Institut für Theoretische Geodäsie der Universität Bonn

# Effiziente numerische Verfahren zur sphärischen harmonischen Analyse von Satellitendaten

Inaugural–Dissertation zur
Erlangung des akademischen Grades
Doktor–Ingenieur (Dr.–Ing.)
der Hohen Landwirtschaftlichen Fakultät
der Rheinischen Friedrich–Wilhelms–Universität
zu Bonn

vorgelegt am 03. Mai 2006 von

Dipl.–Ing. Christian Boxhammer
aus Mönchengladbach

D 98

| | |
|---|---|
| Referent: | Prof. Dr. techn. W.–D. Schuh |
| 1. Koreferent: | Prof. Dr.–Ing., Dr.–Ing. E.h. mult. K. R. Koch (em.) |
| 2. Koreferent: | Prof. Dr.–Ing. K. H. Ilk |
| Tag der mündlichen Prüfung: | 16.06.2006 |
| Gedruckt bei: | Diese Dissertation ist auf dem Hochschulschriftenserver der ULB Bonn `http://hss.ulb.uni-bonn.de/diss_online` elektronisch publiziert. (Erscheinungsjahr: 2006) |

# Effiziente numerische Verfahren zur sphärischen harmonischen Analyse von Satellitendaten


## Zusammenfassung

In dieser Arbeit wird ein Umnummerierungsverfahren für die unbekannten Parameter bei der sphärischen harmonischen Analyse vorgestellt. Durch die Umsortierung der unbekannten Parameter wird ein vorgegebenes Besetztheitsmuster, die sogenannte *Kitematrix*, in der Normalgleichungsmatrix erzeugt. Diese erlaubt eine effiziente Kombination von hochauflösenden, regelmäßigen Daten mit niedrigauflösenden, nicht regelmässigen Daten. Die bisherige Beschränkung des Kite Nummerierungschemas, fest an einen minimalen Entwicklungsgrad von 2 und einen konstanten, maximalen Entwicklungsgrad gebunden zu sein, wird durch das vorgestellte *Freie Kite–Nummerierungschema* aufgehoben. Eine Methode zur Speicherung der schwach besetzten Kitematrix wird entwickelt und alle für die Lösung des Normalgleichungssystems notwendigen Algorithmen, die an dieses Speicherschema angepasst wurden, werden ausführlich besprochen. Das hier vorgestellte Verfahren kann zur strengen Lösung von regelmäßigen und nicht regelmäßigen Daten eingesetzt werden. Ein weiterer Einsatzbereich ist die Vorkonditionierung bei einem iterativen Verfahren. Mit dem Freien Kite–Nummerierungschema ist im Zuge dieser Arbeit das iterative Ausgleichungsverfahren PCGMA von Schuh (1996) erweitert worden. Das PCGMA Verfahren musste dazu neu implementiert werden und das entstandene Programm wird im Zuge des ESA Projektes GOCE zur Feinabstimmung des funktionalen und stochastischen Modells auf der Basis von Echtdaten dienen. Weil die Datenmenge des Satelliten GOCE sehr groß sein wird, ist eine Parallelverarbeitung auf mehreren Prozessoren auch bei diesem effizienten Verfahren zwingend notwendig. Die realisierte Parallelisierungsstrategie wird vorgestellt. Die Effizienz des Freien Kite Nummerierungschemas bei der Vorkonditionierung wird am Schluss der Arbeit durch drei verschiedene Testszenarien belegt.


# Efficient Numerical Methods for the Spherical Harmonic Analysis of Satellite Data


## Summary

This thesis introduces a numbering scheme for the unknown parameters in the spherical harmonic analysis using satellite data. By reordering the unknown parameters a specific sparsity pattern in the normal equation matrix, the so–called *kite matrix* will be created. This allows an efficient combination of high–resolution regular data sets with low–resolution non–regular data sets. The present restriction of the kite numbering scheme which is the fixed minimum harmonic degree of 2 and the constant maximum degree is abolished by the *free kite numbering scheme*. A technique for storing the sparse kite matrix is developed and the algorithms which are needed to solve a system of normal equations and which had to be adapted to this technique are discussed in detail. The free kite numbering scheme can be used for the rigorous combination of gridded and non–gridded data sets as well as for the preconditiong within iterative methods. The iterative method PCGMA which was introduced by Schuh (1996) is improved by the free kite numbering scheme. In the course of the GOCE project of ESA the program `pcgma` had to be reimplemented and the resulting program will be used for the fine–tuning of the functional and the stochastic model in the processing of real data from the GOCE satellite. The need for parallel computing is given by the huge number of observations from GOCE and the huge number of unknowns. The realised concept of parallel processing is described. The efficiency of the free kite numbering scheme in preconditioning will be shown by comparison of three different test scenarios.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Schreibweisen, Symbole und Abkürzungen

### Der Doppelpunktoperator

| | |
|---|---|
| 1:10 | Bildet eine Reihe von ganzen Zahlen $z$ von 1 bis 10 |
| 1:2:10 | Bildet eine Reihe von ganzen Zahlen von 1 bis 10 mit Inkrement 2 (1 3 5 7 9) |

### Aufteilung einer Matrix

| | |
|---|---|
| $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ | Eine Matrix mit $m$ Zeilen und $n$ Spalten |
| $\boldsymbol{A}_{ij}$ | Element in der Zeile $i$ und der Spalte $j$ der Matrix $\boldsymbol{A}$ |
| $\boldsymbol{A}(i,:)$ | Der $i$-te Zeilenvektor von $\boldsymbol{A}$ |
| $\boldsymbol{A}(:,i)$ | Der $i$-te Spaltenvektor von $\boldsymbol{A}$ |
| $\boldsymbol{A} = \left[\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\right]$ | Matrix als Menge von Spaltenvektoren |
| $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_1 \\ \vdots \\ \boldsymbol{a}_n \end{bmatrix}$ | Matrix als Menge von Zeilenvektoren |
| $\boldsymbol{A}(1{:}4, 3{:}7)$ | Untermatrix von $\boldsymbol{A}$, die die Zeilen 1 bis 4 aus den Spalten 3 bis 7 enthält. |

### Blockmatrizen

Als Blockindizes werden immer griechische Buchstaben verwendet, um zwischen Blocknotation und skalarer Notation zu unterscheiden.

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & \ldots & \boldsymbol{A}_{1\xi} \\ \vdots & & \vdots \\ \boldsymbol{A}_{\eta 1} & \ldots & \boldsymbol{A}_{\eta\xi} \end{bmatrix} \begin{matrix} m_1 \\ \\ m_\eta \end{matrix} \qquad \begin{matrix} n_1 & & n_\xi \end{matrix}$$

Darstellung der Matrix $\boldsymbol{A}$ als eine $\eta \times \xi$ Blockmatrix
Hierbei gilt: $\sum_{\alpha=1}^{\eta} m_\alpha = m$ und $\sum_{\beta=1}^{\xi} n_\beta = n$.
Mit dieser Notation hat der Block $\boldsymbol{A}_{\alpha\beta}$ die Dimension $m_\alpha \times n_\beta$.

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_1 \\ \vdots \\ \boldsymbol{A}_\eta \end{bmatrix} \begin{matrix} m_1 \\ \\ \\ m_\eta \end{matrix}$$

Darstellung der Matrix $\boldsymbol{A}$ als ein $\eta \times 1$ Blockvektor. Ein Block enthält jeweils $m_\alpha, \alpha \in 1, \ldots, \eta$, Zeilen der Matrix.

# 1 Einleitung

## 1.1 Die Satellitenmissionen CHAMP, GRACE und GOCE

Das heutige Wissen über das Schwerefeld der Erde und seine geometrische Form reicht in vielen Anwendungen geowissenschaftlicher Disziplinen noch nicht aus. Die Kenntnis eines Schwerefeldes mit höherer Genauigkeit und räumlicher Auflösung ist notwendig, um die physikalischen Zusammenhänge im Inneren der Erde, die Strömungen der Ozeane und das Zusammenspiel von Kontinentverschiebungen, schmelzenden Eismassen und Meeresspiegelveränderungen untersuchen zu können (ESA 1999, S. 9).
Dieses Ziel kann nur mit Hilfe der Satellitengeodäsie erreicht werden, da eine globale, homogene Erfassung der Erde durch Messungen notwendig ist, um eine sphärische harmonische Analyse des Erdschwerefeldes mit entsprechend hohem Entwicklungsgrad berechnen zu können. Besonders die drei Satellitenmissionen CHAMP (Challenging Minisatellite Payload), GRACE (Gravity Recovery And Climate Experiment) und GOCE (Gravity and Steady-State Ocean Circulation Explorer) dienen der Erforschung der Erde als komplexes System. Die Bestimmung des Erdschwerefeldes spielt in diesem Zusammenhang eine zentrale Rolle.
Der Satellit CHAMP wurde im Juli 2000 in die Umlaufbahn gebracht und dient der Bestimmung des Schwerefeldes, des Magnetfeldes und der Ionosphäre und Troposphäre der Erde (Reigber et al. 2004). Die Mission CHAMP und die Auswertung der Daten wird unter der Federführung des Geoforschungszentrums Potsdam (GFZ) durchgeführt, aber auch anderen Institutionen stehen diese Daten zur Verfügung. So wurden z. B. von Mayer-guerr et al. (2005) Schwerefeldmodelle basierend auf CHAMP Daten veröffentlicht. Diese erlauben jedoch nur eine Schwerefeldbestimmung bis zum maximalen Grad von 62, wie auch von Koch (2005) mit Hilfe von Monte–Carlo–Simulationen unabhängig bestätigt werden konnte.
Die zwei Satelliten der GRACE-Mission wurden im März 2002 in ihre Umlaufbahn gebracht. Sie fliegen in einem Abstand von 220 km, wobei die Entfernung zwischen ihnen durch Mikrowellenmessungen mit $\mu$m Genauigkeit gemessen wird. Die Zielsetzung von GRACE ist, ein Schwerefeld mit 1 cm Genauigkeit in den Geoidhöhen bis zum maximalen Entwicklungsgrad von 125 zu erreichen (Tapley et al. 2005). Die GRACE Mission ist eine amerikanisch-deutsche Kooperation zwischen der NASA und dem Deutschen Zentrum für Luft- und Raumfahrt (DLR).
Die GOCE–Mission ist ein Projekt der Europäischen Weltraumbehörde ESA und soll das Schwerefeld bis zum Entwicklungsgrad 240 mit 1 cm Genauigkeit bestimmen, was einer räumlichen Auflösung von ca. 100 km entspricht. Dieses ehrgeizige Ziel kann nur mit Hilfe der Satellitengradiometrie (SGG, satellite gravity gradiometry) erreicht werden, dessen Prinzip z. B. in Rummel (1985) erläutert wird. Eine ausführliche Beschreibung der Missionsziele findet sich in ESA (1999 und 2002).
An Bord des GOCE–Satelliten befinden sich zwei komplementäre Sensoren, SST-hl (satellite-to-satellite tracking im high-low Modus) dient der Erfassung des langwelligen Schwerefeldsignals und die genannte SGG der Erfassung des kurzwelligen Signals.
Durch die hohe Genauigkeit und räumliche Auflösung überschreitet man bei der Datenauswertung zur GOCE–Mission die Kapazität heutiger Computer, da einerseits die Auswertezeiten sehr lang werden und andererseits die Gleichungssysteme so groß werden, dass sie nicht mehr in den Hauptspeicher verfügbarer Computersysteme passen.
Daher wurden im Vorfeld der GOCE–Mission verschiedene Untersuchungen durchgeführt, um ressourcenschonendere Auswerteverfahren zu finden. Schuh (1996) stellte das Verfahren PCGMA (Preconditioned Conjugate Gradient Multiple Adjustment) vor, welches auf der nach Schwarz (1970) modifizierten Form der Methode der Konjugierten Gradienten (CG) aufbaut. In dieser modifizierten Form liefert es die Lösung eines Normalgleichungssystems unter Benutzung der Beobachtungsgleichungen, ohne die Normalgleichungsmatrix explizit aufzustellen. Das Verfahren PCGMA ergänzt das modifizierte CG Verfahren durch die Möglichkeit unkorrelierte Gruppen von Beobachtungen sowohl in Form von Normalgleichungen, als auch in Form von Beobachtungsgleichungen miteinander zu kombinieren. Die Konvergenz des Verfahrens PCGMA wird durch Vorkonditionierung verbessert, indem für bestimmte Beobachtungsgruppen idealisierte Annahmen getroffen werden, die bei der Erfassung nur näherungsweise zutreffen. Das Verfahren PCGMA ist in dem Programm `pcgma` bereits von Auzinger (1997) und Auzinger und Schuh (1998) implementiert worden und die Parallelisierung ist von Plank (2002) begonnen worden.
Ein weiteres effizientes Verfahren, welches zur Datenauswertung der GOCE–Mission verwendet wird, basiert auf dem sogenannten *semianalytischen Ansatz (SA)* (Sneeuw 2000). Dieser Ansatz interpretiert die Messungen als Zeitreihe und berechnet das Schwerefeld mit Hilfe von Fouriertransformationen im Frequenzbereich. Dieses Verfahren liefert jedoch nur eine Näherungslösung, da es auf Annahmen bezüglich der Messwertverteilung beruht, die bei der Erfassung der Echtdaten nicht vollständig zutreffen werden.

Im Zuge des GOCE Projektes entwickelte PLANK (2004) ein Programmsystem, welches hochauflösende GOCE Schwerefeldmodelle über die parallele, verteilte Aufstellung der vollbesetzten Normalgleichungsmatrix und über einen separaten, parallelen Löser berechnet. Er nannte dieses Verfahren *Distributed Non-approximatative Adjustment* (DNA). Der Vorteil und zugleich Nachteil dieses Verfahrens liegt in der vollständigen Aufstellung der Normalgleichungen, denn es liefert damit zusätzlich zur Lösung auch die Varianz–Kovarianzmatrix, die Laufzeiten sind aber im Vergleich zum iterativen Programm `pcgma` extrem hoch.
Die drei Methoden SA, PCGMA und DNA werden im Rahmen des GOCE Projektes im Auftrag der ESA weiterentwickelt und implementiert, um nach dem Start des GOCE Satelliten die Datenauswertung durchzuführen. Dieses Softwaresystem ist Teil des *GOCE High-Level Processing Facility* (HPF, RUMMEL et al. (2004)). Das HPF ist ein dezentrales Hard– und Softwaresystem zur wissenschaftlichen Auswertung der GOCE Level 1b Daten zur Berechnung des offiziellen GOCE Schwerefeldmodells. Die Durchführung unterliegt dem *European GOCE Gravity Consortium (EGG-C)* (RUMMEL et al. 2004), welches eine Kooperation 10 Europäischer Forschungseinrichtungen ist. Das Institut für Theoretische Geodäsie der Universität Bonn ist zusammen mit den Partnerinstituten

- Institut für Navigation und Satellitengeodäsie der Universität Graz,
- Institut für Weltraumforschung der Österreichischen Akademie der Wissenschaften,
- Institut für Astronomische und Physikalische Geodäsie der Universität München,

an der Erstellung der Software und der Datenauswertung für das Arbeitspaket WP 6000 des HPF beteiligt. Dieses Softwaresystem ist in zwei Teile gegliedert (PAIL et al. 2005):

1. *Quicklook Gravity Analysis*
   Mit Hilfe der SA Methode wird eine schnelle, approximative Schwerefeldlösung berechnet, um die Qualität der Messungen während der Mission zu überwachen und um eine erste Näherung für das Fehlerverhalten der Messinstrumente zu berechnen.

2. *Core Solver*

   a) *Tuning machine*
   Mit dem Programm `pcgma` wird das Fehlerverhalten des SGG Messinstrumentes genauer untersucht, um optimale Dekorrelationsfilter zu schätzen. Dies ist ein iterativer Prozess, weshalb wiederholte Programmdurchläufe notwendig sind. Dabei kommt die Stärke der Methode PCGMA, eine schnelle und strenge Schwerefeldlösung zu liefern, zur Geltung. Zusätzliche Programmdurchläufe mit `pcgma` werden benötigt, um den optimalen Gewichtungsfaktor zwischen den SST und den SGG Daten zu ermitteln.

   b) *Final Solver*
   Berechnung einer strengen Schwerefeldlösung über die Aufstellung des vollbesetzten Normalgleichungssystems. Diese Lösung erfordert hohen Zeitaufwand und wird daher am Ende der Prozessierungskette durchgeführt, wenn die Tuningparameter wie Gewichtungsfaktor und optimale Filter bereits bestimmt worden sind. Der Final Solver liefert eine vollbesetzte Varianz–Kovarianzmatrix der geschätzten Parameter.

Der zentrale Teil des Verfahrens PCGMA ist die Vorkonditionierung, welche durch eine idealisierte Messanordnung auf ein schwach besetztes Normalgleichungssystem führt. Bei der Cholesky–Zerlegung und der Inversion von schwach besetzten Matrizen entstehen abhängig von der Besetztheitsstruktur der Matrix Füllelemente, weshalb SCHUH (1996) ein Umnummerierungsverfahren für die unbekannten Schwerefeldkoeffizienten entwickelt hat, das die Elemente der Normalgleichungsmatrix so anordnet, dass bei der Cholesky–Zerlegung kein Füllelement entsteht. Aufgrund des drachenartigen Aussehens des Besetztheitsmusters der Normalgleichungsmatrix wurde diese Nummerierung *Kite–Nummerierung* genannt.
Durch diese Vorkonditionierung kann beim Verfahren PCGMA eine sehr gute Konvergenz erreicht werden, so dass innerhalb von wenigen Iterationen (10-20) ein System mit 60 000 Parametern gelöst werden kann. Allerdings könnte durch eine datenadaptive Anpassung der Kite–Struktur die Konvergenz des Verfahrens nochmals deutlich gesteigert werden. Im ursprünglichen Entwurf des PCGMA Verfahrens (SCHUH 1996) wurde zwar schon spezielles Augenmerk auf diese Möglichkeiten gelegt, so wurden zur Dekorrelation der Messungen diskrete Filter entwickelt, die eine parallele Verarbeitung ermöglichen, aber eine konsequente Umsetzung als paralleles

Programm wurde nur als Prototyp implementiert (Plank 2002). Aufgrund von Nachfragen der GOCE Anwender (z. B. Ozeanographie) wurde auch zunehmend die Forderung nach der Varianz–Kovarianzinformation stärker, welche durch iterative Verfahren aber in der Regel nur sehr umständlich bereitgestellt werden kann. Hier wurde in den letzten Jahren intensive Forschungsarbeit geleistet, um diesen Nachteil der Methode PCGMA zu beseitigen. Die entwickelte Methode baut auf Vorarbeiten von Gundlich et al. (2003) auf, wo große Kovarianzmatrizen unter Umgehung der Inversion über Monte–Carlo Simulation direkt aus den Normalgleichungen abgeleitet wurden.
Die Berechnung von großen Kovarianzmatrizen über Monte-Carlo Simulation wurde von Koch et al. (2004) blockweise formuliert, um die vorherrschenden Matrix–Vektor Multiplikationen durch Matrix–Matrix Multiplikationen zu ersetzen. Diese können mit Hilfe hardwareoptimierter Bibliotheken effizienter berechnet werden. Um die Effizienz noch weiter zu steigern, nutzten sie die 15 Computer des Rechenclusters des Instituts für Theoretische Geodäsie, um das Verfahren zu parallelisieren.
Dieses Verfahren von Gundlich et al. (2003) wurde von Alkhatib und Schuh (2006) so erweitert, dass die explizite Aufstellung der Normalgleichungsmatrix nicht mehr notwendig ist. Ausserdem erfolgte die vollständige Integration des Monte–Carlo Verfahrens in den PCGMA Algorithmus. Dadurch gewinnt das in dieser Arbeit behandelte Verfahren PCGMA die Möglichkeit, die Kovarianzmatrix der geschätzten Parameter angeben zu können.

## 1.2 Zielsetzung der Arbeit

In dieser Arbeit wird ein Umnummerierungsverfahren für die unbekannten Parameter bei der sphärischen harmonischen Analyse vorgestellt. Dieses Verfahren baut auf dem Kite–Nummerierungschema auf, überwindet jedoch dessen Beschränkung, an einen festen minimalen Grad von 2 und einen konstanten maximalen Entwicklungsgrad gebunden zu sein. Es ist nun möglich, regelmäßig verteilte, hochauflösende Daten, die eine blockdiagonale Struktur in der Normalgleichungsmatrix erzeugen, streng mit niedrig auflösenden, vollbesetzten Normalgleichungssystemen zu kombinieren und dabei für jede Ordnung einen minimalen und einen maximalen Grad anzugeben, ohne dass dadurch die Kitestruktur der Normalgleichungsmatrix verloren geht. Aufgrund dieser Möglichkeit, die Menge der unbekannten Parameter frei bestimmen zu können, wurde das Verfahren welches in Boxhammer und Schuh (2006) erstmals vorgestellt wurde, *Freies Kite–Nummerierungsschema (FKN)* genannt.
Das Freie Kite–Nummerierungsschema ist durch die systematische Aufarbeitung der bisherigen Kitestruktur mit den folgenden Arbeitsschritten entstanden:

1. *Strenge Definition der Geometrie der Kitematrix*
   Die besondere Eigenschaft der Kitematrix, keine Füllelemente bei der Choleskyreduktion zu erzeugen und daher mit wenig Speicherplatzbedarf speicherbar zu sein, ist nur gegeben, wenn bestimmte Annahmen bezüglich der Geometrie des Besetztheitsmusters zutreffen.

