused on here) are mixed with multi-threading concepts (not addressed in detail in this thesis), the number of processors/cores per node gets important. Within this context, it does not matter for the chosen parallelization concepts if a cluster consists of 25 nodes with two processors each and again with four cores each (all in all 200 cores) or 200 nodes each of them equipped with a single core processor.

Although these details are not important for the conceptual design and the implementation, they are important for the final performance. Especially the performance of the individual cores involved in the cluster must be comparable and the network connection between the nodes should be as fast as possible. In addition the connection to a file-server which should serve as a data server should be fast. As every core has only a direct access to (parts) of the local nodes main memory, the network connection between the nodes is used to share (intermediate) results between the processes of the software running on the individual cores. The first standard concept which is needed for the development of parallel programs is a concept of sharing data between the compute cores of a cluster via the network connection.

3.2 A Standard for Distributed Parallel Programming: MPI

Within HPC, a common standard for the development of massive parallel programs exists. This standard, the Massage Passing Interface (MPI) Standard (MPI-Forum, 2009), is the basis for every massive parallel software for HPC. Different implementations of this standard; i.e. for instance OpenMPI (Gabriel et al., 2004), IntelMPI (Intel, 2013) or MPICH (Balaji et al., 2013) provide basic features for the development of massive parallel programs for the use on compute clusters making use of one to several thousands cores[2].

[2]Note that an alternative to MPI exits, i.e. the Parallel Virtual Machine (PVM, Sunderam, 1990, Geist et al., 1996). As it is not used within this thesis, but provides similar features and might be an option for some readers, the references are provided as a starting point for a comparison. In context of numerics and linear algebra MPI is (currently) more commonly used.

As the full name of that standard library indicates, the basic feature provided by MPI is an interface for the communication of messages (i.e. data) between processes of a parallel program being executed on different cores and nodes of a compute cluster via the network connection. The basic idea and some basic features are summarized in the following. There is no syntax provided as it is very well documented e.g. in Gropp et al. (1999a,b), Karniadakis and Kirby (2003) or Aoyama and Nakano (1999). Comparable to the BLAS and LAPACK libraries, the interface to the functions is realized via a data pointer to the data and an integer number referring to the number of elements (to be communicated).

3.2.1 Basic MPI Idea and Functionality

The MPI implementations provide startup scripts, which allow to start a program serial on N cores provided as a list of hosts (i.e. a node list and a number of cores per node, nodes are specified via the IP address or the hostname). N instances of the same program are launched on the N cores such that N *processes* of the same program are running on N cores. Without using special MPI commands in the program, so far only the same serial program is executed N times on different cores. Every core executes the instructions in the program and thus performs exactly the same operations. All variables created in the program are local with respect to the process and exist in in the (local) memory of every core. They have a local content and can only be modified by the process itself as the memory is only accessible by the core. The programmer is responsible to use special MPI functions to achieve, that every core works on a partial problem or on partial data set and consequently, the whole problem is solved in parallel using the resources of the N cores involved.

To achieve that, MPI arranges all processes involved in a so called *communicator* and assigns a unique identifier to every process which is called *rank* in the context of MPI. This rank n is an integer number $\mathsf{n} \in \{0, \dots, \mathsf{N}-1\}$ which can be used by the programmer to assign different tasks to individual processes or to achieve that every process applies the same instructions but to different parts (i.e. subsets) of the data.

In addition to the organizational features, MPI provides communications routines which can be used to communicate data between the processes (send and receive operations and extensions based on that). In this context, the rank is used as an address for the message passing. These concepts are addressed in some more detail in the following. A nice introduction to the development of MPI programs is given in e.g., Karniadakis and Kirby (2003) or Rauber and Rünger (2013, Chap. 5). Detailed information about all functionalities can be found in the MPI standard (MPI-Forum, 2009) and in Gropp et al. (1999a,b).

Point-to-Point Communication So called point-to-point communication can be used to share data between exactly two processes. A process with rank n_1 can send data to another process with rank n_2. For that purpose, process n_1 has to call a MPI send routine to send the data and in addition it has to be guaranteed by the programmer that n_2 allocates memory for the data to be received and calls the proper MPI receive function. Different send and receive function exist, mainly differing in the return behavior (blocking vs. non-blocking, buffered vs. synchronous). These different sends (and receives) are explained in detail for instance in Gropp et al. (1999a, Chap. 2–4).

Collective Communication Besides point-to-point communication, MPI libraries provide routines for collective communication, i.e. communication where all processes of the communicator are involved as senders and/or as receivers. E.g., data contained in the local memory of one process is send to all other processes of the communicator (broadcast), data stored in the local variables (memory) of the cores is collected in the memory of a single core and concatenated with an operation (reduce), or data which is stored in the local memory of a single process is regularly distributed over all processes (scatter(v)). The following main collective operations exist (see e.g. for a detailed complete description Gropp et al., 1999a, Aoyama and Nakano, 1999) and for the theoretical concepts (Rauber and Rünger, 2013, Sect. 3.6.2):

- `Bcast`: Distribute data available on a single process as a copy to all processes of the communicator.
- `Gather(v)`: Collect data regularly distributed over all processes consecutively in an array on a single process (inverse to `Scatter(v)`).
- `Scatter(v)`: Distribute data stored in an array on a single process regularly over all processes (inverse to `Gather(v)`).
- `Reduce`: Collect the content of variables (of same dimension) from all processes and concatenate them with an operation (sum, minimum,...). The result is stored on a single process.
- Advanced combinations of the functionalities mentioned above (e.g. a `Reduce` operation followed by a `Bcast`).

Advanced Features Some advanced MPI features, which are partially used in the following, are summarized in more detail. References to the technical details are given, as a technical overview would extend the scope of the summary of basic concepts:

MPI Topologies and Intra-Communicators (Gropp et al., 1999a, Chap. 4.2, 7.4):
A standard MPI communicator can be visualized as a one-dimensional vector, where all processes are arranged according to their ranks. For many applications, an alternative virtual arrangement of the processors is better suited, e.g. when thinking about data distribution. For instance, if later on a two dimensional matrix is distributed, it is straightforward to distribute it over a two dimensional processor grid instead of a one dimensional linear one. For such cases MPI provides the concept of virtual topologies, which allows to set up the communicator in special topologies (with the corresponding neighboring information). This topologies might be one-dimensional Cartesian grids (standard), two-dimensional, or more general n-dimensional Cartesian grids but also special topologies as graphs. For many algorithms a virtual arrangement of the processes as such topologies might be helpful to produce easier code e.g. for message passing to neighbors. For example using a two-dimensional Cartesian grid, 2D coordinates can be used to address the processes for message passing instead of the one dimensional rank, which makes the handling of neighboring processes easier. In addition to the implementational benefits, the MPI topologies can be linked to the network topology used for the network connection of the cluster nodes. A technical and theoretical overview about the design of topologies for network connections is given in (Rauber and Rünger, 2013, Sect. 2.5).

In addition to the arrangement of the processes as topologies, MPI provides the functionality to define sub-communicators in those topologies (e.g. grouping all processes of a column in a 2D Cartesian topology into an additional communicator). This enables to call collective communication for the processes of the sub-communicator only. For many algorithms, that is a useful extension which helps to organize the data communication.

The concept of Cartesian grids provided directly by MPI is not used here. Instead, an alternative with an extention to MPI is used, which arranges the processes as a Cartesian two-dimensional grid. This is a prerequisite for an implementation of the concept of block-cyclic distributed matrices as introduced later on in this chapter.

Parallel Data In- and Output (Gropp et al., 1999b, Chap. 3):
At some points in HPC, data I/O plays an important role. Analyzing huge data sets and operating with large matrices (e.g. normal equations or covariance matrices in adjustment theory with several

GB to TB in size) requires efficient I/O. In addition to communication routines and to the organization of communication, MPI provides a concept for parallel I/O from/to binary files. A framework for parallel file access is provided as well as seeking functions such that all processes can access and read from the file in parallel at different positions. The performance of the parallel I/O of course mainly depends on the filesystem (and its network connection) the file is read from. In addition to normal (partial) reading and seeking, process specific file masks exist which can be used to create process specific views on files so that each process only "sees" the data which should be read by the process following a user defined distribution scheme.

Extension of the Matrix Class with MPI Communication Functionality Sequences of MPI function calls can be easily used to implement communication routines as member function of classes. Listing 2.1 shows some functions for communicating objects of the matrix class. Most of them are implemented with a sequence of MPI basic communication routines. For instance sending a matrix requires three MPI send calls. First of all the two integers are send representing the dimension of the matrix (e.g. two sends of a single integer). Afterwards, with a third send call, the data is send as a RC double values. Consequently, the receive function is implemented using 3 basis MPI receive calls. Two are performed to receive the dimension of the matrix, this information is used afterwards to adjust the dimension of the receiving `this` matrix and thus to allocate the data memory. The third call then receives directly RC double values and writes them into the associated data vector **a**. As these implementations are straightforward, they are not discussed in more detail. These functions are mentioned here as the next sections shows a first simple implementation of a parallel least squares adjustment as it can be realized with basic MPI usage.

3.2.2 Simple MPI Programs to Solve Adjustment Problems

A very simple MPI based implementation of a least squares adjustment procedure is presented here. It is used as a motivation for the MPI usage and to illustrate the basic MPI concept. It demonstrates, and helps to get into, parallel MPI based thinking. This simple program computes least squares estimates for coefficients of a Fourier series given observations $\boldsymbol{\ell}$ at points $\mathbf{t}$. The observations are uncorrelated and have equal variances, thus the weight matrix is the identity matrix $\mathbf{I}$. The observation equations (OEQs) are set up and the system of normal equations (NEQs, $\mathbf{N}$ and $\mathbf{n}$) is assembled. Afterwards, the NEQs are solved to derive the solution and $\mathbf{N}$ is inverted to derive the variance covariance matrix of the parameters. The whole program as a parallel MPI implementation is listed in Listing 3.1. Most original MPI calls are consciously hidden in the implementation of member functions of the matrix class. Only the calls to the self implemented member functions are visible.

Simple Parallelization Concept for Adjustment Problems A very simple parallelization concept for adjustment procedures can be implemented if the following prerequisites hold:

- The observations are assumed to be uncorrelated (no correlations exist in the covariance matrix, it is a diagonal matrix).
- The NEQs are limited in their dimension and fit into the main memory of every core involved (limited parameter space).

Assuming the parameter space to be small, but the number of uncorrelated observations M to be huge, the assembly of the NEQs can be parallelized very well. For uncorrelated observations the addition theorem for NEQs holds (e.g. Koch (1999, p. 177), Meissl (1982, Sect. A.10.2)) which says that the NEQs can be separately computed for portions of the observations independently

Listing 3.1: Example of a simple parallelization of an adjustment problem with MPI.

```
 1  using namespace std;
 2  int main( int argc, char *argv[] )
 3  {
 4    // Initialize MPI
 5    MPI::Init( );
 6    // initialize a timer
 7    TicToc timer; timer.tic();
 8    // determine how many processes were started
 9    int size = MPI::COMM_WORLD.Get_size();
10    int rank = MPI::COMM_WORLD.Get_rank();
11    // initialize (empty) matrices
12    Matrix l,t,A,N,n;
13    // only the master process (rank == 0) reads all observations
14    if( rank == 0 )
15    {
16      l.binaryRead("../data/l.gdk");
17      t.binaryRead("../data/t.gdk");
18    }
19    // regularly distribute observations to all size processes (MPI Scatterv calls)
20    l.scatterv(0); t.scatterv(0);
21    // now, l and t are filled with a part of the observations
22    // obtain the observation equations for the part of observations on every core
23    Fourierseries f(2000, 1.0);
24    f.getDesign(t, A);
25    // compute the NEQs for that part locally on every process
26    N.isATAof(A);
27    n.plusProductOf( A, l, 't', 'n', 1.0 );
28    // and collect them on rank 0 and sum the NEQs up
29    N.reduce( 0, MPI::SUM );
30    n.reduce( 0, MPI::SUM );
31    // only rank 0 solves the NEQs now (serial)
32    if( rank == 0 )
33    {
34       // compute serial solution of NEQs on rank 0 only and save the result
35       n.solveNEQ(N);
36       N.invCholReduced();
37       n.binarySave("../data/xs.gdk");
38       cout<<" Assembly and solution took "<<timer.toc()<<" secs using "<<size<<" cores"<<endl;
39    }
40    // end MPI
41    MPI::Finalize();
42  }
```

and that the partial NEQs can by combined by addition to derive the NEQs composed from all observations. Thus, a simple approach is to distribute the M observations in preferably equal parts over the N involved cores. Every core can independently assemble the NEQs for approximately M/N observations. As the computation of $\mathbf{N} = \mathbf{A}^T\mathbf{A}$ is an expensive operation, the runtime is significantly reduced, as it is now performed in parallel. All processes set up the NEQs for the whole parameter space, but for a portion of the observations only. The final NEQs are the sum of the local, i.e. portioned NEQs assembled by all processes. Sending all NEQs to a single process and summing them up yields the finally combined NEQs including all observations. The system of NEQs can then be serially solved on a single core using e.g., the fast LAPACK solvers. The most expensive step for a huge number of observations, it is the computations of $\mathbf{N} = \mathbf{A}^T\mathbf{A}$, is now completely performed in parallel.

Mapping of the Concept to a MPI Program The concept described above is mapped into a MPI based C++ program in Listing 3.1. To clarify the basics and to emphasize the general MPI operating mode, the code is commented line by line:

- l. 5: The MPI function `Init` is called. Other MPI function calls are allowed afterwards. In the background the library is initialized, e.g. the ranks are assigned to the running processes.
- l. 9: The integer variable `size` is created on every process. The value assigned is the same on every process. It is the number of processes the MPI program was launched with. It is requested with the MPI function `Get_size`.
- l. 10: The integer variable `rank` is created on every process. The value assigned differs on every process. It is the rank of the processes. It is requested with the MPI function `Get_rank`.
- l. 12: Empty Objects of the class `Matrix` are created on every process/core.

l. 14–18: If-statement: The instructions are only executed by the process which rank equals `0`. Thus, only one process reads the observations and locations from disk. On rank `0`, the matrices $\boldsymbol{\ell}$ and $\mathbf{t}$ are filled with the whole set of observations. On all other processes the matrices remain empty.

l. 20: The member function `Scatterv` of the class `Matrix` is called on every core for the matrices $\boldsymbol{\ell}$ and $\mathbf{t}$. This is an self implemented member function which hides the original MPI calls. The function takes the content of the `this` matrix on rank `0` (argument of the function) and distributes the elements of the vector (in this case implemented for vectors only) in equal parts to the `size` involved processes. Every process gets "number of rows" integer divided by `size` elements. The remainder of the integer division is distributed such that the first elements get one extra element. The distribution itself is then performed with the MPI `Scatterv` function. Afterwards, the received portion of the vector is written into the formally empty matrices. On rank `0` the whole vector is overwritten with the portion which goes to rank `0`. After the function calls, all processes have their observations in their local vectors $\boldsymbol{\ell}$ and $\mathbf{t}$. $\boldsymbol{\ell}$ and $\mathbf{t}$ now differ on every core. A virtual concatenation of the local matrices would result in the original vectors as read from disk.

l. 23: Every process creates an object of the class `Forierseries` with parameters of the series to be estimated (order and basis frequency).

l. 24: The object of the class is used to derive the design matrix $\mathbf{A}$ locally on every core for the positions $\mathbf{t}$ of the observations $\boldsymbol{\ell}$. $\mathbf{A}$ again differs on every core.

l. 26–27: All processes perform the same operation, i.e. computation of the partial NEQs, but for different data (observations). Afterwards every core has the NEQs (in $\mathbf{N}$ and $\mathbf{n}$) assembled from its local part of the original observations.

l. 29–30: All processes call the self implemented member function `Reduce` of the class `Matrix`. The function calls send all local matrices $\mathbf{N}$ (and the right hand side vector $\mathbf{n}$) to the process with rank `0` (argument of the function) and combines the matrices there with the sum operation. After the function calls, on rank `0`, the local matrices $\mathbf{N}$ and $\mathbf{n}$ are overwritten with the result, i.e. the sum of all local NEQs. On the other ranks, $\mathbf{N}$ and $\mathbf{n}$ remain unchanged and still contain the partial NEQs. Internally the collective MPI function `Reduce` is called by all processes.

l. 32–39: The process with rank `0` contains now the combined NEQs. Thus, only this process solves the NEQs serially calling proper LAPACK functions integrated within the member functions of the class `Matrix`. The solution and covariance matrix is derived and e.g. written to a file.

l. 41: Finally all processes call the MPI function `Finalize`, which ends the MPI library and destroys the MPI specific objects. Afterwards no more calls to the MPI library are allowed.

3.3 Distributed Matrices

Sect. 3.1 and 3.2 gave a basic introduction to standard concepts for massive parallel program development with a special focus on numeric applications. Exemplarily a very simple MPI program was introduced at the end, to illustrate the basic MPI concept and to clarify the cycle of a MPI based parallel program. For the simple parallelization concept introduced there, two main limitations were pointed out. To circumvent this limitations, the flexible concept of distributed matrices is introduced. Within this concept, a matrix is not stored in the memory of a single core, but stored distributed over the joint local main memory of all cores of the cluster involved. The benefit is that matrices can get extremely huge, as the joint memory of all cores is available. The disadvantage is that computations with these matrices get more and more challenging, as communication between the cores is required to share the distributed stored matrix elements if required on another core.

Fortunately, libraries like BLAS and LAPACK exist for distributed matrices as well, which provide the computational functionalities of those, but for matrices stored distributed.

Although nowadays there are shared memory computers with a large amount of main memory (e.g. 512 GB) with several compute cores, at some point it becomes unreasonable to perform computations on such large matrices with only a few cores (typically these nodes have 32–64 cores only). In addition, many of the widely used BLAS and LAPACK implementations only use 32 bit integer numbers for the array indexing, such that computations with this libraries are limited to matrices with less then 2^{31} entries (i.e. e.g. a $46\,340 \times 46\,340$ matrix, spherical harmonic NEQs of degree and order 215). For a higher general flexibility, it makes sense to switch to distributed stored matrices to use the memory of all cores to store the matrices as well as the computing power of all cores in the cluster to perform operations on/with this matrices. Within this chapter, some terms of distributed computing are defined and a (quasi) standard concept in scientific computing for the handling of distributed matrices is introduced. This concept is mapped into an object oriented class `DistributedMatrix` with an easy to use flexible interface.

3.3.1 Compute Core Grid for Distributed Matrices

Within Sect. 3.2 the concept of MPI topologies and Cartesian compute grids was introduced. In parallel HPC an alternative library is used to provide comparable features (required for later used libraries as well). This library, the Basic Linear Algebra Communication Subprograms (BLACS, Dongarra and Whaley, 1997) is an extension to MPI (MPI is still required as basis). As a main feature, the library always organizes the compute cores as a two-dimensional grid of cores (processes) and assigns two-dimensional coordinates (in addition to the rank). These coordinates (r, c) are used to address the processes (similar to the MPI Cartesian topologies) of the compute core grid (or process grid). As an additional functionality, MPI like functions are provided to directly communicate two-dimensional fields with point-to-point or collective communication. The collective communications are designed to communicate along the whole compute core grid or along specific scopes of that, like columns or rows of that Cartesian grid. As communications can still be organized with MPI only, the main external feature of the library is the setup of the compute core grid. However, additional libraries, which are used later on, use the BLACS routines for internal communication. Using the BLACS library, the program is again started with N processes on N cores using the standard MPI startup. These N cores of the cluster are organized per default as a two-dimensional $\mathsf{R} \times \mathsf{C} = \mathsf{N}$ compute core grid whose dimension is specified by the user/programmer. Fig. 3.2 shows the setup of the two-dimensional compute core grid and the symbols as used in this thesis. The two-dimensional process coordinates can be uniquely converted to a MPI rank. Within the following it is always assumed that the compute grid is organized as a two-dimensional grid. Nevertheless note that one-dimensional grids, i.e. grids with $\mathsf{R} = 1$ or $\mathsf{C} = 1$, are possible without limitations as they are special cases of a two-dimensional grid.

3.3.2 Standard Concept for the Handling of Distributed Matrices in HPC

3.3.2.1 Block-cyclic Distribution of Matrices

The concept for the distribution and the steps involved are summarized in the following, the explanation is supported by the example illustrated in Fig. 3.3. A (quasi-) standard concept for the distribution of matrices over a Cartesian compute core grid of dimension $\mathsf{R} \times \mathsf{C}$ (cf. Fig. 3.3(a)) is the so called two-dimensional block-cyclic distribution (Sidani and Harrod 1996, Blackford et al. 1997, Chap. 4 or Rauber and Rünger 2013, Sect. 2.5). Given a general dense matrix $\mathbf{A}$ of dimension $R \times C$ (cf. Fig. 3.3(b)), the whole matrix (often called *global matrix* or *global view* on the matrix)

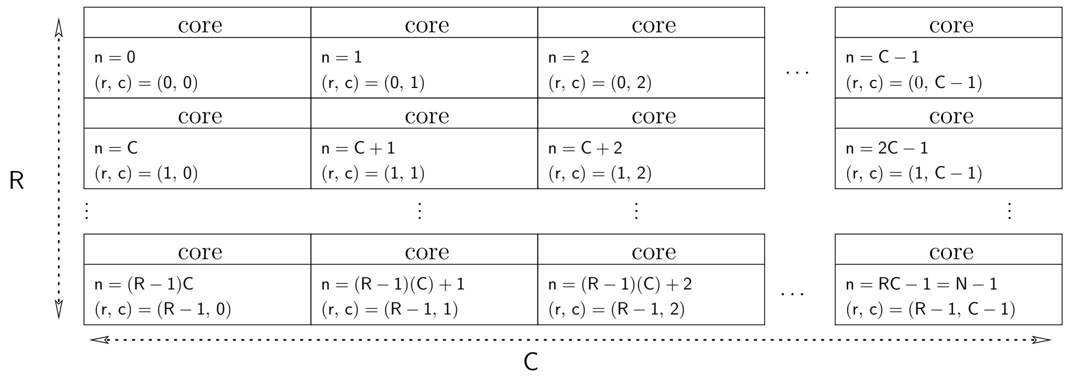


Figure 3.2: Cores of a compute cluster virtually arranged as two-dimensional compute core grid as done by BLACS.

is in a first step divided into blocks $\mathbf{A}_{i,j}$ of an arbitrary block size $b_r \times b_c$ (cf. Fig. 3.3(c)) which is defined by the programmer (or later on by the user),

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,J-1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,J-1} \\ \vdots & \vdots & \cdots & \vdots \\ \mathbf{A}_{I-1,0} & \mathbf{A}_{I-1,1} & \cdots & \mathbf{A}_{I-1,J-1} \end{bmatrix} . \tag{3.1}$$

The number of blocks J along a row and I along a column are

$$I = \left\lceil \frac{R}{b_r} \right\rceil \text{ and } J = \left\lceil \frac{C}{b_c} \right\rceil . \tag{3.2}$$

Except the matrix blocks $\mathbf{A}_{I-1,*}$ and $\mathbf{A}_{*,J-1}$, all blocks are of dimension $b_r \times b_c$. $\mathbf{A}_{I-1,*}$ and $\mathbf{A}_{*,J-1}$ might be of a smaller dimension as they are the rest blocks. Their dimension is related to the remainder, i.e. $R\%b_r$ and $C\%b_c$.

These blocks are now cyclically distributed along the Cartesian compute core grid (cf. Fig. 3.3(d)). The blocks of the first row ($i = 0$) are distributed cyclically to the processors of the first row of the compute core grid $(0, c)$. Block $\mathbf{A}_{0,j}$ is stored on processor with coordinates $(0, j\%C)$. The second row of blocks ($i = 1$) is distributed cyclically along the second row of the compute core grid ($\mathsf{r} = 1$). The general rule, to compute the process coordinates where a block is stored on, can be written as

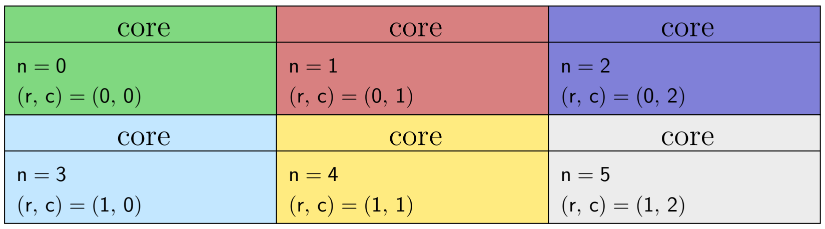
$$\mathbf{A}_{i,j} \mapsto (i\%\mathsf{R}, j\%\mathsf{C}) . \tag{3.3}$$

If $I > \mathsf{R}$ (it is the typical case), the i-th row ($i = \mathsf{R}$) will be distributed again over the first row of the processors grid (cyclic). For the special case $I = \mathsf{R}$ and $J = \mathsf{C}$ a standard block-distribution without the cyclic repetition is achieved.

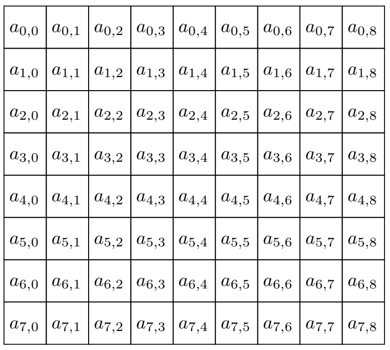
All matrix blocks mapped to a core (r, c) are arranged in a serially stored *local matrix* (cf. Fig. 3.3(e)) on the process via

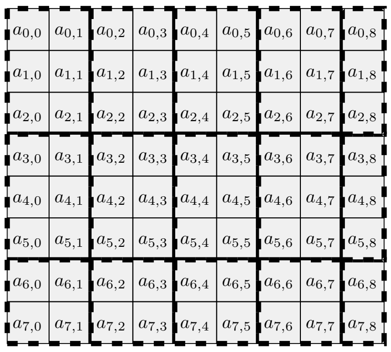
$$\mathbf{A}^{l}_{\mathsf{r},\mathsf{c}} = \begin{bmatrix} \mathbf{A}_{i,j} & \mathbf{A}_{i,j+\mathsf{C}} & \cdots \\ \mathbf{A}_{i+\mathsf{R},j} & \mathbf{A}_{i+\mathsf{R},j+\mathsf{C}} & \cdots \\ \vdots & \vdots & \cdots \end{bmatrix} . \tag{3.4}$$

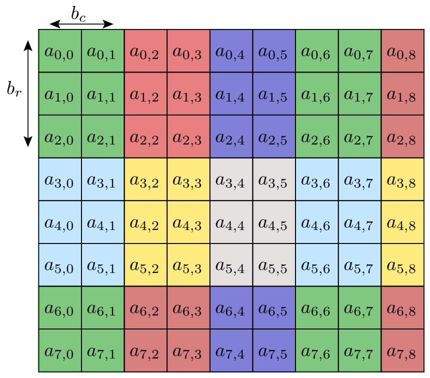
This composed local matrix is stored as a serial matrix in the main memory associated with the process (r, c). It is called *local matrix*. The serial local matrix is then again stored in a one-dimensional field as introduced in Sect. 2.2.1 to map the matrix to the linear memory (cf. Fig. 3.3(f)).



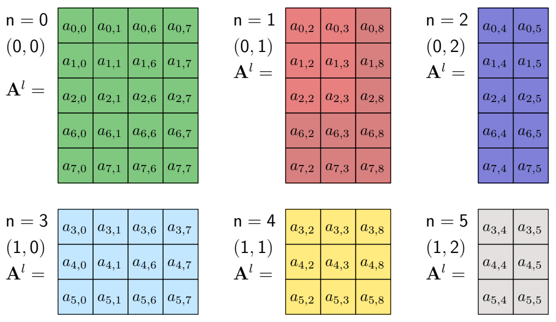
(a) 2×3 compute core grid of N cores the matrix should be distributed to.


(b) Example matrix $\mathbf{A}$ of dimension 8×9 to be distributed.

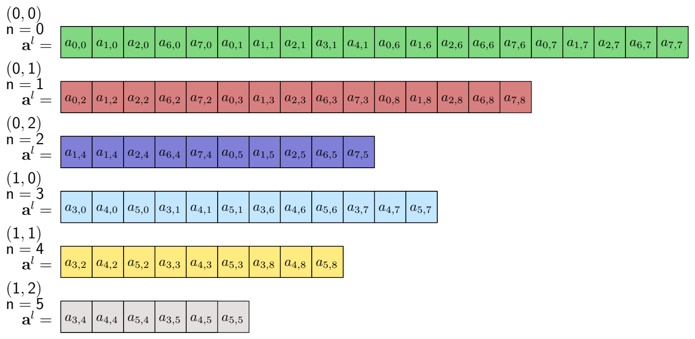

(c) Example matrix partitioned into blocks of size $b_r \times b_c = 3 \times 2$.



(d) Assignment of the 3×2 blocks to the cores.



(e) Local matrices $\mathbf{A}^l_{\mathsf{r,c}}$ as stored on the individual cores.



(f) One-dimensional fields $\mathbf{a}^l_{\mathsf{r,c}}$ the local matrices $\mathbf{A}^l_{\mathsf{r,c}}$ are mapped to on the individual cores.

Figure 3.3: Block-cyclic distribution of a 8×9 matrix on a 2×3 compute core grid using the distribution parameters $b_r \times b_c = 3 \times 2$.

Now, instead of operating on the global matrix, local operations have to be performed on $\mathbf{A}^l_{\mathsf{r,c}}$, containing elements of the global matrix but not necessarily neighboring with respect to the global view on the matrix. It is important to realize that neighboring elements of $\mathbf{A}$ occur only as neighboring elements in the local matrices $\mathbf{A}^l_{\mathsf{r,c}}$ within the sub-blocks of size $b_r \times b_c$. Note that via the choice of the processor grid dimension and the dimension of the sub-blocks $b_r \times b_c$ nearly every distribution of a matrix can be achieved. One-dimensional distributions, one-dimensional cyclic-distributions or block-distributions (without the cyclic recurrence) are possible as special cases of this general scheme, see Blackford et al. (1997, Chap. 4.3) or Rauber and Rünger (2013, Sect. 3.5).

Finally, the concept of block-cyclic distribution of matrices is demonstrated by a small example matrix, distributing a 8×9 matrix along a $\mathsf{R} \times \mathsf{C} = 2 \times 3$ compute core grid using the distribution specific parameters $b_r \times b_c = 3 \times 2$. The example, including the resulting local matrices on the different cores, is illustrated in Fig. 3.3.

3.3.2.2 Index and Element Computations for Block-cyclic Distributed Matrices

Given a matrix $\mathbf{A}$ of dimension $R \times C$, the distributions parameters b_r and b_c and the dimension of the compute core grid $\mathsf{R} \times \mathsf{C}$ the block-cyclic distribution is unique. This section is used to introduce computation formulas which provide the connection of global entries of $\mathbf{A}$ and local entries in $\mathbf{A}^l_{\mathsf{r,c}}$.

Dimension of the Local Matrices With the known distribution parameters, the dimension $R^l_{\mathsf{r,c}} \times C^l_{\mathsf{r,c}}$ of the local matrices $\mathbf{A}^l_{\mathsf{r,c}}$ can be directly computed for every core. For the local matrix on process (r,c) it is

$$R^l_{\mathsf{r,c}} = ((R \div b_r) \div \mathsf{R})\, b_r + (\mathsf{r} < ((R \div b_r)\,\%\mathsf{R}))\, b_r + (\mathsf{r} == ((R \div b_r)\,\%\mathsf{R}))\,(R\%b_r)\,, \tag{3.5a}$$

$$C^l_{\mathsf{r,c}} = ((C \div b_c) \div \mathsf{C})\, b_c + (\mathsf{c} < ((C \div b_c)\,\%\mathsf{C}))\, b_c + (\mathsf{c} == ((C \div b_c)\,\%\mathsf{C}))\,(C\%b_c)\,, \tag{3.5b}$$

where the meaning of the involved operands (done for the rows only, analog for columns) is as follows:

- $R \div b_r$: Is the number of *full* blocks, the global matrix $\mathbf{A}$ is partitioned along a column.
- $((R \div b_r) \div \mathsf{R})$: Is the number of *full* blocks each core of the grid's column gets for sure (minimum per core).
- $((R \div b_r) \div \mathsf{R})\, b_r$: Is the number of entries, resulting from the *full* blocks, each core of the grid's column gets for sure (above bullet).
- $(R \div b_r)\,\%\mathsf{R}$: Number of remaining full blocks, which have to be additionally distributed over the first cores of the grid's column.
- $(\mathsf{r} < ((R \div b_r)\,\%\mathsf{R}))$: Is 1 if process belongs to the first "remaining full blocks" processes of the grid's column and thus gets an additional full block. It is 0 otherwise.
- $(\mathsf{r} < ((R \div b_r)\,\%\mathsf{R}))\, b_r$: Number of additional entries for the first "remaining full blocks" processes of the column of the grid. It is b_r if process belongs to the first ones and 0 otherwise.
- $(R\%b_r)$: Number of rest entries of the matrix, not distributed as full blocks.
- $(\mathsf{r} == ((R \div b_r)\,\%\mathsf{R}))$: Is 1 if the processor is the followup process of the processor which got the last additional full block, 0 (i.e. no additional elements) otherwise.

Global Matrix Indices from Local Indices Given the dimension of the compute core grid, the dimension of the global matrix $\mathbf{A}$ and the distribution parameters, the indices of a local entry $(r^l_{\mathsf{r,c}}, c^l_{\mathsf{r,c}})$ on process (r,c) can be converted to the index of that entry in the global matrix (r,c) , i.e. $\mathbf{A}(r,c) == \mathbf{A}^l_{\mathsf{r,c}}(r^l_{\mathsf{r,c}}, c^l_{\mathsf{r,c}})$. The global indices of the matrix entry are computed from the local ones via

$$r = \left(r^l_{\mathsf{r,c}} \div b_r\right) \mathsf{R} b_r + \mathsf{r} b_r + r^l_{\mathsf{r,c}}\%b_r, \tag{3.6a}$$

$$c = \left(c^l_{\mathsf{r,c}} \div b_c\right) \mathsf{C} b_c + \mathsf{c} b_c + c^l_{\mathsf{r,c}}\%b_c, \tag{3.6b}$$

where the involved operands can be explained as (done for the rows only, analog for columns) to clarify the equation:

- $\left(r^l_{\mathsf{r,c}} \div b_r\right)$: Number of full blocks each core has at least in its local matrix before the block, the entry in row $r^l_{\mathsf{r,c}}$ belongs to.
- $\left(r^l_{\mathsf{r,c}} \div b_r\right) \mathsf{R}$: Number of full blocks in the global matrix which are at least before the global block the local entry $r^l_{\mathsf{r,c}}$ belongs to.

- $\left(r^l_{\mathsf{r,c}} \div b_r\right) \mathsf{R} b_r$: Number of global entries, resulting from full blocks in the global matrix which are at least before the block the local entry $r^l_{\mathsf{r,c}}$ belongs to.
- $\mathsf{r} b_r$: Number of entries resulting from additional full blocks on the processes which are before r in the grid.
- $r^l_{\mathsf{r,c}} \% b_r$: Rest entries, which do not correspond to a full block, before the entry.

Local Indices and Process Coordinates from a Global Index of an Entry Given an entry (r, c) of the global matrix, the following set of formulas provides the computation of the process coordinates and the entry in the local matrix the element is stored in. Knowing the distribution parameters and the compute core grid dimension, the process coordinates can be computed with

$$\mathsf{r} = (r \div b_r) \% \mathsf{R}, \tag{3.7a}$$
$$\mathsf{c} = (c \div b_c) \% \mathsf{C}, \tag{3.7b}$$

where the involved operands are

- $(r \div b_r)$: Index of full block from the global matrix the entry belongs to.
- $(r \div b_r) \% \mathsf{R}$: Process row coordinate that block is stored on.

After the process coordinates are determined, the entries position in that local matrix can be computed via

$$r^l_{\mathsf{r,c}} = ((r \div b_r) \div \mathsf{R})\, b_r + (\mathsf{r} < ((r \div b_r) \% \mathsf{R}))\, b_r + (\mathsf{r} == ((r \div b_r) \% \mathsf{R}))\, (r \% b_r), \tag{3.8a}$$
$$c^l_{\mathsf{r,c}} = ((c \div b_c) \div \mathsf{C})\, b_c + (\mathsf{c} < ((c \div b_c) \% \mathsf{C}))\, b_c + (\mathsf{c} == ((c \div b_c) \% \mathsf{C}))\, (c \% b_c). \tag{3.8b}$$

where the involved operands are

- $r \div b_r$: Is the number of *full* blocks, the global matrix $\mathbf{A}$ is partitioned into up to entry r (along a column).
- $((r \div b_r) \div \mathsf{R})$: Is the number of *full* blocks each core of the grid's column has for sure (minimum per core) before the global entry r can occur.
- $((r \div b_r) \div \mathsf{R})\, b_r$: Number of entries, resulting from the *full* blocks, each core of the grid's column has for sure before the global entry r can occur.
- $(r \div b_r) \% \mathsf{R}$: Number of additional remaining full blocks in the global matrix before the block r belongs into.
- $(\mathsf{r} < ((r \div b_r) \% \mathsf{R}))$: Is 1 if process belongs to the first "additional remaining full blocks" processes of the grids column and thus the process gets has an additional full block stored. It is 0 otherwise.
- $(\mathsf{r} < ((r \div b_r) \% \mathsf{R}))\, b_r$: Number of additional entries for the first "additional remaining full blocks" processes of the grids column. It is b_r if process belongs to the first ones and 0 otherwise.
- $(r \% b_r)$: Number of rest entries of the matrix before row r, not distributed as full blocks.
- $(\mathsf{r} == ((r \div b_r) \% \mathsf{R}))$: Is 1 if the processor is the followup process of the processor who got the last additional full block, 0 (i.e. no additional elements) otherwise.

3.3.3 Standard Libraries for Computations with Block-cyclic Distributed Matrices

Standard HPC libraries exist to perform basic matrix computations and standard linear algebra operations with block-cyclic distributed matrices. Similar to BLAS and LAPACK (cf. Sect. 2.3), the Parallel Basic Linear Algebra Subprograms (PBLAS, Blackford et al., 1997, Choi et al., 1995b) containing basic matrix computations for block-cyclic distributed matrices exist. In general the PBLAS library provides the same functionality as the serial BLAS library, organized in the three levels as well. To provide the functionality, PBLAS routines extensively use BLACS and thus MPI based communication to share the data (i.e. matrix entries) between the processes if needed during the computations. The actual computations are serially performed on the cores extensively using the serial BLAS library. Linking optimized BLAS routines for the used hardware then provides optimized PBLAS routines as well. Instead of LAPACK, the SCAlable Linear Algebra Package (ScaLAPACK, Blackford et al., 1997, Choi et al., 1992, 1995a) provides the LAPACK functionality operating on block-cyclic distributed matrices. Again, message passing is organized by BLACS, basis computations are performed with PBLAS and the serial computations are performed with the serial BLAS and LAPACK functions.

Both libraries are mainly Fortran implementations (with some C extensions) which are available as open source libraries in NETLIB/SCALAPACK (2012). As used later on for the implementation of block-cyclic distributed matrices as a class, the description and references to a block-cyclic distributed matrix as used by the libraries are shortly summarized here. Note that, as mainly Fortran is used, PBLAS and SCALAPACK use an array in indexing starting with 1, in contrast to 0 as used as a standard here (C++ like). As the PBLAS and SCALAPACK functions use collective MPI (BLACS) communication, it has to be taken care of, that all processes of the compute core grid (or at least the correct subset) call the PBLAS or SCALAPACK function.

The interface to the PBLAS (and or SCALAPACK) routine for a block-cyclic distributed matrix can be grouped into four parts:

1. Description of the Distribution of a Matrix: The description of the block-cyclic distribution of a matrix is passed to the PBLAS (or SCALAPACK) as an integer field of length nine (cf. Blackford et al., 1997, Chap. 4.2, 4.3.3). Within the context of distributed HPC and these libraries this nine element integer field is called *array descriptor* **d**. It contains the major description of the matrix and its block-cyclic distribution. This array descriptor is stored on every process of the compute core grid the matrix is distributed to. The entries of the descriptor at the positions are for in-core dense matrices[3]:

0. An integer characterizing the type of the matrix. Within PBLAS and SCALAPACK the following types of matrices are known: in-core dense matrix (value 1) used here only (narrow band and tridiagonal matrices (value 501), narrow band and tridiagonal right-hand-side matrices (value 502) and out-of-core dense matrices (value 601)). This entry is constant for all processes of the compute core grid (equal to 1) in this thesis.
1. Contains the BLACS context the PBLAS and SCALAPACK routines communicate in. A BLACS context is comparable to a MPI communicator. In BLACS it is an integer number. This parameter is the same on every process, it is the number returned by the BLACS initialization function. Typically zero if only one BLACS communicator is used.

[3]Note that the meaning changes for other matrix types, i.e. narrow band and tridiagonal matrices , narrow band and tridiagonal right-hand-side matrices and out-of-core dense matrices. As only in-core dense matrices are used here, the meaning is only explained for that matrix type.

2. Number of rows of the whole i.e. global matrix, called R within this thesis. The entry is the same on every process.
3. Number of columns of the whole i.e. global matrix, called C within this thesis. The entry is the same on every process.
4. Dimension of the sub-blocks in row direction the matrix is partitioned in before distribution, called b_r in this thesis. The entry is the same on every process.
5. Dimension of the sub-blocks in column direction the matrix is partitioned in before distribution, called b_c in this thesis. The entry is the same on every process.
6. Row coordinate of the process where the block-cyclic distribution starts. I.e. the row coordinate of the process the first block of the global matrix is distributed to. Here and in the examples given above it is $\mathsf{r} = \mathsf{0}$. For a better balancing, especially handling many small matrices, alternatives would be possible. Nevertheless, within this thesis the entry is set to $\mathsf{0}$ as default. The entry is the same on every process.
7. Column coordinate of the process where the block-cyclic distribution starts. I.e. the column coordinate of the process the first block of the global matrix is distributed to. Here and in the examples given above it is $\mathsf{r} = \mathsf{0}$. For a better balancing, especially handling many small matrices, alternatives would be possible. Nevertheless, within this thesis the entry is set to $\mathsf{0}$ as default. The entry is the same on every process.
8. Leading dimension of the (serial) local matrix $\mathbf{A}^l_{(\mathsf{r},\mathsf{c})}$ on the process with coordinates (r,c). Using, as in this thesis, column-major order (cf. Sect. 2.2) it is the number of rows of $\mathbf{A}^l_{(\mathsf{r},\mathsf{c})}$, $R^l_{(\mathsf{r},\mathsf{c})}$. This entry in the array descriptor differs for processes of different processor grids rows.

With the array descriptor provided, and knowing the compute core grid dimension (retrievable with BLACS routines) the matrix distribution is uniquely defined. Within every PBLAS or SCALAPACK function call a block-cyclic distributed matrix is passed to, a reference to this array descriptor of the matrix is passed to as well.

2. Accessing the Matrix Data The local data of the matrices are referenced by pointers to the first element of the local matrices $\mathbf{A}^l_{(\mathsf{r},\mathsf{c})}$, in C++ it is $\&\mathbf{A}^l_{(\mathsf{r},\mathsf{c})}(0,0)$. As every process of the compute core grid calls the PBLAS/SCALAPACK function, it is another pointer on every core, pointing to the local matrix stored on that core.

3. Operating on Sub-Matrices SCALAPACK as well as PBLAS provide the functionality of performing computations on sub-matrices. For that reason, the global row and column coordinate of the first entry where the sub-matrix starts can be provided to the functions (typically called $i*$ and $j*$ in the function description, +1 as this entries start counting with 1). In addition, the dimension of the sub-matrix must be provided, if sub-matrices are used. It differs from the dimension contained in the array descriptor.

4. Operations on Matrices Typically, via character arguments, operations can be performed on the matrix, before the actual computation is performed by the function. These operations can be for instance a transpose operation. It tells the routine e.g., if either $\mathbf{AB}$ or $\mathbf{A}^T\mathbf{B}$ should be computed. These parameters are very function specific and are explained in detailed function descriptions (e.g. in Blackford et al., 1997, Part II). Note that these parameters typically have a consequence on the other function arguments like the dimension of the (sub-)matrices. See again the function descriptions for details.

With this basic knowledge, PBLAS and SCALAPACK routines can be used, if the matrices are available in the introduced block-cyclic distribution.

3.3.4 Implementation as a C++ Class

The introduced block-cyclic distribution of matrices was implemented as a C++ class which is the basis for the applications introduced later (cf. Chap. 6–8) and for many other applications and projects which require massive parallel HPC in the group of Theoretical Geodesy at the IGG Bonn. All in all, a class `DistributedMatrix` is implemented consisting of about 5000 lines of C++ code. The main features of the developed class are

- the management of the block-cyclic distribution,
- an interface to manipulate entries in the block-cyclic distributed matrices,
- the mapping between local and global entries,
- a simplified interface to PBLAS and SCALAPACK computing routines,
- a parallel I/O of matrices from block-cyclic distribution to files or from files directly to block-cyclic distributed matrices,
- a distribution of a serial stored matrix to a block-cyclic distributed matrix,
- or vice versa the collection of a block-cyclic distributed matrix on a single process into a serial stored matrix,
- and an implementation of row and or column reordering (i.e. sequences of row and column interchanges).

An overview about the implemented class `DistributedMatrix` is given in a simplified excerpt from the header file which is provided in Listing 3.2. Constructing an object of the class `DistributedMatrix`, an object of the class is created within every process of the compute core grid. The local attributes (mainly the local serial Matrix $\mathbf{A}^{l}_{(\mathsf{r},\mathsf{c})}$ and the array descriptor $\mathbf{d}$) of the class are created in the main memory of every core. The class provides a function interface to fill the local matrices with data and to perform computations or operations on the local matrix.

3.3.4.1 Distributed Matrix Reading with MPI

As it is a flexible feature of the class, the concept for parallel distributed I/O operations directly into block-cyclic distributed matrices (and vice versa from) block-cyclic distributed matrices is introduced here. Although some technical details are omitted, the MPI basic routines used are given and the idea behind that concept is introduced. The process is described for file reading. For file writing the process is comparable, using corresponding MPI write functions instead of the read functions. The same binary file format as for serial matrices is used for both block-cyclic distributed matrices and serial matrices (cf. Sect. 2.2.2).

MPI provides I/O routines for opening files for distributed reading and writing (Gropp et al., 1999b, Sect. 3.2). Each process of the grid opens the MPI file (`MPI::File::Open`). Every process reads – starting at the begin of the file – two integer numbers (`MPI::File::Read`), i.e. the global dimension ($R \times C$) of the matrix contained in the file (or in an advanced format the complete header of known size). Afterwards the resize function of the `DistributedMatrix` class is called, such that the local memory for the local matrices is allocated and an empty `DistributedMatrix` of the read dimension is created. The position in the file is on all processes the start position of the RC double data representing the matrix entries.

To read the data in parallel directly into the local memory of the distributed matrices, advanced MPI derived data types for block-cyclic distributed arrays can be used to represent block-cyclic distributed matrices (`MPI::Datatype::Create_darray`). The distributed array data type depends on the compute core grid dimension and the parameters b_r and b_c of the block-cyclic distribution.

Listing 3.2: Simplified header file defining the main features of the class `DistributedMatrix`.

```cpp
#ifndef DistributedMatrix_H
#define DistributedMatrix_H
class PARALLELDIAGONALMATRIX; class PARALLELBLOCKDIAGONALMATRIX; class PARALLELSPARSEMATRIX;
using namespace std;

class DistributedMatrix
{
  public:
    enum MATRIX_TYPE { GENERAL, SYMMETRIC_L, SYMMETRIC_U, TRIANGULAR_L, TRIANGULAR_U };
    // constructors
    DistributedMatrix();
    DistributedMatrix( int blacs_context, int R, int C, int br, int bc, int startRow, int startCol );
    DistributedMatrix( const DistributedMatrix &M );
    DistributedMatrix & operator=( const DistributedMatrix & C );
    // destructor
    ~DistributedMatrix();
    // access distribution and dimension parameters of the matrix
    size_t Rl() const;
    size_t Cl() const;
    size_t R() const;
    size_t C() const;
    int br() const;
    int bc() const;
    int context() const { return( _arrayDescriptor.at(1) ); };
    int* d(); // Pointer to array descriptor
    const int* d() const; // Pointer to array descriptor
    MATRIX_TYPE type( ) const { return(_type) ;}
    void setType( MATRIX_TYPE t );
    // manipulate size/distribution of matrix
    void setParameters( int blacs_context, int R, int C, int br, int bc, int startRow, int startCol );
    void resize( int R, int C ); // change dimension of global matrix
    void expand( size_t R, size_t C ); // add columns to global matrix to dimension R x C
    void trim( size_t R, size_t C ); // reduce dimension to R x C
    // data access
    Matrix & localMat(){ return( _A ); };
    Matrix localMat() const { return( _A ); };
    double & operator()( int r, int c); // access local entries, write
    double operator()( int r, int c) const; // access local entries, read
    // pointer to the first element of local serial stored matrix
    inline double* feld(){ return( _A.feld() ); };
    inline double* feld() const { return( _A.feld() ); };
    inline double* colPtr( size_t c );
    // index/entry position computations between local and global matrix
    int colInGlobalMat( int cl ) const;
    int rowInGlobalMat( int rl ) const;
    void posInLocalMat( int r, int c, int & rl, int & cl, int & rowProcIdx, int & colProcIdx );
    // collect a distributed matrix, distribute a serial matrix
    void isDistributed( Matrix & A, int fromRank=0, int offset = 0 );
    void collectOnRank( int onRank, Matrix &Ajoint );
    // computing routines manly referencing SCALAPCK PBLAS functions
    // different multiplications including special cases like symmetric/triangular matrices
    void plusAtAof( DistributedMatrix & A, double w = 1.0, char tran = 't', int ia=1, int ja=1 );
    void plusProductOf(DistributedMatrix &A,char tranA,DistributedMatrix &B,char tranB,double w,int ic=1,int jc=1);
    void plusProductOf( DistributedMatrix & A, PARALLELDIAGONALMATRIX & D );
    void plusProductOf( PARALLELDIAGONALMATRIX & D, DistributedMatrix & A );
    void isProductofSymUnsym( DistributedMatrix & symmA, DistributedMatrix & B, double w );
    void plusProductofSymUnsym( DistributedMatrix & symmA, DistributedMatrix & B,  double w );
    ...
    DistributedMatrix & operator*=( double scal );
    DistributedMatrix & operator-=( DistributedMatrix & C );
    DistributedMatrix & operator+=( DistributedMatrix & C );
    DistributedMatrix & operator+=( PARALLELDIAGONALMATRIX & C );
    DistributedMatrix operator-( DistributedMatrix & C ) const;
    DistributedMatrix operator+( DistributedMatrix & C ) const;
    DistributedMatrix operator+( PARALLELDIAGONALMATRIX & C ) const;
    ...
    // solve and inversion
    void isSolutionOf( DistributedMatrix & N );
    void isSolutionOfCholReduced( DistributedMatrix & N );
    void isSolutionOfTriag( const DistributedMatrix & R, char side='L', char tranA='N', int ib=1, int jb=1 );
    void isSolutionOfTriag( const PARALLELBLOCKDIAGONALMATRIX & R, char side='L', char tranA='N' );
    void chol();
    void isInvOfCholReduced( );
    // special computations
    void eigenvalues( Matrix & ev );
    void eigenvalues( Matrix & ev, DistributedMatrix & Z);
    double trace( );
    // parallel I/O
    void binarySave( string filename, int mode = 1 );
    void binaryRead( string filename, int mode = 1 );
    // reordering with index vector
    void reorder( vector<size_t> p, bool perm=true ); // this -> this(idx,idx)
    void reorderCols( vector<size_t> p, bool perm=true ); // this -> this(:,idx)
    void reorderRows( vector<size_t> p, bool perm=true ); // this -> this(idx,:)
  private:
    Matrix _Al;
    MATRIX_TYPE _type;
    vector<int> _arrayDescriptor;
    // management routines
    void _determineLocalMatrixSize( );
};
#endif // DistributedMatrix_H
```

Passing this parameters in a MPI specific form to the function, an internal MPI data type representing the distribution over the grid is created. This data type defines which parts of the block-cyclic distributed matrix are visible for which processes of the compute core grid. This virtual data type is set as a *file view* on the opened file (`MPI::File::Set_view`). This file view puts a virtual mask over the file's data, that afterwards each process only sees the matrix entries which belong to its local matrix. The data of the other processes are faded out in the view of an individual process. Thus, every process has another view on the file. Virtually it is like a file which only contains the processes local data (following the specified block-cyclic distribution). After the file view is set on every core, the local data of the processes are read in parallel via a call of the `MPI::File::Read_all` function on every core. Via pointers, the data can be directly read into the local memory which is the local matrix $\mathbf{A}^{l}_{\mathsf{r},\mathsf{c}}$. For the technical details see Gropp et al. (1999b, Chap. 3.3 and 3.4). A comparable process is implemented for the parallel writing of a matrix from a block-cyclic distributed matrix to a serial file. Mainly the reading functions are changed to writing functions. Of course, the header (e.g. matrix dimension) is only written to the file by a single process. Using the same functions already introduced, even symmetric or triangular matrices stored in a packed format in a binary file can be read in parallel into a block-cyclic distributed matrix (or written from a block-cyclic distributed matrix into a file with packed storage).

To get a feeling of the time spent for the parallel matrix I/O, some numbers are given. A systematic study, is hard to perform as the derived numbers vary a lot. Especially in a HPC environment the runtime measurements are significantly influenced by other activities on the HPC file system as well in the (Infiniband) network. Variations around a factor of at least two are often observed. In addition, they depend on the block-cyclic distribution parameters.

Using a compute cluster of standard nodes via a standard Ethernet (1 Gbit/s) connection and standard server disks mounted as a network file system (NFS), the runtime for reading and writing matrices of dimension $17\,000 \times 17\,000$ (2 GB) to $63\,000 \times 63\,000$ (30 GB) is in the range of 100 s to 1000 s, strongly depending on the disk performance and the network activities. Anyway, matrices of dimension $100\,000 \times 100\,000$ (75 GB) can be successfully read into (written from) the block-cyclic distribution in about 2000 s.

The performance significantly increases when the software component is used in a dedicated HPC environment (JUROPA at FZ Jülich). Both important components, the network connection and the file system are faster. Whereas the nodes are connected via Infiniband (40 Gbit/s), the data is read from a parallel file system designed for parallel HPC (lusture, e.g. Bauke and Mertens, 2006, p. 65). Matrices up to dimension $100\,000 \times 100\,000$ (75 GB) are read and written in $20 - 100$ s, depending on the compute core grid and the block-cyclic distribution. The basic conclusion is that, e.g., reading and writing of a 30 GB matrix can be performed in less then 40 s. Independent of the block-cyclic distribution and the compute core grid, the operation can be performed in $40 - 90$ s. Even matrices of dimension $520\,000 \times 520\,000$ (2 TB) are read in e.g. 2100 s as a block-cyclic distributed matrix (64×64 compute core grid, $b_r = b_c = 64$) and written from a block-cyclic distribution to a file in 3500 s. Note again that after the introduced reading/writing operation, the matrices are directly stored in the specified block-cyclic distribution or written from block-cyclic distribution to a serial file. Additional distribution/collection of the matrix entries are decrepit.

3.3.4.2 Simple Example of an Adjustment Procedure

Listing 3.3 shows a simple adjustment problem (it solves the same problem as in Sect. 3.2.2) using block-cyclic distributed matrices and the massive parallel computations as provided by the member functions of the class (mostly PBLAS and SCALAPACK calls). The simplified interface of block-cyclic distributed matrix handling as developed within this thesis is used.

Listing 3.3: Example of a parallelization of an adjustment problem using the implemented interface to block-cyclic distributed matrices.

```cpp
using namespace std;
int main( int argc, char *argv[] )
{
  int context;
  // Initialize SCALAPCK and in the background BLACS (and MPI)
  int rowsInGrid = atoi(argv[1]);
  int colsInGrid = atoi(argv[2]);
  scalapack_sl_init( context, rowsInGrid, colsInGrid );
  // initialize a timer
  TicToc timer; timer.tic();
  // specify the distribution parameters
  int br = 32; int bc = 32;
  // initialize (empty) block-cyclic distributed matrices
  DistributedMatrix l( context, 0, 0, br, bc, 0, 0 );
  DistributedMatrix t( context, 0, 0, br, bc, 0, 0 );
  DistributedMatrix A( context, 0, 0, br, bc, 0, 0 );
  DistributedMatrix N( context, 0, 0, br, bc, 0, 0 );
  DistributedMatrix n( context, 0, 0, br, bc, 0, 0 );
  // read observations and control points into distributed matrices
  l.binaryRead("../data/l.gdk");
  t.binaryRead("../data/t.gdk");
  // object of example class Fourier series which fills the local parts of the design matrices
  Fourierseries f(50000, 1.0);
  f.getDesign(t, A); // i.e. fill the local parts of design matrix on every core
  // compute the NEQs for that portion locally on every process
  N.plusATAof(A);
  n.plusProductOf( A, 't', l, 'n', 1.0 );
  // solve the system of equations in place
  n.isSolutionOf(N);
  // N is overwritten with chol(N), compute inverse from already Cholesky reduced matrix
  N.isInvOfCholReduced();
  // save solution and VCM
  n.binarySave("../data/xs.gdk");
  N.binarySave("../data/S_xx.gdk");
  int nprow, npcol, myrow, mycol;
  blacs_gridinfo( context, nprow, npcol, myrow, mycol );
  if( (myrow == 0) && (mycol == 0) )
  {
    cout<<" Assembly and solution took "<<timer.toc()<<" secs using "<<nprow<<" x " << npcol << " grid"<<endl;
  }
  // end MPI
  blacs_gridexit( context );
  blacs_exit( 0 );
}
```

The main thing which needs a careful implementation in such an example, using the interface developed in this thesis within adjustment problems, is the setup of matrices in the block-cyclic distribution. E.g. in the example, the setup of the design matrix for the observations is hidden in the class `FourierSeries`. The correct setting of the entries in the local part of the design matrix $\mathbf{A}^{l}_{(\mathsf{r},\mathsf{c})}$ has to be carefully implemented (using the index computations between the local and global view of the matrix cf. Sect. 3.3.2.2). Especially, it has to be guaranteed that the defined parameter order is kept and that the order in the global matrix view is correct. The member functions of the class `colInGlobalMat, rowInGlobalMat, posInLocalMat` help to organize the mapping from global and local rows/columns of the matrix. If that step is done, the use of the class and the rest of the program is straightforward and shows the use of the developed interface without going into the details of the application of block-cyclic distributed matrices and SCALAPACK. Note that the program in Listing 3.3 can be executed on any arbitrary compute core grid. It does not matter if it is of dimension 1×1 or 123×89. The only restriction is, that the matrices used have to fit into the joint main memory of all cores of the compute core grid.

3.3.5 Benefit of the Block-cyclic Distribution

Until now, it was not discussed why particularly the concept of block-cyclic distributed matrices was introduced. Distributing two-dimensional fields over a two-dimensional grid is straightforward but why introducing the *cyclic* distribution of the small *blocks*? Is a simple block distribution of the matrix (without the small sub-blocks which are cyclically distributed) not sufficient? Before going to the applications where the concept of the complicated block-cyclic distribution is used for parallel processing, these questions should be answered.

The main reason, why the cyclic distribution of the small sub-blocks is introduced, is related to an optimized load balancing of all cores of the compute grid. Of course it is clear that there does not exist a distribution which is the most efficient for every numerical algorithm which might be applied to a matrix. The block-cyclic distribution is derived as the most flexible and efficient distribution in Blackford et al. (Sect. 4.3.1, 1997) and it has the advantage, that any other distribution (block-wise, column-wise, row-wise) can be realized as special case, using an appropriate compute core grid (shape) and distribution parameters b_r and b_c. Thus, implementing the general block-cyclic distribution, special cases are included. For specific operations (or applications) where e.g. a block column-wise distribution is well suited for the algorithm applied to the stored matrix, this can be realized with the general implementation by setting the compute core grid to $\mathsf{1} \times \mathsf{N}$ and $b_r = R$ and $b_c = C \div \mathsf{C}$.

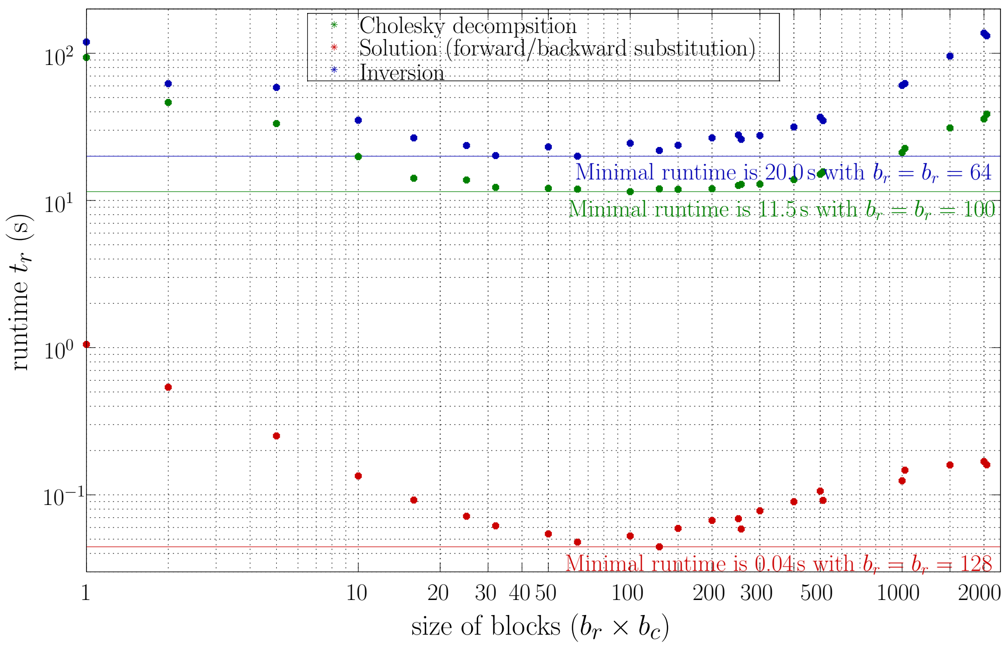
Empirical Proof of Gain of Block-cyclic Distribution: Fig. 3.4 shows the effect of the block-cyclic distribution for the case of the Cholesky factorization of a matrix, the solution of the system of equations (forward and backward substitution) and the computation of the inverse matrix. The empirical runtime study is limited to quadratic sub-blocks, the special case of a block distribution ($b_r = b_c = 2048$) is included. The compute core grid is fixed to a quadratic grid, i.e. $\mathsf{16} \times \mathsf{16}$. A small matrix of dimension $32\,757 \times 32\,757$ is used (spherical harmonic normal equations to degree and order 180) to make the serial computations possible as a reference.

Fig. 3.4 also shows the runtime measurements for the three operations choosing the different dimensions of the sub-blocks. The general characteristics of the three curves representing the example operations are the same. The runtime is very high for very small block dimensions ($b_r = b_c < 20$). For the serial computations on the cores, BLAS routines are used for the computations within PBLAS and SCALAPACK. Choosing the dimension of the sub-blocks too small, the BLAS routines become very inefficient, as they cannot efficiently take advantage of the cache memory. Besides the BLAS efficiency, the results are poor, if the number of blocks the whole matrix is partitioned into is very large. Then, the organizational cost for communication and block access becomes larger within the SCALAPACK and PBLAS routines. The minimal runtime is found between block sizes of $b_r = b_c = 32$ and $b_r = b_c = 256$, it is around the suggested default value of 64 suggested by Blackford et al. (1997, p. 92). Having block sizes within this range, the load balancing of the cores is good and the block-size is well suited for the BLAS optimization of the memory access using the different levels of the cache. Choosing the optimal block-size, there is an additional runtime reduction of about a factor of 6–8. Using larger blocks, the runtime again increases. At some point, the matrix is not block-cyclic distributed but only block distributed. The load balancing of the involved cores is then very bad. It is discussed for the Cholesky factorization in the following.

Illustration of the Gain of the Block-cyclic Distribution: Instead of empirical runtime measurements, the gain of the block-cyclic distribution can be theoretically shown in an example. The Cholesky decomposition of a positive definite matrix $\mathbf{N} = \mathbf{R}^T\mathbf{R}$ can be written as an algorithm operating on blocks of the matrix for a block $\mathbf{R}_{ij}$ as (e.g. Golub and van Loan 1996, Sect. 4.2, Schuh 2001, Sect. 2.4.1):

$$\mathbf{R}_{ii} = \text{chol}\left(\mathbf{N}_{ii} - \sum_{k=0}^{(i-1)} \mathbf{R}_{ki}^T \mathbf{R}_{ki}\right), \text{ for } i = 1..I-1 \tag{3.9}$$

$$\mathbf{R}_{ij} = \mathbf{R}_{ii}^{-T}\left(\mathbf{N}_{ij} - \sum_{k=0}^{(i-1)} \mathbf{R}_{ki}^T \mathbf{R}_{kj}\right), \text{ for } i = 1..I-1,\ j = i+1..J-1 \tag{3.10}$$



(a) Graphical depiction of runtime (t_r) measurements.

operation	serial	minimum		maximum			ratios	
	t_r (s)	b_r, b_c	t_r (s)	b_r, b_c	t_r (s)	max/min	serial/max	serial/min
Cholesky	1083.08	100	11.47	1	93.68	8.16	11.56	94.43
Solution	1.15	128	0.04	1	1.05	23.71	1.10	25.92
Inversion	2342.33	64	19.98	2000	137.08	6.86	17.08	117.23
Total	3426.56	64	31.96	1	213.94	6.69	16.02	107.21

(b) Numerical values for extrema.

Figure 3.4: Runtime for the Cholesky decomposition, the solution by forward and backward substitution and the inversion depending on the choice of the sub-block dimension $b_r = b_c$.

For a partitioning into $I \times J = 3 \times 3$ blocks and block distribution to 3×3 compute core grid, for instance the process $0,0$ gets the matrix block $\mathbf{N}_{00}$ only. The local matrix of process $0,0$ for the block distribution is

$$\mathbf{N}^l_{0,0} = \left[\mathbf{N}_{00}\right]. \tag{3.11}$$

The block Cholesky factorization is computable as follows,

$$\begin{aligned}
\text{on } 0,0: \mathbf{N}_{00} &= \mathbf{R}_{00}^T\mathbf{R}_{00} \Rightarrow \mathbf{R}_{00} = \text{chol}(\mathbf{N}_{00}) && (3.12a)\\
\text{on } 0,1: \mathbf{N}_{01} &= \mathbf{R}_{00}^T\mathbf{R}_{01} \Rightarrow \mathbf{R}_{01} = \mathbf{R}_{00}^{-T}\mathbf{N}_{01} && (3.12b)\\
\text{on } 0,2: \mathbf{N}_{02} &= \mathbf{R}_{00}^T\mathbf{R}_{02} \Rightarrow \mathbf{R}_{02} = \mathbf{R}_{00}^{-T}\mathbf{N}_{02} && (3.12c)\\
\text{on } 1,1: \quad \mathbf{N}_{11} &= \mathbf{R}_{01}^T\mathbf{R}_{01} + \mathbf{R}_{11}^T\mathbf{R}_{11} \Rightarrow \mathbf{R}_{11} = \text{chol}\left(\mathbf{N}_{11} - \mathbf{R}_{01}^T\mathbf{R}_{01}\right) && (3.12d)\\
\text{on } 1,2: \quad \mathbf{N}_{12} &= \mathbf{R}_{01}^T\mathbf{R}_{02} + \mathbf{R}_{11}^T\mathbf{R}_{12} \Rightarrow \mathbf{R}_{12} = \mathbf{R}_{11}^{-T}\left(\mathbf{N}_{12} - \mathbf{R}_{01}^T\mathbf{R}_{02}\right) && (3.12e)\\
\text{on } 2,2: \mathbf{N}_{22} &= \mathbf{R}_{02}^T\mathbf{R}_{02} + \mathbf{R}_{12}^T\mathbf{R}_{12} + \mathbf{R}_{22}^T\mathbf{R}_{22} \Rightarrow \mathbf{R}_{22} = \text{chol}\left(\mathbf{N}_{22} - \mathbf{R}_{02}^T\mathbf{R}_{02} - \mathbf{R}_{12}^T\mathbf{R}_{12}\right), && (3.12f)
\end{aligned}$$

assuming that block i, j is stored in the main memory of process i,j. The colors represent the time step (step 1, step 2, step 3, step 4, step 5, step 6, step 7) in which the partial computation can be performed by the indicated process. All intermediate results required for the step are available as already computed (by another process). Within this small example, ten operations have to be performed (i.e. three Cholesky factorizations, three triangular solves, and four matrix multiplications). Within the first step, while the factorization of the first block is performed on process $(0, 0)$, the other processes cannot do anything, as they require the already Cholesky reduced parts. Afterwards, $(0, 0)$ provides $\mathbf{R}_{00}$ to the processes of the first compute grids row (communication operation), the two multiplications in step two can be done in parallel on $(0, 1)$ and $(0, 2)$. The other processes completed their tasks $(0, 0)$ or have to wait on the results. Within the third step, three processes perform the computations in parallel, as the needed matrices are already computed. Within steps four to seven only a single process performs computations and the other ones are pending (waiting for results needed or are already finished). Using this distribution, only the upper triangular of the compute core grid is involved as the upper triangular of the symmetric matrix is stored only in that local memory. Assuming that all operations approximately take the same time, instead of ten only seven time steps are required (factor 1.4) using 9 cores instead of a single process. At most three processes perform computations in parallel, the others are pending (waiting for results needed or are already finished).

Now, instead of a block distribution, assume a simple block-cyclic distribution of the same matrix to the same compute core grid. The matrix is partitioned into $I \times J = 6 \times 6 = 36$ sub-blocks (cf. (3.1)), compared to nine blocks (3×3) in the upper example. The block size $(b_r \times b_c)$ is chosen such that every core gets two sub-blocks (in row and column direction), i.e. four sub-blocks in total. For instance the process $0, 0$ gets the matrix blocks $\mathbf{N}_{00}$, $\mathbf{N}_{03}$, $\mathbf{N}_{30}$ and $\mathbf{N}_{33}$. The local matrix of the block-cyclic distribution would be

$$\mathbf{N}^{l}_{0,0} = \begin{bmatrix} \mathbf{N}_{00} & \mathbf{N}_{03} \\ \mathbf{N}_{30} & \mathbf{N}_{33} \end{bmatrix}. \tag{3.13}$$

As the number of rows and columns of the sub-blocks is halved compared to the upper example: a single sub-block contains only a fourth of the elements. Note that the total number of matrix elements on every process remains the same.

The block Cholesky factorization is then (at time step 1, step 2, step 3, step 4, step 5, step 6, step 7, step 8, step 9, step 10, step 11, step 12, step 13, step 14, step 15, step 16) computed distributed over the processes locally for instance from (without indicating the required communication of the already factorized matrix blocks)

$$\text{on } 0,0 : \mathbf{R}_{00} = \operatorname{chol}(\mathbf{N}_{00}) \tag{3.14a}$$
$$\text{on } 0,1 : \mathbf{R}_{01} = \mathbf{R}_{00}^{-T}\mathbf{N}_{01} \tag{3.14b}$$
$$\text{on } 0,2 : \mathbf{R}_{02} = \mathbf{R}_{00}^{-T}\mathbf{N}_{02} \tag{3.14c}$$
$$\text{on } 0,0 : \mathbf{R}_{03} = \mathbf{R}_{00}^{-T}\mathbf{N}_{03} \tag{3.14d}$$
$$\text{on } 0,1 : \mathbf{R}_{04} = \mathbf{R}_{00}^{-T}\mathbf{N}_{04} \tag{3.14e}$$
$$\text{on } 0,2 : \mathbf{R}_{05} = \mathbf{R}_{00}^{-T}\mathbf{N}_{05} \tag{3.14f}$$

$$\text{on } 1,1 : \mathbf{R}_{11} = \operatorname{chol}\left(\mathbf{N}_{11} - \mathbf{R}_{01}^{T}\mathbf{R}_{01}\right) \tag{3.14g}$$
$$\text{on } 1,2 : \mathbf{R}_{12} = \mathbf{R}_{11}^{-T}\left(\mathbf{N}_{12} - \mathbf{R}_{01}^{T}\mathbf{R}_{02}\right) \tag{3.14h}$$
$$\text{on } 1,0 : \mathbf{R}_{13} = \mathbf{R}_{11}^{-T}\left(\mathbf{N}_{13} - \mathbf{R}_{01}^{T}\mathbf{R}_{03}\right) \tag{3.14i}$$
$$\text{on } 1,1 : \mathbf{R}_{14} = \mathbf{R}_{11}^{-T}\left(\mathbf{N}_{14} - \mathbf{R}_{01}^{T}\mathbf{R}_{04}\right) \tag{3.14j}$$
$$\text{on } 1,2 : \mathbf{R}_{15} = \mathbf{R}_{11}^{-T}\left(\mathbf{N}_{15} - \mathbf{R}_{01}^{T}\mathbf{R}_{05}\right) \tag{3.14k}$$

$$\text{on } 2,2 : \mathbf{R}_{22} = \operatorname{chol}\left(\mathbf{N}_{22} - \mathbf{R}_{02}^{T}\mathbf{R}_{02} - \mathbf{R}_{12}^{T}\mathbf{R}_{12}\right) \tag{3.14l}$$
$$\text{on } 2,0 : \mathbf{R}_{23} = \mathbf{R}_{22}^{-T}\left(\mathbf{N}_{23} - \mathbf{R}_{02}^{T}\mathbf{R}_{03} - \mathbf{R}_{12}^{T}\mathbf{R}_{13}\right) \tag{3.14m}$$
$$\text{on } 2,1 : \mathbf{R}_{24} = \mathbf{R}_{22}^{-T}\left(\mathbf{N}_{24} - \mathbf{R}_{02}^{T}\mathbf{R}_{04} - \mathbf{R}_{12}^{T}\mathbf{R}_{14}\right) \tag{3.14n}$$
$$\text{on } 2,2 : \mathbf{R}_{25} = \mathbf{R}_{22}^{-T}\left(\mathbf{N}_{25} - \mathbf{R}_{02}^{T}\mathbf{R}_{05} - \mathbf{R}_{12}^{T}\mathbf{R}_{15}\right) \tag{3.14o}$$

$$\text{on } 0,0 : \mathbf{R}_{33} = \text{chol}\left(\mathbf{N}_{33} - \mathbf{R}_{03}^T\mathbf{R}_{03} - \mathbf{R}_{13}^T\mathbf{R}_{13} - \mathbf{R}_{23}^T\mathbf{R}_{23}\right) \tag{3.14p}$$

$$\text{on } 0,1 : \mathbf{R}_{34} = \mathbf{R}_{33}^{-T}\left(\mathbf{N}_{34} - \mathbf{R}_{03}^T\mathbf{R}_{04} - \mathbf{R}_{13}^T\mathbf{R}_{14} - \mathbf{R}_{23}^T\mathbf{R}_{24}\right) \tag{3.14q}$$

$$\text{on } 0,2 : \mathbf{R}_{35} = \mathbf{R}_{33}^{-T}\left(\mathbf{N}_{35} - \mathbf{R}_{03}^T\mathbf{R}_{04} - \mathbf{R}_{13}^T\mathbf{R}_{15} - \mathbf{R}_{23}^T\mathbf{R}_{25}\right) \tag{3.14r}$$

$$\text{on } 1,1 : \mathbf{R}_{44} = \text{chol}\left(\mathbf{N}_{44} - \mathbf{R}_{04}^T\mathbf{R}_{04} - \mathbf{R}_{14}^T\mathbf{R}_{14} - \mathbf{R}_{24}^T\mathbf{R}_{24} - \mathbf{R}_{34}^T\mathbf{R}_{34}\right) \tag{3.14s}$$

$$\text{on } 1,2 : \mathbf{R}_{45} = \mathbf{R}_{44}^{-T}\left(\mathbf{N}_{45} - \mathbf{R}_{04}^T\mathbf{R}_{05} - \mathbf{R}_{14}^T\mathbf{R}_{15} - \mathbf{R}_{24}^T\mathbf{R}_{25} - \mathbf{R}_{34}^T\mathbf{R}_{35}\right) \tag{3.14t}$$

$$\text{on } 2,2 : \mathbf{R}_{55} = \text{chol}\left(\mathbf{N}_{55} - \mathbf{R}_{05}^T\mathbf{R}_{05} - \mathbf{R}_{15}^T\mathbf{R}_{15} - \mathbf{R}_{25}^T\mathbf{R}_{25} - \mathbf{R}_{35}^T\mathbf{R}_{35} - \mathbf{R}_{45}^T\mathbf{R}_{45}\right). \tag{3.14u}$$

This simple computation scheme assumes, that the process which stores $\mathbf{N}_{ij}$ performs the computation which is required to determine $\mathbf{R}_{ij}$. Now 16 time steps are needed to perform the computations. As the blocks on every core are smaller by a factor of four, each step is faster by a factor of four. Thus, in units of the simple block distribution $16/4 = 4$ time units are needed using the block-cyclic distribution compared to 7 time units using the block distribution only. Now up to 8 processes are able to perform partial computations in parallel (e.g. in step 3). The lower triangular of the processor grid gets involved as it stores elements from the upper triangular of the matrices as well. This simple example shows that using a block-cyclic distribution with small sub-blocks creates a better load balancing and more processes can perform partial computations in parallel. The example tries to illustrate the gain of the block-cyclic distribution with a simple setup and a symbolic example of a computation. Note that some simplifications were introduced (e.g. time for communication was ignored) but this example might help for illustration. A better distribution of the computations would be possible (requiring more communication of blocks, using symmetry more efficient). Even more (smaller) sub-blocks would improve the load balancing as more processes can perform computations in parallel and the amount of "wait" time gets smaller. The operations (computations) are faster due to the smaller dimensions of the blocks.

4. Mathematical and Statistical Description of the Adjustment Problem

Within the previous chapters the basic concepts for numeric computations and for parallel numeric implementations were discussed. This chapter introduces the general form of the adjustment problem which is solved with the concepts and implementations introduced and derived in Chap. 2 and 3. All methods introduced here are implemented in the HPC environment to solve the general adjustment problem using the framework developed. The general methods introduced here are specialized for specific applications and solvers later on in Chapters 6, 7 and 8.