2. *Regelbasierte Aufstellung der Reihenfolge der Parameter*
   Die Aufstellung der Reihenfolge der Parameter wurde von den Berechnungen mit der Kitematrix getrennt, um die Komplexität des Problems zu reduzieren. Die Menge der Parameter wird in verschiedene Zonen eingeteilt, um die Kitestruktur zur erhalten. Dazu sind Regeln entwickelt worden, die diese Einteilung automatisch vornehmen.

3. *Speicherschema für Kitematrizen*
   Eine Matrix kann mit dem entwickelten Speicherschema gespeichert werden, sobald sie die Definition einer Kitematrix erfüllt. Dabei werden die Blöcke der Kitematrix als zusammenhängende Matrizen gespeichert, um schnelle Blockalgorithmen nutzen zu können.

4. *Algorithmen*
   Die Algorithmen der linearen Algebra, die mit der Kitematrix durchgeführt werden müssen, sind an ihre spezielle Struktur angepasst worden, so dass diese Algorithmen mit jeder Kitematrix durchgeführt werden können, die der Definition genügt.

5. *Implementierung*
   Das Programm `pcgma` ist mit allen vorgestellten Algorithmen zum Umgang mit der Kitematrix neu implementiert worden. Das Programm ist eine voll funktionsfähige Software, die Teil der Erfüllung des ESA Vertrages zur Auswertung der GOCE Daten ist. Das Programm ist mit der objektorientierten Programmiersprache C++ implementiert und mit Hilfe des MPI Standards für parallele Programmierung

parallelisiert worden. Durch die Aufteilung des Programmes in einzelne Klassen ist es übersichtlich und erweiterbar und konnte bereits erfolgreich mit bis zu 512 Prozessoren eingesetzt werden.

Die Arbeit ist wie folgt aufgebaut. Kapitel 2 gibt eine Einführung in das Anwendungsfeld der Kite–Nummerierung. Die sphärische harmonische Analyse und die bei regelmässiger Verteilung der Messpunkte auftretenden Orthogonalitäten werden dargestellt. Datensätze, die diese Voraussetzungen erfüllen, mit Datensätzen, die diese Voraussetzungen nicht erfüllen, zu kombinieren ist nur begrenzt möglich, da auftretende Füllelemente in der Normalgleichungsmatrix die Ausnutzung der schwachen Besetztheit erschweren. Die Kite–Nummerierung dient dazu, das Auftreten dieser Füllelemente zu verhindern.
Das Freie Kite–Nummerierungschema wird in Kaptiel 3 vorgestellt. Die Form der Kitematrix wird definiert und der Algorithmus zur Aufstellung der Reihenfolge der unbekannten Parameter wird besprochen. Der Zusammenhang zwischen der Reihenfolge der Parameter und den Positionen der Nichtnullelemente in der Kitematrix wird durch einen Algorithmus realisiert.
Kapitel 4 behandelt die Bildung der Normalgleichungen sowohl für den vollbesetzten Fall, als auch für die schwach besetzte Kitematrix. Weiterhin werden Aspekte der Optimierung und Parallelisierung dieses aufwändigen Rechenschrittes beleuchtet. Die Entstehung von Füllelementen bei schwach besetzten Normalgleichungsmatrizen während der Cholesky Reduktion wird ausführlich behandelt, ebenso wie die an die Struktur der Kitematrix angepassten Algorithmen der Linearen Algebra.
Der Einsatz des Freien Kite–Nummerierungsschemas, sowohl bei der direkten Lösung, als auch bei der iterativen Lösung wird in Kapitel 5 dargestellt. Das Verfahren PCGMA wird detailliert erläutert, um anschliessend die neue Vorkonditionierungsstrategie vorstellen zu können, die durch das Freie Kite–Nummerierungschema möglich geworden ist. Am Ende des Kaptiels wird ein schematischer Überblick über die durchgeführte Umsetzung gegeben.
Die Ergebnisse von numerischen Untersuchungen zur Verbesserung der Konvergenz des Verfahrens PCGMA durch an die Daten angepasste Vorkonditionierung mit dem Freien Kite–Nummerierungsschema werden in Kapitel 6 gezeigt.
Die Arbeit wird mit einem Ausblick auf den zukünftigen Einsatz des Freien Kite–Nummerierungsschemas bei der GOCE–Datenauswertung und die Möglichkeiten zu weitergehenden Untersuchungen abgeschlossen.

# 2 Anwendungen

## 2.1 Sphärische harmonische Analyse

Für die mathematische Repräsentation vieler physikalischer Prozesse unseres *Systems Erde* werden die räumlichen Kugelfunktionen (*solid spherical harmonics*)

$$r^{-(\ell+1)}\, P_{\ell m}\left(\sin\varphi\right) \cos m\lambda \qquad \text{bzw.} \qquad r^{-(\ell+1)}\, P_{\ell m}\left(\sin\varphi\right) \sin m\lambda \tag{1}$$

als Basisfunktionen gewählt. Dabei geben $r, \varphi, \lambda$ die räumlichen Polarkoordinaten an und $P_{\ell m}\left(\sin\varphi\right)$ bilden die Legendre'schen Funktionen vom Grad $\ell$ und der Ordnung $m$ mit $\ell \geq 0$ und $m \leq \ell$. Als Lösung des Laplace'schen Operators für den Aussenraum der Kugel weisen diese Basisfunktionen einen globalen Träger auf, zeichnen sich aber speziell bei globaler kontinuierlicher Datenüberdeckung als Orthogonalsystem aus (Bernhard Hoffmann-Wellenhof und Helmut Moritz 2005 S. 21). Diese Basisfunktionen bilden eine vollständige Basis wodurch jede beliebige stückweise stetige Funktion $f\left(\varphi, \lambda\right)$ auf der Einheitskugel ($r = 1$) durch eine Linearkombination der Kugel(flächen)funktionen in einer unendlichen Reihe

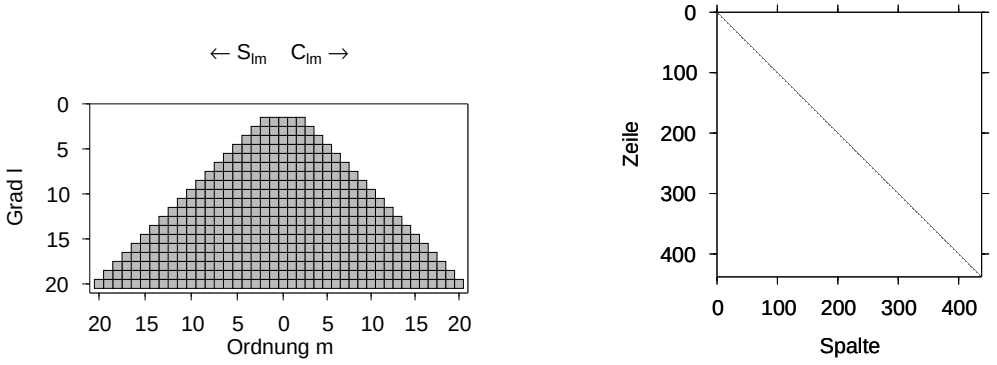
$$f\left(\varphi, \lambda\right) = \sum_{\ell=0}^{\infty} \sum_{m=0}^{\ell} C_{\ell m}\, P_{\ell m}(\sin\varphi) \cos m\lambda \;+\; S_{\ell m}\, P_{\ell m}(\sin\varphi) \sin m\lambda \tag{2}$$

mit $S_{\ell 0} = 0$, dargestellt werden kann. Die Koeffizienten $C_{\ell m}$ und $S_{\ell m}$ können unabhängig voneinander durch die Integration über die Einheitskugel durch

$$C_{\ell m} \;=\; \frac{1}{\iint\limits_{\sigma} \left(P_{\ell m}\left(\sin\varphi\right)\cos m\lambda\right)^2 d\sigma} \iint\limits_{\sigma} f(\varphi,\lambda)\, P_{\ell m}(\sin\varphi) \cos m\lambda \, d\sigma \tag{3}$$

$$S_{\ell m} \;=\; \frac{1}{\iint\limits_{\sigma} \left(P_{\ell m}\left(\sin\varphi\right)\sin m\lambda\right)^2 d\sigma} \iint\limits_{\sigma} f(\varphi,\lambda)\, P_{\ell m}(\sin\varphi) \sin m\lambda \, d\sigma \tag{4}$$

mit $d\sigma = \cos\varphi d\varphi d\lambda$, ermittelt werden. Zur Darstellung von band-begrenzten Funktionen genügt eine endliche Anzahl von Koeffizienten, so dass die Reihenentwicklung bei einem bestimmten Grad $\ell_{\max}$ abgebrochen werden kann. Um die Koeffizienten $C_{\ell m}$ und $S_{\ell m}$ übersichtlich anzuordnen, eignet sich eine Darstellung in Form von Koeffizientendreiecken (Abbildung 1). Die Ordinate weist von oben nach unten auf den zunehmenden Grad $\ell$ hin. Auf der Abszisse sind die Ordnungen $m$ aufgetragen, wobei nach rechts die Kosinuskoeffizienten $C_{\ell m}$ und nach links die Sinuskoeffizienten $S_{\ell m}$ aufgetragen sind.




(a) Koeffizientendreieck (b) Normalgleichungen

Abbildung 1: Darstellung von Koeffizienten $C_{\ell m}$ und $S_{\ell m}$ und deren Korrelationen

Die Abbildung 1(b) zeigt die Korrelationen zwischen den Koeffizienten, die durch die Nicht-Null Einträge in den Normalgleichungen darstellbar sind. Die Diagonalstruktur weist darauf hin, dass alle Koeffizienten unabhängig voneinander berechnet werden können. Die Berechnung der Koeffizienten $C_{\ell m}$ und $S_{\ell m}$, auch *sphärische harmonische Analyse* genannt, führt in diesem Fall auf eine einfach zu lösende inverse Aufgabe.
Diese auf die Orthogonalitätsbedingungen der Basisfunktionen zurück zu führenden Eigenschaften werden jedoch gestört, wenn anstelle der kontinuierlichen Funktionen nur Funktionswerte an diskreten Punkten zur Verfügung

stehen. Die Berechnung der Koeffizienten $C_{\ell m}$ und $S_{\ell m}$ führt zu einem Ausgleichungsproblem mit einem voll besetzten Normalgleichungssystem.

Somit kann die Berechnung der einzelnen Komponenten nicht mehr isoliert betrachtet werden. Die vorhandenen Korrelationen zwischen allen Koeffizienten erfordern die Lösung eines vollbesetzten Systems. Eine Darstellung der Abhängigkeiten wie in Abbildung 1(b) würde zu einer vollbesetzten Matrix führen. Durch die Beachtung zusätzlicher Bedingungen zur örtlichen Verteilung der diskreten Punkte können einzelne Orthogonalitäten auch bei diskreten Punkten erhalten bleiben. Im Speziellen können die Orthogonalitätsbeziehungen zwischen den trigonometrischen Funktionen genutzt werden. Liegt eine Abtastung an $2L$ äquidistanten Stützstellen über die volle Periode $2\pi$ vor, $\lambda_i = i\frac{2\pi}{2L}$, dann gilt

$$\sum_{i=0}^{2L-1} \cos m\lambda_i \, \cos k\lambda_i = (1 + \delta_{m0} + \delta_{mL}) \, L \, \delta_{mk} \tag{5}$$

$$\sum_{i=0}^{2L-1} \sin m\lambda_i \, \sin k\lambda_i = (1 - \delta_{m0} - \delta_{mL}) \, L \, \delta_{mk} \tag{6}$$

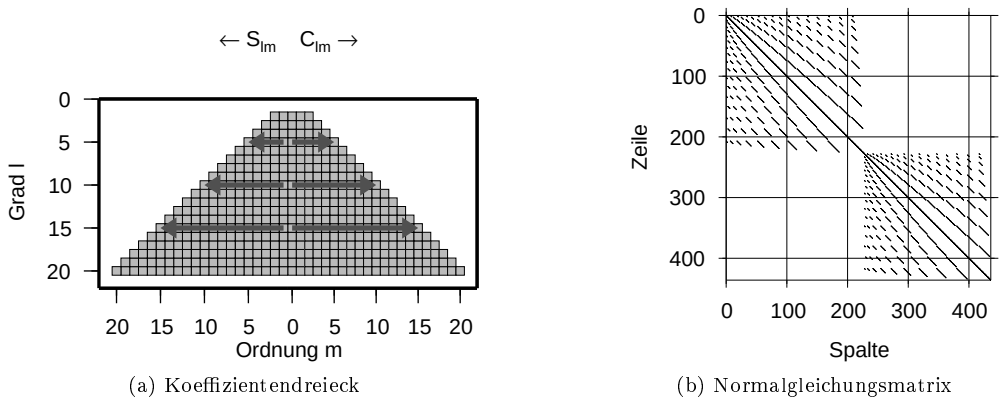
$$\sum_{i=0}^{2L-1} \cos m\lambda_i \, \sin k\lambda_i = 0 \tag{7}$$

mit dem Kronecker Symbol $\delta_{mk}$. Unter Berücksichtigung der Orthogonalitäten (5) und (6) werden alle Koeffizienten mit unterschiedlichen Ordnungen $m$ voneinander unabhängig und aufgrund der Orthogonalität (7) werden die Sinuskoeffizienten von den Kosinuskoeffizienten unabhängig. Des weiteren können noch Symmetrie- und Antisymmetrieeigenschaften der Legendre'schen Funktionen bezüglich des Äquators ausgenutzt werden

$$P_{\ell m}(-sin\varphi) = (-1)^{(\ell - m)} P_{\ell m}(sin\varphi) \; , \tag{8}$$

wodurch innerhalb einer Ordnung die Koeffizienten mit geradem Grad von den Koeffizenten mit ungeradem Grad unabhängig werden.

Vielfach wurden die Parameter in der Vergangenheit gradweise sortiert, wie z. B. im EGM 96 (Lemoine et al. 1996), weshalb diese Nummerierung im folgenden mit Standardnummerierung bezeichnet wird. Meistens werden die Kosinuskoeffizienten $C_{\ell m}$ vorweg nummeriert und anschließend die Sinuskoeffizienten $S_{\ell m}$ bei leicht geänderter Schleifenstruktur (ohne Koeffizienten $m = 0$) nachgeführt. Die Reihenfolge der Koeffizienten bei dieser Nummerierung kann durch eine einfache Schleifenkonstruktion veranschaulicht werden (Algorithmus 2.1).




(a) Koeffizientendreieck (b) Normalgleichungsmatrix

Abbildung 2: Standardschema

**Algorithmus 2.1 (Standardschema)**

```
1  #include "parameter.h"
2
3  void standardschema_reihenfolge( vector<Parameter> &stack, int l_max )
4  {
5     int m_max = l_max;
6
7     for( int l = 0; l <= l_max; ++l )
8        for( int m = 0; m <= l; ++m )
9           stack.push_back( Parameter( 'C', l, m ) );    // Kosinuskoeffizienten
10
11    for( int l = 1; l <= l_max; ++l )
12       for( int m = 1; m <= l; ++m )
13          stack.push_back( Parameter( 'S', l, m ) );    // Sinuskoeffizienten
14 }
```

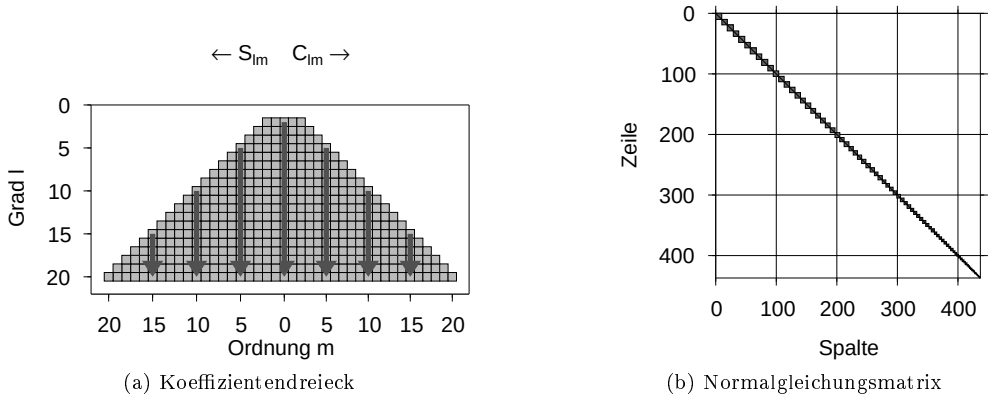
Die Reihenfolge der Parameter, die der Algorithmus 2.1 erzeugt, ist nun für $\ell_{\min} = 0$ und $\ell_{\max} = 4$ dargestellt.

$$C_{00}\ C_{10}\ C_{11}\ C_{20}\ C_{21}\ C_{22}\ C_{30}\ C_{31}\ C_{32}\ C_{33}\ C_{40}\ C_{41}\ C_{42}\ C_{43}\ C_{44}\ S_{11}\ S_{21}\ S_{22}\ S_{31}\ S_{32}\ S_{33}\ S_{41}\ S_{42}\ S_{43}\ S_{44}$$

Wie in Abbildung 2(a) dargestellt, entspricht diese Nummerierungsmethode einer primären horizontalen Bewegungsrichtung. Das sich aus dieser Parametersortierung ergebende Besetztheitsmuster zeigt diagonale Streifen parallel zur Hauptdiagonalen (Abbildung 2(b)).

Die Organisation der Speicherung dieser schwach besetzten Matrix wäre entsprechend aufwändig, kann aber wesentlich vereinfacht werden, indem die Parameter ordnungsweise nummeriert werden. Dies entspricht einer vertikalen Bewegungsrichtung im Koeffizientenschema (Abbildung 3(a)).

Diese Nummerierungsart nachfolgend als *Blockschema* bezeichnet, erzeugt eine leicht zu speichernde Besetztheitsstruktur der Normalgleichungsmatrix. Zudem entstehen bei der Choleskyreduktion keine Füllelemente. Die




(a) Koeffizientendreieck (b) Normalgleichungsmatrix

Abbildung 3: Blockschema: Koeffizientendreieck und Korrelationen bei rotationssymmetrischer Punktverteilung

Reihenfolge der Koeffizienten im *Blockschema* ist in Algorithmus 2.2 dargestellt, wobei hier auch die Symmetrie bezüglich des Äquators genutzt wird. Somit erfolgt innerhalb der Ordnungen eine weitere Aufteilung zwischen Koeffizienten mit geraden und ungeraden Graden.

**Algorithmus 2.2 (Blockschema)**

```
1  #include "parameter.h"
2
3  void blockschema_reihenfolge( vector<Parameter> &stack, int l_max )
4  {
5     for( int m = 0; m <= l_max; m++ )
6     {
7        // C_lm anordnen
8        for( int l = m; l <= l_max; l += 2 )
9           stack.push_back( Parameter( 'C', l, m ) );  // gerade/ungerade Koeffizienten
10
11       for( int l = m+1; l <= l_max; l += 2 )
12          stack.push_back( Parameter( 'C', l, m ) );  // ungerade/gerade Koeffizienten
13
14       // S_lm anordnen
15       if( m != 0 )
16       {
17          for( int l = m; l <= l_max; l += 2 )
18             stack.push_back( Parameter( 'S', l, m ) );  // gerade/ungerade Koeffizienten
19
20          for( int l = m+1; l <= l_max; l += 2 )
21             stack.push_back( Parameter( 'S', l , m ) ); // ungerade/gerade Koeffizienten
22       }
23    }
24 }
```

Die Reihenfolge der Parameter, die der Algorithmus 2.2 erzeugt, ist nun für $\ell_{\min} = 0$ und $\ell_{\max} = 4$ dargestellt.

$$C_{00}\ C_{20}\ C_{40}\ C_{10}\ C_{30}\ C_{11}\ C_{31}\ C_{21}\ C_{41}\ S_{11}\ S_{31}\ S_{21}\ S_{41}\ C_{22}\ C_{42}\ C_{32}\ S_{22}\ S_{42}\ S_{32}\ C_{33}\ C_{43}\ S_{33}\ S_{43}\ C_{44}\ S_{44}$$

Anzumerken ist jedoch, dass die Blockdiagonalstruktur nur erhalten bleibt, wenn alle Datenpunkte rotationssymmetrisch bezüglich der Nord-Süd-Achse in regelmäßigen Abständen entlang der Parallelkreise verteilt sind und Symmetrie bezüglich des Äquators vorliegt. Die Datendichte auf jedem Parallelkreis kann mit unterschiedlichen Gitterweiten gewählt werden. Es ist darauf zu achten, dass diese Datenpunkte mit homogener Genauigkeit vorliegen. Polare Löcher verschlechtern zwar die Kondition des Gesamtsystems, haben aber keinen Einfluss auf die blockdiagonale Struktur. Die Unabhängigkeit zwischen den Ordnungen erlaubt die Einzelauswertung Ordnung für Ordnung. Durch diese effiziente Form der sphärischen harmonischen Analyse können auch sehr hochauflösende Modelle berechnet werden (Colombo 1981; Rummel et al. 1993). Vielfach wird diese blockorientierte Methode als *fast spherical harmonic analysis* bezeichnet.

**Ausnutzung der Symmetrie bezüglich des Äquators:** Da alle Koeffizienten mit geraden Graden unabhängig von denen mit ungeraden Graden werden, wenn die Datenpunkte symmetrisch bezüglich des Äquators verteilt sind, entstehen mit Ausnahme der Ordnung Null im Blockschema für jede Ordnung vier Diagonalblöcke (Abbildung 4).

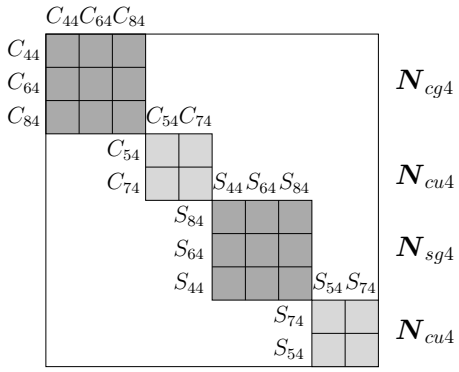



Abbildung 4: Die vier Diagonalblöcke zur Ordnung $m = 4$ bei $\ell_{\max} = 8$

Diese vier Blöcke zur Ordnung $m$ werden im folgenden mit $\boldsymbol{N}_{cgm}$, $\boldsymbol{N}_{cum}$, $\boldsymbol{N}_{sgm}$ und $\boldsymbol{N}_{sum}$ bezeichnet, wobei die Indizes $c$ und $s$ Kosinus und Sinus und die Indizes $g$ und $u$ gerade, bzw. ungerade Grade $\ell$ bedeuten. Wie aus Abbildung 4 zu ersehen ist, enthält der Teil $\boldsymbol{x}_{cg4}$ des Parametervektors $\boldsymbol{x}$ die Koeffizienten $C_{44}, C_{64}, C_{84}$.

**Fazit:** Liegen Messdaten für die sphärische harmonische Analyse auf einem geographischen Gitter vor, bei dem die Punkte gleichabständig und gleichgenau entlang der Breitenkreise verteilt sind, fallen die Korrelationen zwischen den Koeffizienten ungleicher Ordnung weg und die Normalgleichungsmatrix ist schwach besetzt. Mit einer ordnungsweisen Nummerierung der Parameter ist die Normalgleichungsmatrix blockdiagonal und kann daher sehr leicht gespeichert werden. Sind die Breitenkreise, auf denen die Messpunkte liegen zudem symmetrisch bezüglich des Äquators angeordnet, so werden die Koeffizienten mit ungeraden Graden unabhängig von denen mit geraden Geraden, so dass weitere Korrelationen entfallen. Durch Gruppierung der entsprechenden Koeffizienten innerhalb der Ordnungen werden die Diagonalblöcke noch kleiner und es entstehen zu jeder Ordnung, ausser der Ordnung Null und der Ordnung $m = \ell_{\max}$, vier Diagonalblöcke. Ein entsprechender Datensatz wird *regelmäßiger Datensatz* genannt und kann unter Verwendung des *Blockschemas* sehr effizient ausgewertet werden.

## 2.2 Regelmäßige und unregelmäßige Daten im Vergleich

In diesem Abschnitt wird untersucht, welche Auswirkung die Eigenschaft der Regelmäßigkeit eines Datensatzes auf die Normalgleichungsmatrix, sowohl bezüglich des Speicherplatzbedarfs als auch bezüglich der Verarbeitungszeit hat. Dazu wird von einem hypothetischen Datensatz ausgegangen, der zur sphärischen harmonischen Analyse des Schwerefeldes bis zum maximalen Entwicklungsgrad von 360 dienen soll. Zur Hervorhebung des Unterschiedes zwischen regelmäßigen Daten und nicht regelmäßigen Daten wird zwischen zwei Fällen unterschieden; im ersten Fall ist der Datensatz regelmäßig und im zweiten Fall nicht. Der Speicherplatzbedarf und die benötigte Rechenzeit für die Choleskyzerlegung der Normalgleichungsmatrix werden gegenübergestellt.

**Beispiel 2.1** *Gegeben sei ein globaler Datensatz, der dazu dienen soll, eine sphärische harmonische Analyse des Erdschwerefeldes bis zum Grad und zur Ordnung 360 durchzuführen. Die Anzahl der unbekannten Parameter ergibt sich durch*

$$n = (\ell_{\max} + 1)^2, \tag{9}$$

*daraus ergeben sich 130321 unbekannte Parameter bei dem maximalen Entwicklungsgrad von 360. Zum Vergleich sollen zwei Fälle betrachtet werden*

*a) Der Datensatz ist regelmäßig und hat daher bei der Verwendung des Blockschemas eine blockdiagonale Struktur.*

*b) Der Datensatz ist nicht regelmäßig und die Normalgleichungsmatrix ist vollbesetzt.* ■

**Gegenüberstellung des Speicherplatzbedarfs:** Im Fall a) enthält die Normalgleichungsmatrix vier Diagonalblöcke für die Ordnungen 1 bis 359 und zwei Diagonalblöcke für die Ordnungen 0 und 360. In Ordnung 0 gibt es keine Sinuskoeffizienten und in Ordnung 360 keine ungeraden Koeffizienten, sondern nur die beiden Koeffizienten $C_{360\,360}$ und $S_{360\,360}$. Das Speichern der Normalgleichungsmatrix kann z. B. sehr leicht in einem Vektor von Matrizen erfolgen. Die Blöcke werden mit zunehmender Ordnung immer kleiner. Der Block $\boldsymbol{N}_{cgm}$ der Ordnung $m$ enthält die Normalgleichungselemente zu den Kosinuskoeffizienten deren Grad $\ell$ gerade ist. Er hat eine Seitenlänge von $(\ell_{\max} - m)/2 + 1$, wenn $\ell_{\max}$ und $m$ gerade sind, $(\ell_{\max} - m)/2$, wenn $\ell_{\max}$ und $m$ ungerade sind, und $(\ell_{\max} - m + 1)/2$, wenn $\ell_{\max}$ und $m$ ungleiche Parität haben. Der Block $\boldsymbol{N}_{cum}$ der Ordnung $m$ hat die gleiche Seitenlänge wie $\boldsymbol{N}_{cgm}$, wenn die Parität von $\ell_{\max}$ und $m$ unterschiedlich ist, ansonsten ist er ein Element größer, wenn $\ell_{\max}$ und $m$ beide ungerade sind bzw. ein Element kleiner, wenn $\ell_{\max}$ und $m$ beide gerade sind. Die Seitenlänge eines Normalgleichungsblocks hängt bei einem regelmäßigen Datensatz daher von dem Betrag und der Parität von $\ell, m$ und $\ell_{\max}$ ab. Die Seitenlänge eines beliebigen Normalgleichungsblocks kann mit Hilfe der Tabelle 1 bestimmt werden. Die Sinusblöcke sind so groß wie die Kosinusblöcke. Der gesamte Speicherplatzbedarf für die blockweise Speicherung der Normalgleichungsmatrix im Falle des regelmäßigen Datensatzes ( 2.1 a)) bis Grad und Ordnung 360 läßt sich mit Hilfe von Tabelle 1 und eines kleinen Computerprogramms berechnen. Dazu wird die Funktion `count_elements`$(\ell_{\max}, m)$ eingesetzt, die die Anzahl der Normalgleichungseinträge für die Ordnung $m$ bei einem maximalen Grad von $\ell_{\max}$ berechnet (Algorithmus 2.3).

| $\ell_{\max}$ | $m$ | $l$ | $\#C_{\ell m}$ | $\#S_{\ell m}$ |
|---|---|---|---|---|
| gerade | gerade | gerade | $(\ell_{\max}-m)/2+1$ | $(\ell_{\max}-m)/2+1$ |
| | | ungerade | $(\ell_{\max}-m)/2$ | $(\ell_{\max}-m)/2$ |
| | ungerade | gerade | $(\ell_{\max}-m+1)/2$ | $(\ell_{\max}-m+1)/2$ |
| | | ungerade | $(\ell_{\max}-m+1)/2$ | $(\ell_{\max}-m+1)/2$ |
| ungerade | gerade | gerade | $(\ell_{\max}-m+1)/2$ | $(\ell_{\max}-m+1)/2$ |
| | | ungerade | $(\ell_{\max}-m+1)/2$ | $(\ell_{\max}-m+1)/2$ |
| | ungerade | gerade | $(\ell_{\max}-m)/2$ | $(\ell_{\max}-m)/2$ |
| | | ungerade | $(\ell_{\max}-m)/2+1$ | $(\ell_{\max}-m)/2+1$ |

Tabelle 1: Seitenlängen der Normalgleichungsblöcke im Blockschema

**Algorithmus 2.3 (Funktion `count_elements`)**

```
1  int count_elements( int l_max, int m )
2  {
3     int n;
4
5     if( (l_max % 2) == (m % 2) )
6        // Blöcke sind unterschiedlich groß.
7        n = power( (l_max - m)/2 + 1, 2 ) + power( (l_max - m)/2, 2 );
8     else
9        // Blöcke sind gleich groß.
10       n = 2 * power( (l_max -m + 1) / 2, 2 );
11
12    if( m != 0 )
13       // Sinuskoeffizienten sind vorhanden.
14       n = n * 2;
15
16    return n;
17 }
18
```

Das Computerprogramm summiert die Anzahl der Einträge für alle Ordnungen auf und das Ergebnis ist die Anzahl der Nichtnullelemente der Normalgleichungsmatrix bei Vorliegen eines regelmäßigen Datensatzes. Abbildung 5(a) zeigt die Größen der Normalgleichungsblöcke bei Beispiel 2.1; die Gesamtanzahl der Nichtnullelemente ist 15 682 201. Um diese Normalgleichungsmatrix im Datentyp *double* der Programmiersprache C zu speichern sind $15\,682\,201 \cdot 8/1024^2 = 120$ Megabyte notwendig.

In Fall b) ist die Normalgleichungsmatrix voll besetzt und daher muss eine $(\ell_{\max}+1)^2 \times (\ell_{\max}+1)^2$ Matrix mit $\ell_{\max} = 360$ an Speicher reserviert werden. Dies sind

$$\frac{(\ell_{\max}+1)^2 \cdot (\ell_{\max}+1)^2}{1024^3} 8\,[\text{byte}] \approx 127\,\text{GB, für den Fall } \ell_{\max} = 360$$

Dies entspricht ungefähr dem tausendfachen Speicherplatzbedarf des regelmäßigen Datensatz aus Fall a).

**Gegenüberstellung des Rechenaufwandes:** Die Lösung des Gleichungssystems wird, falls keine Kovarianzmatrix gefordert ist, mit Hilfe der Choleskyzerlegung der Normalgleichungsmatrix berechnet. Zur Choleskyzerlegung der schwach besetzten Normalgleichungsmatrix müssen die Diagonalblöcke einzeln berechnet werden. Die Anzahl der Rechenoperationen für die Choleskyzerlegung ist $1/6\,n^3 + n^2 c + O(n^2, c)$. Für die Abschätzung des Rechenaufwandes sind die Terme kleiner als Ordnung 3 vernachlässigbar. Zur Berechnung der notwendigen Operationen wird die Funktion `count_operations` (Algorithmus 2.4) von der Funktion `count_elements` (Algorithmus 2.3) abgeleitet. Sie berechnet die Anzahl der Operationen für die Choleskyzerlegung der einzelnen Diagonalblöcke der Normalgleichungsmatrix. Für Fall a) sind dies $353\,834\,970 \approx 3.5 \cdot 10^8$ Operationen.

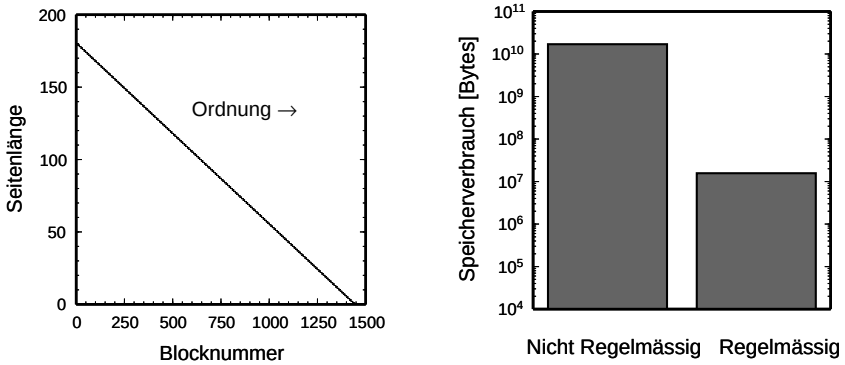
**Algorithmus 2.4 (Funktion `count_operations`)**

```
1  int count_operations( int l_max, int m )
2  {
3     int n;
4
5     if( (l_max % 2) == (m % 2) )
6        // Blöcke sind unterschiedlich groß.
7        n = power( (l_max - m)/2 + 1, 3 ) + power( (l_max - m)/2, 3 );
8     else
9        // Blöcke sind gleich groß.
10       n = 2 * power( (l_max -m + 1) / 2, 3 );
11
12    if( m != 0 )
13       // Sinuskoeffizienten sind vorhanden.
14       n = n * 2;
15
16    return n/6.0;
17 }
18
```

Die Differenz der Anzahl der Rechenoperationen der beiden Fälle ist noch größer, als der Unterschied bezüglich des Speicherplatzbedarfs. Für die Choleskyzerlegung der Normalgleichungsmatrix in Fall b sind

$$\frac{1}{6}((\ell_{\max}+1)^2)^3 \approx 10^{12} \text{ Operationen, für den Fall } \ell_{\max} = 360$$

notwendig. Dies sind dreitausend mal mehr Operationen als bei dem regelmäßigen Datensatz. Der Speicherverbrauch von Fall a und Fall b ist in Abbildung 5(b) gegenübergestellt.




(a) Blockgrößen mit zunehmender Ordnung

(b) Speicherplatzverbrauch

Abbildung 5: Blockgrößen und Speicherverbrauch

**Abschätzung der maximal erreichbaren Auflösung:** Bei einem maximalen Entwicklungsgrad von 360 wird für die Auswertung eines nicht regelmäßigen Datensatzes tausend mal mehr Speicherplatz benötigt als für die Auswertung eines regelmäßigen Datensatzes. Das bedeutet, mit einem Computer, der ca. 120 MB freien Arbeitsspeicher hat, könnte eine sphärische harmonische Analyse bis Grad und Ordnung 360 durchgeführt werden, wenn der Datensatz regelmäßig ist, im vollbesetzten Fall ist der maximale Entwicklungsgrad auf

$$\sqrt[4]{\frac{120 \cdot 1024^2}{8}} - 1 \approx 62$$

beschränkt. Die mit zunehmender Speichermenge fortschreitende Diskrepanz zwischen dem maximal möglichen Entwicklungsgrad bei regelmäßigen und unregelmäßigen Datensätzen zeigt Abbildung 6.
Die Angaben beziehen sich lediglich auf die Speicherung der Normalgleichungsmatrix. In einer tatsächlichen Berechnung muss geringfügig mehr Speicher veranschlagt werden, da die rechte Seite und der Lösungsvektor ebenfalls gespeichert werden müssen. Diese sind im Verhältnis zur Normalgleichungsmatrix jedoch vernachlässigbar klein.

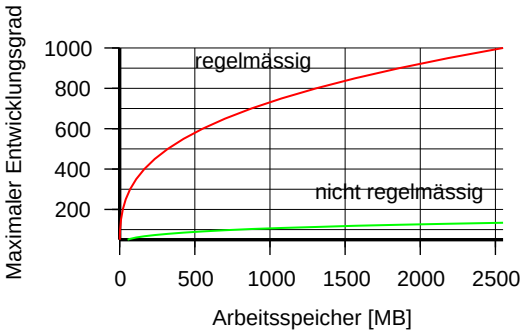



Abbildung 6: Maximal möglicher harmonischer Entwicklungsgrad im Verhältnis zum Arbeitsspeicher

Wird also eine sphärische harmonische Analyse bis zum Grad und Ordnung 360 durchgeführt, so ist das Verhältnis des Speicherplatzverbrauchs 1:1000 und der Rechenzeit 1:3000, wenn der zugrunde liegende Datensatz nicht regelmäßig ist. Daher ist es mit einem heutigen normal ausgestatteten Computer mit zwei Gigabyte Arbeitsspeicher möglich, eine volle Normalgleichungsmatrix bis zum maximalen Entwicklungsgrad von ungefähr 130 zu verarbeiten. Ist der Datensatz jedoch regelmäßig kann man, wie in Abbildung 6 abzulesen, bis Grad und Ordnung 900 auflösen.

### 2.2.1 Näherungsweise regelmäßige Daten

Im letzten Abschnitt wurde gezeigt, dass die Verteilung und das Gewicht der Datenpunkte eine entscheidende Rolle für die sphärische harmonische Analyse spielen. Aus dem Verhältnis der Speicheranforderungen zwischen einem regelmäßigen und einem unregelmäßigen Datensatz von 1 : 1000 folgt, dass Auflösungen über 200 nur schwer über Normalgleichungen lösbar sind. Sind die verfügbaren Daten nicht regelmäßig, so ist eine höhere Auflösung über ein iteratives Verfahren berechenbar, welches auf die Aufstellung der Normalgleichungen verzichtet, wie z. B. das Verfahren der Konjugierten Gradienten in der von Schwarz (1970) für die Ausgleichungsrechnung modifizierten Form. Zur Beschleunigung der Konvergenz benutzt man bei iterativen Verfahren häufig eine Vorkonditionierungsmatrix, die der unbekannten Normalgleichungsmatrix möglichst ähnlich ist. Bei Messungen der Satellitengradiometrie (SGG) kann aufgrund der vollständigen, globalen Überdeckung mit hoher Punktdichte die Regelmäßigkeit angenommen werden. Durch die Vernachlässigung der Korrelationen zwischen Parametern ungleicher Ordnung, ungleicher trigonometrischer Funktion und unterschiedlicher Parität wird eine Approximation der Normalgleichungsmatrix erhalten, die zur Vorkonditionierung im CG Verfahren geeignet ist. Abbildung 7 zeigt eine zweidimensionale, logarithmische Darstellung eines Ausschnitts aus einer SGG Normalgleichungsmatrix mit dem maximalen Entwicklungsgrad von 12, welche aufgrund von simulierten GOCE–Daten berechnet worden ist. Die gut zu erkennende blockweise diagonaldominante Struktur zeigt, dass die Annahme der näherungsweisen Regelmäßigkeit zulässig ist. Die Annahme der symmetrischen Verteilung der Datenpunkte bezüglich des Äquators, ausgedrückt durch (8) ist in diesem Fall jedoch nicht ganz richtig, da noch deutliche Korrelationen zwischen den geraden und den ungeraden Koeffizienten innerhalb einer Ordnung und gleicher trigonometrischer Funktion zu erkennen sind. In Kapitel 6 wird eine Beschreibung der verwendeten Testdaten angegeben und anhand von Beispielrechnungen die Eignung der näherungsweisen SGG Normalgleichungsmatrix zur Vorkonditionierung gezeigt.
Ein Datensatz, der sich aufgrund der näherungsweise erfüllten Orthogonalitäten (5)-(8) zur Vorkonditionierung eignet, soll im folgenden mit *näherungsweise regelmäßiger Datensatz* bezeichnet werden.

## 2.3 Kombination von regelmäßigen und unregelmäßigen Daten

Seit der Einführung von Satellitenmessungen zur Erdschwerefeldbestimmung besteht der Wunsch der Datenkombination von Satellitendaten mit regelmäßigen Datensätzen. Während die zumeist unregelmäßig verteilten Satellitendaten die Koeffizienten mit niedrigeren Graden sehr gut bestimmen, dienen die auf regelmäßigen Gittern bereitgestellten Daten (z. B. Schwereanomalien) der präzisen Bestimmung der Koeffizienten hoher Grade. Bei einfacher Modellbildung wurde bisher mit einer *Patchwork*-Technik gearbeitet so dass Berechnung und Modellverfeinerung (siehe, z. B. OSU91 (Rapp et al. 1991), EGM96 (Lemoine et al. 1996)) schrittweise erfolgte. Bei komplexen Modellen wurde die Korrelation innerhalb der Ordnungen streng berücksichtigt (Balmino

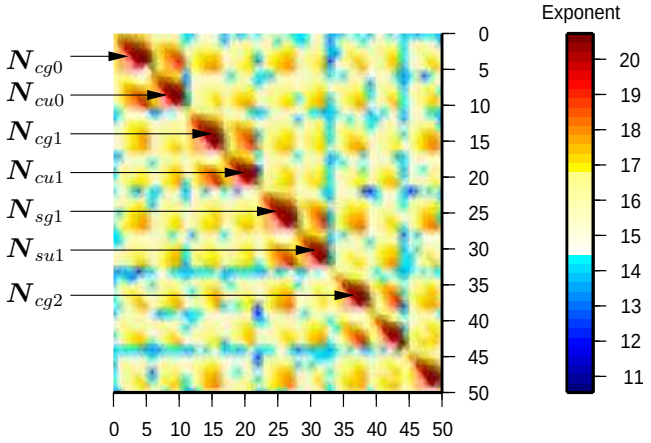



Abbildung 7: Ausschnitt aus einer Normalgleichungsmatrix eines näherungsweise regelmäßigen Datensatzes aufgelöst bis Grad und Ordnung 12

1993). Dazu wurden die Koeffizienten des hochauflösenden Datensatzes im Blockschema angeordnet, so dass eine blockdiagonale Normalgleichungsmatrix entstand. Zur Kombination beider Datensätze wurden die Elemente des niedriger auflösenden, aber vollen Datensatzes, in die große Normalgleichungsmatrix einsortiert. Dies führte jedoch zu einem enormen Anwachsen des Speicherplatzbedarfs für die Kombination. Dieser Vorgang soll nun an Beispiel 2.2 erläutert werden.