4.1 Basic Adjustment Model

The goal of the basic implementation is to find the least squares solution of a linearized Gauss-Markoff Model of the form (e.g. Koch, 1999, Sect. 3.2)

$$\boldsymbol{\ell} + \mathbf{v} = \mathbf{A}\mathbf{x}, \text{ and } \boldsymbol{\Sigma}_{\boldsymbol{\ell}\boldsymbol{\ell}} = \sigma^2 \mathbf{Q}_{\boldsymbol{\ell}\boldsymbol{\ell}}. \tag{4.1}$$

$\mathbf{A}$ is the design matrix, $\boldsymbol{\ell}$ the vector of the observations, $\mathbf{v}$ the observation residuals, σ^2 a variance factor and $\mathbf{Q}_{\boldsymbol{\ell}\boldsymbol{\ell}}$ the cofactor matrix which is assumed to be known in this context. The functional model, described by $\mathbf{A}$ a and $\mathbf{x}$ can be arbitrary — physically or mathematically motivated. The least squares solution for the unknown parameters $\mathbf{x}$ follows from the solution of the NEQs (cf. e.g. Koch, 1999, p. 160)

$$\mathbf{A}^T \boldsymbol{\Sigma}_{\boldsymbol{\ell}\boldsymbol{\ell}}^{-1} \mathbf{A}\mathbf{x} = \mathbf{A}^T \boldsymbol{\Sigma}_{\boldsymbol{\ell}\boldsymbol{\ell}}^{-1} \boldsymbol{\ell}, \tag{4.2a}$$

$$\mathbf{N}\mathbf{x} = \mathbf{n}. \tag{4.2b}$$

In general, it is assumed that the observation vector $\boldsymbol{\ell}$ is composed from independent observation groups. They can be either available as raw observations (observation equations — OEQs) plus an error description or as preprocessed NEQs already set up for subsets of the target parameter space.

4.1.1 Individual Data Sets

On the one hand there is a set of (linearized) OEQs for different groups $o \in \{0, \ldots, O-1\}$

$$\boldsymbol{\ell}_o + \mathbf{v}_o = \mathbf{A}_o \mathbf{x}, \text{ and } \boldsymbol{\Sigma}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o} = \sigma_o^2 \mathbf{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}, \tag{4.3}$$

where $\mathbf{A}_o$ are the design matrices, $\boldsymbol{\ell}_o$ the vectors of observations, σ_o^2 are unknown variance components and $\mathbf{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}$ the cofactor matrices which are assumed to be known in this context (for instance derived from empirically estimated covariance functions, e.g. Koch et al. 2010, Becker 2012; or derived by digital decorrelation filters, e.g. Schuh 2002, Siemes 2008, Schuh et al. 2010).

A second set of observation groups $n \in \{0, \ldots, N-1\}$ is a set of observations which were already preprocessed and are available in form of their least squares NEQs. These data sets are provided as a solution vector $\mathbf{x}_n$ and more frequently with an unconstrained, i.e. unregularized, full covariance matrix $\boldsymbol{\Sigma}_n$. They are used to directly recover the NEQs via

$$\mathbf{N}_n = \boldsymbol{\Sigma}_n^{-1} \text{ and } \mathbf{n}_n = \boldsymbol{\Sigma}_n^{-1} \mathbf{x}_n. \tag{4.4}$$

These NEQs (or covariance matrix plus solution vector) are sufficient statistics (cf. e.g. Kargoll, 2007, Sect. 2.5.2) of the original observations, assuming a realistic functional and stochastic model in their assembly. They can thus be used instead of the original observations without loss of information. These datasets consists of the NEQs

$$\frac{1}{\sigma_n^2}\mathbf{N}_n\mathbf{x} = \frac{1}{\sigma_n^2}\mathbf{n}_n, \tag{4.5a}$$

$$\Leftrightarrow \frac{1}{\sigma_n^2}\mathbf{\Sigma}_n^{-1}\mathbf{x} = \frac{1}{\sigma_n^2}\mathbf{\Sigma}_n^{-1}\mathbf{x}_n \tag{4.5b}$$

$$M_n \text{ in addition required for variance component estimation,} \tag{4.5c}$$

$$\lambda_n := \boldsymbol{\ell}_n^T\mathbf{Q}_{\boldsymbol{\ell}_n\boldsymbol{\ell}_n}^{-1}\boldsymbol{\ell}_n \text{ in addition required for variance component estimation,} \tag{4.5d}$$

where $\mathbf{N}_n = \mathbf{\Sigma}_n^{-1}$ are the preprocessed normal matrices and $\mathbf{n}_n = \mathbf{\Sigma}_n^{-1}\mathbf{x}_n$ the preprocessed normal vectors provided. For the use of variance component estimation (VCE) to determine the unknown variances σ_n^2, the number of observations M_n, used in the assembly of the NEQs, and the product $\lambda_n := \boldsymbol{\ell}_n^T\mathbf{Q}_{\boldsymbol{\ell}_n\boldsymbol{\ell}_n}^{-1}\boldsymbol{\ell}_n$ have to be additionally known.

4.1.2 Combined Solution

The joint least squares solution $\tilde{\mathbf{x}}$, determined from all observation groups, results from the solution of the combined system of NEQs. Assuming the individual groups to be independent, the (weighted) sum of the individual groups (addition theorem for NEQs, cf. Koch 1999, p. 177, Meissl 1982, Sect. A.10.2) yields the combined NEQs, i.e.

$$\mathbf{A}^T\mathbf{\Sigma}_{\boldsymbol{\ell\ell}}^{-1}\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{\Sigma}_{\boldsymbol{\ell\ell}}^{-1}\boldsymbol{\ell} \tag{4.6a}$$

$$\left(\sum_{n=0}^{N-1}\frac{1}{\sigma_n^2}\mathbf{N}_n + \sum_{o=0}^{O-1}\frac{1}{\sigma_o^2}\mathbf{A}_o^T\mathbf{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}^{-1}\mathbf{A}_o\right)\mathbf{x} = \sum_{n=0}^{N-1}\frac{1}{\sigma_n^2}\mathbf{n}_n + \sum_{o=0}^{O-1}\frac{1}{\sigma_o^2}\mathbf{A}_o^T\mathbf{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}^{-1}\boldsymbol{\ell}_o, \tag{4.6b}$$

$$\mathbf{N}\mathbf{x} = \mathbf{n} \tag{4.6c}$$

$$\text{and } \lambda = \sum_{n=0}^{N-1}\frac{1}{\sigma_n^2}\lambda_n + \sum_{o=0}^{O-1}\frac{1}{\sigma_o^2}\boldsymbol{\ell}_o^T\mathbf{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}^{-1}\boldsymbol{\ell}_o \tag{4.6d}$$

$$\text{and } M = \sum_{n=0}^{N-1}M_n + \sum_{o=0}^{O-1}M_o. \tag{4.6e}$$

$\mathbf{N}$ and $\mathbf{n}$ are the normal matrix and normal vector of the weighted and combined NEQs. The quantities λ and M (number of observations which entered the combined NEQs) are computed for a later use of the combined NEQ in VCE. The combined NEQs are solved to determine the unknown parameters in $\mathbf{x}$ from all groups. In addition, the unknown weights $w_o = \frac{1}{\sigma_o^2}$ and $w_n = \frac{1}{\sigma_n^2}$ have to be iteratively estimated from the data.

So far it is assumed, that all OEQs $\mathbf{A}_o$ and all NEQs $\mathbf{N}_n$ are assembled for all unknown parameters and are assembled in the same parameter ordering (i.e. a so called numbering scheme). As this is not the case when combining complementary data, a special handling will be introduced. As the different applications differ in problem size (i.e. number of observations and/or number of unknown parameters), special assembly and solutions techniques are implemented for the different applications in Chap. 6, 7 and 8 to solve for the parameters $\mathbf{x}$.

4.2 Data Weighting

Combining complementary data types to determine the joint least squares solution requires to address the question of weighting the individual observation groups. Thus, in addition to the unknown solution $\mathbf{x}$, an individual weight w_i for each group $i \in \{n, o\}$ should be estimated from the data. A data adaptive concept within satellite geodesy was developed by Koch and Kusche (2002) who interpreted the weights within a data combination procedure as inverse variance components (VCs, Förstner, 1979). In addition Koch and Kusche (2002) used variance component estimation (VCE) to determine a regularization parameter (i.e. the weight of the prior information). Generally, VCE is used to derive weights for an arbitrary number of complementary observation groups (e.g. van Loon, 2008). Using VCE, the weights $w_i = 1/\sigma_i^2$ can be determined for each group. Following again Koch and Kusche (2002), Förstner (1979) and Koch (2007, Sect. 5.2.4) they are iteratively determined via

$$\sigma_i^{(\eta)2} = \frac{\Omega_i^{(\eta)}}{M_i - \Upsilon_i^{(\eta)}} = \frac{\Omega_i^{(\eta)}}{r_i^{(\eta)}}, \tag{4.7}$$

assuming again that the observation groups are independent. M_i represents the number of observations in group i and $\Upsilon_i^{(\eta)}$ the number of parameters determined by group i. $\Omega_i^{(\eta)}$ is the weighted squared sum of residuals after iteration η

$$\Omega_i^{(\eta)} = \mathbf{v}_i^{(\eta)T} \mathbf{Q}_{\boldsymbol{\ell}_i\boldsymbol{\ell}_i}^{-1} \mathbf{v}_i^{(\eta)} = \left(\mathbf{A}_i\mathbf{x}^{(\eta)} - \boldsymbol{\ell}_i\right)^T \mathbf{Q}_{\boldsymbol{\ell}_i\boldsymbol{\ell}_i}^{-1} \left(\mathbf{A}_i\mathbf{x}^{(\eta)} - \boldsymbol{\ell}_i\right) \tag{4.8a}$$

$$= \mathbf{x}^{(\eta)T}\mathbf{N}_i\mathbf{x}^{(\eta)} - 2\mathbf{x}^{(\eta)T}\mathbf{n}_i + \boldsymbol{\ell}_i^T \mathbf{Q}_{\boldsymbol{\ell}_i\boldsymbol{\ell}_i}^{-1}\boldsymbol{\ell}_i \tag{4.8b}$$

and $r_i^{(\eta)}$ the partial redundancy (e.g Koch and Kusche, 2002)

$$r_i^{(\eta)} = M_i - \frac{1}{\sigma_i^{(\eta)2}}\operatorname{trace}\left(\mathbf{N}^{-1}\mathbf{N}_i\right) = M_i - \frac{1}{\sigma_i^{(\eta)2}}\operatorname{trace}\left(\mathbf{N}^{-1}\mathbf{A}_i^T\mathbf{Q}_{\boldsymbol{\ell}_i\boldsymbol{\ell}_i}^{-1}\mathbf{A}_i\right) = M_i - \Upsilon_i^{(\eta)}. \tag{4.9}$$

Whereas the variance components can be directly computed, when the inverse $\mathbf{N}^{-1}$ of the full NEQs is determined using (4.7), (4.8a) and (4.9), it is computational demanding for huge systems and impossible if iterative solvers are used to compute the solution $\mathbf{x}$. To handle large systems, Koch and Kusche (2002) proposed to use a Monte Carlo (MC) based stochastic trace estimator to determine the trace in (4.9). As a preparation for the use in huge systems and for an iterative solver, where $\mathbf{N}$ and thus $\mathbf{N}^{-1}$ are unknown, it is useful to introduce the stochastic estimator as well. According to Hutchinson (1990), the estimator for the trace

$$\operatorname{trace}(\mathbf{B}) = E\left\{\mathcal{P}^T\mathbf{B}\mathcal{P}\right\}, \qquad \mathcal{P} \sim \mathcal{U}(-1,1) \text{ with discrete uniform distribution,} \tag{4.10}$$

has minimal variance if $\mathbf{B}$ is symmetric. Thus (4.9) can be rewritten, to obtain symmetric matrices in the trace term. Two cases have to be distinguished: the groups available as OEQs and groups available as NEQs.

4.2.1 Partial Redundancy for Groups of NEQs

Introducing the Cholesky decomposition $\mathbf{N}_n = \mathbf{R}_n^T\mathbf{R}_n$, the partial redundancy can be rewritten as

$$r_n = M_n - \frac{1}{\sigma_n^2}\operatorname{trace}\left(\mathbf{N}^{-1}\mathbf{N}_n\right) = M_n - \frac{1}{\sigma_n^2}\operatorname{trace}\left(\mathbf{N}^{-1}\mathbf{R}_n^T\mathbf{R}_n\right) \tag{4.11a}$$

$$= M_n - \frac{1}{\sigma_n^2}\operatorname{trace}\left(\mathbf{R}_n\mathbf{N}^{-1}\mathbf{R}_n^T\right), \tag{4.11b}$$

because the trace is invariant under cyclic permutations (e.g. Koch, 1999, p. 40). Now, due to the quadratic form, $\mathbf{R}_n\mathbf{N}^{-1}\mathbf{R}_n^T$ is a symmetric matrix. Inserting the stochastic estimation from (4.10) the partial redundancy yields

$$r_n = M_n - \frac{1}{\sigma_n^2}E\left\{\mathcal{P}_n^T\mathbf{R}_n\mathbf{N}^{-1}\mathbf{R}_n^T\mathcal{P}_n\right\}, \text{with } \bar{\mathcal{P}}_n := \mathbf{R}_n^T\mathcal{P}_n \tag{4.12a}$$

$$= M_n - \frac{1}{\sigma_n^2}E\left\{\bar{\mathcal{P}}_n{}^T\mathbf{N}^{-1}\bar{\mathcal{P}}_n\right\}. \tag{4.12b}$$

Inserting K realizations of $\mathcal{P}_n$ instead of the expectation value arranged as columns in the matrix $\mathbf{P}_n$ and applying the analogous transformation

$$\bar{\mathbf{P}}_n = \mathbf{R}_n^T \mathbf{P}_n \tag{4.13}$$

the estimate for the partial redundancy as mean value from the K samples is

$$\tilde{r}_n = M_n - \tilde{\Upsilon}_n = M_n - \frac{1}{\sigma_n^2}\frac{1}{K}\text{trace}\left(\bar{\mathbf{P}}_n^T \mathbf{N}^{-1} \bar{\mathbf{P}}_n\right). \tag{4.14}$$

Note that for readability, the iteration indexes $^{(\eta)}$ and $^{(\eta-1)}$ were omitted. For all σ_n, the values of the $\eta - 1$ iteration are used in the iterative computations.

4.2.2 Partial Redundancy for Groups of OEQs

Introducing the Cholesky decomposition of the weight matrix $\mathbf{Q}_{\boldsymbol{\ell}_o \boldsymbol{\ell}_o}^{-1} = \mathbf{G}_o^T \mathbf{G}_o$, the partial redundancy from (4.9) can be rewritten as

$$r_o = M_o - \frac{1}{\sigma_o^2}\text{trace}\left(\mathbf{N}^{-1}\mathbf{A}_o^T \mathbf{G}_o^T \mathbf{G}_o \mathbf{A}\right) = M_o - \frac{1}{\sigma_o^2}\text{trace}\left(\mathbf{G}_o \mathbf{A} \mathbf{N}^{-1} \mathbf{A}_o^T \mathbf{G}_o^T\right), \tag{4.15}$$

with $\mathbf{G}_o \mathbf{A}_o \mathbf{N}^{-1} \mathbf{A}_o^T \mathbf{G}_o^T$ being symmetric. Thus again inserting the stochastic trace estimator yields

$$r_o = M_o - \frac{1}{\sigma_o^2} E\left\{\mathcal{P}_o^T \mathbf{G}_o \mathbf{A} \mathbf{N}^{-1} \mathbf{A}_o^T \mathbf{G}_o^T \mathcal{P}_o\right\}, \text{with } \bar{\mathcal{P}}_o := \mathbf{A}_o^T \mathbf{G}_o^T \mathcal{P}_o \tag{4.16a}$$

$$r_o = M_o - \frac{1}{\sigma_o^2} E\left\{\bar{\mathcal{P}}_o^T \mathbf{N}^{-1} \bar{\mathcal{P}}_o\right\}. \tag{4.16b}$$

Replacing again the expectation value by K realizations arranged in the matrix $\mathbf{P}_o$ and applying the same transformation $\bar{\mathbf{P}}_o = \mathbf{A}_o^T \mathbf{G}_o^T \mathbf{P}_o$ the estimate for the partial redundancy is

$$\tilde{r}_o = M_o - \tilde{\Upsilon}_o = M_o - \frac{1}{\sigma_o^2}\frac{1}{K}\text{trace}\left(\bar{\mathbf{P}}_o^T \mathbf{N}^{-1} \bar{\mathbf{P}}_o\right), \tag{4.17}$$

already introducing the mean value of all realizations.

4.2.3 Computations of VCs Using the MC Approach

Whereas Ω_i can be directly computed from (4.8a) for $i \in \{n, o\}$, the computation of the partial redundancies, especially the trace term $\bar{\mathbf{P}}_i^T \mathbf{N}^{-1} \bar{\mathbf{P}}_i$ in (4.14) and (4.17), has to be adapted for both groups n and o. As, depending on the used solver, $\mathbf{N}^{-1}$ is not necessarily computed, an alternative computation is derived here. Defining

$$\mathbf{Z}_i := \mathbf{N}^{-1}\bar{\mathbf{P}}_i, \qquad \Leftrightarrow \mathbf{N}\mathbf{Z}_i = \bar{\mathbf{P}}_i, \tag{4.18}$$

$\mathbf{Z}_i$ can be directly determined via solving the system of NEQs for additional right hand sides $\bar{\mathbf{P}}_i$. Thus, for every group o, K additional right hand sides

$$\bar{\mathbf{P}}_o = \mathbf{A}_o^T \mathbf{G}_o^T \mathbf{P}_o, \text{ with dimension } M_i \times K \tag{4.19}$$

and for each group n

$$\bar{\mathbf{P}}_n = \mathbf{R}_n^T \mathbf{P}_n, \text{ with dimension } U_i \times K, \text{ extended to } U \times K \text{ by } \mathbf{0}. \tag{4.20}$$

are sampled. Instead of (4.6c) the system with $NK + OK + 1$ right hand sides

$$\mathbf{N}\begin{bmatrix}\mathbf{x} & \mathbf{Z}_1 & \cdots & \mathbf{Z}_O & \mathbf{Z}_1 & \cdots & \mathbf{Z}_N\end{bmatrix} = \begin{bmatrix}\mathbf{n} & \bar{\mathbf{P}}_1 & \cdots & \bar{\mathbf{P}}_O & \bar{\mathbf{P}}_1 & \cdots & \bar{\mathbf{P}}_N\end{bmatrix} \tag{4.21}$$

has to be solved. The variance components can be then derived without explicit knowledge of $\mathbf{N}^{-1}$ via

$$\sigma_i^{(\eta)2} = \frac{\Omega_i}{M_i - \frac{1}{\sigma_i^{(\eta-1)2}}\frac{1}{K}\text{trace}\left(\bar{\mathbf{P}}_i^T \mathbf{Z}_i\right)}. \tag{4.22}$$

4.3 Numbering Schemes and Reordering

Eq. (4.6b) assumes that all observation groups n are available in the same parameter space and the same parameter ordering. In addition it is assumed, that the OEQs are set up for the same (i.e. entire) parameter space. Within adjustment problems the parameter space corresponds to the columns of the design matrix and the rows and columns of the NEQ matrix (and the rows of the right hand side). Thus, a column of the design matrix, a row or column of $\mathbf{N}$ is always related to the specific parameter $\mathbf{x}_i$. Within the setup of OEQs, the column i has to be linked to the specific parameter $\mathtt{p}$, to set up the corresponding base function. Combining different data sets, especially if they are already available as NEQs, the ordering of the parameters of the different groups has to be unified. It can not be expected, that the preprocessed NEQs are set up for the entire target parameter space, as not all observation types are sensitive to all parameters set up in the functional model (e.g. group specific parameters, biases, instrument specific parameters, etc.). Typically, within adjustment theory, the order of the parameters in the column of the design matrix or in the rows of the parameter vector are called numbering scheme of the parameters, this term will be used here as well.

As there exists no "default" parameter order which groups the parameters of more than two parameter subsets in blocks, such that the combination of matrices via block additions without gaps is possible, so called reordering of the parameters (i.e. interchanges of columns and/or rows of involved matrices) is mandatory.

4.3.1 Numbering Schemes

To be as flexible as possible and not restricted to a special numbering scheme, symbolic numbering schemes are used within this work. These general numbering schemes are sequences (one dimensional fields) of objects of a class `parameter`, which mainly includes an unique parameter identifier (or symbolic name) which makes the parameters comparable. For every NEQ involved, an associated symbolic numbering scheme is provided, e.g., a file listing the parameter order symbolically (via a identifier or name). These parameters can be mapped to an object and corresponding parameters can be identified via compare algorithms. The further concepts require, that the parameters are comparable. Thus compare operators, like `operator==( const parameter & p2 )` and `operator<( const parameter & p2 )`, are defined for them.

Arranging a sequence of these parameters in a field, e.g. in a `std::vector<Parameter>`, defines a symbolic numbering scheme which can be associated with matrices (their rows and/or columns). The column and/or row i of a matrix corresponds to the parameter $\mathbf{p}(i)$ using a class `NumberingScheme`. As parameters are comparable, a "search" operation in the vector of symbolic parameters is possible and as the operator `operator<( const parameter & p2 )` is defined, a sorting of the parameters is possible. Within this work, a numbering scheme related to a matrix is denoted as $\mathbf{p}$, whereas the ith parameter of that numbering scheme is $\mathbf{p}(i)$. A numbering scheme contains all U unknown parameters $i \in \{0, ..., U-1\}$. An specific parameter and numbering scheme class implementation is shown for applications related to the gravity field in Sect. 5.3.

4.3.2 Reordering Between Symbolic Numbering Schemes

The goal of this section is to derive an algorithm for the reordering of rows and/or columns of a matrix given in a source numbering scheme $\mathbf{p}_\mathrm{f}$ to another target numbering scheme $\mathbf{p}_\mathrm{t}$. Only a single assumption on this numbering scheme is necessary, either $\mathbf{p}_\mathrm{f} \subseteq \mathbf{p}_\mathrm{t}$ or $\mathbf{p}_\mathrm{t} \subseteq \mathbf{p}_\mathrm{f}$. The algorithms provided work in both directions.

Index Vector for Reordering A first quantity to derive is the so called index vector $\mathbf{i}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}$, the entry at position i contains the index, the coefficient $\mathbf{p}_\mathrm{t}(i)$ can be found in $\mathbf{p}_\mathrm{f}$, and thus $\mathbf{p}_\mathrm{t}(i) = \mathbf{p}_\mathrm{f}(\mathbf{i}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}(i))$. Given a matrix $\mathbf{A}$ in $\mathbf{p}_\mathrm{f}$, its rows are reordered to $\mathbf{p}_\mathrm{t}$ via

$$\mathbf{A}_{\mathbf{p}_\mathrm{t}} = \mathbf{A}_{\mathbf{p}_\mathrm{f}}(\mathbf{i}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}, :), \tag{4.23}$$

its columns via

$$\mathbf{A}_{\mathbf{p}_\mathrm{t}} = \mathbf{A}_{\mathbf{p}_\mathrm{f}}(:, \mathbf{i}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}), \tag{4.24}$$

and for quadratic matrices rows and columns via

$$\mathbf{A}_{\mathbf{p}_\mathrm{t}} = \mathbf{A}_{\mathbf{p}_\mathrm{f}}(\mathbf{i}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}, \mathbf{i}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}), \tag{4.25}$$

using the well known MatLab/Octave like notation. The index vector $\mathbf{i}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}$ can be computed with the descriptive Alg. 4.1. which works for both cases, i.e. $\mathbf{p}_\mathrm{f} \subseteq \mathbf{p}_\mathrm{t}$ or $\mathbf{p}_\mathrm{t} \subseteq \mathbf{p}_\mathrm{f}$.

Note that if $\mathbf{p}_\mathrm{f} \subseteq \mathbf{p}_\mathrm{t}$, the matrix to be reordered will be extended by $\mathbf{0}$ rows and/or columns to obtain the correct size of $\mathbf{p}_\mathrm{t}$`.size()`, the index vector for this parameters not originally contained in $\mathbf{p}_\mathrm{f}$ are set to values corresponding to these $\mathbf{0}$ rows and/or columns at the end of the matrices (variable e in Alg. 4.1).

For the case $\mathbf{p}_\mathrm{t} \subseteq \mathbf{p}_\mathrm{f}$, the index vector is extended to the dimension of $\mathbf{p}_\mathrm{f}$. The last entries i.e. $\mathbf{p}_\mathrm{f}$`.size()` to $\mathbf{p}_\mathrm{t}$`.size()`-1 are set to the indices of the entry itself to guarantee, that the coefficients not contained in $\mathbf{p}_\mathrm{t}$ are reordered to the end of the matrix. Applying the reordering to a matrix, these last rows/columns are deleted after the reordering.

Although Alg. 4.1 can be easily understood, it is not very efficient due to the "`find`" operation, i.e. a search operation in a vector which might have millions of entries for huge dimensional parameter spaces. An alternative, i.e. a more efficient way can be performed via sorting the numbering scheme. This much faster algorithm is summarized in Alg. 4.2. It reduces the complexity from $\mathcal{O}(n^2/2)$ to $\mathcal{O}(n\log(n))$. For a test case of two numbering schemes of 520 000 parameters, the runtime is reduced from 706 s to 0.2 s.

Permutation Vector for Reordering An alternative notation/operation, which is better suited for the block-cyclic distributed matrices, is a so called permutation or pivoting vector. In contrast to an index vector, where the column and or row interchanges are assumed to be performed simultaneously, a permutation vector (at least as used here) contains a sequence of serial row and column permutations i.e. a sequential swapping of two rows/columns starting at begin (index 0) of the vector. In contrast to the index vector, already performed swapping operations are taken into account in the representation. An entry in the permutation vector at position i means that the row i is swapped with row $\boldsymbol{\psi}(i)$. To be more precise, the current content of row/column i is swapped with the current content of row/column $\boldsymbol{\psi}(i)$. Note that the content might change with every swapping operation. Now, the old entry of position i is in row $\boldsymbol{\psi}(i)$, thus the index vector needs to be updated. A remaining entry i in the subsequent elements of $\mathbf{i}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}$ has to be replaced by the entry $\boldsymbol{\psi}(i)$. The procedure to convert an index vector to a permutation vector is summarized in Alg. 4.3.

A more efficient but less obvious algorithm is shown in Alg. 4.4. The basic idea is to avoid the search operation via introducing a second vector which stores the position of an entry k in the vector. This reduces the complexity from $\mathcal{O}(n^2/2)$ to $\mathcal{O}(2n)$. The runtime is reduced from 59 s to 0.01 s, for an example index vector with 520 000 entries.

To apply a permutation vector to rows/and or columns, the operator $\mathbf{\Psi}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}(\,\cdot\,)$ is defined. This operator performs the serial permutations of rows ($\mathbf{\Psi}^r_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}$) as given by the vector $\boldsymbol{\psi}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}$, the operator $\mathbf{\Psi}^c_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}$ performs the column interchanges and $\mathbf{\Psi}^{r,c}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}$ performs the interchanges for rows and columns.

Algorithm 4.1: Simple version to compute an index vector from two symbolic numbering schemes.

Data:

`NumberingSchme` $\mathbf{p}_{\text{from}}$	Symbolic numbering scheme source matrix is ordered in
`NumberingSchme` $\mathbf{p}_{\text{into}}$	Symbolic numbering scheme matrix should be reordered to

```
// initialization of index vector
vector<size_t> $\mathbf{i}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}$ ( $\mathbf{p}_{\text{into}}$.size(),0 )
// start value for fill in indices for parameters in $\mathbf{p}_{into}$ but not in $\mathbf{p}_{from}$, inserted at the end
size_t $e = \mathbf{p}_{\text{from}}$.size()
// loop over all parameters in $\mathbf{p}_{into}$
for $k = 0$ to $\mathbf{p}_{into}$.size() do
    // find index of parameter $\mathbf{p}_{into}(i)$ in $\mathbf{p}_{from}$
    $i$ =find($\mathbf{p}_{\text{from}}$.begin(), $\mathbf{p}_{\text{from}}$.end(), $\mathbf{p}_{\text{into}}(i)$ )
    // if parameter found insert index i, otherwise fill in value outside of $\mathbf{p}_{from}$.size()
    if $i$<$\mathbf{p}_{from}$.size() then
        $\mathbf{i}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}(k) = i$
    else
        $\mathbf{i}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}(k) = e$
        $e++$
    end
end
// special case if $\mathbf{p}_{into} \subseteq \mathbf{p}_{from}$: extend index vector to size of $\mathbf{p}_{from}$
if $\mathbf{p}_{into}$.size() $<$ $\mathbf{p}_{from}$.size() then
    $\mathbf{i}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}$.resize($\mathbf{p}_{\text{from}}$.size())
    // the remaining parameters are sorted to the end as they are not contained in $\mathbf{p}_{into}$,
    // applied to a matrix, these last rows/columns will be removed
    for $k = \mathbf{p}_{into}$.size() to $\mathbf{p}_{from}$.size() do
        $\mathbf{i}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}(k) = k$
    end
end
return $\mathbf{i}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}$// index vector performing reordering from $\mathbf{p}_{\text{from}}$ to $\mathbf{p}_{\text{into}}$
```

Example Tab. 4.1 gives a small scale example of reordering and permutation for two symbolic numbering schemes. Index and permutation vector are provided for both directions. In addition, the first steps of a sequential permutation are characterized.

4.3.3 Reordering of Block-cyclic Distributed Matrices

If the permutation vector between two numbering schemes is known, these permutations have to be applied to block-cyclic distributed matrices to reorder the matrices to the target numbering scheme. The reason why the permutation vector was introduced is, that there exist a SCALAPACK helper routine, which is internally used for pivoting during the solution of a system of equations. The routine `pdlapiv` can be directly used to perform the reordering of a distributed matrix, where the input is a permutation vector (The usage of `pdlaswp` is also possible). The routine applies the permutation as given by $\boldsymbol{\psi}_{\mathbf{p}_\text{f}\mapsto\mathbf{p}_\text{t}}$ to either rows or columns. Applying the function twice, first to permute rows and secondly to permute columns, both are reordered. Beside the standard input of the block-cyclic distribution of the matrix, the function requires the input of the permutation vector in a special form. The permutation vector (consisting of integers only) needs to be passed as a block-cyclic distributed vector of integers, such that the local entry $\boldsymbol{\psi}^{l}_{\mathbf{p}_\text{f}\mapsto\mathbf{p}_\text{t}}(i)$ contains the global row/column, the local row/column i has to be swapped with. The serial integer vector is brought to a block-cyclic distributed vector as introduced for the block-cyclic distributed matrices in Sect. 3.3.

Algorithm 4.2: Fast version to compute an index vector from two symbolic numbering schemes.

Data:

`NumberingSchme` $\mathbf{p}_{\text{from}}$	Symbolic numbering scheme source matrix is ordered in
`NumberingSchme` $\mathbf{p}_{\text{into}}$	Symbolic numbering scheme matrix should be reordered to

```
// initialization of index vector
vector<size_t> $\mathbf{i}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}$ ( $\mathbf{p}_{\text{into}}$.size(),0 )
// start value for fill in indices for parameters in $\mathbf{p}_{into}$ but not in $\mathbf{p}_{from}$, inserted at the end
size_t $e = \mathbf{p}_{\text{from}}$.size()
// store current index of parameter in helping atribut _i of each individual parameter
$\mathbf{p}_{\text{from}}$.setIndex()
// sort numbering schme with implemented sort function (operator<)
$\mathbf{p}_{\text{from}}$.sort()
// loop over all parameters in $\mathbf{p}_{into}$
for $k = 0$ to $\mathbf{p}_{into}$.size() do
    // find index of parameter $\mathbf{p}_{into}(i)$ in $\mathbf{p}_{from}$ in sorted numbering scheme
    $i$ =find($\mathbf{p}_{\text{from}}$.begin(), $\mathbf{p}_{\text{from}}$.end(), $\mathbf{p}_{\text{into}}(i)$ )
    // if parameter found, insert index i, otherwise fill in value outside of $\mathbf{p}_{from}$.size()
    if $i<\mathbf{p}_{from}$.size() then
        $\mathbf{i}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}(k) = \mathbf{p}_{\text{from}}.p(i).i()$
    else
        $\mathbf{i}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}(k) = e$
        $e++$
    end
end
// special case if $\mathbf{p}_{into} \subseteq \mathbf{p}_{from}$: extend index vector to size of $\mathbf{p}_{from}$
if $\mathbf{p}_{into}$.size() $<$ $\mathbf{p}_{from}$.size() then
    $\mathbf{i}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}$.resize($\mathbf{p}_{\text{from}}$.size())
    // the reamaining parameters are sorted to the end as they are not contained in $\mathbf{p}_{into}$,
    // applied to a matrix, these last rows/columns will be removed
    for $k = \mathbf{p}_{into}$.size() to $\mathbf{p}_{from}$.size() do
        $\mathbf{i}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}(k) = k$
    end
end
return $\mathbf{i}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}$// index vector performing reordering from $\mathbf{p}_{\text{from}}$ to $\mathbf{p}_{\text{into}}$
```

The reordering operations are implemented in the member functions `reorder, reorderCols` and `reorderRows` of the `DistributedMatrix` class in Listing 3.2.

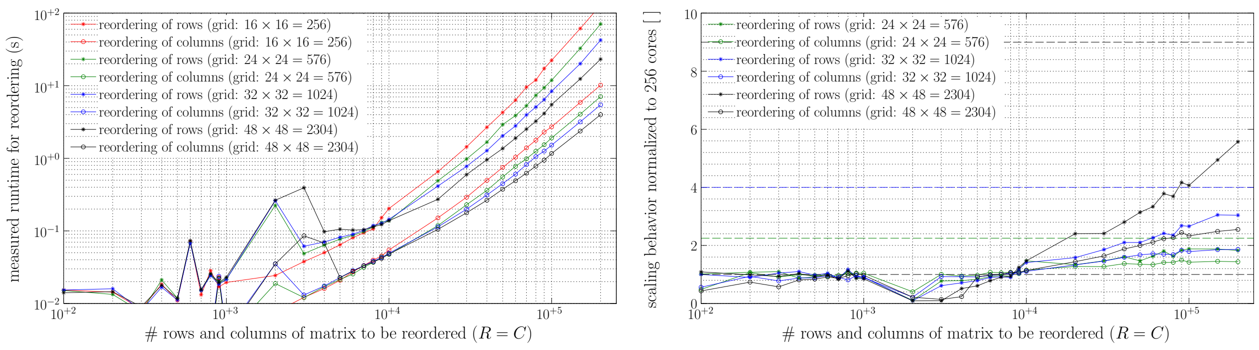
Fig. 4.1 gives an overview about the required runtime for the reordering of rows and columns of distributed matrices of different dimension on different quadratic processor grids (for the distribution parameters default values of $b_r = b_c = 64$ were used) to get an idea of the order of magnitude. The index vector was randomly generated (random shuffle of an index vector). The main conclusions are: i) the reordering of columns is much faster than the reordering of rows (factor of three to ten). This could be expected as the column access in memory is much faster using the column major order for matrices (cf. Sect. 2.2.1) for the locally stored matrices. ii) for matrices of dimension lower than $20\,000 \times 20\,000$ the reordering is performed in less than 1 s on all grids. For the reordering of columns, this even holds for matrices smaller than $80\,000 \times 80\,000$. Although there is no real scaling behavior of the reordering operations with the number of cores (cf. Fig. 4.1(b)), the most important thing is that the performance increases on larger compute core grids and does not drop to additional organizational requirements (at least for matrices above dimension $10\,000 \times 10\,000$ the scaling is above 1.0 for all cases analyzed).

Algorithm 4.3: Simple version to convert an index vector to a permutation vector.

Data:

`NumberingSchme` $\mathbf{i}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}$ index vector to be converted to permutation vector

```
// initialization of permutation vector
vector<size_t> $\boldsymbol{\psi}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}} = \mathbf{i}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}$
size_t $p = 0$; // loop over entries of index vector
for $k = 0$ to $\mathbf{i}_{\mathbf{p}_{from}\mapsto\mathbf{p}_{into}}$.size() do
    // find current index in the subsequent part of index vector
    $p$ =find($\mathbf{p}_{\text{from}}$.begin()+$k$, $\mathbf{p}_{\text{from}}$.end(), $k$ )
    // if index found, replace it by its new position $\boldsymbol{\psi}_{\mathbf{p}_{from}\mapsto\mathbf{p}_{into}}(k)$ (after swapping)
    if found then
        $\boldsymbol{\psi}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}(p) = \boldsymbol{\psi}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}(k)$
    end
end
return $\boldsymbol{\psi}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}$// sequential permutation vector corresponding to $\mathbf{i}_{\mathbf{p}_{\text{from}}\mapsto\mathbf{p}_{\text{into}}}$
```



(a) Mean absolute runtime measured for the reordering. (b) Scaling behavior of reordering operations.

Figure 4.1: Runtime analysis of the row ($\star$) and column ($\circ$) reordering operations (index vector randomly generated). The colors represent different dimensions of the processor grid. The scaling is normalized to 256 cores and given for every matrix dimension.

4.4 Combined System of NEQs

Assuming the OEQs to be set up for the whole parameter space in the correct numbering scheme, but combining NEQs set up for a subset of the parameters only, (4.6b) can be rewritten, taken into account the concepts of reordering and permutation. Assuming that the target numbering scheme (associated with $\mathbf{N}$) covers the entire parameter space, (4.6b) has to be rewritten as

$$\left(\sum_{n=0}^{N-1}\frac{1}{\sigma_n^2}\begin{bmatrix}\mathbf{N}_n & \mathbf{0}_{U_n\times U-U_n}\\ \mathbf{0}_{U-U_n\times U_n} & \mathbf{0}_{U-U_n\times U-U_n}\end{bmatrix}(\mathbf{i}_{\mathbf{p}_n\mapsto\mathbf{p}},\mathbf{i}_{\mathbf{p}_n\mapsto\mathbf{p}})+\sum_{o=0}^{O-1}\frac{1}{\sigma_o^2}\mathbf{A}_o^T\mathbf{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}^{-1}\mathbf{A}_o\right)\mathbf{x}$$
$$=\sum_{n=0}^{N-1}\frac{1}{\sigma_n^2}\begin{bmatrix}\mathbf{n}_n\\ \mathbf{0}_{U-U_n\times 1}\end{bmatrix}(\mathbf{i}_{\mathbf{p}_n\mapsto\mathbf{p}})+\sum_{o=0}^{O-1}\frac{1}{\sigma_o^2}\mathbf{A}_o^T\mathbf{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}^{-1}\boldsymbol{\ell}_o, \tag{4.26}$$

Algorithm 4.4: Efficient version to convert an index vector to a permutation vector avoiding the find operation.

```
Data:
  NumberingSchme $\mathbf{i}_{\mathbf{p}_{\mathrm{from}}\mapsto\mathbf{p}_{\mathrm{into}}}$    index vector to be converted to permutation vector

// initialization of permutation vector
vector<size_t>  $\boldsymbol{\psi}_{\mathbf{p}_{\mathrm{from}}\mapsto\mathbf{p}_{\mathrm{into}}} = \mathbf{i}_{\mathbf{p}_{\mathrm{from}}\mapsto\mathbf{p}_{\mathrm{into}}}$
size_t $p = 0$
// help vector, entry $\mathbf{h}(k)$ contains index where value $k$ is stored in $\boldsymbol{\psi}_{\mathbf{p}_{from}\mapsto\mathbf{p}_{into}}$
vector<size_t>  h( $\mathbf{i}_{\mathbf{p}_{\mathrm{from}}\mapsto\mathbf{p}_{\mathrm{into}}}$.size(), 0 )
for $k = 0$ to $\mathbf{i}_{\mathbf{p}_{from}\mapsto\mathbf{p}_{into}}$.size() do
    $\mathbf{h}(\mathbf{i}_{\mathbf{p}_{\mathrm{from}}\mapsto\mathbf{p}_{\mathrm{into}}}) = k$
end
// loop over entries of index vector
for $k = 0$ to $\mathbf{i}_{\mathbf{p}_{from}\mapsto\mathbf{p}_{into}}$.size() do
    // index of number k follows from h instead of find operation
    $p = \mathbf{h}(k)$
    // check if entry k is in subsequent part of vector
    if $p > k$ then
        $\boldsymbol{\psi}_{\mathbf{p}_{\mathrm{from}}\mapsto\mathbf{p}_{\mathrm{into}}}(p) = \boldsymbol{\psi}_{\mathbf{p}_{\mathrm{from}}\mapsto\mathbf{p}_{\mathrm{into}}}(k)$
        // update the vector h, value $\boldsymbol{\psi}_{\mathbf{p}_{from}\mapsto\mathbf{p}_{into}}(k)$ is no in position $p$
        $\mathbf{h}(\boldsymbol{\psi}_{\mathbf{p}_{\mathrm{from}}\mapsto\mathbf{p}_{\mathrm{into}}}(k)) = p$
    end
end
return $\boldsymbol{\psi}_{\mathbf{p}_{\mathrm{from}}\mapsto\mathbf{p}_{\mathrm{into}}}$ // sequential permutation vector corresponding to $\mathbf{i}_{\mathbf{p}_{\mathrm{from}}\mapsto\mathbf{p}_{\mathrm{into}}}$
```

Table 4.1: Example for the reordering of two symbolic numbering schemes, given are the index vectors and permutation vectors in both directions according to Alg. 4.2 and 4.4.

(a) First direction

$\mathbf{p}_\mathrm{f}$	$\mathbf{p}_\mathrm{t}$	$\mathbf{i}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}$	$\boldsymbol{\psi}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}$	$\mathbf{p}_\mathrm{f}(\mathbf{i}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}})$
C 3 0	C 2 0	16	16	0
C 4 0	C 2 1	17	17	0
C 3 1	C 2 2	18	18	0
C 4 1	C 3 0	0	16	c 3 0
S 3 1	C 3 1	2	18	c 3 1
S 4 1	C 3 2	6	6	c 3 2
C 3 2	C 3 3	10	10	c 3 3
C 4 2	C 4 0	1	17	c 4 0
S 3 2	C 4 1	3	16	c 4 1
S 4 2	C 4 2	7	17	c 4 2
C 3 3	C 4 3	11	11	c 4 3
C 4 3	C 4 4	14	14	c 4 4
S 3 3	C 5 0	19	19	0
S 4 3	C 5 1	20	20	0
C 4 4	C 5 2	21	21	0
S 4 4	C 5 3	22	22	0
	C 5 4	23	23	0
	C 5 5	24	24	0
	S 2 1	25	25	0
	S 2 2	26	26	0
	S 3 1	4	25	s 3 1
	S 3 2	8	23	s 3 2
	S 3 3	12	26	s 3 3
	S 4 1	5	23	s 4 1
	S 4 2	9	24	s 4 2
	S 4 3	13	25	s 4 3
	S 4 4	15	26	s 4 4
	S 5 1	27	27	0
	S 5 2	28	28	0
	S 5 3	29	29	0
	S 5 4	30	30	0
	S 5 5	31	31	0

(b) Second direction

$\mathbf{p}_\mathrm{f}$	$\mathbf{p}_\mathrm{t}$	$\mathbf{i}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}$	$\boldsymbol{\psi}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}$	$\mathbf{p}_\mathrm{f}(\mathbf{i}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}})$
C 2 0	C 3 0	3	3	c 3 0
C 2 1	C 4 0	7	7	c 4 0
C 2 2	C 3 1	4	4	c 3 1
C 3 0	C 4 1	8	8	c 4 1
C 3 1	S 3 1	20	20	s 3 1
C 3 2	S 4 1	23	23	s 4 1
C 3 3	C 3 2	5	23	c 3 2
C 4 0	C 4 2	9	9	c 4 2
C 4 1	S 3 2	21	21	s 3 2
C 4 2	S 4 2	24	24	s 4 2
C 4 3	C 3 3	6	23	c 3 3
C 4 4	C 4 3	10	23	c 4 3
C 5 0	S 3 3	22	22	s 3 3
C 5 1	S 4 3	25	25	s 4 3
C 5 2	C 4 4	11	23	c 4 4
C 5 3	S 4 4	26	26	s 4 4
C 5 4		16	16	c 5 4
C 5 5		17	17	c 5 5
S 2 1		18	18	s 2 1
S 2 2		19	19	s 2 2
S 3 1		20	20	s 3 1
S 3 2		21	21	s 3 2
S 3 3		22	22	s 3 3
S 4 1		23	23	s 4 1
S 4 2		24	24	s 4 2
S 4 3		25	25	s 4 3
S 4 4		26	26	s 4 4
S 5 1		27	27	s 5 1
S 5 2		28	28	s 5 2
S 5 3		29	29	s 5 3
S 5 4		30	30	s 5 4
S 5 5		31	31	s 5 5

(c) Application of $\boldsymbol{\psi}_{\mathbf{p}_\mathrm{f}\mapsto\mathbf{p}_\mathrm{t}}$ (2. direction)

step:	0.	1.	2.	···	31.
0	c 2 0	c 3 0	c 3 0		c 3 0
1	c 2 1	c 2 1	c 4 0		c 4 0
2	c 2 2	c 2 2	c 2 2		c 3 1
3	c 3 0	c 2 0	c 2 0		c 4 1
4	c 3 1	c 3 1	c 3 1		s 3 1
5	c 3 2	c 3 2	c 3 2		s 4 1
6	c 3 3	c 3 3	c 3 3		c 3 2
7	c 4 0	c 4 0	c 2 1		c 4 2
8	c 4 1	c 4 1	c 4 1		s 3 2
9	c 4 2	c 4 2	c 4 2		s 4 2
10	c 4 3	c 4 3	c 4 3		c 3 3
11	c 4 4	c 4 4	c 4 4		c 4 3
12	c 5 0	c 5 0	c 5 0		s 3 3
13	c 5 1	c 5 1	c 5 1		s 4 3
14	c 5 2	c 5 2	c 5 2		c 4 4
15	c 5 3	c 5 3	c 5 3	···	s 4 4
16	c 5 4	c 5 4	c 5 4		c 5 4
17	c 5 5	c 5 5	c 5 5		c 5 5
18	s 2 1	s 2 1	s 2 1		s 2 1
19	s 2 2	s 2 2	s 2 2		s 2 2
20	s 3 1	s 3 1	s 3 1		c 2 2
21	s 3 2	s 3 2	s 3 2		c 2 0
22	s 3 3	s 3 3	s 3 3		c 5 0
23	s 4 1	s 4 1	s 4 1		c 5 2
24	s 4 2	s 4 2	s 4 2		c 2 1
25	s 4 3	s 4 3	s 4 3		c 5 1
26	s 4 4	s 4 4	s 4 4		c 5 3
27	s 5 1	s 5 1	s 5 1		s 5 1
28	s 5 2	s 5 2	s 5 2		s 5 2
29	s 5 3	s 5 3	s 5 3		s 5 3
30	s 5 4	s 5 4	s 5 4		s 5 4
31	s 5 5	s 5 5	s 5 5	···	s 5 5
swap:	0&3	1&7	2&4		

for the use with an index vector. Using the introduced permutation operator, i.e.

$$\left(\sum_{n=0}^{N-1}\frac{1}{\sigma_n^2}\mathbf{\Psi}^{r,c}_{\mathbf{p}_n\mapsto\mathbf{p}}\left(\begin{bmatrix}\mathbf{N}_n & \mathbf{0}_{U_n\times U-U_n}\\ \mathbf{0}_{U-U_n\times U_n} & \mathbf{0}_{U-U_n\times U-U_n}\end{bmatrix}\right)+\sum_{o=0}^{O-1}\frac{1}{\sigma_o^2}\mathbf{A}_o^T\mathbf{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}^{-1}\mathbf{A}_o\right)\mathbf{x}$$
$$=\sum_{n=0}^{N-1}\frac{1}{\sigma_n^2}\mathbf{\Psi}^{r}_{\mathbf{p}_n\mapsto\mathbf{p}}\left(\begin{bmatrix}\mathbf{n}_n\\ \mathbf{0}_{U-U_n\times 1}\end{bmatrix}\right)+\sum_{o=0}^{O-1}\frac{1}{\sigma_o^2}\mathbf{A}_o^T\mathbf{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}^{-1}\boldsymbol{\ell}_o, \tag{4.27}$$

assuming $\mathbf{N}_n$ and $\mathbf{n}_n$ being the original NEQs as they are available. Their numbering scheme is denoted as $\mathbf{p}_n$. These NEQs for the subset of the parameters are extended with zeros, if required. Afterwards the index-vector or the permutation vector is applied to the temporary extended NEQs. Afterwards, from a mathematical point of view the NEQs can be combined performing a simple addition as the parameter order and parameter space is adjusted to the defined target numbering scheme $\mathbf{p}$.

4.5 Summary

The developed framework provides the general modules to solve any adjustment problems of the introduced general form and special cases like for instance $O = 0$, $N = 0$ or same parameter spaces in all groups, etc. Of course, application specific extensions are required, i.e. for instance the definition of symbolic parameters and numbering schemes, implementation of the observation equations (design matrices), tailored decorrelation procedures or routines to read the observations and the required observations meta data. The developed basic framework will now be used in different solvers and specialized for specific applications from global gravity field determination (spherical harmonic analysis). Three applications with different problem characteristics are chosen to set up and solve the adjustment problem for the unknown parameters. Within a detailed description it is shown how the developed framework can be used for an efficient implementation of a specific problem. Although this specialization is shown for gravity field applications only, the concepts provided and the approach towards the solution can be easily transferred to adjustment problems with any other observation type and other functional models (mathematical base functions). The concepts and the implementation strategy implemented assume, that the observation equations are dense. For sparse systems, better suited alternatives exist (e.g. Paige and Saunders, 1982, Dongarra et al., 1994, Saad, 2000).

Part II

Specialization and Application to Global Gravity Field Recovery

5. Recovery of Global Gravity Field Models

The computational, implementational and statistical basics discussed in the previous chapters will be specialized to implement a specific adjustment problem based on the developed framework in a HPC environment. Within this chapter, the basics required for the applications related to the determination of the global Earth's gravity field (Chapters 6–8) are summarized and the required introduction from physical and theoretical geodesy is provided.

Global Earth's gravity field models are often described as a finite series of spherical harmonic coefficients (e.g. Heiskanen and Moritz, 1993, p. 59). The potential in a point with spherical coordinates $(\lambda,\, \theta,\, r)$ is

$$V\left(r,\theta,\lambda\right) = \frac{GM}{a}\sum_{l=0}^{l_{\max}}\left(\frac{a}{r}\right)^{l+1}\sum_{m=0}^{l}\left(c_{lm}\cos\left(m\lambda\right)+s_{lm}\sin\left(m\lambda\right)\right)P_{lm}\left(cos\theta\right), \tag{5.1}$$

where l and m denote the spherical harmonic degree and order (d/o), c_{lm} and s_{lm} the coefficients of the spherical harmonic series, a the equatorial radius of the Earth reference ellipsoid, $P_{lm}\left(\,\cdot\,\right)$ the fully normalized associated Legendre functions, and GM the gravitational constant of the Earth. $l_{\max}$ is the degree of expansion which defines the spatial resolution, which is approximately $2\pi a/\left(2l_{\max}\right)$.

In general, three types of global gravity field models can be distinguished, mainly grouped by the observations they are computed from and the resolution of the models (sensitivity of observations entering the solutions). This chapter is an extension of the introduction and collection of methods already published in Brockmann et al. (2014c), so some parts of this chapter are close to that paper.

5.1 Types of Global Gravity Field Models and State of the Art

The first type is the class of satellite-only gravity field models, where consistent gravity field models are derived from the observations from a single dedicated satellite mission like CHAMP (CHAllenging Mini-Satellite Payload, Reigber et al., 2002), GRACE (GRAvity recovery and Climate Experiment, Tapley et al., 2004) or GOCE (Gravity field and steady-state Ocean Circulation Explorer, ESA, 1999). Each of the models reflect the particular strengths of the satellites observation techniques. The observations are analyzed by experts in the missions, who spend a lot of effort in the functional and the stochastic modeling (e.g. van Loon, 2008, Beutler et al., 2010, Mayer-Gürr et al., 2010b, Pail et al., 2011a). Meanwhile, in addition to the coefficients, due to a lot of effort in stochastic modeling, realistic covariance matrices are provided for some of the models (see for GRACE and GOCE e.g. Mayer-Gürr et al., 2010a,b, Pail et al., 2011a, Schuh et al., 2010). From a computational point of view, the individual satellites are only sensitive to the long and medium wavelengths of the gravity field. Therefore, the models are only resolved to a limited spherical harmonic degree of currently about 250–280 for the GOCE mission (Brockmann et al., 2013, Bruinsma et al., 2013). For CHAMP, the models are available up to d/o 150, which corresponds to less then 23 000 unknowns. For GRACE, current models are resolved to d/o 180 (less then 33 000 unknowns).

For the processing of the satellite-only models, the computational challenge does not occur from the number of parameters to be estimated, but from the huge number of observations (e.g. several hundreds of millions for GRACE or GOCE collected over several years) and the complex stochastic properties of the data (see e.g. Schuh et al., 2010, Rummel et al., 2011). To reduce computing time, the following examples of approximations and simplifications were introduced in current models published. For instance, Bruinsma et al. (2013) perform a down sampling of the 1 Hz GOCE gradiometer data to 0.25 Hz data to reduce the computing time within the estimation process of the spherical harmonic gravity field coefficients. In addition, a simple band-pass filter is used, which filters out signal outside the gradiometers measurement band. Details on the parallel processing concepts used are not provided. Migliaccio et al. (2011) process GOCE data in geographical patches which are assumed to be independent to handle the computational requirements in their collocation based gravity field determination. Correlations are reduced introducing overlapping borders. Schall et al. (2014) apply a down sampling filter to reduce the 1 Hz GOCE data to 0.2 Hz. Thus, only one fifth of the data is used in the NEQ assembly. To handle the stochastic model, the data is processed in short arcs of 15 min length, for which a full covariance matrix derived from empirically derived covariance functions can be set up. Correlations between the short arcs are modeled by an empirical bias parameter per short arc. The same processing strategy was proposed by Mayer-Gürr et al. (2005), Mayer-Gürr (2006), Mayer-Gürr et al. (2010a) for the analysis of CHAMP and GRACE observations. Due to the partioning into short arcs, the processing can be parallelized very well, as the NEQs can be assembled independently for every arc, thus a concept close to Sect. 3.2.2 can be used for parallel processing. Beutler et al. (2010) use an alternative approach with an simplified stochastic model, where only observation specific weights are used for the observations, correlations are neglected. The correlations are absorbed by many empirical parameters, which they call stochastic pulses. The parallel implementation concept is not discussed in detail. In addition, even for satellite only models with a limited parameter space, iterative solvers are often used to derive solutions (Baur, 2009, Xie, 2005, Farahani et al., 2013) e.g. for parameter tuning (Brockmann et al., 2010). The final model is then often set up via the assembly and solution of full NEQs (Pail and Plank, 2003, Bruinsma et al., 2013), to derive in addition the full error covariance matrix. All of the above mentioned approaches produce good solutions and the provided solutions affirm the strategies used. But especially for the approaches which use very simple stochastic models, it was shown that an external calibration of the formal error estimates is often required, to provided error estimates in a realistic order of magnitude.

Within this work, satellite only models are derived from the GOCE mission in Chap. 6. The work contributed to the official ESA models of the so-called time-wise gravity field models (EGM_TIM_RL01, EGM_TIM_RL02, EGM_TIM_RL03, EGM_TIM_RL04 and currently the preparation of the final EGM_TIM_RL05, Pail et al., 2010a, Schuh et al., 2010, Pail et al., 2011a,b, Brockmann et al., 2013). The goal is a flexible implementation, where simplifications to reduce computing time are avoided. Instead, the solution process should be implemented as a straightforward Gauss-Markoff model where all observations with a tailored observation noise model enter the solutions. A flexible environment is implemented where advanced processing models can be tested and used, independently of the required computational requirements. Compared to other solutions and approaches it could be shown, that especially the quality of the covariance matrix improves, such that the formal error estimates are meaningful. An empirical calibration of the derived covariance matrix becomes unnecessary.

An exception of satellite-only models with respect to the maximal resolution are the recently derived Lunar gravity field models. With the data available from the very low orbiting Gravity Recovery And Interior Laboratory (GRAIL) (Zuber et al., 2013) mission, lunar gravity field models are computed up to spherical harmonic d/o 660 (Lemoine et al., 2013, Konopliv et al., 2013) using rigorous solution techniques as well. The resolution is high, but due to the short mission lifetime of about six months (and a 5 s sampling) the amount of data being processed is small compared to the Earth's gravity field missions. Lemoine et al. (2013) as well as Konopliv et al. (2013) use QR factorization based solvers, without providing details on the parallel implementation concept. A possible out-of-core implementation of an updated QR decomposition is given in Gunter and Van De Geijn (2005) and the development of an in-core method in Baboulin et al. (2009).

The second class of models consists of satellite-only gravity field models, which combine models of the different satellites e.g., on the level of NEQs (Pail et al., 2010b, Bruinsma et al., 2013). As the resolution of those models is limited again by the resolution of the satellite models, the NEQs are quite small and not bigger then 40 GB. Thus, for this models, the computational challenges are limited (Pail et al., 2010b, Bruinsma et al., 2013) if preprocessed NEQs are used. Farahani et al. (2013) start on the level of OEQs. They use an iterative solver to process the data from the missions in a joint inversion. As Bruinsma et al. (2013) and Förste et al. (2011) use approximated stochastic models, they only combine subsets if the NEQs apply empirical derived weighting factors to obtain the final solution. Within this thesis, a flexible solver was implemented (cf. Chap. 7), which can be used to estimate gravity fields of this class. The developed software and methods contribute to the satellite-only models of the GOCO0xS series (Gravity Observation Combination Consortium, Pail et al., 2014), where a special case of the direct solution methods (i.e. $O = 0$) described in Chap. 7 is used. Until now, three generations of models were computed (GOCO01S, GOCO02S and GOCO03S Pail et al., 2010b, Goiginger et al., 2011, Mayer-Gürr et al., 2012) combining data from complementary satellite observations.

The third class is the class of high resolution combined gravity field models, which account in addition to the satellite observations (in form of their NEQs) also for terrestrial measurements like gravity anomalies over land and altimetry over the oceans. Using these measurements, which are sensitive for higher degrees, the resolution of the model can be considerably increased. Current combined models are available for d/o 360 (GIF48, Ries et al., 2011), 1 949 (EIGEN6C, Förste et al., 2008, 2012) and 2 190 (EGM2008, Pavlis et al., 2012) which corresponds to the number of parameters in the range of 130 000 to $4.8 \cdot 10^6$. Using altimetry, which measures the sum of the geoid and the Ocean's dynamic topography, a separation of both quantities has to be introduced.

Whereas the GIF48 model was computed via the assembly and solution of full NEQs (Ries et al., 2011), the higher degree models were computed introducing approximations and simplifications which reduce the computational requirements. For instance, EIGEN6C was computed via averaging two solutions, i.e. a full NEQ solution from d/o 2 to 370 and a block diagonal solution for the gridded high resolution data from 2 to degree 1949 (Förste et al., 2008, 2012). The final solution was composed from the full NEQ solution for degrees 2 to 260 and for degrees 370 to 1949 from the block diagonal solution. For degrees 260 to 370 the coefficients of both sets were averaged. EGM2008 was computed from an entire block diagonal solution. Also the involved satellite NEQs were approximated as a block diagonal form (Pavlis et al., 2012). Precise details are not provided[4]. In addition, parameters describing the mean dynamic ocean topography are estimated in an iterative procedure instead of in a joint estimation process. The approach requires a global equidistant data set, where all observations enter the solution with an equal weight. For instance marine data provided by Andersen and Knudsen (2009) is used in EGM2008. Regular grids which are an output of a collocation based interpolation of a-long track altimetry is made available for the mean sea surface and the marine geoid, which is then used in EGM2008 computation. Reguzzoni and Sansò (2012) suggest to estimate combined models via a combination of a full covariance matrix of the satellite data and a block diagonal matrix of the high resolution terrestrial data using an iterative solver. None of the mentioned approaches is able to use original along track sea surface heights measured by altimetry. They all require the altimetry data to be regularly gridded and to be reduced by a mean dynamic topography model. In addition, it is assumed that they are transformed to gravity anomalies. Correlations in the gridded data, resulting from the instruments and from the gridding process, are neglected.

Nevertheless, there are some studies, which demonstrate from a computational point of view, that the assembly and solution of full NEQs is possible in reasonable time up to d/o 600–720 (Fecher et al., 2011, Brockmann et al., 2014b). Within the Chapters 7 and 8 methods and implementations are derived to obtain rigorous least squares solutions up to degree 720 using the direct solution method and rigorous iterative least squares solver to derive solutions actually up to d/o 1440. Using this implementations, rigorous solutions for the scenarios described above become computable.

[4]Note that there exist many possibilities to approximate and recovery of a set of block diagonal normal equations from the full spherical harmonic solution vector and its covariance matrix.

A basic feature of the implementation is, that for instance altimetry observations can be used as along-track measurements with an along track error model. Compared to other applications (e.g. finite elements), as long as the data distribution is irregular and/or correlations between the observations exist, dense systems of equations are produced within global gravity field determination. The physical requirements (such as consistent data handling, data homogenization, data reduction, reference systems) and limitations (e.g. commonly used spherical approximation and required corrections, data availability) are not discussed within this thesis. A summary about introduced approximations and physical corrections typically required is provided in Gruber (2000, Chap. 1, Chap. 2)

5.2 Specific Adjustment Models for Gravity Field Recovery

Within global gravity field determination, the unknown parameters $\mathbf{x}$ are mainly composed of the spherical harmonic parameters c_{lm} and s_{lm}, describing the gravity field. They are estimated, depending on the type of the gravity field, from various (complementary) observation types in a least squares adjustment. The goal of the applications and solvers implemented in Chapters 6, 7 and 8 is to find the least squares solution for the unknown spherical harmonic parameters arranged in a vector $\mathbf{x}$. In general, it is assumed that several observation groups should be combined, which might be available as raw observations (observation equations — OEQs) which are either point-wise gravity measurements or along-track measurements of a satellite platform. In addition, (band-limited) NEQs already derived in an independent preprocessing step are integrable into the adjustment. A typical group are the NEQs of a gravity field satellite mission like CHAMP, GRACE or GOCE, where experts in the individual mission did the assembly of the model and provide the solution. If the solution vector and an unregularized full covariance matrix is provided, the NEQs can be reconstructed and used without loss of information in a further combined gravity field adjustment, adding additional data sources.

All three application specific implementations introduced here are special cases of (4.6b):

- Within Chap. 6 satellite-only models of the GOCE missions are computed. Different observation types are analyzed and combined to a derive the final solutions. As the satellite-to-satellite tracking data is separately analyzed by an other group, it enters the solution as a NEQ group. Two regularization matrices are introduced as NEQs, too, such that within the GOCE application holds $N = 3$. The gradiometer observations enter the solution process as OEQs. Calibrated gravity gradient measurements are used as input. As the observations of different gradient components and different periods of time have different characteristics, they are processed as several OEQ groups ($O \gg 1$, depending on the time period analyzed).

- Within Chap. 7, a general direct solver for (4.6b) is implemented. It is designed to combine NEQs preprocessed from the data of the individual satellites (SLR, GRACE and GOCE in the simulations) with high resolution observations. Within the performed simulations, this are either different groups of ground gravity measurements (gravity anomaly data sets) or different groups of along-track altimetry observations. These groups contain the higher degree gravity signal (more sensitive) and are compared to the NEQs groups responsible for the higher dimensional parameter space (hundreds of thousands of parameters).

- Within Chap. 8, a comparable scenario is analyzed, but with the assumption that the parameter space is even larger such that a solution of (4.6b) via the assembly and solution of full NEQs is unreasonable. As an alternative, to derive a in a computational sense rigorous least squares solution, an iterative solver is implemented in the HPC environment, which is able to handle a larger number of unknown parameters (millions of parameters).

For the three applications, the processing and implementation is described in detail. Some hints on the implementation of the observations equations are provided in the specific chapter, if required. Within all applications, the physical models are of minor interest. The focus is on the massive parallel implementation of the specific tasks in the HPC environment using the derived basic framework. Thus, the simulations in Chap. 7 and 8 are in a physical sense simplified but demonstrate the possibilities provided by the use of the proposed HPC concepts and the devolved framework.

5.3 Numbering Schemes for Gravity Field Determination

Especially when analyzing gravity data, the NEQs are set up only for the parameters, the original observations are sensitive for, to derive usable stand-alone satellite-only gravity field models. For example, combining a GOCE model (sensitive to at least d/o 250, 30 GB NEQ) with Satellite Laser Ranging (SLR, sensitive to d/o 5, e.g. Maier et al., 2012) the parameter space of both groups is significantly different. For the combination, it does not make sense to set up SLR NEQs up to d/o 250, instead, the band-limited NEQs have to be combined. There exists no "default" parameter order for the spherical harmonic coefficients which groups the parameters of more than two band-limited subsets in blocks. For such cases, the combination of matrices via block additions is impossible. Reordering of the satellite NEQs at runtime becomes mandatory. For that reason the general concept of symbolic numbering schemes and parameters is specialized to parameters required for spherical harmonic analysis and global gravity field determination (mainly spherical harmonic parameters).

5.3.1 Special Numbering Schemes

With respect to spherical harmonic gravity field parameters, tailored numbering schemes with different properties exist. Three special cases are named here, as their properties are used in some of the applications presented later on. An overview of available special numbering schemes and their properties is discussed in Schuh (1996, Chap. 2).

Order-wise Numbering Parameters are grouped in blocks of parameters of same spherical harmonic order. Fulfilling some prerequisites (gridded data), the NEQ matrix $\mathbf{N}$ would result – due to orthogonalities of the spherical harmonic base functions – in a block diagonal matrix (e.g. Colombo, 1981, Reguzzoni and Sansò, 2012). These prerequisites would be, that the observations are equidistant along the parallels with a constant accuracy. Although this prerequisites are heavily violated in the applications analyzed here, it can be expected that resulting NEQs are at least block diagonal dominant if the order-wise numbering scheme is chosen.

Degree-wise Numbering The spherical harmonic parameters are arranged per degree, i.e. parameters of the same degree are neighbors within the NEQs. This scheme is very flexible if the resulting model and its covariance matrix is used in a truncated version, as e.g. the covariance matrix of the truncated model is just a block sub-matrix of the complete matrix.

Free kite Numbering This is a special numbering scheme which was developed for the combination of regular and irregular distributed data, i.e. data producing block diagonal NEQs and lower resolution full NEQs. This numbering scheme is used as preconditioner in GOCE data processing with the GOCE tailored iterative solver (cf. Boxhammer, 2006, Boxhammer and Schuh, 2006).

Listing 5.1: Header file defining the main features of the class `Parameter`.

```
#ifndef PARAMETER_H
#define PARAMETER_H

using namespace std;

class Parameter
{
        public:
                //====================================================
                /* types of Parameters,
                 *      SH_C -> spherical harmonic cosine coefficient
                 *      SH_S -> spherical harmonic sine coefficient
                 *      FE   -> coefficient of finite element base functions
                 *      HELP -> group specific "help" Parameters like biases
                 *      OTHER-> wild card for some additional Parameters
                 *      UNDEFINED -> default value, not yet known Parameter type
                 */
                enum PARAMETER_TYPE { SH_C, SH_S, FE, HELP, OTHER, UNDEFINED };
                // Constructors
                Parameter( );
                Parameter( PARAMETER_TYPE t, size_t l, size_t m = 0 , size_t i = 0);
                Parameter( const Parameter & p );
                //====================================================
                // Destructor
                ~Parameter();
                //====================================================
                // comparison methods and operators
                bool operator==( const Parameter & p ) const;
                bool operator< ( const Parameter & p ) const;
                ...
                //====================================================
                // get and set attributes
                PARAMETER_TYPE type() const { return(_type);}
                ...
                void set( PARAMETER_TYPE t, size_t l, size_t m = 0 , size_t i = 0);
                //====================================================
                // other functions
                ...
        private:
                //====================================================
                // attributes
                PARAMETER_TYPE _type;
                /* specific Parameter information
                 * _l   -> degree of spherical harmonic coefficient (SH_C, SH_S)
                 *      -> node the intermeshing Parameter belongs to (FE)
                 *      -> ID of observation group the Parameter belongs to (HELP)
                 *      -> not used yet (OTHER)
                 *      -> 0 (UNDEFINED)
                 */
                size_t _l;
                /* _m   -> order of spherical harmonic coefficient (SH_C, SH_S)
                 *      -> type of FE parameter (0: constant, 1: trend, 2: annual period) (FE)
                 *      -> type of help Parameter, group specific (HELP)
                 *      -> not used yet (OTHER)
                 *      -> 0 (UNDEFINED)
                 */
                size_t _m;
                /* _i   -> temporary help variable (All types)
                 */
                size_t _i;
};
#endif // PARAMETER_H
```

5.3.2 Symbolic Numbering Schemes for Gravity Field Recovery

For the gravity data analyzed within this work, these parameters are mainly spherical harmonic parameters of different d/o, but might also be finite element parameters (not directly addressed) to estimate additional quantities as e.g., the mean dynamic ocean's topography (Becker et al., 2012, Becker, 2012, Becker et al., 2013) or some other parameters like group specific biases. A symbolic parameter is mapped into a class `parameter` as shown in Listing 5.1. Each parameter has a defined type (e.g. spherical harmonic, finite element, ...) and some additional attributes which store the specific properties (i.e. three unsigned integers) to define e.g. the degree l and order m of a spherical harmonic parameter. In addition, compare operators, like `operator==( const parameter & p2 )` and `operator<( const parameter & p2 )`, are defined there.

Arranging a sequence of these parameters in a field, e.g. in a `std::vector<Parameter>`, defines a symbolic numbering scheme which can be associated with global matrices (their rows and/or columns). The column and/or row i of a matrix corresponds to the parameter $\mathbf{p}(i)$ using the

Listing 5.2: Header file defining the main features of the class `NumberingScheme`.

```cpp
#ifndef NUMBERINGSCHEME_H
#define NUMBERINGSCHEME_H

using namespace std;

class NumberingScheme
{
        public:
                // Constructors
                NumberingScheme( );
                NumberingScheme( string schemeFile );
                NumberingScheme( const NumberingScheme & orig );
                NumberingScheme( size_t size );
                // Destructor
                ~NumberingScheme();
                // Operators
                NumberingScheme & operator=( const NumberingScheme &orig );
                bool operator==( const NumberingScheme &orig );
                // other methods
                size_t findIdx( const Parameter & p ) const;
                size_t size( ) const;
                vector<size_t> permutationVec( const NumberingScheme &ns2 ) const;
                void readFromFile( string schemeFile );
                void extendToMaxDegree( int lmaxi );
                void trimToMaxDegree( size_t lmax );
                void extendToMinDegree( int lmini );
                void fillDegreeWiseAlt( int fromDeg, int toDeg );
                void fillOrderWiseAlt( int fromDeg, int toDeg );
                void writeSymbolicScheme( string fn );
                void sort();
                // accesses parameters
                const Parameter & p( size_t i ) const {return(_p.at(i));}
                Parameter & p( size_t i ){return(_p.at(i));}
                inline int nrParam() const { return( _p.size() );}
        private:
                vector<Parameter> _p;
};
#endif // NUMBERINGSCHEME_H
```

specialized class `NumberingScheme` as shown in Listing 5.2. This special implementation of the class `Parameter` and of the class `NumberingScheme` can be directly used for the presented reordering procedures in Sect. 4.3.

5.4 Analyzing Gravity Field Models

This section is used to summarize some quantities, which are used to analyze and compare global gravity field models derived within the application chapters. Typical quantities, which are applied to visualize the results are introduced. All quantities shown rely on the comparison of different models available as a series of spherical harmonic coefficients or on the quality of the full covariance matrix of the estimated model.

5.4.1 Spectral Domain: Degree (Error) Variances

A simple one-dimensional characterization of gravity field models and their quality are the degree (error) variances. Signal degree variances are the quadratic sum over all orders of that degree (e.g. Heiskanen and Moritz, 1993, p. 259), i.e.

$$\sigma(l)^2 = \sum_{m=0}^{l} c_{lm}^2 + s_{lm}^2 \tag{5.2}$$

and are a kind of power spectral density. For the case of GOCE, the near zonal coefficients are badly determined. An attempt to define a measure which is less sensitive to large errors in the polar regions, the zonals are often neglected in the computation[5],

$$\bar{\sigma}(l)^2 = \sum_{m=0}^{l} \begin{cases} c_{lm}^2 + s_{lm}^2 & \text{if } m > \theta_0 l \text{ cf. (6.17)} \\ 0 & \text{otherwise} \end{cases} . \tag{5.3}$$

The same quantities can be derived for the error estimates of the model. Using the formal errors (i.e. variances) of the coefficients derived from the inverse NEQs, the degree error variances are

$$\sigma_\sigma(l)^2 = \sum_{m=0}^{l} \left(\sigma_{c_{lm}}^2 + \sigma_{s_{lm}}^2\right), \tag{5.4a}$$

$$\bar{\sigma}_\sigma(l)^2 = \sum_{m=0}^{l} \begin{cases} \sigma_{c_{lm}}^2 + \sigma_{s_{lm}}^2 & \text{if } m > \theta_0 l \text{ cf. (6.17)} \\ 0 & \text{otherwise} \end{cases} . \tag{5.4b}$$

These are estimates for the power of the error at spherical harmonic degree l. The degree error variances can also be computed empirically, compared to a reference model with coefficients $\hat{c}_{lm}$ and $\hat{s}_{lm}$ via

$$\sigma_\Delta(l)^2 = \sum_{m=0}^{l} \left((c_{lm} - \hat{c}_{lm})^2 + (s_{lm} - \hat{s}_{lm})^2 \right) \tag{5.5a}$$

$$\bar{\sigma}_\Delta(l)^2 = \sum_{m=0}^{l} \begin{cases} (c_{lm} - \hat{c}_{lm})^2 + (s_{lm} - \hat{s}_{lm})^2 & \text{if } m > \theta_0 l \text{ cf. (6.17)} \\ 0 & \text{otherwise} \end{cases} . \tag{5.5b}$$

All degree variances can be computed in terms of geoid heights, multiplying the quantities with the factor R^2, where R is the mean earth radius using the spherical approximation.

5.4.2 Space Domain

Using the spherical harmonic coefficients, functionals of the gravity field can be computed in the spatial domain. Illustrating and analyzing the differences of the functionals between two models is an easy but efficient validation tool (having in mind that both models are not error free).

5.4.2.1 Potential, Geoid Heights and Gravity Anomalies

The potential can be computed from the coefficients at a point (λ, φ, r) via (5.1). For geoid heights i.e. approximately using Bruns formula (e.g. Heiskanen and Moritz, 1993, p. 85)

$$N(r, \theta, \lambda) = \frac{V(r, \theta, \lambda) - U_0(r, \theta, \lambda)}{\gamma}, \tag{5.6}$$

with the normal gravity γ and the rotation-symmetric normal potential $U_0(r, \theta, \lambda)$ referring to a reference ellipsoid. For gravity field anomalies i.e. in spherical approximation (e.g. Heiskanen and Moritz, 1993, p. 89)

$$\Delta g(r, \theta, \lambda) = \frac{GM}{r^2} \sum_{l=0}^{l_{\max}} (l-1) \left(\frac{a}{r}\right)^l \sum_{m=0}^{l} \left(c_{lm} \cos(m\lambda) + s_{lm} \sin(m\lambda)\right) P_{lm}(cos\theta) . \tag{5.7}$$

[5]For details see Sect. 6.2.3.

5.4.2.2 Error Propagation

With the covariance matrix of the spherical harmonic coefficients $\mathbf{\Sigma}_{\mathbf{xx}}$ available, the error propagation from the error of the coefficients to the error of a gravity field functional f is straightforward. Setting up the functional matrix $\mathbf{F}_f$ for a point grid $(\lambda_i, \theta_i, r_i)$ and a functional using the linear equations in Sect. 5.4.2.1, the error propagation is (e.g. Koch, 1999, p. 99)

$$\mathbf{\Sigma}_{ff} = \mathbf{F}_f \mathbf{\Sigma}_{\mathbf{xx}} \mathbf{F}_f^T, \tag{5.8}$$

applying the standard concept of variance propagation to derive the full covariance matrix of the gravity field functional of the predefined point grid.

5.4.3 Contribution of Observation Groups to Estimates of Single Coefficients

To measure the contribution of one of the observation groups i to the finally combined solution, the partial redundancies are (often) used (e.q. Koch, 2007, p. 146). Within this context, the estimated parameters $\mathbf{x}_i$, determined from a single observation group i, can be seen as pseudo observations with the full covariance matrix $\mathbf{\Sigma}_{i,\mathbf{xx}} = (w_i \mathbf{N}_i)^{-1}$.

The contribution is measured via partial redundancies

$$\mathbf{W}_i = \mathbf{\Sigma}_{\mathbf{xx}} w_i \mathbf{N}_i = \mathbf{N}^{-1} w_i \mathbf{N}_i, \tag{5.9}$$

where the j-th element of the diagonal part $\operatorname{diag}(\mathbf{W}_i)$ describes the contribution of group i to the final solution. It can be interpreted as the percentage, to which extend the parameter $\mathbf{x}_j$ is determined by group i. To determine this contribution, the full NEQs respectively covariance matrices are used. The sum over all contribution matrices $\mathbf{W}_i$ ends up exactly in the identity matrix, as

$$\sum_{i=1}^{I} \mathbf{W}_i = \sum_{i=1}^{I} \mathbf{\Sigma}_{\mathbf{xx}} w_i \mathbf{N}_i, \tag{5.10a}$$

$$= \sum_{i=1}^{I} \mathbf{N}^{-1} w_i \mathbf{N}_i, \tag{5.10b}$$

$$= \mathbf{N}^{-1} \sum_{i=1}^{I} w_i \mathbf{N}_i, \tag{5.10c}$$

where $\sum_{i=1}^{I} w_i \mathbf{N}_i$ is the definition of $\mathbf{N}$. It follows that

$$\sum_{i=1}^{I} \mathbf{W}_i = \mathbf{N}^{-1} \sum_{i=1}^{I} w_i \mathbf{N}_i = \mathbf{N}^{-1} \mathbf{N} = \mathbf{I}. \tag{5.11}$$

It has to be mentioned that, due to correlations in $\mathbf{N}_i$, the results have to be carefully interpreted. The results of the analysis provide a first idea. In contrast to partial redundancies r_i, where it is guaranteed that $0 \leq r_i \leq 1.0$, the only restriction is that $\sum_i \mathbf{W}_i(j,j) = 1.0$. As the parameters are coupled due to correlations, individual contributions $\mathbf{W}_i(j,j)$ might get negative or a value above 1.0. For such cases, the results are hardly interpretable. Nevertheless, the analysis provide some hints about the contributions of groups to individual parameters or parameter sub-spaces. Note that the matrix $\mathbf{W}_i$ corresponds to the matrix in the trace operator used in Sect. 4.2 to determine the total number of parameters Υ_i determined by group i (e.g. in (4.9)).