**Beispiel 2.2 (Datenkombination)** *Zur globalen sphärischen harmonischen Analyse sind zwei Normalgleichungssysteme aufgestellt worden. Gleichungssystem $h$* (hochauflösend) *ist aus einem regelmäßigen, globalen Datensatz entstanden und enthält als unbekannte Parameter $\boldsymbol{x}_h$ Schwerefeldkoeffizienten bis Grad und Ordnung $\ell_{\max_h} = 360$*

$$\boldsymbol{N}_h \, \boldsymbol{x}_h = \boldsymbol{n}_h \ , \tag{10}$$

*wobei die Normalgleichungsmatrix $\boldsymbol{N}_h$ aus der Designmatrix $\boldsymbol{A}_h$ durch $\boldsymbol{N}_h = \boldsymbol{A}_h^T \boldsymbol{A}_h$ gebildet worden ist. Durch die Orthogonalitäten, die aufgrund der regelmäßigen Datenverteilung vorliegen, ist die Normalgleichungsmatrix $\boldsymbol{N}_h$ eine Blockdiagonalmatrix mit vier Diagonalblöcken pro Ordnung $m \neq 0$ und zwei Diagonalblöcken für die Ordnungen $m = 0$ und $m = 360$. Die einzelnen Blöcke der Normalgleichungsmatrix werden mit $\boldsymbol{N}_{tpm}$ bezeichnet wobei der Index $t$ (trigonometrische Funktion) die Buchstaben $c$ und $s$ und der Index $p$ (Parität) die Buchstaben $g$ (gerade) und $u$ (ungerade) annehmen kann (siehe Abbildung 4). Der dritte Index bezeichnet die Ordnung $m$ und nimmt Werte zwischen 0 und $\ell_{\max_h}$ an. Analog dazu werden die Spalten in $\boldsymbol{A}_h$ zu Blockspalten $\boldsymbol{A}_{tpm}$ zusammengefasst (siehe Abbildung 8).*
*Das zweite Normalgleichungssystem $n$* (niedrige Auflösung) *ist aus einem nicht regelmäßigen, globalen Datensatz entstanden und enthält als unbekannte Parameter $\boldsymbol{x}_n$ Schwerefeldkoeffizienten bis Grad und Ordnung $\ell_{\max_n} = 50$*

$$\boldsymbol{N}_n \boldsymbol{x}_n = \boldsymbol{n}_n \ . \tag{11}$$

*Die Normalgleichungsmatrix $\boldsymbol{N}_n$ ist aus der Designmatrix $\boldsymbol{A}_n$ durch $\boldsymbol{N}_n = \boldsymbol{A}_n^T \boldsymbol{A}_n$ hervorgegangen. Da die Orthogonalitäten zwischen den Kugelfunktionen durch die nicht regelmäßige Datenverteilung gestört werden, ist die Normalgleichungsmatrix $\boldsymbol{N}_n$ eine vollbesetzte Matrix.* ■

Zur Schätzung der Parameter $\boldsymbol{x}$, die die Vereinigungsmenge der Parameter $\boldsymbol{x}_n$ und $\boldsymbol{x}_h$ darstellen, sollen die Normalgleichungssysteme $h$ und $n$ in einer gemeinsamen Ausgleichung miteinander kombiniert werden. Die Schätzwerte der unbekannten Parameter ergeben sich bei dieser Kombination (siehe z. B. Koch (1999, S. 177)) zu

$$\begin{aligned} \tilde{\boldsymbol{x}} &= \left( \boldsymbol{A}_h^T \boldsymbol{A}_h + \bar{\boldsymbol{A}}_n^T \bar{\boldsymbol{A}}_n \right)^{-1} \left( \boldsymbol{A}_h^T \boldsymbol{\ell}_h + \bar{\boldsymbol{A}}_n^T \bar{\boldsymbol{\ell}}_n \right) \\ &= \left( \boldsymbol{N}_h + \bar{\boldsymbol{N}}_n \right)^{-1} \left( \boldsymbol{n}_h + \bar{\boldsymbol{n}}_n \right) \ , \end{aligned} \tag{12}$$

wobei $\boldsymbol{N}_n$, $\boldsymbol{A}_n$ und $\boldsymbol{\ell}_n$ mit Nullzeilen und Spalten erweitert wurden, damit die Gleichungssysteme $h$ und $n$ die gleiche Dimension haben und somit die Vektor- und Matrixaddition definiert sind. Die aus dieser Erweiterung resultierenden Vektoren und Matrizen sind mit einem Überstrich gekennzeichnet.

**Zur Größenordnung der Systeme:** Die Gleichungssysteme in Beispiel 2.2 haben sehr unterschiedliche Größenordnungen, das Gleichungssystem $h$ hat $361^2 = 130321$ unbekannte Parameter, demnach hat die Normalgleichungsmatrix $\boldsymbol{N}_h$ die Dimension $130321 \times 130321$. Da sie jedoch eine blockdiagonale Struktur hat, werden für ihre Speicherung nur ca. 120 Megabyte benötigt. Das Gleichungssystem $n$ hat $51^2 = 2601$ unbekannte Parameter, demnach hat die Normalgleichungsmatrix $\boldsymbol{N}_n$ die Dimension $2601 \times 2601$. Da sie im Gegensatz zur Normalgleichungsmatrix $\boldsymbol{N}_h$ vollbesetzt ist, benötigt ihre Speicherung $(2601^2 \cdot 8)/1024^2 = 51$ Megabyte.
Für die Gleichungssysteme $h$ und $n$ zusammen werden damit ca. 170 Megabyte benötigt, was heutzutage mit jedem Computer zu bewältigen ist. Wäre die Normalgleichungsmatrix $\boldsymbol{N}_h$ jedoch nicht schwach besetzt und hätte sie nicht eine solche Blockdiagonalstruktur, so belegte sie $(130321^2 \cdot 8)/1024^3 = 126$ Gigabyte. Dies wäre auf einem einem heutigen Computer nicht zu bewältigen. Durch die Ausnutzung der Orthogonalitäten ist die benötigte Menge an Speicherplatz um den Faktor 1080 reduziert worden.

**Das Zuordnungsproblem:** Durch die Reihenfolge, in die das Blockschema die Parameter in dem Gleichungssystem $h$ bringt, werden die Parameter des Gleichungssystems $n$ auseinandergezogen. Dadurch wächst die Speicheranforderung wieder so stark an, dass die Ausnutzung der Orthogonalitäten nur noch einen geringen Vorteil bietet. Um dies zu erläutern, wird zunächst die Designmatrix $\boldsymbol{A}_h$ betrachtet. Sie besteht aus Blockspalten $\boldsymbol{A}_{tpm}$ mit $t \in \{c, s\}$, $p \in \{g, u\}$ und $m \in \{0, \cdots, 360\}$ (Abbildung 8).

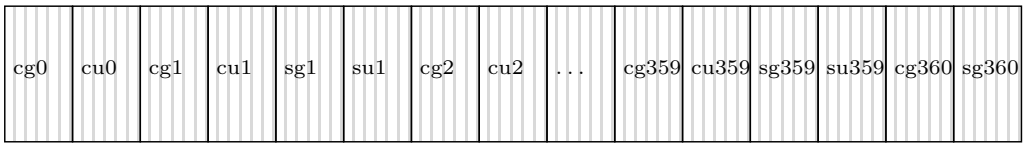
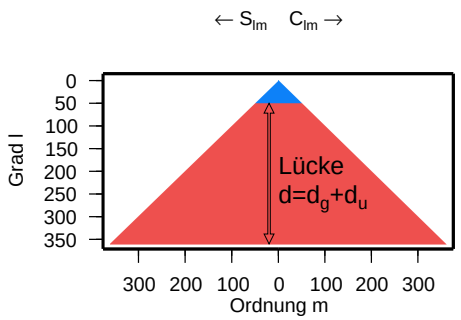
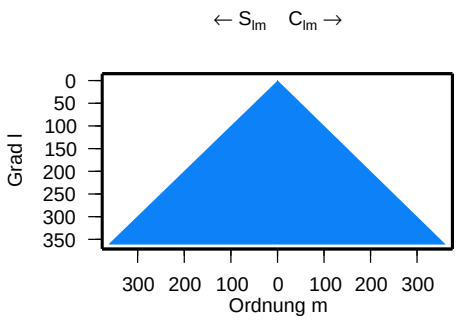



Abbildung 8: Schematische Darstellung der Einteilung der Designmatrix $\boldsymbol{A}_h$ in Blockspalten $\boldsymbol{A}_{tpm}$. In der Abbildung sind nur die Indizes $tpm$ der einzelnen Blockspalten dargestellt.

Alle Vektoren der Blockspalte $\boldsymbol{A}_{t_i p_j m_k}$ sind aufgrund der bei den Kugelfunktionen auftretenden Orthogonalitäten orthogonal zu den Vektoren einer anderen Blockspalte $\boldsymbol{A}_{t_l p_m m_n}$, wenn $i \neq l$ oder $j \neq m$ oder $k \neq n$ ist. Da bei dem Matrizenprodukt $\boldsymbol{A}_h^T \boldsymbol{A}_h$ jede Spalte mit sich selbst und allen anderen skalar multipliziert wird und dabei jeweils ein Element der Normalgleichungsmatrix erzeugt wird, bleiben nur die Diagonalblöcke $\boldsymbol{N}_{tpm}$ in der Normalgleichungsmatrix übrig und deren Dimension entspricht der Breite der Blockspalten $\boldsymbol{A}_{tpm}$. In Abbildung 8 sind alle Blockspalten gleich breit, da es sich nur um eine schematische Darstellung handelt.
Im Gegensatz dazu sind die Spalten der Designmatrix $\boldsymbol{A}_n$ nicht orthogonal, so dass gilt $(A(:,i))^T \boldsymbol{A}(:,j) \neq 0$ auch für alle $i \neq j$, d. h. die Normalgleichungsmatrix $\boldsymbol{N}_n$ ist voll besetzt.
Da im Blockschema des Gleichungssystems $h$ die Parameter nach Ordnungen gruppiert sind, stehen alle Grade zu einer Ordnung hintereinander. Die unbekannten Parameter $\boldsymbol{x}_n$ und die Spalten der Designmatrix $\boldsymbol{A}_n$ des kleinen Systems müssen in die gleiche Reihenfolge gebracht werden, damit die Normalgleichungsmatrizen und die rechte Seite wie in 12 addiert werden können. Betrachtet man nun die Koeffzientendreiecke der Gleichungssysteme $n$ und $h$ in Abbildung 9, so sieht man, dass es bei der Auflösung 50 weniger Grade zu einer Ordnung gibt als bei der Auflösung 360.




(a) Gleichungssystem $n$ bis Grad und Ordnung 50




(b) Gleichungssystem $h$ bis Grad und Ordnung 360

Abbildung 9: Koeffizientendreiecke der Gleichungssysteme aus Beispiel 2.2

| $d = \ell_{\max_h} - \ell_{\max_n}$ | $\ell_{\max_h}$ | $d_u$ | $d_g$ |
|---|---|---|---|
| gerade | | $d/2$ | $d/2$ |
| ungerade | gerade | $(d-1)/2$ | $(d+1)/2$ |
| | ungerade | $(d+1)/2$ | $(d-1)/2$ |

Tabelle 2: Formeln zur Berechnung der Größen der Lücken $d_g$ und $d_u$ zwischen den Nebendiagonalblöcken in der Normalgleichungsmatrix bei der Kombination eines regelmäßigen mit einem nicht regelmäßigen Datensatz

Es entsteht daher bei der Zuordnung der geraden Grade $g$ und der ungeraden Grade $u$ innerhalb jedes Blocks $\boldsymbol{A}_{cgm}$, $\boldsymbol{A}_{cum}$,$\boldsymbol{A}_{sgm}$ und $\boldsymbol{A}_{sum}$ eine Lücke am Ende des Blocks. Diese wird, wie oben erwähnt, mit Nullspalten aufgefüllt, damit die richtigen Koeffizienten zugeordnet werden und die Matrizenmultiplikation definiert ist. Die Länge $d_g$ bzw. $d_u$ dieser Lücke ist jeweils für gerade Grade $g$ und ungerade Grade $u$ konstant, wie in Abbildung 9(a) abzulesen ist, wenn man entlang der $m$-Achse wandert. Der Betrag $d = d_g + d_u$ kann mit folgender Formel berechnet werden:

$$d = \ell_{\max_h} - \ell_{\max_n} = d_u + d_g \tag{13}$$

Der Wert $d_u$ bezeichnet die entstehende Lücke innerhalb eines Blocks $\boldsymbol{A}_{cum}$ oder $\boldsymbol{A}_{sum}$ und $d_g$ analog dazu die Lücke innerhalb $\boldsymbol{A}_{cgm}$ oder $\boldsymbol{A}_{sgm}$. Die Beträge dieser Lücken können mit Hilfe der Formeln in Tabelle 2 berechnet werden.
Abbildung 10 zeigt an einem kleineren Beispiel mit $\ell_{\max_n} = 10$ und $\ell_{\max_h} = 12$ das Ergebnis der Zuordnung der Spalten der Designmatrix $\boldsymbol{A}_n$ des niedrigauflösenden Systems zu den Spalten der Designmatrix $\boldsymbol{A}_h$ des hochauflösenden Systems. Herausgehoben ist der Teil der Designmatrix

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}_h \\ \bar{\boldsymbol{A}}_n \end{pmatrix}$$

der die Spalten zur Ordnung 8 und 9 enthält. Die Designmatrix $\bar{\boldsymbol{A}}_n$ ist durch Einfügen von Nullspalten aus der Designmatrix $\boldsymbol{A}_n$ entstanden.

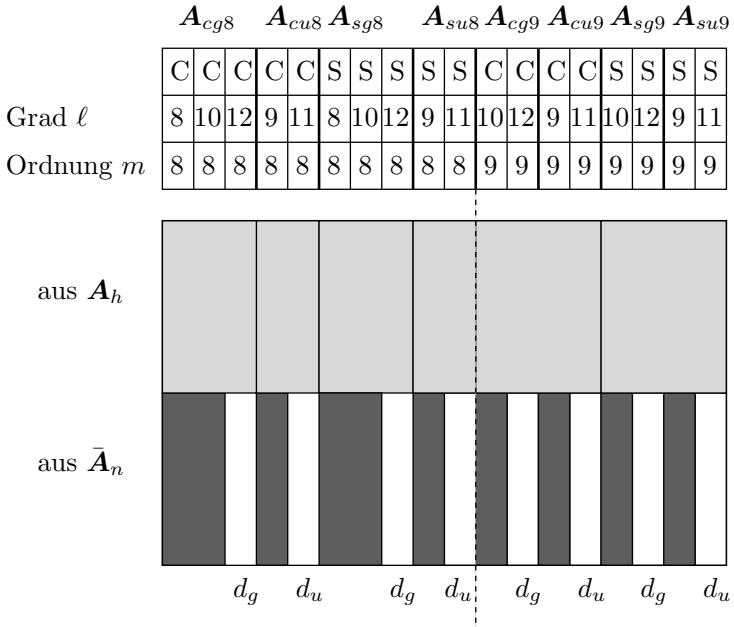



Abbildung 10: Zuordnung der Spalten der Designmatrix $\boldsymbol{A}_n$ des kleinen Systems zu den Spalten der Designmatrix $\boldsymbol{A}_h$ des großen Systems

Hellgrau dargestellt sind die Spalten eines Blocks, die zu allen übrigen Spalten ausserhalb des Blocks paarweise orthogonal sind. Dunkelgrau eingezeichnet sind Spalten, die nicht orthogonal zu den übrigen Spalten sind und daher Elemente außerhalb der Diagonalblöcke erzeugen. Durch Hinzufügen der Spalten $\bar{\boldsymbol{A}}_n$ zu $\boldsymbol{A}_h$ werden

auch dort die entsprechenden Spalten linear abhängig. Durch die Lücken in der Designmatrix $\boldsymbol{A}$ entsteht in der Normalgleichungsmatrix $\boldsymbol{N} = \boldsymbol{N}_h + \boldsymbol{N}_n$ ein Schachbrett ähnliches Muster. In Abbildung 11 ist dieses Schachbrettmuster schematisch dargestellt.

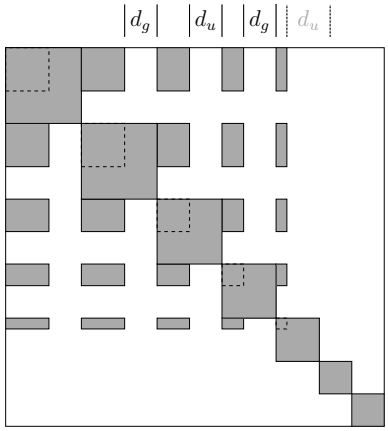



Abbildung 11: Schachbrettmuster durch Kombination einer schwach besetzten mit einer vollbesetzten Normalgleichungsmatrix

Zur Speicherung dieser Matrix müßte der obere Block, der das Schachbrettmuster enthält, als eine zusammenhängende Matrix gespeichert werden. Die restlichen Diagonalblöcke können nach wie vor als einzelne Matrizen gespeichert werden. Da der obere Block Korrelationen zwischen den Ordnungen enthält, wird der nachfolgend mit $\boldsymbol{N}_k$ bezeichnet. Er muß als zusammenhängende Matrix gespeichert werden, da bei der Choleskyzerlegung oder Inversion Füllelemente in ihm entstehen (Siehe dazu Anhang 4.2). Das Entstehen der Füllelemente kann in Abbildung 12 nachvollzogen werden. Dort ist ein Block $\boldsymbol{N}_k$ aus einer Kombination $\ell_{\max_n}$ und $\ell_{\max_h}$, seine Choleskyzerlegung und seine Inverse dargestellt. Um diese Abbildungen zu ermöglichen, sind die Gleichungssysteme aus Beispiel 2.1 verkleinert worden, so dass gilt $\ell_{\max_h} = 20$ und $\ell_{\max_n} = 10$

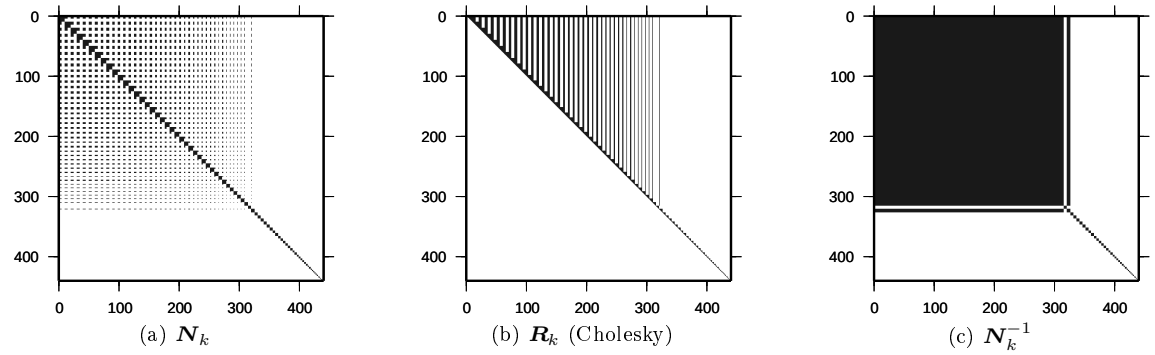



Abbildung 12: Entstehung von Füllelementen durch Operationen der Linearen Algebra

Diese Füllelemente machen die strenge Datenkombination wie in (12) aufwändig, da der Block $\boldsymbol{N}_k$ sehr groß wird. Die Seitenlänge des Blocks $\boldsymbol{N}_k$ setzt sich zusammen aus der Seitenlänge der Normalgleichungsmatrix $\boldsymbol{N}_n$ und zwei Lücken $d$ pro Ordnung, jeweils $d = d_u + d_g$ für Sinus und $d = d_u + d_g$ für Kosinus. Anhand des Koeffizientendreiecks aus Abbildung 9 sieht man, dass $d$ schneller größer wird als die Seitenlänge der Diagonalblöcke des kleinen Systems $n$. Tabelle 3 zeigt die Größe des Blocks $\boldsymbol{N}_k$ bei verschiedenen Kombinationen. Die Größe $r$ des Blocks $\boldsymbol{N}_k \in \mathbb{R}^{r \times r}$ wurde mit der Formel

$$r = 2\,\ell_{\max_n}\,d + d \qquad \text{mit} \qquad d = \ell_{\max_h} - \ell_{\max_n} \tag{14}$$