6. Application: Gravity Field Determination from Observations of the GOCE Mission

6.1 Introduction to the GOCE Mission

The goal of the GOCE (Gravity field and steady-state Ocean Circulation Explorer) satellite mission is the modeling of the time-invariant component of the Earth's gravity field to the accuracy of at least 1 mGal for gravity anomalies and $1-2$ cm for the geoid at a spatial resolution of at least 100 km (cf. ESA, 1999, Floberghagen et al., 2011). The model for the global gravity field typically consists of a finite series of globally defined spherical harmonics (see e.g. Heiskanen and Moritz, 1993, p. 57ff). From a mathematical point of view, the problem leads to an extremely high dimensional, inverse, and ill-conditioned data-fitting problem, where 60 000–80 000 unknown coefficients of the basis functions have to be estimated from hundreds of millions GPS satellite-to-satellite tracking (SST) and satellite gravity gradiometry (SGG) observations. As the system of equations is overdetermined, the final set of parameters is determined as the least squares solution resulting from a linear Gauss-Markoff model. The numerical assembly and solution of the combined normal equations resulting from this problem is further complicated by the strong data correlations within the gravity gradiometry observations as well as by the different weighting of the three complementary data types (SST, SGG and regularizing prior information (REG)).

As mentioned above, two basic observation groups are measured by GOCE and used to derive the Earth's gravity field. The first group are code and phase GPS (Global Positioning System) observations collected by the onboard GPS receiver. Within this context, they are used as preprocessed kinematic orbit positions of the satellite, as they result from a kinematic orbit determination (cf. Bock et al., 2011, 2014). As the gravitational attraction affects the satellite, the orbit of the satellite (its position derived with GPS) can be linked to the gravity field using Newton's equation of motion (e.g. Seeber, 2003, p. 66). The second group, which is in the main focus within this thesis, are the observations of the three-axis gradiometer, which measures the second derivative of the potential (5.1) in a gradiometer fixed reference frame (GRF, cf. EGG-C, 2010a, Chap. 8.2). Differences of accelerations are measured along a 0.5 m baseline. In general, all six elements of the symmetric gravity tensor are measured. Nevertheless, due to design constraints, only four of the the six non redundant components can be measured with a high precision (see e.g. Rummel et al., 2011, Pail et al., 2011a, Floberghagen et al., 2011).

GOCE orbited the Earth in the mean altitude of 255 km[6] from 2009 to November 2013. The orbit was chosen sun synchronous, which results in polar gaps with an opening angle of about 6.5°. The poles are not crossed by the satellite during the whole mission lifetime (cf. Fig. 6.1). Due to the data gaps and the fact that the gravity field signal is damped at satellite altitude, determining the gravity field parameters from the noisy and correlated observations yields an inverse ill-posed problem. As gravimetric problems generally tend to be ill-posed, (e.g. Schwintzer et al., 1997, van Gelderen and Koop, 1997, Ilk et al., 2002, Hofmann-Wellenhof and Moritz, 2005), a kind of regularization can be applied, which can be seen as a priori knowledge in Bayesian sense (e.g. Koch, 2007, p. 151). It is optionally introduced as a third observation group, to derive smoother and more stable estimates for spherical harmonic parameters. Especially for GOCE-only models regularization is required, if the final spherical harmonic model is cut off and not evaluated in the synthesis to the provided maximal resolution. Nevertheless an unconstrained solution without regularization is computable.

[6]The satellites altitude was reduced in several steps down to 225 km at the end of the mission to improve the signal to noise ratio for a measurement period of at least 8 months.

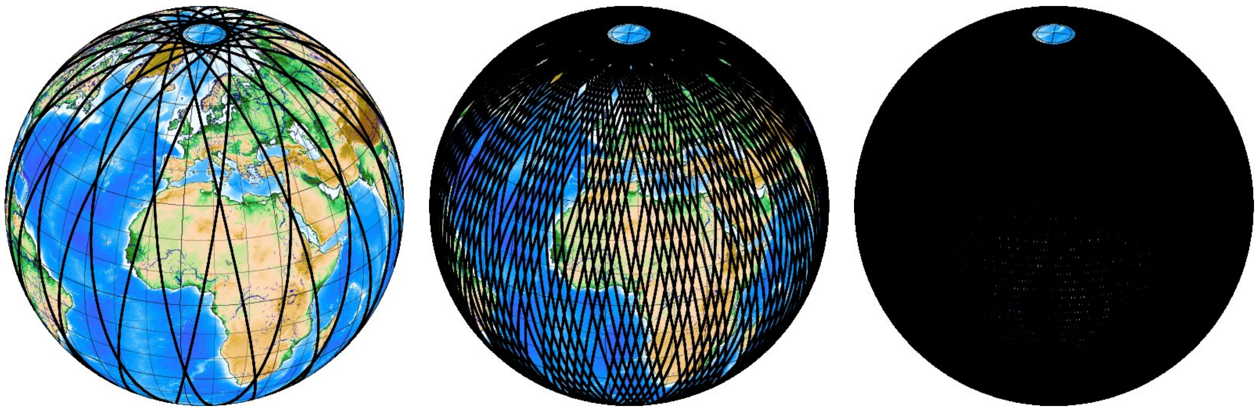

(a) Coverage after one day. (b) Coverage after one week. (c) Coverage after one month.

Figure 6.1: Coverage of the GOCE satellites ground track on the Earth's surface after one day, one week and one month.

The goal of this Chapter is to develop a massive parallel software implementation which is used to determine the unknown spherical harmonic coefficients from the observations collected by the GOCE satellite. The two observation groups, high-low satellite-to-satellite tracking (SST) and the gravity gradient measurements (SGG) are merged and stabilized by a third, i.e. a regularization group. The unknown parameters are then derived in a joint weighted least squares adjustment. From a computational point of view, the challenges arise from the huge numbers of highly correlated observations, which are more than $430 \cdot 10^6$. Compared to the other applications in Chap. 7 and 8, the number of unknowns is small. Depending on the data volume used, current and expected GOCE based gravity field models are resolved for spherical harmonics of d/o 250–280, which corresponds to less than 80 000 parameters (Brockmann et al., 2013, Bruinsma et al., 2013). Nevertheless, HPC is needed to derive a rigorous solution and a full covariance matrix in reasonable time. For the GOCE missions, this can be done using tailored iterative solvers (as developed by Schuh, 1996, Boxhammer, 2006) or via the assembly and solution of full NEQs (e.g. Plank, 2004, Pail and Plank, 2002). The computational demanding tasks are the decorrelation of the observation equations (cf. Sect. 6.2.2.2) and afterwards the assembly of the SGG normal equations, i.e. the computation of $\mathbf{A}^T\mathbf{A}$ for hundreds of millions of observations. The goal of this section is to implement a massive parallel flexible software package to assemble the full normal equations from the GOCE gradiometry observations, combine them with high-low SST measurements and solve for regularized (i.e. stabilized) spherical harmonic parameters describing the static part of the Earth's gravity field. In addition to an already existing iterative solver, which is mainly used for the purpose of parameter tuning (cf. Brockmann et al., 2010), a more flexible software package is derived. The additional features of the solver are for instance:

- The full covariance matrix of the solution can be determined.
- The decorrelation is implemented more general, alternative noise models can be used (e.g. decorrelation filters changing in time).
- The estimation of weights is independently possible for gradient components and time periods of the time series.
- The off diagonal components can be used in the analysis.
- An arbitrary matrix can be used as regularization matrix.
- Various components for model analysis are available.
- Outlier flagging is possible individually per gravity gradient component.

Although, the number of parameters is quite small, the size of the resulting NEQs is $30 - 46$ GB and thus too large for single computers.

The implemented software module is based on the developed class of the block-cyclic distributed matrices introduced in Chap. 3.3. They are used to directly set up the observation equations, apply the decorrelation, assemble the NEQs and to solve for the parameters and group specific weighting factors. As the functional and stochastic model of the so called time-wise approach (Sünkel et al., 1995, Pail and Plank, 2002, Pail et al., 2006, 2011a), which is used within the official ESA release, is quite well known, the method will only be briefly described. Instead, the focus is put to the implementation and the computational challenges. Finally, some results are given for the four official time-wise gravity field models which were computed with a significant contribution by the developed software. Details are given especially for the most recent model computed, which is the fourth release, i.e. EGM_TIM_RL04 in Sect. 6.5. An outlook towards the final model which will be EGM_TIM_RL05 using all mission data is given.

Although the processing strategy is fixed to the time-wise method (constraints within a research project), alternatives exist and are applied for GOCE data analysis. Rummel et al. (1993) generally distinguish between two general approaches, processing strategies which follow a time-wise approach[7] and the strategies which follow a space-wise approach.

The space-wise methods transform the observations on regular grids, from which the spherical harmonic coefficients can be easily recovered (e.g. Sansò and Tscherning, 2003, Migliaccio et al., 2011). Strategies which follow the time-wise approach interprete the measurements as time series along the satellites orbit. If for instance the observations are equidistant in time and the orbit is an exact repeat orbit, semi-analytic time-wise approaches exist which are numerically very efficient as they are solved using a fast Fourier transform (see e.g. Sneeuw, 2000, Pail and Wermuth, 2003, Mayrhofer et al., 2010). Other strategies which follow a time-wise approach process the data in-situ, the observation equations are set up at the exact observation location and orientation of the satellite (e.g. Bruinsma et al., 2013, Schall et al., 2014, Pail et al., 2011a). Transformations of the measurements are avoided. Although all mentioned studies follow a time-wise approach, the strategies slightly differ, mainly in the stochastic model used in the adjustment. As a kind of compromise of both approaches Ditmar et al. (2003a,b), Farahani et al. (2013) suggest to set up the observations equations efficiently for a 3D spherical grid and derive them for the exact observation location via interpolation.

6.2 The Physical, Mathematical and Stochastic Problem

The goal in this chapter is to derive the unknown spherical harmonic gravity field parameters from the observations of the satellite mission GOCE. For the case of GOCE, the general NEQs (4.6b) can be rewritten as

$$\left(\frac{1}{\sigma^2_{\text{sst}}}\mathbf{N}_{\text{sst}} + \sum_r \frac{1}{\sigma^2_r}\mathbf{N}_r + \sum_g \sum_{s=0}^{S-1} \frac{1}{\sigma^2_{g,s}} \mathbf{A}^T_{g,s}\mathbf{Q}^{-1}_{\boldsymbol{\ell}_{g,s}\boldsymbol{\ell}_{g,s}}\mathbf{A}_{g,s}\right)\mathbf{x}$$
$$= \frac{1}{\sigma^2_{\text{sst}}}\mathbf{n}_{\text{sst}} + \sum_r \frac{1}{\sigma^2_r}\mathbf{n}_r + \sum_g \sum_{s=0}^{S-1} \frac{1}{\sigma^2_{g,s}} \mathbf{A}^T_{g,s}\mathbf{Q}^{-1}_{\boldsymbol{\ell}_{g,s}\boldsymbol{\ell}_{g,s}}\boldsymbol{\ell}_{g,s}, \tag{6.1}$$

where $\mathbf{N}_{\text{sst}}$ and $\mathbf{n}_{\text{sst}}$ are the preprocessed normal equations of the SST observation group, $\mathbf{N}_r$ and $\mathbf{n}_r$ different normal equations which contain the regularizing prior information and $\mathbf{A}_{g,s}$ the SGG

[7]The approach followed here is named time-wise approach, too. This should be seen as an official name of the strategy and product (within HPF) and should not be mixed with the general concept (cf. Rummel et al., 1993).

design matrix for tensor component g and data segment s, $\mathbf{Q}_{\boldsymbol{\ell}_{g,s}\boldsymbol{\ell}_{g,s}}$ the cofactor matrix of the gravity gradient observations and $\boldsymbol{\ell}_{g,s}$ the vector of the observations itself. For the SGG observations, the observation time-series is divided into segments $s \in \{0, \dots, S-1\}$ without gaps for which individual NEQs are assembled and individual weights are determined. Different tensor components and segments are assumed to be uncorrelated. In addition to the division into segments, individual NEQs are set up for the individual tensor components $g \in \{XX, XY, XZ, YY, YZ, ZZ\}$ for every segment s. $1/\sigma_*^2 = w_*$ are the unknown weights for the involved groups to be estimated in addition to the unknown spherical harmonic parameters $\mathbf{x}$.

6.2.1 SST Processing

The processing of the SST observations is not the focal point of this work. Instead, SST observations are used as preprocessed normal equations. Nevertheless, the SST processing which is performed by project partners (ITSG, TU Graz) is shortly summarized and required references are provided. As within this work the results obtained by the short arc integral equation approach as used in Mayer-Gürr et al. (2005), Mayer-Gürr (2006) and Mayer-Gürr et al. (2010a) for CHAMP and GRACE are used, only this method is summarized[8]. Comparisons of alternative methods can be found in Mayer-Gürr (2006), Reubelt (2009), Löcher (2010), Reubelt et al. (2012) and especially for GOCE real data analysis in Baur et al. (2014). The original method was proposed by Schneider (1967) for orbit determination and refined for gravity field determination by Reigber (1969).

The satellites position $\mathbf{r}(t)$ in the inertial reference system can be linked to the Earth's potential via the Newton Euler equation of motion (e.g. Seeber, 2003, p. 66)

$$\ddot{\mathbf{r}}(t) = \frac{1}{m}\mathbf{K}(t) = \frac{1}{m}\sum_i \mathbf{K}_i(t). \tag{6.2}$$

$\ddot{\mathbf{r}}(t)$ is the satellites acceleration, m its mass and $\mathbf{K}(t)$ the sum of all forces acting on the satellite. One of the forces $\mathbf{K}_i(t)$ is the (static) gravitational force which is the gradient of the potential given in (5.1),

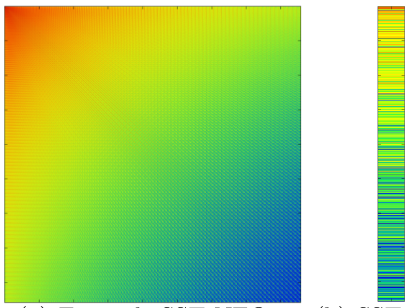
$$\frac{1}{m}\mathbf{K}_{\mathrm{gf}} = \mathbf{g}_{\mathrm{gf}}(\lambda, \varphi, r) = \nabla V(\lambda, \varphi, r) = \begin{bmatrix} \partial V/\partial x \\ \partial V/\partial y \\ \partial V/\partial z \end{bmatrix}. \tag{6.3}$$

Thus, integrating (6.2) twice (e.g. following the summary in Mayer-Gürr, 2006, p. 21f) yields an integral relation of the satellites position $\mathbf{r}(t)$ to the spherical harmonic gravity field parameters and thus the observation equations for high low satellite to satellite tracking

$$\mathbf{r}(t) = \mathbf{r}_A + \dot{\mathbf{r}}_A(t - t_A) + \frac{1}{m}\int_{t_A}^{t}\int_{t_A}^{t'} \sum_i \mathbf{K}_i(t'')\, dt''dt' + \int_{t_A}^{t}\int_{t_A}^{t'} \mathbf{g}_{\mathrm{gf}}(c_{lm}, s_{lm})\, dt''dt'. \tag{6.4}$$

$\mathbf{r}_A$ and $\dot{\mathbf{r}}_A$ are the unknown integration constants which are initial position and velocity at time t_A. These values can be co-estimated in the least squares adjustment and are then eliminated from the system of normal equations. All other forces $\mathbf{K}_i$ acting on the satellite have to be reduced by models, accelerometer measurements or co-estimated. One possible approach is described in Mayer-Gürr (2006, chapter 3.2). The background standards for GOCE data processing (cf. EGG-C, 2010a) are taken into account.

[8]Note that in the older releases (01-03) of the time-wise models the energy balance method (e.g. Gerlach et al., 2003) was used (e.g. Pail et al., 2011a).


(a) Example SST NEQ (b) SST RHS

```
# parameters used to assemble the NEQ
pathToN              = ./N_tug_0002.gdk
pathTon              = ./n_tug_0002.gdk
lTPl                 = 2.67835565709962e+11
numObsUsed           = 40383471
name                 = GO_SST_SHORTARC_MOP4
tide_system          = tide_free
GM_used              = 3.9860044150e+14
a_used               = 6378136.3
numberingSchemeNeq   = ./parameter_0002.ns
ltlReducedModel      = ~/data/earth/grs80.gfc
```

(c) NEQ info file

```
maxdeg 130
maxord 130
nrpara 17157
C 2 0
C 2 1
S 2 1
C 2 2
S 2 2
...
C 130 130
S 130 130
```

(d) Numbering scheme

Figure 6.2: Data belonging to the SST normal equation set.

The stochastic modeling of the (pseudo) observation errors, which are the errors of the satellites positions in this case, is essential for the combination with complementary data in (6.1). For the short arc integral equation approach, Mayer-Gürr (2006) and Mayer-Gürr et al. (2010a) developed a method which accounts for an approximative (i.e. banded) covariance matrix of the position resulting from the orbit determination geometry as well as for correlations in time via empirically estimated covariance functions. To handle the estimated covariance matrices, the 1 Hz observations are divided into short orbit arcs of 600 s length, such that the full covariance matrix for every arc can be set up locally as a full matrix. Lower frequency errors are absorbed by empirical bias parameters per arc.

The result of this analysis are normal equations, which are set up for the spherical harmonic coefficients from degree 2 up to a maximal degree l_{max}. As the resolution of high-low SST is limited to at most d/o 150, the matrices are quite small and – from a computational point of view – relatively easy to handle. In addition to the normal equations, i.e. the normal matrix $\mathbf{N}_{\text{sst}}$ and the right hand side $\mathbf{n}_{\text{sst}}$ (RHS), additional information is needed for the combination according to (6.1). A summary of an exemplary data set is given in Fig. 6.2. The additional information is the number of observations M_{sst} used in the assembly of the NEQs and the result of the product $\lambda_{\text{sst}} := \boldsymbol{\ell}_{\text{sst}}^T \mathbf{P}_{\text{sst}} \boldsymbol{\ell}_{\text{sst}}$ which are required for the estimation of variance components. Furthermore, physical constants like GM, a and the tide system used in the processing have to be known to guarantee consistency. Last but not least, the parameter order in the system of NEQs has to be provided with already assembled normal equations. This is done via an associated symbolic numbering scheme cf. Sect. 4.3, which in addition provides the maximal resolution. The information summarized in Fig. 6.2 is produced by project partners and is handled as sufficient statistics of the original observations within this work.

6.2.2 SGG Processing

This section summarizes the SGG processing of the so called time-wise approach (Sünkel et al., 1995, Pail and Plank, 2002, Pail et al., 2006, 2011a). As the observation equations and the processing details are straightforward and well documented, the details can be found in the given references. As this work focuses on the implementation of a direct solution method in a HPC environment and the application to real data analysis, this paragraph is only a short summary of the general processing ideas.

6.2.2.1 Functional and Physical Model

The GOCE 3-axis gradiometer measures acceleration differences along a very short baseline (0.5 m Floberghagen et al., 2011). The differences in the accelerations correspond to the derivative of the gravitational force and thus to the second derivative of the potential in the gradiometer reference frame (GRF), defined by the three ideal orthogonal gradiometer arms (cf. EGG-C, 2010a, Chap. 8.2),

$$\mathbf{T}\left(\mathbf{r}_{\mathrm{GRF}(t)}\right) = \nabla\nabla V = \nabla\mathbf{g} = \begin{bmatrix} V_{X_{\mathrm{GRF}}X_{\mathrm{GRF}}} & V_{X_{\mathrm{GRF}}Y_{\mathrm{GRF}}} & V_{X_{\mathrm{GRF}}Z_{\mathrm{GRF}}} \\ & V_{Y_{\mathrm{GRF}}Y_{\mathrm{GRF}}} & V_{Y_{\mathrm{GRF}}Z_{\mathrm{GRF}}} \\ \text{sym.} & & V_{Z_{\mathrm{GRF}}Z_{\mathrm{GRF}}} \end{bmatrix} \tag{6.5a}$$

$$= \begin{bmatrix} V_{XX} & V_{YY} & V_{XZ} \\ & V_{YY} & V_{YZ} \\ \text{sym.} & & V_{ZZ} \end{bmatrix}. \tag{6.5b}$$

This measurements have to be calibrated and reduced by the centrifugal term, which is assumed to be already done for the data used within this work. For the calibration and the Level 1B preprocessing of the gradiometer measurements see e.g. Cesare and Catastini (2008), Stummer et al. (2011, 2012), Frommknecht et al. (2011) and Stummer (2013). It is avoided to transform the gravity gradients into the earth fixed reference frame (EFRF), e.g. the International Terrestrial Reference Frame (ITRF, cf. EGG-C, 2010a, Chap. 4), where (5.1) is valid, as it would yield a mixture of the accurate and inaccurate gradients (e.g. Fuchs and Bouman, 2011). Due to the technical design, the diagonal of (6.5a) and $V_{X_{\mathrm{GRF}}Z_{\mathrm{GRF}}}$ can be measured with high precision within the measurement band width (MBW) of $5 \cdot 10^{-3}$ Hz to 0.1 Hz ($10 - 20$ mE/$\sqrt{\mathrm{Hz}}$, Rummel et al., 2011). $V_{X_{\mathrm{GRF}}Y_{\mathrm{GRF}}}$ and $V_{Z_{\mathrm{GRF}}Z_{\mathrm{GRF}}}$ are about a factor of 30 to 70 times less accurate (Rummel et al., 2011). To avoid the transformation of the observed quantities, the observation equations can be transformed instead. They can be set up in the GRF, the SGG observations are measured in. Thus, the second derivative of (5.1), which is valid in the EFRF, has to be transformed, i.e. rotated, to the GRF to set up the observation equations there. As described in Hausleitner (1995, Sect. 4) this can be done in a stepwise approach for every observation i measured at time t:

1. The observation equations of the second derivative of the potential can be directly written as a linear combination of the second derivatives of (5.1) (Hausleitner, 1995, Chap. 4.1), according to λ and φ in a local north oriented Cartesian frame, (LNOF, cf. EGG-C, 2010b, p. 23) which is located in the satellites center of mass: $\Rightarrow \mathbf{A}_t^{\mathrm{LNOF}}$.
2. Rotate the design matrix $\mathbf{A}_t^{\mathrm{LNOF}}$ from the LNOF to the inertial frame (IRF, cf. EGG-C, 2010b, p. 23) accounting for the Earth's rotation allying $\mathbf{R}(t)^{\mathrm{LNOF2IRF}}$: $\Rightarrow \mathbf{A}_t^{\mathrm{IRF}}$.
3. Rotate the design matrix $\mathbf{A}_t^{\mathrm{IRF}}$ from the inertial frame to the gradiometer reference frame (cf. EGG-C, 2010a, p. 22). The rotation matrix $\mathbf{R}^{\mathrm{IRF2GRF}}$ is determined by star sensor observations, i.e. the satellites orientation in space. It is available as the GOCE EGG_IAQ product (cf. EGG-C, 2009, Sect. 6.1.8): $\Rightarrow \mathbf{A}_t^{\mathrm{GRF}}$.

The basic idea of the time-wise method is to handle the gradiometer observations along the satellites orbit as an equidistant time series for each individual tensor component g. For that reason, the entire observation time series is grouped into the gapless segments s and arranged in observation vectors $\boldsymbol{\ell}_{g,s}$ for each tensor component g. All observations in a single vector are equidistant in time (1 s for the real data) and sorted chronological.

Whereas there are high correlations within the observation time series of a single component V_g arranged in the observation vector $\boldsymbol{\ell}_{g,s}$, it is assumed and empirically verified (Krasbutter and Schuh, 2010) that there are no significant correlations in the noise of the observation time-series of two different tensor components g_1 and g_2,

$$\boldsymbol{\Sigma}_{\boldsymbol{\ell}_{g_1,s},\boldsymbol{\ell}_{g_2,s}} = \mathbf{0}, \text{ for } g_1 \neq g_2 \text{ and } g_1, g_2 \in G = \{XX,\, XY,\, XZ,\, YY,\, YZ,\, ZZ\} \tag{6.6}$$

and that there are no correlations between observations before and after a data gap, i.e.

$$\boldsymbol{\Sigma}_{\boldsymbol{\ell}_{g,s_1},\boldsymbol{\ell}_{g,s_2}} = \mathbf{0}, \text{ for } s_1 \neq s_2 \text{ and } s_1, s_2 \in S = \{0, \ldots, S-1\}. \tag{6.7}$$

Thus, the combined normal equation can be written as in (6.1). The partial NEQs for an individual tensor component g and a gapless segment s

$$\mathbf{N}_{g,s} = \frac{1}{\sigma_{g,s}^2}\mathbf{A}_{g,s}^T\mathbf{Q}_{\boldsymbol{\ell}_{g,s}\boldsymbol{\ell}_{g,s}}^{-1}\mathbf{A}_{g,s}, \qquad \mathbf{n}_{g,s} = \frac{1}{\sigma_{g,s}^2}\mathbf{A}_{g,s}^T\mathbf{Q}_{\boldsymbol{\ell}_{g,s}\boldsymbol{\ell}_{g,s}}^{-1}\boldsymbol{\ell}_{g,s} \tag{6.8}$$

can be separately assembled. The essential part of the time-wise method is the modeling of the observation error and the correlations within the observations $\boldsymbol{\ell}_{g,s}$ as a data-adaptive estimation/approximation model of $\mathbf{Q}_{\boldsymbol{\ell}_{g,s}\boldsymbol{\ell}_{g,s}}$ is derived. As this is an essential part, yielding additional computational challenges for the implementation, the details of the stochastic modeling of $\boldsymbol{\ell}_{g,s}$ are given in the next section.

6.2.2.2 Stochastic Model

The observation vector of a single segment s can consist of millions of observations, it is impossible to model the error characteristics of the observations with a full covariance (or cofactor) matrix. The memory requirements would result in hundreds to thousands TB. Instead, an alternative strategy to model the observation cofactor matrix was developed by Schuh (1996) and extended by Schuh (2002, 2003), Schuh et al. (2006) and Siemes (2008). Instead of setting up a covariance (or cofactor) matrix, digital cascaded Auto-Regressive Moving Average (ARMA) filters were used as decorrelation or whitening filters. The original problem (for a single tensor component g and a single data segment s) in terms of the observation equations

$$\boldsymbol{\ell}_{g,s} + \mathbf{v}_{g,s} = \mathbf{A}_{g,s}\mathbf{x} \qquad \boldsymbol{\Sigma}_{\boldsymbol{\ell}_{g,s}\boldsymbol{\ell}_{g,s}} = \sigma_{g,s}^2\mathbf{Q}_{\boldsymbol{\ell}_{g,s}\boldsymbol{\ell}_{g,s}} \tag{6.9}$$

is transformed to a simpler problem applying a linear (cascaded) digital filter $\boldsymbol{\mathcal{F}}$ to the observation equations

$$\boldsymbol{\mathcal{F}}\boldsymbol{\ell}_{g,s} + \boldsymbol{\mathcal{F}}\mathbf{v}_{g,s} = \boldsymbol{\mathcal{F}}\mathbf{A}_{g,s}\mathbf{x} \tag{6.10a}$$

$$\Leftrightarrow \bar{\boldsymbol{\ell}}_{g,s} + \bar{\mathbf{v}}_{g,s} = \bar{\mathbf{A}}_{g,s}\mathbf{x} \qquad \boldsymbol{\Sigma}_{\bar{\boldsymbol{\ell}}_{g,s}\bar{\boldsymbol{\ell}}_{g,s}} = \sigma_{g,s}^2\boldsymbol{\mathcal{F}}\mathbf{Q}_{\boldsymbol{\ell}_{g,s}\boldsymbol{\ell}_{g,s}}\boldsymbol{\mathcal{F}}^T = \sigma_{g,s}^2\mathbf{I}. \tag{6.10b}$$

The filter is estimated from the data itself as an approximation of the covariance matrix in terms of a complete decorrelation (e.g. Koch, 1999, p. 154), such that $\sigma_{g,s}^2\boldsymbol{\mathcal{F}}\mathbf{Q}_{\boldsymbol{\ell}_{g,s}\boldsymbol{\ell}_{g,s}}\boldsymbol{\mathcal{F}}^T = \sigma_{g,s}^2\mathbf{I}$ holds as good as possible.

Filter Concept The basic idea is to represent the observation noise time series as a cascaded ARMA process. A cascaded process is a consecutive sequence of ARMA processes. Each individual process $k \in \{0 \ldots K-1\}$ is called cascade. The process is adjusted to the gradiometer noise and then inverted to obtain the corresponding decorrelation filter (e.g. , Chap. 5). This filter is composed of individual filter cascades $k \in \{0 \ldots K-1\}$. Each of the individual filter cascades k is described by two sets of coefficients i.e. α_q^k for the recursive and β_p^k for the non-recursive part of the filter. Their applicability for the modeling of the gradiometer noise was studied in Schuh (2002, 2003), Schuh et al. (2006), Siemes (2008) and the use for the real data was demonstrated in Schuh et al. (2010), Krasbutter et al. (2011b,a) and Krasbutter et al. (2014). As a general form, a single cascade ARMA filter can be applied to a correlated observation vector $\boldsymbol{\ell}$ as filter input via

$$\bar{\ell}_i = \sum_{p=0}^{P^k}\beta_p^k\ell_{t-p} + \sum_{q=1}^{Q^k}\alpha_q^k\bar{\ell}_{t-q}, \tag{6.11}$$

resulting in the filter output, i.e. the decorrelated observation vector $\bar{\boldsymbol{\ell}}$. P^k and Q^k denote the filter order of the non-recursive and of the recursive parts. For cascaded filters, (6.11) is consecutively applied for every cascade k (see Alg. 6.1 and Sect. 6.3.3 for details) with $\bar{\boldsymbol{\ell}}$ as new input $\boldsymbol{\ell}$. (6.11) can be rewritten as matrix-vector operations (or matrix-matrix operations if the columns of the design matrix are to be filtered), rewriting (6.11) as

$$0 = \sum_{p=0}^{P^k} \beta_p^k \ell_{t-p} + \sum_{q=1}^{Q^k} \alpha_q^k \bar{\ell}_{t-q} - \bar{\ell}_i, \qquad \alpha_0^k := -1 \tag{6.12a}$$

$$0 = \sum_{p=0}^{P^k} \beta_p^k \ell_{t-p} + \sum_{q=0}^{Q^k} \alpha_q^k \bar{\ell}_{t-q} \tag{6.12b}$$

$$0 = \mathbf{F}_{\beta^k}\boldsymbol{\ell} - \mathbf{F}_{\alpha^k}\bar{\boldsymbol{\ell}} \tag{6.12c}$$

$$\mathbf{F}_{\alpha^k}\bar{\boldsymbol{\ell}} = \mathbf{F}_{\beta^k}\boldsymbol{\ell} \tag{6.12d}$$

$$\bar{\boldsymbol{\ell}} = \mathbf{F}_{\alpha^k}^{-1}\mathbf{F}_{\beta^k}\boldsymbol{\ell}, \qquad \bar{\boldsymbol{\ell}} = \texttt{forwardSubstitution}\left(\mathbf{F}_{\alpha^k}, \mathbf{F}_{\beta^k}\boldsymbol{\ell}\right), \tag{6.12e}$$

and defining the matrices

$$\mathbf{F}_{\alpha^k} := \begin{bmatrix} -\alpha_0^k & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ -\alpha_1^k & -\alpha_0^k & 0 & \ddots & \ddots & \ddots & \cdots & \cdots & 0 \\ \vdots & -\alpha_1^k & -\alpha_0^k & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ -\alpha_q^k & \ddots & -\alpha_1^k & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ -\alpha_{Q^k}^k & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 & 0 \\ \vdots & \ddots & -\alpha_{Q^k}^k & \ddots & \ddots & \ddots & \ddots & -\alpha_0^k & 0 \\ 0 & 0 & 0 & -\alpha_{Q^k}^k & -\alpha_{Q^k-1}^k & \cdots & -\alpha_q^k & \cdots & -\alpha_0^k \end{bmatrix} \tag{6.13}$$

and

$$\mathbf{F}_{\beta^k} := \begin{bmatrix} \beta_0^k & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ \beta_1^k & \beta_0^k & 0 & \ddots & \ddots & \ddots & \cdots & \cdots & 0 \\ \vdots & \beta_1^k & \beta_0^k & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ \beta_p^k & \ddots & \beta_1^k & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ \beta_{P^k}^k & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 & 0 \\ \vdots & \ddots & \beta_{P^k}^k & \ddots & \ddots & \ddots & \ddots & \beta_0^k & 0 \\ 0 & 0 & 0 & \beta_P^k & \beta_{P^k-1}^k & \cdots & \beta_p^k & \cdots & \beta_0^k \end{bmatrix} . \tag{6.14}$$

Note that the solution of (6.12e) always exists (as there are no zero entries in the diagonal) and can be efficiently determined via forward substitution (e.g. Golub and van Loan, 1996, p. 88). As the filter used within the SGG decorrelation comprises several cascades, the individual filters k are consecutively applied to the observations (and the design matrix). Applying (6.12e) K times results in Alg. 6.1.

Algorithm 6.1: Application of a cascaded filter with K cascades to a matrix.

Data:

$\mathbf{F}_{\alpha^k}$	filter matrix of recursive part kth cascade, $k \in \{0, \dots, K-1\}$
$\mathbf{F}_{\beta^k}$	filter matrix of non-recursive kth cascade, $k \in \{0, \dots, K-1\}$
$\mathbf{L}$	matrix (or vector) to be filtered

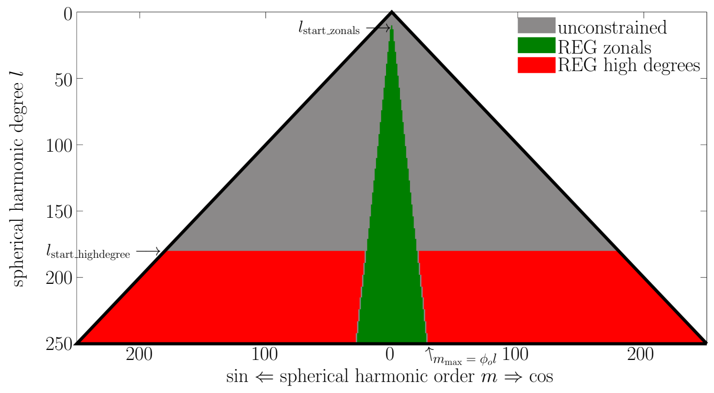
```
1  // initialization
2  $\bar{\mathbf{L}}^{(0)} = \mathbf{L}$
3  // loop over all cascades k
4  for $k = 0$ to $K-1$ do
5      // apply non-recursive part of cascade k
6      $\hat{\mathbf{L}}^{(k)} = \mathbf{F}_{\beta^k}\bar{\mathbf{L}}^{(k-1)}$
7      // apply recursive part of cascade k
8      $\bar{\mathbf{L}}^{(k)} = \text{forwardSubstiution}\left(\mathbf{F}_{\alpha^k}, \hat{\mathbf{L}}^{(k)}\right)$// numerical stable solution
9  end
10 $\bar{\mathbf{L}} = \bar{\mathbf{L}}^{(K-1)}$
11 return $\bar{\mathbf{L}}$// matrix filtered with all cascades
```

Filter Design The coefficients of the filter cascades are estimated data-adaptive from the observations. A cascaded ARMA process is adjusted to the estimated gradiometer noise (i.e. the residuals of the observations after a gravity field adjustment, Schuh, 2002, 2003, Siemes, 2008). As the filters are iteratively refined, the adjustment is repeated with different filters to estimate an update of the residuals. The iteration is started either with a known gravity field model or with a known filter model. From real data analysis it is known that the filter estimation convergence is fast. Normally, convergence can be seen after two to four iterations (Schuh et al., 2010, Krasbutter et al., 2011a, Brockmann et al., 2013), depending on the quality of the initial filter or gravity field model. For iterating the filter design, instead of using the solver developed in this work, the fast conjugate gradient based solver is used, as the NEQs are not needed (Schuh, 1996, Boxhammer, 2006, Siemes, 2008, Brockmann et al., 2010). Some details on the filter estimates covering the real data analysis are shown in Sect. 6.5.3.1.

6.2.3 Constraints

As global gravity field determination from satellite data represents an ill-posed problem, a kind of regularization or smoothness conditions have to be applied, if a gravity field solution just based on GOCE observations should be estimated. In the special case of GOCE observations, two different effects occur. The polar gap as a consequence of the sun synchronous orbit mainly yields unstable estimates of the near zonal coefficients (van Gelderen and Koop, 1997). In addition, the signal is damped in the satellites altitude. This yields a poor signal to noise ratio and causes a high amplitude of the error in the higher spherical harmonic degrees. Adding some smoothness conditions for the higher frequencies, the entry of high frequency noise into the solution can be reduced. Nevertheless, some signal might be damped as well. The regularization has to be applied, if the model is evaluated for a subset of the parameters only, which is the case in many applications. The coefficients constrained by the regularization are summarized in Fig. 6.3.

Regularization for the Polar Gap Different strategies exist to stabilize the solution despite the polar gap. One strategy is to introduce observations in the polar areas, for instance real measurements e.g., from the Arctic Gravity Project (ArcGP, Forsberg and Kenyon, 2000) and from the Antarctic Geoid Project (AntGP, Scheinert, 2005, Scheinert et al., 2008). Instead of real measurements, synthesized observations from global models can be used, e.g. gravity anomalies on a



unconstrained
REG zonals
REG high degrees
$l_{\text{start\_zonals}}$
$l_{\text{start\_highdegree}}$
$m_{\max} = \phi_o l$
spherical harmonic degree l
0
50
100
150
200
250
200
100
0
100
200
sin ⇐ spherical harmonic order m ⇒ cos

Figure 6.3: Coefficients zones constrained by the regularization. The unconstrained coefficients are shown in grey, the coefficients mostly affected by the polar gap in green and the regularized high degree coefficients in red.

grid (see e.g. Yi, 2012). A tailored concept for the GOCE mission was developed by Metzler and Pail (2005) and Metzler (2007) who implemented a spatially restricted regularization method. Within this work, a simple but very efficient type of regularization is used. The prior information that the spherical harmonic coefficients are zero – with a certain variance – is used. As form of a Thikonov regularization, a diagonal regularization matrix is constructed, using Kaula's rule of thumb (cf. Kaula, 1966, p. 98) for degree variances, which approximates empirically the signal loss of the higher degrees of the Earth's gravity field. Pseudo observations for individual coefficients

$$\mathcal{E}(x_i) = 0, \qquad \sigma_{x_i x_i} = \frac{10^{-5}}{l^2} \tag{6.15}$$

are introduced for the near zonal coefficients x_i. The diagonal elements of the regularization matrix are then

$$\mathbf{N}_{\text{zonals}}(i,i) = 10^{10} l^4, \qquad \mathbf{n}_{\text{zonals}}(i) = 0 \tag{6.16}$$

if the parameter in row/column i corresponds to a degree above a certain start degree ($l_{\text{start\_zonals}}$) and if the order m is influenced by the polar gap. This is determined via (van Gelderen and Koop, 1997)

$$m < m_{\max} = \theta_0 l. \tag{6.17}$$

θ_0 is the opening angle of the gap in radians, which is approximately 6.5° for GOCE. The affected coefficient zone is shown in green in Fig. 6.3. Alg. 6.2 summarizes the setup of the diagonal matrix.

Regularization of the Higher Degrees A second regularization group is used to constrain the coefficients of the higher degrees. The signal to noise ratio should be improved. A Kaula regularization according to (6.15) is applied to all coefficients above a certain start degree. Koch et al. (2012) study Monte Carlo based approaches to determine an optimal start degree for the regularization. The near zonal coefficients already constrained are excluded. The affected coefficient zone is shown in red in Fig. 6.3. The design of the regularization matrix is summarized in Alg. 6.3.

Two matrices are used to obtain individual weights for the zonal and the high degree regularization (cf. Fig. 6.3).

Algorithm 6.2: Setup of the regularaization matrix for the polar gap.

Data:

θ_0	opening angle of polar gap in radians
$\mathbf{p}$	symbolic numbering scheme for the result matrix
$l_{\text{start\_zonals}}$	start degree of the regularization

```
// Loop over all parameters
for $i = 0$ to p.size() do
    $\mathbf{N}_{\text{zonals}}(i,i) = 0$
    $\mathbf{n}_{\text{zonals}}(i) = 0$
    // if coefficient i is a spherical harmonic coefficient (sine or cosine)
    if (p(i).type() == SH_C) || (p(i).type() == SH_S) then
        $l = \mathbf{p}(i)$.l()
        $m = \mathbf{p}(i)$.m()
        // if the degree is above the start degree and the order is affected by the gap
        if $(l >= l_{start\_zonals})$ && $(m < \theta_0 l)$ then
            $\mathbf{N}_{\text{zonals}}(i,i) = 10^{10} l^4$
        end
    end
end
$\bar{\mathbf{L}} = \bar{\mathbf{L}}^{(K)}$
return $\mathbf{N}_{\text{zonals}}$, $\mathbf{n}_{\text{zonals}}$ // Regularisation matrix and zero RHS
```

6.2.4 Data Combination and Joint Solution

As the combination of the observation groups and the computation of the joint solution is a simple – because smaller – problem of the high degree application as described in Chap. 7, all computational and implementational details are given there. The next Sect. 6.3 will focus on the assembly of the gradiometry NEQs. Assume the NEQs are already assembled, the following steps are performed to derive the joint solution (implementational details and computational challenges are given in Chap. 7):

1. Recall all NEQs from disk and perform the weighted update of $\mathbf{N}$ and $\mathbf{n}$ according to (6.1). Account for different parameter numbering schemes and different maximal spherical harmonic resolutions.
2. Add diagonal regularization matrices to the distributed NEQs.
3. Compute the Cholesky decomposition of $\mathbf{N}$.
4. Determine the solution $\mathbf{x}$ via for- and backward substitution.
5. If needed, compute the covariance matrix of the parameters $\mathbf{\Sigma_{xx}} = \mathbf{N}^{-1}$ inverting the already Cholesky reduced matrix $\mathbf{N}$.
6. Derive estimates for the weights using (4.7), (4.8a) and (4.9).
7. Iterate until weights converge to derive final weighted solution for the unknowns $\mathbf{x}$ and their covariance matrix $\mathbf{\Sigma_{xx}} = \mathbf{N}^{-1}$.

6.3 Gradiometry NEQ Assembly in a HPC Environment

For the application of GOCE data processing the computational challenge is a fast and efficient setup of the normal equations for the gradiometry observations. Four basic steps have to be performed to derive the NEQs as a block-cyclic distributed matrix. (i) The observations have to be distributed

Algorithm 6.3: Setup of the regularization matrix for the higher degree coefficients.

Data:

θ_0	opening angle of polar gap in radians
$\mathbf{p}$	symbolic numbering scheme for the result matrix
$l_{\text{start\_highdegree}}$	start degree of the regularization

```
1  // Loop over all parameters
2  for $i = 0$ to $\mathbf{p}$.size() do
3      $\mathbf{N}_{\text{highdeg}}(i,i) = 0$
4      $\mathbf{n}_{\text{highdeg}}(i) = 0$
5      // if coefficient i is a spherical harmonic coefficient (sine or cosine)
6      if ($\mathbf{p}(i)$.type() == SH_C) || ($\mathbf{p}(i)$.type() == SH_S) then
7          $l = \mathbf{p}(i)$.l()
8          $m = \mathbf{p}(i)$.m()
9          // if the degree is above the start degree and the order is not affected by the gap
10         if ($l >= l_{start\_highdegree}$) && ($m >= \theta_0 l$) then
11             $\mathbf{N}_{\text{highdeg}}(i,i) = 10^{10} l^4$
12         end
13     end
14 end
15 $\bar{\mathbf{L}} = \bar{\mathbf{L}}^{(K)}$
16 return $\mathbf{N}_{\text{highdeg}}$, $\mathbf{n}_{\text{highdeg}}$// Regularization matrix and zero RHS
```

along the compute core grid, (ii) the design matrix for the second derivative of the potential in the GRF has to be set up as a block-cyclic distributed matrix, (iii) the design matrix and the observation vector have to be filtered, i.e. decorrelated and (iv) finally the update of the NEQs i.e. the computation of $\bar{\mathbf{A}}_{g,s}^T \bar{\mathbf{A}}_{g,s}$ has to be performed.

For simplicity, only the implementation for a single segment s and a single tensor component g is described. However, the procedure is the same for every segment s and every tensor component g. Thus, within this chapter, the subscripts s and g are neglected in the equations. The objective of this section is to describe the massive parallel implementation of the computation of $\mathbf{N}_{g,s}$ including the four steps mentioned above. As every segment can consist of millions of observations, denoted as M_s, the segments are subdivided into blocks b ($b \in \{0, 1, \ldots, M_s/b_{\text{obs}}\}$) of an arbitrary block size b_{obs} and are processed in a block-wise approach. Within every step, $\mathbf{N}_{g,s}$ is updated for b_{obs} observations.

6.3.1 Distribution of the Observations Along the Compute Core Grid

Within a loop, the design matrix for b_{obs} observations is set up for all U parameters. Thus, an empty distributed matrix $\mathbf{A}$ of dimension $b_{\text{obs}} \times U$ is set up in a first step. After that, depending on the dimension of the compute core grid and the parameters of the block-cyclic distribution, on each core the local matrix $\mathbf{A}^l_{\mathsf{r},\mathsf{c}}$ of dimension $R^l_{\mathsf{r},\mathsf{c}} \times C^l_{\mathsf{r},\mathsf{c}}$ exists. Its dimension, depending on the coordinates of the processor, can be obtained via (3.5). Whereas the parameters are distributed over the columns of the compute core grid, the observations are distributed over the rows of the compute core grid. Hence, all local matrices $\mathbf{A}^l_{\mathsf{r},*}$ of a certain row r of the compute core grid contain entries for the same observations, but for different subspaces of the parameters. All local matrices $\mathbf{A}^l_{*,\mathsf{c}}$ along a column c of the compute core grid contain as columns the same subspace of parameters, but for a different subset of the observations. Thus, the observations i.e. the gravity gradients of component g plus the rotation matrix and the satellites position are distributed over the rows of the compute core grid. Every row of the compute core grid processes the same observations but only for a subspace of the parameters. For the chosen decorrelation concept, i.e. the decorrelation by digital filters, it is important to guarantee a chronological ordering of the observations. As the sparse filtering matrices (cf. Sect. 6.2.2.2) contradict the concept of the block-cyclic distribution, an alternative filtering is implemented (cf. Sect. 6.3.3). This local filtering which requires the local part of the observation vector $\boldsymbol{\ell}^l_{\mathsf{r,c}}$ (and the rows of the local part of the design matrices $\mathbf{A}^l_{\mathsf{r,c}}$) to be sorted chronological and containing equidistant consecutive observations. As the order of the rows of the global matrix $\mathbf{A}$ does not matter for the final computation of $\mathbf{A}^T\mathbf{A}$, an ordering of the observations in the global matrix views of $\mathbf{A}$ and $\boldsymbol{\ell}$ tailored for the filtering is chosen. The observations are distributed such that not the global view on the matrix $\mathbf{A}$ produces chronological ordered equidistant observations but the virtually composed matrices

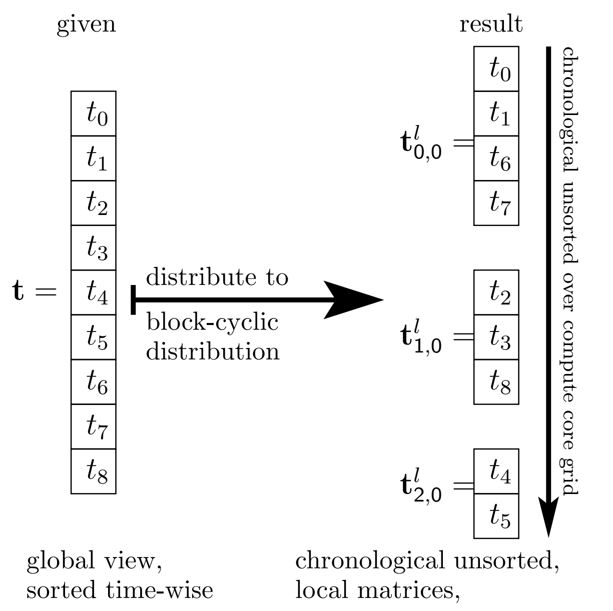
$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{A}^l_{\mathsf{0,0}} & \mathbf{A}^l_{\mathsf{0,1}} & \mathbf{A}^l_{\mathsf{0,2}} & \cdots & \mathbf{A}^l_{\mathsf{0,C-1}} \\ \mathbf{A}^l_{\mathsf{1,0}} & \mathbf{A}^l_{\mathsf{1,1}} & \mathbf{A}^l_{\mathsf{1,2}} & \cdots & \mathbf{A}^l_{\mathsf{1,C-1}} \\ \mathbf{A}^l_{\mathsf{2,0}} & \mathbf{A}^l_{\mathsf{2,1}} & \mathbf{A}^l_{\mathsf{2,2}} & \cdots & \mathbf{A}^l_{\mathsf{2,C-1}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{A}^l_{\mathsf{R-1,0}} & \mathbf{A}^l_{\mathsf{R-1,1}} & \mathbf{A}^l_{\mathsf{R-1,2}} & \cdots & \mathbf{A}^l_{\mathsf{R-1,C-1}} \end{bmatrix} \text{ and } \hat{\boldsymbol{\ell}} = \begin{bmatrix} \boldsymbol{\ell}^l_{\mathsf{0,0}} \\ \boldsymbol{\ell}^l_{\mathsf{1,0}} \\ \boldsymbol{\ell}^l_{\mathsf{2,0}} \\ \vdots \\ \boldsymbol{\ell}^l_{\mathsf{R-1,0}} \end{bmatrix}. \qquad (6.18)$$

Thus, the filter operations can be performed row by row of the compute core grid sequentially, but with limited communication, as a) the observations in each individual local matrix are equidistant and sorted chronological and b) all local matrices are chronological ordered along of a column of the compute core grid and are equidistant. To achieve this distribution for the observation vector and the meta information (satellite position and orientation in space), b_{obs} observations are distributed along a column of the compute core grid. Assuming the number of rows of the local matrix to be $R^l_{\mathsf{r,c}}$ (computed with (3.5)), the first $R^l_{\mathsf{0},*}$ of the b_{obs} are send by the master process to the processors of the first processor grid's row, the next $R^l_{\mathsf{1},*}$ observations are send to the processes of the second processor grid's row and so on. The last processor row gets the last $R^l_{\mathsf{R-1},*}$ observations of the total b_{obs} observations. The data are distributed via a sequence of MPI send operations. Afterwards, each process has the observations to be processed and the local observations on each processes are sorted chronological and are equidistant in time. Fig. 6.4 illustrates the distribution with as simple example. A chronological sorted vector $\mathbf{t}$ of dimension 9×1 is distributed over a 3×1 compute core grid with $b_r = 2$. The chosen distribution is compared to the standard distribution, i.e. distributing the chronological sorted OEQs following the block-cyclic distribution.

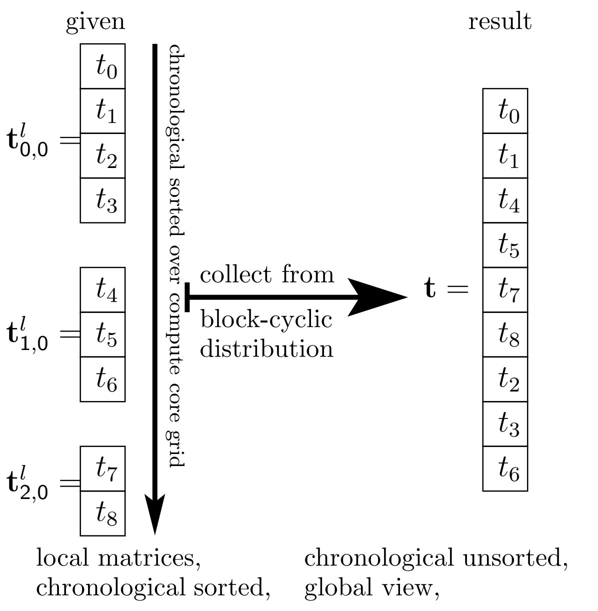
6.3.2 Assembly of the Design Matrices

Within the next step, the design matrix has to be set up in the GRF according to the procedure summarized in Sect. 6.2.2.1. The resulting matrix should be directly the distributed matrix $\mathbf{A}$, each processor has to fill its local part $\mathbf{A}^l_{\mathsf{r,c}}$ for its part of the observations and its sub-space of the parameters. The following steps are performed sequentially for every observation the process is responsible for. All processes of the compute core grid perform the setup in parallel. Processes of the same column of the compute core grid perform the steps for different observations (in a global view, local rows of $\mathbf{A}$, i.e. rows of $\mathbf{A}^l_{\mathsf{r,c}}$) but for the same parameter subset. The processes of a same row of the compute core grid perform the steps for different subset of the overall parameters (in the global view, local columns of $\mathbf{A}$, i.e. column of $\mathbf{A}^l_{\mathsf{r,c}}$) but for the same observations. The only computations which might be redundant on some cores is the recursive computation of Legendre functions which are computed recursive for every spherical harmonic order. The amount of redundant computations depends on the target numbering scheme of $\mathbf{A}$.

Within the first step, a serial $6 \times C^l_{\mathsf{r,c}}$ matrix is set up containing the design matrix of the gravity gradient for all six tensor components in the LNOF for the $C^l_{\mathsf{r,c}}$ local parameters, i.e. the parameters corresponding to the locally stored columns of the local part of the design matrix $\mathbf{A}^l_{\mathsf{r,c}}$. This local matrix is then rotated into the GRF applying tensor rotations (cf. Sect. 6.2.2.1). The rows of this


given
result
$\mathbf{t} =$
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8
distribute to
block-cyclic distribution
$\mathbf{t}^l_{0,0} =$ t_0 t_1 t_6 t_7
$\mathbf{t}^l_{1,0} =$ t_2 t_3 t_8
$\mathbf{t}^l_{2,0} =$ t_4 t_5
chronological unsorted over compute core grid
global view, sorted time-wise
chronological unsorted, local matrices,

(a) Standard block-cyclic distribution of a chronological sorted vector. The vector is distributed as a block-cyclic distributed matrix. **Not used for the SGG observation equations!**


given
result
$\mathbf{t}^l_{0,0} =$ t_0 t_1 t_2 t_3
$\mathbf{t}^l_{1,0} =$ t_4 t_5 t_6
$\mathbf{t}^l_{2,0} =$ t_7 t_8
chronological sorted over compute core grid
collect from
block-cyclic distribution
$\mathbf{t} =$
t_0 t_1 t_4 t_5 t_7 t_8 t_2 t_3 t_6
local matrices, chronological sorted,
chronological unsorted, global view,

(b) The local matrices are chronological filled. The global view (collected) block-cyclic distributed matrix is not chronological sorted. **Used for SGG observation equations!**

Figure 6.4: Distribution of the GOCE SGG observation equations as a chronological sorted block-cyclic distributed matrix and the tailored distribution for the filtering.

$6 \times C^l_{\mathsf{r},\mathsf{c}}$ matrix correspond to the tensor components $g \in G$. The row associated with the gradient component $\mathbf{A}$ should be computed for, is copied to the ith row of $\mathbf{A}^l_{\mathsf{r},\mathsf{c}}$, such that the distributed matrix is filled observation per observation. In addition, the processes of the first compute core grids column $\mathsf{c} = 0$ copy the observations to their local matrix of the observation vector $\boldsymbol{\ell}^l_{\mathsf{r},\mathsf{c}}$. $\mathbf{A}^l_{\mathsf{r},\mathsf{c}}$ and $\boldsymbol{\ell}^l_{\mathsf{r},\mathsf{c}}$ are now available in a distributed form, having a row-wise ordering which is suited for the next step which is the filtering. It is important to realize that the observations are not sorted chronological in the view on the global matrix $\mathbf{A}$.

6.3.3 Applying the Decorrelation by Recursive and Non-Recursive Digital Filters

The implementation of the decorrelation is the most technical step within the procedure. From the implementational point of view a compromise between a general and flexible implementation and an efficient approach has to be found. The challenges are the correlations between all observations of a segment s and the recursive part of the used decorrelation filter model. The used sequence of non-recursive and recursive filters is able to model correlations along thousands to hundreds of thousand of observations. As, due to memory limitations, $\mathbf{A}_{s,g}$ can never be set up at once, even distributed over the compute core grid, only parts of the OEQs with a size b_{obs} can be set up at the same time. Thus, the filtering needs to be implemented as a block-wise filtering, accounting for correlations in the past from previous blocks. In addition, the rows of the design matrix with only b_{obs} rows are already distributed in blocks over the compute core grids rows (and columns), as for the OEQs the concept of block-cyclic distributed matrices is used. Thus, an additional block filtering is required. In summary, there is a twice nested block filtering. This is furthermore complicated by the fact that recursive filters are used. As an input, they require the already filtered foregoing parts of the time series. Last but not least, the filters used are cascaded filters, and thus this steps have to be repeated K times for every cascade k of the filter.

6.3.3.1 Implementation of the Non-Recursive Filter

As the filtering of different columns of a matrix can be independently performed, the given procedure is the same for all columns of the compute core grid. The filtering is performed totally in parallel by different columns of the compute core grid. Hence the focus is only on a single column of the compute core grid, e.g. for the column $\mathsf{c} = 0$. First of all, it is assumed that the filter only consists of a single cascade. For the descriptive development of the algorithm a compute core grid with only $\mathsf{R} = 3$ rows is assumed. The first step of the filtering is the application of the non-recursive part, i.e. mathematically written for the chosen distribution of the OEQs cf. (6.12c) the computations of

$$\begin{bmatrix}\bar{\mathbf{A}}^l_{0,0}\\ \bar{\mathbf{A}}^l_{1,0}\\ \bar{\mathbf{A}}^l_{2,0}\end{bmatrix} = \mathbf{F}_{\beta^k}\begin{bmatrix}\mathbf{A}^l_{0,0}\\ \mathbf{A}^l_{1,0}\\ \mathbf{A}^l_{2,0}\end{bmatrix} = \begin{bmatrix}\mathbf{F}_{\beta^k 0,0} & \mathbf{0} & \mathbf{0}\\ \mathbf{F}_{\beta^k 1,0} & \mathbf{F}_{\beta^k 1,1} & \mathbf{0}\\ \mathbf{0} & \mathbf{F}_{\beta^k 2,1} & \mathbf{F}_{\beta^k 2,2}\end{bmatrix}\begin{bmatrix}\mathbf{A}^l_{0,0}\\ \mathbf{A}^l_{1,0}\\ \mathbf{A}^l_{2,0}\end{bmatrix} \tag{6.19a}$$

$$= \begin{bmatrix}\mathbf{F}_{\beta^k 0,0}\mathbf{A}^l_{0,0}\\ \mathbf{F}_{\beta^k 1,0}\mathbf{A}^l_{0,0} + \mathbf{F}_{\beta^k 1,1}\mathbf{A}^l_{1,0}\\ \mathbf{F}_{\beta^k 2,1}\mathbf{A}^l_{1,0} + \mathbf{F}_{\beta^k 2,2}\mathbf{A}^l_{2,0}\end{bmatrix}, \tag{6.19b}$$

illustrating the band-matrix $\mathbf{F}_{\beta^k}$ as block diagonal matrix plus an additional second diagonal block. Note that the matrices $\mathbf{F}_{\beta^k \mathsf{r,c}}$ are no local matrices in terms of the block-cyclic distributed matrices, but a serially stored matrix with subscripts just identifying the block in the overall matrix $\mathbf{F}_{\beta^k}$. If the serial filter matrices $\mathbf{F}_{\beta^k \mathsf{r,c}}$ are locally stored on the cores as full general matrices, time-invariant filter matrices with a Toeplitz structure as well as time-variant more general filters can be handled. The filtering can then be performed via matrix-matrix operations (e.g. BLAS Level 3). In the following, the implementation is discussed for time-invariant filters with constant coefficients, cf. (6.14). This representation is possible, as due to the used filter order (typically 50 to 300, 5000 for special cases), it can be always achieved that $R^l_{\mathsf{r,c}} > P^k$, e.g. via the choice of b_{obs}.

More general, each process (r, c) can locally compute the filtered part of its local design matrix (part of the observation vector) via

$$\bar{\mathbf{A}}^l_{\mathsf{r,c}} = \mathbf{F}_{\beta^k \mathsf{r,r-1}}\mathbf{A}^l_{\mathsf{r-1,c}} + \mathbf{F}_{\beta^k \mathsf{r,r}}\mathbf{A}^l_{\mathsf{r,c}} \tag{6.20}$$

if it shares the data $\mathbf{A}^l_{\mathsf{r-1,c}}$ or $\mathbf{F}_{\beta^k \mathsf{r,r-1}}\mathbf{A}^l_{\mathsf{r-1,c}}$ with the process $(\mathsf{r}-1, \mathsf{c})$ (row above in compute core grid). The filter matrices are set up serially for each process. For the special case $\mathsf{r} = 0$ and for the first block $b = 0$ of dimension b_{obs} of a segment there is no data $\mathbf{A}^l_{\mathsf{r-1,c}}$, i.e. data from the past. For this case $\mathbf{A}^l_{\mathsf{r-1,c}} = \mathbf{0}$ has to be assumed. For the case $\mathsf{r} = 0$ and $b > 0$, $\mathbf{A}^l_{\mathsf{R-1,c}}$ from the last block $b-1$ is used as data from the past. This matrix has to be kept in memory. This operations, i.e. two matrix-matrix multiplications can be performed with fast BLAS-L3 routines serially. But, $\mathbf{F}_{\beta^k \mathsf{r,r}}$ and $\mathbf{F}_{\beta^k \mathsf{r,r-1}}$ are high dimensional ($R^l_{\mathsf{r,c}} \times R^l_{\mathsf{r,c}}$ and $R^l_{\mathsf{r,c}} \times R^l_{\mathsf{r-1,c}}$, respectively) and thus the serial computation is time-consuming. Using standard filters both matrices are sparse, as they are banded matrices as $P^k < R^l_{\mathsf{r,c}}$ (for time-variant and time-invariant filters). Eq. (6.19a) is illustrated by Fig. 6.5 to show the properties of the computations. The first obvious idea is to represent the involved matrices as triangular matrices, thus instead of the BLAS-L3 matrix-matrix multiplication (`dgemm`) the triangular matrix-matrix multiplication can be used (`dtrmm`). The second diagonal matrices $\mathbf{F}_{\beta^k \mathsf{r,r-1}}$ are even sparser, as only the upper triangular of dimension $P^k - 1 \times P^k - 1$ is non-zero. Thus process (r, c) does only need the last $P^k - 1$ rows of $\mathbf{A}^l_{\mathsf{r-1,c}}$ or of $\mathbf{F}_{\beta^k \mathsf{r,r-1}}\mathbf{A}^l_{\mathsf{r-1,c}}$. (6.20) is rewritten as

$$\bar{\mathbf{A}}^l_{\mathsf{r,c}} = \begin{bmatrix}\mathbf{F}^{\ominus}_{\beta^k}\mathbf{A}^{l\ominus}_{\mathsf{r-1,c}}\\ \mathbf{0}\end{bmatrix} + \mathbf{F}_{\beta^k \mathsf{r,r}}\mathbf{A}^l_{\mathsf{r,c}} \tag{6.21}$$

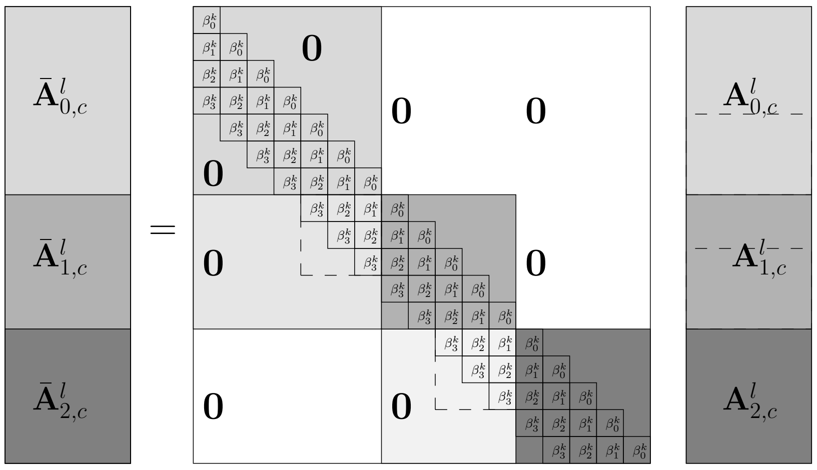

$\bar{\mathbf{A}}^l_{0,c}$
$\bar{\mathbf{A}}^l_{1,c}$
$\bar{\mathbf{A}}^l_{2,c}$
=
β^k_0 β^k_1 β^k_2 β^k_3
0
$\mathbf{A}^l_{0,c}$
$\mathbf{A}^l_{1,c}$
$\mathbf{A}^l_{2,c}$

Figure 6.5: Graphical depiction of the non-recursive filtering of the block-cyclic distributed matrices. The shape of involved matrices is shown. A time-invariant filter with constant coefficients is assumed (Toeplitz band matrix).

defining the sub-matrices as

$$\mathbf{A}^{l\ominus}_{\mathsf{r}-1,\mathsf{c}} := \mathbf{A}^{l\ominus}_{\mathsf{r}-1,\mathsf{c}}(end - P^k + 1 : end, :) \tag{6.22a}$$

$$\mathbf{F}^{\ominus}_{\beta^k} := \mathbf{F}_{\beta^k \mathsf{r},\mathsf{r}-1}(0 : P^k - 1, end - P^k + 1 : end). \tag{6.22b}$$

Each process sets up its local matrices $\mathbf{A}^l_{\mathsf{r},\mathsf{c}}$, sends its sub-matrix $\mathbf{A}^{l\ominus}_{\mathsf{r},\mathsf{c}}$ to process $(\mathsf{r}+1,\mathsf{c})$, performs the filtering $\mathbf{F}_{\beta^k \mathsf{r},\mathsf{r}}\mathbf{A}^l_{\mathsf{r},\mathsf{c}}$ and receives his $\mathbf{A}^{l\ominus}_{\mathsf{r}-1,\mathsf{c}}$ to update the filtered vector by the remaining part $\mathbf{F}^{\ominus}_{\beta^k}\mathbf{A}^{l\ominus}_{\mathsf{r}-1,\mathsf{c}}$. The last client of the compute core grids column, i.e. $(\mathsf{R}-1,\mathsf{c})$ sends his matrix $\mathbf{A}^{l\ominus}_{\mathsf{r},\mathsf{c}}$ again to the client $(0,\mathsf{c})$ as it is needed for the next block loop $b+1$, where the next b_{obs} observations of the same segment are processed.

This first, not very efficient but quite simple and massive parallel implementation of the non-recursive filtering of distributed design matrices is summarized in Alg. 6.4. The implementation has the advantages, that it is very flexible and can for example handle time-variant filters where the coefficients may vary in time.

The local computation of the product $\mathbf{F}_{\beta^k \mathsf{r},\mathsf{r}}\mathbf{A}^l_{\mathsf{r},\mathsf{c}}$ on each process can be accelerated using the special properties of the time-invariant filter matrix. For the case of real data, the number of local rows $R^l_{\mathsf{r},\mathsf{c}}$ on every process is in the order of several thousand observations. Thus, compared to the order P^k of used filters (which is typically 2 to 500 for real data processing, depending on the cascade k) the number of rows is large. The larger the imbalance between P^k and $R^l_{\mathsf{r},\mathsf{c}}$, the sparser the filter matrix. $\mathbf{F}_{\beta^k \mathsf{r},\mathsf{r}}$ contains many zeros on each core. The product, to be compute serially on each process is visualized in Fig. 6.6. The filter matrix $\mathbf{F}_{\beta^k \mathsf{r},\mathsf{r}}$ can be partioned into $P \times P$ triangular matrices, $\mathbf{F}^o_{\beta}$ the light grey ones and $\mathbf{F}^u_{\beta}$ dark grey ones. Also composing the matrix to be filtered into blocks of P rows, e.g. $\mathbf{A}^{l,j}_{\mathsf{r},\mathsf{c}}$ cf. Fig. 6.6, the product of that matrices can be rewritten as

$$\bar{\mathbf{A}}^{l,j}_{\mathsf{r},\mathsf{c}} = \mathbf{A}^{l,j}_{\mathsf{r},\mathsf{c}}\mathbf{F}^u_{\beta} + \mathbf{A}^{l,j-1}_{\mathsf{r},\mathsf{c}}\mathbf{F}^o_{\beta}. \tag{6.23}$$

Two main advantages are: i) Now small "full" triangular matrices are involved in the computations and ii) the filter matrices to be stored are only triangular of dimension $P \times P$ (storable in a single $P \times P$ matrix) and are now constant for the whole loop over all blocks b. This alternative filtering can be efficiently implemented in place via again using the BLAS-L3 routine for triangular matrix-matrix products (`dtrmm`) which is capable to operate on sub-matrices as well. This is important

Algorithm 6.4: Application of the non-recursive filter to a distributed matrix.

```
Data:  vector<double>      $\boldsymbol{\beta}$   vector of $P$ filter coefficients
       DistributedMatrix   $\mathbf{A}$   distributed matrix to be filtered
       int                 $b$   number of block of size $b_{\text{obs}}$ to be processed

// determine my process coordinates and compute core grid dimension
size_t r, c, R, C
blacs_gridinfo( $\mathbf{A}$.context(), r, c, R, C )
// determine size of my local part of $\mathbf{A}$
size_t $R^l = \mathbf{A}$.Rl()
size_t $C^l = \mathbf{A}$.Cl()
// send last $P-1$ rows of $\mathbf{A}^l$ to next processor row
Matrix  $\mathbf{A}^{\ominus} = \mathbf{A}$.localMat($end - P + 1 : end, :$)
$\mathbf{A}^{\ominus}$.Send($(r+1)\%R, c$)
if r == 0 then
    // Initialize $\mathbf{A}^{\ominus}$ with zeros if member of the first processor grids row, and $b == 0$
    if $b$ == 0 then
        $\mathbf{A}^{\ominus} = \mathbf{0}$
    else
        // If $b > 0$, take memory from last round $b-1$, i.e. $\mathbf{A}^{\ominus\ominus}$
        $\mathbf{A}^{\ominus} = \mathbf{A}^{\ominus\ominus}$
    end
    // Save the very last $P-1$ rows for possible next block $b_{obs}$ observations $b+1$
    Matrix  $\mathbf{A}^{\ominus\ominus}$.Recv($R-1, c$)
else
    $\mathbf{A}^{\ominus}$.Recv($r-1, c$)
end
// set up serial filter matrix of size $R^l \times R^l$ according to (6.14)
Matrix  $\mathbf{F}_{\beta}$ =setUpFilterMatrix($\boldsymbol{\beta}$)
// apply filter in place using BLAS-L3 dtrmm
$\mathbf{A}$.localMat() = $\mathbf{F}_{\beta}\mathbf{A}$.localMat()
// update first $P-1$ rows of with missing part from the past
// set up filter matrix of size $P-1 \times P-1$ according to (6.22b)
Matrix  $\mathbf{F}_{\beta}^{\ominus}$ =setUpFilterMatrix($\boldsymbol{\beta}$)
$\mathbf{A}$.localMat()($1 : P-1, :$)+ = $\mathbf{F}_{\beta}^{\ominus}\mathbf{A}^{\ominus}$
return $\mathbf{A}$// as in place filtered matrix
```

for the last block j, as there might be sub-matrices of $\mathbf{F}_{\beta}^{o}$ and $\mathbf{F}_{\beta}^{u}$ involved and for accessing the sub-matrices $\mathbf{A}^{l,j}$. The implementation can only be performed in place, if the loop starts at the bottom of the vector. The unfiltered values from the past are needed, which are then stored in the beginning of the vector and are not overwritten with the filtered values. The extended, more efficient but technical implementation of the filtering is summarized in Alg. 6.5.

6.3.3.2 Implementation of the Recursive Part of the Filter

Having applied the non-recursive part of the filter of cascade k, the recursive part of the cascade k has to be applied. A recursion generally contradicts a massive parallel implementation – if the time series is distributed and not locally stored in the memory of a single core. Within the recursion, the filtering step needs the already filtered observations from the past (cf. (6.11)). Instead of a real massive parallel implementation, a simple but flexible parallelization was chosen. As for the non-recursive part, the recursive filtering is written as a simple example ($\mathsf{R} = 3$ and shown only for one column of the compute core grid). As for the non-recursive filtering, the recursive filtering

Algorithm 6.5: Application of the non-recursive filter to a distributed matrix in extended form.

	`vector<double>`	$\boldsymbol{\beta}$	vector of P filter coefficients
Data:	`DistributedMatrix`	$\mathbf{A}$	distributed matrix to be filtered
	`int`	b	number of block of size b_{obs} to be processed

```
// determine my process coordinates and compute core grid dimension
size_t r, c, R, C
blacs_gridinfo( $\mathbf{A}$.context(), r, c, R, C )
// determine size of my local part of $\mathbf{A}$
size_t $R^l = \mathbf{A}$.Rl()
size_t $C^l = \mathbf{A}$.Cl()
// send last $P-1$ rows of $\mathbf{A}^l$ to next processor row
Matrix  $\mathbf{A}^{\ominus} = \mathbf{A}$.localMat($end - P + 1 : end, :$)
$\mathbf{A}^{\ominus}$.Send($(\mathsf{r}+1)\%\mathsf{R}, \mathsf{c}$)
if $\mathsf{r} == 0$ then
    // Initialize $\mathbf{A}^{\ominus}$ with zeros if member of the first processor grids row, and $b == 0$
    if $b == 0$ then
        $\mathbf{A}^{\ominus} = \mathbf{0}$
    else
        // If $b > 0$, take memory from last round $b-1$, i.e. $\mathbf{A}^{\ominus\ominus}$
        $\mathbf{A}^{\ominus} = \mathbf{A}^{\ominus\ominus}$
    end
    // Save the very last $P-1$ rows for possible next block $b_{obs}$ observations $b+1$
    Matrix  $\mathbf{A}^{\ominus\ominus}$ .Recv($\mathsf{R}-1, \mathsf{c}$)
else
    $\mathbf{A}^{\ominus}$.Recv($\mathsf{r}-1, \mathsf{c}$)
end
// set up triangular filter matrices of size $P \times P$ according to Fig. 6.6
Matrix  $\mathbf{F}^u_\beta, \mathbf{F}^o_\beta$ =setUpFilterMatrix($\boldsymbol{\beta}$)
// apply filter in place, starting from the back, triangular matrix matrix product (dtrmm)
// rest block smaller then $P$
int  $k = R^l\%P$
$\mathbf{A}$.localMat($end - k : end, :$) = $\mathbf{F}^u_\beta(0 : k-1, 1 : k-1)\mathbf{A}$.localMat($end - k : end, :$)
$\mathbf{A}$.localMat($end - k : end, :$)+ = $\mathbf{F}^o_\beta(0 : k-1, :)\mathbf{A}$.localMat($end - k - P : end - P, :$)
// Loop over the full blocks exept the first $P$
for $b = r^l - k - 1 - P$ to $P - 1$ do
    $\mathbf{A}$.localMat($b : b + P - 1, :$) = $\mathbf{F}^u_\beta\mathbf{A}$.localMat($b : b + P - 1, :$)
    $\mathbf{A}$.localMat($b : b + P - 1 : end, :$)+ = $\mathbf{F}^o_\beta(0 : k-1, :)\mathbf{A}$.localMat($b - P - 1 : b, :$)
end
// Processing of the first block (only $\mathbf{F}^u_\beta$ needed)
$\mathbf{A}$.localMat($0 : P-1, :$) = $\mathbf{F}^u_\beta\mathbf{A}$.localMat($0 : P-1, :$)
// update first $P-1$ rows with missing part from the past
// set up filter matrix of size $P-1 \times P-1$ according to (6.22b)
Matrix  $\mathbf{F}^{\ominus}_\beta$ =setUpFilterMatrix($\boldsymbol{\beta}$)
$\mathbf{A}$.localMat($1 : P-1, :$)+ = $\mathbf{F}^{\ominus}_\beta\mathbf{A}^{\ominus}$
return $\mathbf{A}$// as in place filtered matrix
```

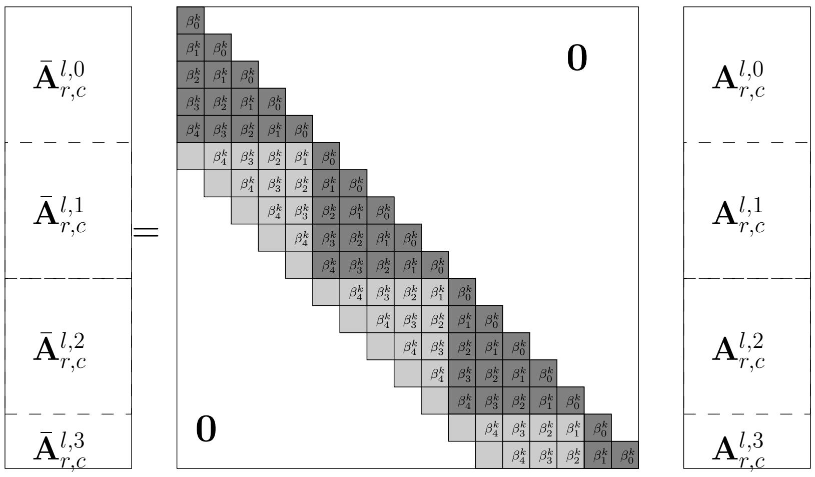

$\bar{\mathbf{A}}^{l,0}_{r,c}$
$\bar{\mathbf{A}}^{l,1}_{r,c}$
$\bar{\mathbf{A}}^{l,2}_{r,c}$
$\bar{\mathbf{A}}^{l,3}_{r,c}$
=
0
0
β^k_0 β^k_1 β^k_2 β^k_3 β^k_4
$\mathbf{A}^{l,0}_{r,c}$
$\mathbf{A}^{l,1}_{r,c}$
$\mathbf{A}^{l,2}_{r,c}$
$\mathbf{A}^{l,3}_{r,c}$

Figure 6.6: Graphical depiction of the serial part of non-recursive filtering on a single process. Special focus is on the shape and the partioning of the involved matrices. Shown is the case of a non-recursive time-invariant filter.

of different columns of the compute core grid remains independent. Thus, different columns of the compute core grid perform the filtering in parallel. For the first block $b = 0$ with b_{obs} observations the filtering is done via the solution of the system of equations (cf. (6.12d)), again representing the banded matrix $\mathbf{F}_{\alpha^k}$ as a blockdiagonal matrix including a second diagonal

$$\begin{bmatrix}\mathbf{A}^l_{0,0}\\ \mathbf{A}^l_{1,0}\\ \mathbf{A}^l_{2,0}\end{bmatrix} = \mathbf{F}_{\alpha^k}\begin{bmatrix}\bar{\mathbf{A}}^l_{0,0}\\ \bar{\mathbf{A}}^l_{1,0}\\ \bar{\mathbf{A}}^l_{2,0}\end{bmatrix} = \begin{bmatrix}\mathbf{F}_{\alpha^k 0,0} & \mathbf{0} & \mathbf{0}\\ \mathbf{F}_{\alpha^k 1,0} & \mathbf{F}_{\alpha^k 1,1} & \mathbf{0}\\ \mathbf{0} & \mathbf{F}_{\alpha^k 2,1} & \mathbf{F}_{\alpha^k 0,0}\end{bmatrix}\begin{bmatrix}\bar{\mathbf{A}}^l_{0,0}\\ \bar{\mathbf{A}}^l_{1,0}\\ \bar{\mathbf{A}}^l_{2,0}\end{bmatrix} \tag{6.24a}$$

$$= \begin{bmatrix}\mathbf{F}_{\alpha^k 0,0}\bar{\mathbf{A}}^l_{0,0}\\ \mathbf{F}_{\alpha^k 1,0}\bar{\mathbf{A}}^l_{0,0} + \mathbf{F}_{\alpha^k 1,1}\bar{\mathbf{A}}^l_{1,0}\\ \mathbf{F}_{\alpha^k 2,1}\bar{\mathbf{A}}^l_{1,0} + \mathbf{F}_{\alpha^k 1,1}\bar{\mathbf{A}}^l_{2,0}\end{bmatrix}. \tag{6.24b}$$

$\mathbf{A}^l_{\mathsf{r,c}}$ are the local matrices to be filtered and are $\bar{\mathbf{A}}^l_{\mathsf{r,c}}$ the filtered output matrices.

More general, on a single core, the operations

$$\mathbf{A}^l_{\mathsf{r,c}} = \mathbf{F}_{\alpha^k \mathsf{r,c-1}}\bar{\mathbf{A}}^l_{\mathsf{r-1,c}} + \mathbf{F}_{\alpha^k \mathsf{r,r}}\bar{\mathbf{A}}^l_{\mathsf{r,c}} \tag{6.25a}$$

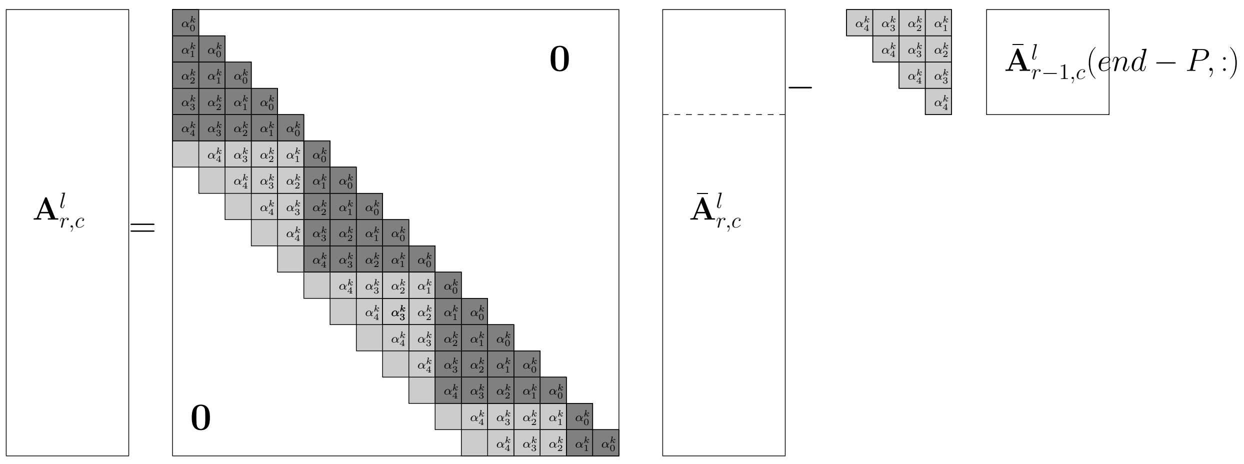
$$\mathbf{A}^l_{\mathsf{r,c}} - \mathbf{F}_{\alpha^k \mathsf{r,c-1}}\bar{\mathbf{A}}^l_{\mathsf{r-1,c}} = \mathbf{F}_{\alpha^k \mathsf{r,r}}\bar{\mathbf{A}}^l_{\mathsf{r,c}} \tag{6.25b}$$

$$\bar{\mathbf{A}}^l_{\mathsf{r,c}} = \text{solve}\left(\mathbf{F}_{\alpha^k \mathsf{r,r}}, \mathbf{A}^l_{\mathsf{r,c}} - \mathbf{F}_{\alpha^k \mathsf{r,c-1}}\bar{\mathbf{A}}^l_{\mathsf{r-1,c}}\right) \tag{6.25c}$$

$$\bar{\mathbf{A}}^l_{\mathsf{r,c}} = \mathbf{F}^{-1}_{\alpha^k \mathsf{r,r}}\left(\mathbf{A}^l_{\mathsf{r,c}} - \mathbf{F}_{\alpha^k \mathsf{r,c-1}}\bar{\mathbf{A}}^l_{\mathsf{r-1,c}}\right) \tag{6.25d}$$

have to be performed. As visible in (6.25c), process (r, c) needs the already filtered output of process $(\mathsf{r}-1, \mathsf{c})$ to solve the local filtering. The filtering is implemented only partially in parallel, to obtain a flexible implementation. Only the processes of different columns c of the compute core grid perform the filtering in parallel.

After the local filter operation, the filtered results are send to the next row $\mathsf{r}+1$ of the compute core grid, which themselves had to wait until $(\mathsf{r}-1, c)$ finished the filtering. The matrix to be filtered is reduced by the product $\mathbf{F}_{\alpha^k \mathsf{r,c-1}}\bar{\mathbf{A}}^l_{\mathsf{r-1,c}}$ and the system of equations has to be solved. The product can be rewritten, as compared to the non-recursive part of the filter as again $\mathbf{F}_{\alpha^k \mathsf{r,c-1}}$ is sparse. Only the upper triangle is filled with $Q-1$ diagonals (comparable to Fig. 6.5, demonstrated in Fig. 6.7).



$\mathbf{A}^l_{r,c}$
$=$
$\mathbf{0}$
$\mathbf{0}$
α^k_0 α^k_1 α^k_2 α^k_3 α^k_4
$\bar{\mathbf{A}}^l_{r,c}$
$-$
$\bar{\mathbf{A}}^l_{r-1,c}(end-P,:)$

Figure 6.7: Graphical depiction of the serial part of recursive filtering on a single process and the partioning of the involved matrices.

Instead of defining $\mathbf{F}_{\alpha^k\mathsf{r,c-1}}$, we can define the (on all processes) constant triangular matrix (for the case of time-invariant filters)

$$\mathbf{F}_\alpha^\ominus := \begin{bmatrix} \alpha_Q & \alpha_{Q-1} & \cdots & \alpha_1 \\ 0 & \alpha_Q & \cdots & \alpha_2 \\ \vdots & 0 & \alpha_Q & \vdots \\ 0 & 0 & 0 & \alpha_Q \end{bmatrix} \tag{6.26}$$

of dimension $Q \times Q$. Thus, only the first Q rows of $\mathbf{A}^l_{\mathsf{r,c}}$ have to be reduced by

$$\Delta\bar{\mathbf{A}}^l_{\mathsf{r-1,c}} := \mathbf{F}^\ominus_\alpha \bar{\mathbf{A}}^l_{\mathsf{r-1,c}}(end-Q:end,:), \tag{6.27a}$$

$$= \mathbf{F}^\ominus_\alpha \mathbf{A}^\ominus, \tag{6.27b}$$

using the defined sub-matrix (already filtered values from the past)

$$\mathbf{A}^\ominus := \bar{\mathbf{A}}^l_{\mathsf{r-1,c}}(end-Q:end,:). \tag{6.28}$$

After the reduction, the system of equations is solved (cf. Fig. 6.7). This can be performed via the solution of a triangular system (forward substitution, LAPACK routine `dtrsvm`) or via the solution of a banded system which also exists in the LAPACK library for serial matrices (`dgbsv`). Alg. 6.6 summarizes the (partially) parallel distributed filtering for the recursive part of the filter. Again, the implementation is flexible, and can be extended to filter coefficients changing in time very easily.

6.3.3.3 Application of Cascaded Filters

For general filters, consisting of multiple cascades, the non-recursive and recursive parts are applied consecutive for every cascade. For all cascades $k \in \{0, \dots, K-1\}$ first the non-recursive part, and afterwards the recursive part is applied to the matrices to be filtered. The procedure is shortly summarized in Alg. 6.7 and more precise later on in Alg. 6.8, summarizing the whole procedure of GOCE gradiometry NEQ assembly.

Two final things on the filtering have to be mentioned. As can been seen from the algorithms above, the filtering of the very first observations of every segment can only be performed approximative, as there are no observations from the past. Instead, zeros are assumed for the matrices $\mathbf{A}^l_{\mathsf{r-1,c}}$ im (6.20) and $\mathbf{A}^\ominus$ (cf. (6.28)). As this results in unreasonable filtered values, the first observations of a

Algorithm 6.6: Application of the recursive filter to a distributed matrix.

```
       vector<double>       $\boldsymbol{\alpha}$   vector of $Q$ filter coefficients
Data:  DistributedMatrix    $\mathbf{A}$   distributed matrix to be filtered
       int                  $b$   number of block of size $b_{\text{obs}}$ to be processed

// determine my process coordinates and compute core grid dimension
size_t r, c, R, C
blacs_gridinfo( $\mathbf{A}$.context(), r, c, R, C )
// determine size of my local part of $\mathbf{A}$
size_t $R^l = \mathbf{A}$.Rl()
size_t $C^l = \mathbf{A}$.Cl()
if r == 0 then
    // Initialize $\mathbf{A}^{\ominus}$ with zeros if member of the first processor grids row, and $b == 0$
    if $b == 0$ then
        $\mathbf{A}^{\ominus} = \mathbf{0}$
    else
        // If $b > 0$, take memory from last round $b-1$, i.e. $\mathbf{A}^{\ominus\ominus}$
        $\mathbf{A}^{\ominus} = \mathbf{A}^{\ominus\ominus}$
    end
    // Save the very last $Q-1$ rows for possible next block $b_{obs}$ observations $b+1$
    Matrix $\mathbf{A}^{\ominus\ominus}$.Recv($\mathsf{R}-1, \mathsf{c}$)
else
    // Blocking Receive, wait until matrix completely received
    $\mathbf{A}^{\ominus}$.Recv($\mathsf{r}-1, \mathsf{c}$)
end
// set up filter matrix of size $Q \times Q$ according to (6.26)
$\mathbf{F}_{\alpha}^{\ominus}$ =setUpFilterMatrix($\boldsymbol{\alpha}$)
// reduce first $Q-1$ rows with preceding filtered rows
$\mathbf{A}$.localMat($end - Q : end, :$)$-= \mathbf{F}_{\alpha}^{\ominus}\mathbf{A}^{\ominus}$
// set up filter matrix of size $R^l \times R^l$ according to (6.13)
$\mathbf{F}_{\alpha}$ =setUpFilterMatrix($\boldsymbol{\alpha}$)
// solve system of equations in place, either as band-system or as triangular system (dtrsvm,dgbsv)
$\mathbf{A}^l$ =solve($\mathbf{F}_{\alpha}, \mathbf{A}^l$)
// send last $Q-1$ rows of $\mathbf{A}^l$ to next processor row
$\mathbf{B} = \mathbf{A}$.localMat($end - Q + 1 : end, :$)
$\mathbf{B}$.Send($\mathsf{r} + 1\%\mathsf{R}, \mathsf{c}$)
return $\mathbf{A}$// as in place filtered matrix
```

segment cannot be used within the adjustment. The number of observations which are not properly decorrelated depends on the properties of the filter. This is called warmup phase (e.g. Schuh, 2003) of the filter, the length of the warmup phase and thus the amount of observations affected, can be numerically determined (e.g. Siemes, 2008, p. 74). These observations are removed always at the start of a new segment after the filtering and not assembled into the final NEQs.

Processing a long time-series in blocks b of size b_{obs} requires the so called filter memory, especially when using cascaded filters. After each run b, where all cascades are applied to the observations of that block, the unfiltered very last $P-1$ observations for the non-recursive filter and the filtered $Q-1$ already filtered observations have to be memorized. They serve within the next loop iteration where the next b_{obs} observations are processed as information from the past. Within Alg. 6.5 and 6.6, this is nothing else than the matrices $\mathbf{A}^{\ominus}$ of the last compute core grids row, i.e. the processes $(\mathsf{R}-1, \mathsf{c})$. They are send again to the processors $(0, \mathsf{c})$, to use them as $\mathbf{A}^{\ominus}$ within the next loop pass $b+1$. These matrices have to be stored for every cascade individually, to have the correct information of the past observations at the correct filtering status of the observations before. Within Alg. 6.5 and 6.6 these matrices are stored as $\mathbf{A}^{\ominus\ominus}$ on the processors $(0, \mathsf{c})$.

Algorithm 6.7: Application of an ARMA filter to a distributed matrix.

Data:	DistributedMatrix	$\boldsymbol{\alpha}^k$	vector of Q_k filter coefficients for cascade k
	vector<double>	$\boldsymbol{\beta}^k$	vector of P_k filter coefficients for cascade k
	vector<double>	$\mathbf{A}$	distributed matrix to be filtered

```
1 // loop over all filter cascades $k$
2 for $k = 0$ to $K - 1$ do
3     // apply non-recursive part of cascade $k$ according to Alg. 6.5
4     $\mathbf{A}$ = applyNonRecursivePart($\mathbf{A}$,$\boldsymbol{\beta}^k$)
5     // apply recursive part of cascade $k$ according to Alg. 6.6
6     $\mathbf{A}$ = applyRecursivePart($\mathbf{A}$,$\boldsymbol{\alpha}^k$)
7 end
8 return $\mathbf{A}$// as in place filtered matrix
```

6.3.4 Computation and Update of the NEQs

The finial step in the loop is the update of the normal equations, i.e. the computation of

$$\mathbf{N} = \mathbf{N} + \mathbf{A}^T\mathbf{A}, \ \mathbf{n} = \mathbf{n} + \mathbf{A}^T\boldsymbol{\ell} \text{ and } \lambda = \lambda + \mathbf{l}^T\mathbf{l} \tag{6.29}$$

for the observation block b of b_{obs} observations. As $\mathbf{A}$ and $\boldsymbol{\ell}$ are already decorrelated and in the block-cyclic distributed matrix storage scheme, the computations are directly performed calling PBLAS-L3 routines. Whereas $\mathbf{N} = \mathbf{N} + \mathbf{A}^T\mathbf{A}$ and $\lambda = \lambda + \mathbf{l}^T\mathbf{l}$ can be directly computed using the symmetric rank update (`pdsyrk`), $\mathbf{n} = \mathbf{n} + \mathbf{A}^T\boldsymbol{\ell}$ is computed with the general matrix-matrix product (`pdgemm`). All computations can be performed in place. Having the matrices in the right storage scheme, this step is trivial as directly routines from the library are called.

6.3.5 Composition of the Overall Assembly Algorithm

The implemented parts described above are connected and the whole algorithm is summarized in Alg. 6.8. It is extended with an outlier flagging. Existing flags are used to set entries corresponding to an outlier to zero. The outlier information is either provided as individual flags for every component g or as a single flag, which flags all components, if an outlier is observed in one of the used gradient components. The operation is performed after the decorrelation to keep the time series gapless for the filter process.

6.4 Runtime Analysis and Performance Analysis

Within this section the implemented parts (cf. Sect. 6.3) are analyzed with respect to their scaling behavior in a HPC environment. The test computations were performed on the JUROPA supercomputer at Forschungszentrum Jülich (Jülich, 2013). Thus, the timings and main conclusion are related to this machine and the hard– and software (libraries optimized for that machine) used there. The details on the used data set (real data) are summarized in Tab. 6.1. Only a small scenario was chosen, i.e. just the processing of a single component g and a single segment s as the implementation of the NEQ assembly is linear as well in the components as in the observations (assuming comparable filter orders for all segments and components which is the case for the real data). All runtime results shown were obtained via an empirical measurement of the wall clock time.

Algorithm 6.8: Algorithm of GOCE NEQ assembly.

Data:

`vector<double>`	$\boldsymbol{\alpha}^k$	vector of Q_k filter coefficients for cascade k
`vector<double>`	$\boldsymbol{\beta}^k$	vector of P_k filter coefficients for cascade k
`vector<bool>`	$\mathbf{f}$	outlier flags
`NumberingScheme`	$\mathbf{p}$	target numbering scheme

```
// determine my process coordinates and compute core grid dimension
size_t r, c, R, C
blacs_gridinfo( $\mathbf{A}$.context(), r, c, R, C )
size_t U = $\mathbf{p}$.size()
// Initialize distributed matrices for NEQs
DistributedMatrix $\mathbf{N}(U,U)$; DistributedMatrix $\mathbf{n}(U,1)$; DistributedMatrix $\lambda(1,1)$
// Initialize distributed matrices for OEQs
DistributedMatrix $\mathbf{A}(b_{\text{obs}},U)$; DistributedMatrix $\mathbf{l}(b_{\text{obs}},1)$
// Master reads all observations
if (r == 0) && (c == 0) then
    Matrix $\mathbf{L}$ =readObservationsFromFile()
end
// loop over all observations in blocks of $b_{obs}$
for $b = 0, b < M, b+ = b_{obs}$ do
    // distribute observations b to $b + b_{obs}$ (cf. Sect. 6.3.1)
    if (r == 0) && (c == 0) then
        $\mathbf{L}(b : b + b_{\text{obs}} - 1)$.distributeAlongGrid()
    else
        $\mathbf{l}$.localMat().Recv$(0,0)$
    end
    // fill local part of design matrix in GRF (cf. Sect. 6.3.2)
    $\mathbf{A}$.localMat() =fillDesignMatrix($\mathbf{p}$)
    // Apply decorrelation (cf. Sect. 6.3.3, Alg. 6.7)
    // Loop over all cascades k
    for $k = 0$ to $K - 1$ do
        // apply non recursive part of cascade k accord. to Alg. 6.5
        $\mathbf{A}$.localMat() = applyNonRecursivePart($\mathbf{A}$.localMat() ,$\boldsymbol{\beta}^k$)
        $\mathbf{l}$.localMat() = applyNonRecursivePart($\mathbf{l}$.localMat() ,$\boldsymbol{\beta}^k$)
        // apply recursive part of cascade k accord. to Alg. 6.6
        $\mathbf{A}$.localMat() = applyRecursivePart($\mathbf{A}$.localMat() ,$\boldsymbol{\alpha}^k$)
        $\mathbf{l}$.localMat() = applyRecursivePart($\mathbf{l}$.localMat() ,$\boldsymbol{\alpha}^k$)
    end
    // Set rows corresponding to flagged outliers to 0
    setZeros($\mathbf{A}$.localMat() ,$\mathbf{f}$)
    setZeros($\mathbf{l}$.localMat() ,$\mathbf{f}$)
    // Weighted update of NEQs (cf. Sect. 6.3.4)
    $\mathbf{N} = \mathbf{N} + w_{c,t}\mathbf{A}^T\mathbf{A}$; $\mathbf{n} = \mathbf{n} + w_{c,t}\mathbf{A}^T\mathbf{l}$; $\lambda = \lambda + w_{c,t}\mathbf{l}^T\mathbf{l}$
end
// Save NEQs on disk (cf. Sect. 3.3.4.1)
saveOnDisk($\mathbf{N}$, $\mathbf{n}$, $\lambda$)
// optional: solve for paramters (cf. Sect. 6.2.4)
$\mathbf{x}$ = solve($\mathbf{N}, \mathbf{n}$)
$\boldsymbol{\Sigma}_{\mathbf{xx}} = \mathbf{N}^{-1}$
return
```

Table 6.1: Used SGG data set for performance analysis of the implementation. S is the number of segments processed, K the number of cascades the filter consists of. P^k and Q^k are the orders of the recursive and non-recursive part of the filter cascade.

SGG observations				filters					Parameters	
#	V_{gg}	real data	S	K	P^0	Q^0	P^1	Q^1	d/o	#
6 132 019	XX-only	11/2009-01/2010	$1(s=0)$	2	1	1	50	50	$2-250$	62 997

6.4.1 Analysis of Scaling Behavior (Fixed Distribution Parameters)

Within this section, the scaling behavior of the individual steps is analyzed in detail. The distribution parameters of the block-cyclic matrix distribution were fixed ($b_s = b_r = 80$), only the number of cores used was varied. As it is known that the PBLAS and SCALAPACK routines work best for (close to) square matrices and (close to) quadratic compute core grids (see in addition Sect. 6.4.2, Blackford et al., 1997, p. 92), for this runtime analysis b_{obs} was fixed to 63 000 (nearly square matrix $\mathbf{A}$) and the compute core grid was always chosen quadratic. As on JUROPA only multiples of eight cores can be requested, the total numbers of cores N used are some non-obvious numbers as e.g. $\mathsf{N} = \mathsf{1\,936}$ ($\mathsf{44} \times \mathsf{44}$ compute core grid, $\mathsf{242}$ nodes each $\mathsf{8}$ cores). The given absolute runtime measurements are the mean values for the processing of a single block of $b_{\text{obs}} = 63\,000$ observations. Average values for the runtime were computed from the processing of 6 132 019/63 000 $\approx$ 97 blocks. The implementation has a linear behavior with respect to the number of blocks processed, as exactly the same steps are repeated in a loop over the blocks b.

Fig. 6.8 shows the results of the runtime analysis. Whereas Fig. 6.8(a) shows the mean absolute runtime measurement for the processing of a single block of 63 000 observations depending on the number of used cores ($\mathsf{8} \times \mathsf{8} = \mathsf{64}, \ldots, \mathsf{52} \times \mathsf{52} = \mathsf{2\,704}$), Fig. 6.8(b) shows the relative contribution of the three most intensive operations to the total runtime. This three operations are i) the setup of the OEQs (cf. Sect. 6.3.2), ii) the decorrelation (filtering) of the OEQs (cf. Sect. 6.3.3) and iii) the update of the NEQs, i.e. mainly the computation of $\mathbf{A}^T\mathbf{A}$ (cf. Sect. 6.3.4). Two obvious conclusions can be drawn, the least intensive step is the setup of the observation equations, depending on the compute core grid, 1.5 % to 2.5 % of the total runtime are needed. The most intensive operation is the update of the normal equations, mainly the computation of $\mathbf{A}^T\mathbf{A}$. Depending on the compute core grid size 80 % to 90 % of the total runtime are needed. As this operation is (and should be) directly performed with the existing SCALAPACK routine `pdsyrk`, an additional optimization of this step is impossible. The step where optimization is possible is the application of the decorrelation filters, especially the recursive part of the filters. Nevertheless, only 5 % to 15 % of the runtime are spend there, and thus the gain of an optimization is limited. These optimization of course would be an additional specialization to the filter currently used and a tailored optimization would cause a loss of flexibility in trying alternative filters and decorrelation strategies.

Working with parallel implementations, the scaling behavior of the implementation is of major interest. Within the context of HPC, scaling (or speedup) is the factor s the runtime decreases increasing the number of cores by another factor a (e.g. Rauber and Rünger, 2013, Sect. 4.2). Ideally, ignoring additional costs of the parallelization, this behavior is linear, doubling the number of cores should halve the runtime. Due to additional costs as e.g. the required communication of the cores, this ideal scaling could never be reached (e.g. Karniadakis and Kirby, 2003, p. 70f, Bauke and Mertens, 2006, p. 10f). The scaling $s_1(\mathsf{N})$ as a function of totally used cores N can be computed via

$$s_1(\mathsf{N}) = \frac{t_1}{t_{\mathsf{N}}} \tag{6.30}$$

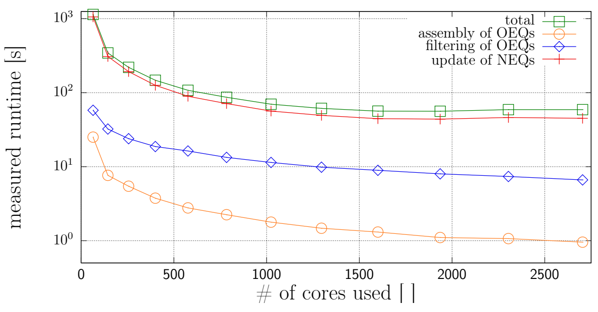
where t_1 is the runtime the algorithm takes using a single core, whereas t_N is the measured runtime using N cores. In case an application cannot be executed on a single core (e.g. due to memory limitations), a reference runtime is defined, e.g. the runtime on the minimal number of cores N_1 required. The scaling is then normalized to that runtime such that

$$s_{\mathsf{N}_1}(\mathsf{N}) = \frac{t_{\mathsf{N}_1}}{t_\mathsf{N}}. \tag{6.31}$$

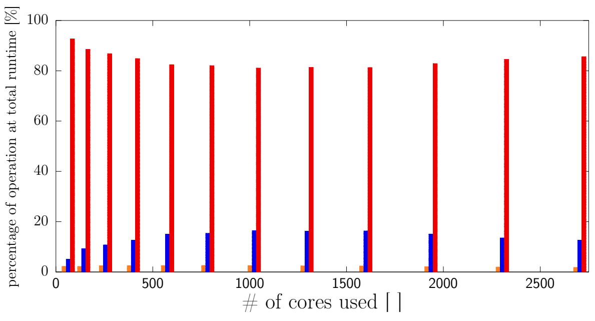
This scaling behavior for the GOCE NEQ assembly as well as for the individual tasks is shown in Fig. 6.8(c). The scaling is normalized to the minimal number of cores needed, i.e. $\mathsf{N} = 64$ for the used data on which the NEQs could be assembled in reasonable time. A nearly linear behavior can be observed going from $\mathsf{N} = 64$ to $\mathsf{N} = 1\,296$ cores. Taking 20.25 times more cores yields a 18.5 times faster runtime. The scaling behavior differs for the three most intensive partial operations. Whereas it is still close to linear for 1 600 cores for the NEQ update (i.e. the $\mathbf{N}+ = \mathbf{A}^T\mathbf{A}$ computation), it starts to get non linear for the OEQ assembly using 784 cores. Due to the implementation of the recursive filtering, the scaling of the filtering is not at all linear in the number of cores, as a real parallel processing is done only for the columns of the compute core grid. Hence, the scaling is better than $\sqrt{\mathsf{N}} = \mathsf{C}$ (grey dashed line in Fig. 6.8(c) and 6.8(d)). Nevertheless, for the overall operation, there is the possibility to use up to 1 600 cores with a benefit for the overall runtime. Using even more cores, e.g. 2 704 there is a negative effect, the overall runtime slightly increases. The additional effort to operate on 2 704 cores is larger than the gain of the computing power of the additional cores. Whereas the runtime for the OEQ assembly and the filtering decreases, it slightly increases of the NEQ update. Thus, the increase of the total runtime is related to the PBLAS `pdsyrk` routine and the required communication therein. Another reason might be the performance of the used BLAS-L3 routines, which perform the local serial computations. It is possible that the locally stored sub-matrices become to small ($\approx 1\,200 \times 1\,200$) to make an efficient use of the serial BLAS-L3 routines. These statements and the drawn conclusions result from an empirical test and are only valid for this exact configuration. E.g. increasing the spherical harmonic resolution and thus the number of unknowns is expected to result in a better scaling behavior for even higher number of cores than tested here.

At some points in Fig. 6.8(c), the empirically derived scaling is better than the ideal scaling which is theoretical impossible. This might have several reasons, e.g. the runtime measurement is never error free. A general hardware dependence is possible. Within each setup, different nodes were involved in the compute core grid (assigned by the cluster management software). The results are normalized to the runtime required on 64 cores, inaccuracies in that runtime measurement are propagated to the scaling. Last but not least, due to the matrix distribution, the size of the local matrices is varies for each compute core grid setup. There are some cases where the local size of the matrices is by chance tailored to the cache memory of the local cores. It is possible that the gain due to a more efficient cache-use (especially by the BLAS-L3 routines) is seen in the scaling curve. Nevertheless, the main conclusion that up to 1 600 cores can be used for this scenario is quite obvious. Finally, Fig. 6.8(d) demonstrates that the poor scaling of the filtering totally depends on the implementation of the recursive filter, which takes most of the runtime of filtering.

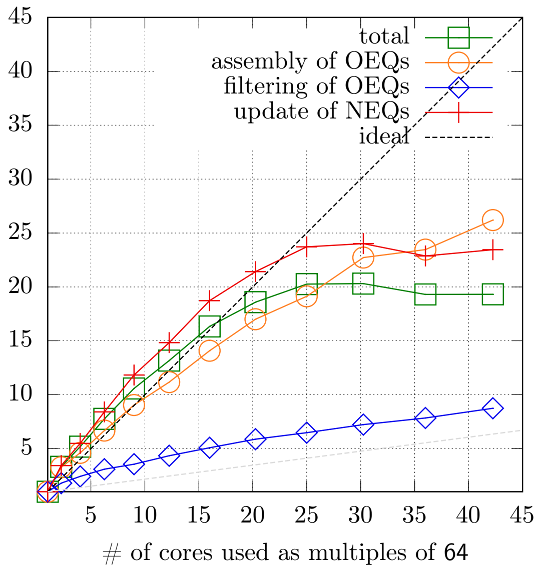
Solution of the NEQs The topic of solving and inverting the NEQs will be addressed for larger systems in Chap. 7. For the sake of completeness and to proof that the solution and inversion of the NEQs is not a problem within GOCE data processing (with respect to runtime) some measurements of the runtime are provided. Nevertheless, the special HPC libraries have to be used to make the system solvable at all. As all matrices are block-cyclic distributed, SCALAPACK functions are directly usable for the solution and inversion. The solution of the NEQs (via Cholesky and forward/backward substitution) is separated from the inversion of the normal equation matrix. The



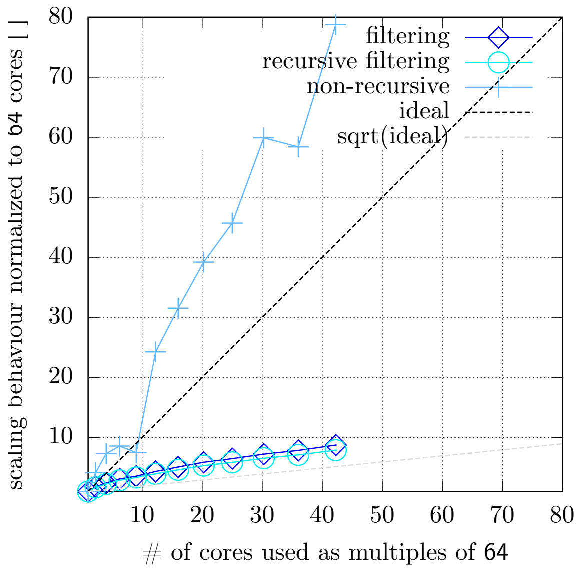
(a) Mean absolute runtime measured for the operations for the processing of a block of 63 000 observations.



(b) Percentage of the three most intensive operations to the total runtime.



(c) Scaling behavior of the operations normalized to the runtime using the $8 \times 8 = 64$ compute core grid.



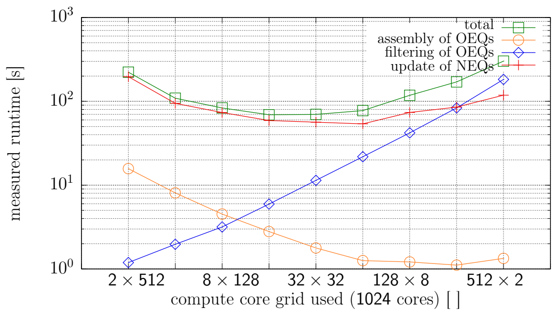
(d) Scaling behavior of the recursive and non-recursive filter implementation normalized to the runtime using the $8 \times 8 = 64$ compute core grid.

Figure 6.8: Graphical depiction of the measured performance of the implemented algorithm. Results are shown for all operations (green) and the three most intensive operations: setup of OEQs (orange), filtering of the OEQs (blue) and the update of the NEQs (red).

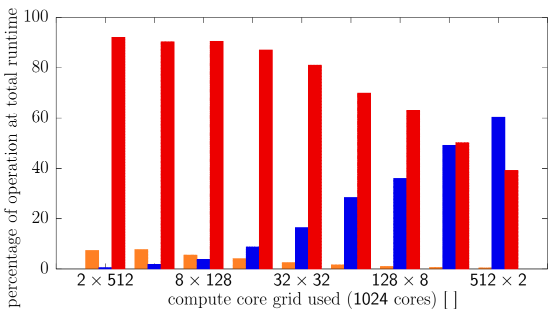
inverse (the covariance matrix of the parameters) is computed from the already Cholesky reduced matrix. The time needed to solve the system of equations is just 290 s on 64 cores and 16 s on 1 600 cores. The inverse is computed in 560 s on 64 cores and 45 s on 1 600 cores. This should just demonstrate, that within this context, deriving the solution and the covariance matrix is less of a challenge (compared to the time for the assembly of the NEQs). Additional details are given in Chap. 7, where larger NEQs are handled and solved.

6.4.2 Analysis of Compute Core Grid (Fixed Distribution Parameters and Fixed Number of Cores)

This section is used to empirically demonstrate, that a quadratic arrangement of the compute core grid is a proper choice for this application and a good compromise for the three major operations. For that reason, the setup is fixed as in Sect. 6.4.1 ($b_r = b_c = 80$, $b_{\text{obs}} = 63\,000$). The number of cores N was fixed to $\mathsf{N} = 1024$, the shape of the compute core grid was varied as shown in Fig. 6.9



(a) Mean absolute runtime measured for the operations for the processing of a block of 63 000 observations.



(b) Percentage of the three most intensive operations to the total runtime.

Figure 6.9: Graphical depiction of the measured performance of the implemented algorithm varying the shape of the grid for the constant number of $\mathsf{N} = 1024$ cores. Results are shown for all operations (green) and the three most intensive operations (set up of OEQs (orange), filtering of the OEQs (blue) and the update of the NEQs (red)).

from $\mathsf{R} \times \mathsf{C} \in \{2 \times 512, 4 \times 128, \ldots, 512 \times 2\}$. Fig. 6.9 shows the total runtime and the measured runtime for the three main operations. The total minimum of the runtime can be observed for the (near) quadratic grids (16×64, 32×32, 64×16. This corresponds to the minimum runtime which is needed for the intensive update of the NEQs (SCALAPACK routine `pdsyrk`). The characteristics of the implementation of the filtering can be seen for the "extrema", i.e. grids close to one dimensional. Whereas the filtering is extremely fast on a 2×512 grid, where only two processor rows are involved (512 columns of the compute core grid filter massive parallel), it is extremely slow on a 512×2 grid, where 512 rows in the grid are involved (which have to wait for each other during the recursive filtering). This behavior is expected, as only columns of the compute core grid perform the recursive filtering in parallel and filtering along the rows is performed sequential. Minimizing the rows of the compute core grid is very well suited for the current implementation of the filtering, but it contradicts the performance of `pdsyrk` and the setup of the OEQs. The latter has the inverse behavior. It is slow for the 2×512 compute core grid, whereas it is fast for the 512×2 compute core grid. This behavior is again plausible, as for the 2×512 compute core grid a lot of redundant computations have to be performed within the recursion formulas for the Legendre functions or a single observation (distribution of columns of $\mathbf{A}$). A large number of columns of the compute core grid C means that the parameters (of a single observation) are extremely distributed, and thus on every core of the grid's row the Legendre functions (for the same point) have to be computed. For the best case, i.e. the 512×2 grid, the Legendre functions for a single point (observation) have to be computed only on two cores, which minimizes redundant computations within the recursion formulas. In summary, Fig. 6.9 shows that a (nearly) quadratic compute core grid is a reasonable general choice, minimizing the total runtime and is thus a compromise for all three major operations involved.

6.4.3 Analysis of Distribution Parameters (fixed Compute Core Grid)

Parameters, which effect the performance of the SCALAPACK computing routines are the block-size of the block-cyclic distribution b_r and b_c (cf. Sect. 3.3). Their choice heavily depends on the used hardware (e.g. processor cache sizes, network connection, ...) and on the SCALAPACK/PBLAS routines used. The optimal choice for a specific application on a specific platform can only be found empirically. Fig. 6.10 shows the runtime analysis fixing all other parameters instead of b_r and b_c. b_r and b_c which are varied, but always choosing quadratic blocks $b_r = b_c$. The number of compute cores is fixed to $\mathsf{N} = 1024$ using a 32×32 compute core grid.

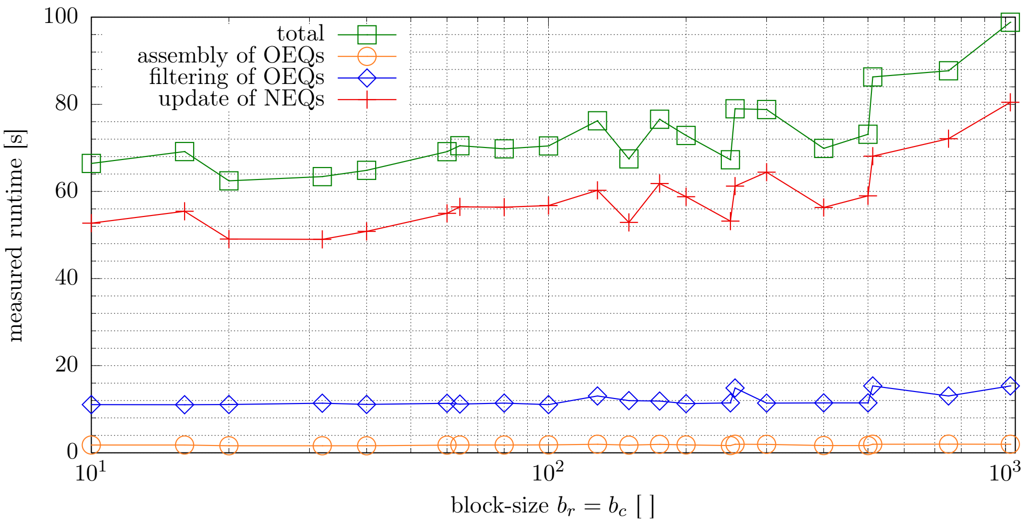


Figure 6.10: Graphical depiction of the measured runtime using different block-sizes $b_r = b_c$ for the block-cyclic matrix distribution. The number of cores is constant $\mathsf{N} = 1024$. Results are shown for all operations (green) and the three most intensive operations (set up of OEQs (orange), filtering of the OEQs (blue) and the update of the NEQs (red)).

Fig. 6.10 shows the measured runtime for all three major tasks, depending on the block size $b_r = b_c$. The runtime is again dominated for the runtime of the NEQ update. Thus, the minimal total runtime corresponds to the minimum runtime of the NEQ update. That is observed for block sizes $b_r = b_c = 20$ and $b_r = b_c = 32$. Except for very large block-sizes the other operations (OEQ setup and filtering) are more or less independent of the block size. For large values of $b_r = b_c$ the load balancing suffers, such that individual cores have local matrices up to b_r additional rows (and b_c more columns) and thus, more observations (and columns) have to be processed locally. The bad load balance can also be seen for the NEQ update for block-sizes larger than 100. All in all, the choice of the block-size between 20 and 100 seems to be a reasonable choice, although the minimum is obtained for 20-32. The range corresponds to the default value of 64 suggested by (Blackford et al., 1997, p. 92). Thus, the chosen value of 80 in Sect. 6.4.1 and 6.4.2 was not ideal choice.

6.5 Results of GOCE Real Data Analysis

As one part of the thesis, the software developed was used to determine GOCE gravity field models from real data. Within the projects GOCE-HPF (ESA) and REAL-GOCE (Geotechnologien-/BMBF) the developed software was used within the determination of the official ESA time-wise gravity field models (EGM_TIM). This section is used to present some of the models, this work significantly contributed to. This section abandons a little from the computational point of view and turns to the results obtained with the presented implementation.

Meanwhile four releases of the time-wise models were officially published, differing in the used data volume, but also some methodical aspects. About 63 000 parameters were estimated from 278 768 016 highly correlated observations for the most recent model, i.e. the EGM_TIM_RL04. Within this context, details are only given for this most recent model. The intermediate results and details always refer to that. The older releases are shown mainly to demonstrate the progress made along the GOCE mission. A final release, covering the data from the complete mission is in the meantime finalized (which is called EGM_TIM_RL05, contains more than 440 000 000 observations and is resolved to d/o 280, 78 957 parameters, see Brockmann et al. (2014a) for details). Some

Table 6.2: Used official GOCE products for gravity field recovery from real data.

	product		
SST	SST_PKI	SST_PCV	
	precise kinematic orbits	covariance of positions (band)	
	observetd position	stochastic model	
SGG	SST_PSO	EGG_NOM_2	EGG_IAQ
	precise science orbits	calibrated gravity grdients	star camera observations
	georeference	raw observations	orientation of gradiometer

results towards that release are shown here in addition. The very preliminary model is named EGM_TIM_RL05p0, and in some figures the close to final model EGM_TIM_RL05p4 is shown.

As the GOCE time-wise gravity field models purely depend on GOCE data and a lot of effort is invested into the modeling of the stochastic behavior of the observations, the models

- are a good demonstrator for the progress along the mission and a demonstrator of the added value of GOCE compared to other gravity field information available,
- are self-consistent,
- come along with a high quality and meaningful full covariance matrix.

6.5.1 Used Data for the Real Data Analysis

For the real data analysis official ESA GOCE Level 1B data was used. Tab. 6.2 summarizes the used products. The data used in the different releases is shown in Fig. 6.11 as a time-line representing the available and used data. The colors represent the coherent gapless segments s the SGG data was divided to and for which individual decorrelation filters were estimated. For each of the segments s, as well as for every used tensor component $g \in G$, individual normal equations $\mathbf{N}_{s,g}$ and $\mathbf{n}_{s,g}$ were assembled.

6.5.2 SST Data and Solutions

As mentioned in Sect. 6.2.1, the SST part enters the solution in terms of already preprocessed NEQs. In Tab. 6.3 some facts about the SST solutions used in the final combined official releases are displayed. In addition, Fig. 6.12 shows degree (error) variances cf. (5.4b) and (5.5b) with respect to the more accurate ITG-Grace2010s model which is used as reference. The progress in the solutions is visible, especially from the first to the fourth release. The progress from release 02 to release 03 is only minor. A huge progress can be observed with the switch of the processing method from release 03 to 04 (cf. Sect. 6.2.1). There are huge improvements over the whole spectral range. In addition, the stochastic model significantly improved when using the alternative short arc integral equation method. Whereas the releases 01 to 03 had to be scaled with a weight of 0.20/0.16/0.11 (already applied in Fig. 6.12), an inverse variance component close to 1.0 was estimated for the release 04 SST NEQs (cf. Sect.6.5.4). Note that using the fourth release SST NEQs, there are still large differences between the empirical and formal degree error variances for the lower degrees $(2 - 40)$. This is expected to result from systematic errors in the kinematic orbits, which do not average out when integrating over a large time span. This systematic errors cannot be modeled stochastically and thus enter the solution. Current investigations of the experts in the generation of the kinematic satellite orbits seem to identify errors which are related to the ionosphere. They enter the kinematic orbits and thus the gravity field (SST) solution (Bock et al., 2014).

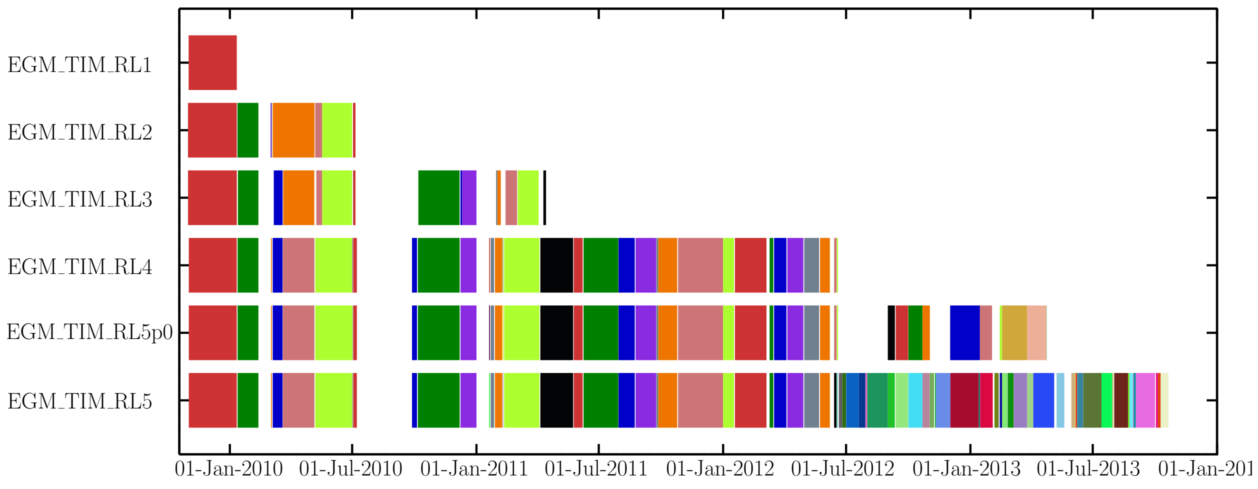


Figure 6.11: Data segments used in the official GOCE releases and in addition the data used in the current preliminary version EGM_TIM_RL05p0 and the overall mission data which will be included in release 5. The colors represent coherent gapless segments. Note that some segments are so short that they are not visible in the figure.

Table 6.3: Some details on the SST solutions used within the real data processing. The NEQs were provided by project parteners from ITSG, TU Graz.

release	method	timespan	d/o	U_n
EGM_TIM_RL01	energy balance	11/2009–01/2010	2–100	10 097
EGM_TIM_RL02	energy balance	11/2009–07/2010	2–100	10 097
EGM_TIM_RL03	energy balance	11/2009–04/2011	2–100	10 097
EGM_TIM_RL04	short arc integral equation	11/2009–06/2012	2–130	17 157
EGM_TIM_RL05p0	short arc integral equation	"	"	"
EGM_TIM_RL05	short arc integral equation	11/2009–10/2013	2–150	22 797

The full normal equations which are solved individually here, are used in Sect. 6.5.4 within a combination with the SGG NEQs assembled within this work. Fig. 6.12 gives a first idea about the accuracy of the SST solutions, which are at the $1-2$ mm level only for the spherical harmonic degrees $2-20/30/50$ (depending on the release). The contributions of the SST part to the final solutions is studied in detail in Sect. 6.5.4.

6.5.3 SGG Observations and Solutions

This sections shows the results from the processing of the SGG observations, derived within this work. The available gradiometer data with a sampling rate of 1 Hz was partioned into continuous, gapless data segments which are assumed to be independent (cf. Fig. 6.11 and Tab. 6.4). This assumption holds, as there will be a new filter warmup at the beginning of each new segment, such that correlations are removed. The observations used for filter warmup (typically 1 000 to 5 000 at the beginning of a segment, depending on the adjusted filters) cannot be used within the adjustment. New segments are artificially introduced if a significant change in the gradiometer noise characteristics can be observed, or if there are artifacts like jumps in the observation/residual time series. For the most recent model, the data segments used within the processing are shown in Fig. 6.11. The nomenclature of the official ESA releases is used, as the results derived within this

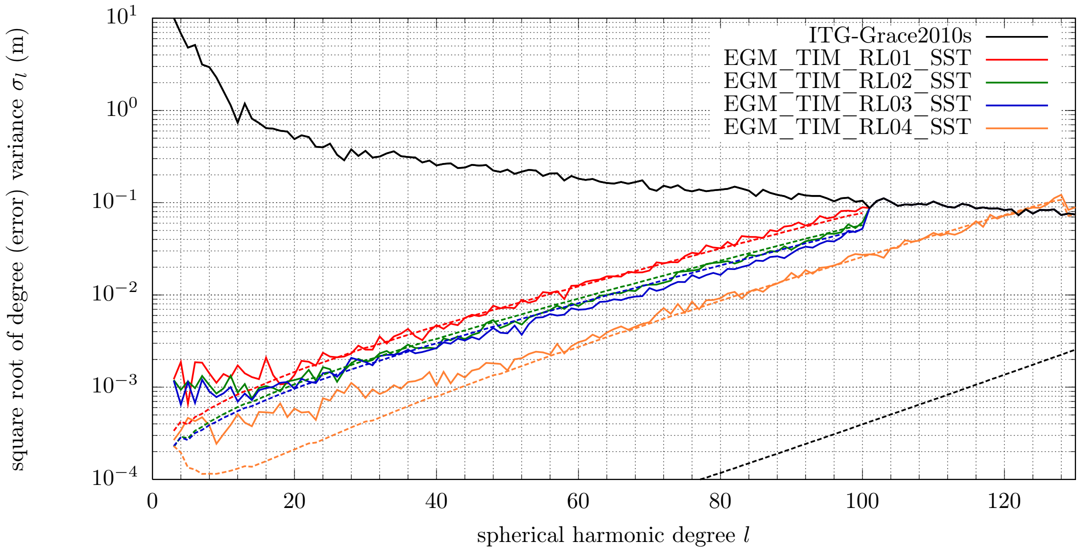


Figure 6.12: Degree (error) variances of the four used SST solutions with respect to the superior ITG-Grace2010s model. Degree error variances estimated from coefficient differences are shown in solid lines whereas the error variances computed from formal errors are shown in dashed lines.

Table 6.4: Some details of the SGG observations used within the real data processing. M is the number of observations.

release	components G	timespan	d/o	U	S	M
EGM_TIM_RL01	xx, yy, zz	11/2009–01/2010	2–224	50 172	1	$3 \cdot 6\,161\,834$
EGM_TIM_RL02	xx, yy, zz	11/2009–07/2010	2–250	62 997	9	$3 \cdot 19\,477\,946$
EGM_TIM_RL03	xx, xz, yy, zz	11/2009–04/2011	2–250	62 997	16	$4 \cdot 31\,289\,605$
EGM_TIM_RL04	xx, xz, yy, zz	11/2009–06/2012	2–250	62 997	41	$4 \cdot 69\,692\,004$
EGM_TIM_RL05p0	xx, xz, yy, zz	11/2009–05/2013	2–250	62 997	51	$4 \cdot 86\,336\,504$
EGM_TIM_RL05*	xx, xz, yy, zz	11/2009–10/2013	2–280	78 957	87	$4 \cdot 109\,799\,264$

*The numbers provided for release 5 are preliminary, as the processing is ongoing during the preparation of this thesis.

work directly entered the official solutions. In addition, the data segments as derived for the older models are shown. The gaps visible are outages from the satellite or indicate missing data due to satellite problems or maneuvers. All in all, the data was divided into 41 segments of different length as summarized in Tab. 6.4.

6.5.3.1 Estimating Decorrelation Filters and Outliers for the SGG Observations

The results given within this Section are only related to the fourth release. Nevertheless the procedure is the same for the older releases and the final release 5. As mentioned in Sect. 6.1 the SGG data are highly correlated. The first step for the data-adaptive estimation of decorrelation filters is to derive an estimate for the gradiometer noise, individually for each data segment s and each tensor component g. This is done within an iterative full gravity field adjustment:

1. Estimate an initial gravity field from the SGG observations, using either (i) a-priori decorrelation filters (e.g. from pre-mission simulations, estimated for an older release) or (ii) decorrelation filters estimated from residuals with respect to an existing gravity field model (e.g. from GRACE or older GOCE models)

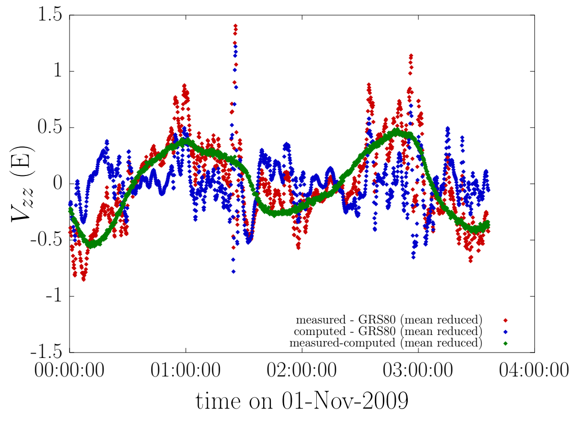
2. Compute SGG residuals $\mathbf{v}_{g,s} = \mathbf{A}_{g,s}\tilde{\mathbf{x}} - \boldsymbol{\ell}_{g,s}$ as realization of the gradiometer noise and adjust ARMA filters for every segment and component (c.f. Schuh et al., 2006, Krasbutter et al., 2011a, Siemes, 2008, Chap. 5).
3. Introduce new segments s if a change in the noise characteristics or huge jumps in the data can be observed. The segment concerned is split into two, and thus an additional segment is inserted.
4. Use the filtered residuals to identify large outliers (account for filtering effects like e.g. smearing).
5. Repeat the full scale gravity recovery starting at 1). If the filters and the outlier detection converge, stop the iterative procedure.

The result of this so called tuning steps are estimates for the decorrelation filter models and flag information on outliers found in the data. As, depending on the data quality and the initial filter estimates, the procedure has to be repeated up to thirty times (including iterative outlier identification in decorrelated residuals), this iterative procedure is performed with the existing fast iterative conjugate gradient based PCGMA solver (Preconditioned Conjugate Gradient Multiple Adjustment, tailored for the GOCE case, cf. Schuh, 1996, Boxhammer, 2006, Brockmann et al., 2010). PCGMA is only used for the so called tuning-process (e.g. Brockmann et al., 2010) to derive outliers and decorrelation filters and afterwards the newly developed massive-parallel solver is used to assemble the full NEQs, estimate the weights and solve for the spherical harmonic coefficients.

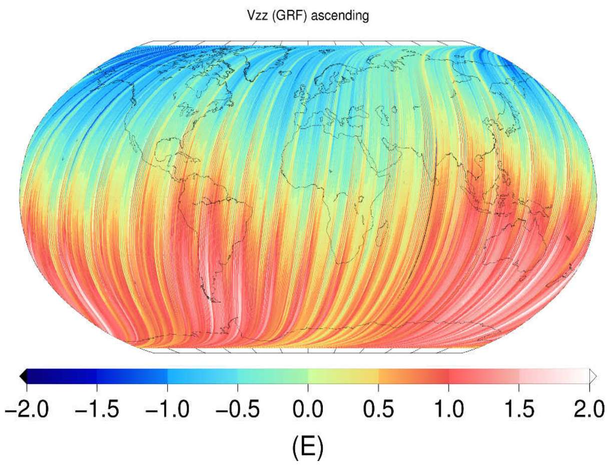
To get an idea of the gradiometer noise characteristics, Fig. 6.13(a) illustrates a short part of the time-series including the expected V_{ZZ} signal computed from a model (blue line), the V_{ZZ} measurements itself and a first guess for the noise, i.e. the difference measured minus computed V_{ZZ} gravity gradients. The measurements, as well as the synthesized gravity gradients are reduced by the normal potential from the GRS80, all quantities – the noise estimate as well – are in addition reduced by a mean value. The basic conclusions of the figures are: i) the signal and the noise are in the same order of magnitude, but ii) the noise has a very long-wavelength characteristic whereas the signal to be recovered is of high frequency. This is in addition demonstrated by Fig. 6.13(b), where the noise estimates are plotted for ascending orbits exemplarily for the V_{ZZ} component in the spatial domain.

Another possibility to show the gradiometer noise characteristics is the illustration in the spectral domain. The square root of the power spectral density (PSD) can be compared to the PSD of the inverse filter, which makes both comparable. Fig. 6.14 shows two examples of the PSD of a filter (exemplarily for the V_{ZZ} component for segment $s = 0$), approximating the gradiometer noise in two different ways. The first filter is quite simple, it approximates the main characteristics, the flat behavior within the MBW and the increasing power for the long-wavelengths. The second filter is more complex with additional numerical requirements, which in addition approximates the peaks of the noise occurring at multiples of the orbital frequencies. Although the second filter is a better approximation, these filters are not used in the processing, as there are numerical instabilities and a large data loss due to the large warmup of the involved Notch filters, which model the peaks (e.g. Schuh et al., 2010).

Fig. 6.15 summarizes all finally estimated filters for the 41 segments the available SGG data of the 04th release were divided into. Shown are the converged filters which were used in this work for the final NEQ assembly. The colors represent the different segments. For XX (cf. Fig 6.15(a) and 6.15(b)) and XZ (cf. Fig 6.15(c) and 6.15(d)) the gradiometer noise seems to be very stable over the whole time-span used. All estimated filters are similar. Nevertheless, there are some variations outside the MBW, in the low frequency part of the spectrum. Nearly the same holds true for the ZZ component with two exceptions (cf. Fig 6.15(g) and 6.15(h)), one is the very first segment ($s = 0$) which has a worse performance. This segment contains the very first data before



(a) Gradiometer noise for the V_{ZZ} component as time series along the orbit (green, mean reduced). As comparison synthesized gravity gradients signal (blue, mean and GRS80 reduced) and the raw observations (red, mean and GRS80 reduced), Pail et al. (2011a).



(b) Gradiometer noise for the V_{ZZ} component for ascending satellite tracks in the spatial domain. The picture is dominated by the long wavelength once per revolution error. Shown is the noise estimate for the data from 11/2009.

Figure 6.13: Illustration of the gradiometer noise estimates, in the time domain as well as in the spatial domain.

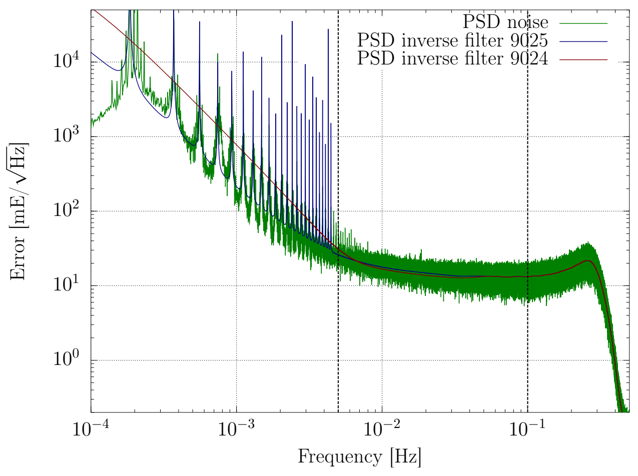


Figure 6.14: PSD of estimated gradiometer noise for segment $s = 0$ and tensor component $c = ZZ$. In addition, decorrelation filters are shown in terms of their inverse PSD. A simple filter used in the standard processing approximating the major characteristics in the spectrum (no. 9024) and a more complex one, approximating the characteristic peaks at the multiples of the once per revolution frequency (no. 9025).

an anomaly of the satellite. After the anomaly, the performance of the gradiometer, especially the performance of the ZZ component improved. The anomaly required a switch from the main on-board computer to the redundant part, but it is not understood if that was the reason for the performance improvement. The second segment for which the filter significantly differs, is a segment close to the end of the time-series. It is a short segment where the estimation of the decorrelation filter is not very stable. In addition, with the end of the used time series, the noise became more and more non-stationary (stationarity is indirectly assumed using this kind of decorrelation filters) due to increased solar activity and thus a rougher environment around the satellite. That is the reason why the performance of the YY component seemingly worsens over time (cf. Fig 6.15(e) and

6.15(f)). Analyzing the data in detail, it is again a geographically correlated increase of the noise around the magnetic poles which leaks into the MBW when computing the spectrum. Outside this geographical regions, the noise characteristics of the gradiometer is quite stable (e.g. Siemes et al., 2013). This is confirmed by Fig. 6.16 which shows (some) of the identified outliers (from the filtered residuals) which obviously correlate with the magnetic poles.

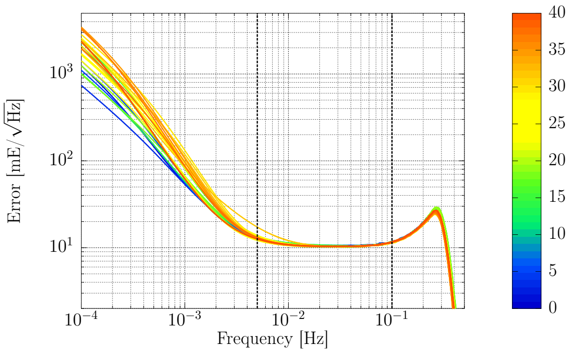
6.5.3.2 Component- and Segment-wise Solutions

In the software developed, the decorrelation filters estimated (cf. Sect. 6.5.3.1) and the outliers identified were used to assemble the SGG normal equations for all $S = 41$ segments for all four accurately measured tensor components XX, XZ, YY and ZZ. Thus, $4 \cdot 41$ NEQs were assembled for spherical harmonic d/o 2 to d/o 250. That results in 164 NEQs, each of them requiring about 30GB disk space. Before they are combined, the individual NEQs were solved (if possible, i.e. if positive definite, i.e. if the data segment is long enough) to derive sub-solutions which can be used to demonstrate the consistency of the solution. Fig. 6.17 shows the segment wise and component wise solutions for all $4 \cdot 21$ NEQs individually solvable in terms of degree error variances with respect to the finally combined and thus superior EGM_TIM_RL04 solution and the corresponding degree error estimates from the formal error estimates. For most of the solvable segments, a nice agreement between the degree error variances and the empirical degree error variances can be observed.

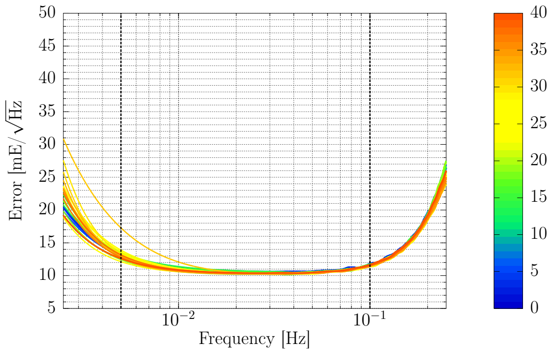
To demonstrate the consistency in more detail, Fig. 6.18 focuses on two segments, which show the individual solutions in terms of degree error variances with respect to the finally combined EGM_TIM_RL04 model, which should be superior due to a much larger amount of input data. The empirically estimated degree error variances and the degree variances computed from formal errors again agree very well, which shows that the error information is meaningful and reflects the error of the derived model. Whereas a sub-solution from a long segment is chosen in Fig. 6.18(a), a solution from a short segment (approximately 14 days) is shown in Fig. 6.18(b). Of course the second solution is of poor quality, but this is correctly reflected by the formal error estimates. Thus, within the final combination, the NEQs of the segments enter the combination with a proper weight matrix and produce a correctly weighted combined solution.

6.5.4 Combined Solutions

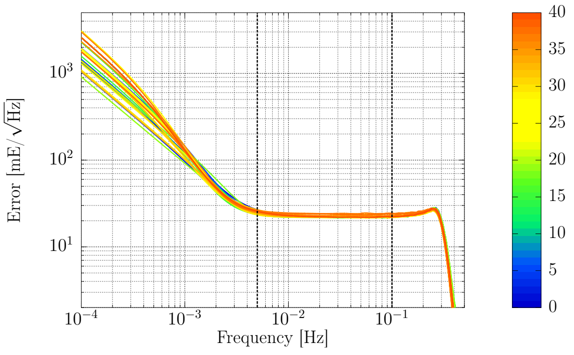
The normal equations of the individual segments and tensor components have to be combined to derive an optimal solution from all observations collected. A joint weighted system of NEQs is computed cf. (6.1) adding the SST NEQ matrix and the diagonal Kaula regularization matrices. Variance component estimation cf. Sect. 4.2 is used to iteratively estimate weights for the individual NEQs of the segments and of the tensor components. Tab. 6.5 shows the estimated weights for all involved 164 SGG NEQs, the two regularization matrices and the SST NEQ. As the estimated filters account for a variance of unit weight, most of the estimated weights are close to 1.0. That means, the filter estimates are reasonable and the observations of the segments and components are consistent among each other. Some exceptions can be observed. For some segments and some components the weight is estimated close to 0.0, these values are highlighted in red. These segments are all very short, e.g. some hours only, with a lot of identified outliers, which correspond to satellite maneuvers. These segments contain outliers and the observations are not supported by the other segments. They are identified via the VCE as unusable and the weight obtained is close to zero, corresponding to a huge variance. There are some remaining segments/components, with estimated weights significantly different from 0.0 and from 1.0. They are highlighted in green in Tab. 6.5. Most of these segments are again very short. Mostly they are too short to derive a stable estimate for an individual decorrelation filter. For those segments, a standard filter is used, which is not perfectly



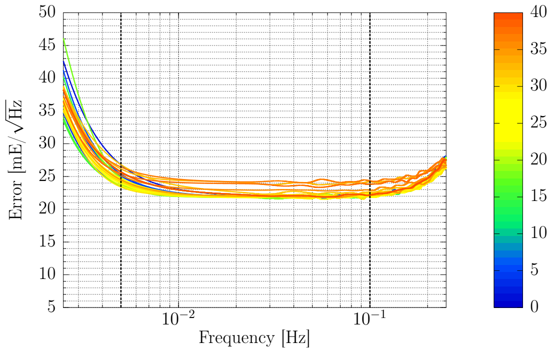
(a) Used filters for the XX component (logarithmic scale).



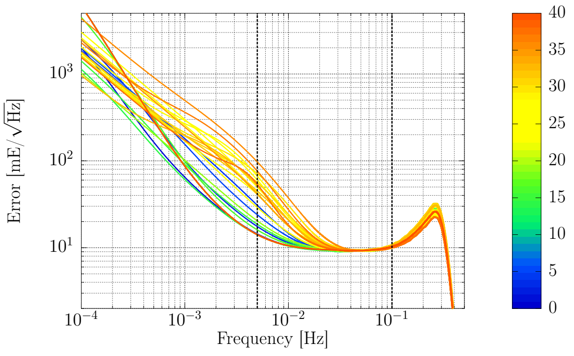
(b) Used filters for the XX component, linear plot with focus on the MBW.



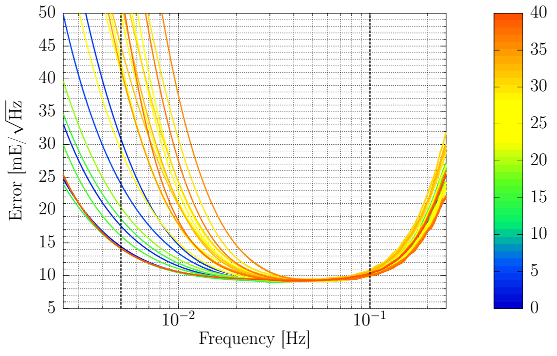
(c) Used filters for the XZ component (logarithmic scale).



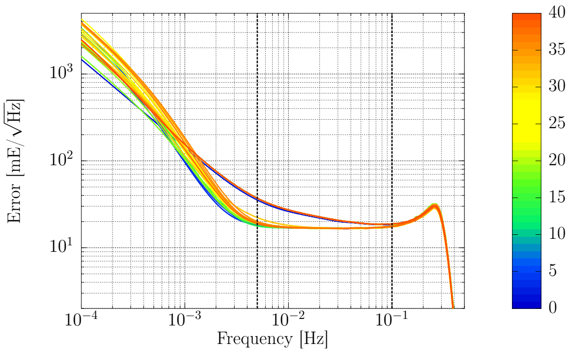
(d) Used filters for the XZ component, linear plot with focus on the MBW.



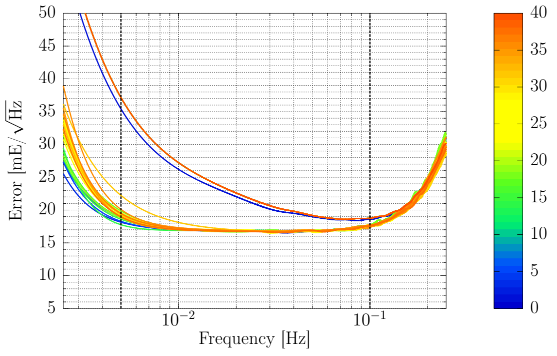
(e) Used filters for the YY component (logarithmic scale).



(f) Used filters for the YY component, linear plot with focus on the MBW.

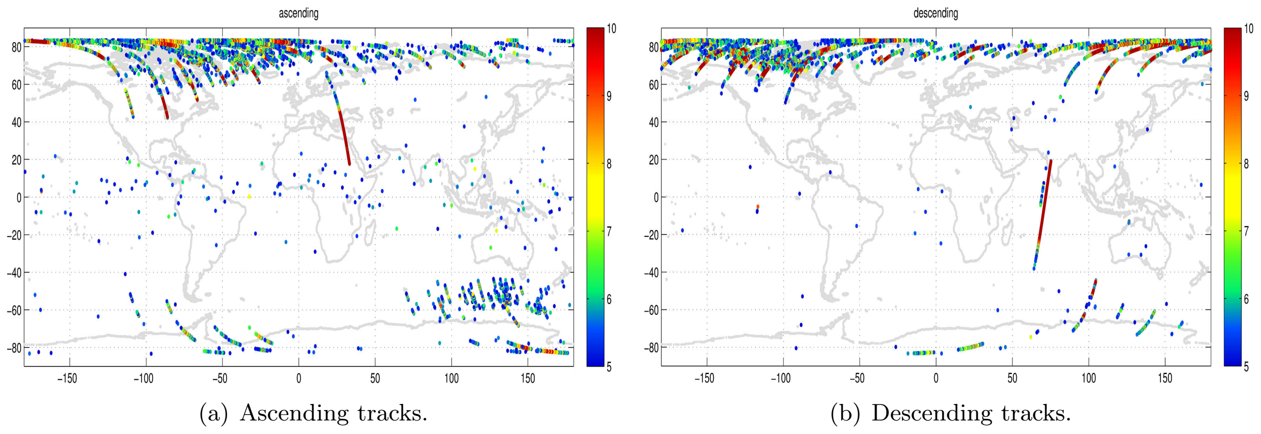


(g) Used filters for the ZZ component (logarithmic scale).



(h) Used filters for the ZZ component, linear plot with focus on the MBW.

Figure 6.15: Illustration of the used filters in the spectral domain.



(a) Ascending tracks. (b) Descending tracks.

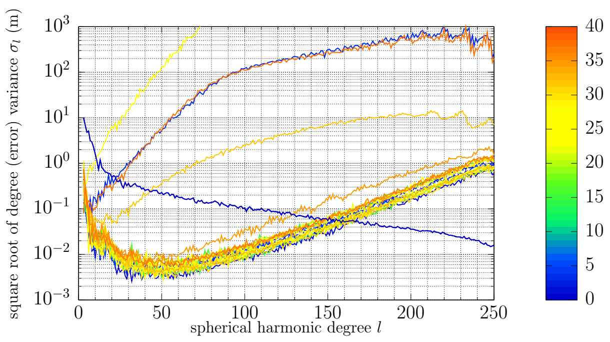
Figure 6.16: Some identified outliers in the data, Many of them show an obvious correlation to the magnetic poles.

suited for the real observations in the segment. The estimated weights adjust this standard filters to a better fit. Within the spectrum, the weighting is nothing else than a shift of the filter. The spectral shape cannot be changed by VCE. Some differences from 1.0 result from the fact that a decorrelation filter is always adjusted only to parts of the data from the segment, e.g., if the segments contain outliers, if there is a slightly other behavior of the noise as in the rest of the segments, VCE derives a weight to adjust these differences.

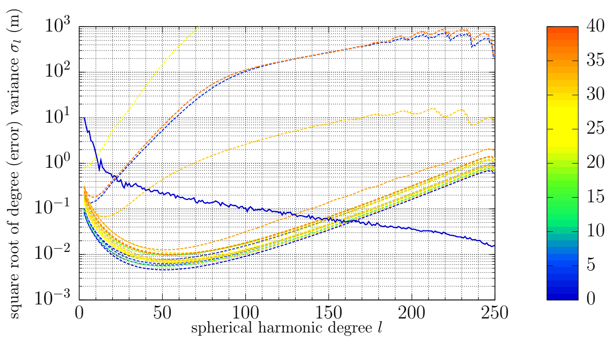
Starting with the weight 1.0 for all involved NEQs, three iterations of VCE are performed to derive weights converging on the third digit (cf. 6.5). For that procedure, a combined NEQ is set up using the weights from the former iteration and solve for the parameters and an new estimate of weights cf. (4.7), (4.8a) and (4.9).

The weights after the third iteration shown in Tab. 6.5 are used to combine a single gradiometer NEQ, and to finally combine the SGG NEQs with the SST and regularization matrices (using the weights estimated) to derive the final solution. This NEQs are recalled from disk, combined in a weighted addition and solved for the unknown coefficients. For details on the implementation see Chap. 7, where the same procedure is used but for higher resolution models (e.g. how the matrices in different parameter ordering and differing resolution can be combined). Fig. 6.19 shows the SST, the SGG as well as the combined solution in terms of degree variances compared to the (in the shown lower degrees) superior ITG-Grace2010s model. It gives a first idea how and where the different observation groups contribute to the final solution. The main contribution of the SST part is from degree 2 to degree 10 and the main contribution of the SGG part is from degree 40 to degree 220. The high degree regularization significantly starts to act at degree 220.

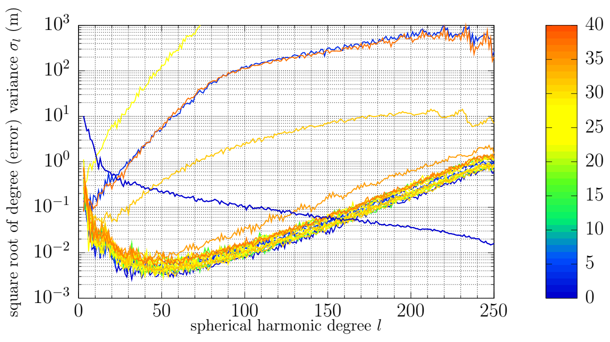
This contribution of the individual normal equations can be shown in an approximation via partial redundancies. They can be computed for an observation group i via (5.9). Plotting this contributions in a coefficient triangle clearly shows, where the different observation groups have their strengths and weaknesses. This can be seen for the observation groups used within EGM_TIM_RL04 in Fig. 6.20. The use of HPC makes such analyses and computations very simple. The sensitivities of the different observation groups on the level of spherical harmonics is clearly visible. In addition, the minor contribution of V_{XZ} is shown, a component, which observes the lowest signal. Compared to the analysis in terms of degree variances, a more detailed picture of the contributions is visible. E.g., it can be shown, that the SST observations contribute up to degree 100 for specific sectorial coefficients. In addition, the coefficients which are mainly determined by the regularization can be easily identified in Fig. 6.20(e) and 6.20(f).



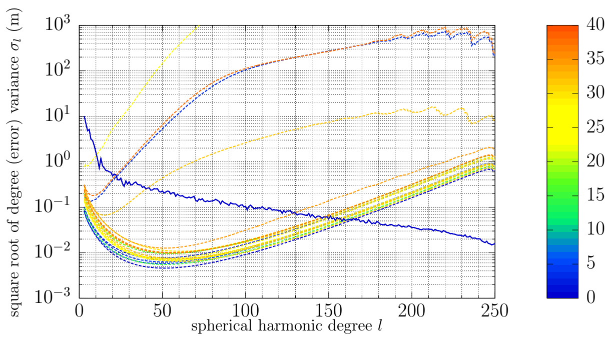
(a) Degree error variances from coefficient differences for the XX-component solutions of solvable segments.



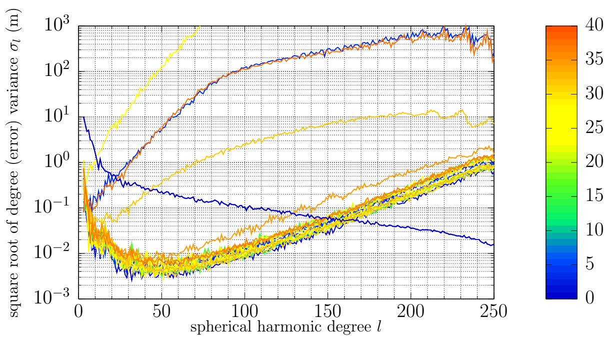
(b) Degree error variances from formal errors for the XX-component solutions of solvable segments.



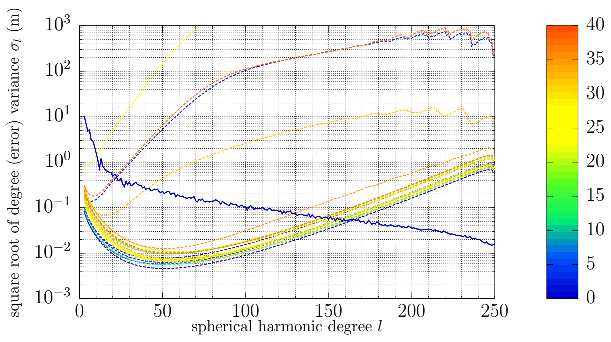
(c) Degree error variances from coefficient differences for the XZ-component solutions of solvable segments.



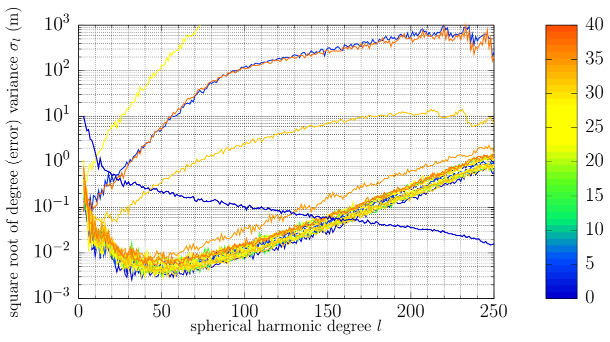
(d) Degree error variances from formal errors for the XZ-component solutions of solvable segments.



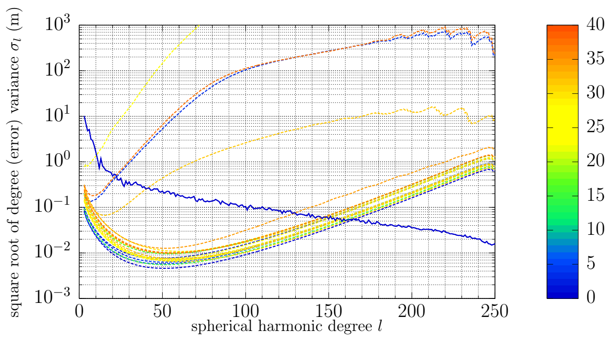
(e) Degree error variances from coefficient differences for the YY-component solutions of solvable segments.



(f) Degree error variances from formal errors for the YY-component solutions of solvable segments.



(g) Degree error variances from coefficient differences for the ZZ-component solutions of solvable segments.



(h) Degree error variances from formal errors for the ZZ-component solutions of solvable segments.

Figure 6.17: Segment-wise and component-wise SGG-only solutions with respect to EGM_TIM_RL04 in terms of degree variances.

Table 6.5: Estimated weights for the EGM_TIM_RL04 NEQs after 3 VCE iterations.