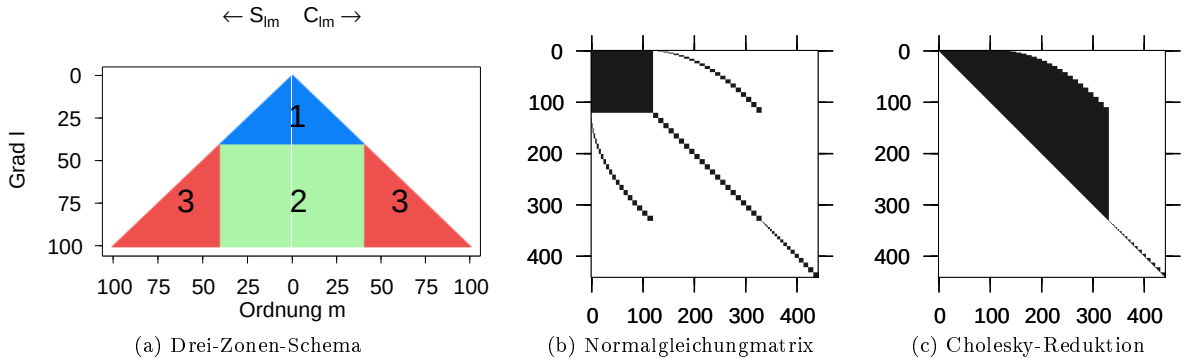
berechnet.
Auffällig ist, dass nicht nur der maximale Entwicklungsgrad des kleinen Systems die Speicheranforderung erhöht, sondern auch der des hochauflösenden, blockdiagonalen Systems. So reichen z. B. drei Gigabyte Speicher aus, um die strenge Kombination $\ell_{\max_h} = 200$, $\ell_{\max_n} = 100$ zu rechnen, wohingegen die Kombination $\ell_{\max_h} = 360$, $\ell_{\max_n} = 100$ schon über 20 Gigabyte Speicher benötigt die einem handelsüblichen PC nicht zur Verfügung stehen.

| Regelmäßige Daten | | Nicht regelmäßige Daten | | Kombination | |
|---|---|---|---|---|---|
| $\ell_{\max_h}$ | Dimension $\boldsymbol{N}_h$ | $\ell_{\max_n}$ | Dimension $\boldsymbol{N}_n$ | Dimension $\boldsymbol{N}_k$ | Speicherplatz $\boldsymbol{N}_k$ |
| 200 | 40401 | 25 | 676 | 8925 | 608 MB |
| | | 50 | 2601 | 15150 | 1.7 GB |
| | | 75 | 5776 | 18875 | 2.7 GB |
| | | 100 | 10201 | 20100 | 3.0 GB |
| 360 | 130321 | 25 | 676 | 17085 | 2.2 GB |
| | | 50 | 2601 | 31310 | 7.3 GB |
| | | 75 | 5776 | 43035 | 13.8 GB |
| | | 100 | 10201 | 52260 | 20.3 GB |

Tabelle 3: Explosion der Speicheranforderung bei der Datenkombination im Blockschema

### 2.3.1 Nummerierungsschema von Bosch

Um die Struktur der Normalgleichungen zu vereinfachen, schlug daher Bosch (1993) eine Nummerierung vor, bei der die Koeffizienten in drei Zonen eingeteilt werden. Zur eindeutigen Identifizierung bezeichnen wir dieses Schema daher als *Drei-Zonen-Schema* (Abbildung 13(a)).




(a) Drei-Zonen-Schema (b) Normalgleichungmatrix (c) Cholesky-Reduktion

Abbildung 13: Sortierung der Parameter nach Bosch

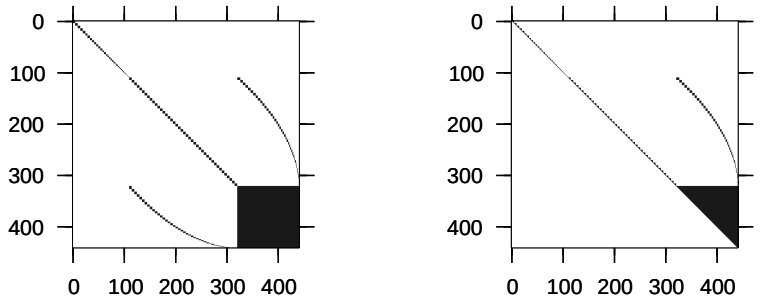
Zone Eins bilden die voll korrelierten Parameter der niedrigeren Grade bis $\ell_{\max_n}$, die zweite Zone bilden die Koeffizienten höherer Grade ($> \ell_{\max_h}$), aber kleinerer Ordnung als $\ell_{\max_h}$, und den letzten Abschnitt bilden die Koeffizienten hoher Grade und hoher Ordnungen. Innerhalb der einzelnen Zonen werden Parameter gleicher Ordnungen mit (15) gruppiert.

$$C_{\ell m}, S_{\ell m} \in \begin{cases} \text{Zone 1} & \text{falls} \quad 0 \le m \le \ell_{\max_n} & \text{und} \quad 0 \le \ell \le \ell_{\max_n} \\ \text{Zone 2} & \text{falls} \quad 0 \le m \le \ell_{\max_n} & \text{und} \quad \ell_{\max_n} < \ell \le \ell_{\max_h} \\ \text{Zone 3} & \text{falls} \quad \ell_{\max_n} < m \le \ell_{\max_h} & \text{und} \quad \ell_{\max_n} < \ell \le \ell_{\max_h} \end{cases} \tag{15}$$

Damit tritt zwar eine kompaktere Struktur in den Normalgleichungen auf (Abbildung 13(b)), trotzdem treten bei der Reduktion wieder Füllelemente (Abbildung 13(c)), mit der gleichen Anzahl auf, wie bei der Verwendung des Blockschemas.

### 2.3.2 Kiteschema

Im Zuge der *GOCE* Studien wurde von Schuh (1996) ein Nummerierungsschema vorgeschlagen, das diesen Mangel behebt. Durch die Umkehr der Reihenfolge der Parameter bei der Nummerierung wurde ein Schema entwickelt, bei welchem keinerlei Füllelemente während der Reduktion entstehen. Die drachenartige Stuktur der Normalgleichungen führte auf den Namen *Kiteschema*. Somit wird eine strenge Datenkombination von hochauflösenden, rotationssymmetrischen Datenquellen, die nur ordnungsweise korreliert sind, mit beliebig verteilten, voll korrelierten Daten für die Bestimmung der niedrigeren Grade ohne Füllelemente möglich. Diese Datenkombination kann somit auch für sehr hochauflösende Systeme (z. B. Grad/Ordnung 720) ohne Probleme auf einem

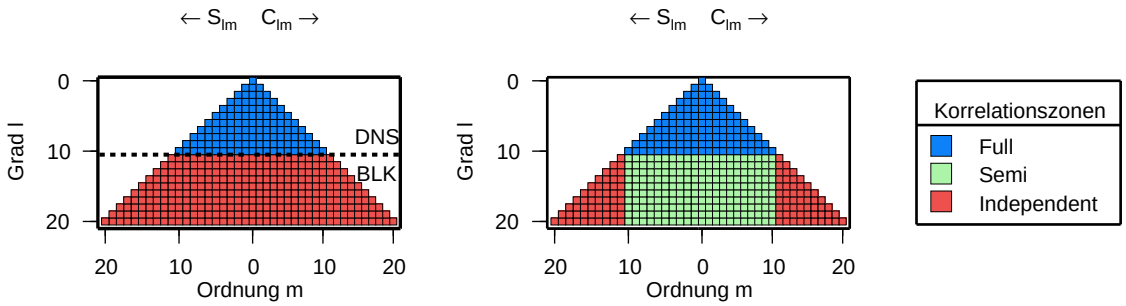



(a) Normalgleichungsmatrix (b) Cholesky Reduktion

Abbildung 14: Sortierung der Parameter im Kiteschema

Standardrechner berechnet werden. Neben der Lösung des Systems kann auch sehr effizient die partielle Inverse, d. h. die strenge Inverse für ausgewählte Elemente, ermittelt werden (Auzinger und Schuh 1998).

**Aufteilung der Koeffizienten in drei Zonen:** Im Folgenden soll die Entstehung der drei Zonen und die damit verbundene Struktur der Kitematrix erläutert werden. Hierzu werden zunächst nur die Koeffizienten und deren Korrelationen aufgrund des regelmäßigen Datensatzes berücksichtigt. Der erste Schritt ist die Aufteilung der Koeffizienten in die Menge *DNS (dense)* und *BLK (block)*, durch Einführung eines maximalen Entwicklungsgrades $\ell_{\max_{\mathrm{DNS}}}$ (Abbildung 15), welcher dem maximalen Entwicklungsgrad des unregelmäßigen Datensatzes entspricht, dessen vollbesetzte *(dense)* Normalgleichungsmatrix später addiert werden soll. Abbildung 15(a) zeigt die neu entstandenen Parameterzonen. Da die Koeffizienten bei Vorliegen eines regelmäßigen Datensatzes nur ordnungsweise bei gleicher trigonometrischer Funktion und gleicher Parität des Grades korreliert sind, sind in der Ansicht des Koeffizientendreiecks nur Parameter korreliert, die übereinander stehen. Aufgrund dieser Korrelationen in vertikaler Richtung entstehen innerhalb des Koeffizientendreiecks die drei Zonen des Schemas von Bosch. Diese Zonen sollen nun mit *Full*, *Semi* und *Independent* bezeichnet werden (Abbildung 15(b)):




(a) Parametermengen (b) Korrelationszonen

Abbildung 15: Parametermengen und Korrelationszonen

- *Full*
  Die Parameter dieser Zone entsprechen den Parametern des voll besetzten Normalgleichungssystems des unregelmäßigen Datensatzes. Sie werden am Ende der Parameterreihenfolge angeordnet.

- *Semi*
  Die Parameter der Zone *Semi* besitzen die gleiche Ordnung wie die des unregelmäßigen Datensatzes, haben demgegenüber höhere Grade. Aufgrund der Korrelationen innerhalb der Ordnung existieren Korrelationen zu der Zone *Full*. Diese Zone wird in der Mitte der Parameterreihenfolge angeordnet.

- *Independent*
  Die Parameter der Zone *Independent* sind völlig unabhängig von den Parametern der Zone *Full* und damit von den Parametern des unregelmäßigen Datensatzes, da ihre Ordnungen höher sind, als die maximale Ordnung $\ell_{\max_{\mathrm{DNS}}}$. Die Parameter der Zone *Independent* stehen stets am Anfang der Parameterreihenfolge.

**Korrelationen zwischen den Zonen in der Normalgleichungsmatrix:** Durch das Kiteschema werden die drei Zonen in der Reihenfolge *Independent,Semi,Full* angeordnet. Aufgrund der Korrelationen zwischen den Zonen kann die Kitematrix, wie in Abbildung 16 dargestellt, in verschiedene Sektoren aufgeteilt werden.

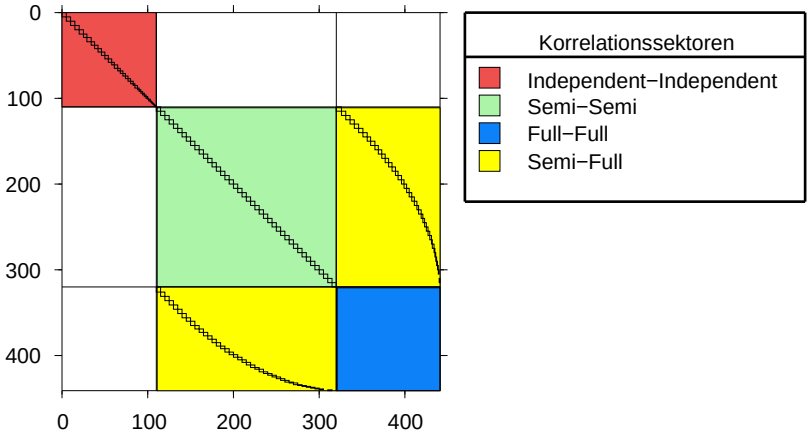
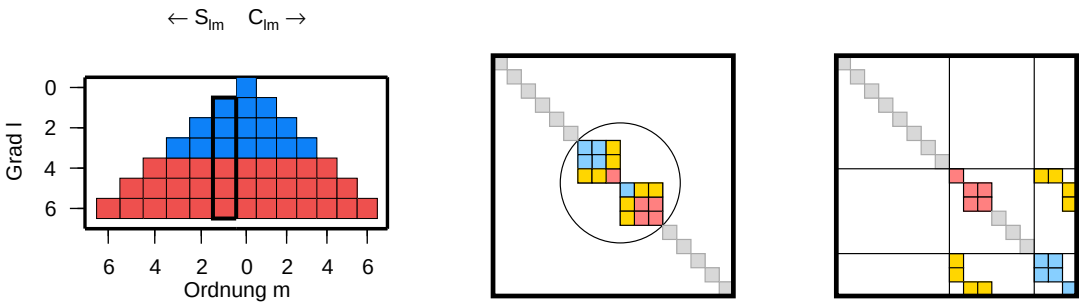



Abbildung 16: Aufteilung der Kitematrix in verschiedene Sektoren

Da die Parameter der Zone *Independent* unabhängig von allen anderen sind, gibt es in der ersten Blockzeile und der ersten Blockspalte in Abbildung 16 nur den Sektor *Independent-Independent*, der die Korrelationen zwischen den Parametern aus der Zone *Independent* enthält. Die Zonen *Semi* und *Full* sind teilweise miteinander korreliert, da die Parameter der Ordnungen $m$ mit $m \leq \ell_{\max_{\mathrm{DNS}}}$ auf beide Zonen verteilt sind. Somit sind zusätzlich zu den Sektoren *Semi-Semi* und *Full-Full* auch die Sektoren *Semi-Full* mit Nichtnullelementen belegt. In den Sektoren *Independent-Independent* enstehen während der Reduktion keine Füllelemente. Analoges gilt für den Sektor *Semi-Semi* im mittleren Bereich. Für den Sektor *Semi-Full* kann zunächst angenommen werden, dass unter den Kite–Flügeln Füllelemente enstehen. Betrachtet man jedoch die Struktur genauer, so erkennt man, dass zu jedem Diagonalblock im mittleren Abschnitt *Semi-Semi* exakt nur ein Block in *Semi-Full* korrespondiert. Da pro (Block-) Zeile somit nur jeweils ein Nebendiagonalblock existiert, verschwinden alle Skalarprodukte zwischen unterschiedlichen (Block-)Spalten. Damit entstehen beim Reduktionschritt (z.B. Cholesky) keine zuätzlichen Füllelemente.

**Zur Entstehung der Nebendiagonalblöcke:** Am Beispiel einer einzelnen Ordnung soll nun die Entstehung der Kite–Flügel dargestellt werden. Ausgehend von der Nummerierung im Blockschema und der daraus resultierenden Blockdiagonalmatrix werden die Veränderungen diskutiert, die sich durch die Umsortierung in das Kiteschema ergeben. Dazu wird eine Ordnung $m_i$ mit $m_i \leq \ell_{\max_{\mathrm{DNS}}}$ herausgegriffen, deren Parameter teilweise in der Parametermenge *DNS* und teilweise in der Parametermenge *BLK* liegt. Zur Vereinfachung der Darstellung sollen nur die Sinuskoeffizienten $S_{\ell m_i}$ betrachtet werden. Da innerhalb der Ordnung $m_i$ nur die Parameter miteinander korreliert sind, deren Grad die gleiche Parität hat, entstehen beim Blockschema in der Normalgleichungsmatrix zwei Diagonalblöcke. Durch Einführung der beiden Parametermengen *DNS* und *BLK* werden diese beiden Diagonalblöcke, die in Abbildung 17(b) hervorgehoben sind, weiter unterteilt.
Blau dargestellt sind die Korrelationen zwischen Parametern aus dem Bereich *Full*. Rot kennzeichnet die Korrelationen zwischen Parametern aus dem Bereich *Semi*. Die Korrelationen zwischen den Parametern dieser beiden Zonen stellen die gelb eingezeichneten Sektoren dar. Beim Übergang auf die Kite–Nummerierung werden die Diagonalblöcke des Blockschemas auseinandergezogen und in die Sektoren *Semi-Semi* und *Full-Full* einsortiert. Die Korrelationen zwischen *Semi* und *Full* kommen dadurch in den Sektor *Semi-Full* und bilden die Kite–Flügel. Diese stellen nicht die Korrelationen zwischen den regelmäßigen und den nicht regelmäßigen Daten dar, sondern nur die Korrelationen zwischen den niedrigen und den hohen Graden innerhalb der Ordnung beim regelmäßigen Datensatz.
Zur Kombination von regelmäßigen, hochauflösenden Daten mit dem Kiteschema werden zunächst die Elemente der Kitematrix berechnet, die aufgrund des regelmäßigen Datensatzes entstehen. Der letzte Block der Kitematrix, der dem Korrelationssektor *Full-Full* entspricht, ist in dieser Matrix noch blockdiagonal (Abbildung 18(a)). Seine Koeffizienten haben die gleiche Reihenfolge wie die der vollbesetzten Normalgleichungsmatrix (Abbildung 18(b)) des unregelmäßigen Datensatzes. Anschließend wird die Kitematrix vervollständigt, indem die vollbesetzte Normalgleichungsmatrix zu dem Korrelationssektor *Full-Full* addiert wird (Abbildung 18(b)).

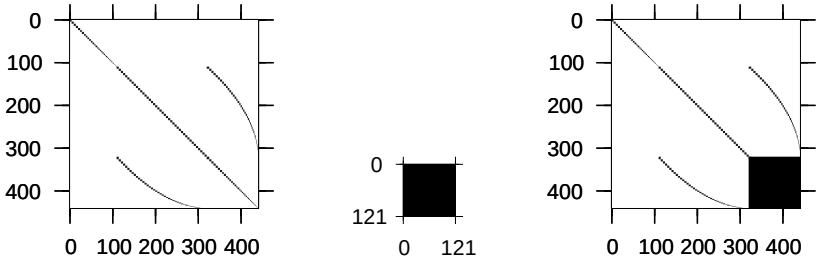



(a) Aufteilung der Ordnung $m_i$ in *BLK* und *DNS*

(b) Ausgesuchte Korrelationen $S_{\ell m_i}$ im Blockschema

(c) Dieselben Korrelationen im Kiteschema

Abbildung 17: Entstehung der Kite–Flügel beim Übergang vom Blockschema auf das Kiteschema




(a) Beitrag des regelmäßigen Datensatzes

(b) Beitrag des unregelmäßigen Datensatzes

(c) Kombinierte Kitematrix

Abbildung 18: Vorgehen beim Aufstellen der Kitematrix

Das Kiteschema kann sehr effizient für die strenge Lösung von kombinierten Modellen mit gegitterten, hochauflösenden Daten eingesetzt werden, dient aber auch als ausgezeichnete Näherungslösung, wenn die hochauflösenden Daten zwar nicht in rotationssymmetrischer, aber nahezu regelmäßiger Form vorliegen. Hier kann die Kitematrix zur Vorkonditioniererung verwendet und eine strenge Lösung sehr effizient über iterative Techniken berechnet werden. Das speziell für die GOCE Auswertung entwickelte PCGMA–Verfahren(Preconditioned Conjugate Gradient Multiple Adjustment (Schuh 1996)) verwendet das Kiteschema zur Datenkombination der SST (satellite-to-satellite tracking) und SGG (satellite gravity gradiometer) Daten.

# 3 Freies Kite–Nummerierungsschema *FKN*

Wie sich bei den neueren GOCE Simulationen zeigte, weist das Kiteschema einige Unzulänglichkeiten auf. So wurden die Parameter für jede Ordnung $m$ immer mit Grad $\ell_{\min} = \max(2, m)$ begonnen und durch einen konstanten Grad $\ell_{\max_{\mathrm{DNS}}}$ für die voll korrelierten Parameter bzw. $\ell_{\max_{\mathrm{BLK}}}$ für das hochauflösende Modell begrenzt. Bei vielen Erdschwerefeldmodellierungen (z. B. EIGEN-1S (Reigber et al. 2002), EIGEN-2 (Reigber et al. 2003)) werden inzwischen variable Schranken eingeführt, um bestimmte Resonanzfrequenzen besser berücksichtigen zu können.
Um diesen Anforderungen nachkommen zu können, wird hier eine Methodik vorgestellt, die eine variable Abgrenzung im Kite-Schema erlaubt. Dieses im folgenden Kapitel beschriebene Verfahren wird als *Freies Kite–Nummerierungsschema (FKN)* bezeichnet und setzt sich im Wesentlichen aus drei Kernkomponenten zusammen

1. Algorithmus für die Aufstellung der Reihenfolge der Parameter
2. Speicherung und Addressierung der Nichtnullelemente der Normalgleichungsmatrix
3. Choleskyreduktion, Vorwärts- und Rückwärtseinsetzen und unvollständige Inversion mit dem gewählten Speicherschema

## 3.1 Aufstellung der Reihenfolge der Parameter

Im Kiteschema wird die Reihenfolge der Parameter zunächst symbolisch aufgestellt und gespeichert. Dadurch werden die Aufstellung und die Anwendung der Nummerierung voneinander getrennt. Die symbolische Speicherung der Parameter erfolgt in einer ASCII Datei mit folgendem Aussehen:

```
1  # kite numbering scheme
2  maxgrad 360
3  maxordnung 360
4  nrparam 130321
5  C 0 0
6  C 2 0
7  C 4 0
8  C 6 0
9  C 8 0
10 C 10 0
11 usw.
```

In den Algorithmen ist es an verschiedenen Stellen notwendig, die trigonometrische Funktion, den Grad oder die Ordnung zu vergleichen. Innerhalb der Algorithmen wird daher ein Parameter in einem Objekt gespeichert, das die Elemente $\mathsf{CS}$, $\ell$, und $m$ enthält, die einen Parameter eindeutig identifizieren.