(a) Weights for V_{XX} NEQs.

s	$w^{(0)}$	$w^{(1)}$	$w^{(2)}$	$w^{(3)}$
0	1.000	0.980	0.973	0.973
1	1.000	1.041	1.024	1.024
2	1.000	0.912	0.905	0.905
3	1.000	1.018	0.992	0.992
4	1.000	0.946	0.937	0.937
5	1.000	0.980	0.965	0.965
6	1.000	0.000	0.000	0.000
7	1.000	0.000	0.000	0.000
8	1.000	0.000	0.000	0.000
9	1.000	0.000	0.000	0.000
10	1.000	0.000	0.000	0.000
11	1.000	0.000	0.000	0.000
12	1.000	0.000	0.000	0.000
13	1.000	0.580	0.577	0.577
14	1.000	1.003	0.986	0.986
15	1.000	1.026	1.018	1.018
16	1.000	1.019	1.014	1.014
17	1.000	0.853	0.848	0.848
18	1.000	1.018	1.004	1.004
19	1.000	1.015	1.004	1.004
20	1.000	0.446	0.445	0.445
21	1.000	0.449	0.448	0.448
22	1.000	0.427	0.426	0.426
23	1.000	1.000	0.996	0.996
24	1.000	1.104	1.098	1.098
25	1.000	1.001	0.996	0.996
26	1.000	0.984	0.979	0.979
27	1.000	1.008	1.002	1.002
28	1.000	1.001	0.991	0.991
29	1.000	0.616	0.613	0.613
30	1.000	1.002	0.999	0.999
31	1.000	1.020	1.016	1.016
32	1.000	1.023	1.016	1.016
33	1.000	1.088	1.084	1.084
34	1.000	1.119	1.110	1.110
35	1.000	1.000	0.998	0.998
36	1.000	1.070	1.067	1.067
37	1.000	1.022	1.018	1.018
38	1.000	1.050	1.045	1.045
39	1.000	0.014	0.014	0.014
40	1.000	1.109	1.100	1.100

(b) Weights for V_{YY} NEQs.

s	$w^{(0)}$	$w^{(1)}$	$w^{(2)}$	$w^{(3)}$
0	1.000	1.000	0.998	0.998
1	1.000	1.020	1.013	1.013
2	1.000	1.060	1.058	1.058
3	1.000	1.004	1.001	1.001
4	1.000	0.975	0.974	0.974
5	1.000	1.009	1.008	1.008
6	1.000	0.001	0.001	0.001
7	1.000	0.001	0.001	0.001
8	1.000	0.001	0.001	0.001
9	1.000	0.001	0.001	0.001
10	1.000	0.003	0.003	0.003
11	1.000	0.001	0.001	0.001
12	1.000	0.001	0.001	0.001
13	1.000	0.558	0.557	0.557
14	1.000	1.001	0.991	0.991
15	1.000	1.022	1.017	1.017
16	1.000	0.977	0.976	0.976
17	1.000	0.862	0.860	0.860
18	1.000	0.929	0.928	0.928
19	1.000	1.004	0.998	0.998
20	1.000	0.221	0.221	0.221
21	1.000	0.145	0.145	0.145
22	1.000	0.273	0.273	0.273
23	1.000	0.949	0.947	0.947
24	1.000	1.018	1.017	1.017
25	1.000	1.006	1.005	1.005
26	1.000	0.992	0.990	0.990
27	1.000	1.015	1.014	1.014
28	1.000	1.073	1.071	1.071
29	1.000	0.371	0.370	0.370
30	1.000	1.045	1.042	1.042
31	1.000	1.103	1.100	1.100
32	1.000	0.997	0.996	0.996
33	1.000	1.049	1.044	1.044
34	1.000	1.002	1.000	1.000
35	1.000	0.878	0.877	0.877
36	1.000	1.002	1.001	1.001
37	1.000	0.985	0.984	0.984
38	1.000	0.971	0.968	0.968
39	1.000	0.216	0.216	0.216
40	1.000	0.964	0.962	0.962

(c) Weights for V_{ZZ} NEQs.

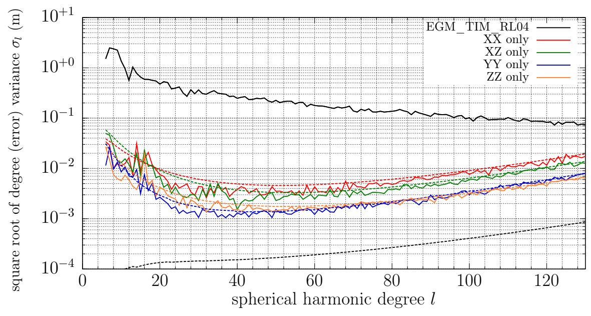
s	$w^{(0)}$	$w^{(1)}$	$w^{(2)}$	$w^{(3)}$
0	1.000	0.967	0.958	0.958
1	1.000	1.048	1.030	1.030
2	1.000	1.128	1.116	1.116
3	1.000	1.046	1.034	1.034
4	1.000	1.044	1.035	1.035
5	1.000	1.040	1.032	1.032
6	1.000	0.000	0.000	0.000
7	1.000	0.000	0.000	0.000
8	1.000	0.000	0.000	0.000
9	1.000	0.000	0.000	0.000
10	1.000	0.000	0.000	0.000
11	1.000	0.000	0.000	0.000
12	1.000	0.000	0.000	0.000
13	1.000	0.880	0.872	0.872
14	1.000	1.033	1.024	1.024
15	1.000	1.047	1.039	1.039
16	1.000	1.031	1.024	1.024
17	1.000	1.129	1.117	1.117
18	1.000	1.057	1.039	1.039
19	1.000	1.013	1.001	1.001
20	1.000	0.713	0.708	0.708
21	1.000	0.917	0.915	0.915
22	1.000	0.731	0.725	0.725
23	1.000	1.081	1.074	1.074
24	1.000	1.041	1.037	1.037
25	1.000	1.001	0.997	0.997
26	1.000	1.053	1.044	1.044
27	1.000	1.080	1.072	1.072
28	1.000	1.033	1.025	1.025
29	1.000	0.737	0.731	0.731
30	1.000	1.060	1.053	1.053
31	1.000	1.061	1.055	1.055
32	1.000	1.046	1.037	1.037
33	1.000	1.090	1.081	1.081
34	1.000	1.120	1.109	1.109
35	1.000	1.030	1.026	1.026
36	1.000	1.049	1.045	1.045
37	1.000	1.060	1.055	1.055
38	1.000	1.056	1.046	1.046
39	1.000	1.227	1.213	1.213
40	1.000	1.197	1.184	1.184

(d) Weights for V_{XZ} NEQs.

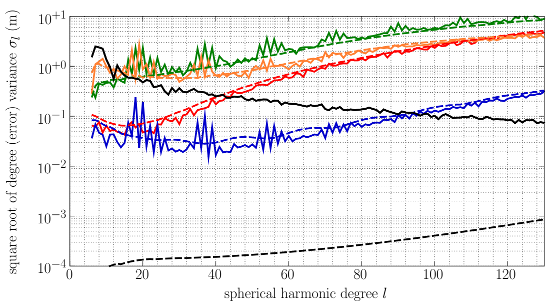
s	$w^{(0)}$	$w^{(1)}$	$w^{(2)}$	$w^{(3)}$
0	1.000	0.997	0.994	0.994
1	1.000	1.054	1.044	1.044
2	1.000	0.895	0.892	0.892
3	1.000	1.016	1.002	1.001
4	1.000	1.024	1.016	1.016
5	1.000	1.031	1.026	1.026
6	1.000	0.003	0.003	0.003
7	1.000	0.003	0.003	0.003
8	1.000	0.002	0.002	0.002
9	1.000	0.002	0.002	0.002
10	1.000	0.004	0.004	0.004
11	1.000	0.000	0.000	0.000
12	1.000	0.003	0.003	0.003
13	1.000	1.396	1.390	1.390
14	1.000	0.948	0.940	0.940
15	1.000	0.974	0.968	0.968
16	1.000	0.998	0.994	0.994
17	1.000	0.847	0.844	0.844
18	1.000	1.102	1.093	1.093
19	1.000	1.021	1.002	1.002
20	1.000	1.647	1.638	1.638
21	1.000	1.829	1.817	1.818
22	1.000	1.871	1.859	1.859
23	1.000	1.027	1.020	1.020
24	1.000	1.099	1.096	1.096
25	1.000	1.012	1.007	1.007
26	1.000	0.932	0.923	0.923
27	1.000	0.992	0.987	0.987
28	1.000	1.036	1.030	1.030
29	1.000	1.793	1.782	1.782
30	1.000	0.891	0.880	0.880
31	1.000	1.064	1.054	1.054
32	1.000	1.111	1.103	1.103
33	1.000	1.175	1.156	1.156
34	1.000	1.067	1.063	1.063
35	1.000	0.961	0.956	0.956
36	1.000	1.097	1.092	1.092
37	1.000	1.052	1.049	1.049
38	1.000	0.995	0.990	0.990
39	1.000	0.275	0.275	0.275
40	1.000	1.281	1.275	1.275

(e) Weights for SST/REG NEQs.

s	$w^{(0)}$	$w^{(1)}$	$w^{(2)}$	$w^{(3)}$
SST	1.000	0.923	0.997	0.997
REG_1	1.000	0.680	0.691	0.693
REG_2	1.000	0.716	0.716	0.716

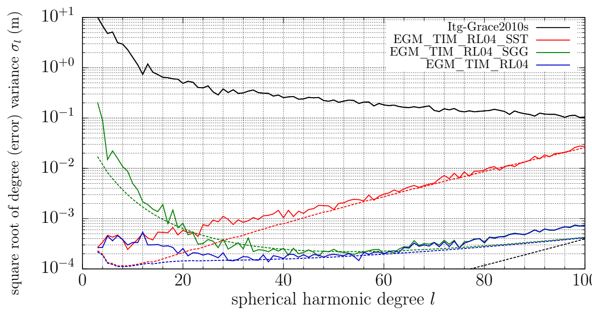


(a) Degree error variances for segment $s = 0$ (long).

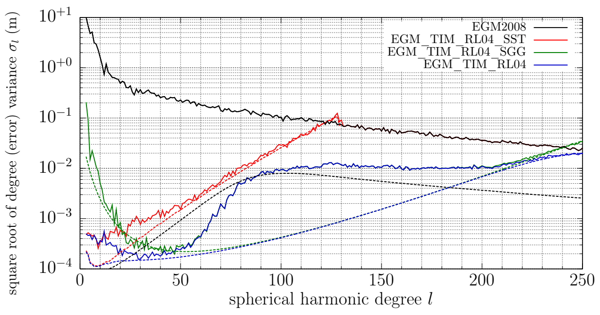


(b) Degree error variances for segment $s = 32$ (short).

Figure 6.18: Illustration of degree variances of two selected segments with respect to EGM_TIM_RL04. The agreement of empirical and formal error estimates should be shown.



(a) Solution with respect to ITG-Grace2010s.



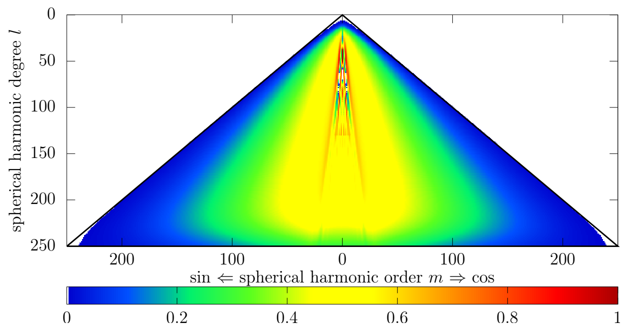
(b) Solution with respect to EGM2008.

Figure 6.19: Degree error variances of the SST, SGG and combined GOCE EGM_TIM_RL04 solution for the lower degrees with respect to the ITG-Grace2010s model, which is assumed to be superior up to degree 70 and with respect to the combined model EGM2008 which is assumed to be superior in the higher degrees above d/o 210. Between degree 70 and 210 GOCE models are the most accurate global gravity field models and can thus not be validated with simple model comparison techniques.

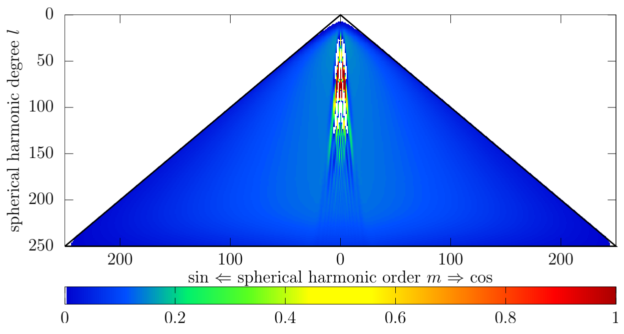
6.5.5 Model Comparison and Validation

The method and implementation summarized in this chapter was used to estimate the GOCE-only models of the so called time-wise approach. As stated earlier, four official models were computed and released. In addition, a fifth (EGM_TIM_RL05p0) and a sixth model (and EGM_TIM_RL05p4) was computed within the preparation of the fifth release, including NEQs of three nearly completed cycles of a lower mean satellite orbit and the additional data towards mission end. The observations of these cycles have a better signal to noise ratio for the higher frequencies and thus improve the estimated models mainly in the higher spherical harmonic degrees. All six time-wise solutions computed so far are shown in terms of degree variances in Fig. 6.21 with respect to the ITG-Grace2010s and the EGM2008 model. In spectral regions, where these models are assumed to be superior, the improvements along the releases are clearly visible (degree 2–60 for ITG-Grace2010s, degree 2–40 and 220–250 for EGM2008) in the empirical degree error variances (solid lines). For all other spectral ranges the improvements are visible in the degree variances computed from the formal errors (dashed lines).

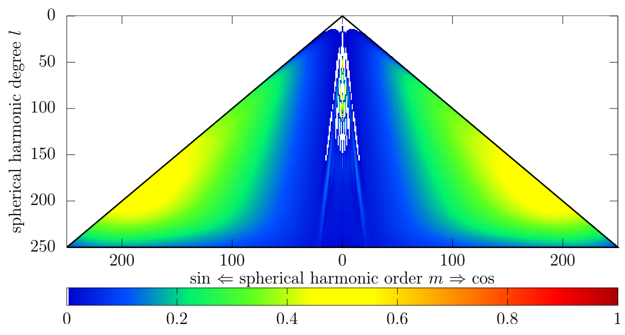
In addition to the spectral domain, the models can be compared to existing models in the spatial domain. Gravity field functionals, e.g., geoid heights (cf. (5.6)) or gravity anomalies (cf. (5.7)) can



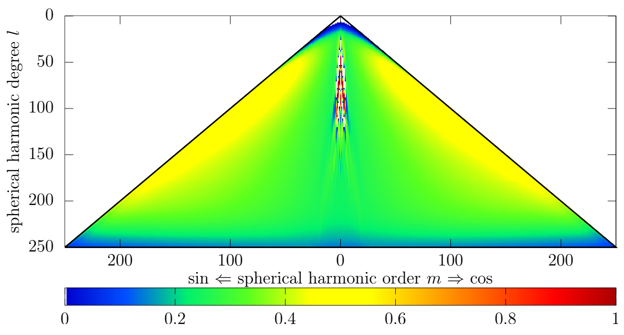
(a) Contribution of the V_{XX} NEQs.



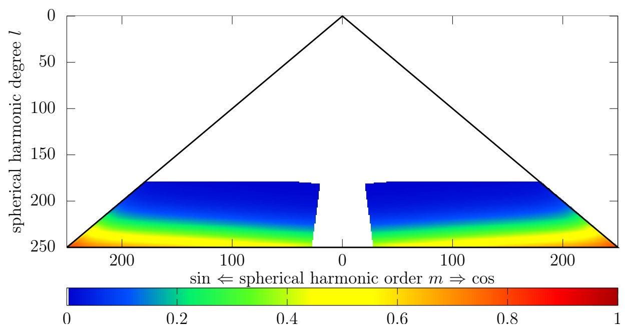
(b) Contribution of the V_{XZ} NEQs.



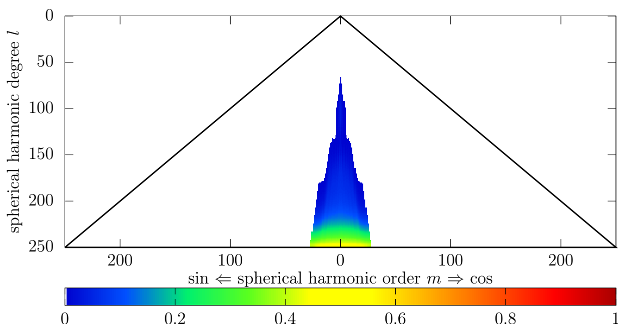
(c) Contribution of the V_{YY} NEQs.



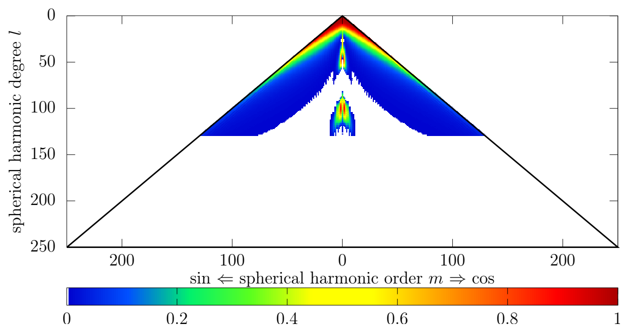
(d) Contribution of the V_{ZZ} NEQs.



(e) Contribution of the high degree regularization.

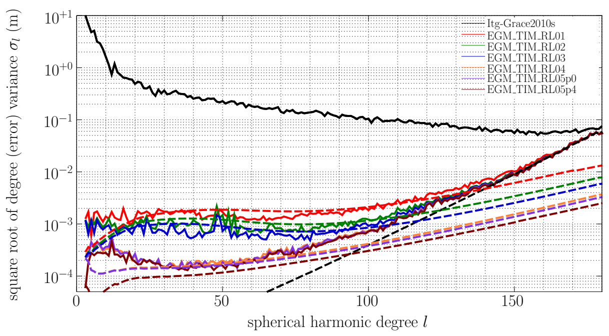


(f) Contribution of regularization of near zonal coefficients.

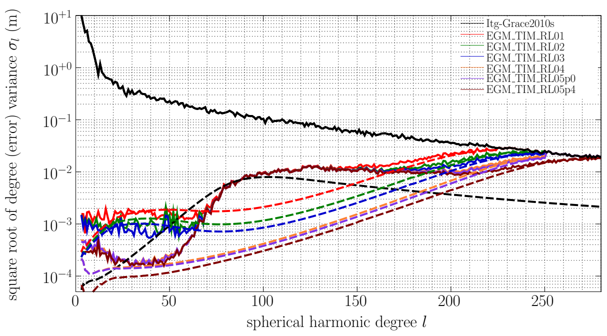


(g) Contribution of the SST NEQ.

Figure 6.20: Contributions of the individual groups within the EGM_TIM_RL04 release computed via (5.9).



(a) Solutions with respect to ITG-Grace2010s.



(b) Solutions with respect to EGM2008.

Figure 6.21: Degree (error) variances of the six computed time-wise solutions for the lower degrees with respect to the ITG-Grace2010s model, which is assumed to be superior up to degree 60 and with respect to the combined model EGM2008 which is assumed to be superior in the higher degrees above d/o 220. Between degree 70 and 210 GOCE models are the most accurate gravity field information and can thus not be validated.

be computed from the GOCE models and from e.g. the EGM2008 model and the differences at a specific truncation degree can be analyzed. The Figs. 6.22(a)–6.22(e) show geoid heights with respect to the EGM2008 model truncated at degree 200 for all releases computed. This figure can be used to demonstrate the progress along the mission. Whereas for release 01 only large differences over the continents (resulting from no or erroneous terrestrial data in EGM2008) are clearly visible, the marine geoid is dominated by the error of the GOCE model (noise). EGM2008 is superior there. Going on to release 02, the significant errors over land remain, but the noise over the ocean significantly reduces. This characteristics remain the same up to release 04 and 05p4. But, for the newer releases, significant differences occur over the ocean also, which is assumed to be additional signal of the marine geoid which was observed by GOCE for the first time and is not included in EGM2008 (cf. Figs. 6.22(g)–6.22(k)). This is especially true for i) coastal areas, where altimetry contained in EGM2008 is known to be inaccurate, ii) high latitudes, where the available altimetry data is limited and iii) in rough areas, e.g. the circumpolar current. This additional signal is highlighted in local zooms in a coastal area in Figs. 6.22(g)–6.22(k), which clearly show an example for new marine geoid signal observed by GOCE.

Finally, Tab. 6.6 gives an overview of the mean global accuracies reached with the five GOCE models at different truncation degrees of the spherical harmonic expansion and thus at different spatial resolutions. The given numbers $s(l_{\max}, \mathbf{x}_1, \mathbf{x}_2)$ result from root mean square error (RMS) of the functional $F(\mathbf{x})$ (geoid height and gravity anomalies) with respect to EGM2008 over an area with N points p_n (functional evaluated on a $0.25° \times 0.25°$ grid) truncated at different degrees $l_{\max}$. The empirical RMS provided is in dependence of the area, the test model and the truncation degree

$$s_{l_{\max}} = s(l_{\max}, \mathbf{x}_1, \mathbf{x}_2) = \sqrt{\frac{1}{N}\sum_{i}^{N}\left(F(\mathbf{x}_1, p_i) - F(\mathbf{x}_2, p_i)\right)^2} \tag{6.32}$$

$$= \sqrt{\frac{1}{N}\sum_{i}^{N}\left(F(\mathbf{x}_{\text{EGM2008}}, p_i) - F(\mathbf{x}_{\text{EGM\_TIM\_RL*}}, p_i)\right)^2}. \tag{6.33}$$

The accuracy denoted as σ is the mean error derived from a variance propagation of the full covariance matrix of the spherical harmonic coefficients (from the GOCE solution only) to the gravity field functional over the region. The EGM2008 model error is neglected, as no precise error information is available. The value provided is the mean value over the evaluated grid. The numbers are given

Table 6.6: Error estimates from the GOCE time-wise models for the different releases. s is the RMS with respect to EGM2008 and σ the mean value derived from variance propagation (GOCE error only). The south pacific area is a smooth area over the open ocean, where EGM2008 is assumed to be of high quality.

(a) Geoid heights

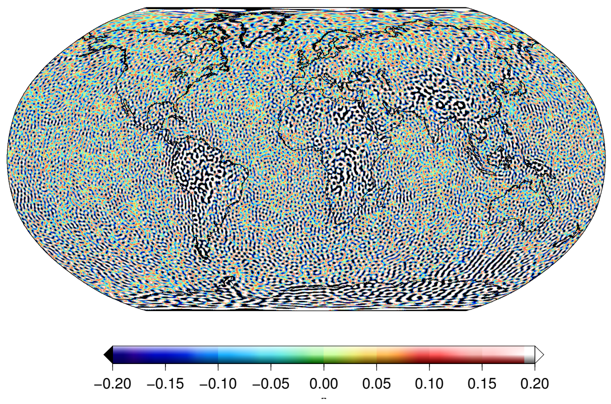
		geoid heights (cm) to spherical harmonic d/o											
area	release	180		200		220		240		250		280	
		s	σ	s	σ	s	σ	s	σ	s	σ	s	σ
south	EGM_TIM_RL01	8.9	8.4	14.2	14.0	18.2	21.3						
pacific	EGM_TIM_RL02	5.6	4.7	9.5	8.3	13.8	14.2	17.5	21.1	18.4	24.3		
	EGM_TIM_RL03	4.4	3.6	6.8	6.5	10.7	11.4	15.0	18.0	16.2	21.2		
	EGM_TIM_RL04	3.2	2.2	4.8	4.1	7.6	7.7	11.9	13.4	13.5	16.7		
	EGM_TIM_RL05p0	3.0	1.9	4.2	3.5	6.6	6.5	10.2	11.4	11.7	14.4		
	EGM_TIM_RL05p4	2.9	1.5	3.7	2.8	5.4	5.0	8.2	8.9	9.7	11.4	13.5	19.7
±80°	EGM_TIM_RL01	14.0	7.2	18.5	11.9	23.6	18.6						
	EGM_TIM_RL02	12.5	4.1	15.6	7.0	19.3	12.0	23.6	18.4	25.6	21.7		
	EGM_TIM_RL03	12.2	3.2	14.8	5.5	17.9	9.6	21.6	15.4	23.5	18.6		
	EGM_TIM_RL04	12.0	2.0	14.3	3.6	16.4	6.5	19.3	11.2	21.0	14.2		
	EGM_TIM_RL05p0	11.9	1.8	14.2	3.0	16.1	5.3	18.4	9.4	19.9	12.0		
	EGM_TIM_RL05p4	12.0	1.4	14.1	2.4	15.8	4.2	17.5	7.3	18.7	9.4	22.9	17.2

(b) Anomalies

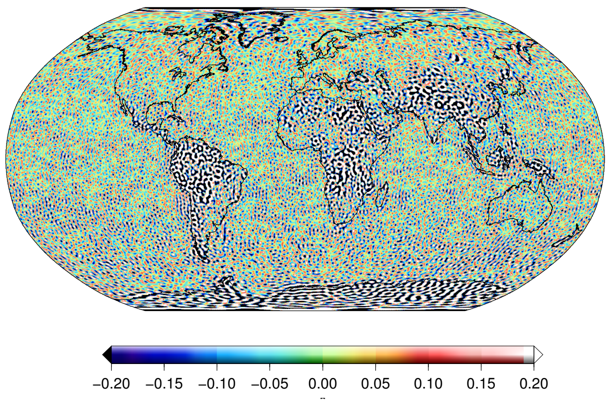
		anomalies (mGal) to spherical harmonic d/o											
area	release	180		200		220		240		250		280	
		s	σ	s	σ	s	σ	s	σ	s	σ	s	σ
south	EGM_TIM_RL01	2.18	2.05	3.91	3.89	5.39	6.50						
pacific	EGM_TIM_RL02	1.35	1.16	2.64	2.33	4.19	4.39	5.63	7.07	6.06	8.38		
	EGM_TIM_RL03	1.04	0.89	1.86	1.81	3.28	3.55	4.95	6.07	5.48	7.40		
	EGM_TIM_RL04	0.74	0.56	1.30	1.18	2.31	2.43	3.99	4.59	4.67	5.91		
	EGM_TIM_RL05p0	0.68	0.48	1.11	0.98	2.00	2.02	3.41	3.90	4.04	5.12		
	EGM_TIM_RL05p4	0.66	0.39	0.97	0.78	1.61	1.58	2.73	3.05	3.37	4.08	5.07	7.71
±80°	EGM_TIM_RL01	3.18	1.76	4.76	3.32	6.72	5.69						
	EGM_TIM_RL02	2.79	1.02	3.90	1.97	5.37	3.73	7.23	6.21	8.10	7.55		
	EGM_TIM_RL03	2.70	0.79	3.66	1.54	4.88	2.98	6.53	5.24	7.39	6.54		
	EGM_TIM_RL04	2.66	0.52	3.51	1.01	4.36	2.03	5.65	3.86	6.44	5.05		
	EGM_TIM_RL05p0	2.64	0.46	3.47	0.85	4.24	1.68	5.31	3.22	6.03	4.28		
	EGM_TIM_RL05p4	2.68	0.36	3.44	0.69	4.12	1.31	4.94	2.50	5.52	3.37	7.70	6.79

for two areas, i.e. global area excluding the polar gap and a small area in the open south pacific, where the geoid as well as the dynamic topography is smooth such that EGM2008 should be of high quality there. The global RMS values are large and always dominated by the large errors over land. To give a statement on the mean global accuracy of the most recent GOCE model one has to rely on the propagated accuracy which is 3.6 cm and 1.0 mGal at degree and order 200 (i.e. 100 km resolution) for the EGM_TIM_RL04 and 2.4 cm and 0.7 mGal for the preliminary EGM_TIM_RL05p4. The propagated accuracies are approved by the local comparison in the south pacific. For degrees 220–250 where EGM2008 is assumed to be dominant, $s \leq \sigma$, which approves that the propagated numbers are realistic.

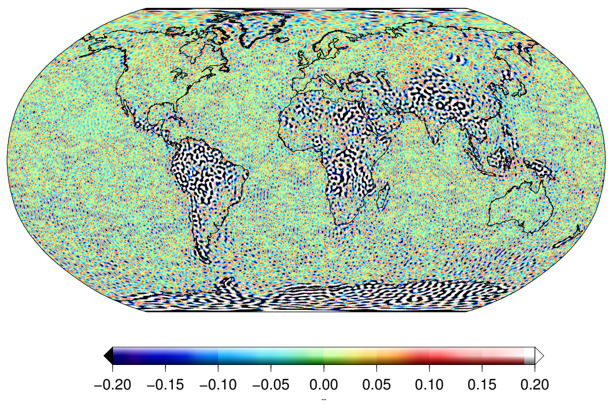
More sophisticated and advanced validation procedures for GOCE models are developed and performed in for instance Gruber et al. (2011), Hirt et al. (2011), Rexer et al. (2014), Becker et al. (2014b) and Voigt and Denker (2014).



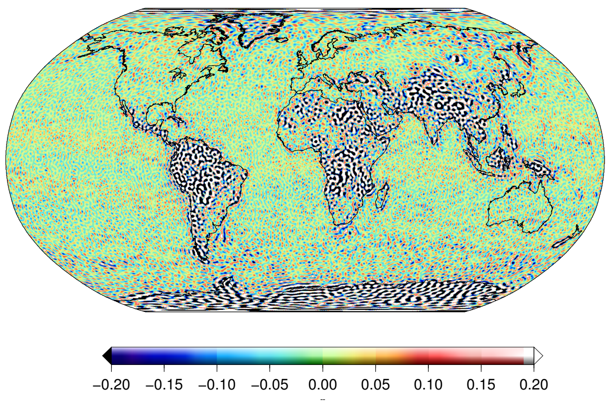
(a) EGM_TIM_RL01.



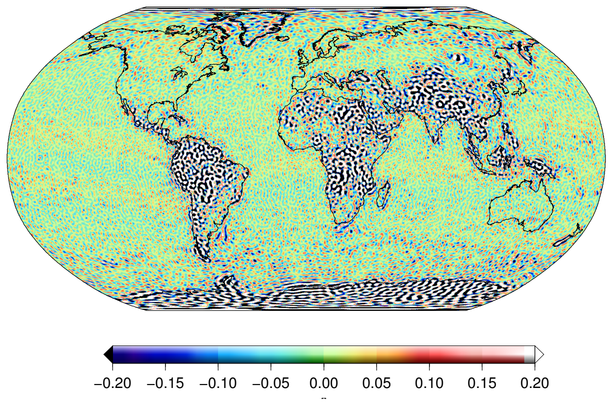
(b) EGM_TIM_RL02.



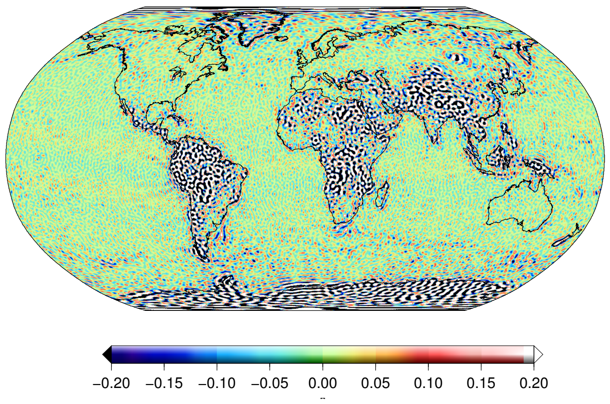
(c) EGM_TIM_RL03.



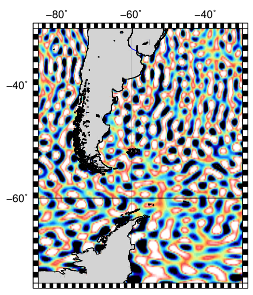
(d) EGM_TIM_RL04.



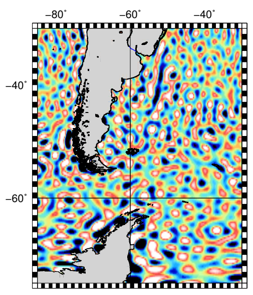
(e) EGM_TIM_RL05p0.



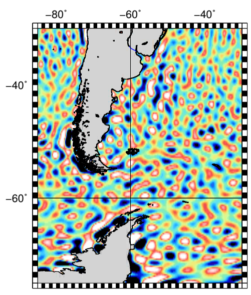
(f) EGM_TIM_RL05p4.



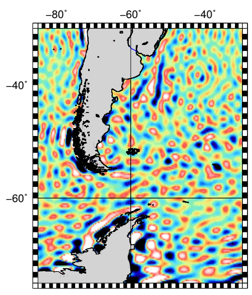
(g) EGM_TIM_RL01.



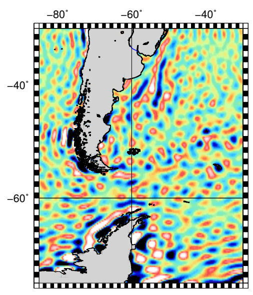
(h) EGM_TIM_RL02.



(i) EGM_TIM_RL03.



(j) EGM_TIM_RL04.



(k) EGM_TIM_RL05p4.

Figure 6.22: Differences of the time-wise releases to EGM2008 in terms of Geoid heights (m) with a global and a local (i.e. costal) focus truncated at d/o 200 (100 km).

7. Application: High Degree Gravity Field Determination Using a Direct Solver

The concepts of high performance and scientific computing introduced in Chap. 2 and 3 are used to implement a high degree gravity field solver based on the assembly and solution of full normal equations. Approximations and simplifications as summarized in Sect. 5.1 should be avoided to derive a consistent estimate of the model and a realistic full covariance matrix which could be assembled into further process models. With the full NEQ available, the covariance matrix of the estimated parameters can be directly accessed.

A motivation for the implementation of the direct solver in the HPC environment are – amongst others – the studies performed by Becker et al. (2012), Becker (2012) and Becker et al. (2013). A concept of a joint rigorous estimation of the ocean's mean dynamic topography (MDT) and the marine gravity field was developed from the combination of along-track altimetry observations and gravity field NEQs. In contrast to other studies, along-track altimetry was used to estimate the MDT together with an update for the gravity field in a rigorous one-step approach, propagating the errors of the observations throughout the whole processing. Within the studies performed so far, the joint estimation was limited to a local area over the oceans (although the gravity field was parameterized as spherical harmonics). Due to computational limits, the focus for the MDT was local and the resolution of the gravity field was limited to d/o 300, although the altimetry contains higher resolution gravity field signal. As in these studies, only the MDT and its full covariance matrix was of interest for the assembly into ocean models, the mentioned constraints were acceptable. Anyway, in contrast to other studies, which typically subtract a gravity field model from an independently processed and gridded mean sea surface without error information and apply some (Gaussian) filtering (e.g. Knudsen et al., 2011, Bingham et al., 2011), a rigorous approach was developed which provides the covariance matrix of the MDT in addition. This requires to start at the level of along-track altimetry observations, to perform the rigorous error propagation.

Extending the method developed in Becker (2012) to a global scope and defining the gravity field as a target quantity, too, the computational challenges further increase. Thus, in a first step, a HPC solver is required, which can assemble NEQs from altimetry OEQs for the high degree signal contained in the observations. As a second step (not in the focus of this thesis) the OEQs have to be extended for the additional parameters describing the MDT (e.g. finite elements). This application, amongst others, leads to the motivation to implement a rigorous solver for global gravity field determination in a HPC environment with a flexible design, which has the possibility to set up the full covariance matrix for a large amount of parameters (hundreds of thousands). The implementation developed in this thesis serves as the basis for a further extension for special applications, as e.g. the application outlined above. The physically simplified simulation scenario is used as a demonstrator, characterizing the solvable dimensions of the adjustment problem and the performance in the HPC environment. It provides a general starting point into the mapping of this kind of applications to HPC environments. It is shown that the approximations and simplifications, like e.g. the gridding of altimetry observations, can be avoided to finally derive higher quality models and covariance matrices in a reasonable time.

Within this chapter, the implementation of the solver in a HPC environment is presented. In contrast to the GOCE application in Chap. 6 the computational challenge is not the number of highly correlated observations but the huge dimensional parameter space (number of unknown parameters). The concept of block-cyclic distributed matrices is used for the assembly of NEQs and their solution. The implemented solver has the following key features (mostly defined before as target properties for possible further applications of the developed basis implementation):

- It obtains the least squares solution based on full normal equations.
- It is tailored for huge dimensional parameter spaces (up to at least half a million of unknown parameters, corresponding to spherical harmonic d/o 720).
- It has the capability to process a large number of observations (i.e. several millions).
- The processing of an arbitrary number of observation groups is foreseen.
- Variance component estimation is implemented for relative weighting of all observation groups.
- The combination of preprocessed normal equations as sufficient statistics of original observations and original (e.g. point wise) observations and their variance/covariance information can be used as observation groups.
- The implementation is flexible (e.g. include group specific parameters, extension of new observation equations, ...).
- A general concept which can be easily adapted for adjustment applications not related to gravity field determination is used.
- The developed software is executable in environments with only a few cores and in HPC environments with tens of thousands of compute cores.

This chapter is an extension of the study already published in Brockmann et al. (2014b), so some general parts of the chapter are taken from that study. Although the MDT is not covered at this stage, the implementation is in progress and the task should be solved in further research projects within the department.

7.1 Problem Description

The basic mathematical/statistical problem of the combination of independent observation groups with the addition theorem of NEQs is already described in (4.6b) or more precisely in (4.27). The combined normal equation should be assembled from band-limited normal equations $n \in \{0, \dots, N-1\}$ (differing in resolution, size and in the parameter order) and from observation groups $o \in \{0, \dots, O-1\}$, which are assumed to contain the (very) high resolution gravity field signal and thus produce the high dimensional parameter space.

Whereas the NEQs n have only to be recalled from disk, reordered and properly added to a subspace of $\mathbf{N}$, the challenge for the processing of the OEQs o is an efficient and fast computation of the partial normal equations $\mathbf{N}_o = \mathbf{A}_o^T \mathbf{Q}_{oo}^{-1} \mathbf{A}_o$ for the whole parameter space as defined by the symbolic target numbering scheme $\mathtt{p}$. This parameter space can either include spherical harmonic coefficients of a certain resolution or, although not presented in detail here, non spherical harmonic parameters observed for instance by a subset of observation groups only. For example, the developed software package is used to estimate the ocean's dynamic topography from altimetry observations and the marine gravity field in an one step approach (Becker et al., 2013, 2014b). As a challenging part, hundreds of thousands of parameters are forseen, e.g. for the rigorous estimation of gravity field models up to d/o 720 (dimension of $\mathbf{N}$ is $519\,837 \times 519\,837$, its size is approximately 2 TB). The implemented concept is not limited to that resolution, but at some point with more and more unknown parameters, the setup of the full normal equations become unreasonable (computational and storage requirements). For even higher dimensional parameter spaces, alternative solution concepts are better suited (see Chap. 8).

7.2 Assembly and Solution of the Combined NEQs

The groups provided as NEQs and the groups available as observations are treated differently. Consequently, this chapter divides the processing of this groups into two parts. For the whole

Table 7.1: Information provided together with the NEQs.

```
# parameters used to assemble the NEQ
# NEQ itself
name = GOCE_MOP4_ZZ_VCE0004                   # name of the observation group
pathTon = GOCE_MOP4_ZZ_VCE0004_n.gdk          # path to RHS n (binary format)
pathToN = GOCE_MOP4_ZZ_VCE0004_N.gdk          # path to NEQ matrix N (binary format)
lTPl = 3.721873844140784454e+10               # product lambda = l'Pl
numObsUsed = 68075886                         # # observations NEQs were computed from
numberingSchemeNeq = GOCE_MOP4_ZZ_VCE0004.ns  # symbolic numbering scheme associated with N,n
# gephysical constants used
tide_system = tide_free                       # tide system used during assembly
GM_used = 3.986004415000000000e+14            # GM used during assembly
a_used = 6.378136459999999963e+06             # a (mean Earth radius) used during assembly
```

section, it is assumed that a target symbolic numbering scheme **p** is defined, which contains all parameters to be estimated. It is the numbering scheme associated with the finally combined NEQs $\mathbf{N}$ and $\mathbf{n}$ and defines the parameter order in the final output matrices (NEQs and/or covariance matrix).

7.2.1 Update of the Combined NEQ with Groups Provided as NEQs

The combination of normal equations seems to be a quite simple task, as just the (weighted) sum of the NEQs has to be computed. Nevertheless, from the implementational point of view, this step is not as simple as it seems, because (i) the NEQs n differ in size, as they are only assembled for a subset of the parameters $\mathbf{N}$ should be assembled for, and (ii) the parameter order is different and generally not compatible with the target numbering scheme **p** defined for $\mathbf{N}$. For that reason, the concepts of reordering and symbolic numbering schemes were introduced in Sect. 4.3 and 5.3. Using the concepts introduced in Chap. 3, 4 and 5 and the implementations derived there, the task becomes simpler.

For every NEQ group, the details given in Tab. 7.1 are provided. In addition to this information, a weight $w_n^{(0)}$ is defined to be used in the combination. It is iteratively updated using VCE. A group available as NEQs is handled as an object of the class `NeqGroup` as described by Listing 7.1.

Alg. 7.1 gives an overview of the processing steps of the individual NEQs. With the preparations performed in Chap. 3, 4 and 5 most of the steps are straightforward. The only step which has to be discussed in detail is the setup of the additional right hand sides for VCE (cf. Sect. 4.2.1). Nevertheless, the following paragraphs shortly provide an overview how the steps for every NEQ can be implemented in a HPC environment. The following steps are sequentially repeated for every NEQ $n \in \{0, \ldots, N-1\}$ (n-loop in Alg. 7.1).

Original NEQs from Disk (l. 6–11, Alg. 7.1) The first step is to recall the original NEQs from disk and map them as distributed matrices to the memory of the involved cores. This can be directly performed using the parallel I/O from the MPI standard as implemented in Sect. 3.3.4.1 for the `DistributedMatrix` class. After the parallel reading operation the NEQs are available as distributed matrices. The only constraint on the dimension of the NEQ groups is, that the NEQ fits into the joint memory of all cores involved in the compute core grid. In addition, every core involved reads the symbolic numbering scheme the NEQs are stored in. Again, the numbering scheme for the input matrices is arbitrary. The weights used in current iteration are directly applied scaling $\mathbf{N}_n$ and $\mathbf{n}_n$ (cf. l. 11).

Algorithm 7.1: Update of $\mathbf{N}$ and $\mathbf{n}$ by groups n provided as band-limited NEQs.

Data:

`vector<NeqGroup>` $\mathbf{g}$	information on NEQs for the groups $n \in \{0, \dots, N-1\}$
`NumberingScheme` $\mathbf{p}$	symbolic target numbering scheme covering the parameters
`double` GM, a, tide system	constants to be used in target NEQ
`size_t` K	number of MC samples for additional RHS

```
// initialization of combined NEQs, account for MC RHS
DistributedMatrix $\mathbf{N}(U,U)$,    $\mathbf{n}(U, 1+(N+O)K)$
size_t  $M = 0$,    double  $\lambda = 0$
// Loop over all groups provided as NEQs
for $n = 0$ to $N-1$ do
    // Temporal matrices for NEQs from file
    DistributedMatrix $\mathbf{N}_n$, $\mathbf{n}_n$
    // Read NEQs and meta data
    $\mathbf{N}_n$.binaryRead( $\mathbf{g}$.at($n$).pathToN() ),    $\mathbf{n}_n$.binaryRead( $\mathbf{g}$.at($n$).pathTon() )
    NumberingScheme $\mathbf{p}_n$($\mathbf{g}$.at($n$).numberingSchemeNeq() )
    $\mathbf{N}_n * =$ $\mathbf{g}$.at($n$).w(),    $\mathbf{n}_n * =$ $\mathbf{g}$.at($n$).w()
    // If needed convert geophysical parameters (tide system, $GM$, $a$)
    _convertNeqs($\mathbf{N}_n$, $\mathbf{n}_n$)
    // Extend NEQs to dimension of $\mathbf{N}$
    $\mathbf{N}_n$.extendWithZeros($\mathbf{N}$.R(),$\mathbf{N}$.C()),    $\mathbf{n}_n$.extendWithZeros($\mathbf{N}$.R())
    // Reordering operation cf. Sect. 4.3
    vector<size_t> $\boldsymbol{\psi}_{\mathbf{p}_n \mapsto \mathbf{p}}$ = $\mathbf{p}$.permutationVec($\mathbf{p}_n$)
    $\mathbf{N}_n$.reorder($\boldsymbol{\psi}_{\mathbf{p}_n \mapsto \mathbf{p}}$),    $\mathbf{n}_n$.reorderRows($\boldsymbol{\psi}_{\mathbf{p}_n \mapsto \mathbf{p}}$)
    // Update combined NEQs with current group
    $\mathbf{N} + = \mathbf{N}_n$,    $\mathbf{n}(:,0) + = \mathbf{n}_n$
    if (r == c == 0) then
        $M + =$ $\mathbf{g}$.at($n$).anzObs(),    $\lambda + =$ $\mathbf{g}$.at($n$).w() $\cdot$ $\mathbf{g}$.at($n$).ltl()
    end
    // Generate and transform RHS for MC trace estimation
    DistributedMatrix $\mathbf{P}_n$($\mathbf{N}_n$.R(),$K$)
    $R^l, C^l = \mathbf{P}_n$.localMat().size()
    // Fill local parts of matrix with random number $\pm 1$
    for $c = 0$ to $C^l - 1$ do
        for $r = 0$ to $R^l - 1$ do
            $\mathbf{P}_n$.localMat($r, c$) $= \pm 1$
        end
    end
    // Recover original NEQ (e.g. from disk)
    $\mathbf{N}_n$.binaryRead( $\mathbf{g}$.at($n$).pathToN() ),    $\mathbf{N}_n * =$ $\mathbf{g}$.at($n$).w()
    _convertNeqs($\mathbf{N}_n$, $\mathbf{n}_n$)
    $\mathbf{N}_n$.chol()
    // transform samples cf. (4.13)
    $\mathbf{P}_n = \mathbf{N}_n^T \mathbf{P}_n$
    $\mathbf{P}_n$.extendWithZeros($\mathbf{N}$.R(),$K$)
    $\mathbf{P}_n$.reorderRows($\boldsymbol{\psi}_{\mathbf{p}_n \mapsto \mathbf{p}}$)
    // update columns of RHS belonging to MC samples of current group
    $\mathbf{n}(:, (n-1)K + 1 : nK) + = \mathbf{P}_n$
end
return $\mathbf{N}, \mathbf{n}, \lambda, M$// combined NEQs
```

Listing 7.1: Header file defining the main features of the class `NeqGroup`.

```
class NeqGroup
{
        public:
                //=====================================================
                //Constructors Destructors
                NeqGroup();
                ...
                ~NeqGroup();
                //=====================================================
                // methods
                // get attributes
                // NEQ information
                string pathToN() const;
                string pathTon() const;
                double ltl( ) const;
                double numObsUsed( ) const;
                NumberingScheme & ns() {return _ns;};
                // constants and reference systems
                earthTideSystems::TIDE_SYSTEM tide_system( ) const;
                double GM() const;
                double a() const;
                double w( ) const;
                // set parameters and other functions
                ...
        private:
                string name;
                string pathTon;
                string pathToN;
                double ltl;
                size_t numObsUsed;
                NumberingScheme _ns;
                // constants and reference systems
                earthTideSystems::TIDE_SYSTEM tide_system;
                double _GM;
                double _a;
};
```

Conversion of Physical Constants/Reference Systems (l. 13, Alg. 7.1) For a consistent combination, physical constants (GM, a) have to be unified, if alternative values were used in the original assembly of $\mathbf{N}_n$ and $\mathbf{n}_n$. The transformations are not given in detail here, as they are provided in many text books and standard documents (see e.g. EGG-C, 2010a) and the physical consistence is not in the scope of this study. The same holds for references as for instance the permanent tide system of the spherical harmonic coefficients. In addition, the RHS might be transformed, if for instance a reference gravity field model (approximated values in linearization) was reduced from the raw observations in the original NEQ assembly.

Update of Normals (l. 14–23, Alg. 7.1) The NEQs $\mathbf{N}_n$ and $\mathbf{n}_n$ are extended with zeros to adjust the dimension to the dimension of $\mathbf{N}$ and $\mathbf{n}$. The extended matrix is brought to the numbering of $\mathbf{N}$ and $\mathbf{n}$ applying the reordering as discussed in detail in Sect. 4.3. After the determination of the permutation vector $\boldsymbol{\psi}_{\mathbf{p}_n \mapsto \mathbf{p}}$ (cf. Alg. 4.2 and 4.4), rows and columns of $\mathbf{N}_n$ and rows of $\mathbf{n}_n$ are reordered according to Sect. 4.3.3. The update of $\mathbf{N}$ and $\mathbf{n}$ is then a simple addition. The number of observations M (cf. (4.6e)) and λ (cf. (4.6d)) of the combined NEQs are updated accordingly (cf. l. 21–23). For this (and some of the later steps, e.g. for the observation equations), it is required that the main memory of all cores is sufficient to store the whole NEQ matrix twice in memory.[9]

Additional Right Hand Sides for VCE (l. 27–42, Alg. 7.1) As the NEQs for the individual groups (especially for the OEQ groups cf. Sect. 7.2.2) are not stored separately by default, the Monte Carlo based trace estimator introduced is used to estimate the partial redundancy (cf. Sect. 4.2.1) within the direct solver. For that reason, additional sample based right hand sides have to be introduced. As the quality of the sample based MC trace estimation depends on the number of

[9]The alternative, reordering the first U_n rows and columns of $\mathbf{N}$ to $\mathbf{p}_n$ is not discussed in detail here. With the provided concept it is easily possible. This alternative strategy is more efficient with respect to main memory but has the disadvantage that two reordering steps are required (1. reorder the first U_n rows and columns of $\mathbf{N}$ to $\mathbf{p}_n$, 2. update, 3. reorder back to original numbering scheme).

generated samples, it is assumed that for every involved group n or o, K samples are generated to derive a mean estimate. Thus, $(N+O)K$ additional right hand sides are generated. Assuming the first column of $\mathbf{n}$ to be associated with the observations itself, the next NK are associated with the MC samples of the NEQ groups and later on the next OK to the OEQ groups. For a single group n, the columns $c_b = (n-1)K+1$ to $c_e = nK$ are the columns related to the MC based samples for trace estimation. Only this columns are updated via the transformed samples according to (4.13).

First of all, a distributed matrix of dimension $U_n \times K$ is generated and filled with random numbers following the uniform distribution with values $+1$ and -1. Random numbers are drawn in parallel but serially on every core, and the local matrices $\mathbf{P}^l_{\mathsf{r,c}}$ are filled on every core. Using parallel programs and random number generators needs a carful check if these generators are suited for parallel implementations to derive independent samples on the local cores (e.g. Katzgraber, 2010, Mertens, 2009). Within this study, a BOOST (Boost, 2013a) random number generator is used (Chap. 24 Boost, 2013b). The seed of the generator was composed by the local CPU time of the individual cores and the MPI rank of the calling process. Numerical tests have shown that the random numbers generated on different cores are independent.

The drawn samples are transformed according to (4.13). Similar to the processing of the original right hand side $\mathbf{n}_n$, the final step is to adjust the size and the parameter order of the transformed samples (cf. l. 39–40, Alg. 7.1). Finally, the additional right hand sides are copied to the corresponding columns of $\mathbf{n}$, i.e. $(n-1)K+1$ to nK.

7.2.2 Update of the Combined NEQ with Groups Provided as OEQs

This section covers the update of the combined normal equations $\mathbf{N}$ with the groups provided as original observations (OEQs). At this stage it is assumed, that the observations are uncorrelated, but have an individual observation accuracy. Consequently, (4.6b) is computationally simplified, as $\mathbf{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}$ are diagonal matrices only. Nevertheless, the use of full covariance/cofactor matrices e.g. via using empirical estimated covariance functions or – like in the GOCE application – digital decorrelation filters, is forseen in the implementation. For this general case, i.e. foreseeing correlated observations, block-cyclic distributed matrices are used for the observation equations, i.e. for the design matrices $\mathbf{A}_o$, the cofactor matrix $\mathbf{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}$ and the observation vector $\boldsymbol{\ell}$. For the case of uncorrelated observations, the setup of the OEQ, especially the design matrix is a task of low complexity (still depending on the functional model) and thus fast compared to the update of the NEQs with $\mathbf{N}+= w_o\mathbf{A}_o^T\mathbf{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}^{-1}\mathbf{A}_o$. Thus, in addition an alternative concept is implemented, computing local parts of the NEQ avoiding communication and using serial BLAS routines for the computations. The consequences are that the computations are faster, as no communication during the computations are needed but the drawback is that the OEQ are set up locally more then once and redundant on different processes.

For both concepts implemented, the observations and the meta data are stored in objects of the class `OeqGroup` as summarized in Listing 7.2. The class provides the member functions to set up the entries of the design matrix as well as the storage of the data. The details on the setup of the observation equations are not given here, as it is a well known and documented step (as e.g., the recursive computation of Legendre functions) and depends a lot on the observation type. Although all applications are related to global gravity field determination, the developed concept as well as the implemented software package can be readily used for the processing of other geodetic data sets.

7.2.2.1 Distributed Computation of NEQs using Block-cyclic Distributed Matrices

The easiest way to compute the update of the combined NEQs is to set up the observation equations as block-cyclic distributed matrices. It is the most flexible way with respect to the number of

Listing 7.2: Header file defining the main features of the class `OeqGroup`.

```
class OeqGroup
{
        public:
                //====================================================
                //Constructors/Destructors
                OeqGroup();
                ...
                ~OeqGroup();
                //====================================================
                // methods
                // get attributes
                // OEQ information
                double weight( );
                string file( );
                size_t size( );
                // load observations and meta data from disk
                void loadObservations( );
                // design matrix for single observation i
                void buildDesign( double * Aptr, int offset, int cols, NumberingScheme & ns );
                string name();
                ...
        private:
                string _filename;
                double _weight;
                georeferencedObservations _pointObs;
                earthmodelFunctionals::FUNCTIONAL_TYPE _funcToCompute;

};
```

parameters and observations. Especially if a complex decorrelation step is required, the use of block-cyclic distributed matrices for the OEQs provides the most flexible options. Alg. 7.2 summarizes the update step. The details are discussed in the following. As, comparably to the data of the GOCE mission (cf. Sect. 6.3), it can not be expected that the OEQs for a single group o can be set up at once, instead due to the possible huge number of observations, the observations are again processed sequentially in blocks of size b_{obs}. Thus, the block-cyclic distributed design matrix is set up only for b_{obs} observations at once (b-loop in Alg. 7.2).

The local matrices $\mathbf{A}^{l}_{\mathsf{r},\mathsf{c}}$ of the block-cyclic distributed design matrices are filled for the local observations and for the local parameter subset locally on the compute cores. All entries are locally computed, although redundant computations during the recursive computation of Legendre polynomials might occur. Afterwards, $\mathbf{N}$, $\mathbf{n}$ and λ are directly updated using SCALAPACK and PBLAS routines (mainly `pdgemm` and `pdsyrk`) which are used in member functions of the `DistributedMatrix` class. As CMO is used to store all (local) matrices, instead of $\mathbf{A}_o$, $\mathbf{A}_o^T$ is assembled, therefore entries for a single observation can be directly written in coherent parts of the memory (column-wise).

Processing of Groups o As described in Alg. 7.2, the O groups are processed independently in a sequential loop. There is no parallel processing of different groups o. For that reason, details on the implementation are only given for a single group o. Note that in the presented form of the implementation it is assumed, that all processes involved read all observations from disk and have all observations available in the main memory. The alternative, that a single process reads all observations and distributes them is not considered here.

Setup of Design Matrices, (cf. Alg. 7.2, l. 20–26) The basic idea is to setup the design matrix as a block-cyclic distributed matrix as introduced in Chap. 3. As the number of observations in a group o is not limited, it is assumed that the observations are processed in blocks of size b_{obs}. The blocks of size b_{obs} are processed sequential (b-loop in Alg. 7.2). Only the observations within a single block are processed in parallel. Note that the final step is missing in Alg. 7.2, all shown steps have to be repeated for the smaller rest block of size $M_o\%b_{\text{obs}}$. A block-cyclic distributed design matrix is set up at once for b_{obs} observations and for all parameters U. To store entries corresponding to a single observations in continuous parts of the local memory, instead of $\mathbf{A}_o$, $\mathbf{A}_o^T$ of dimension $U \times b_{\text{obs}}$ is set up in the implementation which uses CMO (cf. Sect. 2.2.1).

Algorithm 7.2: Update of $\mathbf{N}$ and $\mathbf{n}$ by groups o provided as OEQs. The updates of $\mathbf{N}$ and $\mathbf{n}$ are computed from block-cyclic distributed observation equations.

Data:

`vector<OeqGroup>` **g**	information on observations for the groups $o \in \{0, \dots, O-1\}$
`NumberingScheme` **p**	symbolic target numbering scheme covering the parameters
`double` GM, a, tide_system	constants to be used in target NEQ
`size_t` K	number of MC samples for additional RHS
`size_t` b_{obs}	number of observations to be processed at once

```
// determine my process coordinates and compute core grid dimension
size_t r, c, R, C
blacs_gridinfo( $\mathbf{A}$.context(), r, c, R, C )
// initialization of combined NEQs, account for MC RHS
size_t U = $\mathbf{p}$.size()   size_t $M = 0$   double $\lambda = 0$
DistributedMatrix $\mathbf{N}(U, U)$,   $\mathbf{n}(U, 1 + (N+O)K)$
DistributedMatrix $\mathbf{A}^T(U, b_{\text{obs}})$,   $\mathbf{l}(b_{\text{obs}}, 1 + (N+O)K)$
// Determine dimension of local matrices
size_t $R^l_{\mathbf{A}^T} = \mathbf{A}^T$.Rl(),  size_t $C^l_{\mathbf{A}^T} = \mathbf{A}^T$.Cl(),  size_t $R^l_{\mathbf{l}} = \mathbf{l}$.Rl(),  size_t $C^l_{\mathbf{l}} = \mathbf{l}$.Cl()
// parameters associated with local rows ($\mathbf{p}_r$) of $\mathbf{A}^T$
NumberingSchme $\mathbf{p}_r(R^l_{\mathbf{A}^T})$
for $r = 0$, $r < r^l_{\mathbf{A}^T}$, $r{+}{+}$ do
    $\mathbf{p}_r(r) = \mathbf{p}(\mathbf{A}^T$.rowInGlobalMat($r$) )
end
// Loop over all groups provided as OEQs
for $o = 0$, $o < O$, $o{+}{+}$ do
    g.at($o$).loadObservations()          // Load all observations (and meta data) of group o from file
    // Loop over all observations of group o in blocks of $b_{obs}$
    for $b = 0$, $b <$g.at($o$).size(), $b{+}{=}\, b_{obs}$ do
        // setup $\mathbf{A}^T$ for local observations and local parameters
        for $c = 0$, $c < C^l_{\mathbf{A}^T}$, $c{+}{+}$ do
            // overall index of observation of group o corresponding to column c
            size_t $i = b + \mathbf{A}^T$.colInGlobalMat($c$)
            // fill columns with design entries for observation i and local parameters in $\mathbf{p}_r$
            fillDesign($\mathbf{A}$.localMat().colPtr($c$), g.at($o$).obs($i$), $\mathbf{p}_r$ )
        end
        // fill local part of distributed $\mathbf{l}$ (observations c = 0, MC samples (VCE))
        for $c = 0$, $c < C^l_{\mathbf{l}}$, $c{+}{+}$ do
            for $r = 0$, $r < R^l_{\mathbf{l}}$, $r{+}{+}$ do
                if $\mathbf{l}$.colInGlobalMat(c)$== 0$ then
                    size_t $i = b + \mathbf{l}$.rowInGlobalMat($r$)
                    $\mathbf{l}$.localMat()($r, c$) = g.at($o$).l($i$)
                else
                    if $\mathbf{l}$.colInGlobalMat(c)$\in \{1 + NK + oK, \dots, 1 + NK + (o+1)K - 1\}$ then
                        $\mathbf{l}$.localMat()($r, c$) $= \pm 1$
                    end
                end
            end
        end
        // update combined NEQs with observations b to $b + b_{obs} - 1$
        $\mathbf{N}{+}{=}$ g.at($o$).w() $\mathbf{A}^T\mathbf{A}$,   $\mathbf{n}{+}{=}$ g.at($o$).w() $\mathbf{A}^T\mathbf{l}$      // Update $\mathbf{N}$ (pdsyrk), $\mathbf{n}$ (pdgemm)
        $\lambda{+}{=}$ g.at($o$).w() $\mathbf{l}^T\mathbf{l}(0,0)$,   $M{+}{=}\, b_{\text{obs}}$  // Update $\lambda$ (pdsyrk), number of used observations
    end
    // Same operations for the rest block ($M_o \% b_{obs}$)...
end
return $\mathbf{N}, \mathbf{n}, \lambda, M$// combined NEQs
```

After the initialization of the block-cyclic distributed matrix (cf. 7.2, l. 7), the dimensions of the local matrices are known (cf. (3.5)) on every core. Using (3.6), the local column index can be associated with the global index of the observation in the array of the overall observations. The column is filled with the design matrix for that observation and for the parameters associated with the local rows of $\mathbf{A}_o^T$ (local numbering scheme p_r). Within the implementation, the parameters are locally extracted from the global numbering scheme (cf. Alg. 7.2, l. 10–14). Afterwards, the setup of $\mathbf{A}_o^T$ is performed on all cores at the same time and thus in parallel. Entries for the local observations and the local parameters are computed locally without communication. Depending on the functional of the gravity field the observation equations for the spherical harmonic coefficients are computed according to the formulas given in Sect. 5.4.2.1. During the assembly $\mathbf{A}_o^T$, redundant computations on different processes occur within the recursive computations of the Legendre polynomials (hidden in the symbol `fillDesign`). An easy to use strategy to minimize the redundant computations via the choice of a distribution dependent numbering scheme is discussed in Sect. 8.3.3.2, where the fast assembly of the distributed design matrix is more important. The strategy introduced there can be easily applied here, as only the target numbering scheme p is changed. But note that within this implementation, the setup of $\mathbf{A}_o^T$ is of minor importance with respect to the total runtime (cf. Sect. 7.4).

RHS for VCE, (cf. Alg. 7.2, l. 27–39) The setup of additional right hand sides for VCE is straightforward. According to (4.19), the random samples are transformed to additional RHS the same way as the original observations are. Thus, the vector of observations is extended by additional columns (i.e. $(N + O)K$). The K columns corresponding to group o are filled with the random samples following the uniform distribution ($+1$ or -1 with probability 0.5).

Update of NEQs, (cf. Alg. 7.2, l. 40–42) As all involved matrices are set up as block-cyclic distributed matrices, the computations of the update of the combined system of NEQs is performed directly with PBLAS (SCALAPACK) routines included in the member functions of the `DistributedMatrix` class. Mainly this are `pdsyrk` for $\mathbf{N}$ and λ, and `pdgemm` for the computation of the update of the RHS $\mathbf{n}$. Note that the computation of λ is only of importance for the original observations, i.e. column $c = 0$ of the matrix $\boldsymbol{\ell}$.

7.2.2.2 Local Computation of NEQs using Serial Design Matrices

For the case that the computation of $\mathbf{A}_o$ is simple, especially if there are no correlations between the observations and thus $\mathbf{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}$ is only diagonal, the NEQs can be updated locally without communication between the processes. Instead of communication between processes during the computation of $\mathbf{A}_o^T\mathbf{A}_o$, parts of $\mathbf{A}_o$ are set up more than once locally and redundant on different processes as serial matrices. The local parts of $\mathbf{N}$, i.e. $\mathbf{N}^l_{\mathsf{r},\mathsf{c}}$, are then computed without communication between the processes using serial BLAS routines (`dsyrk` and `dgemm`). For the efficient implementation shown in Alg. 7.3, it is assumed that i) the compute core grid is quadratic, i.e. $\mathsf{R} = \mathsf{C}$ and ii) that the block-cyclic distribution block-size is quadratic, i.e. $b_r = b_c$. The details are discussed in the following paragraphs. The performance compared to the update with $\mathbf{A}_o$ as a block-cyclic distributed matrix is assessed in Sect. 7.4.1.

Local Parameters on Process (r, c) **(cf. Alg. 7.3, l. 7–13)** The combined NEQs are initialized as a block-cyclic distributed matrix. Afterwards, due to the fixed distribution parameters, the dimension of the local matrices $\mathbf{N}^l_{\mathsf{r},\mathsf{c}}$ is fixed and known. Using the member functions of the block-cyclic distributed matrix which uses (3.6), the parameters corresponding to the local rows (p_r) and the local columns (p_c) of $\mathbf{N}^l_{\mathsf{r},\mathsf{c}}$ can be easily extracted from the global numbering scheme p of $\mathbf{N}$.

Algorithm 7.3: Updates of $\mathbf{N}$ and $\mathbf{n}$ by groups o provided as OEQs. The update of $\mathbf{N}$ and $\mathbf{n}$ are computed locally without data communication.

```
Data:
        vector<OeqGroup> g          information on observations for the groups $o \in \{0, \dots, O-1\}$
        NumberingScheme p           symbolic target numbering scheme covering the parameters
        double $GM$, $a$, tide_system   constants to be used in target NEQ
        size_t $K$                  number of MC samples for additional RHS
        size_t $b_{\mathrm{obs}}$   number of observations to be processed at once
size_t r, c, R, C,  blacs_gridinfo( $\mathbf{A}$.context(), r, c, R, C )   // determine process coordinates & compute core grid dim.
// initialization of combined NEQs, account for MC RHS
size_t U = p.size(),   size_t  $M = 0$   double  $\lambda = 0$
DistributedMatrix $\mathbf{N}(U, U)$,   $\mathbf{n}(U, 1 + (N + O)K)$,   Matrix $\mathbf{A}_{\mathsf{r}}, \mathbf{A}_{\mathsf{c}}, \boldsymbol{\ell}$
// Determine dimension of local matrices
size_t $R^l_{\mathbf{N}} = \mathbf{N}$.Rl(),   size_t $C^l_{\mathbf{N}} = \mathbf{N}$.Cl(), size_t $R^l_{\mathbf{n}} = \mathbf{n}$.Rl(),   size_t $C^l_{\mathbf{n}} = \mathbf{n}$.Cl()
NumberingSchme $\mathbf{p}_{\mathsf{r}}(R^l_{\mathbf{N}})$, $\mathbf{p}_{\mathsf{c}}(C^l_{\mathbf{N}})$ // local numbering schemes, parameters associated with local rows ($\mathbf{p}_{\mathsf{r}}$) & columns ($\mathbf{p}_{\mathsf{c}}$) of $\mathbf{N}$
for $r = 0$, $r < R^l_{\mathbf{N}}$, $r{+}{+}$ do
    $\mathbf{p}_{\mathsf{r}}(r) = \mathbf{p}(\mathbf{N}$.rowInGlobalMat$(r)$ )
end
for $c = 0$, $c < C^l_{\mathbf{N}}$, $c{+}{+}$ do
    $\mathbf{p}_{\mathsf{c}}(c) = \mathbf{p}(\mathbf{N}$.colInGlobalMat$(c)$ )
end
// Loop over all groups of provided as OEQs
for $o = 0$ to $O - 1$ do
    g.at($o$).loadObservations()                    // Load all observations (and meta data) of group o from file
    // Loop over all observations of group o in blocks of $b_{obs}$
    for $b = 0$, $b <$g.at($o$).size(), $b{+}= b_{obs}$ do
        if  r == c then
            $\mathbf{A}_{\mathsf{r}}$.resize($b_{obs}$,$r^l_{\mathbf{N}}$)                // Process all $b_{obs}$ observations
            fillDesign($\mathbf{A}_{\mathsf{r}}$, g.at($o$).obs($b : b + b_{\mathrm{obs}}$), $\mathbf{p}_{\mathsf{r}}$ )   // Fill design matrices with observations
            $\mathbf{N}$.localMat()$+= 0.5$g.at($o$).w() $\mathbf{A}_{\mathsf{r}}^T \mathbf{A}_{\mathsf{r}}$    // Update local matrices of $\mathbf{N}$ (dsyrk)
        else
            // Process half of observations, (shared between (r, c) and (c, r))
            if  r > c then
                int $b_l = b_{obs} \div 2$;   int $b_s = b$
            else
                int $b_l = b_{obs} \div 2 + b_{obs}\%2$;   int $b_s = b + b_{obs} \div 2$
            end
            $\mathbf{A}_{\mathsf{r}}$.resize($b_l$,$r^l_{\mathbf{N}}$);   $\mathbf{A}_c$.resize($b_l$,$c^l_{\mathbf{N}}$)       // Fill design matrices with observations
            fillDesign($\mathbf{A}_{\mathsf{r}}$, g.at($o$).obs($b_s : b_s + b_l$), $\mathbf{p}_{\mathsf{r}}$ ),   fillDesign($\mathbf{A}_{\mathsf{c}}$, g.at($o$).obs($b_s : b_s + b_l$), $\mathbf{p}_{\mathsf{c}}$ )
            $\mathbf{N}$.localMat()$+=$ g.at($o$).w() $\mathbf{A}_{\mathsf{r}}^T \mathbf{A}_c$          // Update local matrices of $\mathbf{N}$ (dgemm)
        end
        // Computation of RHS (including MC trace RHS)
        if  $C^l_{\mathbf{n}} > 0$ then
            $\mathbf{A}_{\mathsf{r}}$.resize($b_{obs}$,$r^l_{\mathbf{N}}$)   $\boldsymbol{\ell}$.resize($b_{obs}$,$c^l_{\mathbf{n}}$)     // Process for all $b_{obs}$ observations
            fillDesign($\mathbf{A}_{\mathsf{r}}$, g.at($o$).obs($b : b + b_{\mathrm{obs}}$), $\mathbf{p}_{\mathsf{r}}$ )   // Fill design matrices with observations
            if r == 0 then
                for $c = 0$, $c < \boldsymbol{\ell}$.C(), $c{+}{+}$ do
                    for $r = 0$, $r < \boldsymbol{\ell}$.R(), $r{+}{+}$ do
                        if $\mathbf{n}$.colInGlobalMat(c)== 0 then
                            $\boldsymbol{\ell}$.localMat()$(r, c) =$ g.at($o$).l($b + r$)
                            $\lambda{+}=$ g.at($o$).w() $\boldsymbol{\ell}^T \boldsymbol{\ell}(0, 0)$, $M{+}= b_{\mathrm{obs}}$     // Update $\lambda$ (dsyrk) and M
                        else
                            if $\mathbf{n}$.colInGlobalMat(c)$\in \{1 + NK + oK, .., 1 + NK + (o + 1)K - 1\}$ then
                                $\boldsymbol{\ell}(r, c) = \pm 1$
                            end
                        end
                    end
                end
            end
            $\boldsymbol{\ell}$.Bcast(0, c, "colum")             // broadcast $\boldsymbol{\ell}$ to processes of same column
            n.localMat()$+=$ g.at($o$).w() $\mathbf{A}_{\mathsf{r}}^T \boldsymbol{\ell}$          // Update RHS (dgemm)
        end
    end
    // Same operations for the rest block ($M_o\%b_{obs}$)...
end
// Final synchronization between (r, c) and (c, r)
$\mathbf{N} = \mathbf{N} + \mathbf{N}^T$
return $\mathbf{N}, \mathbf{n}, \lambda, M$ // combined NEQs
```

Within the algorithm, they are stored as local symbolic numbering schemes on every process r, c, containing the sub-space of the parameters associated either with the rows ($\mathbf{p}_\mathsf{r}$) or with the columns ($\mathbf{p}_\mathsf{c}$) of the local matrix $\mathbf{N}^l_{\mathsf{r},\mathsf{c}}$.

Processing of Groups o (cf. Alg. 7.3, l. 14) As described in Alg. 7.3, the O groups are processed independently in a sequential loop (o-loop). There is no parallel processing of different groups o. For that reason, details on the implementation are only given for a single group o. Note that in the presented form of the implementation all processes involved read all observations from disk and have all observations available in the main memory. This is necessary as for this assembly of $\mathbf{N}$, all observations are required on all processes.

Setup of Local Serial Design Matrices (cf. Alg. 7.3, l. 19–32) The basic idea is to set up two design matrices locally, corresponding to the parameters of the locally stored NEQs $\mathbf{N}^l_{\mathsf{r},\mathsf{c}}$, i.e. $\mathbf{p}_\mathsf{r}$ and $\mathbf{p}_\mathsf{c}$. Setting up the serial locally stored matrices $\mathbf{A}_r$ for all b_obs observations and the parameters contained in $\mathbf{p}_r$ and $\mathbf{A}_c$ for all b_obs observations and the parameters contained in $\mathbf{p}_c$, the local update of $\mathbf{N}^l_{r,c}$ is

$$\mathbf{N}^l_{\mathsf{r},\mathsf{c}} += w_o \mathbf{A}_\mathsf{r}^T \mathbf{A}_\mathsf{c}. \tag{7.1}$$

This can be efficiently locally computed without communication using the BLAS-L3 routine `dgemm`. The processing of the blocks of size b_obs is serial (b-loop). Only the observations within a single block are processed in parallel (cf. Alg. 7.3, l. 18). Until now, the symmetry of $\mathbf{N}$ is not accounted for and the processes (r, c) and (c, r) perform the same operations as $\mathbf{N}^l_{\mathsf{r},\mathsf{c}} = \mathbf{N}^{lT}_{\mathsf{c},\mathsf{r}}$ ($\mathbf{A}_r = \mathbf{A}_c$ and $\mathbf{A}_c = \mathbf{A}_r$). To account for the symmetry, instead of setting up $\mathbf{A}_r$ and $\mathbf{A}_c$ for all b_obs observations, process (r, c) can set up the design matrices $\mathbf{A}_r$ and $\mathbf{A}_c$ for the first $b_\text{obs} \div 2$ observations and (c, r) for the second half of the observations ($b_\text{obs} \div 2 + b_\text{obs}\%2$).

Local Computation of $\mathbf{N}^l_{\mathsf{r},\mathsf{c}}$ for $\mathsf{r} = \mathsf{c}$ (cf. Alg. 7.3, l. 20–22) The computation for the processes with $\mathsf{r} = \mathsf{c}$ can be handled specially. With the prerequisites mentioned above (i.e. a quadratic compute core grid and quadratic sub-blocks), for the diagonal processes with $\mathsf{r} = \mathsf{c}$ it is the special case that $\mathbf{p}_\mathsf{r} = \mathbf{p}_\mathsf{c}$. The local part of the NEQs can be updated with a local computation via

$$\mathbf{N}^l_{\mathsf{i},\mathsf{i}} += 0.5 w_o \mathbf{A}_\mathsf{i}^T \mathbf{A}_\mathsf{i}, \tag{7.2}$$

where $\mathbf{A}_\mathsf{i}$ is a serial locally stored design matrix of dimension $b_\text{obs} \times C^l_\mathbf{N}$ assembled for all b_obs observations but only for the subspace $\mathbf{p}_r = \mathbf{p}_c$ of the parameters. The update can be locally computed with the BLAS-L3 `dsyrk` function. As `dysrk` is approximately two times faster than `dgemm`, the computation time of the diagonal cores is approximately the same as for the non-diagonal cores of the compute core grid, although approximately two times more observations are processed. The additional factor of 0.5 is required for the final synchronization step (cf. paragraph below and Alg. 7.3, l. 59).

Local Computation of $\mathbf{N}^l_{\mathsf{r},\mathsf{c}}$ for $\mathsf{r} \neq \mathsf{c}$ (cf. Alg. 7.3, l. 24–32) As described above, the observations for processes (r, c) and (c, r) are divided into two parts, each process has to process only half of them. As $\mathbf{p}_r \neq \mathbf{p}_c$, the matrices $\mathbf{A}_r$ and $\mathbf{A}_c$ differ on the cores. Both processes assemble $\mathbf{N}^l_{\mathsf{r},\mathsf{c}}$ respectively $\mathbf{N}^l_{\mathsf{c},\mathsf{r}}$ but only for approximately $b_\text{obs} \div 2$ observations. The local update of the local part of $\mathbf{N}$ is then again performed with a serial BLAS-L3 routine, i.e. `dgemm` to compute

$$\mathbf{N}^l_{\mathsf{r},\mathsf{c}} += w_o \mathbf{A}_\mathsf{r}^T \mathbf{A}_\mathsf{c}. \tag{7.3}$$

As only half of the observations are processed on the non-diagonal cores, the `dgemm` call takes approximately the same time as the `dysrk` call on the diagonal processes[10]. Later on, the local matrix data between processes (r,c) and (c,r) has to be synchronized via communication. But this is done with a single synchronization after the processing of all O groups. It is the final step in Alg. 7.3, l. 59 and not repeated within the o-loop.

Computation of the RHS (cf. Alg. 7.3, l. 34–53) Using the serial NEQ computation the computation, of the RHS needs to be adapted, too. In contrast to the computation of $\mathbf{N}$, some communication is required here. The generated random numbers for the additional MC RHS are required on every process of the same compute core grid's column, which stores local sub-matrices of $\mathbf{n}$. For simplicity, the first row of the compute core grid ($\mathsf{r} = 0, *$) sets up $\mathbf{l}$ as a serial matrix, fills it with the observations and draws the random samples (cf. Alg. 7.3, l. 36–37). Afterwards it is broadcasted to all processes of the column (cf. Alg. 7.3, l. 52). Thus, it is guaranteed that within the computation of

$$\mathbf{n}^l_{\mathsf{r},\mathsf{c}} += w_o \mathbf{A}^T_{\mathsf{r}} \mathbf{l} \tag{7.4}$$

the same random samples are used within a column of the compute core grid. The update is performed again with a local computation using the BLAS-L3 `dgemm` function. As a collective MPI `Bcast` operation is used, the computation is synchronized, and the processors not involved in the computation of the RHS are idle.

The implementation given in Alg. 7.3 is not the most efficient but the simplest one. After the update of $\mathbf{N}^l_{\mathsf{r},\mathsf{c}}$, all processes on which local matrices $\mathbf{n}^l_{\mathsf{r},\mathsf{c}}$ of $\mathbf{n}$ exist (i.e. $C^l_{\mathbf{n}} > 0$), completely set up $\mathbf{A}_{\mathsf{r}}$ for all b_{obs} observations (it is redundant, as parts were already computed for the update of $\mathbf{N}^l_{\mathsf{r},\mathsf{c}}$). Afterwards the computations are straightforward (cf. Alg. 7.3, l. 53). Tailored but complex optimizations, which require a lot of communication, are conceivable, but not integrated in the current implementation.

Final synchronization of N, (cf. Alg. 7.3, l. 59) The implemented method which accounts for the symmetry sets up $\mathbf{N}^l_{\mathsf{r},\mathsf{c}}$ and $\mathbf{N}^l_{\mathsf{c},\mathsf{r}}$ for half of the observations each. Thus, the joint matrix is

$$\mathbf{N} = \mathbf{N}^T + \mathbf{N}, \tag{7.5}$$

which can be computed with PBLAS routines. The factor 0.5 in (7.2) ensures that the "diagonal" blocks $\mathbf{N}^l_{\mathsf{i},\mathsf{i}}$ do not enter the $\mathbf{N}$ twice.

7.2.3 Solution of Combined NEQs and VCE

After the assembly of the NEQs for the NEQ and OEQ groups, the solution for the unknown parameters and the inversion of the NEQs is straightforward, as the system of NEQs is directly available as a block-cyclic distributed matrix. SCALAPACK routines for the Cholesky reduction of $\mathbf{N}$ (`pdpotrf` in the member function `chol`) are used to factorize $\mathbf{N}$. The unknown parameters are then determined via forward and backward substitution (calling `pdpotrs` in the member function `isSolutionOfCholReduce`). The first column in the solution vector $\mathbf{x}(:,0)$ are the unknown spherical harmonic coefficients, and the columns $\mathbf{Z} = \mathbf{x}(:,1:C-1)$ the additional parameters required for VCE. With the former right hand sides, the vector $\bar{\mathbf{P}} = \mathbf{n}(:,1:C-1)$ in (4.22), the trace term and thus the partial redundancy in (4.22) can be directly computed with PBLAS calls, as $\mathbf{Z}$ and $\bar{\mathbf{P}}$

[10] `dysrk` accounts for the symmetry of the result matrix and is thus approximately two times faster than `dgemm` of same matrix dimensions.

are block-cyclic distributed matrices. The only challenge is to operate on the proper sub-matrices to use the correct columns corresponding to the samples for group i. As a final step the sum of squared residuals Ω_n is computed for every NEQ group according to (4.8b), recalling the original NEQs from disk. Note that the correct parameter subset of $\mathbf{x}(:,0)$ is extracted and afterwards reordered to the numbering scheme associated with the original NEQs $\mathbf{N}_n$ and $\mathbf{n}_n$ using the already introduced reordering concepts (cf. Sect. 4.3.2 and 4.3.3). For the OEQs, the design matrices are set up as block-cyclic distributed matrices and $\mathbf{v}_o = \mathbf{A}_o\mathbf{x}(:,0) - \boldsymbol{\ell}_o$ is computed such that Ω_o directly is computed with (4.8a) again using computations for the block-cyclic distributed matrix $\mathbf{v}_o$. The final VCs directly follow from (4.22). They are stored in additions to the NEQs and the solution on disk.

7.3 A Closed-Loop Simulation Scenario

Within this section, a test scenario is described. A closed loop simulation is used to analyze and verify the implementation. It should be demonstrated, what scenarios can be analyzed with the derived implementation (mainly parameter space and amount of observations). At some points, the scenario is physically unrealistic (simplified physical functional and stochastic models), but this has no consequences on the feasibility study.

7.3.1 Simulation of Test Data Sets

The EGM2008 (Pavlis et al., 2012) model is used to generate $N+O = 21$ synthetic data sets. $N = 4$ data sets are created as NEQs. For this purpose, real satellite NEQs from GRACE, GOCE (SST and SGG) and SLR satellites are used to generate synthetic normal vectors based on EGM2008 and a noise vector is added. $O = 17$ groups are created as simulated observations of different gravity field functionals. The simulated NEQ and OEQ observation groups were analyzed using the implementation described above and a combined gravity field model was estimated from the data. In addition, unknown weights as inverse VCs are estimated.

7.3.1.1 Data Sets based on NEQs

Real satellite NEQs from SLR, GRACE and GOCE are used to generate synthetic normal vectors based on EGM2008. A noise generated from the real covariance matrix is added. Thus, the simulated normal equations are generated from

$$\mathbf{N}_n = \boldsymbol{\Sigma}_n^{-1} \tag{7.6}$$

$$\mathbf{n}_n = \boldsymbol{\Sigma}_n^{-1}\left(\mathbf{x}_{\text{EGM2008}} + \mathbf{e}_n\right), \text{ with } \mathbf{e}_n \text{ as a special realization of } \boldsymbol{\mathcal{E}}_n \sim \mathcal{N}\left(\mathbf{0}, \sigma_n^2\boldsymbol{\Sigma}_n\right). \tag{7.7}$$

$\mathbf{e}_n$ is a generated realization of the normal distributed noise vector $\boldsymbol{\mathcal{E}}_n$ with mean value $\mathbf{0}$ and variance $\sigma_n^2\boldsymbol{\Sigma}_n$. For $\boldsymbol{\Sigma}_n$, a real data covariance matrix of the mission (or observation technique) is used to have realistic characteristics of the model (noise characteristics and band-limited resolution). The number of observations M_n was taken from the real data models, too. Finally, for the later use of VCE $\lambda_n = \boldsymbol{\ell}_n^T\mathbf{Q}_{\boldsymbol{\ell}_n\boldsymbol{\ell}_n}^{-1}\boldsymbol{\ell}_n$ is computed for the simulated NEQs using (4.8a) via

$$\sigma_n^2 = \frac{\left(\mathbf{x}_{\text{EGM2008}} + \mathbf{e}_n\right)^T\mathbf{N}_n\left(\mathbf{x}_{\text{EGM2008}} + \mathbf{e}_n\right) - 2\left(\mathbf{x}_{\text{EGM2008}} + \mathbf{e}_n\right)^T\mathbf{n}_n + \lambda_n}{M_n - U_n} \tag{7.8}$$

and fixing σ_n^2 to a defined value, λ_n is then

$$\lambda_n = \sigma_n^2\left(M_n - U_n\right) - \left(\mathbf{x}_{\text{EGM2008}} + \mathbf{e}_n\right)^T\mathbf{N}_n\left(\mathbf{x}_{\text{EGM2008}} + \mathbf{e}_n\right) + 2\left(\mathbf{x}_{\text{EGM2008}} + \mathbf{e}_n\right)^T\mathbf{n}_n \tag{7.9}$$

U_n is the number of parameters the NEQ is set up for. All information provided for a simulated NEQ group s summarized in Tab. 7.1 and is available for the simulated data sets. The individual data sets generated and their characteristics are summarized in Tab. 7.2.

Table 7.2: Data sets based on NEQs used in the closed-loop simulation scenario.

n	name	NEQ/$\mathbf{\Sigma}_n$	reference	$\mathbf{n}_n$	d/o	σ_n
0	SLR	IWF-SLR	Maier et al. (2012)	EGM2008+ $\mathcal{N}\left(\mathbf{0}, \sigma_n^2 \mathbf{N}_n^{-1}\right)$	2–5	0.0606
1	GRACE	ITG-Grace2010s	Mayer-Gürr et al. (2010b)	EGM2008+ $\mathcal{N}\left(\mathbf{0}, \sigma_n^2 \mathbf{N}_n^{-1}\right)$	2–180	1.2910
2	GOCE_SST	EGM_TIM_RL04	Brockmann et al. (2013)	EGM2008+ $\mathcal{N}\left(\mathbf{0}, \sigma_n^2 \mathbf{N}_n^{-1}\right)$	2–130	0.7454
3	GOCE_SGG	EGM_TIM_RL04	Brockmann et al. (2013)	EGM2008+ $\mathcal{N}\left(\mathbf{0}, \sigma_n^2 \mathbf{N}_n^{-1}\right)$	2–250	0.8165

7.3.1.2 Data sets based on OEQs

The simulation of observations is straightforward. To simulate different data sets, a global grid, covering the whole Earth is partioned into different groups. Each observation group stands for a data set from a specific geographical region. Over land, gravity anomalies (in spherical approximation) are simulated from EGM2008 for spherical harmonic degrees 2–720 as point-wise measurements. As observation error, uncorrelated noise from a diagonal covariance matrix $\mathbf{\Sigma}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o} = \sigma_o^2 \boldsymbol{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}$ was added. The diagonal entries $\sqrt{\boldsymbol{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}(i,i)}$ of the cofactor matrix are randomly generated for every data point following an uniform distribution between 0.9 and 1.1. This corresponds to a variation of the standard deviation within a single group of ±10 %. This diagonal cofactor matrix is assumed to be known in the algorithm, whereas the value of σ_o used in generation of the noise is estimated from the data. A regular 0.2°×0.2° grid was used over the land areas and afterwards partioned to the geographical zones.

Over the oceans, altimetry observations along a virtual CryoSat-2 orbit (reference orbit provided by ESA) are simulated from EGM2008. As a simplification for this study, the dynamic ocean topography was neglected, thus it is assumed altimetry directly observes the static geoid[11]. Six 30-day sub-cycles (sc1–sc6) of different accuracies are introduced as individual observation groups for which a weight is estimated. The along-track observations were sampled with 1/3 Hz (for the simulations). The noise is generated in the same way as for the data sets over land. The typical along track error characteristic of altimetric measurements was neglected in the simulation, instead a white noise was added. For that reason, the standard deviation of a single observations was chosen higher than expected for real data with an a-long track error characteristic.

All necessary information on all $O = 17$ data sets is listed in Tab. 7.3. The data sets (the pointwise standard deviations of the observations) are in addition shown in Fig. 7.1 (accuracy levels in terms of standard deviations of the involved data sets).

7.3.2 Results of the Closed Loop Simulation

The developed software is used to derive the spherical harmonic coefficients for d/o 2–720 (519 837 parameters, 2 TB NEQ matrix) and the estimates for the unknown weights from the simulated data introduced above ($> 4 \cdot 10^6$ observations plus 4 NEQ groups). This section only demonstrates the estimates for the parameters and the covariance and does not cover the runtime analysis of implementation which is covered in Sect. 7.4.

The simulated data sets are combined to compute a least squares solution together with the weights. The in a computational sense rigorous solution is computed as a demonstrator for an application of the implementation to real data. Starting the iteration process with $\sigma_o = \sigma_n = 1.0$ for the VC and thus $w_i = 1$, $\eta_{\max} = 4$ VCE iterations are performed. $K = 5$ additional RHS for the Monte Carlo based trace estimation were used within VCE. All in all the system was set up and

[11]For studies, where parts of the derived implementation were used to co-estimate the MDT, see Becker et al. (2014a, 2013, 2014b).

Table 7.3: Data sets o introduced to the algorithm on the level of observations and used in the simulation. The observations were generated from EGM2008 for d/o 2–720.

o	name	functional	n_o	σ_o	$\sqrt{\boldsymbol{Q}_{\mathbf{l}_o\mathbf{l}_o}(i,i)}$
0	CryoSat-2 sc1	geoid height	577 272	12.0 cm	$\sim \mathcal{U}(0.9, 1.1)$
1	CryoSat-2 sc2	geoid height	577 293	13.0 cm	$\sim \mathcal{U}(0.9, 1.1)$
2	CryoSat-2 sc3	geoid height	577 479	14.0 cm	$\sim \mathcal{U}(0.9, 1.1)$
3	CryoSat-2 sc4	geoid height	577 211	15.0 cm	$\sim \mathcal{U}(0.9, 1.1)$
4	CryoSat-2 sc5	geoid height	577 479	12.1 cm	$\sim \mathcal{U}(0.9, 1.1)$
5	CryoSat-2 sc6	geoid height	577 211	13.1 cm	$\sim \mathcal{U}(0.9, 1.1)$
6	Africa	anomalies	63 750	21.0 mGal	$\sim \mathcal{U}(0.9, 1.1)$
7	Antarctica	anomalies	155 576	22.0 mGal	$\sim \mathcal{U}(0.9, 1.1)$
8	Australia	anomalies	17 276	14.0 mGal	$\sim \mathcal{U}(0.9, 1.1)$
9	Eurasia	anomalies	163 691	13.0 mGal	$\sim \mathcal{U}(0.9, 1.1)$
10	Greenland	anomalies	17 061	24.0 mGal	$\sim \mathcal{U}(0.9, 1.1)$
11	Indonesia	anomalies	5 095	17.0 mGal	$\sim \mathcal{U}(0.9, 1.1)$
12	Island	anomalies	497	12.0 mGal	$\sim \mathcal{U}(0.9, 1.1)$
13	New Zealand	anomalies	724	11.0 mGal	$\sim \mathcal{U}(0.9, 1.1)$
14	North America	anomalies	75 048	10.0 mGal	$\sim \mathcal{U}(0.9, 1.1)$
15	South America	anomalies	38 504	25.0 mGal	$\sim \mathcal{U}(0.9, 1.1)$
16	North Pole Cap	anomalies	23 400	23.0 mGal	$\sim \mathcal{U}(0.9, 1.1)$

solved for 106 RHS ($K(O+N)+1$). All steps introduced in Sect. 7.2 (NEQ assembly, solution and VCE) are repeated four times with updated weights. Fig. 7.2 shows the four derived solutions (named FULL_00–FULL_03) as solid lines in terms of degree variances with respect to the "true" EGM2008 model (black solid line). In addition, the estimated formal errors are shown as dashed lines. The sub-plot which covers the whole spectrum (degrees 2–720) shows only a different behavior of the solution FULL_00 and the other solutions. A difference between FULL_01, FULL_02 and FULL_03 in the coefficient differences and in the formal errors is not visible. As the standard deviations are all assumed to be 1.0 in FULL_00, i) the error estimates are much to optimistic and ii) the combination of the different data sets does not work properly. As the surface data enters the solution with higher weight, it dominates the solution, also in the overlapping areas with the satellite data (available as NEQs in the degree range 2–250). After the first VCE iteration, the estimated standard deviations are in the correct order of magnitude. The solution, as well as the error estimates significantly improve (green, blue and orange line). As the formal error estimates and the differences agree well, not all lines are visible in the plot.

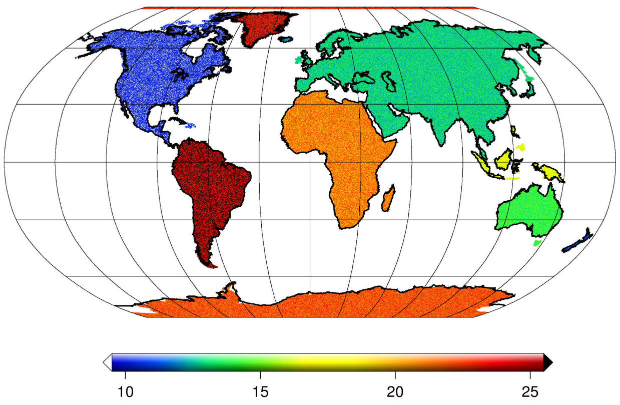
Three additional zooms are shown in Fig. 7.2, zooming into the low, medium and high degree range. The zoom plots show some additional differences between the FULL_01 and FULL_03 solution. Larger differences occur in the error estimates. Using degree variances only, differences between the FULL_02 and FULL_03 solutions are not visible. As it is a simulation only, used for a proof of concept, no other comparisons are done for the different solutions. To demonstrate that the developed software can handle the presented simulation scenario, the degree variance comparison is sufficient. Demonstrating additional differences between the solutions has no added value.

To verify the estimated variance components used for relative weighting, Tab. 7.4 shows the initial, the estimated as well as the true standard deviations ($\sigma_{n,o} = \sqrt{1/w_{n,o}}$) for all groups n and o used in the simulation. The estimates of all four iterations are provided. The relative error with respect to the true value

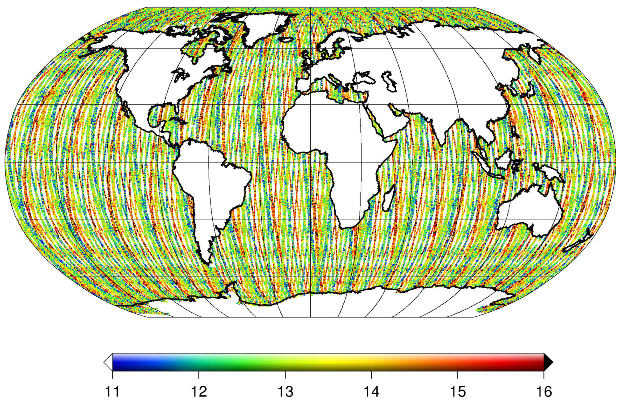
$$\delta_{n,o}^{(\nu)} := \frac{|\sigma_{n,o}^{(\nu)} - \sigma_{n,o}^{\text{true}}|}{\sigma_{n,o}^{\text{true}}} \tag{7.10}$$

is provided for the first and the final estimate.

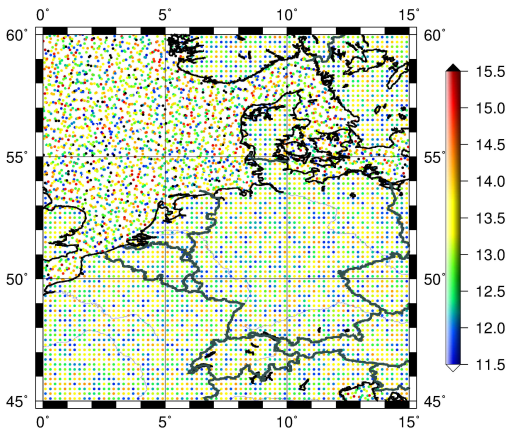
After four iterations, all standard deviations, except for four groups, are recovered with an error smaller or equal to 0.5 %. Of course, the larger the group, the better the estimate for the variance



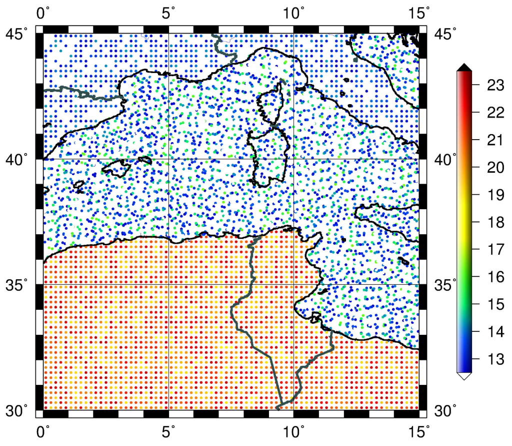
(a) Standard deviations of generated observation of the surface data (in mGal).



(b) Standard deviations of generated observation over the ocean areas (in cm, subset only).



(c) Local zoom into the standard deviations of generated observation over the ocean areas (in cm over the ocean, mGal in land areas).

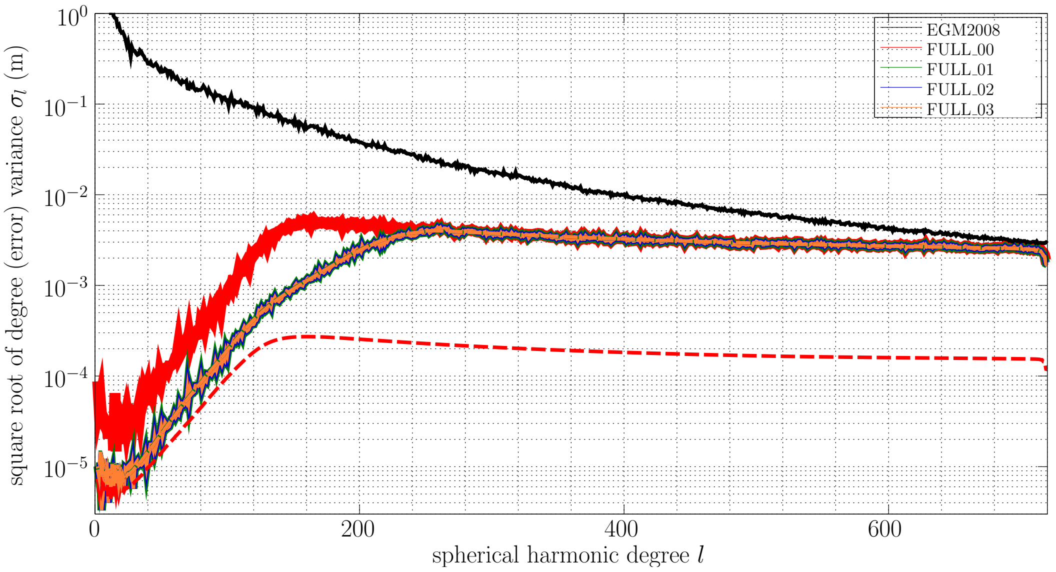


(d) Local zoom into the standard deviations of generated observation over the ocean areas (in cm over the ocean, mGal in land areas).

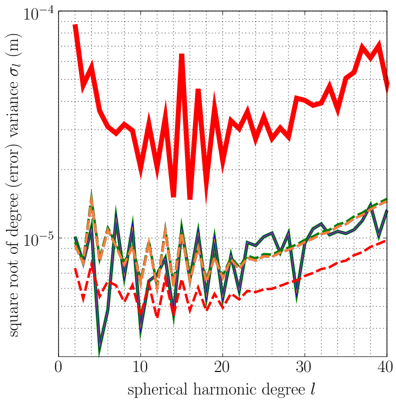
Figure 7.1: Accuracies of the simulated observation data sets. It should be noted that different scales and units are used.

component, as the number of samples the VC is estimated from is higher. For the NEQ groups, larger refers to the number of parameters, whereas for the OEQ groups it refers to the number of observations (dimension of the generated samples). The four groups, where the error remains above 0.5 % are the four smallest groups. On the one hand, i.e. group $n = 0$ (SLR NEQs), in which the sample used to generate the synthetic RHS has only 32 entries. A small sample of 32 values was used to generate the stochastic RHS, which is used to estimate the VC later on. Nevertheless, the error of that group is 0.9 % only. The same holds for the remaining larger errors in the OEQ groups 11, 12 and 13. The errors there are 1.7 %, 6.3 % and 3.8 %. But again the number of observations is small in the groups (5095, 497 and 724 respectively). These groups are much smaller compared to the other groups (4 to 1000 times, cf. Tab. 7.3).

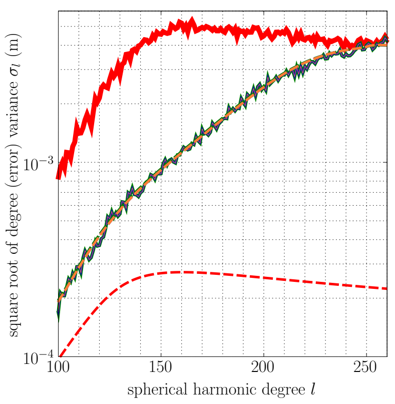
Another source of errors in the estimates of the variance components is the stochastic part within the used Monte Carlo based estimator. This part covers only the partial redundancy $r_{n,o}$ (cf. (4.14) and (4.17)). Effects of an error in the estimates of the partial redundancy, or more precisely in $\Upsilon_{n,o}$ are studied in Brockmann and Schuh (2010). The analysis is not performed here, as the error strongly depends on the configuration of the data sets. Because the estimates were derived as the mean value from $K = 5$ samples per group, the error from the stochastic trace estimation is assumed



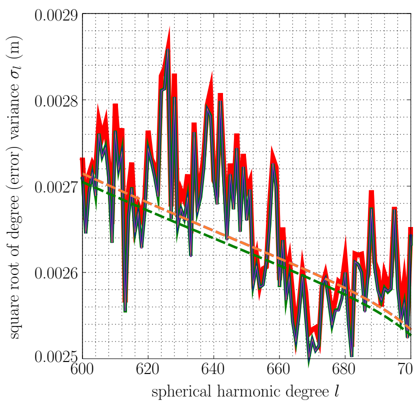
(a) For the whole spectrum.



(b) Zoom to low degrees.



(c) Zoom to medium degrees.



(d) Zoom to high degrees (linear!).

Figure 7.2: Degree (error) variances of the four VCE solutions with respect to the true EGM2008 model. Degree error variances estimated from coefficient differences are shown as solid lines, whereas the error variances computed from formal errors are shown as dashed lines. Note that the formal error estimates are hard to see for iterations 1–3, as they agree with the solid lines.

to be small. Some details are studied in Sect. 8.4.1, where the same scenario is analyzed with the iterative solver.

7.3.3 Application of the Full Covariance Matrix as Demonstrator

The assembled combined NEQ — or its inverse as the full covariance matrix — can be easily used in a further analysis or in applications requiring the full NEQ or covariance matrix within the developed framework. To demonstrate that for instance the tool for variance propagation can be easily used with the high resolution NEQ, Fig. 7.3(a) shows the propagation of the full covariance matrix to geoid height errors for the full spectrum (degree 2 to 720) to points on a regular $0.2° \times 0.2°$ grid. The runtime was less than 3.5 h on a 64×64 compute core grid.

Table 7.4: Standard devitions $\sigma_{n,o} = 1/\sqrt{w_{n,o}}$ as derived by VCE for OEQ and NEQ groups. The values are provided for all iterations performed. The last two columns provide the relative error w.r.t. the true value which was used in data generation.

group n/o		$\sigma_{n,o}^{(0)}$	$\sigma_{n,o}^{(1)}$	$\sigma_{n,o}^{(2)}$	$\sigma_{n,o}^{(3)}$	$\sigma_{n,o}^{(4)}$	$\sigma_{n,o}^{\text{true}}$	$\frac{\lvert\sigma_{n,o}^{(1)}-\sigma_{n,o}^{\text{true}}\rvert}{\sigma_{n,o}^{\text{true}}}$	$\frac{\lvert\sigma_{n,o}^{(4)}-\sigma_{n,o}^{\text{true}}\rvert}{\sigma_{n,o}^{\text{true}}}$
n	0	1.0000	0.0642	0.0612	0.0612	0.0612	0.0606	6.0 %	0.9 %
	1	1.0000	1.3187	1.2900	1.2900	1.2900	1.2910	2.1 %	0.1 %
	2	1.0000	0.7460	0.7453	0.7453	0.7453	0.7454	0.1 %	0.0 %
	3	1.0000	0.8347	0.8165	0.8165	0.8165	0.8165	2.2 %	0.0 %
o	0	1.0000	12.1605	12.0208	12.0063	12.0045	12.0000	1.3 %	0.0 %
	1	1.0000	13.0110	12.9915	12.9922	12.9932	13.0000	0.1 %	0.1 %
	2	1.0000	13.9302	13.9988	14.0084	14.0084	14.0000	0.5 %	0.1 %
	3	1.0000	14.8300	14.9925	15.0086	15.0099	15.0000	1.1 %	0.1 %
	4	1.0000	12.2302	12.1252	12.1139	12.1136	12.1000	1.1 %	0.1 %
	5	1.0000	13.1059	13.0771	13.0756	13.0757	13.1000	0.0 %	0.2 %
	6	1.0000	20.9709	21.0673	21.0717	21.0985	21.0000	0.1 %	0.5 %
	7	1.0000	21.9317	21.9460	21.9471	21.9421	22.0000	0.3 %	0.3 %
	8	1.0000	13.9250	13.9456	13.9732	13.9924	14.0000	0.5 %	0.1 %
	9	1.0000	13.0154	13.0258	13.0406	13.0337	13.0000	0.1 %	0.3 %
	10	1.0000	24.0069	24.0118	24.0505	24.0857	24.0000	0.0 %	0.4 %
	11	1.0000	16.4228	16.5811	16.7056	16.7105	17.0000	3.4 %	1.7 %
	12	1.0000	12.6329	12.6125	12.5659	12.7543	12.0000	5.3 %	6.3 %
	13	1.0000	10.6277	10.6284	10.5441	10.5860	11.0000	3.4 %	3.8 %
	14	1.0000	10.0205	9.9896	9.9912	9.9864	10.0000	0.2 %	0.1 %
	15	1.0000	24.7752	24.9982	24.9744	25.0293	25.0000	0.9 %	0.1 %
	16	1.0000	22.9758	22.9806	22.9874	22.9867	23.0000	0.1 %	0.1 %

In addition, Fig. 7.3(b) shows the eigenvalues λ of the full NEQ matrix, computed with the SCALAPACK `pdsyev` function. The condition κ of the NEQ is

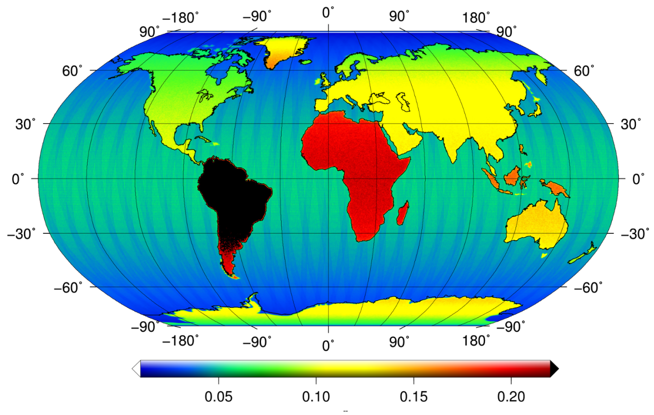
$$\kappa := \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}} = 1.97 \cdot 10^8. \tag{7.11}$$

The runtime was 13.5 h on a 64×64 compute core grid.

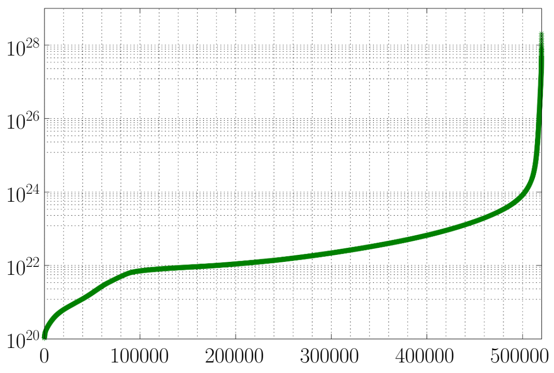
The results should not be analyzed further, but are used to show that the computations can be easily done using the developed framework. The results will be used in the future to evaluate the quality of the preconditioners within iterative solvers (e.g. for the preconditioner models discussed in Sect. 8.3.1.1). The huge NEQ can be easily handled and operated on using the developed framework. Further analysis, like computations of eigenvectors, or contribution analysis can be performed the same way as done for the GOCE application (cf. Sect. 6.5.4) for the high resolution NEQs.

7.4 Runtime Analysis of Assembly and Solution

This section shows the runtime characteristics of the implemented solver. The overall characteristics as well as the characteristics of the individual steps are assessed. The focus is put to the scaling behavior of the individual steps. As the most demanding step is the update of the NEQs with the high resolution groups provided as OEQs, this step is analyzed in more detail. The simulation scenario mentioned above was used for the performance analysis. To save computing time, only partial data sets were analyzed in the different setups modifying the compute core grid and/or the block-cyclic distribution. To get an idea, the runtime of a single VCE iteration for the whole simulation mentioned above was 10.5 h using a 64×64 ($\mathsf{N} = 4096$ cores) compute core grid. Thus, the assembly and solution of NEQ with 520 000 parameters from 4 million observations is possible in 10.5 hours (including VCE). The runtime was 12 h, including the storage of the 2 TB NEQs on disk. All (performance) simulations were performed on JUROPA at FZ Jülich (Jülich, 2013).



(a) Full covariance matrix propagated to geoid height errors (m).



(b) Spectrum of eigenvalues of the full NEQ matrix.

Figure 7.3: Application and further analysis of the high resolution full covariance matrix (or NEQ) to demonstrate that a further use of the huge covariance matrix/NEQ matrix is easily possible within the derived framework.

7.4.1 Assembly of NEQs

7.4.1.1 Update with NEQ Groups

As long as the combined NEQ $\mathbf{N}$ should be composed from high resolution OEQ and band-limited NEQ groups, the update of the $\mathbf{N}$ with the groups n provided as NEQs is a less challenging step with respect to runtime. Provided that the full NEQs (as in the test scenario, which is close to a probably real-data scenario) only cover a small subset of the parameter space and their number N is small, the processing time of the groups n remain insignificant. E.g., $\mathbf{N}$ is updated with the tiny SLR NEQs in about 10 s (using a compute core grid of 64×64, including all necessary steps introduced in Sect. 7.2.1). For this tiny NEQs, it does not make sense to study the compute core grid dimension and the performance using other parameters, as the compute core grid is totally over-dimensioned for that system of NEQs. For the largest NEQs used here, the 30 GB GOCE SGG NEQs, the whole update step takes less then 500 s (already including the computation of w_n), varying with the compute core grid. Compared to the runtime of the processing of the OEQ groups it is the same as needed for the processing of 75 000 observations and thus about 1.5 % of the total runtime. It does not make sense to optimize the block-cyclic distribution or compute core grid with respect to the NEQ groups. Of course, in other scenarios, e.g. only combining preprocessed (higher-dimensional) NEQs, an analysis makes sense. Anyway, the percentages from the individual steps involved in NEQ processing are given in Tab. 7.5 to provide information where potential optimizations could be useful and the computing time is spend.

The runtime needed is dominated (32 %) by the reordering step (cf. Alg. 7.1, l. 14–18). The runtime of the step is significantly influenced by the numbering schemes. Depending on $\mathbf{p}$ and $\mathbf{p}_n$, the number of required row and column interchanges significantly vary. In the worst case, U_n row and column interchanges have to be performed. In the best case, i.e. $\mathbf{p} = \mathbf{p}_n(1 : U_n)$, no row and column interchanges are needed at all. The second most expensive steps are the I/O operations (26 %), reading the original NEQs from disk twice (for the update of $\mathbf{N}$ and the sample transformation + VCE). If the conversions of the NEQs (physical constants and reduction of the RHS) are included, these steps require (37 %) of the total runtime. The remaining time is mainly spend for the RHS generation (25 %), covering the individual steps of the Cholesky factorization of $\mathbf{N}_n$, the transformation of samples and reordering of RHS for the update of the combined RHS. Note that the runtime for all steps significantly depends on the parameter space and thus the dimension of $\mathbf{N}_n$.

7.4.1.2 Update with OEQ Groups

Only the processing of the group $o = 0$ is used for the runtime analysis of the update of $\mathbf{N}$ with the groups provided as OEQs as the runtime is linear in the number of the observations (sequential loop over observations in blocks of b_{obs}).

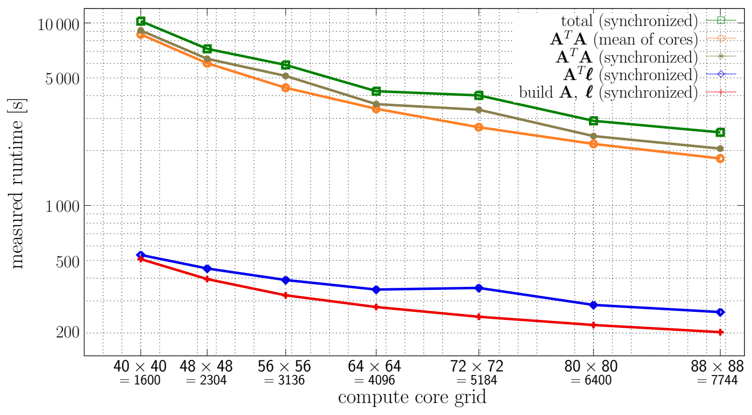
Local Computation of NEQs Using Serial Design Matrices Fig. 7.4 shows the runtime analysis and the scaling behavior of the method as introduced in Sect. 7.2.2.2 (especially Alg. 7.3). The block size b_{obs} is automatically determined at runtime, such that (if adequate memory is available) the resulting local serial design matrices are approximately square ($b_{\text{obs}} = 2\,(U \div \mathsf{R})$) matrices. The block-cyclic distribution parameters where fixed to $b_r = b_c = 64$ and a quadratic compute core grid was varied from 40×40 (i.e. $\mathsf{N} = 1600$) to 88×88 (i.e. $\mathsf{N} = 7744$).

The green curves show the runtime and the scaling results for the whole processing step, i.e. the update of $\mathbf{N}$ with the observations of OEQ group $o = 0$ (l. 16–57 in Alg. 7.3). For all compute core grids analyzed the runtime is significantly decreasing if the number of cores in the compute core grid increases. The scaling shows a close to linear behavior for all grids. For the extrema analyzed, i.e. using 4.84 times more cores (7744 instead of 1600), the runtime is 4.1 times faster. For this very high number of cores, this scaling behavior is still very good. The curve is not smooth (e.g. peak for the 72×72 grid), as the measurement of the wall clock time result from a single run only. The higher runtime might be a random result, related to the hardware as the involved nodes differ using different compute core grids or to a general higher activity in the HPC environment. Alternatively, it might be related to an inappropriate combination of the block-cyclic distribution, compute core grid and the number of observations and parameters analyzed for that setup. A final explanation could not be identified.

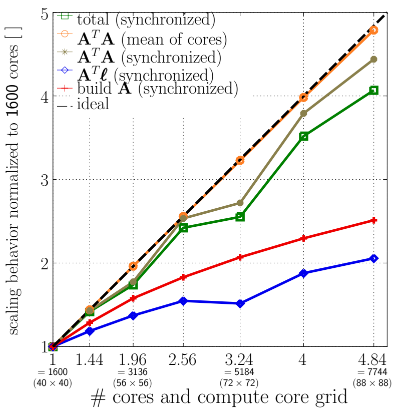
The brown and the orange curves show the runtime behavior for the update of $\mathbf{N}$ with the observations of group o, i.e. the computation of $\mathbf{N}^l_{\mathsf{r,c}} += \mathbf{A}^T_{\mathsf{r}}\mathbf{A}_{\mathsf{c}}$ (cf. l. 22 respectively 32 in Alg. 7.3). The runtime for the local computation of $\mathbf{N}^l_{\mathsf{r,c}}$ is measured in every pass of the b-loop where b_{obs} observations are processed at once. The runtime is measured individually on every core. After that computation and before the update of the RHS, a synchronization step is required (as global MPI

Table 7.5: Runtime needed for the steps of the update of $\mathbf{N}$ with the GOCE SGG NEQs using a 64×64 compute core grid ($b_r = b_c = 64$). In addition the percentage of the steps at the total runtime is provided.

	operation	time (s)	%
cf. Alg. 7.1, l. 10	read NEQs	51.1	13.6
cf. Alg. 7.1, l. 13	conversions (inc. RHS reduction)	20.0	5.3
cf. Alg. 7.1, l. 14–18	extend and reorder	118.7	31.6
cf. Alg. 7.1, l. 19	update $\mathbf{N}$, $\mathbf{n}$	0.4	0.1
cf. Alg. 7.1, l. 34	read NEQs	49.8	13.2
cf. Alg. 7.1, l. 35	conversions (inc. RHS reduction)	20.3	5.4
for (4.8b)	reorder and trim $\mathbf{x}$	6.4	1.7
for (4.8b)	compute Ω_n	22.3	5.9
cf. Alg. 7.1, l. 36	Cholesky of $\mathbf{N}_n$	31.1	8.3
cf. Alg. 7.1, l. 27–42	generate and transform MC RHS	28.5	7.6
cf. Alg. 7.1, l. 39f.	extend and reorder MC RHS	27.4	7.3
cf. Alg. 7.1	total	376.0	100.0



(a) Absolute runtime measured for the operations for the assembly of the NEQs for group $o = 0$ on different compute core grids.



(b) Scaling behavior of the operations normalized to the runtime using the 40 × 40 = 1600 compute core grid.

Figure 7.4: Measured performance of the implemented algorithm for the computation of NEQs from the OEQ groups using serial design matrices. Results are shown for all operations involved (green) and the three most intensive operations (computation of $\mathbf{A}_o^T\mathbf{A}_o$ (orange, brown), computation of the RHS (blue) and the (repeated) setups of $\mathbf{A}_o$ (red)). The difference between the three most intensive steps shown and the total runtime is not shown here. It is in the order of less then 10 s.

communication is performed to share the random MC samples) in the implementation. Two variations of runtime measurements are shown. For every core, the sum of all b-loop passes is computed and the average of the runtime from all cores is shown as the orange line. This runtime measurement is a kind of unsynchronized measurement, as the synchronization time is neglected and the timer starts when the computation starts and stops directly when the local computation finishes. Some cores are idle and have to wait until all other cores performed the computation. Thus the pure runtime of the computation as well as the scaling are very smooth and the scaling is good, linear and nearly ideal. The mean runtime measurement covers only the serial computation time (cf. l. 22 respectively 32 in Alg. 7.3).

In contrast to that, the brown line shows the runtime for the $\mathbf{N}^l_{\mathsf{r,c}}$ computations, including the synchronization time. Instead of an average value, the runtime of the slowest core is summed up in every pass of the b-loop. Thus the total runtime gets slower as it is driven by the slowest computation in every loop pass, and consequently the scaling slightly degrades. Nevertheless the performance measurement is better suited for a fair computation of the scaling behavior. Although the scaling is not ideal, the behavior is still good (4.4 instead of the perfect 4.84) and remains linear. As the total runtime is dominated by this computation step (approximately 81 %–89 %), the brown curve behaves as the green curve, showing the total runtime. Due to several reasons, the individual computing time on the individual cores differs in the range of −1 %–13 % compared to the average computing time. Whereas the effect of load balancing, i.e. differences in the size of the local matrices $\mathbf{N}^l_{\mathsf{r,c}}$ on different cores can be quantified with +1 %–1 %, the major systematic effect of different computing time is that the computation on the diagonal cores (cf. l. 22 in Alg. 7.3) is slightly slower (`dsyrk` instead of `dgemm` but for twice the number of observations). This effect can be quantified with 3.5 %–7 %. For the maximal value seen in the study, i.e. a 13 % slower runtime, the reason could not be identified. It was measured on a core in the middle of the compute core grid, without special properties with respect to the data distribution or communication requirements. It might be a random effect or related to system or hardware events.

The blue curve covers the runtime of the RHS computation, including the generation of the Monte Carlo samples (l. 34–53 in Alg. 7.3). The runtime for this step is decreasing with an increasing number of cores. But the scaling is far away from linear. This is to be expected, because only the cores storing local matrices of the RHS vector perform computations. For the setup analyzed, i.e., 106 RHS and $b_r = 64$, only two columns of the compute core grid are involved in the RHS computations. Thus the scaling is not linear but related to the square root of $\sqrt{N}$ (or more precisely $\sqrt{\mathsf{N}/\mathsf{N}_1}$). For the extrema analyzed, i.e. using 4.84 times more cores (7744 instead of 1600), the runtime is 2.1 times faster ($\sqrt{4.84} = 2.2$). Note that this computations are synchronized. Collective communication along a column of the compute core grid is used to share the generated random samples.

The last runtime measurement shown is the one for the repeated setup of $\mathbf{A}$ ($\mathbf{A}_r$ and $\mathbf{A}_c$ in l. 34–53 in Alg. 7.3). Although the scaling is not linear, the runtime increases with the number of additional cores in the compute core grid. Only 5 %–8 % of the runtime are spend for the setup of $\mathbf{A}$ (cf. l. 20–21 or l. 30–31). The scaling of this step is closely related to the numbering scheme, which can be optimized to reduce redundant recursive computations during Legendre function computation (cf. Sect. 8.3.3.2). The optimized numbering scheme introduced there was not included in this computation. The runtime and the scaling behavior of this step depends on the block-cyclic distribution parameters, the compute core grid and significantly the chosen predefined target numbering scheme.

All other steps of Alg. 7.3 like data I/O and further initializations are not shown in detail. These operations are very fast and are in the order of less than 10 s for all compute core grids (Difference of total runtime and the runtime required for the three steps mentioned).

Distributed Computation of NEQs using Block-cyclic Distributed Matrices For applications of the implemented software where correlated observations are analyzed, the method which sets up the design matrix as a block-cyclic distributed matrix (cf. Sect. 7.2.2.1) has to be used. The performance is not studied in much detail, as it is dominated by the `pdsyrk` performance and thus depends only on the SCALAPACK/PBLAS performance. Nevertheless, some general studies on the performance are provided, as this studies for such high dimensional matrices are rare in literature. Again, only the observations of group $o = 0$ were analyzed.

Comparison to the "serial" Implementation and Compute Core Grid Dependence
Within a first study, the runtime is compared to the tailored method studied above. The simulation was performed on the 40×40, 64×64 and 88×88 compute core grid. $b_r = b_c = 64$ was chosen and $b_{\text{obs}} = 250\,000$ (for 40×40 compute core grid, memory limitation) and $b_{\text{obs}} = 520\,000$ (for both other grids, approximately square design matrix) were processed in a single pass of the b-loop.

Tab. 7.6 shows the wall clock time for the computation of $\mathbf{N}$ for the first group $o = 0$. The main conclusion is, that the self implemented version for uncorrelated observation is always faster, from 2.2 to 1.3 times, than the standard implementation using the block-cyclic distributed matrices for the observation equations. The performance of the block-cyclic method for the 40×40 compute core grid is poor. This is related to the PBLAS performance of `pdsyrk`, as due to memory limitation, the block-cyclic distributed design matrix is not quadratic, but of dimension $250\,000 \times 519\,837$ within the b loop. In the "serial" method, parts of $\mathbf{A}_o$ are set up more then once. It is obvious, that the runtime for the setup of $\mathbf{A}_o$ is much higher. The same holds for the computation of the RHS, where in the serial method, parts of $\mathbf{A}_o$ are newly set up. In the block-cyclic method $\mathbf{A}_o$ is only computed once. In addition, within the block-cyclic methods, all cores are involved in the RHS computation, as every core stores parts of $\mathbf{A}_o$.

Due to the limited number of compute core grids studied, the scaling of the block-cyclic method is not presented in detail. As in addition the performance of the smallest compute core grid is poor (see above), the numbers of the scaling are not meaningful. The scaling for the block-cyclic method from the 64×64 to 88×88 compute core grid (1.9 times more cores) can be realistically

Table 7.6: Runtime measurements comparing the two implementations for the computations of NEQs from OEQs (serial cf. Alg. 7.2, Sect. 7.2.2.1, block-cyclic cf. Alg. 7.3, Sect. 7.2.2.2). Whereas the compute core grid is varied, the block-cyclic distribution is fixed for both methods to $b_r = b_c = 64$.

grid	implementation	runtime	$\mathbf{A}^T\mathbf{A}$	$\mathbf{A}^T\boldsymbol{\ell}$	build $\mathbf{A}$	other
		s	s	s	s	s
40×40	serial	10 237	9 098	536	508	95
	block-cyclic	22 951	not separately measured			
64×64	serial	4 228	3 591	346	278	13
	block-cyclic	6 010	6 001	4	5	0
88×88	serial	2 517	2 050	260	202	5
	block-cyclic	3 360	3 351	6	2	1

determined. The runtime for all operations is 1.8 times faster, which is slightly better than for the serial implementation, where the scaling is 1.7.

Block-cyclic Distribution
The compute core grid was fixed to 64×64 and the parameters of the block-cyclic distribution b_r and b_c ware varied. Fig. 7.5(a) shows the runtime for different cases with $b_r = b_c$. Only the characteristics of the total runtime is shown, as it is dominated by the runtime for the computation of $\mathbf{N}+ = \mathbf{A}_o^T\mathbf{A}_o$. For all cases analyzed, the runtime for the setup of $\mathbf{A}_o^T$ as well as the computation of $\mathbf{n}+ = \mathbf{A}_o^T\boldsymbol{\ell}_o$ is insignificant and varies between 3 s and 7 s. Three simple conclusions follow:

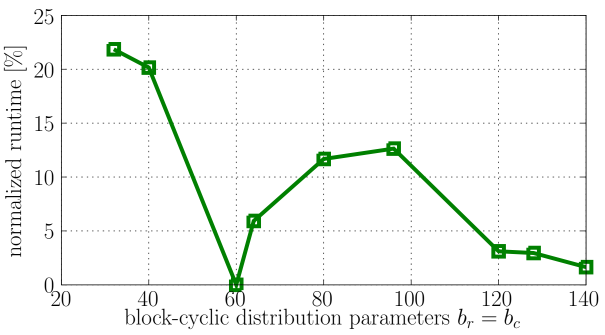
- The minimum is found for $b_r = b_c = 60$. The suggested default value performs slightly worse (5 % additional runtime).
- Small block-sizes perform worst, about 20 % additional runtime is needed.
- The largest block-sizes tested, $b_r = b_c > 120$ again have close to minimal runtime characteristics.

Choosing the default value $b_r = b_c = 64$ again serves a good runtime (for `pdsyrk`, as dominant).

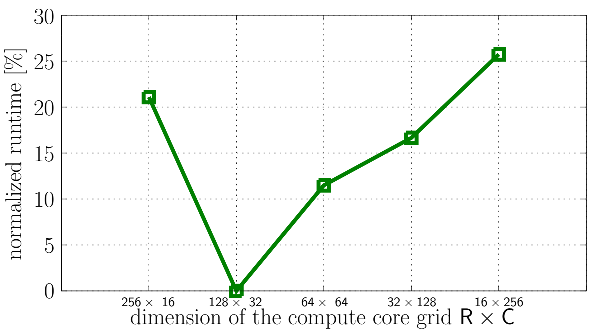
Shape of the Compute Core Grid
The shape of the compute core grid was studied for a compute core grid with $\mathsf{N} = 4096$ compute cores. The update of $\mathbf{N}$ with group $o = 0$ was repeated varying the shape of the compute core grid. The block-cyclic distribution parameters were fixed to $b_r = b_c = 64$. Fig. 7.5 shows the runtime changes normalized to the value compute core grid for which the runtime was minimal (128×32). As again the runtime is dominated by the computation of $\mathbf{N}+ = \mathbf{A}_o^T\mathbf{A}_o$ and thus `pdsyrk`, the following conclusions mainly hold for the SCLAPACK function:

- The quadratic compute core grid is the second best, nevertheless, 11 % additional runtime is required.
- The close to quadratic grid, with more rows than columns (128×32) has the best performance characteristics.
- The other compute core grids tested perform worse, with 15 % to 25 % additional runtime required.

As for both, the block-cyclic distribution parameters and the shape of the compute core grid improvements of 5 % to 10 % are observable, it is useful to perform some tests on a subset of the observations on the used hardware. The runtime reduction for a full-scale analysis of a huge data volume can be significant. The quadratic compute core grid and the suggested default values for $b_r = b_c = 64$ seem to be good stating values, but additional performance gain in the range of 10 % to 15 % remains possible.



(a) Runtime change for different parameters of the block-cyclic distribution. Shown is the additional runtime normalized to the minimal runtime obtained for $b_r = b_c = 60$.



(b) Runtime change for different shapes of the compute core grid. Shown is the additional runtime, normalized to the minimal runtime obtained for the 128×32 compute core grid.

Figure 7.5: Dependence of the runtime for NEQ assembly of the compute core grid and the block-cyclic distribution.

7.4.2 Solving and Inverting the NEQs

Deriving the solution of the NEQs via forward and backward substitution is an easy task working with block-cyclic distributed matrices and SCALAPACK, as this tasks can directly be solved with existing SCALAPACK functions (i.e. `pdpotrf` for the Cholesky decomposition, `pdpotrs` for the forward and backward substitution and `pdpotri` for the computation of the inverse from the already factorized matrix). Within the runtime analysis shown, the NEQ from the simulation scenario was used ($519\,837 \times 519\,837$) with a RHS of dimension ($519\,837 \times 200$).

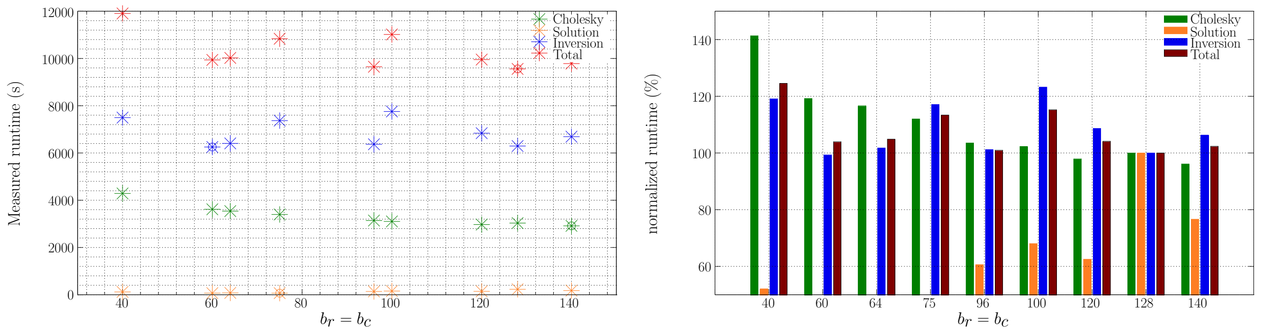
7.4.2.1 Choice of the Block-cyclic Distribution Parameters

Within the runtime test, the compute core grid is fixed to 64×64 and the parameters of the block-cyclic distribution b_r and b_c are varied. Using the different block-sizes, the runtime for the Cholesky decomposition, forward and backward substitution and the inversion of the already Cholesky reduced matrix are measured. The measured wall clock time for the whole operation and the three partial steps are shown in Fig. 7.6.

The main conclusion is that using SCALAPACK and the implemented interface, it is possible to derive the solution in less than one hour and the inverse in less than two hours, having in mind the dense large system with 519 837 unknowns. Thus, in less than three hours the whole task is solvable without computationally motivated approximations.

Whereas the relative runtime of the partial steps behaves as expected (1/3 Cholesky, 2/3 inversion and solution insignificant), the inversion and the Cholesky reduction show different behaviors when varying b_r and b_c. The runtime for the Cholesky decomposition slightly decreases with increasing $b_r = b_c$. Additional runtime of 47 % to 16 % is needed for $(b_r = b_c) < 75$. Above 75, the variations are in the range of $1 - 7$ %, which might not be significant, as the numbers are derived from a single experiment only.

In contrast to the Cholesky decomposition the runtime for the inversion does not provide obvious systematic characteristics. Instead some minima can be seen, for which the runtime is very similar (using $b_r = b_c = 60$, 64, 96 and 128). It looks like a more or less constant but noisy behavior. With the runtime measurements performed (the numbers were derived from a single run), clear conclusions can not be drawn. The suggested default value of 64 is only 2.5 % slower than the



(a) Absolute runtime measured. The minimal values are additionally marked with circles.

(b) Runtime normalized to minimal runtime of the individual operation.

Figure 7.6: Results for the wall-clock measurements of the solution and inversion of the NEQs ($519\,837 \times 519\,837$ with a RHS of dimension $519\,837 \times 200$). Analyzed are the parameters of the block-cyclic distribution, using a fixed 64×64 compute core grid. The runtime for the total operation as well as for the partial operations (Cholesky decomposition, solution via forward- and backward-substitution as well as the computation of the inverse from the already Cholesky reduced NEQs) are shown.

minimal value observed. As the total runtime is dominated by the inversion, the general conclusions for the optimal block-size for all operations are the same. As the dimension of the local matrices is above 8000×8000, the number of full sub-blocks of dimension $b_r \times b_c$ is still large for all values studied for b_r and b_c. It varies from 200 (for $b_r = b_c = 40$) to 60 (for $b_r = b_c = 140$). As the load leveling does not significantly change, the performance is similar for all studies block-sizes. An exception is the smallest block-size which seems to be too small for an efficient cache-use and thus the used BLAS-L3 routines within the SCALAPACK implementation of `pdpotrf` and `pdpotri`.

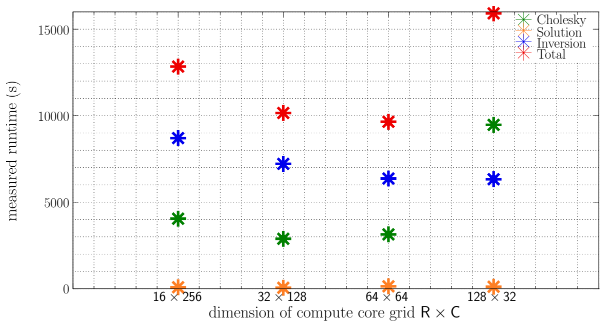
Due to the small number of columns of the RHS (200 only) and the general small runtime (1 min to 3 min) the solution step is very sensitive for the dimension of the sub-blocks. For the larger block sizes the performance becomes poor, as only two columns of the compute core grid store the RHS. As the runtime is insignificant compared to the other operations, it does not make sense to optimize with respect to its behavior and thus it is not discussed.

The minimal runtime of 160 min is found for $b_r = b_c = 96$ and $b_r = b_c = 128$. It only differs in a few seconds.

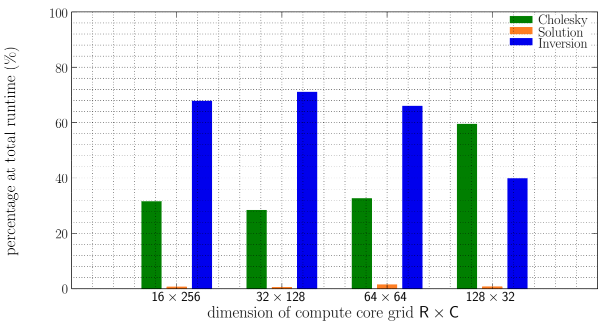
7.4.2.2 Choice of the Compute Core Grid

Due to computation time limits, the number of cores and the scaling is not studied as the steps involved in the final solver are performed purely completely SCALAPACK routines. In different setups is was found that the performance with $\mathsf{N} = 4096$ is best. Instead a small test computation is performed to demonstrate the dependence of the runtime from the used grid dimension. For that purpose the Cholesky decomposition, the solution and the inversion are repeated, using $\mathsf{N} = 4096$ cores and the distribution parameters $b_r = b_c = 96$ (cf. Sect. 7.4.2.1). The shape of the grid was varied from 16×256 to 128×32. The measured runtime and the percentage of the individual steps at the total runtime are shown in Fig. 7.7.

The SCALAPACK implementation of the inversion and the Cholesky factorization show an inverse behavior. Whereas the runtime needed for the inversion is decreasing, using more rows (R) in the compute core grid, the runtime of the Cholesky factorization increases. The consequence is, that the best performance for the whole solution step is found for the used quadratic compute core



(a) Absolute runtime measured using different shapes of the compute core grid.



(b) Percentage of the three operations of the total runtime.

Figure 7.7: Measured performance of the solver with different shapes of the compute core grid. Results are shown for all operations involved (red) and the three individual operations (Cholesky decomposition in green, solution via forward- and backward-substitution in orange and the computation of the inverse from the already Cholesky reduced NEQs in blue).

grid (64×64). For the Cholesky decomposition many operations can be performed on a (block-) row of the matrix in parallel (cf. (3.9)). During the computation of the inverse, operations can be performed in parallel along a (block-) column of the already factorized matrix. For the grid with $\mathsf{R} > \mathsf{C}$, the inversion is faster than the Cholesky factorization.

7.5 Application to Real Data

Only parts of the software were used to analyze real data yet. The software was used for the combination of real data already preprocessed as NEQs only and thus $O = 0$. For instance it was used in the preparation of the satellite-only models of the GOCO (Gravity Observation Combination Consortium, Pail et al., 2014) series (GOCO01S, GOCO02S and GOCO03S Pail et al., 2010b, Goiginger et al., 2011, Mayer-Gürr et al., 2012). In addition it was used for the VCE within the estimation local MDT models combining gravity field NEQs with NEQs derived from altimetry including a MDT representation as finite elements (for details see Becker et al., 2014a, 2013, 2014b). A further development for a joint global estimation of the gravity field and the MDT is planed, extending the method developed in Becker (2012) to the global scope.

It is impossible to quantify the gain of the proposed rigorous solution compared to the currently used alternatives. But for instance with the currently used approaches, it will never be possible to introduce altimetric measurements in their original form — as along track sea surface height measurements. They can only be analyzed as a gridded product. For such gridded products neither error descriptions nor correlations exist. Thus, starting from the original observations will always produce better solutions. The study and solution concept presented in this chapter should contribute (to the computational aspects) on the way towards, for example, an analysis of physically motivated observation equations for altimetry, which allows to jointly derive estimates for the gravity field, the dynamic ocean topography and why not e.g. additionally a deterministic model for the ocean tides.

8. Application: Ultra High Degree Gravity Field Determination Using an Iterative Solver

The motivation for the iterative solver presented in this chapter remains the same as for the direct solver from Chap. 7. The advantage of the iterative solver is that, either the parameter space can be increased or the solution for the same scenario is found but with reduced runtime (or on smaller compute core grids available). As the covariance matrix of the parameters is not available using the iterative solver, both solvers can be used in conjunction. For instance, as less computational resources are required, the iterative solver can be used for parameter tuning (iterative data adaptive estimation of VCs or refinement of the stochastic model) and the direct solver (cf. Chap. 7) for the final solution, with the covariance matrix as an additional result.

This chapter is an extension of the study already published in Brockmann et al. (2014c), so some parts of the chapter are taken from that study.

8.1 Problem Description

With an increasing number of unknowns, the estimation of the parameters via the assembly and solution of the combined normal equations becomes unreasonable. On the one hand the computing time increases and on the other hand the resulting NEQs or covariance matrices of several TB in size are not easy to handle and get hardly usable in further applications. Nevertheless it makes sense to implement a rigorous least squares solver to avoid computationally motivated approximations and simplifications as done so far within combined global gravity field determination (Förste et al., 2012, Pavlis et al., 2012, Reguzzoni and Sansò, 2012) for models with a very high resolution resolved to very high spherical harmonic degrees (cf. Sect. 5.1).

Instead of the direct solution via the assembly of full NEQs, iterative solvers can be used to determine (after convergence) a rigorous least squares solution or to determine tuning parameters for the direct solver. As a wide range of iterative solvers exists, only the references to the overview chapters in Golub and van Loan (1996, Chap. 10), Dongarra et al. (1990b, Chap. 7) and Schuh (1996, Chap. 8) are provided. Eijkhout (1998) provides an overview of freely available serial and parallel libraries. These solvers are often used, even if the parameter space is not huge, to derive a fast solution (in the context of gravity field determination see e.g. Ditmar et al., 2003a, Klees et al., 2003, Pail and Plank, 2003, Xie, 2005, Baur et al., 2008, Baur, 2009, Brockmann et al., 2010, Guangbin et al., 2012, Farahani et al., 2013). As the mentioned studies use the iterative solver for satellite only models (CHAMP, GRACE or GOCE), the number of parameters is limited (15 000-90 000). They are often applied as serial implementations to solve the systems of equations with little computing resources, e.g. on standard PCs or, as their parallel implementation on shared memory computers is simple and straightforward, parallelized for shared memory systems. In contrast to the application in global gravity field determination, iterative solvers are often used for large but sparse systems of equations, e.g. solving discrete differential equations with a finite element approximation (see e.g. Löf, 2004). This chapter presents an implementation of a massive parallel iterative least squares solver, which is capable to compute the rigorous solution for gravity field models with hundreds of thousands to millions of unknown parameters. The basic computational advantage compared to the direct solution in Chap. 7 is that the time consuming computation of the product $\mathbf{A}_i^T\mathbf{P}_i\mathbf{A}_i$ is substituted by two iteratively repeated matrix-vector like products involving the design matrix (matrix-matrix products when VCE is involved). Setting up a high quality preconditioner $\mathbf{N}_\oplus$ — i.e. a sparse approximation of $\mathbf{N}$ — the number of required iterations is small.

The goal of this chapter is not to provide an alternative solution technique, but to present details implementing a more or less well known iterative solver for dense systems in a massive parallel HPC environment using the derived basic framework. A massive parallel implementation in a HPC environment of the iterative preconditioned conjugate gradient (PCG) solver following the (predefined) fundamentals is derived:

- The implementation should allow to determine the combined solution of an arbitrary number of preprocessed normal equations.
- The implementation should be able to use an arbitrary number of observation groups of different observations types. Inclusion of covariance information in terms of covariance functions should be possible.
- For every observation group involved (NEQ and OEQ) a weight in terms of inverse VCs should be estimable.
- Group specific parameters should be possible.
- Different preconditioner types should be possible.
- The solver should be capable to operate on an arbitrary number of cores from ten to tens of thousands.

In contrast to the GOCE application in Chap. 6 the computational challenge is not the number of highly correlated observations. Whereas the direct solver in Chap. 7 was implemented to set up the full NEQs for a high dimensional parameter space and thus to derive the full covariance matrix, the challenge here is to derive a non-approximative least squares solution for even higher dimensional parameter spaces. For the combination of NEQs and OEQs the *Preconditioned Conjugate Gradient Multiple Adjustment* (PCGMA) algorithm was proposed by Schuh (1995, 1996) as a connection of the conjugate gradient algorithm proposed by Hestenes and Stiefel (1952) and the extension for adjustment problems by Schwarz (1970). It is extended by VCE following the line of thought of Kusche (2003) and Alkhatib (2007), i.e. the application and integration of the MC based trace estimation into the iterative solver. In this chapter the basic (serial) PCGMA algorithm is introduced, a massive parallel implementation in a HPC environment is developed and a simulation scenario is used to demonstrate the possibilities and performance of the solver. The performance analysis of a first full scale closed loop simulation is used to identify weaknesses of this first version of the implementation. Several setups of the block-cyclic distribution and the compute core grid are tested to identify usable setups for the iterative solver and to identify parts in the implementation where potential for optimization remains. Due to the positive experiences with PCGMA for GOCE data processing and existing studies (e.g. Schuh, 1996, Baur et al., 2008), the decision is fixed to PCGMA and alternative iterative solvers are not discussed here.

8.2 Basic Algorithm Description of PCGMA including VCE

8.2.1 Basic PCGMA Algorithm

The basic PCGMA algorithm is summarized in Alg. 8.1, as proposed by Schuh (1996) for the data combination problem cf. (4.6b) and extended by Boxhammer (2006). The expensive computations for the assembly of $\mathbf{N}$, especially $\mathbf{N}_o = \mathbf{A}_o^T \mathbf{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}^{-1} \mathbf{A}_o$, are avoided. Instead only a few matrix-vector multiplications have to be performed in each iteration (especially for the computation of $\mathbf{h}^{(\nu)}$, cf. l. 11 in Alg. 8.1). The result of the algorithm is a least squares estimate of the unknown parameters $\tilde{\mathbf{x}}$ after ν iterations. In contrast to direct solution method (cf. Chap. 7), a covariance matrix of the parameters cannot be determined in a direct efficient way. Monte-Carlo Simulations might be used to obtain approximations for the covariance matrix (e.g. Gundlich et al., 2003, Alkhatib and Schuh, 2007, Alkhatib, 2007, Migliaccio et al., 2008).

Because the individual steps of the algorithms are well known and are documented in many textbooks and articles (e.g. Golub and van Loan, 1996, Shewchuk, 1994, Björck, 1998, Schuh, 1996, Press et al., 2007, Chap. 3) the algorithm and the individual steps are not reviewed. Instead they are collected in an algorithmic form in Alg. 8.1.

Algorithm 8.1: PCGMA algorithm following Schuh (1996).

Data:

$\mathbf{N}_\oplus$	sparse preconditioning matrix	$\nu_{\max}$	maximal number of PCGMA iterations
$\mathbf{A}_o$	design matrix of group $o \in \{0...O-1\}$	$\boldsymbol{\ell}_o$	vector of observations $o \in \{0...O-1\}$
$\mathbf{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}$	cofactor matrix of group $o \in \{0...O-1\}$	w_o	weight for OEQ group $o \in \{0...O-1\}$
$\mathbf{N}_n$	normal matrix for group $n \in \{0...N-1\}$	$\mathbf{n}_n$	right hand side of NEQ for $n \in \{0...N-1\}$
w_n	weight for NEQ group $n \in \{0...N-1\}$	$\mathbf{x}^{(0)}$	initial values for parameters

```
$\mathbf{r}^{(0)} = \sum_n^{N-1} w_n \left(\mathbf{N}_n\mathbf{x}^{(0)} - \mathbf{n}_n\right) + \sum_o^{O-1} w_o\mathbf{A}_o^T\mathbf{Q}_{\mathbf{l}_o\mathbf{l}_o}^{-1}\left(\mathbf{A}_o\mathbf{x}^{(0)} - \boldsymbol{\ell}_o\right)$ // compute initial joint residuals
$\boldsymbol{\rho}^{(0)} = \text{solve}\left(\mathbf{N}_\oplus, \mathbf{r}^{(0)}\right)$ // apply preconditioner to residuals
$\boldsymbol{\pi}^{(0)} = -\boldsymbol{\rho}^{(0)}$ // initial search direction
// PCGMA - iterations
for $\nu = 0$ to $\nu_{\max} - 1$ do
    if $\nu > 0$ then
        // update step for search direction
        $e_\nu = \frac{\mathbf{r}^{(\nu)T}\boldsymbol{\rho}^{(\nu)}}{\mathbf{r}^{(\nu-1)T}\boldsymbol{\rho}^{(\nu-1)}}$
        $\boldsymbol{\pi}^{(\nu)} = -\boldsymbol{\rho}^{(\nu)} + e_\nu\boldsymbol{\pi}^{(\nu-1)}$
    end
    $\mathbf{h}^{(\nu)} = \sum_n^{N-1} w_n\mathbf{N}_n\boldsymbol{\pi}^{(\nu)} + \sum_o^{O-1} w_o\mathbf{A}_o^T\mathbf{Q}_{\mathbf{l}_o\mathbf{l}_o}^{-1}\mathbf{A}_o\boldsymbol{\pi}^{(\nu)}$
    // update step for parameters
    $q^{(\nu)} = \frac{\mathbf{r}^{(\nu)T}\boldsymbol{\rho}^{(\nu)}}{\boldsymbol{\pi}^{(\nu)T}\mathbf{h}^{(\nu)}}$
    $\mathbf{x}^{(\nu+1)} = \mathbf{x}^{(\nu)} + q^{(\nu)}\boldsymbol{\pi}^{(\nu)}$
    // update step for residuals
    $\mathbf{r}^{(\nu+1)} = \mathbf{r}^{(\nu)} + q^{(\nu)}\mathbf{h}^{(\nu)}$
    // apply preconditioner to residuals
    $\boldsymbol{\rho}^{(\nu+1)} = \text{solve}\left(\mathbf{N}_\oplus, \mathbf{r}^{(\nu+1)}\right)$
end
return $\mathbf{x}^{(\nu_{\max})}$
```

8.2.2 PCGMA Algorithm including VCE

The basic PCGMA algorithm in Alg. 8.1 is extended by VCE for an arbitrary number of variance components. Because $\mathbf{N}$ is not computed within iterative solvers, but needed in standard VCE cf. (4.9), the Monte Carlo based method is used as introduced in (4.14) and (4.17) (cf. Koch and Kusche, 2002, Alkhatib, 2007, Brockmann and Schuh, 2010).

Whereas Ω_i can be directly computed as for the direct solver, the computation of the partial redundancies, especially the trace term in (4.14) and (4.17), has to be adapted as introduced in Sect. 4.2.3. Including these steps, i.e. setting up the additional RHS and solving for the additional parameters, into Alg. 8.1 yields the PCGMA algorithm with VCE as shown in Alg. 8.2.

The intention of Alg. 8.2 is to give a quite simple overview about the implemented algorithm. The mathematical steps are not discussed in detail. Thus so far a major point is not addressed and not taken into account in the algorithm. Not all NEQs are assembled for the same set of parameters. Instead individual NEQs may just be assembled for a (small) subset of the parameters to be estimated. The NEQs differ in their dimension and consequently in their parameter ordering. This point is addressed in the following section, in which the implementation in the HPC environment is discussed and a concept for a massive parallel implementation is proposed.

Algorithm 8.2: PCGMA algorithm extended for VCE.

Data:

$\mathbf{N}_\oplus$	sparse preconditioning matrix	$\nu_{\max}$	maximal number of PCGMA iterations
$\mathbf{A}_o$	design matrix of group $o \in \{0...O-1\}$	$\boldsymbol{\ell}_o$	vector of observations $o \in \{0...O-1\}$
$\mathbf{Q}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}$	cofactor matrix of group $o \in \{0...O-1\}$	w_o	weight for OEQ group $o \in \{0...O-1\}$
$\mathbf{N}_n$	normal matrix for group $n \in \{0...N-1\}$	$\mathbf{n}_n$	right hand side of NEQ for $n \in \{0...N-1\}$
w_n	weight for NEQ group $n \in \{0...N-1\}$	$\mathbf{x}^{(0,0)}$	initial values for parameters
$\eta_{\max}$	maximal number of VCE iterations	k	number of Monte Carlo samples

```
for $\eta = 0 < \eta_{\max}$ do
    $\mathbf{X}^{(\eta,0)} = \begin{bmatrix}\mathbf{X}^{(\eta-1,0)} & \mathbf{0} & \cdots & \mathbf{0}\end{bmatrix}$ // compose start solution, 0 for the k*(N+O) MC parameters
    $\mathbf{R}^{(0)} = \mathbf{0}$ // initialize combined residuals with 0, dimension: # parameter $\times (N+O)k+1$
    // add contribution by NEQ group n
    for $n = 0 < N$ do
        // Generate Monte Carlo Samples for all groups n, dimension: #parameter × k
        $\mathbf{P}_n = \pm 1 \sim \mathcal{U}(-1,1)$
        // sample transformation
        $\mathbf{R}_n = \text{chol}(\mathbf{N}_n)$ // operation chol applies Cholesky factorization
        $\bar{\mathbf{P}}_n = \mathbf{R}_n^T \mathbf{P}_n$
        // right hand sides for group n, first column to insert $\bar{\mathbf{P}}_n$: $1 + (n-1)k$
        $\mathbf{B}_n = \begin{bmatrix}\mathbf{n}_n & \mathbf{0} & \cdots & \mathbf{0} & \bar{\mathbf{P}}_n & \mathbf{0} & \cdots & \mathbf{0}\end{bmatrix}$
        $\mathbf{R}^{(0)} += w_n^{(\eta-1)} \left(\mathbf{N}_n \mathbf{X}^{(\eta,0)} - \mathbf{B}_n\right)$
    end
    // add contribution by OEQ group o
    for $o = 0 < O$ do
        // Generate Monte Carlo Samples for all groups o, dimension: #observations × k
        $\mathbf{P}_o = \pm 1 \sim \mathcal{U}(-1,1)$
        $\mathbf{G_o} = \text{chol}\left(\mathbf{Q}^{-1}_{\boldsymbol{\ell}_o\boldsymbol{\ell}_o}\right)$
        // right hand sides for group o, first column to insert $\mathbf{P}_o$: $1 + Nk + (o-1)k$
        $\mathbf{L}_o = \begin{bmatrix}\mathbf{G_o}\boldsymbol{\ell}_o & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{P}_o & \mathbf{0} & \cdots & \mathbf{0}\end{bmatrix}$
        $\mathbf{R}^{(0)} += w_o^{(\eta-1)} \mathbf{A}_o^T {\mathbf{G_o}}^T \left(\mathbf{G_o}\mathbf{A}_o \mathbf{X}^{(\eta,0)} - \mathbf{L}_o\right)$
        $\bar{\mathbf{P}}_o = -\mathbf{R}^{(0)}(1 + Nk + (o-1)k : 1 + Nk + (o-1)k + K, :)$
    end
    $\boldsymbol{\Gamma}^{(0)} = \text{solve}\left(\mathbf{N}_\oplus, \mathbf{R}^{(0)}\right)$ // apply preconditioner to residuals
    $\boldsymbol{\Pi}^{(0)} = -\boldsymbol{\Gamma}^{(0)}$ // initial search direction
    // PCGMA - iterations
    for $\nu = 0$ to $\nu_{\max} - 1$ do
        if $\nu > 0$ then
            // update step for search direction
            $\mathbf{E}_\nu = \frac{\mathbf{R}^{(\nu)T}\boldsymbol{\Gamma}^{(\nu)}}{\mathbf{R}^{(\nu-1)T}\boldsymbol{\Gamma}^{(\nu-1)}}$          element-wise division!
            $\boldsymbol{\Pi}^{(\nu)} = -\boldsymbol{\Gamma}^{(\nu)} + \mathbf{E}_\nu \circ \boldsymbol{\Pi}^{(\nu-1)}$          $\circ :=$ scaling of column k with $\mathbf{E}_\nu(k,k)$!
        end
        $\mathbf{H}^{(\nu)} = \sum_n^{N-1} w_n \mathbf{N}_n \boldsymbol{\Pi}^{(\nu)} + \sum_o^{O-1} w_o \mathbf{A}_o^T {\mathbf{G_o}}^T \mathbf{G_o} \mathbf{A}_o \boldsymbol{\Pi}^{(\nu)}$
        // update step for parameters
        $\mathbf{Q}^{(\nu)} = \frac{\mathbf{R}^{(\nu)T}\boldsymbol{\Gamma}^{(\nu)}}{\boldsymbol{\Pi}^{(\nu)T}\mathbf{H}^{(\nu)}}$          element-wise division!
        $\mathbf{X}^{(\eta,\nu+1)} = \mathbf{X}^{(\eta,\nu)} + \mathbf{Q}^{(\nu)} \circ \boldsymbol{\Pi}^{(\nu)}$          $\circ :=$ scaling of column k with $\mathbf{Q}_\nu(k,k)$!
        // update step for residuals
        $\mathbf{R}^{(\nu+1)} = \mathbf{R}^{(\nu)} + \mathbf{Q}^{(\nu)} \circ \mathbf{H}^{(\nu)}$          $\circ :=$ scaling of column k with $\mathbf{Q}_\nu(k,k)$!
        // apply preconditioner to residuals
        $\boldsymbol{\Gamma}^{(\nu+1)} = \text{solve}\left(\mathbf{N}_\oplus, \mathbf{R}^{(\nu+1)}\right)$
    end
    // update weights of OEQ groups i (o and n) using eq. (4.7) and (4.8a), (4.22)
    $w_i^{(\eta+1)} = \text{computeVCE}(\mathbf{X}^{(\eta,\nu_{\max})},\ \bar{\mathbf{P}}_i)$
    $\eta = \eta + 1$
end
return $\mathbf{X}^{(\eta_{\max},\nu_{\max})}$, $w_o^{(\eta_{\max})}$, $w_n^{(\eta_{\max})}$
```

8.3 Computational Aspects and Parallel Implementation

The basic algorithm and a simple form of its parallel implementation is well known. Within this section, the massive parallel implementation of the PCGMA algorithm tailored for the combination of complementary data on a HPC cluster is summarized. Standard concepts of scientific HPC as introduced in Sect. 2 and 3 are used to implement the solver presented in Alg. 8.2. Using the standard concepts, a portable implementation, which is able to run on nearly every HPC cluster is developed. In addition the design is (nearly) independent of the number of cores. It is capable to operate on only a few to tens of thousands of computing cores. The goal is a flexible implementation of the algorithm, allowing for an easy extension of the algorithm e.g. for new observation types or additional parameters to be estimated. The whole Alg. 8.2 is implemented based on the concept of block-cyclic distributed matrices (cf. Chap. 3), such that huge matrices can be used and all steps of the solver are operated in parallel.

The computational most demanding steps in Algorithm 8.2 are (i) the design of the preconditioner, (ii) the preconditioning step and (iii) the computation of the update vectors $\mathbf{H}^{(\nu)}$ (as well as the initial residuals $\mathbf{R}^{(0)}$). This steps will be discussed in detail in the following.

8.3.1 Setup of a Preconditioning Matrix

As the expected convergence rate strongly depends on the condition of $\mathbf{N}$ (e.g. Golub and van Loan, 1996, p. 521) and gravimetric problems tend to be ill-conditioned (distribution of the data, for satellite data see e.g. Schwintzer et al., 1997, Ilk et al., 2002), one essential step for the convergence of the PCGMA algorithm is the design of a preconditioner $\mathbf{N}_{\oplus}$. This preconditioner should have the following properties: i) it reflects the main characteristics of $\mathbf{N}$, ii) it is sparse and easily computable and iii) a factorization (e.g. Cholesky factorization) is sparse as well. Whereas very special preconditioners might be designed for very individual problems and setups (e.g. Boxhammer and Schuh, 2006), also very simple designs are able to considerably improve the convergence rate. As shown by Boxhammer (2006, p. 68), a simple block diagonal preconditioner can be a very efficient approximation and significantly accelerates the PCGMA convergence for spherical harmonic analysis.