### 3.1.1 Freies Blockschema

Um eine größere Flexibilität bei der Erdschwerefeldmodellierung zu ermöglichen, werden Nummerierungsschemata notwendig, die es erlauben, bestimmte Koeffizienten gezielt aus- oder abzuwählen. Die starre Begrenzung auf einen maximalen Grad für alle Ordnungen erweist sich in der Praxis als zu unflexibel, um auf Stärken wie Resonanzfrequenzen und Schwächen, wie schlecht bestimmte, zonale Koeffizienten aufgrund des polaren Lochs, eines Modells eingehen zu können.
Gesucht ist daher die Möglichkeit, gezielt Parameter hinzuzunehmen oder wegzulassen. Um dies zu ermöglichen, wird nun das Blockschema (Algorithmus 2.2) durch die Einführung eines minimalen Grades $\ell_{\min}$ und eines maximalen Grades $\ell_{\max}$ für jede Ordnung $m$ erweitert. Diese Grenzen können in drei Vektoren $\boldsymbol{m}$, $\boldsymbol{\ell_{\min}}$ und $\boldsymbol{\ell_{\max}}$ abgebildet werden, die die Menge der angesetzten Parameter eindeutig beschreiben. Diese Vektoren sind gleichlang und haben so viele Elemente wie Ordnungen angesetzt werden. Die Anzahl der Ordnungen soll mit $o$ bezeichnet werden. Das so erweiterte Blockschema wird *freies Blockschema* genannt.

$$
\begin{aligned}
\boldsymbol{m} &= \left(m_1, m_2, \ldots, m_{\mathrm{o}}\right) && \in \mathbb{R}^{o\times 1} \\
\boldsymbol{\ell}_{\min} &= \left(\ell_{\min_1}, \ell_{\min_2}, \ldots, \ell_{\min_o}\right) && \in \mathbb{R}^{o\times 1} \\
\boldsymbol{\ell}_{\max} &= \left(\ell_{\max_1}, \ell_{\max_2}, \ldots, \ell_{\max_o}\right) && \in \mathbb{R}^{o\times 1}
\end{aligned}
$$

Das Prinzip soll nun an einem einfachen Beispiel gezeigt werden.

**Beispiel 3.1 (Freies Blockschema: )** *Gegeben seien folgende Vektoren der minmalen und maximalen Grade für alle Ordnungen m zwischen 0 und 7:*

$$\begin{aligned} \boldsymbol{m} &= (0,1,2,3,4,5,6,7) \\ \boldsymbol{\ell}_{min} &= (2,2,2,3,5,5,6,7) \\ \boldsymbol{\ell}_{max} &= (6,7,6,7,6,7,6,7) \end{aligned}$$

*Gesucht ist die Reihenfolge und die Anzahl der ausgewählten Parameter im freien Blockschema.* ■

Das Koeffizientendreieck zu diesem Beispiel (Abbildung 19) besitzt keine echte Dreiecksform mehr, sondern an allen drei Seiten zeigen sich unregelmäßige Einbuchtungen, die von den ausgelassenen Parametern herrühren.

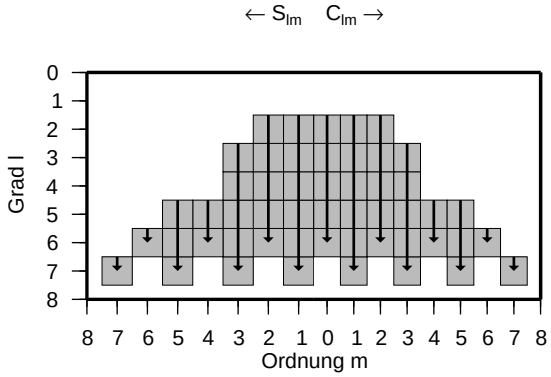



Abbildung 19: Freies Blockschema mit individuellem minimalem und maximalem Grad pro Ordnung

Der Algorithmus 3.1 für die Bestimmung der Reihenfolge der Parameter im freien Blockschema funktioniert analog zu dem Algorithmus 2.2 des klassischen Blockschemas, mit dem Unterschied, dass statt der Grenzen 0 und $\ell_{\max}$ jetzt die Vektoren $\boldsymbol{\ell}_{\min}$ und $\boldsymbol{\ell}_{\max}$ für die Grenzen von $\ell$ eingesetzt werden. Da die Anzahl der unbekannten Parameter durch die freien Grenzen nicht mehr mit (9) berechnenbar ist, werden die auftretenden Parameter in einer Datenstruktur abgelegt, welche das Anhängen von neuen Elementen am Ende erlaubt. Diese Datenstruktur wird mit *Stack* bezeichnet. Die Anzahl der unbekannten Parameter entspricht der Länge des Stacks, den der Algorithmus erzeugt.

**Algorithmus 3.1 (Freies Blockschema)**

```
1  #include "parameter.h"
2
3  void freies_blockschema_reihenfolge( vector<Parameter> &stack, Vector l_min, Vector l_max )
4  {
5     // C_lm anordnen
6     for( int m = 0; m < l_max.size(); m++ )
7     {
8        for( int l = l_min[m]; l <= l_max[m]; l += 2 )
9           stack.push_back( Parameter( 'C', l, m ) );  // gerade/ungerade Koeffizienten
10
11       for( int l = l_min[m]+1; l <= l_max[m]; l += 2 )
12          stack.push_back( Parameter( 'C', l, m ) );  // ungerade/gerade Koeffizienten
13
14       // S_lm anordnen
15       if( m != 0 )
16       {
17          for( int l = l_min[m]; l <= l_max[m]; l += 2 )
18             stack.push_back( Parameter( 'S', l, m ) );  // gerade/ungerade Koeffizienten
19
20          for( int l = l_min[m]+1; l <= l_max[m]; l += 2 )
21             stack.push_back( Parameter( 'S', l , m ) ); // ungerade/gerade Koeffizienten
22       }
23    }
24 }
```

Der Algorithmus *Freies Blockschema* (3.1) ermittelt für das Beispiel 3.1 51 Parameter in der folgenden Reihenfolge:

$$
\begin{array}{l}
C_{20}\ C_{40}\ C_{60}\ C_{30}\ C_{50}\ C_{21}\ C_{41}\ C_{61}\ C_{31}\ C_{51}\ C_{71}\ S_{21}\ S_{41}\ S_{61}\ S_{31}\ S_{51}\ S_{71} \\
C_{22}\ C_{42}\ C_{62}\ C_{32}\ C_{52}\ S_{22}\ S_{42}\ S_{62}\ S_{32}\ S_{52}\ C_{33}\ C_{53}\ C_{73}\ C_{43}\ C_{63}\ S_{33}\ S_{53} \\
S_{73}\ S_{43}\ S_{63}\ C_{54}\ C_{64}\ S_{54}\ S_{64}\ C_{55}\ C_{75}\ C_{65}\ S_{55}\ S_{75}\ S_{65}\ C_{66}\ S_{66}\ C_{77}\ S_{77}
\end{array}
$$

Diese Parameter werden in der oben beschriebenen Form in einer ASCII Datei gespeichert, damit andere Programme diese Reihenfolge in ihrer Berechnung verwenden können.

### 3.1.2 Freies Kiteschema

Mit dem neu entwickelten *Freien Kite–Nummerierungsschema (FKN)* können nun Daten kombiniert werden, wobei beide Modelle jeweils freie Grenzen besitzen, wie es im *Freien Blocknummerierungschema* eingeführt wurde. Das Freie Kite–Nummerierungschema garantiert, dass in diesem allgemeinen Fall die resultierende Normalgleichungsmatrix eine Kitematrix bleibt, um eine Datenkombination ohne Füllellemente zu ermöglichen. So kann beispielsweise die in den Abbildungen 15 und 16 dargestellte Konfiguration mit den festen Grenzen 2, $\ell_{\mathrm{DNS}}$ und $\ell_{\mathrm{BLK}}$ durch folgende Vektoren im FKN beschrieben werden:

$$
\begin{aligned}
\boldsymbol{m} &= \big(0,1,2,3,4,\ldots,19,20\big) \\
\boldsymbol{\ell}_{\mathrm{min_{DNS}}} &= \big(2,2,2,3,4,\ldots,9,10,-1,-1,\ldots,-1\big) \qquad & \boldsymbol{\ell}_{\mathrm{min_{BLK}}} &= \big(2,2,2,3,4,\ldots,19,20\big) \\
\boldsymbol{\ell}_{\mathrm{max_{DNS}}} &= \Big(\underbrace{10,10,\ldots,10}_{\text{Ordnung 0 bis 10}},\ \underbrace{-1,\ldots,-1}_{\text{Ordnung 11 bis 20}}\Big) & \boldsymbol{\ell}_{\mathrm{max_{BLK}}} &= \big(20,20,\ldots,20\big)\,.
\end{aligned}
$$

Aus diesen Vektoren kann FKN nun die symbolische Reihenfolge der Parameter und die Positionen der Nichtnullelemente in der Normalgleichungsmatrix berechnen. Während das *Kiteschema* auf komplexen Programmen mit starren Rechenvorschriften aufgebaut ist, die keine weitere Flexibilisierung zulassen, wurde bei der Weiterentwicklung des FKN auf eine klar strukturierte, regelbasierte Logik geachtet.

**Regelbasierte Verarbeitung:** Der Algorithmus zur Aufstellung der Parameterreihenfolge mit FKN besteht aus zwei Schleifen und einem regelbasierten Entscheidungsbaum innerhalb des inneren Schleifenrumpfes. Die äussere Schleife iteriert über alle Ordnungen und die innere Schleife iteriert vom minimalen Grad bis zum maximalen Grad der jeweiligen Ordnung entsprechend dem Algorithmus 3.1. Im Unterschied zu Algorithmus 3.1

liegen hier sowohl für Parameter aus der Korrelationszone DNS als auch für Parameter aus der Korrelationszone BLK die Vektoren der minimalen und maximalen Grade vor. Ist zu einer Ordnung $i$ kein minmaler und maximaler DNS Grad vorhanden, so werden die Vektoren $\boldsymbol{\ell}_{\min_{\mathrm{DNS}}}$ und $\boldsymbol{\ell}_{\max_{\mathrm{DNS}}}$ an der Stelle $i$ mit dem Wert -1 ausgefüllt. Der FKN Algorithmus benutzt für seine innere Schleife das Minimum der beiden minimalen Grade und das Maximum der beiden maximalen Grade für die betreffende Ordnung, wobei Vektorelemente mit dem Wert -1 ignoriert werden. Innerhalb der Schleifen werden drei Stacks aufgebaut. Jeder Parameter durchläuft nun den Entscheidungsbaum und wird entweder auf einen der drei Stacks gelegt oder verworfen. Die drei Stacks enthalten nach Durchlaufen der Schleife die Parameter der Zonen *Full*, *Semi* und *Independent*. Der Entscheidungsbaum arbeitet nach folgenden Regeln:

**Algorithmus 3.2 (Zuordnung zu einer Korrelationszone)**
Untersucht wird der Parameter $\{CS\}_{lm}$:

a) Gibt es innerhalb der Ordnung $m$ keine Parameter der Korrelationszone DNS, d. h. $\boldsymbol{\ell}_{\min_{\mathrm{DNS}}}(m) = -1$ und $\boldsymbol{\ell}_{\max_{\mathrm{DNS}}}(m) = -1$ so liegt der Parameter in der Zone *Independent*, sonst in *Semi* oder *Full*

b) Liegt der Grad $\ell$ für diesen Parameter innerhalb des minimalen und des maximalen Grades der Korrelationszone DNS für diese Ordnung $m$, d. h. $\boldsymbol{\ell}_{\min_{\mathrm{DNS}}}(m) \leq \ell \leq \boldsymbol{\ell}_{\max_{\mathrm{DNS}}}(m)$, so liegt der Parameter in Zone *Full*, sonst in Zone *Semi*.

Der Rahmen zur Aufstellung der Parameterreihenfolge mit diesen Regeln funktioniert nach folgendem Prinzip:

**Algorithmus 3.3 (Freies Kite–Nummerierungsschema)**

```
1  Schleife über alle Ordnungen m
2  {
3     Schleife über alle Grade l dieser Ordnung
4     {
5
6        Gerade Kosinuskoeffizienten
7           in Zone Independent, Semi oder Full einsortieren
8
9        Ungerade Kosinuskoeffizienten
10          in Zone Independent, Semi oder Full einsortieren
11
12       Gerade Sinuskoeffizienten
13          in Zone Independent, Semi oder Full einsortieren
14
15       Ungerade Sinuskoeffizienten
16          in Zone Independent, Semi oder Full einsortieren
17    }
18 }
19
20 Zonen aneinanderhängen
```

Algorithmus 3.3 stellt den Algorithmus zur Aufstellung der Reihenfolge der unbekannten Parameter zum leichteren Verständnis in Textform dar, wohingegen Algorithmus 3.4 eine in C++ programmierte Version ist.

**Algorithmus 3.4 (FKN)**

```
1  #include "parameter.h"
2
3  struct Zones { vector<Parameter> Independent, Semi, Full; };
4
5  void appendToZone( Parameter p, Zones &zones, Vector &dns_min, Vector &dns_max )
6  {
7     // Kein minmaler und maximaler Grad für DNS, daher ist p in Zone Independent
8     if( dns_min[p.m()] == -1 or dns_max[p.m()] == -1 )
9        zones.Independent.push_back( p );
10
11    // Parameter ist in Zone Full
12    else if( dns_min[p.m()] <= p.l()  and p.l() <= dns_max[p.m()] )
13       zones.Full.push_back( p );
14
15    // Parameter ist in Zone Semi
16    else
17       zones.Semi.push_back( p );
18 }
19
20 Zones fkn_reihenfolge( Vector orders, Vector dns_min, Vector dns_max, Vector blk_min, Vector blk_max )
21 {
22    Zones zones;                          // Die drei Zonen
23
24    for( int m = 0; m < orders.size(); m++ )      // C_lm anordnen
25    {
26       int lmin, lmax;                    // kleinster und größter Grad dieser Ordnung
27       if( dns_min[m] != -1 and dns_max[m] != -1 )
28       {
29          lmin = min(dns_min[m],blk_min[m]);
30          lmax = max(dns_max[m],blk_max[m]);
31       }
32       else
33          lmin = blk_min[m], lmax = blk_max[m];
34
35       for( int l = lmin; l <= lmax; l += 2 )
36          appendToZone( Parameter('C', l, m ), zones, dns_min, dns_max );       // gerade/ungerade Koeffizienten
37
38       for( int l = lmin+1; l <= lmax; l += 2 )
39          appendToZone( Parameter( 'C', l, m ), zones, dns_min, dns_max );      // ungerade/gerade Koeffizienten
40
41       if( m != 0 ) {        // S_lm nur für Ordnung größer null
42          for( int l = lmin; l <= lmax; l += 2 )
43             appendToZone( Parameter( 'S', l, m ), zones, dns_min, dns_max );  // gerade/ungerade Koeffizienten
44
45          for( int l = lmin+1; l <= lmax; l += 2 )
46             appendToZone( Parameter( 'S', l , m ), zones, dns_min, dns_max ); // ungerade/gerade Koeffizienten
47       }
48    }
49    return zones;
50 }
```

Das Beispiel 3.1 soll nun erweitert werden, um die Funktionsweise des FKN zu demonstrieren.

**Beispiel 3.2 (Anwendung FKN)** *Angenommen, die gegebenen Vektoren für die Ordnungen und die minimalen und maximalen Grade des Beispiels 3.1 beschreiben die Parametermenge für einen regelmäßigen Datensatz BLK mit einer blockdiagonalen Normalgleichungsmatrix. Diese sollen nun kombiniert werden mit Parametern eines unregelmäßigen Datensatzes DNS, deren Normalgleichungsmatrix vollbesetzt ist. Die Eingabevektoren für den Algorithmus 3.3 bzw. 3.4 sind im folgenden dargestellt. Gibt es zu einer Ordnung keinen Parameter in der Korrelationszone, so ist, wie oben erläutert, sowohl der maximale, als auch der minimale Grad zu dieser Korrelationszone gleich -1.*

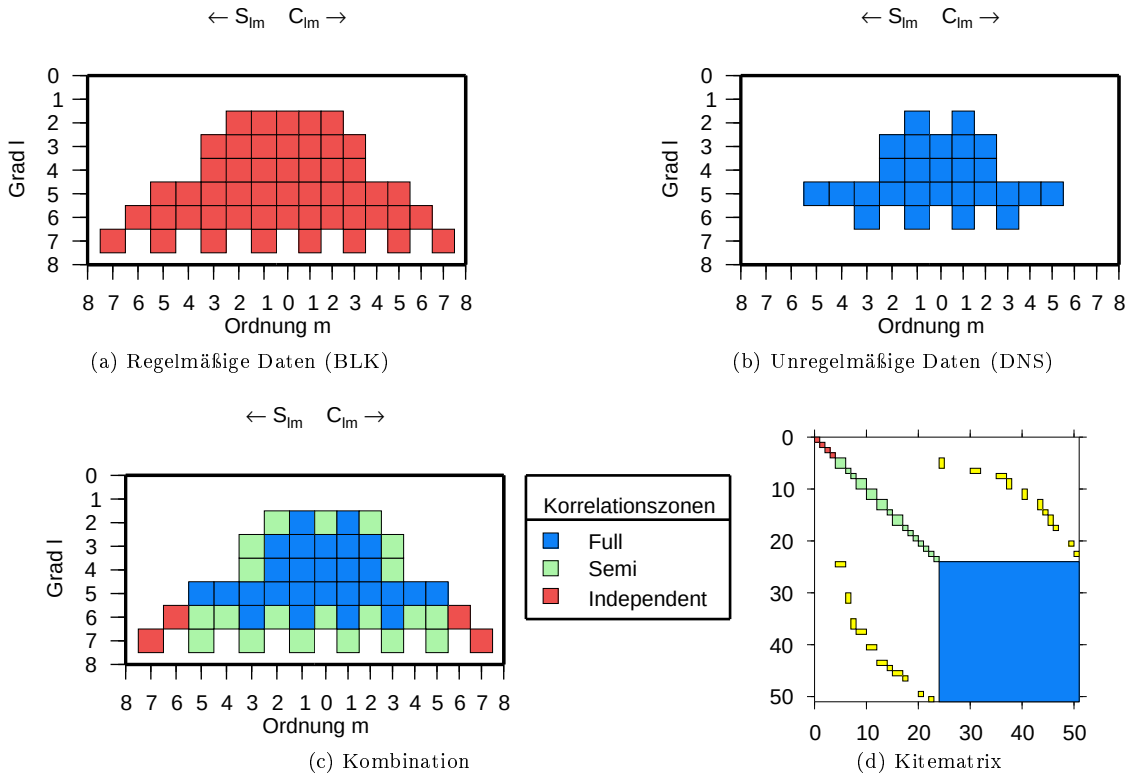
$$\begin{aligned} \boldsymbol{m} &= (0,1,2,3,4,5,6,7) \\ \boldsymbol{\ell}_{\min_{DNS}} &= (3,2,3,5,5,5,-1,-1) \qquad \boldsymbol{\ell}_{\min_{BLK}} = (2,2,2,3,5,5,6,7) \\ \boldsymbol{\ell}_{\max_{DNS}} &= (5,6,5,6,5,5,-1,-1) \qquad \boldsymbol{\ell}_{\max_{BLK}} = (6,7,6,7,6,7,6,7)\,. \end{aligned}$$

*Gesucht ist die Reihenfolge der Parameter im Freien Kiteschema FKN, so dass die Normalgleichungsmatrix des kombinierten Systems eine Kitematrix ist.* ■

Der FKN Algorithmus (Algorithmus 3.4) berechnet mit den Vektoren dieses Beispiels eine Gesamtanzahl von 51 Parametern in den drei Korrelationszonen *Independent*, *Semi* und *Full*. Die Reihenfolge der Parameter ergibt sich durch Aneinanderhängen der drei Zonen in der angegebenen Reihenfolge.

| | | |
|---|---|---|
| *Independent* | (4) | $C_{66}\ S_{66}\ C_{77}\ S_{77}$ |
| *Semi* | (20) | $C_{20}\ C_{60}\ C_{71}\ S_{71}\ C_{22}\ C_{62}\ S_{22}\ S_{62}\ C_{33}\ C_{73}$ |
| | | $C_{43}\ S_{33}\ S_{73}\ S_{43}\ C_{64}\ S_{64}\ C_{75}\ C_{65}\ S_{75}\ S_{65}$ |
| *Full* | (27) | $C_{40}\ C_{30}\ C_{50}\ C_{21}\ C_{41}\ C_{61}\ C_{31}\ C_{51}\ S_{21}\ S_{41}\ S_{61}\ S_{31}\ S_{51}\ C_{42}$ |
| | | $C_{32}\ C_{52}\ S_{42}\ S_{32}\ S_{52}\ C_{53}\ C_{63}\ S_{53}\ S_{63}\ C_{54}\ S_{54}\ C_{55}\ S_{55}$ |

Die einzelnen Koeffizientendreiecke für die Parametermengen BLK und DNS weichen schon von den normalerweise gezeigten Koeffizientendreiecken mit glatten Dreieckskanten ab (Abbildung 20(a) und 20(b)). Das Dreieck für die Kombinationslösung ist darüber hinaus insofern aussergewöhnlich, als dass die drei entstehenden Korrelationszonen *Independent*(rot), *Semi*(grün) und *Full*(blau) unregelmäßig sind und die Zone *Semi* nicht mehr zusammenhängend ist (Abbildung 20(c)). Trotzdem bringt der FKN Algorithmus 3.4 die Parameter in eine Reihenfolge, die eine Normalgleichungsmatrix mit Kitestruktur erzeugt und daher eine strenge Kombination der Datentypen für DNS und BLK erlaubt, ohne dass ein einziges Füllelement entsteht.