8.3.1.1 Shape of the Preconditioning Matrix

Finding and implementing an efficient preconditioner for the aspired huge dimensions is not a simple task. Within global gravity field determination the first choice is always a block diagonal preconditioner. Although the prerequisites for a block diagonal normal equation matrix $\mathbf{N}$ (e.g. Colombo, 1981, Reguzzoni and Sansò, 2012) are not fulfilled, it can be expected, that $\mathbf{N}$ is at least block diagonal dominant, if the spherical harmonic coefficients are arranged by orders. Thus, the easiest way is to implement a block diagonal preconditioner, where only correlations within coefficients of the same order m are modeled. The following block diagonal preconditioners were implemented within this study as a first choice of easy to use preconditioners:

A: four blocks per order: Correlations are assumed to occur only between coefficients of the same order, of the same type and of the same parity of the degree (i.e. only between sine coefficients of odd degree, only between sine coefficients of even degree, only between cosine coefficients of odd degree, only between cosine coefficients of even degree). These matrix would result due to orthogonalities of the spherical harmonics if the data points were equidistant along the parallels with constant accuracy per parallel. In addition equatorial symmetry is expected (for the data accuracy and distribution).

Table 8.1: Memory requirements of block diagonal preconditioners for the three block diagonal approximations A, B and C. The size is provided in GB for different maximal spherical harmonic resolutions $l_{\max}$. It is always assumed that a block is stored as a full matrix.

$l_{\max}$	200	360	500	720	1000	1500	2000
setup A	0.02	0.12	0.31	0.93	2.49	8.39	19.88
setup B	0.04	0.23	0.62	1.86	4.97	16.78	39.77
setup C	0.08	0.47	1.25	3.72	9.95	33.56	79.53

B: two blocks per order: Correlations are assumed to occur between coefficients of the same order and of the same type (i.e. only between sine coefficients, only between cosine coefficients). These matrix would result due to orthogonalities of the spherical harmonics if the data points were equidistant along the parallels with constant accuracy.

C: one block per order: Correlations are assumed to occur between all coefficients of the same order.

For the high degrees which should be processed with the solver, even the block diagonal preconditioner requires a significant amount of memory. To get a feeling for the required memory, Tab. 8.1 summarizes some numbers for memory requirements of several preconditioners of different resolution for the block diagonal models A, B and C (note that a full storage of the symmetric matrix blocks is assumed).

As these memory requirements are several Gigabytes (GB), the preconditioners, especially for the high degree setups, can not be stored serial in the memory of a single compute core. Instead, the preconditioner has to be stored distributed as well. Storing the preconditioner as an array of block-cyclic distributed matrices facilitates its application to block-cyclic distributed matrices (and vectors). Thus, each entry b of the array corresponds to the block $b \in \{0 \dots B-1\}$ of the block diagonal preconditioning matrix

$$\mathbf{N}_{\oplus} = \begin{bmatrix} \mathbf{N}_{\oplus}^{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{N}_{\oplus}^{1} & \mathbf{0} & \mathbf{0} \\ \vdots & \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{N}_{\oplus}^{B-1} \end{bmatrix} = \sum_{o=0}^{O-1} w_o \mathbf{N}_{\oplus,o} + \sum_{n=0}^{N-1} w_n \mathbf{N}_{\oplus,n}. \tag{8.1}$$

The preconditioner models A, B and C can be mapped to this storage scheme. The number of blocks B and the dimension of the blocks $\mathbf{N}_{\oplus}^{0}$ differ for the three models. As for the combined NEQs in Chap. 7, the preconditioner for the joint solution is a weighted combination of the observation groups provided as OEQs and those provided as NEQs.

8.3.1.2 Parallel Computation of the Preconditioner

Preconditioner for OEQ Groups The computation of the preconditioning matrix can be parallelized very well, as the blocks $\mathbf{N}_{\oplus,b}$ can be independently computed on different compute cores. For instance one process can be used as a master process which is responsible for distributing the work to the other compute cores, i.e. the block b which should be processed by that core. The client can compute the block b for a dataset o serial via

$$\mathbf{N}_{\oplus,o}^{b} = \mathbf{A}_o^{bT} \mathbf{Q}_{\boldsymbol{\ell}_o \boldsymbol{\ell}_o}^{-1} \mathbf{A}_o^{b} \tag{8.2}$$

where $\mathbf{A}_o^b$ is the serial design matrix, containing only the columns which corresponds to the parameters included in the symmetric matrix $\mathbf{N}_{\oplus,b}$. The computation is performed for all observations of the group on a single core (in portions of size b_{obs} if necessary) using BLAS-L3 `dysrk`. The parallelization is achieved via a distribution of the blocks. As the parameters within $\mathbf{N}_{\oplus}^b$ are always of the same order m, the recursion formulas for the Legendre functions can be evaluated with only a few redundant computations on the different processes. As the preconditioner is needed in different VCE iterations, the preconditioner is assumed to be stored on disk individually for every observation group o. It is setup in the order-wise numbering scheme $\mathbf{p}_{\oplus}$. Within each order m, the cosine coefficients of even degree, the cosine coefficients of odd degree, the sine coefficients of even degree and the sine coefficients of odd degree are arranged as neighboring blocks, such that all preconditioner models (cf. Sect. 8.3.1.1) are covered by the same numbering scheme $\mathbf{p}_{\oplus}$ (cf. Schuh, 1996, Sect. 2.2.2). The blocks (depending on the the model cf. Sect. 8.3.1.1) are stored as separate full matrices. The combined preconditioner (8.1) is used with different weights of the individual groups within the VCE loop (weights of iteration $\eta - 1$).

Preconditioner for NEQ Groups The preconditioner part for the groups of NEQs $\mathbf{N}_{\oplus,n}^b$ is created from a subset of the provided normal equations matrices

$$\mathbf{N}_{\oplus,n}^b = \text{select}\left(\mathbf{N}_n\right) \tag{8.3}$$

which are available in an arbitrary numbering scheme $\mathbf{p}_n$. The "select" operator should symbolize the steps:

- Create a numbering scheme which follows the parameter order as defined by the order-wise numbering scheme of the combined preconditioner ($\mathbf{p}_{\oplus}$), but without the high degree coefficients, which are not part of the parameter space of $\mathbf{N}_n$ ($:= \mathbf{p}_{\oplus,n}$).
- Reorder the matrix $\mathbf{N}_n$ to the numbering scheme $\mathbf{p}_{\oplus,n}$.
- Go through all blocks b, initialize $i = 0$:
  - Create a numbering scheme for block b, containing all parameters of that block i.e. $\mathbf{p}_{\oplus}^b$.
  - Create the numbering scheme $\mathbf{p}_{\oplus,n}^b$ containing only coefficients of block b which are contained in $\mathbf{N}_n$.
  - Extract the sub-matrix $\mathbf{N}_{\oplus,n}^b := \mathbf{N}_n(i : i + \mathbf{p}_{\oplus,n}^b.\texttt{size()}, i : i + \mathbf{p}_{\oplus,n}^b.\texttt{size()})$ from $\mathbf{N}_n$, which then follows the numbering scheme in $\mathbf{p}_{\oplus,n}^b$.
  - Extend $\mathbf{N}_{\oplus,n}^b$ with zeros to arrive in the dimension of $\mathbf{N}_{\oplus}^b$.
  - Reorder the $\mathbf{N}_{\oplus,n}^b$ from $\mathbf{p}_{\oplus,n}^b$ to $\mathbf{p}_{\oplus}^b$.
  - Perform the update $\mathbf{N}_{\oplus}^b + = w_n \mathbf{N}_{\oplus,n}^b$.
  - Update $i + = \mathbf{p}_{\oplus,n}^b.\texttt{size()}$.

The basic challenge is the implementation of the reordering algorithm (cf. Sect. 4.3), to perform row and column interchanges (reordering of $\mathbf{N}_n$) on the large distributed matrix $\mathbf{N}_n$ to bring arbitrary numbering schemes to an order-wise numbering. With the reordering algorithms implemented, the update of the preconditioner with the NEQ groups is straightforward and is limited to some index and sub-matrix computations.

Combination of the Preconditioner Within each VCE iteration η in Alg. 8.2, the preconditioner is combined as a weighted sum according to (8.1). For the observation groups o the preconditioners $\mathbf{N}_{\oplus,o}$ are recalled directly from disk into an array of block-cyclic distributed matrices within a parallel file reading operation. The matrices $\mathbf{N}_{\oplus,n}^b$ are not stored additionally, instead the "select" operation is directly implemented within the step of weighted addition.

Algorithm 8.3: Application of a block-diagonal block-cyclic distributed preconditioner to a block-cyclic distributed matrix (in place).

Data:

`DistributedMatrix` $\mathbf{R}$	Matrix to be preconditioned
`DistributedBlockdiagonalMatrix` $\mathbf{N}_\oplus$	blockdiagonal preconditioner to be applied

```
// Loop over all blocks b of the preconditioner
for $b = 0; b < \mathbf{N}_\oplus$.blocks(); $b++$ do
    // Perform Cholesky factorization of block b
    $\mathbf{N}_\oplus(b)$.chol()
    int $r_s = \mathbf{N}_\oplus(b)$.firstRow()
    int $r_e = \mathbf{N}_\oplus(b)$.lastRow()
    // compute forward and backward substitution, block b
    $\mathbf{R}$.isSolutionOfTriag($\mathbf{N}_\oplus(b)$, 'L', 'T', $r_s$, $r_e$, 0, $\mathbf{R}$.globalRows())
    $\mathbf{R}$.isSolutionOfTriag($\mathbf{N}_\oplus(b)$, 'L', 'N', $r_s$, $r_e$, 0, $\mathbf{R}$.globalRows())
end
return $\mathbf{R}$// preconditioned matrix
```

8.3.1.3 Application of the Preconditioner to a block-cyclic distributed Matrix

A preconditioner of the form described above, can be easily applied to a matrix (or vector) which is stored as a block-cyclic distributed matrix. The symbolic preconditioning operation as used in Alg. 8.1 (l. 2 and 17) and Alg. 8.2 (l. 25 and 41) for the residual matrix $\mathbf{R}$

$$\mathbf{\Gamma} = \mathbf{N}_\oplus^{-1}\mathbf{R} = \text{solve}\left(\mathbf{N}_\oplus,\ \mathbf{R}\right) \tag{8.4}$$

is numerically solved via a Cholesky factorization and followed by forward and backward substitution (cf. Schuh, 1996, Sect. 8.3). The result is the preconditioned residual matrix $\mathbf{\Gamma}$. The symbolic operation `solve` comprises the following numerical steps which are performed for every block b. The factorization

$$\mathbf{N}_\oplus^b = \mathbf{G}_\oplus^{bT}\mathbf{G}_\oplus^b, \quad \mathbf{G}_\oplus^b = \text{chol}\left(\mathbf{N}_\oplus^b\right). \tag{8.5}$$

and the solution of the system via forward and backward substitution

$$\mathbf{N}_\oplus^b\mathbf{\Gamma}^b = \mathbf{R}^b \tag{8.6a}$$

$$\mathbf{G}_\oplus^{bT}\mathbf{G}_\oplus^b\mathbf{\Gamma}^b = \mathbf{R}, \qquad \mathbf{B}^b := \mathbf{G}_\oplus^b\mathbf{\Gamma}^b \tag{8.6b}$$

$$\mathbf{B}^b = \text{triangularSolve}\left(\mathbf{G}_\oplus^{bT},\ \mathbf{R}^b\right) \tag{8.6c}$$

$$\mathbf{\Gamma}^b = \text{triangularSolve}\left(\mathbf{G}_\oplus^b\mathbf{B}^b\right) \tag{8.6d}$$

can both be performed independently for every block b. The matrices $\mathbf{R}^b$ and $\mathbf{\Gamma}^b$ are sub-matrices of $\mathbf{R}$ and $\mathbf{\Gamma}$ containing all columns, but only those rows the block b refers to in the matrix $\mathbf{N}_\oplus$. As the involved matrix $\mathbf{G}_\oplus^b$ is a triangular (block-cyclic distributed) matrix, the solve operations are computed using the SCALAPACK routine `pdtrsvm` twice. The sub-matrices $\mathbf{R}^b$ and $\mathbf{B}^b$ can be directly passed to the SCALAPACK routine using the row indices in the SCALAPACK function interface. The same is done for the "result" matrix $\mathbf{\Gamma}$. The whole preconditioning step as an in place operation is summarized in Alg. 8.3.

8.3.2 Additional Right Hand Sides for VCE

For the integration of VCE additional RHS sides are introduced according to Sect. 4.2.3 and Alg. 8.2 (l. 6–12 for NEQs and l. 18–21 for OEQs). The distributed vector of observations $\boldsymbol{\ell}$ is extended for every group o with K additional columns, including the raw drawn samples (cf. (4.10)). The transformation of the samples is explicitly done in the computations of the initial residuals (cf. Alg. 8.2, l. 13), as the initial MC parameters are set to zero. The transformed samples, which are needed later for VC computation in (4.22), can then be extracted from the corresponding residual columns (cf. Alg. 8.2, l. 23).

The extension of the RHS for the NEQ groups is straightforward, as the transformed samples are directly the additional RHS to be solved for. The vector of RHS is just extended by the transformed samples (cf. Alg. 8.2 l. 6–12).

8.3.3 Computation of the Residuals $\mathbf{R}^{(0)}$ and of the Update Vector $\mathbf{H}^{(\nu)}$

From a computational point of view, the computation of the update vector $\mathbf{H}^{(\nu)}$ (cf. Alg. 8.2, l. 34) is similar to the computation of the initial residuals $\mathbf{R}^{(0)}$ (cf. Alg. 8.2, l. 13 and 22). Thus, unless stated otherwise, all concepts introduced here are applied in the computation of the initial residuals as well as for the update vector $\mathbf{H}^{(\nu)}$. As the computational requirements differ for OEQs and NEQs, a distinction is introduced in the following.

8.3.3.1 Computation for the NEQ Groups

Method I (memory saving, computationally more expensive): The normal matrices $\mathbf{N}_n$ are available in specific parameter ordering and possibly only for a (small) subset of the target parameters. Within this context, the number of parameters in $\mathbf{N}_n$ is denoted as U_n. The contribution $w_n\mathbf{N}_n\mathbf{\Pi}^{(\nu)}$ to $\mathbf{H}^{(\nu)}$ (or $w_n\mathbf{N}_n\boldsymbol{x}^{(0)} - \mathbf{n}_n$ to $\mathbf{R}^{(0)}$) of a single group n (summarized in l. 34, Alg. 8.2) to the update vector is handled within each PCGMA iteration as follows:

1. Perform a sequence of row interchanges of $\mathbf{\Pi}^{(\nu)}$ so that the parameter order of the first U_n rows corresponds to the parameter order and parameter space of $\mathbf{N}_n$, $\Rightarrow \tilde{\mathbf{\Pi}}^{(\nu)}$.
2. Recall the original NEQ matrix $\mathbf{N}_n$ from disk
3. Perform the multiplication $\tilde{\mathbf{H}}_n^{(\nu)} := w_n\mathbf{N}_n\tilde{\mathbf{\Pi}}^{(\nu)}(1 : U_n, :)$, where $\tilde{\mathbf{\Pi}}^{(\nu)}(1 : U_n, :)$ is the subset of the first U_n rows and all columns.
4. Extend $\tilde{\mathbf{H}}_n^{(\nu)}$ with zeros for all parameters not contained in the parameter space of group n. Reorder the rows of the matrix $\begin{bmatrix}\tilde{\mathbf{H}}_n^{(\nu)} & \mathbf{0}\end{bmatrix}^T$ into the numbering scheme of original $\mathbf{\Pi}^{(\nu)}$ to obtain $\mathbf{H}_n^{(\nu)}$.
5. Perform the update as the addition $\mathbf{H}^{(\nu)} + = \mathbf{H}_n^{(\nu)}$.

Method II (memory consuming, computationally less expensive, I/O reduced): It is simple and straightforward to minimize the I/O operation of reading all N NEQs from disk in every PCGMA iteration. Once in the beginning, the NEQs are combined (using e.g. the algorithm implemented for the direct solution method as introduced in Sect. 7.2.1). The dimension of the new NEQs are determined by the group n with the largest parameter space. Note that this does not effect the estimation of weights for the individual groups. This combined NEQs are held in memory during the whole PCGMA algorithm. The I/O steps for all N groups are avoided within each iteration. In addition, the update step of $\mathbf{H}^{(\nu)}$ has to be computed only once. For large N, the computation time is significantly reduced. The gain cannot be exactly quantified as it depends on N, the hardware (i.e. the I/O performance) and the parameter space of all NEQ groups n. The update step takes the time as needed for the largest NEQs (Steps. 1,3–5 of method I) minus the I/O time (as NEQs are in core). As long as there is no memory shortage, method II has no disadvantages compared to method I.

8.3.3.2 Update for the OEQ Groups

As it is assumed that for the groups introduced as OEQ the OEQs have to be set up for the whole parameter space (with the exception of possible group specific parameters), the processing of the OEQ groups is the main contributer to the total computing time as the high resolution signal is contained there. The computation of the contribution of a group o to the update vector $\mathbf{H}^{(\nu)}$, i.e. $w_o\mathbf{A}_o^T\mathbf{G_o}^T\mathbf{G_o}\mathbf{A}_o\mathbf{\Pi}^{(\nu)}$ is the dominant step with respect to computing time within each iteration. This step mainly consists of five parts,

1. the assembly of the design matrices $\mathbf{A}_o$,
2. the decorrelation $\mathbf{G_o}\mathbf{A}_o$,
3. the computation of the matrix-matrix product $\mathbf{T} := \mathbf{A}_o\mathbf{\Pi}^{(\nu)}$,
4. the computation of the matrix-matrix product $\mathbf{H}_o^{(\nu)} = \mathbf{A}_o^T\mathbf{T}$, and
5. the final update $\mathbf{H}^{(\nu)}+ = \mathbf{H}_o^{(\nu)}$.

The two matrix-matrix products are performed with PBLAS, the decorrelation is performed with a diagonal matrix only (within this study), and the final update step is a simple addition. The only step the computation can be significantly accelerated is the setup of the OEQs (i.e. $\mathbf{A}_o$) itself. As the OEQs have to be set up in each PCGMA iteration ν, the fast setup of the distributed design matrix $\mathbf{A}_o$ is essential within the implementation of the iterative solver. All necessary steps for the update of $\mathbf{H}^{(\nu)}$ by the groups o are summarized in Alg. 8.4.

Numbering Scheme for the PCGMA Algorithm For a fast setup of the design matrices, using block-cyclic distributed matrices, it would be best, if the local matrices $\mathbf{A}_o^l$ would be locally computable without communication with other processes. In addition, redundant computations of several cores should be avoided or at least minimized. Redundant computations might occur during the expensive recursive computations of the Legendre functions, which are performed by orders (e.g. Holmes and Featherstone, 2002). Thus, it is important that all columns of $\mathbf{A}_o$, corresponding to gravity field coefficients of the same order, are within a single local matrix $\mathbf{A}_o^l$, or at least spread over as few local matrices as possible.

Choosing a standard order-wise numbering scheme for the global vector of unknowns $\tilde{\mathbf{x}}$ and thus for the columns of $\mathbf{A}_o$, would result in parameters of many different orders in the columns of the local matrices $\mathbf{A}_{\mathsf{r},\mathsf{c}}^l$, due to the block-cyclic distribution. Instead of an order-wise numbering scheme for $\mathbf{A}_o$, an ordering of the parameters, depending on the parameters of the block-cyclic distribution (b_r and b_c) and the size of the compute core grid (R and C), is determined at runtime. The numbering scheme of the global matrix $\mathbf{A}_o$ is driven by the fact, that the virtual matrix composed by the local matrices along a row of the compute core grid $\hat{\mathbf{A}}_o = \begin{bmatrix}\mathbf{A}_{\mathsf{r},0}^l & \mathbf{A}_{\mathsf{r},1}^l & \mathbf{A}_{\mathsf{r},0}^l \cdots & \mathbf{A}_{\mathsf{r},\mathsf{C}-1}^l\end{bmatrix}$ follows a standard order-wise numbering scheme. Thus, the global numbering scheme might look arbitrary at first view, but results from the fact, that parameters of the same order are collected as good as possible in the same local matrices. This minimizes redundant computations in the recursive computations of Legendre functions.

After b_r, b_c and the dimension of the compute core grid are defined, an optimized numbering scheme, covering the whole parameter space is constructed at runtime. Starting point is a symbolic numbering scheme which follows a standard order-wise numbering $\mathbf{p}_\mathrm{m}$ (see e.g. Schuh, 1996, Sect. 2.2). Using (3.5), the dimension of the local matrices can be computed. Especially the number of columns $C_{\mathsf{r},\mathsf{c}}^l$ of the design matrix $\mathbf{A}_o$ is of interest for a single row of the compute core grid. It determines the number of parameters a column of the compute core grid is responsible for. Starting in the column $\mathsf{c} = 0$, the first $C_{\mathsf{r},0}^l$ parameters of $\mathbf{p}_\mathrm{m}$ are associated with the local matrix $\mathbf{A}_{\mathsf{r},0}^l$, the next $C_{\mathsf{r},1}^l$ parameters of $\mathbf{p}_\mathrm{m}$ are associated with the local matrix $\mathbf{A}_{\mathsf{r},1}^l$ and so on. The local matrices $\mathbf{A}_{\mathsf{r},\mathsf{c}}^l$

Algorithm 8.4: Computation of the update vector $\mathbf{H}^{(\nu)}$ by groups o provided as OEQs using block-cyclic distributed observation equations.

Data:

`vector<OeqGroup>` $\mathbf{g}$	information on observations for the groups $o \in \{0,\dots,O-1\}$
`NumberingScheme` $\mathbf{p}_{\text{bcd}}$	symbolic numbering scheme covering the parameters
`double` GM, a, tide_system	constants to be used in target model
`size_t` K	number of MC samples for additional RHS
`size_t` b_{obs}	number of observations to be processed at once
$\mathbf{\Pi}^{(\nu)}$	current search direction

```
1  // determine my process coordinates and compute core grid dimension
2  size_t r, c, R, C
3  blacs_gridinfo( 0, r, c, R, C )
4  // initialization of block-cyclic distributed matrices, account for MC RHS
5  size_t U = p.size()
6  DistributedMatrix $\mathbf{H}^{(\nu)}(U, 1+(N+O)K)$
7  DistributedMatrix $\mathbf{T}(b_{\text{obs}}, 1+(N+O)K)$
8  DistributedMatrix $\mathbf{A}^T(U, b_{\text{obs}})$
9  // Determine dimension of local matrices
10 size_t $R^l_{\mathbf{A}^T} = \mathbf{A}^T$.Rl()   size_t $C^l_{\mathbf{A}^T} = \mathbf{A}^T$.Cl()   // parameters associated with local rows ($\mathbf{p}_r$) of $\mathbf{A}^T$
11 NumberingSchme $\mathbf{p}_r(R^l_{\mathbf{A}^T})$
12 for $r = 0,\ r < R^l_{\mathbf{A}^T},\ r{+}{+}$ do
13     $\mathbf{p}_r(r) = \mathbf{p}_{\text{bcd}}(\mathbf{A}^T$.rowInGlobalMat$(r))$
14 end
15 // Loop over all groups provided as OEQs
16 for $o = 0,\ o < O,\ o{+}{+}$ do
17     g.at($o$).loadObservations()          // Load all observations (and meta data) of group o from file
18     // Loop over all observations of group o in blocks of $b_{obs}$
19     for $b = 0,\ b <$g.at($o$).size()$,\ b{+}{=}\,b_{obs}$ do
20         // set up $\mathbf{A}^T$ for local observations and local parameters
21         for $c = 0,\ c < C^l_{\mathbf{A}^T},\ c{+}{+}$ do
22             // overall index of observation of group o corresponding to column c
23             size_t $i = b + \mathbf{A}^T$.colInGlobalMat$(c)$
24             // fill columns with design entries for observation i and local parameters in $\mathbf{p}_r$
25             fillDesign($\mathbf{A}$.localMat().colPtr($c$), g.at($o$).obs($i$), $\mathbf{p}_r$ )
26         end
27         // update $\mathbf{H}^{(\nu)}$ with observations $b$ to $b + b_{obs} - 1$
28         $\mathbf{T} := \mathbf{A}\mathbf{\Pi}^{(\nu)}$
29         $\mathbf{H}^{(\nu)} {+}{=} w_o \mathbf{A}^T \mathbf{T}$
30     end
31     // Same operations for the rest block ($M_o \% b_{obs}$)...
32     $\mathbf{A}^T$resize$(U, M_o \% b_{\text{obs}})$
33     // Determine dimension of local matrices
34     $R^l_{\mathbf{A}^T} = \mathbf{A}^T$.Rl()   size_t $C^l_{\mathbf{A}^T} = \mathbf{A}^T$.Cl()
35     // set up $\mathbf{A}^T$ for local observations and local parameters
36     for $c = 0,\ c < C^l_{\mathbf{A}^T},\ c{+}{+}$ do
37         // overall index of observation of group o corresponding to column c
38         size_t $i = M_o \div b_{\text{obs}} + \mathbf{A}^T$.colInGlobalMat$(c)$
39         // fill columns with design entries for observation i and local parameters in $\mathbf{p}_r$
40         fillDesign($\mathbf{A}$.localMat().colPtr($c$), g.at($o$).obs($i$), $\mathbf{p}_r$ )
41     end
42     ...
43 end
44 return $\mathbf{H}^{(\nu)}$// Update vector for PCGMA
```

Algorithm 8.5: Creation of a tailored symbolic numbering scheme at runtime keeping parameters of the same order coherent in the local matrices of a block-cyclic distributed design matrix for the use within PCGMA.

Data:

`NumberingScheme` $\mathbf{p}_m$	Symbolic numbering scheme following a standard order-wise numbering
`size_t` b_r, b_c	Parameters of the block-cyclic distribution
`size_t` R, C	Dimension of the compute core grid

```
// initialization
size_t i = 0
// Initialize an empty numbering scheme of correct size
NumberingScheme $\mathbf{p}_{\text{bcd}}$ ($\mathbf{p}_m$.size())
// Loop over all columns of the compute core grid
for c = 0; c < C; c + + do
    // Determine number of columns of local matrices of compute core grids column c (cf. (3.5))
    $C^l_{*,c}$ = colsInLocalMat($b_c$, C, $\mathbf{p}_m$.size())
    for $c^l = 0; c^l < C^l_{*,c}; c^l + +$ do
        // Determine global column c local column $c^l$ of grid column c corresponds to (cf. (3.6))
        size_t $c$ = colInGlobalMat($c^l$, $b_c$, c$^l$, C)
        $\mathbf{p}_{\text{bcd}}$.p($c$) = $\mathbf{p}_m$.p($i$)
        // update counter
        i++
    end
end
return $\mathbf{p}_{\text{bcd}}$ // symbolic numbering scheme
```

are set up for that parameters, which now cover only a few orders and contain mainly coefficients of the same order.

This concept results in Alg. 8.5, which creates the symbolic numbering scheme covering all parameters and follows the ideas mentioned above. The columns of global matrix $\mathbf{A}_o^l$ follows the new numbering scheme $\mathbf{p}_{\text{bcd}}$, which results in coefficients grouped by their order m in the local matrices $\mathbf{A}^l_{\text{r,c}}$, which minimizes redundant recursions during the evaluation of the Legendre polynomials on different processes.

Setup of the Design Matrix For the observation-wise computation of the design matrix $\mathbf{A}_o$ and using CMO in the local matrix, a setup of $\mathbf{A}_o^T$ is more efficient than a setup of $\mathbf{A}_o$. Setting up $\mathbf{A}_o^T$ instead of $\mathbf{A}_o$ yields to adjacent entries in the compute core's main memory which to the same observation (which are computed in a recursion). They can be accessed more efficiently during the setup. As a matrix-matrix product[12] using $\mathbf{A}_o^T$ and a matrix-matrix product using $\mathbf{A}_o$ has to be computed in the algorithm, there is no consequence for the runtime of the computations of both matrix-matrix products. Thus, the design matrix is setup as $\mathbf{A}_o^T$. Note that setting up $\mathbf{A}_o^T$ changes the setup and argumentation of the numbering scheme creation, as now the parameters are associated with the rows of $\mathbf{A}_o^T$.

Because the number of observations in a single group can be huge, as for the other applications in Chap. 6 and 7, the observations of a single group are processed in blocks of b_{obs} observations. Since in contrast to the GOCE data analysis in Chap. 6, there are no correlations in the observations assumed in this study, the processing of the b_{obs} observations is independent from the other sub-blocks of the same group. The update of $\mathbf{H}^{(\nu)}$ from a single block of these b_{obs} is sequentially computed in a loop over all sub-blocks b (cf. Alg. 8.4, l.28–29).

[12] In fact it is a matrix-vector like product, as the number of columns of the second matrix is much smaller than the number of rows.

8.3.3.3 Reordering and Preconditioning

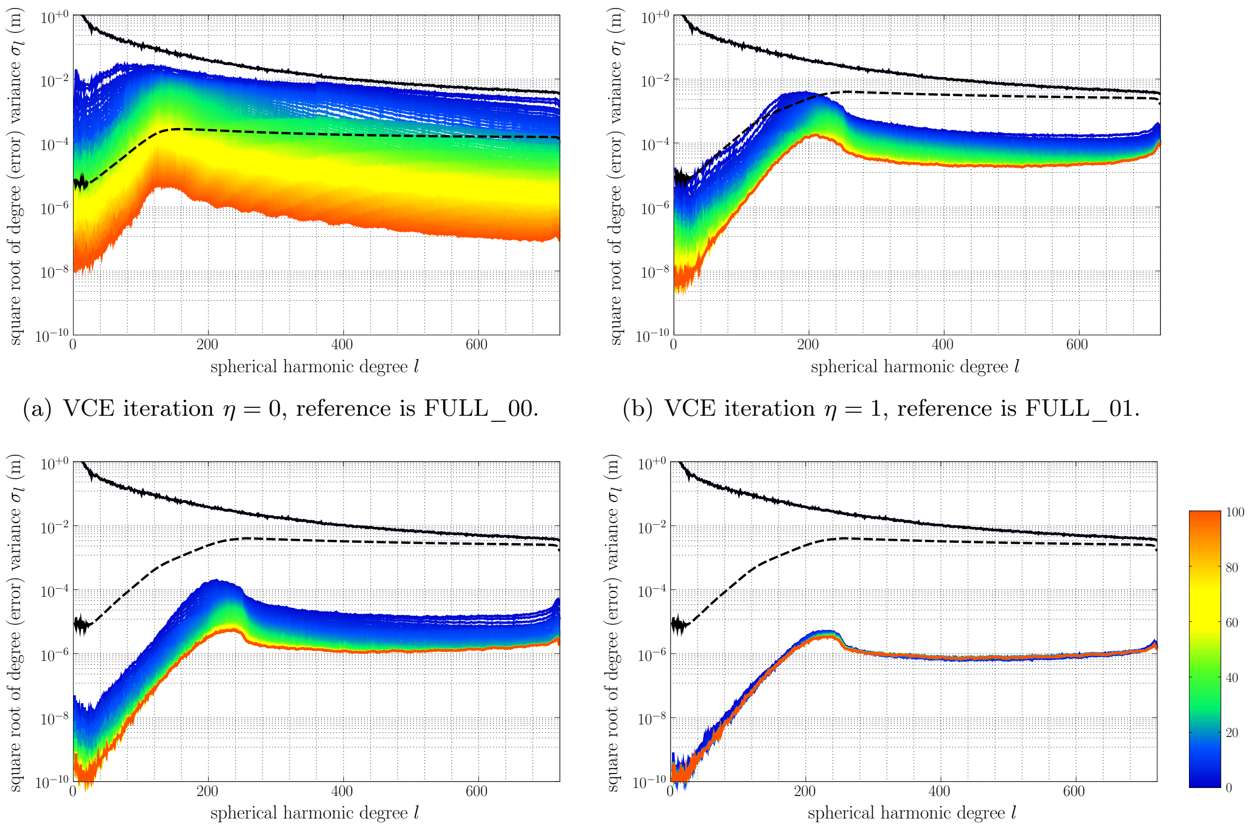
As described above, only block diagonal preconditioners are used within the PCGMA implementation derived so far. Within gravity field recovery, the NEQs are usually block diagonal dominant, if an order-wise numbering scheme is chosen. As the tailored numbering scheme for the setup of the design matrices $\mathbf{p}_{\text{bcd}}$ was introduced, all matrices, which are related to the parameters in the implementation, follow the numbering scheme as described by $\mathbf{p}_{\text{bcd}}$. This of course comprises the residuals $\mathbf{R}^{(\nu)}$, which must be transformed by the preconditioner matrix. Before the preconditioner can be applied, $\mathbf{R}^{(\nu)}$ has to be reordered from $\mathbf{p}_{\text{bcd}}$ to $\mathbf{p}_{\oplus}$. Afterwards, the preconditioning can be performed as described in Sect. 8.3.1.3. After the preconditioning, the preconditioned residuals are reordered from $\mathbf{p}_{\oplus}$ to $\mathbf{p}_{\text{bcd}}$ again, to be consistent to the design matrices and other matrices used within the next PCGMA iteration.

8.4 Closed-Loop Simulation

8.4.1 Proof of Concept

To have a direct reference solution for the spherical harmonic coefficients as well as for the VCs, the same simulation scenario as in Sect. 7.3 was used. The PCGMA implementation was used in $\eta_{\max} = 4$ VCE iterations to recover the coefficients and to derive the estimation of VCs. To guarantee convergence, $\nu_{\max} = 100$ PCGMA iterations are performed per VCE iteration. As a start solution, the independent d/o 360 EGM96 model was chosen (Lemoine et al., 1996), the coefficients above degree 360 were initialized with zeros.

Fig. 8.1 shows the results in terms of degree variances. Differences to the reference solution, i.e. the corresponding direct solution as derived in Sect. 7.3.2 (FULL_00–FULL_03), are plotted for each of the 100 iterations performed as solid lines. The solutions after the last, i.e. the 100th iteration are named ITER_00–ITER_03. Of course, as the same synthetic data is used in the simulations, the reference solution and the PCGMA solution should be theoretically the same after PCGMA convergence, as both are the least squares estimates from the same data set. But in the direct and in the iterative solution, the VCs where independently computed, using the stochastic trace estimator, such that the estimated weights can not be numerically the same. The generated samples differ and the iterative solver has different numerical properties. In addition to the differences, the black dashed lines shows the quality of the reference solution and thus of the solution itself, to demonstrate the accuracy level of the solution. Within the iteration $\eta = 0$, after about 60 iterations, the iterative solutions is within the quality level of the reference solution and thus within the level of the achievable accuracy. As within the iteration $\eta = 1$ the start solution improved (ITER_00), the quality level of the reference solution is reached after about 10 iterations. Afterwards up to iteration 50–60, the iterative solution gets closer to the reference solution. After that an additional improvement is not visible anymore. This is related to slightly differing weights estimated in both solvers, so that the solutions can not be the same within machine precision. For the iteration $\eta = 2$, the start solution is already significantly better than the reference solution's external accuracy. After again 50–60 iterations, the difference with respect to the reference solution is two to four orders of magnitude below the accuracy of the reference solution. Thus, it is safely concluded that the solutions are identical within their error bars. In the final iteration $\eta = 3$, an improvement towards the reference solution is visible only for some degrees and only for the first 10–20 iterations. After that the solutions does not iterate towards the reference solution. This is again due to the weights, which slightly differ, as they are estimated with both solvers using the stochastic estimator. The conclusion is confirmed by the fact, that the largest differences to the reference solution occur within the overlapping area of the satellite and the terrestrial data. Within that spectral range, all observation groups contribute.



(a) VCE iteration $\eta = 0$, reference is FULL_00.

(b) VCE iteration $\eta = 1$, reference is FULL_01.

(c) VCE iteration $\eta = 2$, reference is FULL_02.

(d) VCE iteration $\eta = 3$, reference is FULL_03.

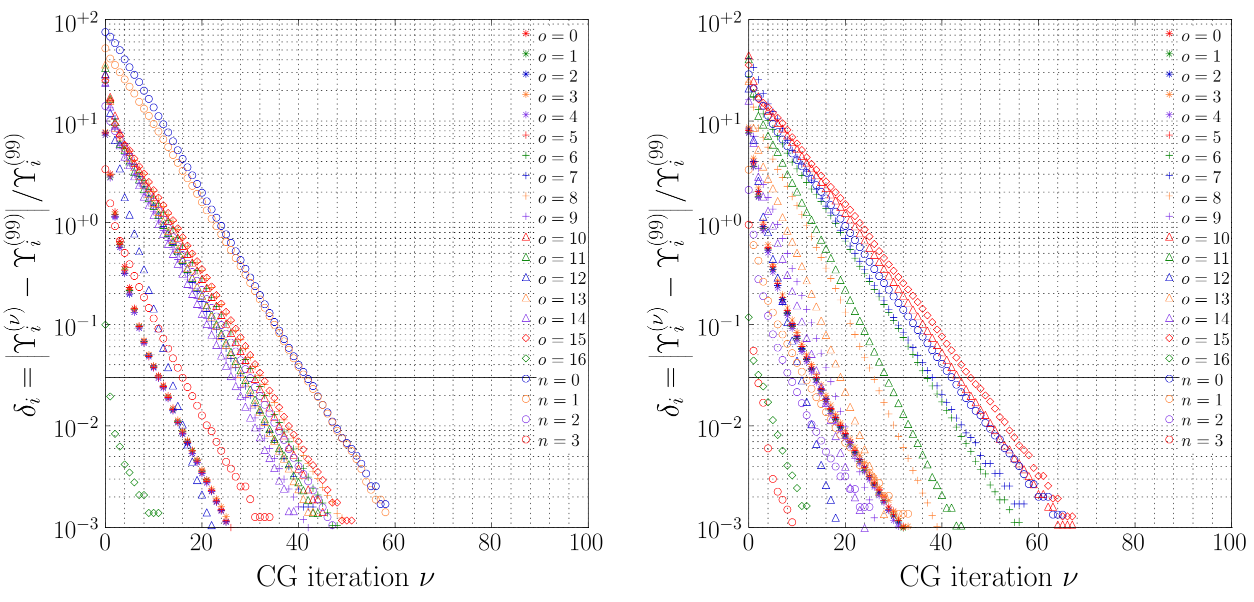
Figure 8.1: Convergence behavior of PCGMA algorithm within the four VCE iterations in terms of degree variances. PCGMA iterations are shown as colored lines. The reference solutions are the solutions obtained via assembly and solution of full NEQs of the same VCE iteration (FULL_0η). The solid black lines are the signal of the reference solution, whereas the dashed black lines show the quality (formal errors) of the reference solution.

For the spherical harmonic coefficients only, an estimate for the required iterations is for $\eta = 0$ 50–60, $\eta = 1$ 50–60, $\eta = 2$ 40–50 and $\eta = 3$ 20–30. All in all, about 150 iterations should be sufficient for the approximately 520 000 spherical harmonic coefficients. After that, the difference of both least squares solutions is two orders of magnitudes below the solution's (external) accuracy. Of course, the numbers provided significantly depend on the quality of the start solution.

The number of required iterations for VCE might differ, as the initial values of the additional parameters required for MC trace estimation are always initialized with zeros[13]. Whereas it can be assumed that Ω_i converges within an iteration η if the spherical harmonic coefficients converged, one has to look into the estimates of the partial redundancy or more precisely into $\tilde{\Upsilon}_i$. For testing purposes this was computed within each of the 100 iterations. Fig. 8.2 shows the estimated value for $\tilde{\Upsilon}_i$ for every group in every CG iteration ν. The plots are shown for iterations $\eta = 0$ and $\eta = 1$. The plots for $\eta = 2$ and $\eta = 3$ look the same as for $\eta = 1$, as the weights change only slightly and the convergence is comparable within iteration $\eta = 1$, $\eta = 2$ and $\eta = 3$.

The convergence behavior changes with the weights. As it depends strongly on the configuration of the group (e.g. number of observations in that group, data sampling, etc), it should not be

[13]In the current implementation, the additional MC random samples for VCE are always newly generated in a new VCE iteration. Thus, no a-priori information about the additional MC parameters is available from a former VCE iteration.



(a) For VCE iteration $\eta = 0$.

(b) For VCE iteration $\eta = 1$.

Figure 8.2: Convergence of the number of parameters $\tilde{\Upsilon}_i$ determined by group i within the PCGMA algorithm. Shown is the relative error with respect to the estimated result of the last, i.e. the 100th iteration. The black solid line marks an empirical derived simple error level of 0.03 %, which results from a simple error estimate of the MC trace estimator i.e. $(U - \sum_i \tilde{\Upsilon}_i)/U$.

discussed in detail. However, it should be emphasized that always initial values of zero are used. The convergence behavior is the same in every VCE iteration, if the weights do not significantly change. To obtain an error of less than 0.1 % for every group, about 35–40 iterations are needed in every VCE Iteration. These are slightly more iterations than for the spherical harmonic coefficients in the iterations $\eta = 2$ and $\eta = 3$. Nevertheless, depending on the application and the configurations of the group, higher errors in the trace estimation are acceptable. The higher the of number of observations in the group, the smaller is the effect on the error for the VCs. It depends on the ratio M_i/Υ_i (cf. Brockmann and Schuh, 2010).

Finally, Tab. 8.2 shows the estimates of the VCs derived with the iterative solver compared to the true values. The same conclusions as for the direct solver (cf. Sect. 7.3.2) hold, especially comparing Tab. 8.2 and 7.4. The error of the derived weights only differs around 0.1 % compared to the errors for the direct solver. Except for the tiny group $o = 12$, the error of the variance component is 0.9 % larger compared to the error obtained with the direct solver. The simulation demonstrates that the integration of VCE in the iterative solver works and the right weights are derived within a number of iterations which is comparable to the iterations required for parameter convergence.

8.4.2 Preconditioners and Convergence

The convergence characteristics totally depend on the analyzed scenario. It is influenced by the interaction of the spherical harmonic resolution (parameter space), the spatial distribution of the observations (on the sphere) and the observation's noise. These characteristics significantly influence the condition of $\mathbf{N}$ and thus the convergence of the iterative solver (e.g. Golub and van Loan, 1996, Sect. 10.2.7). Applying a block diagonal preconditioner as introduced in Sect. 8.3.1.1 modifies the condition of the transformed system. To demonstrate the dependence, consider the usage in the following extreme example. The introduced preconditioner models are applied in a scenario with

Table 8.2: Standard deviations $\sigma_{n,o} = 1/\sqrt{w_{n,o}}$ as derived by VCE for OEQ and NEQ groups. The values are provided for all iterations performed. The last two columns provide the relative error w.r.t. the true value which was used in data generation for iteration $\eta = 1$ and $\eta = 4$ respectively.

group n/o		$\sigma_{n,o}^{(0)}$	$\sigma_{n,o}^{(1)}$	$\sigma_{n,o}^{(2)}$	$\sigma_{n,o}^{(3)}$	$\sigma_{n,o}^{(4)}$	$\sigma_{n,o}^{\text{true}}$	$\frac{\lvert\sigma_{n,o}^{(1)}-\sigma_{n,o}^{\text{true}}\rvert}{\sigma_{n,o}^{\text{true}}}$	$\frac{\lvert\sigma_{n,o}^{(4)}-\sigma_{n,o}^{\text{true}}\rvert}{\sigma_{n,o}^{\text{true}}}$
n	0	1.0000	0.0642	0.0612	0.0612	0.0612	0.0606	6.0 %	0.9 %
	1	1.0000	1.3168	1.2900	1.2900	1.2900	1.2910	2.0 %	0.1 %
	2	1.0000	0.7460	0.7453	0.7453	0.7453	0.7454	0.1 %	0.0 %
	3	1.0000	0.8347	0.8165	0.8165	0.8165	0.8165	2.2 %	0.0 %
o	0	1.0000	12.1588	12.0203	12.0069	12.0047	12.0000	1.3 %	0.0 %
	1	1.0000	13.0111	12.9925	12.9928	12.9932	13.0000	0.1 %	0.1 %
	2	1.0000	13.9303	14.0007	14.0073	14.0085	14.0000	0.5 %	0.1 %
	3	1.0000	14.8289	14.9924	15.0089	15.0101	15.0000	1.1 %	0.1 %
	4	1.0000	12.2302	12.1256	12.1153	12.1133	12.1000	1.1 %	0.1 %
	5	1.0000	13.1070	13.0788	13.0768	13.0761	13.1000	0.1 %	0.2 %
	6	1.0000	20.9596	21.1191	21.1023	21.1156	21.0000	0.2 %	0.6 %
	7	1.0000	21.9253	21.9440	21.9435	21.9424	22.0000	0.3 %	0.3 %
	8	1.0000	13.8841	13.9451	13.9883	13.9038	14.0000	0.8 %	0.7 %
	9	1.0000	13.0290	13.0198	13.0368	13.0350	13.0000	0.2 %	0.3 %
	10	1.0000	23.9928	24.0620	24.0487	24.0531	24.0000	0.0 %	0.2 %
	11	1.0000	16.4410	16.6844	16.6437	16.6698	17.0000	3.3 %	1.9 %
	12	1.0000	12.7355	12.6965	12.7356	12.8659	12.0000	6.1 %	7.2 %
	13	1.0000	10.6570	10.5835	10.4967	10.6022	11.0000	3.1 %	3.6 %
	14	1.0000	10.0176	9.9834	9.9780	9.9806	10.0000	0.2 %	0.2 %
	15	1.0000	24.8471	25.0009	24.9835	25.0392	25.0000	0.6 %	0.2 %
	16	1.0000	22.9761	22.9839	22.9804	22.9809	23.0000	0.1 %	0.1 %

an equidistant data distribution of the observations, in addition, the observations have a constant accuracy (per latitude). The condition of the transformed system is 1.0, as $\mathbf{N} = \mathbf{N}_{\oplus}$. The solver converges, independent of the dimension of the parameter space, after a single iteration. Leaving the ideal data distribution, e.g. by adding/deleting a single observation, will change the situation, as $\mathbf{N}$ results in a dense matrix. Convergence is expected to get worse, the more the idealized scenario (assumed for preconditioner setup) is left.

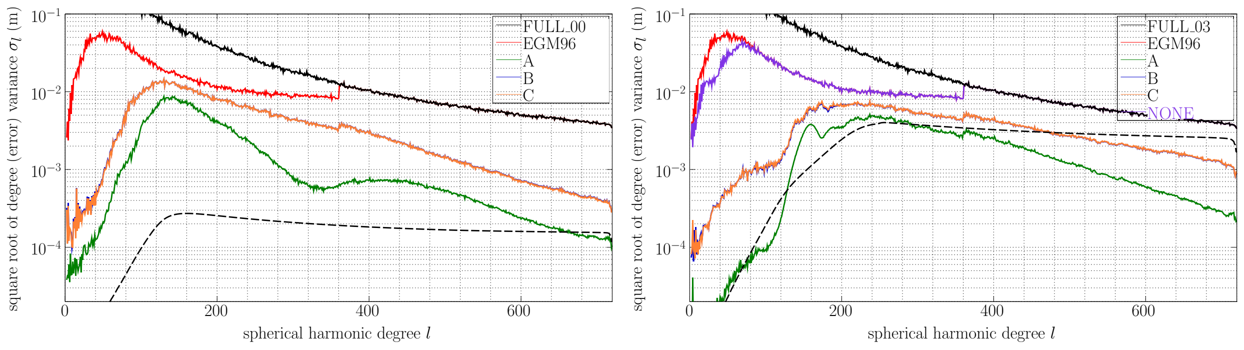
For this purpose, general comments and conclusions for the convergence characteristics are not possible. But it has to be emphasized that:

- The block diagonal preconditioners are always positive definite, and the solution in the preconditioning step is always computed rigorously. The preconditioning is independent of the compute core grid setup and the parallel implementation.
- The preconditioner is chosen motivated by the properties of the spherical harmonic base functions. For the implemented models it is expected that the more uniform the observations are distributed and the more homogenous (per latitude) the observation accuracies, the better the convergence, using this kinds of preconditioners.

Although for the analyzed scenario, the data distribution is chosen (on purpose) irregular, e.g. along the satellites ground track, and the noise characteristics are by far not only latitude dependent (cf. Fig. 7.1), it should be shown that these simple preconditioners improve convergence.

8.4.2.1 Preconditioner models A, B and C

The three preconditioner models A, B and C as described in Sect. 8.3.1.1 were used for the scenario mentioned above. Using the three preconditioners, 10 PCGMA iterations are performed with fixed weights. As the performance of the preconditioners may depend on the weights, two simulations are performed. The first was performed using the weights $w_i^{(0)} = 1.0$ which are inconsistent to the



(a) Weights $w_i^{(0)}$, reference is FULL_00. (b) Weights $w_i^{(3)}$, reference is FULL_03.

Figure 8.3: Convergence of PCGMA using the preconditioner models A, B and C. Shown are the results after $\nu_{\max} = 10$ PCGMA iterations in terms of degree variances. The reference solutions are the solutions obtained via assembly and solution of full NEQs of the same VCE iteration (FULL_0η). The solid black lines are describe the signal of the reference solution and the solid colored lines the difference to the reference solution. The red lines indicate the quality of the start solution EGM96 compared to the reference solution and the dashed black lines the formal errors of the reference solution.

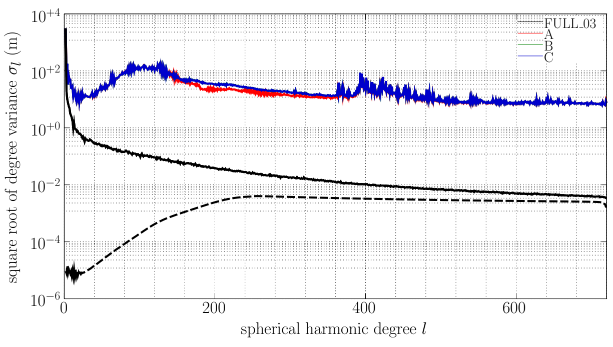
observations and the second for weights consistent to the observations i.e. using the weights as estimated with VCE ($w_i^{(3)}$). For both scenarios, the start solution was fixed to the EGM96 model, which can be characterized as independent of the generated observations and is of low quality compared to EGM2008, which was used as the true model.

Starting with equal weights $w_i^{(0)} = 1.0$, Fig. 8.3(a) shows the results compared to the FULL_00 solution after ten iterations for all three preconditioner models (A, B and C). The first conclusion is, that the solutions of the preconditioner models B and C are of the same quality. The difference is only small and hardly visible in the plot. After ten iterations, the solution obtained with preconditioner A is much closer to the optimal solution. Thus the convergence with preconditioner model A is much better. The additional (incomplete) correlations in models B and C might be meaningless without the neglected correlations outside the block diagonal structure. The result was not expected, as the shape of preconditioner model A assumes equatorial symmetry for the data distribution and data accuracies (which is for sure not the case for the simulation scenario, cf. Fig. 7.1).

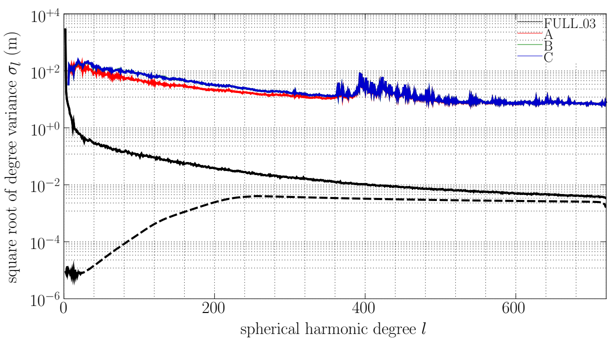
The conclusions are the same for the second run, which started with close to optimal weights $w_i^{(3)}$, Fig. 8.3(b) shows the results compared to the FULL_03 solution after ten iterations for all three preconditioner models. Again, after 10 iterations, the solution computed with preconditioner A is closer to the optimal solution. In addition, for iteration $\eta = 3$, the purple line shows a solution without using a preconditioner. It is demonstrated that all preconditioner models have a positive effect for convergence. The solutions, which use one of the preconditioners, are one to two orders of magnitude better after the ten iterations performed.

8.4.2.2 Solutions from Preconditioner only

As shown e.g. by Reguzzoni and Sansò (2012), just deriving a block diagonal solution, which accounts for individual observations weights and violates the equidistant sampling of the observations does not provide reasonable solutions for the gravity field parameters. Just ignoring the correlations outside the blocks of the same order, destroys the whole system of NEQs. Fig. 8.4 shows the solution which is the result of the preconditioner solution, solving the preconditioner for the right hand sides $\mathbf{n}$ (which both include the individual weights per observation, in addition the data



(a) For all groups o and n.



(b) For OEQ groups o only.

Figure 8.4: Signal degree variances for the solutions derived from the block diagonal preconditioners only. The left plot shows the solution from all groups (NEQ and OEQ), whereas the right plot shows the solution from OEQ groups only. For both cases, the resulting coefficients are unusable. Note that the green and the blue lines (model B and C) are nearly identical and the differences are not visible.

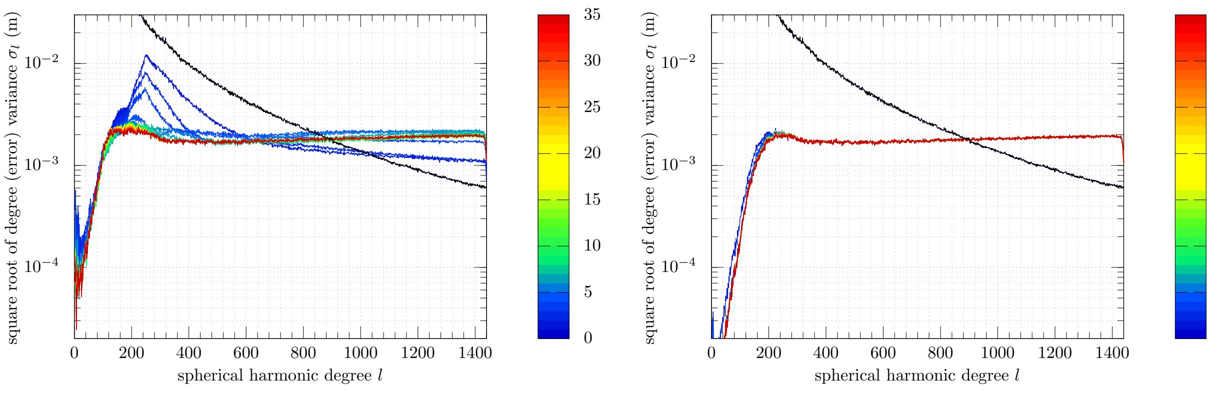
distribution is irregular, cf. Sect. 7.3). The solution was computed for all groups n and o and for the combined OEQ groups o only and for all three preconditioner models implemented. The weights $w_i^{(3)}$ were used in the combination. The derived three solutions are compared to the reference solution FULL_03. Only the signal degree variances are shown. As they are significantly above the signal of the reference solution, the differences are not shown.

Obviously, the solutions are not usable as a gravity field. But as shown in the previous section, all preconditioners accelerate the convergence of PCGMA and can be used as a preconditioner. Thus, the block diagonal preconditioners remain a simple and useful choice. General convergence properties can not be concluded from the simulations already performed, they can only be numerically identified for a fixed scenario. A systematic numerical study, as performed in Brockmann and Schuh (2011), is more useful with a simulation scenario even closer to a real data scenario (e.g. including typical correlations between observations), as for this high dimensional parameter space the computational requirements are huge. But it is for sure a field for further studies. In combination with the direct solver from Chap. 7, an explicit computation of the preconditioned, i.e. transformed, system of equations is possible, whose spectral properties can be studied. The results from such a study can be used in addition to design alternative preconditioners.

8.4.3 High Degree Closed-Loop Simulation

As a proof, that the solver is able to process higher dimensional problems with millions of unknown parameters, a scenario comparable to the scenario introduced in Sect. 7.3 was processed. Without going into many details, the scenario is characterized. Compared to the scenario mentioned above, the resolution of the spherical harmonic series was duplicated. Thus, the parameter space increased by a factor of four. $U = 2\,076\,477$ unknown parameters were estimated in the simulation.

Whereas the NEQ groups are exactly the same as in Sect. 7.3.1.1 (and Tab. 7.2), the OEQ groups had to be adjusted. A higher spatial resolution is required for the increased parameter space. A full CryoSat-2 cycle (partioned into sub-cycles) with 0.5 Hz sampling ($10 \cdot 10^6$ observations) was introduced as 13 groups based on OEQs. Whereas the terrestrial areas are the same as in Tab. 7.3, the observations were generated with a higher spatial resolution on a $0.05° \times 0.05°$ grid. This 11 groups together consist of $8.5 \cdot 10^6$ observations. A white noise, higher than in Sect. 7.3.1.2, was added to all observations, following the same strategy, but with an increased σ_o.



(a) VCE iteration $\eta = 0$, reference is EGM2008. (b) VCE iteration $\eta = 1$, reference is EGM2008.

Figure 8.5: Convergence behavior of PCGMA algorithm within the the first two VCE iterations in terms of degree variances. PCGMA iterations are shown as colored lines. The model used to generate the input data is used as reference solution. The solid black lines are the signal of the reference solution.

Thus, within the simulation $2 \cdot 10^6$ unknowns are estimated from $N = 4$ NEQ groups and $O = 24$ OEQ groups containing $18.5 \cdot 10^6$ noisy observations. In addition, for all groups introduced, $N+O = 28$ VCs were estimated. To save computing resources, $\eta_{\text{max}} = 3$ VCE iterations are performed with $\nu_{\text{max}} = 35$ PCGMA iterations for $\eta = 0$ and $\eta = 1$ and $\nu_{\text{max}} = 10$ for $\eta = 2$. As a start solution, a solution derived from the four NEQ groups only was chosen (derived with the direct solver cf. Chap. 7 for $N = 4$ and $O = 0$, including VCE). Whereas the weights for groups o were initialized with 1.0, the initial weights for groups n were chosen as derived for the NEQ-only solution.

Fig. 8.5 characterizes the solutions. As for this high resolution scenario now direct solution is available, it is compared to the EGM2008 model used for the generation of the observations (and normal equations). Degree variances with respect to the input model EGM2008 are shown for the first two VCE iterations. It shows, that the solution converges along the PCGMA iterations.

Note, due to the simplified white noise assumption, an unrealistically high white noise was added ($20 - 30$ mGal and $22 - 29$ cm respectively). This results in a model with signal to noise ration equal to one already at degree 900 (cf. Fig. 8.5). Nevertheless, this does not affect the proof, that the solver is usable for such a high degree spherical harmonic analysis. The square root of the variance components converged, after two iterations up to the second digit (except for the smaller groups, they converged worse, but are in the correct order of magnitude). A single PCGMA iteration took less than half an hour on JUROPA using a 64×64 compute core grid.

8.5 Runtime analysis of the PCGMA Implementation

The d/o 720 simulation scenario (cf. Sect. 7.3) was used to study the implementation with respect to its performance. For the systematic performance assessment, all $N = 4$ NEQ groups were used and for the OEQ only group $o = 0$ (cf. Tab. 7.3) was used in the PCGMA iterations (sequential processing of OEQ groups). The computing time for the performance analysis was significantly reduced. To keep the relation (close to the simulation scenario described above) between the processing of the NEQ groups and the OEQ groups, the runtime measured for the processing of the single OEQ group was scaled with the factor of 7.0 to approximately scale it up to the processing of $4 \cdot 10^6$ observations. As the runtime is (nearly) linear in the number of observations and the number of additional RHS was accounted for, the provided runtime corresponds to the processing of 4 million observations as used in the complete simulation scenario. The up-scaling is slightly idealized, as I/O and PBLAS performance drops for very small groups. This additional runtime is neglected (small number of observations in the design matrix for e.g. groups with $M_o < b_{\text{obs}}$). But otherwise, the runtime would be dominated by the NEQ group processing, which would not be the case for scenarios the solver was implemented for (dominant high resolution OEQs). $\nu_{\max} = 5$ iterations were performed. The runtime provided always refers to a single PCGMA iteration, which is computed as the mean value from the 5 iterations performed. Within the runtime simulations, the preconditioner model C was used. The computations were again performed on JUROPA at FZ Jülich.

8.5.1 Runtime and Scaling Behavior

Different quadratic compute core grids were used, starting from 8×8 to 56×56 and the mean wall clock time for a single PCGMA iteration was measured. The block-cyclic distribution parameters were fixed to the suggested default value of $b_r = b_c = 64$. The simulation should demonstrate both, the scaling properties and the flexible use of the implementation on different compute core grids (and thus compute cluster systems of different dimension).

8.5.1.1 Initialization Step and Preconditioner Setup

Within a full scale simulation as discussed in Sect. 8.4.1, the processing time is driven by the PCGMA iterations (as they are repeated often). The first steps of PCGMA are separately discussed and shortly summarized. Basically these steps are:

- Reading the preconditioner from disk, combination of the preconditioner for all groups N and O and the Cholesky factorization of the preconditioner
- Generation of additional MC RHS and computation of the initial residuals of the OEQ groups
- Generation of additional MC RHS and computation of the initial residuals of the NEQ groups
- Finally the combination of all N NEQ groups to a single NEQ group for the PCGMA iterations (cf. second paragraph in Sect. 8.3.3.1)

To get an idea of the runtime required for this steps, some numbers are provided for the extrema of the compute core grids analyzed. A systematic performance study for this steps is not performed.

Reading, Combining and Factorizing of the Preconditioner The reading, combining and factorizing of the preconditioner for the described simulation scenario with $O = 17$ and $N = 4$ groups takes 600 s–900 s on the different compute core grids analyzed (quadratic, from 8×8 to 56×56). Within the block diagonal preconditioner, many small to tiny matrices are involved in the blocks. Within this simulation the blocks of the block diagonal preconditioner are composed by $O \cdot 720$ matrices of dimensions 2×2 to 1440×1440 for the OEQ groups. The reading, distribution and combination is not very effective on compute core grids using hundreds to thousands of compute cores. The same holds for the extraction of the blocks from the NEQs for the processing of the NEQ groups. As described by Sect. 8.3.1.2, a sequence of operations has to be performed on the NEQs to extract the tiny sub-blocks, which enter the preconditioner. Depending on the dimension of the original NEQ matrix $\mathbf{N}_n$, a larger number of cores does not provide gain with respect to performance. The effect on performance is rather contrary. As the small blocks of the NEQs are distributed on much more cores, the extraction of the required blocks and the required reordering is more complex and more communication is required over a larger number of involved compute cores. The final step with respect to the preconditioner is quiet fast: the Cholesky factorization of the combined matrix is computed in 3 s–6 s, nearly independent of the compute core grid (as only the small sub-blocks are factorized). Nevertheless, the whole step is solved in 600 s–900 s, which corresponds to the time needed for 1–2 iteration steps (as shown below). These steps of preconditioner handling were not yet optimized to be as flexible as possible and to have the opportunity to study alternative preconditioner types in the future.

Initialization of OEQ Groups The initialization step for the OEQ groups is composed of the computation of the initial residuals and the generation of the MC RHS (cf. Alg. 8.2, l. 17–23). On all compute core grids analyzed, the performance is the same as for the computation of the update vector $\mathbf{H}$ in the PCGMA iterations (analyzed in Sect. 8.5.1.2). The numerical complexity of the computations is the same and the additional generation of MC RHS is insignificant.

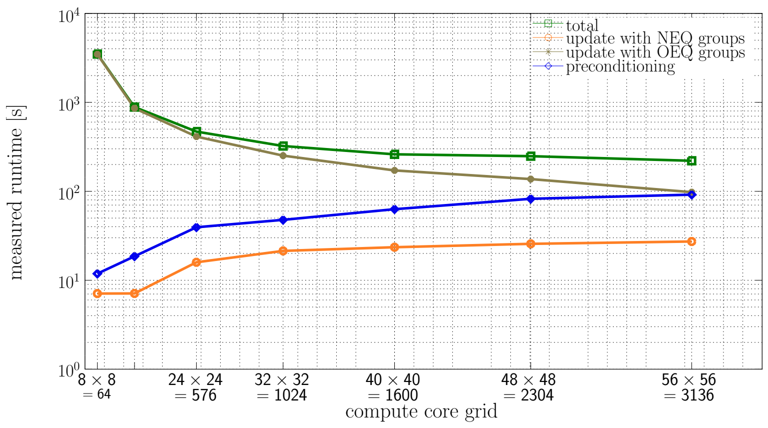
Initialization and Combination of NEQ Groups The initialization step for the NEQ groups is composed, as for the OEQ groups, of the generation of MC RHS and of the computation of the initial residuals. The computation of the initial residuals behaves as the computation of the update vector $\mathbf{H}$ (see Sect. 8.5.1.2). It is faster for the lower dimensional NEQ groups n. Within the computation of the update vector $\mathbf{H}$, only a single, i.e. combined, NEQ matrix is used instead of the individual NEQs of groups $n \in N$. Four additional steps are required here (cf. Sect. 8.3.3.1), i.e. the reading of the original NEQ from disk (in the computation of $\mathbf{H}$ the combined NEQ is kept in core), a possible transformation of the RHS and the Cholesky factorization for transformation of the MC samples. Of course the sample transformation itself needs to be computed, too. As this step is performed for each NEQ group, the performance significantly depends on the size of the original NEQs. For the small SLR NEQs the step is performed in 1 s on the 8×8 compute core grid and in 5 s on the 56×56. For the largest NEQs (GOCE SGG) the initial steps takes 324 s on the 8×8 compute core grid and in 56 s on the 56×56 compute core grid. This step shows, that the used largest compute core grids are over dimensioned for the NEQs used within the simulation. But, as shown later on, the computational demanding task is the processing of OEQ groups, where these higher dimension compute core grids are appropriate.

8.5.1.2 Runtime and Scaling of the PCGMA Iteration Steps

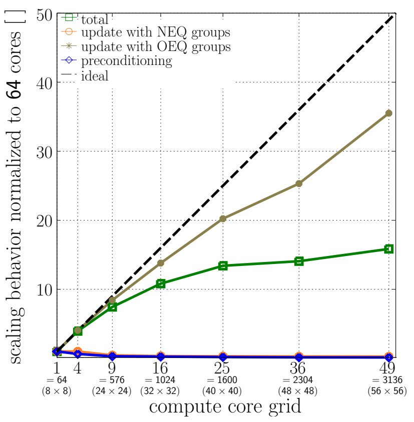
Fig. 8.6 shows the derived performance for the different setups of the compute core grid analyzed. In addition to the runtime (cf. Fig. 8.6(a)), the scaling normalized to the smallest compute core grid, i.e. the 8×8 compute core grid is shown in Fig. 8.6(b). Within this first step, the PCGMA iteration is split into update of $\mathbf{H}^{(\nu)}$ for the NEQ groups (cf. Sect. 8.3.3.1), for the OEQ groups (cf. Sect. 8.3.3.2) and the most intensive non-group specific computation, i.e. the preconditioning. All other non-group specific computations (cf. Alg. 8.2) are not significant and have a joint runtime in the order of 1 s–3 s (one percent level of total runtime).

Total, OEQ group Processing, NEQ group Processing and Preconditioning Although the total runtime (green lines) decreases with a higher number of cores in the compute core grid, the scaling behavior is far away from the ideal curve (black). Using more than 576 compute cores, e.g. 1024, the scaling is 10.8 instead of the ideal value 16. For the largest grid, i.e. using 3136 cores, a scaling of 15.8 is observed, which is poor compared to the ideal value of 49. The steps, responsible for the inappropriate scaling behavior can be identified, differentiating the total runtime into the three most intensive parts of the PCGMA algorithm, i.e. i) the processing of NEQ groups, ii) the processing of OEQ groups and iii) the preconditioning step.

The processing of the NEQ groups (involving the steps presented in Sect. 8.3.3.1) is the fastest step, 7 s–28 s are needed for the scenario analyzed. But, it can be observed that the runtime for the NEQ



(a) Absolute runtime measured for the operations of a single PCGMA iteration on different compute core grids.



(b) Scaling behavior of the operations normalized to the $8 \times 8 = 64$ grid.

Figure 8.6: Measured performance of the implemented steps of a single PCGMA iteration. Results are shown for all operations involved (green) and the three most intensive operations, i.e. the processing of OEQ groups (orange), the processing of NEQ groups (blue) and the steps required for the preconditioning.

processing increases using more cores in the grid (orange lines). Consequently, the scaling is below 1.0. Later on, this step is split up to identify the critical steps in the algorithm.

In contrast to that, the processing of the OEQ groups (brown lines) significantly decreases using more cores in the grid. Up to the largest compute core grid, it is the step of the algorithm which dominates the runtime. The scaling is close to linear, but not ideal. Nevertheless, for the largest compute core grid compared to the smallest compute core grid, a scaling of 35.5 compared to the ideal 49 is observed. The value for the compute core grid using 1600 cores (20.3) is not far from the ideal value (25). For the OEQ groups, the partial steps are individually studied later, too.

The step which destroys the performance and scaling with respect to the total runtime is the preconditioning step. The larger the grid, the higher the runtime for the preconditioning step. The runtime increases from 12 s (for the smallest compute core grid) to 92 s on the highest dimensional compute core grid used. It is composed by the reordering before the preconditioning, the application of the preconditioner, and the reordering after the preconditioning (cf. Sect. 8.3.3.3). The three partial steps are analyzed separately later, to identify where the performance is lost.

Runtime and Scaling for OEQ Group Processing The processing of the OEQ groups per iteration is divided into the following partial steps (cf. Alg. 8.4):

- Assembly of the design matrix $\mathbf{A}_o^T$ $(U \times b_{\text{obs}})$
- Computation of $\mathbf{T} := \mathbf{A}_o\mathbf{\Pi}$
- Update of $\mathbf{H}+ = \mathbf{A}_o^T\mathbf{T}$

The measured runtime and the scaling for these steps, compared to the total runtime for OEQ group processing, are shown in Fig. 8.7(a) and 8.7(b). The computation of the entries of the design matrix is performed locally on the compute cores, thus no communication is required. The scaling is nearly ideal, as for larger grids, less elements have to be computed per core. It is the fastest of all three steps and requires 16 % to 23 % of the total runtime, slightly varying for the different grids. The computation of $\mathbf{T}$ and the update of $\mathbf{H}$ show comparable characteristics. As $\mathbf{A}_o^T$ is set up, the computation of $\mathbf{H}$ is more efficient than the computation of $\mathbf{T}$, where the transposition of $\mathbf{A}_o^T$

is required in the computation. As both of the computations dominate the runtime for the OEQ group processing and they have the same runtime and scaling characteristics, these steps behave as the total runtime of the OEQ group processing with the properties as discussed above.

Runtime and Scaling for NEQ Group Processing The processing of the NEQ groups per iteration is divided into the following partial steps (cf. Sect. 8.3.3.1, second paragraph, steps 1 and 3–5 of method I):

- Reorder and adjust $\mathbf{\Pi}$
- Compute $\mathbf{H}_N = \mathbf{N}_N\mathbf{\Pi}$
- Reorder $\mathbf{H}_N$ and update $\mathbf{H}+ = \mathbf{H}_N$

The results of the analysis are shown in Fig. 8.7(c) and Fig. 8.7(d). The computation of $\mathbf{H}_N = \mathbf{N}_N\mathbf{\Pi}$ performs as expected. The runtime decreases with an increased compute core grid and becomes insignificant (below 1 s) for compute core grids with more than 256 cores (brown line). Although the scaling is not linear, the runtime remains insignificant, and the scaling is increasing. For the largest compute core grid, the computation is performed in less than 0.5 s. The operations problematic with respect to runtime and scaling are the reordering operations, which do not scale and show an increasing runtime with an increasing compute core grid. Whereas within the reordering and adjustment of $\mathbf{\Pi}$ a $U_N \times NK + OK + 1$ vector is reordered, in the reordering of $\mathbf{H}_N$ the dimension increases to $U \times NK + OK + 1$. As with an increasing dimension of the compute core grid the entries of the matrices are spread over more processes, the reordering requires more communication as it is more likely that rows stored in different local matrices have to be swapped. Except for the smallest compute core grid, the reordering operations are the dominating steps (86 % to 99 %). The performance characteristics observed within this study contradicts the results shown in Sect. 4.3.3. In contrast to the reordering performed there, where the index vector was randomly shuffled, the index vector here has a systematic structure which seems to be bad for the performance[14]. As shown later, the runtime for the reordering significantly depends on the block-cyclic distribution parameters. Another reason, why the performance is poor, is that rows in a matrix which is stored in CMO (in the local matrices) are reordered. For the reordering itself, operating on $\mathbf{\Pi}^T$ and $\mathbf{H}^T$ in the algorithm would be more efficient. Of course that will change the characteristics of the algorithm. The systematic analysis studied in this chapter identified the performance bottleneck. Especially with the results shown in Sect. 4.3.3, this characteristics were not expected. Potential for optimization remains, although an adjustment of b_r and b_c easily reduces the runtime required for reordering (for details see Sect. 8.5.2).

Runtime and Scaling for Preconditioning Step The preconditioning step consists of the following operations (cf. Sect. 8.3.3.3):

- Reorder $\mathbf{R}^{(\nu)}$ from $\mathbf{p}_{\text{bcd}}$ to $\mathbf{p}_{\oplus}$
- Apply preconditioner to $\mathbf{R}^{(\nu)}$ (cf. Sect. 8.3.1.3)
- Reorder $\mathbf{\Gamma}^{(\nu)}$ from $\mathbf{p}_{\oplus}$ to $\mathbf{p}_{\text{bcd}}$

As two reordering steps are involved in the preconditioning step, consequently the performance is bad as well (cf. Fig. 8.7(e) and 8.7(f)). Two of the three steps are reordering steps (reorder before

[14]It still has to be verified, but if the systematic reordering between the (in the simulation used) order-wise numbering $\mathbf{p}_n$ associated with the combined NEQs and $\mathbf{p}_{\text{bcd}}$ is a bad case for the reordering, it is easily possible to reorder the combined NEQs once in the beginning to a better suited numbering scheme, which can be faster reordered within the PCGMA iterations. In the current implementation, the (arbitrary) numbering scheme of the largest NEQ group is used for the combined NEQs.

the preconditioning, reordering after preconditioning), the same conclusions as for the NEQ group update step holds. The preconditioning step itself shows a poor performance on higher grids, too. As within the preconditioning step the blocks are sequentially solved, only small matrices are involved which are stored on only a few cores. Whereas the matrix $\mathbf{R}$, the preconditioner is applied to, is spread over many cores (completely block-cyclic distributed, U rows), the small to tiny diagonal blocks of the block diagonal preconditioner are only distributed over a few cores. Within the test scenario (maximal dimension 1440×1440 of a block and $b_r = b_c = 64$) the matrices are distributed maximally over the first 22×22 sub-compute core grid. Except for the smallest compute core grid, each core stores maximally a single block of dimension $b_r \times b_c$ of every block of the block diagonal preconditioner. For the PBLAS/SCALAPACK performance this is very inefficient. As currently only the simplest form of preconditioning is implemented, a huge optimization potential remains. As in the future alternative preconditioners will be studied, the preconditioning step was kept flexible. The performance decrease was not known and not expected before and was identified within this systematic study.

Conclusions for the Performance Analysis Whereas the performance of the iterative solver and scaling is good for the processing of the OEQ groups, negative effects for the processing of the NEQ groups and the preconditioning step are observed when increasing the compute core grid. Nevertheless, three facts have to be kept in mind:

I) The processing of the OEQ groups is from a computational point of view the most intensive step. Especially when increasing the resolution, the number of parameters as well as the required observations increase. The runtime of the OEQ group processing increases significantly.

II) In contrast to that, the runtime of the NEQs will not significantly change. Only the reordering gets more complex, as the parameter space increases. The consequences are empirically accessed (see Sect. 8.4.3).

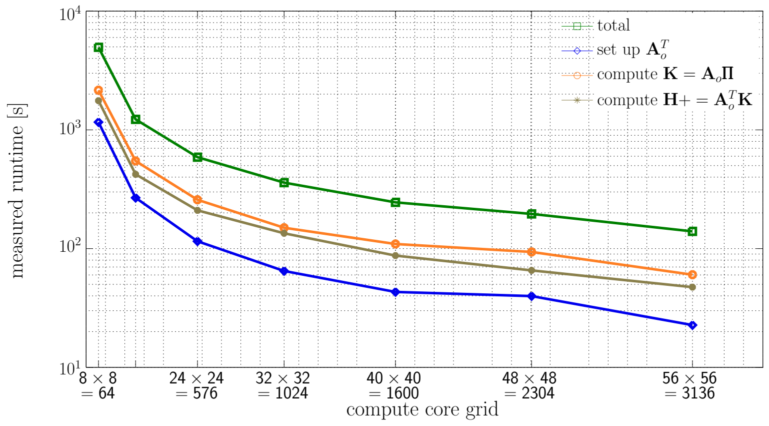
III) The preconditioning step was not optimized.

So far the easiest and most flexible implementation was chosen. In further studies the preconditioning step will be studied in more detail (not only with respect to performance but also with respect to alternative preconditioners). The systematic study introduced here identified the steps, where performance problems arose in the current implementation.

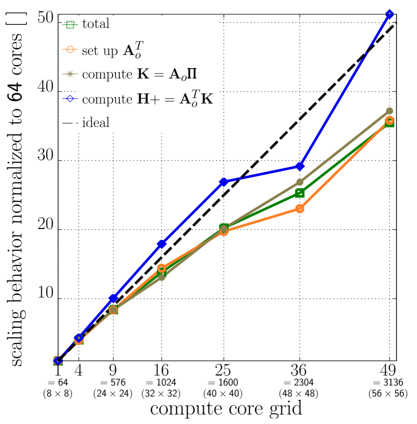
8.5.2 Dependence of the Performance on the Block-Size

The dependence of the performance of the algorithm from the chosen block-size $b_r \times b_c$ is studied within this section. The compute core grid is fixed to 40×40 and the same scenario as described above is analyzed. The block-size is varied and the performance of the individual steps is assessed. Fig. 8.8(a) shows the measured runtime for the total runtime of a single PCGMA iteration and for the individual steps (NEQ group processing, OEQ group processing and preconditioning) in dependence of the chosen block-size for $b_r = b_c$.