(a) Regelmäßige Daten (BLK)

(b) Unregelmäßige Daten (DNS)

(c) Kombination

(d) Kitematrix

Abbildung 20: Koeffizientendreiecke und Kitematrix für Beispiel 3.2

### 3.1.3 Auswahl der zu schätzenden Parameter

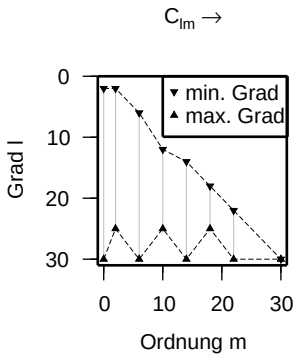
Die Menge der angesetzten Parameter wird durch die drei Vektoren $\boldsymbol{m}$, $\boldsymbol{\ell}_{\min}$ und $\boldsymbol{\ell}_{\max}$ dargestellt. Aus diesen Vektoren berechnet der FKN Algorithmus die zu verwendende Reihenfolge der Parameter. Werden zwei Daten-

typen mit unterschiedlicher Parametrisierung kombiniert, so sind pro Datensatz zwei Vektoren $\boldsymbol{\ell}_{\min}$ und $\boldsymbol{\ell}_{\max}$ notwendig, so dass die Eingabe für den FKN Algorithmus aus insgesamt fünf Vektoren besteht. Die Länge dieser Vektoren wird von der maximalen Ordnung bestimmt, die verwendet werden soll. Es wäre sehr mühsam, diese fünf Vektoren für größere Modelle von Hand aufzustellen. Daher gehört zum freien Kite–Nummerierungsschema auch eine Interpolation, die innerhalb des Koeffizientendreiecks ausgeführt wird. Damit ist es möglich, die notwendigen Eingabevektoren für den FKN Algorithmus komfortabel zu erstellen. Dazu werden für ausgesuchte Ordnungen minimale und maximale Grade angegeben, die als Stützpunkte für die Interpolation dienen. Die minimalen und maximalen Grade für alle anderen Ordnungen werden zwischen diesen Stützpunkten, d. h. entlang der gestrichelten Linien in Abbildung 21(b), linear interpoliert. Abbildung 21(a) zeigt ein Beispiel für die Angabe einiger Stützpunkte für einen Datentyp.

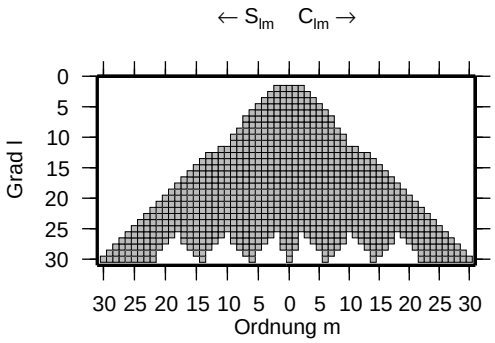
```
# maximale Ordnung
OMAX 30

# Ordnung / min. Grad / max. Grad
B 0 2 30
B 2 2 25
B 6 6 30
B 10 12 25
B 14 14 30
B 18 18 25
B 22 22 30
B 30 30 30
```

(a) Angabe der Stützpunkte für die Interpolation der minimalen und maximalen Grade der Ordnungen




(b) Graphische Darstellung der Stützpunkte




(c) Koeffizientendreieck mit den zwischen den Stützpunkten interpolierten minmalen und maximalen Graden

Abbildung 21: Auswahl der zu schätzenden Parameter durch Angabe von Stützpunkten

Aus den Angaben in Abbildung 21(a) werden die drei Vektoren $\boldsymbol{m}$, $\boldsymbol{\ell}_{\min}$ und $\boldsymbol{\ell}_{\max}$ berechnet, mit denen die notwendige Reihenfolge der Parameter bestimmt wird. Die Syntax dieser Datei ist sehr einfach, das Schlüsselwort OMAX gibt die maximale Ordnung, d. h. die Länge der drei Vektoren $\boldsymbol{m}$, $\boldsymbol{\ell}_{\min}$ und $\boldsymbol{\ell}_{\max}$ an. Die nachfolgenden Zeilen enthalten jeweils in der angegebenen Reihenfolge eine Spezifizierung des Datentyps (B=BLK oder D=DNS), die Ordnung und den minimalen und den maximalen Grad für diese Ordnung. Wird nur ein Datentyp angegeben, so entstehen durch die Interpolation drei Vektoren. Werden zwei Datentypen angegeben, so entstehen fünf Vektoren, jeweils $\boldsymbol{\ell}_{\min}$ und $\boldsymbol{\ell}_{\max}$ für beide Datentypen und den Vektor $\boldsymbol{m}$ der Ordnungen.

## 3.2 Notation innerhalb der Kitematrix

Um die entwickelten Algorithmen, die mit der Kitematrix arbeiten, erläutern zu können, sei an dieser Stelle die verwendete Notation an einem Beispiel erläutert. Die Kitematrix wird mit $\boldsymbol{K} \in \mathbb{R}^{n \times n}$ bezeichnet, wobei $n$ die Anzahl der geschätzten Parameter ist. $\boldsymbol{K}$ wird als $\eta \times \eta$ Blockmatrix $(\boldsymbol{K}_{\alpha\beta})$ dargestellt. Ein Block $\boldsymbol{K}_{\alpha\beta}$ mit $1 \leq \alpha, \beta \leq \eta$ hat die Dimension $n_\alpha \times n_\beta$. Hierbei gilt $n = \sum_{\alpha=1}^{\eta} n_\alpha$. Mit der Blockkoordinate $\tau$ wird die Grenze festgelegt, bis zu der nur Diagonalblöcke und keine Nebendiagonalblöcke vorhanden sind. Diese Grenze ist für den Cholesky Algorithmus wichtig und wird deshalb *Reduktionsgrenze* genannt. Weiterhin existiert in der Kitematrix ein vollbesetzer Block $\boldsymbol{D} \in \mathbb{R}^{n_d \times n_d}$. Dieser Block beinhaltet alle Diagonalblöcke $\boldsymbol{K}_{\tau+1\tau+1}$ bis $\boldsymbol{K}_{\eta\eta}$. Die Dimension

$$n_d = \sum_{\alpha=\tau+1}^{\eta} n_\alpha \tag{16}$$

ist die Summe aller Blockgrößen jenseits der Reduktionsgrenze. Alle Nebendiagonalblöcke $\boldsymbol{K}_{\alpha\beta}$ liegen über $\boldsymbol{D}$. Im Folgenden werden für die Indizierung von Blöcken griechische Buchstaben verwendet, um sie von der skalaren Indizierung zu unterscheiden (Abbildung 22). Die Besetztheitsstruktur der Kitematrix ist durch folgende Eigenschaften festgelegt, welche aufgrund der Symmetrie nur an der oberen Dreiecksmatrix erläutert werden:

- jedes Diagonalelement $k_{ii}$ ist Diagonalelement genau eines quadratischen Diagonalblocks, d. h. jede beliebige Zeile $\boldsymbol{K}(i,:)$ und jede beliebige Spalte $\boldsymbol{K}(:,i)$ geht genau durch einen Diagonalblock,

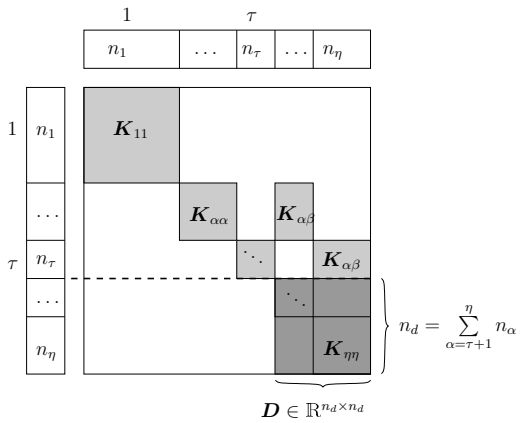



Abbildung 22: Für die Kitematrix verwendete Notation

- zu jedem Diagonalblock außer dem letzten gibt es genau null oder einen Nebendiagonalblock. Diese Blöcke werden *Subblöcke* genannt,
- ein Subblock beginnt und endet in derselben Zeile wie benachbarte Diagonalblock; er muss jedoch nicht quadratisch sein,
- alle Subblöcke liegen oberhalb des letzten Diagonalblocks,
- jeder Subblock liegt in einer eigenen Blockspalte.

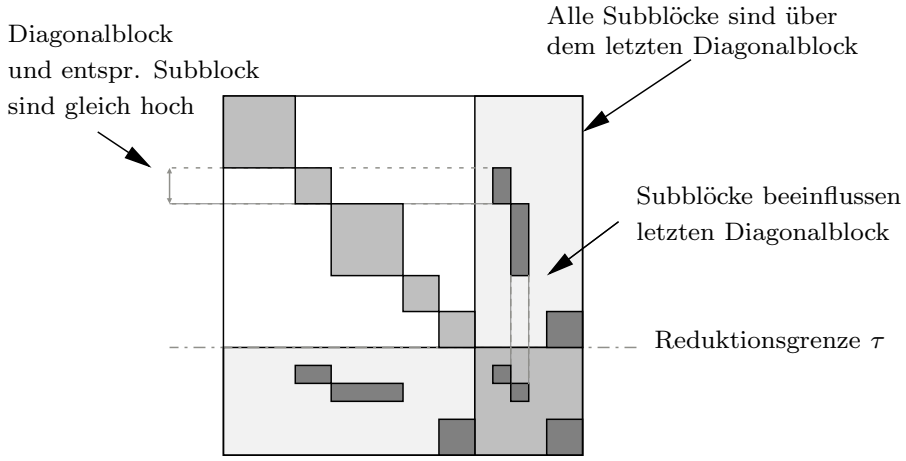



Abbildung 23: Eigenschaften einer Kitematrix

Im Zuge der Programmentwicklung von `pcgma` ist ein speziell für die Eigenschaften der Kitematrix ausgelegtes Speicherschema entwickelt und implementiert worden. Dieses Speicherschema erlaubt das effiziente Laden und Speichern mit minimalem Speicherplatzbedarf sowie das Senden und Empfangen der kompletten Kitematrix über MPI.

## 3.3 Berechnung der Positionen der Nichtnullelemente

Aufgrund der Definitionen der Zonen und der oben getroffenen Annahmen bestehen zwischen einzelnen Parametern Korrelationen. Durch die vorher bestimmte, symbolische Reihenfolge der Parameter wird die Normalgleichungsmatrix so umsortiert, dass folgende Operationen der Linearen Algebra mit der Kitematrix ohne die Entstehung von Füllelementen möglich sind:

- Choleskyzerlegung der Kitematrix $\boldsymbol{K}$, hierbei entstehen keine Füllelemente,

- Inversion der Kitematrix $\boldsymbol{K}$ unter Vernachlässigung entstehender Füllelemente. Es entsteht eine unvollständige Inverse $\boldsymbol{K}^{-1}_{\text{unvollst.}}$ der Kitematrix.

- Lösung des Gleichungssystems $\boldsymbol{KX} = \boldsymbol{B}$ mit Hilfe der Choleskyzerlegung $\boldsymbol{K} = \boldsymbol{R}^T\boldsymbol{R}$ der Kitematrix. Die Matrix $\boldsymbol{B}$ kann mehrere rechte Seiten enthalten, so dass es möglich ist mehrere Lösungen gleichzeitig zu berechnen. Diese Möglichkeit wurde vorgesehen, um später den Algorithmus von ALKHATIB und SCHUH (2006) zur Berechnung der Kovarianzmatrix der geschätzen Parameter mit Hilfe der Monte–Carlo Simulation berechnen zu können.

Um die Kitematrix aufstellen zu können, braucht man jedoch über den symbolischen Parametervektor hinaus die Positionen der Nichtnullelemente in der Kitematrix. Ein Nichtnullelement in der Kitematrix repräsentiert die Korrelation zwischen zwei Parametern. Zur Berechnung der Korrelationen wird der Algorithmus 3.5 verwendet, der jeden Parameter mit allen nachfolgenden Parametern vergleicht und anhand der Informationen Kosinus, Sinus, Ordnung, Parität (Geradheit) der Grade und der Vektoren der minimalen und maximalen Grade ermittelt, ob die beiden Parameter korreliert sind. Sind sie korreliert, wird die Position des resultierenden Nichtnullelements gespeichert. Die Struktur der Kitematrix $\boldsymbol{K} \in \mathbb{R}^{n\times n}$ wird in einem symbolischen Parametervektor $\boldsymbol{p} \in \mathbb{R}^{n\times 1}$ gespeichert, dessen Elemente Objekte mit drei Einträgen sind: Grad $\ell$, Ordnung $m$ und die Information über die trigonometrische Funktion $CS$, die gleich 'C' oder gleich 'S' sein kann.

**Algorithmus 3.5 (Berechnung der Korrelationen)**

```
1  void nonzeros( vector<nonzero> &nz, vector<Parameter> &Reihenfolge, vector<Parameter> &Full )
2  {
3     int n = Reihenfolge.size();                  // Anzahl der Parameter
4
5     for( int i = 0; i < n; ++i )
6        for( int j = i; j < n; ++j )
7           if( correlated( Reihenfolge[i], Reihenfolge[j], Full ) )
8              nz.push_back( nonzero(i,j) );      // Indizes speichern
9  }
10
11
12
```

Die Berechnung der Positionen der Nichtnullelemente in der Kitematrix ist innerhalb des FKN die rechenintensivste Arbeit, da jede Position der Korrelationsmatrix einzeln ausgewertet werden muss. Dies sind bei $n$ unbekannten Parametern $n^2$ Positionen. Da die Korrelationsmatrix jedoch per Definition symmetrisch ist, brauchen nur $n(n+1)/2$ Positionen ausgewertet werden. Dazu sind in Algorithmus 3.5 zwei verschachtelte Schleifen notwendig. Die äussere Schleife iteriert über alle Parameter des symbolischen Parametervektors $\boldsymbol{p}$ und die innere Schleife iteriert über alle Parameter ab der aktuellen Position. Die Funktion correlated in Algorithmus 3.5 prüft mit Algorithmus 3.6, ob die zwei symbolisch dargestellten Parameter korreliert sind.

**Algorithmus 3.6**

```
1  bool correlated( Parameter p1, Parameter p2, vector<Parameter> Full )
2  {
3     if( p1.isElementOf( Full ) and p2.isElementOf( Full ) )
4        return true;                    // p1 und p2 sind korreliert
5
6     if( p1.m() == p2.m()               // gleiche Ordnung
7           and
8           p1.l() % 2 == p2.l() % 2    // gleiche Parität (Geradheit) des Grades
9           and
10          p1.CS() == p2.CS()          // gleiche trigonometrische Funktion
11       )
12          return true;                // p1 und p2 korreliert
13
14    return false;                     // p1 und p2 nicht korreliert
15 }
16
```

Als Ergebnis liefert Algorithmus 3.5 die Koordinaten der Nichtnullelemente der Kitematrix in einer Tabelle von

x-y–Paaren, wie in Tabelle 4 am Beispiel der Kitematrix aus Abbildung 24 gezeigt. Da die Nichtnullelemente in der Kitematrix immer in Blöcken gruppiert sind, kann die Speicherung der Koordinaten stark vereinfacht werden. Hierzu wird von einem Block der Kitematrix jeweils die Zeile und Spalte des ersten Elementes und die Anzahl der Zeilen und Spalten des Blocks gespeichert. Um auf diese blockweise Darstellung der Nichtnullelemente zu kommen, wird die Tabelle der Nichtnullelemente (Tabelle 4) untersucht, indem das Inkrement $\Delta x$ bzw. $\Delta y$ zwischen benachbarten $x$– bzw. $y$–Werten gebildet wird. Die Nichtnullelemente werden nach (17) in eine zeilenweise Darstellung transformiert, bei der pro Zeile zwei verkettete Listen mit den Bezeichnungen `block` und `subblock`, angelegt werden. Jedes Element einer Zeile wird einer der beiden Listen zugeordnet. Beispielsweise wird die zeilenweise Darstellung aus Tabelle 5 mit (17) aus Tabelle 4 erhalten. Aus dieser zeilenweisen Darstellung der Nichtnullelemente kann die Beschreibung der Blöcke der Kitematrix extrahiert werden. Dazu muss in der Tabelle 5 für jeden Diagonalblock (`block`) die erste Zeile aufgesucht werden, welche alle notwendigen Informationen zur Beschreibung der Blöcke enthält. Die erste Zeile eines Diagonalblocks ist daran zu erkennen, dass das erste Element der Liste `block` grösser ist, als in der vorhergehenden Zeile. Dieser Zeile können die gesuchten Koordinaten und Seitenlängen des Blocks und des eventuell vorhandenen Subblocks mit (18) entnommen werden. Beispielsweise wird Tabelle 6 mit (18) aus Tabelle 5 erhalten.

$$\Delta x \begin{cases} = 1 & \text{neue Zeile, aktuelle Liste ist } \texttt{block},\ y\text{–Wert an aktuelle Liste anhängen} \\ = 0 \wedge \Delta y = 1 & y\text{–Wert an aktuelle Liste anhängen} \\ = 0 \wedge \Delta y > 1 & \text{aktuelle Liste ist } \texttt{subblock},\ y\text{–Wert an aktuelle Liste anhängen} \end{cases} \tag{17}$$

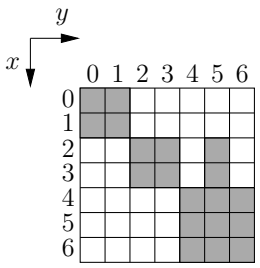



Abbildung 24: Beispiel für eine Kitematrix zur Ermittlung der Blockkoordinaten

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x$ | 0 | 0 | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 | 4 | 4 | 4 | 5 | 5 | 5 | 6 | 6 | 6 |
| $y$ | 0 | 1 | 0 | 1 | 2 | 3 | 5 | 2 | 3 | 5 | 4 | 5 | 6 | 4 | 5 | 6 | 4 | 5 | 6 |

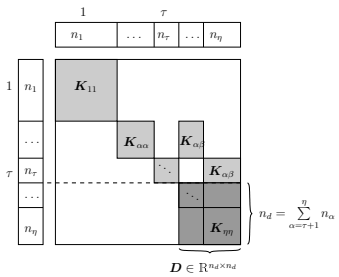
Tabelle 4: Koordinaten der Nichtnullelemente aus Abbildung 24 als Ergebnis von Algorithmus 3.5

| Nummer | `block` | | | `subblock` |
|---|---|---|---|---|
| 0 | 0 | 1 | | |
| 1 | 0 | 1 | | |
| 2 | 2 | 3 | | 5 |
| 3 | 2 | 3 | | 5 |
| 4 | 4 | 5 | 6 | |
| 5 | 4 | 5 | 6 | |
| 6 | 4 | 5 | 6 | |

Tabelle 5: Zeilenweise Repräsentation der Nichtnullelemente in Form von maximal zwei verketteten Listen pro Zeile

| Art | Zeile | Spalte | # Zeilen | # Spalten |
|---|---|---|---|---|
| block | 0 | 0 | 2 | 2 |
| block | 2 | 2 | 2 | 2 |
| subblock | 2 | 5 | 2 | 1 |
| block | 4 | 4 | 3 | 3 |

Tabelle 6: Blockkoordinaten der Kitematrix aus Abbildung 24 extrahiert aus Tabelle 5




(a) Kitematrix

```
# created May 26 2005  6:37
# by class navigation (Compiled: May 20 2005,16:47:51)

# block  row    col    nrows  ncols filename
block      0      0    100    100   block0000.dat
block    100    100     99     99   block0001.dat
block    199    199     99     99   block0002.dat
block    298    298     99     99   block0003.dat
block    397    397     99     99   block0004.dat
block    496    496     99     99   block0005.dat
block    595    595    100    100   block0006.dat
block    695    695     99     99   block0007.dat
```

(b) ASCII file

Abbildung 25: Speicherung der Kitematrix mit Blockkoordinaten

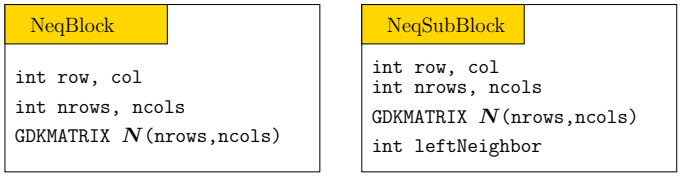
$$\text{block} \begin{cases} \text{Zeile, Spalte} & \text{Erstes Element der Liste } \texttt{block} \\ \text{Seitenlänge} & \text{Länge der Liste } \texttt{block} \end{cases}$$

$$\text{subblock} \begin{cases} \text{Zeile} & \text{Erstes Element der Liste } \texttt{block} \\ \text{Spalte} & \text{Erstes Element der Liste } \texttt{subblock} \\ \text{\# Zeilen} & \text{Länge der Liste } \texttt{block} \\ \text{\# Spalten} & \text{Länge der Liste } \texttt{subblock} \end{cases} \tag{18}$$

Abbildung 25(b) ist ein Beispiel für die blockorientierte Speicherung der Form der Kitematrix in einer ASCII-Datei. Mit den Informationen aus dieser Datei kann die Kitematrix blockweise mit Level 3 BLAS-Routinen (siehe Anhang A) aus der Designmatrix erstellt werden.