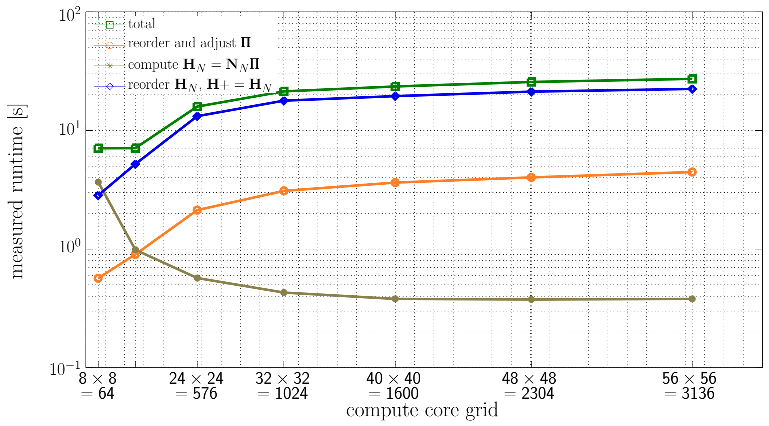
For block-sizes $b_r = b_c < 120$ the runtime is the slowest for all three partial operations and thus for the total runtime. A minimum for the total runtime is observed for $b_r = b_c = 250$. All partial steps show a small runtime for this block-size. For larger block-size, the runtime slightly increases but the variations are small. For $b_r = b_c = 250$, an iteration took 196 s compared to the case $b_r = b_c = 64$ used above in the performance study which took 282 s. The choice of $b_r = b_c = 250$ leads to an improvement by a factor of 1.4. Thus, the scaling compared to 64 cores also improves by a factor of 1.4 (assuming that there is no positive effect of the larger block-size on the smaller compute core grid) and thus gets closer to the ideal scaling. For the 40×40 compute core grid analyzed,



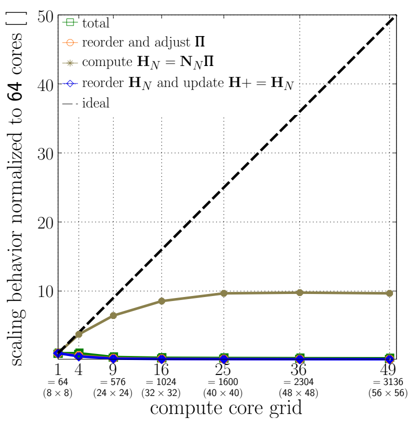
(a) Absolute runtime measured for OEQ group processing.



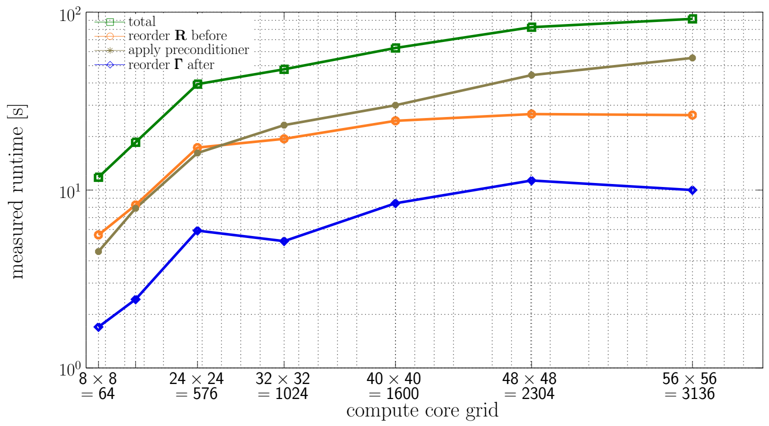
(b) Scaling behavior for OEQ group processing.



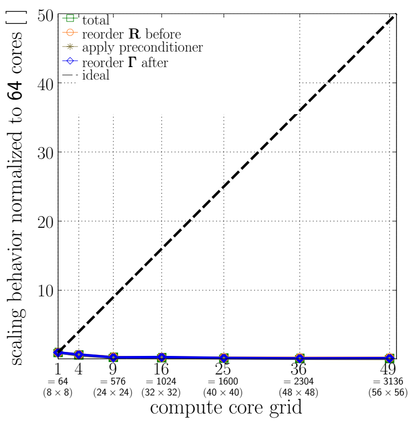
(c) Absolute runtime measured for NEQ group processing.



(d) Scaling behavior for NEQ group processing.

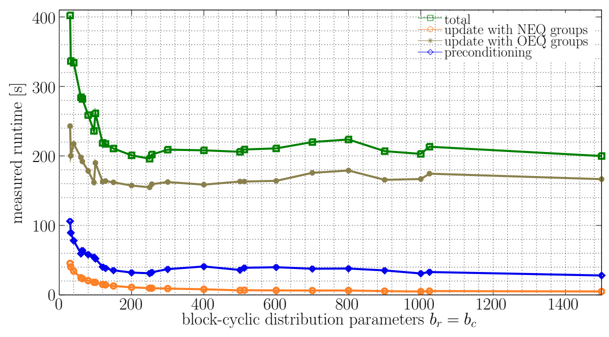


(e) Absolute runtime measured for the preconditioning step.

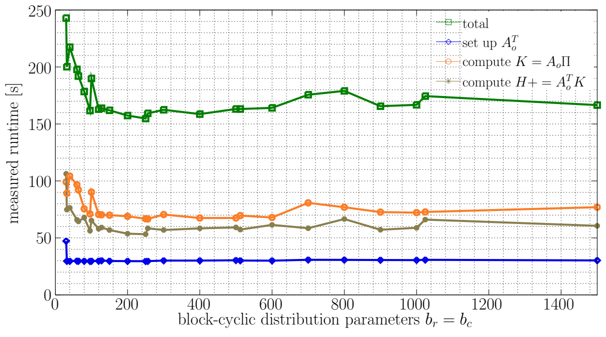


(f) Scaling behavior for the preconditioning step.

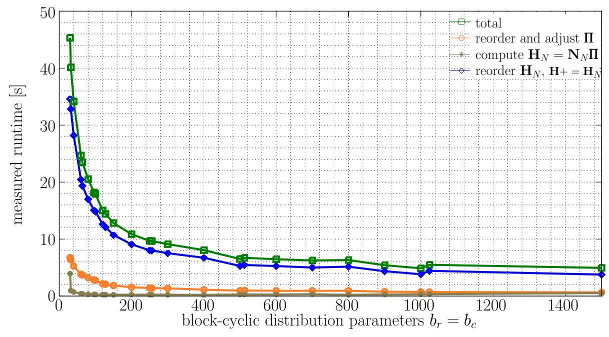
Figure 8.7: Performance and scaling of specific operations per PCGMA iteration. They are individually shown for the OEQ group processing, the NEQ group processing as well as for the preconditioning.



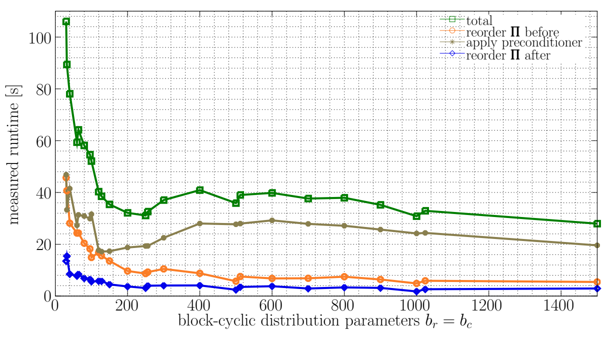
(a) Runtime measured for a single PCGMA iteration.



(b) Runtime measured for OEQ group processing.



(c) Runtime measured for NEQ group processing.



(d) Runtime measured for the preconditioning step.

Figure 8.8: Measured performance of the implemented steps of a single PCGMA iteration depending on the block-size. Results are shown for all operations involved (green) and the most intensive operations, i.e. the processing of OEQ groups (brown), the processing of NEQ groups (orange) and the preconditioning (blue). In addition, these steps are split into the operations involved in the processing.

an improvement is visible for all three individual steps. For the total iteration, the improvement is 1.4, for the NEQ groups 2.4, for the OEQ group processing 1.2 and for the preconditioning 2.1. Especially the steps which had a poor performance on the larger grids improve. It might be useful to repeat the performance analysis performed in Sect. 8.5.1.2 with tailored values for $b_r = b_c$, tuned for the different grids. As this analysis needs a lot computing time, at this stage the analysis is not performed.

Conclusions for the Performance Analysis As in Sect. 8.5.1.2, Fig. 8.8(b) to 8.8(d) breaks the three major tasks down to the individual operations and computations involved. The main conclusions are that i) the total runtime can be significantly decreased adjusting b_r and b_c, ii) the computations performed with SCALAPACK are, except for some extrema (very large or small values for b_r and b_c), not very sensitive to b_r and b_c iii) the reordering improves using larger b_r and b_c, iv) the preconditioning is faster and v) the performance for compute core grids up to 1024–1600 is acceptable for the scenario analyzed. Of course the larger b_r and b_c, the more memory is required on the first compute cores of the compute core grid. The block-cyclic distribution leads to an unbalanced distribution, as there are many small matrix blocks $\mathbf{N}_{\oplus}^{b}$, which are block-cyclic distributed over a few cores only. Indeed, for the larger compute core grids and large values for b_r and b_c, at least a block distribution results for most preconditioner blocks $\mathbf{N}_{\oplus}^{b}$. For the largest values $b_r = b_c > 1440$ the whole preconditioner is stored on the single compute core 0, 0.

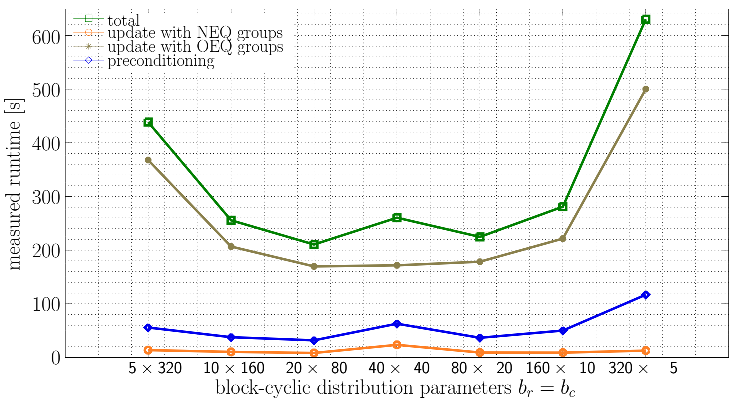


Figure 8.9: Measured performance of the implemented steps of a single PCGMA iteration depending on the shape of the compute core grid. Results are shown for all operations involved (green) and the most intensive operations, i.e. the processing of OEQ groups (brown), the processing of NEQ groups (orange) and the preconditioning (blue).

8.5.3 Shape of the Compute Core Grid

As many of the computations and operations involved operate on vector like matrices, i.e. matrices with many rows (U) but only a few right hand sides ($NK + OK + 1$), non quadratic compute core grids might yield better performance. Fig. 8.9 shows a simple experiment, fixing $\mathsf{N} = 1600$ but varying the shape of the compute core grid over rectangular grids, i.e. 5×320, 10×160, ..., 40×40 upto 320×5. The result is only shown for the overall runtime of a single PCGMA and the three partial steps as discussed above. As the shown runtime curves are representative for the partial steps, the partial steps are not shown. Except for the peak, which occurs at the quadratic grid 40×40, the runtime behaves as expected. For all three steps, the runtime significantly increases towards the extrema, i.e. 5×320 and the 320×5. It is minimal (for all three steps) at the 20×80. For the quadratic compute core grid (40×40), the runtime suddenly increases. This results from an increased runtime in the NEQ group processing and the preconditioning step. It could not be identified, if the higher runtime is significant or a random result related to the hardware or general activity in the HPC environment. To proof, that with respect to performance the close to quadratic compute core grids are better suited than the quadratic compute core grid, the simulations have to be repeated with slightly changing scenarios. This result shows that there is no hint to leave the choice of quadratic compute core grids. But, the 20×80 seems to provide a slightly better performance compared to all other grids.

8.6 Application to Real Data

The software prototype developed here was not yet applied in a real data analysis. Nevertheless, it is planned to extend this basis software for the use with real data for the joint estimation of very high degree gravity field models and the ocean's dynamic topography. Studies on the method were performed in Becker et al. (2014a, 2013) and Becker et al. (2014b). Although main parts were already implemented within this developed software package, they are not discussed within this thesis, as this thesis focuses on the development of the basic concepts and studies the potential of this kind of massive parallel implementation of the solver. Thus, real data specific problems and details are faded out.

9. Summary, Conclusions and Outlook

9.1 Summary and Conclusions

Due to the use of autonomous sensors to collect geodetic data sets, the data sets to be analyzed get more and more complex and the computational requirements to derive an in a computational sense rigorous solution, significantly increase. Especially sensors observing the global System Earth — e.g. carried on satellite platforms — deliver huge data sets, as they either have a high measurement frequency, deliver observation time series over decades or even both. Thus refined higher resolution models, either in space and/or in time, can be adjusted to the measurements resulting in a higher dimensional parameter space of the models. This often results in an inverse model and an overdetermined system of equations has to be solved with respect to the unknown model parameters. In addition to the increasing number of observations (within this thesis: hundreds of thousands to hundreds of millions) and the increasing number of unknown model parameters (within this thesis: tens of thousands to millions) the observations are often highly correlated and require a complex stochastic model for a proper use within a least squares adjustment with complementary observation types.

Within this thesis, concepts, methods and standards from high performance computing were used to develop and implement, in a numerical sense, rigorous solvers for high and huge dimensional least squares adjustment problems. Three representative problems from global gravity field determination (spherical harmonic analysis) were chosen to demonstrate the use of this high performance computing concepts for the analysis of geodetic data sets. Although some physical approximations were included in the simulation scenarios shown, the potential and possibilities resulting from the use of HPC concepts could be demonstrated. After a review of the existing standards and concepts, a special flexible framework for the high dimensional matrix computations was developed as a C++ class. An easy to use interface was developed to handle and manage block-cyclic distributed matrices, i.e. matrices which are stored in the distributed memory of the nodes of a compute cluster. An interface to the standard high performance computing linear algebra libraries (PBLAS and SCALAPACK) is provided to perform standard computations. The routines required for computations were extended with special features required within the solution of least squares adjustment problems. For instance, reordering of matrix rows and/or columns to account for different numbering schemes and/or parameter spaces, treatment of huge dense covariance matrices and variance component estimation was integrated.

The concepts and the basic framework developed was used in three applications from theoretical and physical geodesy, i.e., determination of the global Earth's gravity field parameterized as a spherical harmonic series. Besides a general summary of the mathematical, statistical and physical concepts required, the implementations derived for the three applications, were introduced, discussed and assessed in detail (e.g. with respect to performance).

The first of those applications was the determination of the Earth's global gravity field from the observations of the GOCE mission. The computational challenge here was the huge number of observations ($4.4 \cdot 10^8$), which are in addition highly correlated. The number of unknown parameters was moderate (60 000-80 000). The assembly of full normal equations from SGG observations was discussed, including the setup of the observation equations as block-cyclic distributed matrices, decorrelation of the observation equations by digital decorrelation filters (cascaded autoregressive moving average filters) and the computation of the normal equations (product $\mathbf{A}^T\mathbf{A}$). The massive parallel implementation for the application was developed step by step. It was shown how the basic framework can be used and tailored to a specific problem. The developed software package was used for real data analysis as a part of the official ESA data analysis group (HPF). The models and the progress along the different releases were discussed. Finally, the performance of the software on a high performance computing compute cluster was analyzed in detail. It was successfully used with 64 to 1600 compute cores with a linear and close to ideal scaling behavior.

The second application covered the high degree estimation of combined global gravity field models from complementary data sets. This data sets can be either available as band-limited normal equations or as high resolution raw observations for which the normal equations have to be computed from the observation equations. Within this thesis, the high resolution assembly of normal equations from normal and observation equation groups was implemented in a massive parallel high performance computing environment. It was extended with variance component estimation to estimate relative weights from the data. It was successfully used to set up 2 TB normal equations (520 000 unknowns) from $4 \cdot 10^6$ arbitrary distributed inhomogeneous observations and additional normal equation groups. The assembly and solution was studied with respect to performance. For the derived implementation, a close to optimal linear scaling could be observed for compute core grids using 1600 to 7744 compute cores. A further use of the derived full normal equations was demonstrated.

As a final application, even higher resolution models were made rigorously computable, i.e. without computationally motivated approximations. A prototype of a massive parallel iterative conjugate gradient based solver was implemented using the standard concepts and basic framework introduced. Applying the iterative solver, it was demonstrated, that even least squares solutions for $2 \cdot 10^6$ unknown parameters from $20 \cdot 10^6$ observations and some preprocessed normal equation groups become rigorously computable. A preconditioning and variance component estimation was included in the iterative solver as well. The performance of the prototype of the iterative solver is systematically analyzed and some steps, which show the potential for optimization, were identified. The solver was successfully used with 64 to 4096 compute cores. The parts for the processing of the high resolution observation equations show a linear adequate scaling. Some unexpected performance problems on larger compute core grids remain (related to reordering and reconditioning). Nevertheless, the studied demonstrated the flexibility of the framework and of the solver. Least squares problems compromising 520 000 unknown parameters were solvable in a reasonable time with compute clusters with only 64 compute cores. Even on such small compute clusters, a single PCGMA iteration for the analyzed scenario took less than one hour. Using the solver on compute core grids with 25 times more compute cores ($\mathsf{N} = 1\,600$), reduces the runtime by a factor of 20.

Within this thesis a massive parallel software package for high dimensional adjustment problems (with special focus on spherical harmonic analysis) was developed to make problems solvable, which were not rigorously computable before. Besides the general framework and the basic functionality for adjustment procedures, specific and optimized modules were developed for applications from gravity field determination. It was shown in detail, how this special extension can be implemented using the developed basic framework. The design was kept flexible to easily introduce/adapt alternative observation types or adjustment procedures. The main parts of the solver can be used on only a small compute core grid (e.g. with 16 cores) or on huge compute core grids which make use of ten thousands of compute cores. In addition, a specific adaption to any other geodetic application, which requires the solution of high dimensional adjustment problem is possible and the software was prepared for that.

This thesis demonstrates the value of high performance computing concepts for geodetic applications. Historical established approximations and simplifications, which are still widely used in many studies, often become decrepit, using the concepts of high performance computing. Avoiding approximations and simplifications has a positive effect for the solution itself as well as the corresponding error estimate in form of the covariance matrix, although the positive effect can not be quantified. The basics towards an extension of the observation equations was made (increased parameter space), such that many tasks can be solved in a joint adjustment. Thus multi step approaches can be replaced by a joint inversion. Although a lot of effort has to be spent on the fusion of high performance computing concepts with established algorithms and the knowledge of the data handling, this thesis demonstrates that, if the basics are implemented and understood, their use is straightforward.

9.2 Outlook

This thesis started to systematically map high to huge dimensional adjustment problems to massive parallel high performance compute clusters with a flexible design. The derived and implemented basics open a wide range of potential further studies (numerical as well as methodical) and a huge range of further possible application scenarios (not only related to gravity field determination). So far — as shown by the three applications — the developed concepts and the basic framework was only used within applications related to global gravity field determination. Nevertheless, the chosen design supports to extend the range of applications to all tasks where linear algebra and operations from adjustment theory are involved. Depending on research themes and cooperations of the Theoretical Geodesy group at Institute of Geodesy and Geoinformation at the University of Bonn, the software package will include alternative applications. For instance, for current activities within INSAR (Interferometric Synthetic Aperture Radar) data analysis, operations are performed on a huge data volume, i.e. stacks of INSAR scenes over decades, where the use of high performance computing might be required if the final models are developed.

Furthermore, the solvers can be easily extended to alternative adjustment procedures and estimators, like robust estimators. As they are typically solved iteratively, they require a lot computational resources. The integration of robust estimators into the massive parallel environment is the next logical step.

Of course, the presented applications will be further studied and extended, too. For instance, with release 5 of the GOCE real data models, the official processing of GOCE gravity field models is finalized. Nevertheless, it will make sense to reprocess the whole time series to analyze all parts of the processing chain. It is very likely that, revising the whole data set again, more information and signal can be extracted out of the data. This might for instance be a robust estimation of the decorrelation filters, a more consistent way of identifying outliers (probably coupled with the robust filter estimation), a parameterization of changing filters in time or a possible reprocessing of the level-1B input data. For that purpose the developed software as presented here, will be used for a further reprocessing of the whole data set collected during the GOCE mission. Of course, the software will be extended and/or adapted if, for instance, the filter models and their design will change. In addition, different further studies on the numerical behavior will be performed (condition and data distribution, performance of preconditioners within GOCE data analysis, alternative regularization matrices, etc.).

The direct solver presented is currently extended to handle alternative observation equations and other parameter types. From along track altimetry observations (real data), the geoid and the ocean's dynamic topography can be estimated in a joint least squares adjustment. For that reason, the parameter space and the observation equations are extended with coefficients which can parameterize the ocean's dynamic topography (finite elements). Within this context and for alternative observation equations, the inclusion of correlations in the observations will be necessary (as e.g. for GOCE observations). For the altimetry observations, existing correlations along the tracks are modeled via empirically estimated covariance functions. A decorrelation of the observation equations, based on the covariance functions, has to be implemented (instead of a diagonal matrix as used so far) and efficiently integrated into the solver. Additional components of the solver will be required, e.g. for parameter elimination of group specific parameters or to derive the covariance matrix of the finite element parameters only. Again, the solver itself can be — if required – extended and adapted to any other adjustment problem or analysis of geodetic data sets.

The same extensions as for the direct solver are possible and planed for the iterative solver. However, as demonstrated in this thesis, some open issues remain for the gravity field related application. The performance on higher compute core grids is still improvable, especially for the processing of normal equation groups and for the preconditioning step. Whereas the method for preconditioning currently implemented is most simple and thus flexible, faster implementations with a better load balancing have to be studied for the same preconditioner models. Furthermore, the preconditioner models themselves have to be further explored for the estimation of spherical harmonic parameters as well as for an estimation scenario which includes additional or completely different types of parameters. The block diagonal preconditioners implemented so far can only be used for spherical harmonic coefficients, as their properties were used in the preconditioner design. The software developed in this thesis provides the basis to generally study properties like convergence of alternative preconditioners in the solver as well as the condition of the system of equations. The current study and implementation should be seen as a starting point for a wide range of methodical studies and applications. For all applications shown, of course, if global high resolution real data sets will become available, the software is prepared for an analysis of these data sets. At least over the ocean, these data sets are available as along-track altimetry observations.

It is expected that in all research areas the requirement of high performance computing will significantly increase over the next years and massive parallel implementations and software package will be requested for the analysis of data sets of very different kinds. With the basics performed, the background to solve typical numerical tasks from theoretical geodesy (Monte Carlo simulations, ensemble generation, collocation, adjustment problems of different kind, decorrelation), which require a lot of computing power in the HPC environment, is created.

Acknowledgements

Parts of the study and the publication were founded within the DFG project G/O2000+ (SCHU2305/3-1). This work was financially supported by ESA GOCE HPF (main contract No. 18308/04/NL/MM) and by the BMBF Geotechnologien program REAL-GOCE. Most of the computations were performed on the JUROPA supercomputer at FZ Jülich. The computing time was granted by the John von Neumann Institute for Computing (project HBN15). Some computations were performed on the HPC compute clusters at the IT center of the RWTH Aachen. Access was granted via the Ressourcenverbund NRW. Some computations were performed on the new cluster at the University of Bonn financed via a DFG Forschungsgroßgeräteantrag (INST 217/747-1 FUGG).

The Open Source HPC computing libraries ATLAS, BLAS, LAPACK, OpenMPI, BLACS, PBLAS and ScaLAPACK are gratefully acknowledged. Similarly, the Open Source software packages GMT (Generic Mapping Tools, Wessel and Smith, 1998, 2004) and OCTAVE (Eaton et al., 2009, Octave community, 2014) are gratefully acknowledged.

The joint work within the GOCO consortium (`http://www.goco.eu`) is gratefully acknowledged. The ITG-Grace2010s NEQs are provided online (`http://www.igg.uni-bonn.de/apmg/index.php?id=itg-grace2010`). Special thanks go to the colleagues from TU Graz (Norbert Zehentner, Eduard Höck, Reinhard Mayerhofer, Helmut Goiginger and Torsten Mayer-Gürr) and from TU München (Roland Pail) for the joint work on the GOCE time-wise models within the HPF (WP6000). In addition to the same colleagues from TU GRAZ for the provision of GOCE SST and CHAMP NEQs and to colleagues from the IWF Graz (A. Maier and O. Baur) for the provision of the SLR NEQs. Contributions (data preparation, distribution and preprocessing) by all members of the GOCE HPF are gratefully acknowledged.

The thesis is the result of my studies within the Department of Theoretical Geodesy within the Institute of Geodesy and Geoinformation at the University of Bonn. I would like to acknowledge the support by and the discussions with my supervisor, Prof. Dr.-techn. Wolf Dieter Schuh. In addition I would like to thank the second and third referee, Prof. Dr.Ing. Jürgen Kusche and Prof. Dr. Carsten Burstedde for their discussions, comments and effort reviewing the thesis. The comments and discussions with Prof. Dr.-Ing. Karl-Rudolf Koch helped to improve the thesis. The discussions (not always related to science) with all members and former members of the Theoretical Geodesy group as well as the Astronomical, Physical and Mathematical Geodesy group helped a lot during the preparation of the thesis and in the time at the university and created a nice working environment.

A. Symbols

General

$\tilde{\mathbf{x}}$	estimates for the parameters $\mathbf{x}$
$^{(\nu)}$	iteration index for iteration ν
p	vector containing a symbolic numbering scheme

Random values

$\boldsymbol{\mathcal{A}}$	vector of random variables
$\boldsymbol{E}\{\boldsymbol{\mathcal{A}}\}$,	expectation of a random vector
$\boldsymbol{\Sigma}\{\boldsymbol{\mathcal{A}}\} = \boldsymbol{\Sigma}_{\boldsymbol{\mathcal{A}}\boldsymbol{\mathcal{A}}}$	covariance matrix of a random vector

Mathematical and Algebraic operations

$\text{trace}(\mathbf{A})$	trace of the matrix $\mathbf{A}$
$\text{chol}(\mathbf{N})$	Cholesky decomposition of the positive definite matrix $\mathbf{N} = \mathbf{R}^T\mathbf{R}$
$\text{select}(\mathbf{A})$	sparse matrix which contains selected entries from $\mathbf{A}$ (zero otherwise).
$\mathbf{x} = \text{solve}(\mathbf{N}, \mathbf{n})$	efficient solution of the system of equations $\mathbf{Nx} = \mathbf{n}$ with a proper algorithm (depending on properties of $\mathbf{N}$).
$\div$	integer division
$\%$	modulo operation
$\boldsymbol{\Psi}^{r,c}_{\mathsf{p}_\mathsf{f} \mapsto \mathsf{p}_\mathsf{t}}(\,\cdot\,)$	permutation operator permuting rows (r) and/or columns (c)
$\lceil \cdot \rceil$	round towards plus infinity

Informatics and HPC related symbols

$\&a$	address operator, determines the address of the variable a in the computers memory
N	number of MPI processes used
n	MPI rank of a process
$(\mathsf{R} \times \mathsf{C})$	dimension of a compute core grid
$(\mathsf{r},\, \mathsf{c})$	process coordinates in a two-dimensional compute core grid

Matrix, vector and block-cyclic distributed matrix related notation

a, α, A	scalar values,
$\mathbf{a}$	vector / one-dimensional field/array
$\mathbf{A}$	general matrix / two-dimensional field
$\mathbf{A}(r_1 : r_2, c_1 : c_2)$	sub-matrix of matrix $\mathbf{A}$
$\mathbf{A}^l_{\mathsf{r},\mathsf{c}}$	locale part of block-cyclic distributed matrix $\mathbf{A}$ stored on process $(\mathsf{r},\, \mathsf{c})$
$\mathbf{a}(i) = a_i$	vector/field entry at position i
$\mathbf{A}(r,c)$, $\mathbf{A}_{r,c}$, $a_{r,c}$	matrix entry at position $(r,\ c)$
$b_r \times b_c$	dimension of the sub-blocks for block-cyclic distribution
$i(r,\ c)$	index of the matrix entry $(r,\ c)$ in the one-dimensional field the matrix is stored in
l_d	leading dimension of the matrix storage scheme (RMO or CMO)
i_c	increment of the matrix storage scheme (RMO or CMO)
$(r,\ c)$	coordinates of a matrix entry
$R \times C$	dimension of a matrix
$(r^l_{\mathsf{r},\mathsf{c}},\ c^l_{\mathsf{r},\mathsf{c}})$	coordinates of a matrix entry referring to the local matrix on process $(\mathsf{r},\, \mathsf{c})$
$(R^l_{\mathsf{r},\mathsf{c}} \times C^l_{\mathsf{r},\mathsf{c}})$	dimension of the local matrix of a block-cyclic distributed matrix referring to process $(\mathsf{r},\, \mathsf{c})$

B. Abbreviations

AntGP	Antarctic Geoid Project
ArcGP	Arctic Gravity Project
ARMA	Auto Regressive Moving Average filter
ATLAS	Automatically Tuned Linear Algebra Software
B	Byte
BLACS	Basic Linear Algebra Communication Subprograms
BLAS	Basic Linear Algebra Subprograms
BMBF	Bundesministerium für Bildung und Forschung (Federal Ministry of Education and Research)
CHAMP	CHAllenging Mini-satellite Payload
CMO	Column Major Order
CPU	Central Processing Unit
d/o	spherical harmonic Degree and Order
EFRF	Earth Fixed Reference Frame
EGM96	Earth Geopotential Model 1996
EGM2008	Earth Geopotential Model 2008
EGM_TIM	Earth Gravitational Model from GOCE using the TIMe-wise method
EIGEN	European Improved Gravity model of the Earth by New techniques
ESA	European Space Agency
GRACE	Gravity Recovery And Climate Experiment
GRAIL	Gravity Recovery And Interior Laboratory
GRF	Gradiometer fixed Reference Frame
GRS80	Geodetic Reference System 1980
GOCE	Gravity field and steady-state Ocean Circulation Explorer
GOCE-HPF	GOCE High-level Processing Facility
GOCO	Gravity Observation Combination Consortium
GPS	Global Positioning System
HPC	High-Performance Computing
HPF	(GOCE) High-Level Processing Facility
I/O	Input and Output (file reading and writing)
IGG	Institute of Geodesy and Geoinformation
IRF	Inertial Reference Frame
ITRF	International Terrestrial Reference Frame
JUROPA	Jülich Research On Petaflop Architectures
LAPACK	Linear Algebra PACKage
LNOF	Local North Oriented Cartesian Frame
MBW	Measurement Band Width
MC	Monte Carlo
MDT	Mean Dynamic Topography
MPI	Message Passing Interface
NEQ	Normal EQuation
NFS	Network File System
OEQ	Observation EQuation
PBLAS	Parallel Basic Linear Algebra Subprograms
PCG	Preconditioned Conjugate Gradient
PCGMA	Preconditioned Conjugate Gradient Multiple Adjustment
PSD	Power Spectral Density
PVM	Parallel Virtual Machine
REAL-GOCE	REAL data analysis GOCE
REG	REGularaization group
RHS	Right Hand Side
RMO	Row Major Order
RMS	Root Mean Square error
SC	Scientific Computing
SCALAPACK	SCAlable Linear Algebra PACKage
SGG	Satellite Gravity Gradiometry
SLR	Satellite Laser Ranging
SST	high-low Satellite-to-Satellite Tracking
STL	Standard Template Library
VC	Variance Component
VCE	Variance Component Estimation

List of Figures

List of Tables

List of Algorithms

Listings

Bibliography

H. Alkhatib. *On Monte Carlo methods with applications to the current satellite gravity missions.* PhD thesis, Institute of Geodesy and Geoinformation, University of Bonn, Bonn, Germany, 2007. URL `http://nbn-resolving.de/urn:nbn:de:hbz:5N-10783`.

H. Alkhatib and W.-D. Schuh. Integration of the Monte Carlo covariance estimation strategy into tailored solution procedures for large-scaled least squares problems. *Journal of Geodesy*, 70:53–66, 2007. doi:10.1007/s00190-006-0034-z.

O. B. Andersen and P. Knudsen. DNSC08 mean sea surface and mean dynamic topography models. *Journal of Geophysical Research: Oceans*, 114(C11), 2009. doi:10.1029/2008JC005179.

E. Anderson, Z. Bai, C. Bischof, J. W. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. C. Sorensen. LAPACK: A portable linear algebra library for high-performance computers. Technical Report 20, LAPACK Working Note, 1990. URL `http://www.netlib.org/lapack/lawnspdf/lawn20.pdf`.

E. Anderson, Z. Bai C. Bischof, J. Demel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammerling, A. McKenney, and D. Sorensen. *LAPACK Users Guide.* SIAM, Society for Industrial and Applied Mathematics, Philadelphia, 3rd edition, 1999. URL `http://www.netlib.org/lapack/lug/index.html`.

Y. Aoyama and J. Nakano. RS/6000 SP: Practical MPI programming. Technical Report SG24-5380-00, IBM, International Technical Support Organization, Austin, Texas, 1999. URL `http://www.redbooks.ibm.com/`.

M. Baboulin, L. Giraud, S. Gratton, and J. Langou. Parallel tools for solving incremental dense least squares problems: application to space geodesy. *Journal of Algorithms and Computational Technology*, 19(1):117–133, 2009. doi:10.1260/174830109787186541.

P. Balaji, W. Bland, J. Dinan, D. Goodell, W. Gropp, R. Latham, A. Pena, and R. Thakur. *MPICH User's Guide.* Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, 3.0.4 edition, 2013. URL `http://www.mpich.org`.

H. Bauke and S. Mertens. *Cluster Computing.* Springer, Berlin Heidelberg, 2006. ISBN 978-3540422990.

O. Baur. Tailored least-squares solvers implementation for high-performance gravity field research. *Computers & Geosciences*, 35(3):548 – 556, 2009. doi:10.1016/j.cageo.2008.09.004.

O. Baur, G. Austen, and J. Kusche. Efficient GOCE satellite gravity field recovery based on least-squares using QR decomposition. *Journal of Geodesy*, 82(4-5):207–221, 2008. doi:10.1007/s00190-007-0171-z.

O. Baur, H. Bock, E. Höck, A. Jäggi, S. Krauss, T. Mayer-Gürr, T. Reubelt, C. Siemes, and N. Zehentner. Comparison of GOCE-GPS gravity fields derived by different approaches. *Journal of Geodesy*, pages 1–15, 2014. doi:10.1007/s00190-014-0736-6.

S. Becker. *Konsistente Kombination von Schwerefeld, Altimetrie und hydrographischen Daten zur Modellierung der dynamischen Ozeantopographie.* PhD thesis, Institute of Geodesy and Geoinformation, University of Bonn, Bonn, Germany, 2012. URL `http://nbn-resolving.de/urn:nbn:de:hbz:5n-29199`.

S. Becker, G. Freiwald, M. Losch, and W.-D. Schuh. Rigorous fusion of gravity field, altimetry and stationary ocean models. *Journal of Geodynamics*, 59-60(0):99–110, 2012. doi:10.1016/j.jog.2011.07.006.

S. Becker, M. Losch, J. M. Brockmann, G. Freiwald, and W.-D. Schuh. A tailored computation of the mean dynamic topography for a consistent integration into ocean circulation models. *Surveys in Geophysics*, pages 1–19, 2013. doi:10.1007/s10712-013-9272-9.

S. Becker, J. M. Brockmann, and W.-D. Schuh. Consistent combination of gravity field, altimetry and hydrographic data. In U. Marti, editor, *Procceedings of the International Symposium on Gravity, Geoid and Height Systems (GGHS2012)*, volume 141, page accepted. IAG Symposia, Springer Berlin Heidelberg, 2014a. ISBN 978-3-319-10836-0.

S. Becker, J. M. Brockmann, and W.-D. Schuh. Mean dynamic topography estimates purely based on GOCE gravity field models and altimetry. *Geophysical Research Letters*, 41(6):2063–2069, 2014b. doi:10.1002/2014GL059510.

G. Beutler, A. Jäggi, L. Mervart, and U. Meyer. The celestial mechanics approach: application to data of the GRACE mission. *Journal of Geodesy*, 84(11):661–681, 2010. doi:10.1007/s00190-010-0402-6.

R. J. Bingham, P. Knudsen, O. B. Andersen, and R. Pail. An initial estimate of the north atlantic steady-state geostrophic circulation from GOCE. *Geophysical Research Letters*, 38(1), 2011. doi:10.1029/2010GL045633.

Å. Björck. *Numerical Methods for Least Squares Problems.* SIAM, Society for Industrial and Applied Mathematics, Philadelphia, 1998.

L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPCK Users Guide*. SIAM, Society for Industrial and Applied Mathematics, Philadelphia, 2nd edition, 1997. URL `http://www.netlib.org/scalapack/slug/index.html`.

H. Bock, A. Jäggi, U. Meyer, P. N. A. M. Visser, J. van den IJssel, T. Helleputte, M. Heinze, and U. Hugentobler. GPS-derived orbits for the GOCE satellite. *Journal of Geodesy*, 85(11):807–818, 2011. doi:10.1007/s00190-011-0484-9.

H. Bock, A. Jäggi, G. Beutler, and U. Meyer. GOCE: precise orbit determination for the entire mission. *Journal of Geodesy*, pages 1–14, 2014. doi:10.1007/s00190-014-0742-8.

Boost. Boost C++ libraries. online, 2013a. URL `www.boost.org`. last accessed 2014-09-07.

Boost. Boost 1.55.0 library documentation. online, 2013b. URL `http://www.boost.org/doc/libs/1_55_0/`. last accessed 2014-09-08.

C. Boxhammer. *Effiziente numerische Verfahren zur sphärischen harmonischen Analyse von Satellitendaten*. PhD thesis, Institute of Geodesy and Geoinformation, University of Bonn, Bonn, Germany, 2006. URL `http://nbn-resolving.de/urn:nbn:de:hbz:5N-07990`.

C. Boxhammer and W.-D. Schuh. GOCE gravity field modeling: computational aspects - free kite numbering scheme. In J. Flury, R. Rummel, C. Reigber, M. Rothacher, G. Boedecker, and U. Schreiber, editors, *Observation of the Earth System from Space*, pages 209–224. Springer Berlin Heidelberg, Berlin - Heidelberg, 2006. doi:10.1007/3-540-29522-4_15.

J. M. Brockmann and W.-D. Schuh. Fast variance component estimation in GOCE data processing. In S. P. Mertikas, editor, *Gravity, Geoid and Earth Observation*, volume 135 of *International Association of Geodesy Symposia*, pages 185–193. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-10634-7. doi:10.1007/978-3-642-10634-7_25.

J. M. Brockmann and W.-D. Schuh. Characteristics of different preconditioners used in GOCE gravity field determination applying iterative solvers. In *Paper presented at Geodätische Woche*, Nürnberg, Germany, 27.09.2011 2011. URL `http://www.uni-stuttgart.de/gi/research/Geodaetische_Woche/2011/Session6/S6-09-Brockmann.pdf`.

J. M. Brockmann, B. Kargoll, I. Krasbutter, W.-D. Schuh, and M. Wermuth. GOCE data analysis: From calibrated measurements to the global earth gravity field. In F. Flechtner, T. Gruber, A. Güntner, M. Mandea, M. Rothacher, T. Schöne, and J. Wickert, editors, *System Earth via Geodetic-Geophysical Space Techniques*, Advanced Technologies in Earth Sciences, pages 213–229. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-10228-8_17.

J. M. Brockmann, E. Höck, I. Krasbutter, T. Mayer-Gürr, R. Pail, W.-D. Schuh, and N. Zehentner. Performance of the fourth generation GOCE time-wise earth gravity field model. In *Geophysical Reasearch Abstracts*, volume 15, Vienna, Austria, 2013. URL `http://presentations.copernicus.org/EGU2013-9401_presentation.pdf`.

J. M. Brockmann, and E. Höck N. Zehentner, R. Pail, I. Loth, T. Mayer-Gürr, and W.-D. Schuh. EGM_TIM_RL05: An independent geoid with centimeter accuracy purely based on the GOCE mission. *Geophysical Research Letters*, accepted, 2014a.

J. M. Brockmann, L. Roese-Koerner, and W.-D. Schuh. Use of high performance computing for the rigorous estimation of very high degree spherical harmonic gravity field models. In U. Marti, editor, *Procceedings of the International Symposium on Gravity, Geoid and Height Systems (GGHS2012)*, volume 141 of *International Association of Geodesy Symposia*. Springer Berlin Heidelberg, 2014b. ISBN 978-3-319-10836-0. accepted.

J. M. Brockmann, L. Roese-Koerner, and W.-D. Schuh. A concept for the estimation of high-degree gravity field models in a high performance computing environment. *Studia Geophysica et Geodaetica*, 58:571–594, 2014c. doi:10.1007/s11200-013-1246-3.

S. L. Bruinsma, C. Förste, O. Abrikosov, J.-C. Marty, M.-H. Rio, S. Mulet, and S. Bonvalot. The new ESA satellite-only gravity field model via the direct approach. *Geophysical Research Letters*, 40(14):3607–3612, 2013. doi:10.1002/grl.50716.

C. Cesare and G. Catastini. Gradiometer on-orbit calibration procedure analysis. Technical Report GO-TN-AI-0069, Alenia Aerospazio, 2008. URL `https://earth.esa.int/c/document_library/get_file?folderId=14168&name=DLFE-777.pdf`.

J. Choi, J. J. Dongarra, R. Pozo, and D. W. Walker. ScaLAPACK: A scalable linear algebra for distributed memory concurrent computers. Technical Report 55, LAPACK Working Note, November 1992. URL `http://www.netlib.org/lapack/lawnspdf/lawn55.pdf`.

J. Choi, J. W. Demmel, I. S. Dhillon, J. J. Dongarra, L. S. Ostrouchov, A. P. Petitet, K. Stanley, D. W. Walker, and R. C. Whaley. ScaLAPACK: A portable linear algebra library for distributed memory computers - design issues and performance. Technical Report 95, LAPACK Working Note, March 1995a. URL `http://www.netlib.org/lapack/lawnspdf/lawn95.pdf`.

J. Choi, J. J. Dongarra, L. S. Ostrouchov, A. P. Petitet, D. W. Walker, and R. C. Whaley. A proposal for a set of parallel basic linear algebra subprograms. Technical Report 100, LAPACK Working Note, May 1995b. URL `http://www.netlib.org/lapack/lawnspdf/lawn100.pdf`.

O. L. Colombo. *Numerical methods for harmonic analysis on the sphere.* Number No. 310 in Reports of the Department of Geodetic Science. Ohio State University (OSU), Ohio, 1981.

P. Ditmar, R. Klees, and F. Kostenko. Fast and accurate computation of spherical harmonic coefficients from satellite gravity gradiometry data. *Journal of Geodesy*, 76:690–705, 2003a. doi:10.1007/s00190-002-0298-x.

P. Ditmar, P. N. A. M. Visser, and R. Klees. On the joint inversion of SGG and SST data from the GOCE mission. *Advances in Geosciences*, 1:87–94, 2003b. doi:10.5194/adgeo-1-87-2003.

J. J. Dongarra and R. C. Whaley. A user's guide to the BLACS v1.1. Technical Report 94, LAPACK Working Note, 1997. URL `http://www.netlib.org/lapack/lawnspdf/lawn94.pdf`. originally released 1995.

J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 14(1):1–17, 1988. doi:10.1145/42288.42291.

J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, March 1990a. doi:10.1145/77626.79170.

J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. Van Der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers.* SIAM, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1990b. ISBN 089871270X.

J. J. Dongarra, A. Lumsdaine, R. Pozo, and K. Remington. A sparse matrix library in C++ for high performance architectures. In *Proceedings of the Second Object Oriented Numerics Conference*, pages 214–218, 1994. URL `http://math.nist.gov/sparselib++/`.

K. Dowd and C. R. Severance. *High performance computing - RISC architectures, optimization and benchmarks.* O'Reilly, 2nd edition, 1998. ISBN 978-1-56592-312-6.

J. W. Eaton, D. Bateman, and S. Hauberg. *GNU Octave version 3.0.1 manual: a high-level interactive language for numerical computations.* CreateSpace Independent Publishing Platform, 2009. URL `http://www.gnu.org/software/octave/doc/interpreter`. ISBN 1441413006.

EGG-C. Product specification for level-2 products and auxiliary data products. GO-ID-HPF-GS-0041, 2009. URL `http://www.earth.esa.int/GOCE/`.

EGG-C. GOCE standards 3.2. GO-TN-HPF-GS-011, 2010a. URL `http://www.earth.esa.int/GOCE/`.

EGG-C. GOCE level 2 product data handbook 4.3. GO-MA-HPF-GS-0110, 2010b. URL `http://www.earth.esa.int/GOCE/`.

V. L. Eijkhout. Overview of iterative linear system solver packages. Technical Report 141, LAPACK Working Note, December 1998. URL `http://www.netlib.org/lapack/lawnspdf/lawn141.pdf`.

ESA. *The four candidate earth explorer core missions – Gravity field and steady-state Ocean Circulation*, volume 1233 of *ESA SP*. ESA Publications Division, Noordwijk, Netherlands, 1999.

H. H. Farahani, P. Ditmar, R. Klees, X. Liu, Q. Zhao, and J. Guo. The static gravity field model DGM-1S from GRACE and GOCE data: computation, validation and an analysis of GOCE mission's added value. *Journal of Geodesy*, 87(9):843–867, 2013. doi:10.1007/s00190-013-0650-3.

T. Fecher, R. Pail, and T. Gruber. Global gravity field determination by combining GOCE and complementary data. In L. Ouwehand, editor, *Proceedings of the 4th International GOCE User Workshop, ESA Publication SP-696*. ESA/ESTEC, 07 2011. URL `http://www.spacebooks-online.com/product_info.php?cPath=104&products_id=17254`.

R. Floberghagen, M. Fehringer, D. Lamarre, D. Muzi, B. Frommknecht, C. Steiger, J. Pineiro, and A. da Costa. Mission design, operation and exploitation of the gravity field and steady-state ocean circulation explorer mission. *Journal of Geodesy*, 85(11):749–758, 2011. doi:10.1007/s00190-011-0498-3.

R. Forsberg and S. Kenyon. Artic gravity project. Technical report, National Survey and Cadastre, 2000. URL `http://earth-info.nga.mil/GandG/wgs84/agp/abstract.doc`.

C. Förste, R. Schmidt, R. Stubenvoll, F. Flechtner, U. Meyer, R. König, H. Neumayer, R. Biancale, J.-M. Lemoine, S. L. Bruinsma, S. Loyer, F. Barthelmes, and S. Esselborn. The GeoForschungsZentrum Potsdam/Groupe de Recherche de Gèodésie Spatiale satellite-only and combined gravity field models: EIGEN-GL04S1 and EIGEN-GL04C. *Journal of Geodesy*, 82(6):331–346, 2008. doi:10.1007/s00190-007-0183-8.

C. Förste, S. L. Bruinsma, R. Shako, J.-C. Marty, F.Flechtner, O. Abrikosov, C. Dahle, J.-M. Lemoine, H. Neumayer, R. Biancale, F. Barthelmes, R. König, and G. Balmino. EIGEN-6 A new combined global gravity field model including GOCE data from the collaboration of GFZ-Potsdam and GRGS-Toulouse. In *Geophysical Research Abstracts*, volume 13 of *General Assembly European Geosciences Union*, Vienna, 2011.

C. Förste, S. L. Bruinsma, F. Flechtner, J. C. Marty, J.-M. Lemoine, C. Dahle, O. Abrikosov, H. Neumayer, R. Biancale, F. Barthelmes, and G. Balmino. A preliminary update of the direct approach GOCE processing and a new release of EIGEN-6C. Number 0923 in AGU Fall Meeting, San Francisco, 2012.

W. Förstner. Ein Verfahren zur Schätzung von Varianz- und Kovarianzkomponenten. *Allgemeine Vermessungsnachrichten*, 11:446–453, 1979.

B. Frommknecht, D. Lamarre, M. Meloni, A. Bigazzi, and R. Floberghagen. GOCE level 1b data processing. *Journal of Geodesy*, 85(11):759–775, 2011. doi:10.1007/s00190-011-0497-4.

M. J. Fuchs and J. Bouman. Rotation of GOCE gravity gradients to local frames. *Geophysical Journal International*, 187(2):743–753, 2011. doi:10.1111/j.1365-246X.2011.05162.x.

E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In D. Kranzlmüller, P. Kacsuk, and J. J. Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 3241 of *Lecture Notes in Computer Science*, pages 97–104. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-23163-9. doi:10.1007/978-3-540-30218-6_19.

G. A. Geist, J. A. Kohl, and P. M. Papadopoulos. PVM and MPI: a comparison of features. *Calculateurs Paralleles*, 8:137–150, 1996.

C. Gerlach, N. Sneeuw, P. N. A. M. Visser, and D. Svehla. CHAMP gravity field recovery using the energy balance approach. *Advances in Geosciences*, 1(1):73–80, 2003. doi:10.5194/adgeo-1-73-2003.

H. Goiginger, D. Rieser, T. Mayer-Gürr, R. Pail, T. Fecher, T. Gruber, A. Albertella, J. M. Brockmann W.-D. Schuh, J. Kusche, A. Eicker, A. Jäggi, U. Meyer, W. Hausleitner, E. Höck, A. Maier, S. Krauss, and O. Baur. The combined satellite-only global gravity field model GOCO02S. In *Geophysical Reasearch Abstracts*, volume 13, Vienna, Austria, 2011.

G. H. Golub and C. F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, Maryland, 3rd edition, 1996.

W. Gropp, E. Lusk, and A. Skjellum. *Using MPI - Portable Parallel Programming with the Message-Passing Interface*. Scientific and Engineering Computation Series. MIT Press, Massachusetts, 1999a.

W. Gropp, E. Lusk, and R. Thakur. *Using MPI-2 Advanced features of the Message-Passing Interface*. Scientific and Engineering Computation Series. MIT Press, Massachusetts, 1999b.

T. Gruber. *Hochauflösende Schwerefeldbestimmung aus Kombination von terrestrischen Messungen und Satellitendaten über Kugelfunktionen*. PhD thesis, GeoForschungsZentrum Potsdam, Potsdam, Germany, 2000.

T. Gruber, P. N. A. M. Visser, C. Ackermann, and M. Hosse. Validation of GOCE gravity field models by means of orbit residuals and geoid comparisons. *Journal of Geodesy*, 85(11):845–860, 2011. doi:10.1007/s00190-011-0486-7.

Z. Guangbin, C. Xiaotao, L. Xinfa Z. Xinhang, and L. Yuxing. On numerical methods for determination of earth gravity field model using mass satellite gravity gradiometry data. *Journal of Geodesy and Geodynamics*, 3(1): 57–62, 2012. doi:10.3724/SP.J.1246.2012.00057.

B. Gundlich, K.-R. Koch, and J. Kusche. Gibbs sampler for computing and propagating large covariance matrices. *Journal of Geodesy*, 77(9):514–528, 2003. doi:10.1007/s00190-003-0350-5.

B. C. Gunter and R. A. Van De Geijn. Parallel out-of-core computation and updating of the QR factorization. *ACM Transactions on Mathematical Software*, 31(1):60–78, 2005. doi:10.1145/1055531.1055534.

W. Hausleitner. Orbit and SGG data simulations. Technical report, ESA-Project CIGAR III / Phase 2, WP 221, Final-Report, Part 1, 1995.

W. A. Heiskanen and H. Moritz. *Physical Geodesy*. Institute of Physical Geodesy, Technical University, Graz, reprint edition, 1993.

M. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49 (6), Research Paper 2379, 1952.

C. Hirt, T. Gruber, and W. E. Featherstone. Evaluation of the first GOCE static gravity field models using terrestrial gravity, vertical deflections and EGM2008 quasigeoid heights. *Journal of Geodesy*, 85(10):723–740, 2011. doi:10.1007/s00190-011-0482-y.

B. Hofmann-Wellenhof and H. Moritz. *Physical geodesy*. Springer, Vienna, 2005.

S. A. Holmes and W. E. Featherstone. A unified approach to the clenshaw summation and the recursive computation of very high degree and order normalised associated legendre functions. *Journal of Geodesy*, 76(5):279–299, 2002. doi:10.1007/s00190-002-0216-2.

M. F. Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics*, 19(2):433–450, 1990. doi:10.1080/03610919008812866.

K. H. Ilk, J. Kusche, and S. Rudolph. A contribution to data combination in ill-posed downward continuation problems. *Journal of Geodynamics*, 33:75–99, 2002. doi:10.1016/S0264-3707(01)00056-4.

Intel. *Intel© MPI Library for Linux OS*. Intel Corporation, 2013. URL `http://software.intel.com/en-us/articles/intel-mpi-library-documentation`. last accessed 2014-09-07.

FZ Jülich. JUROPA / HPC-FF. online, September 2013. URL `http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUROPA/JUROPA_node.html`. last accessed 2014-09-07.

B. Kargoll. *On the Theory and Application of Model Misspecification Tests in Geodesy*. PhD thesis, Institute of Geodesy and Geoinformation, University of Bonn, Bonn, Germany, 2007. URL `http://nbn-resolving.de/urn:nbn:de:hbz:5N-11136`.

G. E. Karniadakis and R. M. Kirby. *Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and their Implementation*. Cambridge University Press, 2003. ISBN 0521520800.

H. G. Katzgraber. Random numbers in scientific computing: An introduction. *ArXiv e-prints*, May 2010. URL `http://arxiv.org/abs/1005.4117`.

W. M. Kaula. *Theory of Satellite Geodesy: Applications of Satellites to Geodesy*. Dover Publishing Company, reprint edition, 1966.

R. Klees, P. Ditmar, and P. Broersen. How to handle colored observation noise in large least-squares problems. *Journal of Geodesy*, 76(11-12):629–640, 2003. doi:10.1007/s00190-002-0291-4.

P. Knudsen, R. Bingham, O. B. Andersen, and M.-H. Rio. A global mean dynamic topography and ocean circulation estimation using a preliminary GOCE gravity model. *Journal of Geodesy*, 85(11):861–879, 2011. doi:10.1007/s00190-011-0485-8.

K. R. Koch. *Parameter Estimation and Hypothesis Testing in Linear Models*. Springer Berlin/Heidelberg, 2nd edition, 1999.

K. R. Koch. *Introduction to Bayesian Statistics*. Springer, Heidelberg, 2nd edition, 2007.

K. R. Koch and J. Kusche. Regularization of geopotential determination from satellite data by variance components. *Journal of Geodesy*, 76(5):259–268, 2002. doi:10.1007/s00190-002-0245-x.

K. R. Koch, H. Kuhlmann, and W.-D. Schuh. Approximating covariance matrices estimated in multivariate models by estimated auto- and cross-covariances. *Journal of Geodesy*, 84(6):383–397, 2010. doi:10.1007/s00190-010-0375-5.

K.R. Koch, J. M. Brockmann, and W.-D. Schuh. Optimal regularization for geopotential model GOCO02S by Monte Carlo methods and multi-scale representation of density anomalies. *Journal of Geodesy*, 86(8):647–660, 2012. doi:10.1007/s00190-012-0546-7.

A. S. Konopliv, R. S. Park, D.-N. Yuan, S. W. Asmar, M. M. Watkins, J. G. Williams, E. Fahnestock, G. Kruizinga, M. Paik, D. Strekalov, N. Harvey, D. E. Smith, and M. T. Zuber. The jpl lunar gravity field to spherical harmonic degree 660 from the grail primary mission. *Journal of Geophysical Research: Planets*, 118(7):1415–1434, 2013. doi:10.1002/jgre.20097.

I. Krasbutter and W.-D. Schuh. Untersuchung der Kreuzkorrelation bei der Satellitenmission GOCE. Technical Report 1, University of Bonn, Bonn, Germany, 2010.

I. Krasbutter, J. M. Brockmann, B. Kargoll, and W.-D. Schuh. Stochastic model refinements for GOCE gradiometer data. In U. Münch and W. Dransch, editors, *Observation of the System Earth from Space*, volume 17 of *Geotechnologien Science Report*, pages 70–76, 2011a.

I. Krasbutter, J. M. Brockmann, B. Kargoll, W.-D. Schuh, H. Goiginger, and R. Pail. Refinement of the stochastic model of GOCE scientific data in a long time series. In *4th International GOCE user workshop*, Munich, Germany, 2011b.

I. Krasbutter, J. M. Brockmann, B. Kargoll, and W.-D. Schuh. Adjustment of digital filters for decorrelation of GOCE SGG data. In F. Flechtner, N. Sneeuw, and W.-D. Schuh, editors, *Observation of the System Earth from Space - CHAMP, GRACE, GOCE and future missions*, Advanced Technologies in Earth Sciences, pages 109–114. Springer Berlin Heidelberg, 2014. ISBN 978-3-642-32134-4. doi:10.1007/978-3-642-32135-1_14.

S. Kuhlins and M. Schader. *Die C++ Standardbibliothek*. Springer Berlin/Heidelberg, 2005.

J. Kusche. A Monte-Carlo technique for weight estimation in satellite geodesy. *Journal of Geodesy*, 76(11-12):641–652, 2003. doi:10.1007/s00190-002-0302-5.

C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for FORTRAN usage. *ACM Transactions on Mathematical Software*, 5(3):308–323, September 1979. doi:10.1145/355841.355847.

F. G. Lemoine, S. C. Kenyon, J. K. Factor, R.G. Trimmer, N. K. Pavlis, D. S. Chinn, C. M. Cox, S. M. Klosko, S. B. Luthcke, M. H. Torrence, Y. M. Wang, R. G. Williamson, E. C. Pavlis, R. H. Rapp, and T. R. Olson. *The Development of the Joint NASA GSFC and NIMA Geopotential Model EGM96*. NASA Goddard Space Flight Center, Greenbelt, Maryland, 1996. URL `http://cddis.gsfc.nasa.gov/egm96/egm96.html`.

F. G. Lemoine, S. Goossens, T. J. Sabaka, J. B. Nicholas, E. Mazarico, D. D. Rowlands, B. D. Loomis, D. S. Chinn, D. S. Caprette, G. A. Neumann, D. E. Smith, and M. T. Zuber. High-degree gravity models from GRAIL primary mission data. *Journal of Geophysical Research: Planets*, 118(8):1676–1698, 2013. doi:10.1002/jgre.20118.

A. Löcher. *Möglichkeiten der Nutzung kinematischer Satellitenbahnen zur Bestimmung des Gravitationsfeldes der Erde*. PhD thesis, Institute of Geodesy and Geoinformation, University of Bonn, Bonn, Germany, 2010. URL `http://nbn-resolving.de/urn:nbn:de:hbz:5N-21814`.

H. Löf. *Parallelizing the Method of Conjugate Gradients for Shared Memory Architectures.* Licentiate thesis, Department of Information Technology, Uppsala University, November 2004.

A. Maier, S. Krauss, W. Hausleitner, and O. Baur. Contribution of satellite laser ranging to combined gravity field models. *Advances in Space Research*, 49(3):556–565, 2012. doi:10.1016/j.asr.2011.10.026.

T. Mayer-Gürr. *Gravitationsfeldbestimmung aus der Analyse kurzer Bahnbögen am Beispiel der Satellitenmissionen CHAMP und GRACE.* PhD thesis, University of Bonn, Bonn, Germany, 2006. URL `http://nbn-resolving.de/urn:nbn:de:hbz:5N-09047`.

T. Mayer-Gürr, K. H. Ilk, A. Eicker, and M. Feuchtinger. ITG-CHAMP01: a CHAMP gravity field model from short kinematic arcs over a one-year observation period. *Journal of Geodesy*, 78(7-8):462–480, 2005. doi:10.1007/s00190-004-0413-2.

T. Mayer-Gürr, A. Eicker, E. Kurtenbach, and K.-H. Ilk. ITG-GRACE: Global static and temporal gravity field models from GRACE data. In F. Flechtner, T. Gruber, A. Güntner, M. Mandea, M. Rothacher, T. Schöne, and J. Wickert, editors, *System Earth via Geodetic-Geophysical Space Techniques*, Advanced Technologies in Earth Sciences, pages 159–168. Springer Berlin Heidelberg, 2010a. ISBN 978-3-642-10228-8. doi:10.1007/978-3-642-10228-8_13.

T. Mayer-Gürr, E. Kurtenbach, and A. Eicker. ITG-GRACE2010. online, 2010b. URL `http://www.igg.uni-bonn.de/apmg/index.php?id=itg-grace2010`. last accessed 2014-09-07.

T. Mayer-Gürr, D. Rieser, R. Pail, T. Fecher, T. Gruber, J. M. Brockmann W.-D. Schuh, J. Kusche, A. Eicker, A. Jäggi, U. Meyer, W. Hausleitner, E. Höck, A. Maier, S. Krauss, and O. Baur. The new combined satellite-only model GOCO03S. In *Paper presented at International Symposium on Gravity, Geoid and Height Systems (GGHS2012)*, IAG Symposium, Venice, Italy, 2012.

R. Mayrhofer, R. Pail, and T. Fecher. Quicklook gravity field solutions as part of the GOCE quality assessment. In H. Lacoste-Francis, editor, *Proceedings of the ESA Living Planet Symposium, ESA Publication SP-686.* ESA/ESTEC, 12 2010.

P. Meissl. *Least Squares Adjustment A modern Approach*, volume 43 of *Mitteilungen der geodätischen Institute der Technischen Universität Graz.* Geodätische Institute der Technischen Universität Graz, A-8010 Graz, Steyrergasse 30, 1982. URL `ftp://skylab.itg.uni-bonn.de/schuh/Separata_Meissl/Meissl_1982_Least_Squares_Adjustment_A_Modern_Approach.pdf`.

S. Mertens. Random number generators: A survival guide for large scale simulations. *ArXiv e-prints*, 2009. URL `http://arxiv.org/abs/0905.4238`.

B. Metzler. *Spherical Cap Regularization – A Spatially Restricted Regularization Method Tailored to the Polar Gap Problem.* PhD thesis, TU Graz, Graz, Austria, 2007.

B. Metzler and R. Pail. GOCE data processing: The spherical cap regularization approach. *Studia Geophysica et Geodaetica*, 49(4):441–462, 2005. doi:10.1007/s11200-005-0021-5.

F. Migliaccio, M. Reguzzoni, F. Sansó, and N. Tselfes. An error model for the GOCE space-wise solution by monte carlo methods. In M. G. Sideris, editor, *Observing our Changing Earth*, volume 133 of *International Association of Geodesy Symposia*, pages 337–344. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-85425-8. doi:10.1007/978-3-540-85426-5_40.

F. Migliaccio, M. Reguzzoni, A. Gatti, F. Sansò, and M. Herceg. A GOCE-only global gravity field model by the space-wise approach. In L. Ouwehand, editor, *Proceedings of the 4th International GOCE User Workshop, ESA Publication SP-696.* ESA/ESTEC, 07 2011. URL `http://www.spacebooks-online.com/product_info.php?cPath=104&products_id=17254`.

MPI-Forum. MPI: A message-passing interface standard 2.2. online, 2009. URL `http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf`. last accessed 2014-09-07.

NETLIB/SCALAPACK. ScaLAPACK – Scalable Linear Algebra PACKage. online, 2012. URL `http://www.netlib.org/scalapack/`. last accessed 2014-09-07.

Octave community. GNU Octave, 2014. URL `www.gnu.org/software/octave/`. last accessed 2014-09-07.

C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8(1):43–71, March 1982. doi:10.1145/355984.355989.

R. Pail and G. Plank. Assessment of three numerical solution strategies for gravity field recovery from GOCE satellite gravity gradiometry implemented on a parallel platform. *Journal of Geodesy*, 76(8):462–474, 2002. doi:10.1007/s00190-002-0277-2.

R. Pail and G. Plank. Comparison of numerical solution strategies for gravity field recovery from GOCE SGG observations implemented on a parallel platform. *Advances in Geosciences*, 1:39–45, 2003. doi:10.5194/adgeo-1-39-2003.

R. Pail and M. Wermuth. GOCE SGG and SST quick-look gravity field analysis. *Advances in Geosciences*, 1:5–9, 2003. doi:10.5194/adgeo-1-5-2003.

R. Pail, B. Metzler, B. Lackner, T. Preimesberger, E. Höck, W.-D. Schuh, H. Alkhatib, C. Boxhammer, C. Siemes, and M. Wermuth. GOCE gravity field analysis in the framework of HPF: operational software system and simulation results. In 3^{rd} *GOCE user workshop*, pages 249–256, Frascati, 2006. ESA. URL `http://earth.esa.int/workshops/GOCE06/`.

R. Pail, H. Goiginger, R. Mayrhofer W.-D. Schuh, J. M. Brockmann, I. Krasbutter, E. Höck, and T. Fecher. Global gravity field model derived from orbit and gradiometry data applying the time-wise method. In *ESA Living Planet Symposium*, Bergen, 28.06. - 02.07. 2010 2010a. SP-686.

R. Pail, H. Goiginger, W.-D. Schuh, E. Höck, J. M. Brockmann, T. Fecher, T. Gruber, T. Mayer-Gürr, J. Kusche, A. Jäggi, and D. Rieser. Combined satellite gravity field model GOCO01S derived from GOCE and GRACE. *Geophysical Research Letters*, 37:L20314, 2010b. doi:10.1029/2010GL044906.

R. Pail, S. Bruinsma, F. Miggliaccio, C. Förste, H. Goiginger, W.-D. Schuh, E. Höck, M. Reguzzoni, J. M. Brockmann, O. Abrikosov, M. Veicherts, T. Fecher, R. Mayrhofer, I. Krasbutter, F. Sansó, and C. C. Tscherning. First GOCE gravity field models derived by three different approaches. *Journal of Geodesy*, 85(11):819 – 843, 2011a. doi:10.1007/s00190-011-0467-x.

R. Pail, H. Goiginger, W.-D. Schuh, E. Höck, J.-M. Brockmann, T. Fecher, D. Rieser, I. Krasbutter, and T. Mayer-Gürr. GOCE-only gravity field models derived from 8 months of GOCE data. In *4th International GOCE user workshop*, Munich, Germany, 2011b.

R. Pail, T. Gruber, T. Fecher, M. Rexer, W.-D. Schuh, J. Kusche, J. M. Brockmann, I. Krasbutter, S. Becker, A. Eicker, J. Schall, T. Mayer-Gürr, D. Rieser, N. Zehentner, O. Baur, E. Höck, W. Hausleitner, A. Maier, S. Krauss, A. Jäggi, U. Meyer, and L. Prange. Gravity observation combination (GOCO). online, 2014. URL `http://www.goco.eu/`. last accessed 2014-09-01.

N. K. Pavlis, S. A. Holmes, S. Kenyon, and J. K. Factor. The development and evaluation of the earth gravitational model 2008 (EGM2008). *Journal of Geophysical Research: Solid Earth*, 117(B4), 2012. doi:10.1029/2011JB008916.

G. Plank. *Numerical solution strategies for the GOCE mission by using cluster technologies.* PhD thesis, TU Graz, Graz, Austria, 2004.

W. H. Press, A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, New York, NY, USA, 3rd edition, 2007. ISBN 0521880688, 9780521880688.

T. Rauber and G. Rünger. *Parallel Programming for Multicore and Cluster Systems.* Springer, Berlin, Heidelberg, 3rd edition, 2013.

M. Reguzzoni and F. Sansò. On the combination of high-resolution and satellite-only global gravity models. *Journal of Geodesy*, 86(6):393–408, 2012. doi:10.1007/s00190-011-0526-3.

C. Reigber. *Zur Bestimmung des Gravitationsfeldes der Erde aus Satellitenbeobachtungen.* DGK, Reihe C, 137, Munich, Germany, 1969.

C. Reigber, H. Lühr, and P. Schwintzer. CHAMP mission status. *Advances in Space Research*, 30(2):129 – 134, 2002. doi:10.1016/S0273-1177(02)00276-4.

T. Reubelt. *Harmonische Gravitationsfeldanalyse aus GPS-vermessenen kinematischen Bahnen niedrig fliegender Satelliten vom Typ CHAMP, GRACE und GOCE mit einem hoch auflösenden Beschleunigungsansatz.* PhD thesis, Universität Stuttgart, Stuttgart, Germany, 2009. URL `http://nbn-resolving.de/urn:nbn:de:bsz:93-opus-39258`.

T. Reubelt, N. Sneeuw, and E. W. Grafarend. Comparison of kinematic orbit analysis methods for gravity field recovery. In N. Sneeuw, P. Novák, M. Crespi, and F. Sansò, editors, *VII Hotine-Marussi Symposium on Mathematical Geodesy*, volume 137 of *International Association of Geodesy Symposia*, pages 259–265. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-22077-7. doi:10.1007/978-3-642-22078-4_39.

M. Rexer, C. Hirt, R. Pail, and S. Claessens. Evaluation of the third- and fourth-generation GOCE earth gravity field models with australian terrestrial gravity data in spherical harmonics. *Journal of Geodesy*, 88(4):319–333, 2014. doi:10.1007/s00190-013-0680-x.

J. C. Ries, S. Bettadpur, S. Poole, and T. Richter. Mean background gravity fields for GRACE processing. Paper presented at GRACE Science Team Meeting, Austin, Texas, 2011.

R. Rummel, F. Sansò, M. van Gelderen, M. Brovelli, R. Koop, F. Miggliaccio, E. Schrama, and F. Scerdote. *Spherical harmonic analysis of satellite gradiometry*, volume 39 of *Publications on Geodesy, New Series.* Netherlands Geodetic Commission, Delft, Netherlands, 1993.

R. Rummel, W. Yi, and C. Stummer. GOCE gravitational gradiometry. *Journal of Geodesy*, 85(11):777–790, 2011. doi:10.1007/s00190-011-0500-0.

Y. Saad. *Iterative Methods for Sparse Linear Systems.* SIAM, Society for Industrial and Applied Mathematics, 2nd edition, 2000. URL `http://www-users.cs.umn.edu/~saad/books.html`.

F. Sansò and C. C. Tscherning. Fast spherical collocation: theory and examples. *Journal of Geodesy*, 77(1–2):101–112, 2003. doi:10.1007/s00190-002-0310-5.

J. Schall, A. Eicker, and J. Kusche. The ITG-Goce02 gravity field model from GOCE orbit and gradiometer data based on the short arc approach. *Journal of Geodesy*, 88(4):403–409, 2014. doi:10.1007/s00190-014-0691-2.

M. Scheinert. The antarctic geoid project: Status report and next activities. In C. Jekeli, L. Bastos, and J. Fernandes, editors, *Gravity, Geoid and Space Missions*, volume 129 of *International Association of Geodesy Symposia*, pages 137–142. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-26930-4. doi:10.1007/3-540-26932-0_24.

M. Scheinert, J. Müller, R. Dietrich, D. Damaske, and V. Damm. Regional geoid determination in antarctica utilizing airborne gravity and topography data. *Journal of Geodesy*, 82(7):403–414, 2008. doi:10.1007/s00190-007-0189-2.

M. Schneider. Lösungsvorschlag zum Bahnbestimmungsproblem. Bundesministerium für wissenschaftliche Forschung (BMwF), Weltraumforschung Forschungsbericht FB W 67-35, Deutsche Gesellschaft für Flugwissenschaften (DGF), 1967.

W.-D. Schuh. SST/SGG tailored numerical solution strategies. Technical report, ESA-Project CIGAR III / Phase 2, WP 221, Final-Report, Part 2, 1995.

W.-D. Schuh. *Tailored Numerical Solution Strategies for the Global Determination of the Earth's Gravity Field*, volume 81 of *Mitteilungen der geodätischen Institute der Technischen Universität Graz*. TU Graz, Graz, Austria, 1996.

W.-D. Schuh. *Numerische Verfahren zur geodätischen Optimierung.* lecture notes. Theoretical Geodesy, University of Bonn, 2001. URL `ftp://skylab.itg.uni-bonn.de/schuh/Skriptum/Numerische_Methoden.pdf`.

W.-D. Schuh. Improved modeling of SGG-data sets by advanced filter strategies. In *ESA-Project "From Eötvös to mGal+", WP 2, Final-Report*, pages 113–181. ESA/ESTEC Contract No. 14287/00/NL/DC, 2002.

W.-D. Schuh. The processing of band-limited measurements; filtering techniques in the least squares context and in the presence of data gaps. *Space Science Reviews*, 108(1-2):67–78, 2003. doi:10.1023/A:1026121814042. URL `http://dx.doi.org/10.1023/A%3A1026121814042`.

W.-D. Schuh, C. Boxhammer, and C. Siemes. Correlations, variances, covariances — from GOCE signals to GOCE products. In 3^{rd} *GOCE user workshop*, Frascati, 2006. ESA. URL `http://earth.esa.int/workshops/GOCE06/`.

W.-D. Schuh, J. M. Brockmann, B. Kargoll, I. Krasbutter, and R. Pail. Refinement of the stochastic model of GOCE scientific data and its effect on the in-situ gravity field solution. In *ESA Living Planet Symposium*, Bergen, 28.06. - 02.07. 2010 2010. SP-686.

H. R. Schwarz. Die Methode der konjugierten Gradienten in der Ausgleichsrechnung. *ZfV*, 95:130–140, 1970.

P. Schwintzer, C. Reigber, A. Bode, Z. Kang, S. Y. Zhu, F.-H. Massmann, Raimondo, R. Biancale, G. Balmino, J.M. Lemoine, B. Moynot, J. C. Marty, F. Barlier, and Y. Boudon. Long-wavelength global gravity field models: GRIM4-S4, GRIM4-C4. *Journal of Geodesy*, 71(4):189–208, 1997. doi:10.1007/s001900050087.

G. Seeber. *Satellite Geodesy.* Walter de Gruyter, Berlin, New York, 2nd edition, 2003.

J. R. Shewchuk. An introduction to the conjugate gradient method without the agonising pain. Technical report, Pittsburgh, 1994. URL `http://www.cs.cmu.edu/~jrs/jrspapers.html`.

M. Sidani and B. Harrod. Parallel matrix distributions: Have we been doing it all right? Technical Report 116, LAPACK Working Note, November 1996. URL `http://www.netlib.org/lapack/lawnspdf/lawn116.pdf`.

C. Siemes. *Digital Filtering Algorithms for Decorrelation within Large Least Squares Problems.* PhD thesis, Institute of Geodesy and Geoinformation, University of Bonn, Bonn, Germany, 2008. URL `http://nbn-resolving.de/urn:nbn:de:hbz:5N-13749`.

C. Siemes, M. Fehringer, R. Floberghagen, B. Frommknecht, and R. Haagmans. GOCE gravity gradient performance: Evolution during mission lifetime. In *Paper presented at "The Living Planet Syposium 2013"*, Edinburgh, 2013. ESA.

N. Sneeuw. *A Semi-Analytical Approach to Gravity Field Analysis from Satellite Observations.* PhD thesis, Institute for Astronomical and Physical Geodesy, Technische Universität München, Munich, Germany, 2000.

B. Stroustrup. *Die C++-Programmiersprache.* Addison Wesley Verlag, 3rd edition, 2000. 3827312965.

C. Stummer. *Gradiometer data processing and analysis for the GOCE mission.* PhD thesis, Institute for Astronomical and Physical Geodesy, Technische Universität München, Munich, Germany, 2013. URL `http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20121123-1111698-0-3`.

C. Stummer, T. Fecher, and R. Pail. Alternative method for angular rate determination within the GOCE gradiometer processing. *Journal of Geodesy*, 85(9):585–596, 04 2011. doi:10.1007/s00190-011-0461-3.

C. Stummer, C. Siemes, R. Pail, B. Frommknecht, and R. Floberghagen. Upgrade of the GOCE level 1b gradiometer processor. *Advances in Space Research*, 49(4):739–752, 03 2012. doi:10.1016/j.asr.2011.11.027.

V. S. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency: Practice and Experience*, 2: 315–339, 1990.

H. Sünkel, W. Hausleitner, and W.-D. Schuh. SST/SGG tailored numerical solution strategies. Technical report, ESA-Project CIGAR III / Phase 2, WP 221, Final-Report, 1995.

B. D. Tapley, S. Bettadpur, J. C. Ries, P. F. Thompson, and M. M. Watkins. GRACE measurements of mass variability in the earth system. *Science*, 305(5683):503–505, 2004. doi:10.1126/science.1099192.

J. Thiyagalingam. *Alternative Array Storage Layouts for Regular Scientific Programs.* PhD thesis, Department of Computing, Imperial College, London, UK, 2005.

J. Thiyagalingam, O. Beckmann, and P. Kelly. Minimizing associativity conflicts in morton layout. In R. Wyrzykowski, J. J. Dongarra, N. Meyer, and J. Wasniewski, editors, *Parallel Processing and Applied Mathematics*, volume 3911 of *Lecture Notes in Computer Science*, pages 1082–1088. Springer Berlin / Heidelberg, 2006. doi:10.1007/11752578_131.

M. van Gelderen and R. Koop. The use of degree variances in satellite gradiometry. *Journal of Geodesy*, 71(6): 337–343, 1997. doi:10.1007/s001900050101.

J. van Loon. *Functional and stochastic modelling of satellite gravity data.* PhD thesis, TU Delft, Delft, Netherlands, 2008.

C. Voigt and H. Denker. Regional validation and combination of GOCE gravity field models and terrestrial data. In F. Flechtner, N. Sneeuw, and W.-D. Schuh, editors, *Observation of the System Earth from Space - CHAMP, GRACE, GOCE and future missions*, Advanced Technologies in Earth Sciences, pages 139–145. Springer Berlin Heidelberg, 2014. ISBN 978-3-642-32134-4. doi:10.1007/978-3-642-32135-1_18.

P. Wessel and W. H. F. Smith. New, improved version of generic mapping tools released. *Eos, Transactions American Geophysical Union*, 79(47):579–579, 1998. doi:10.1029/98EO00426.

P. Wessel and W. H. F. Smith. *The Generic Mapping Tools GMT Technical Reference and Cookbook*, 4th edition, 2004.

R. C. Whaley and J. J. Dongarra. Automatically tuned linear algebra software. Technical Report 131, LAPACK Working Note, December 1997. URL `http://www.netlib.org/lapack/lawnspdf/lawn131.pdf`.

R. C. Whaley, A. P. Petitet, and J. J. Dongarra. Automated empirical optimization of software and the ATLAS project. Technical Report 147, LAPACK Working Note, September 2000. URL `http://www.netlib.org/lapack/lawnspdf/lawn147.pdf`.

J. Xie. *Implementation of Parallel Least-Squares Alorithms for Gravity Field Estimation.* Number No. 474 in Reports of the Department of Geodetic Science. Ohio State University (OSU), Ohio, 2005.

W. Yi. An alternative computation of a gravity field model from GOCE. *Advances in Space Research*, 50(3):371 – 384, 2012. doi:10.1016/j.asr.2012.04.018.

M. T. Zuber, D. E. Smith, M. M. Watkins, S. W. Asmar, A. S. Konopliv, F. G. Lemoine, H. J. Melosh, G. A. Neumann, R. J. Phillips, S. C. Solomon, M. A. Wieczorek, J. G. Williams, S. J. Goossens, G. Kruizinga, E. Mazarico, R. S. Park, and D.-N. Yuan. Gravity field of the moon from the gravity recovery and interior laboratory (GRAIL) mission. *Science*, 339(6120):668–671, 2013. doi:10.1126/science.1231507.