## 3.4 Speicherung der Kitematrix

Die Kitematrix wird in Form von zwei Vektoren gespeichert. Der erste Vektor heißt `block` und enthält alle Diagonalblöcke $\boldsymbol{K}_{\alpha\alpha}$ der Kitematrix bis zur Reduktionsgrenze $\tau$. Die Untermatrix $\boldsymbol{D}$ von $\boldsymbol{K}$ bildet als zusammenhängende Matrix das letzte Element des Vektors `block`. Der zweite Vektor heißt analog dazu `subblock` und enthält alle Subblöcke $\boldsymbol{K}_{\alpha\beta}$. Für die Repräsentierung eines Blocks wurde eine eigene Objektklasse entworfen. Sie enthält den jeweiligen Datenblock als eigenständige Matrix und die Koordinaten des ersten Elementes




(a) `NeqBlock` (b) `NeqSubBlock`

Abbildung 26: Objektklassen für die Darstellung der Blöcke innerhalb der Kitematrix

des Blocks bezogen auf die globalen, skalaren Koordinaten in der Kitematrix. In Anlehnung an den englischen Begriff "normal equation", wurde die Objektklasse `NeqBlock` genannt. Zwei Instanzen von `NeqBlock` können

addiert und verglichen sowie über MPI (siehe Anhang A) gesendet und empfangen werden. Eine Instanz von `NeqBlock` mit dem Bezeichner `Kb` kann in einem Programm z. B. über den Aufruf `Kb->Send(i)` an den Prozessor $i$ geschickt werden.
Um die Verknüpfung zwischen den Blöcken auf der Diagonalen und den Blöcken neben der Diagonalen, die für die oben genannten Algorithmen notwendig sind, herstellen zu können wurde von der Objektklasse `NeqBlock` eine Unterklasse `NeqSubBlock` abgeleitet. Diese verhält sich analog zur Basisklasse `NeqBlock`, besitzt jedoch noch die Information, neben welchem Diagonalblock sie in der Kitematrix liegt. Abbildung 26 zeigt graphisch die Struktur der Objektklassen `NeqBlock` und `NeqSubBlock`.
Alle Informationen über die Kitematrix sind programmintern in den beiden Vektoren `block` und `subblock`, die aus Elementen vom Typ `NeqBlock` bzw. `NeqSubBlock` bestehen, enthalten. Es sind keine weiteren Informationen gespeichert. Beim Aufbau der Kitematrix wird in der Membervariablen `NeqBlock::left` gespeichert, welcher Block der linke Nachbar ist. Abbildung 27 zeigt schematisch in welcher Weise die Kitematrix programmintern gespeichert wird. Es werden so wenig Informationen wie möglich gespeichert. Informationen, die für bestimmte Algorithmen notwendig sind, werden während der Laufzeit des jeweiligen Algorithmus aus diesen beiden Vektoren abgeleitet. Dieses Prinzip wurde aus zwei Gründen gewählt. Zum einen ist die Sicherheit der Implementierung dadurch höher, weil die Matrix immer konsistent gespeichert ist. Zum anderen ist die Implementierung weniger fehleranfällig, weil Kopier–, Speicher– und MPI–Versandoperationen ausschliesslich mit diesen beiden Vektoren arbeiten. Desweiteren ist die benötigte Rechenzeit für Operationen mit der Kitematrix im Verhältnis zum Gesamtproblem vernachlässigbar gering, so dass der Mehraufwand, der für das Ableiten von benötigten Informationen aus den Vektoren `block` und `subblock` notwendig ist, in Kauf genommen werden kann.

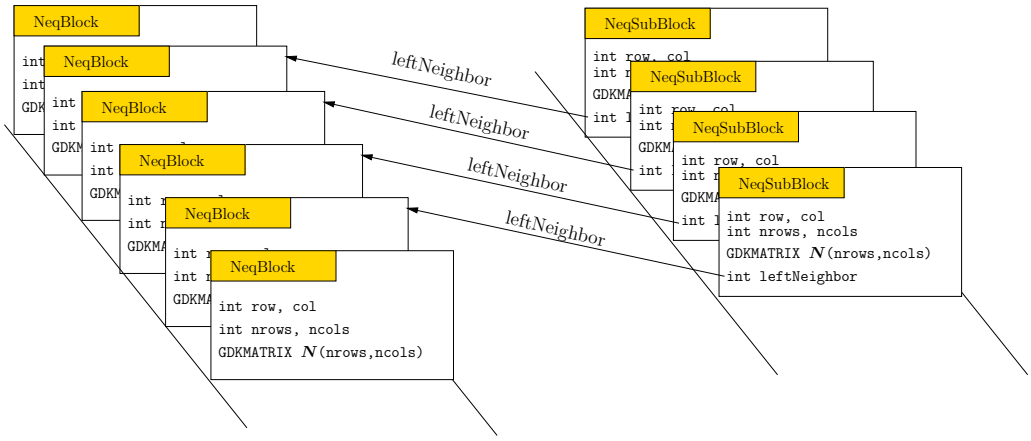



Abbildung 27: Speicherung der Kitematrix in zwei Vektoren

# 4 Bildung der Normalgleichungen

Die Aufstellung der Normalgleichungsmatrix ist bei großen Ausgleichungsproblemen wie z. B. der Satellitengradiometrie eine rechentechnische Herausforderung, da die Anzahl der Parameter und die Überbestimmung dort sehr hoch sind. Beispielsweise stehen bei einer Auflösung von Grad und Ordnung 300 den $n \approx 90\,000$ unbekannten Parametern $m \approx 46$ Millionen Beobachtungen gegenüber, die einem 6-monatigem Beobachtungszeitraum mit drei Messungen pro Sekunde entsprechen. Die Aufstellung der Normalgleichungsmatrix $\boldsymbol{N} \in \mathbb{R}^{n\times n}$ aus der Designmatrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ erfordert $\frac{1}{2}mn^2$ Additionen und $\frac{1}{2}mn^2$ Multiplikationen, wobei der Faktor $\frac{1}{2}$ aufgrund der Symmetrie der Normalgleichungsmatrix anzubringen ist. Eine Inversion der Normalgleichungsmatrix erfordert im Vergleich dazu $n^3$ Multiplikationen und $n^3$ Additionen. Der rechentechnische Aufwand zur Aufstellung der Normalgleichungsmatrix ist bei dem oben genannten Beispiel um den Faktor $\frac{1}{2}\frac{m}{n} = \frac{46\text{ Mio.}}{2\cdot 90\,000} \approx 250$ aufwändiger als die Inversion. Zusätzlich zu dem hohen Rechenaufwand ist auch der Speicherplatzbedarf für diese Operation und die Organisation der Speicherzugriffe zu berücksichtigen, denn die Designmatrix ist im obigen Beispiel ebenfalls um den Faktor 250 größer als die Normalgleichungsmatrix. Da bereits die Normalgleichungsmatrix in diesem Fall bei Speicherung ohne Berücksichtigung der Symmetrie ca. 60 Gigabyte Speicher erfordert, ist der Speicherplatzbedarf der Designmatrix mit 14 Terabyte zu groß für heutige Systeme. Es ist daher notwendig, eine Aufteilung der Designmatrix vorzunehmen, die sowohl die Reduktion des Speicherplatzbedarfs als auch die Verkürzung der Rechenzeit durch gleichzeitige Verwendung mehrerer Computer ermöglicht. Einen kurzen Überblick über die Möglichkeiten, parallel zu rechnen gibt Anhang A.2.
Ein weiterer wesentlicher Aspekt bei der Aufteilung der Berechnung der Normalgleichungsmatrix ist, dass die Geschwindigkeit der Berechnung, gemessen in Millionen Fließkommaoperationen pro Sekunde (MFlops/s), durch blockweise Verarbeitung in Verbindung mit auf die Hardware optimierten Bibliotheken gesteigert werden kann. Anhang A untersucht die Möglichkeit der Beschleunigung von Berechnungen mit optimierten Bibliotheken, gibt einen Überblick über verfügbare Bibliotheken und führt Vergleiche zwischen optimierten und nicht optimierten Routinen durch. Die höchsten Rechengeschwindigkeiten werden dort mit sogenannten Level-3 BLAS und LAPACK Routinen in Verbindung mit Blockgrößen von ca. $300 \times 300$ erreicht. Daher wird im Folgenden darauf hingewiesen, wenn es möglich ist, blockweise Berechnungen mit Level-3 Routinen durchzuführen.
Der erste Teil dieses Abschnitts erläutert die möglichen Strategien zur Aufteilung der Designmatrix bei der Aufstellung der vollbesetzten Normalgleichungsmatrix. Es wird sowohl der zeilenorientierte als auch der spaltenorientierte Zugriff betrachtet und es werden Möglichkeiten der Parallelisierung vorgestellt. Weiterhin werden die Algorithmen vorgestellt, die für das Lösen des Normalgleichungssystems sowie für die Ermittlung der Kovarianzmatrix der unbekannten Parameter verwendet werden. Der zweite Teil behandelt die Lösung von schwach besetzten Normalgleichungssystemen und geht auf die Entstehung von Füllelementen ein. Im dritten Teil werden die zuvor gezeigten Methoden an die Struktur und das Speicherschema der Kitematrix angepasst und die mit der Kitematrix verwendeten Algorithmen beschrieben.

## 4.1 Vollbesetzte Normalgleichungen

### 4.1.1 Aufteilung des Matrizenprodukts

Ausgehend von dem linearen Modell

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{\ell}\boldsymbol{v} \quad \text{mit} \quad \Sigma\{\boldsymbol{\ell}\} = \sigma^2\boldsymbol{I} \tag{19}$$

werden im Gauß–Markoff–Modell die Normalgleichungen

$$\boldsymbol{A}^T\boldsymbol{A}\,\hat{\boldsymbol{x}} = \boldsymbol{A}^T\boldsymbol{\ell} \tag{20}$$

$$\Leftrightarrow \boldsymbol{N}\hat{\boldsymbol{x}} = \boldsymbol{n} \tag{21}$$

mit den Schätzwerten $\hat{\boldsymbol{x}}$ für die unbekannten Parameter erhalten, siehe z. B. Koch (1999, S. 165).

**Spaltenweiser Zugriff:** Bei spaltenweisem Zugriff auf die Designmatrix $\boldsymbol{A}$ ergibt sich jedes Element $n_{ij}$ der Normalgleichungsmatrix $\boldsymbol{N}$ als Skalarprodukt der $i$-ten mit der $j$-ten Spalte

$$n_{ij} = \sum_{k=1}^{m} a_{ki}a_{kj}\,. \tag{22}$$

Skalarprodukte sind innerhalb der BLAS Routinen (siehe Anhang A) als Level-1 eingestuft und besitzen daher wenig Optimierungspotential. Sie sind, wenn möglich durch Matrizenprodukte zu ersetzen, die mit Level-3 BLAS

Routinen mit hohem Optimierungspotential berechnet werden können. Dazu ist die stehende Designmatrix, wie in Abbildung 28 gezeigt, in $\xi$ Blockspalten $\boldsymbol{A}_\alpha \in \mathbb{R}^{m \times n_\alpha}$

$$\boldsymbol{A} = \left(\boldsymbol{A}_\alpha\right) \quad \text{mit} \quad \alpha \in \{1, \ldots, \xi\} \quad \text{und} \quad n = \sum_{\alpha=1}^{\xi} n_\alpha \tag{23}$$

aufzuteilen. Durch diesen blockspaltenweisen Zugriff ergeben sich die $n_\alpha \times n_\beta$ Blöcke $\boldsymbol{N}_{\alpha\beta}$ der Normalgleichungsmatrix durch die Matrizenprodukte

$$\boldsymbol{N}_{\alpha\beta} = \boldsymbol{A}_\alpha^T \boldsymbol{A}_\beta \,. \tag{24}$$

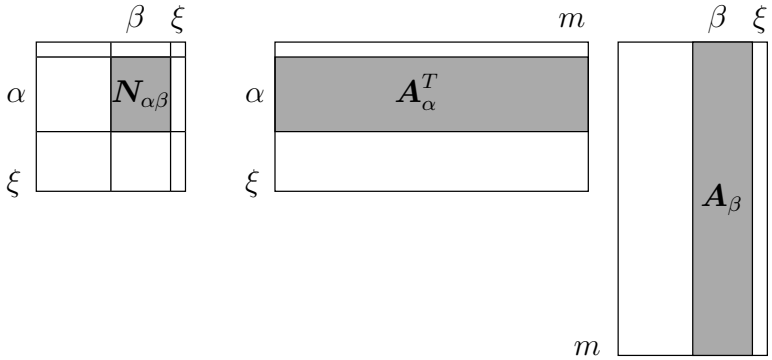



Abbildung 28: Aufteilung der Normalgleichungsmatrix durch blockspaltenweisen Zugriff

**Zeilenweiser Zugriff:** Bei zeilenweisem Zugriff ergibt sich die Normalgleichungsmatrix als Summe von Matrizen $\boldsymbol{N}^{(i)} \in \mathbb{R}^{n \times n}$, mit $i \in \{1, \ldots, m\}$, wobei die Matrizen $\boldsymbol{N}^{(i)}$ durch das dyadische Produkt

$$\boldsymbol{N}^{(i)} = \boldsymbol{a}_i^T \boldsymbol{a}_i \quad \text{mit} \quad \boldsymbol{A} = \begin{pmatrix} \boldsymbol{a}_1 \\ \boldsymbol{a}_2 \\ \vdots \\ \boldsymbol{a}_m \end{pmatrix} \tag{25}$$

der $i$-ten Zeile von $\boldsymbol{A}$ mit sich selbst gebildet werden. Dyadische Produkte sind innerhalb der BLAS-Routinen als Level-2 Operationen eingestuft und bieten ebenso, wie die Skalarprodukte des spaltenweisen Zugriffs wenig Optimierungspotential. Sie sind daher, um eine effiziente Berechnung zu erlauben, durch Matrizenprodukte auszutauschen. Dazu ist die Designmatrix in $\eta$ Blockzeilen $\boldsymbol{A}_\alpha \in \mathbb{R}^{m_\alpha \times n}$

$$\boldsymbol{A} = \left(\boldsymbol{A}_\alpha\right) \quad \text{mit} \quad \alpha \in \{1, \ldots, \eta\} \quad \text{und} \quad m = \sum_{\alpha=1}^{\eta} m_\alpha \tag{26}$$

aufzuteilen. Durch diesen blockzeilenweisen Zugriff ergibt sich die Normalgleichungsmatrix als Summe von Matrizen $\boldsymbol{N}^{(\alpha)}$, mit $\alpha \in \{1, \ldots, \eta\}$, die durch das Matrizenprodukt

$$\boldsymbol{N}^{(\alpha)} = \boldsymbol{A}_\alpha^T \boldsymbol{A}_\alpha \tag{27}$$

der Blockzeilen mit sich selbst gebildet werden.

### 4.1.2 Parallelisierung des Matrizenprodukts

Sowohl der spaltenweise als auch der zeilenweise Zugriff auf die Designmatrix lässt sich parallelisieren.

**Spaltenweiser Zugriff:** Beim spaltenweisen Zugriff ist dazu die Designmatrix in $\xi$ gleichgroße Blockspalten aufzuteilen, wobei $\xi$ die Anzahl der verwendeten Prozessoren ist. Bei der Parallelisierung mit vier Prozessoren kann die Designmatrix z. B. in die Blockspalten

$$\boldsymbol{A} = \left(\boldsymbol{A}_\alpha\right) \quad \text{mit} \quad \boldsymbol{A}_\alpha \in \mathbb{R}^{m \times n_\alpha}, \quad \alpha \in \{1, 2, 3, 4\} \quad \text{und} \quad n = \sum_{\alpha=1}^{4} n_\alpha \tag{28}$$

aufgeteilt werden, was einer Aufteilung der oberen Dreiecksmatrix der symmetrischen Normalgleichungsmatrix in 6 nicht symmetrische und 4 symmetrische Blöcke entspricht (Abbildung 29). Diese Blöcke werden den 4 Prozessoren so zugeordnet, dass jeder Prozessor die gleiche Anzahl Elemente der Normalgleichungsmatrix berechnet. Die Aufteilung führt auf den einzelnen Prozessoren zu den Berechnungen

| Prozessor | Berechnung | Benötigte Blockspalten |
|---|---|---|
| $P_1$ | $\boldsymbol{N}_{11} = \boldsymbol{A}_1^T\boldsymbol{A}_1,\ \boldsymbol{N}_{12} = \boldsymbol{A}_1^T\boldsymbol{A}_2,\ \boldsymbol{N}_{22} = \boldsymbol{A}_2^T\boldsymbol{A}_2$ | $\boldsymbol{A}_1,\ \boldsymbol{A}_2$ |
| $P_2$ | $\boldsymbol{N}_{13} = \boldsymbol{A}_1^T\boldsymbol{A}_3,\ \boldsymbol{N}_{23} = \boldsymbol{A}_2^T\boldsymbol{A}_3$ | $\boldsymbol{A}_1,\ \boldsymbol{A}_2\ \boldsymbol{A}_3$ |
| $P_3$ | $\boldsymbol{N}_{14} = \boldsymbol{A}_1^T\boldsymbol{A}_4,\ \boldsymbol{N}_{24} = \boldsymbol{A}_2^T\boldsymbol{A}_4$ | $\boldsymbol{A}_1,\ \boldsymbol{A}_2\ \boldsymbol{A}_4$ |
| $P_4$ | $\boldsymbol{N}_{33} = \boldsymbol{A}_3^T\boldsymbol{A}_3,\ \boldsymbol{N}_{34} = \boldsymbol{A}_3^T\boldsymbol{A}_4,\ \boldsymbol{N}_{44} = \boldsymbol{A}_4^T\boldsymbol{A}_4$ | $\boldsymbol{A}_3,\ \boldsymbol{A}_4$ |

welche zwar alle vier Prozessoren mit der gleichen Anzahl an Multiplikationen und Additionen belasten, aber viel Kommunikation zwischen den Prozessoren verursachen, da die Blockspalten $\boldsymbol{A}_1$ und $\boldsymbol{A}_2$ auf den Prozessoren 1 bis 3 und die Blockspalten $\boldsymbol{A}_3$ und $\boldsymbol{A}_4$ auf den Prozessoren $P_2$ bis $P_4$ benötigt werden. Aufgrund des Datenvo-

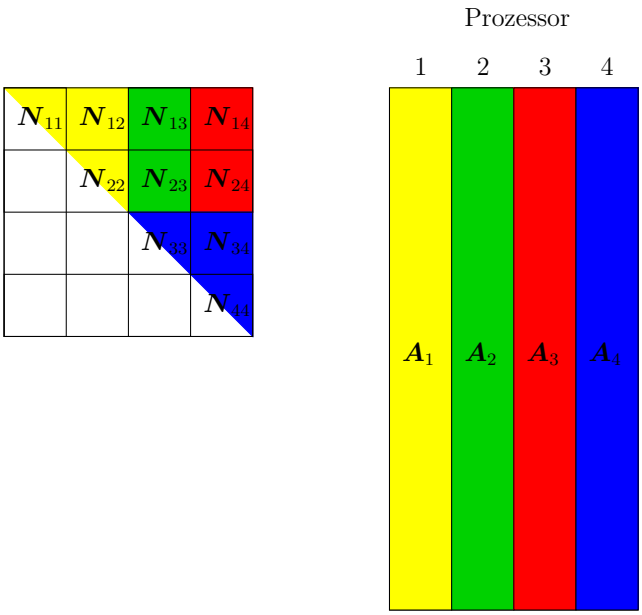



Abbildung 29: Aufteilung der Normalgleichungsmatrix auf 4 Prozessoren bei blockspaltenweisem Zugriff

lumens der Designmatrix würde der dadurch auftretende Netzwerkverkehr die Berechnung deutlich verzögern. In den meisten Fällen wird es daher günstiger sein, die Teile der Designmatrix, die auf einem Prozessor benötigt werden, auch von diesem berechnen zu lassen, wodurch der Netzwerkverkehr entfällt. Der Vorteil der Parallelisierung mit blockspaltenweisem Zugriff ist die Aufteilung der Normalgleichungsmatrix in Blöcke, die unabhängig voneinander im Netzwerk berechnet und gespeichert werden können, wodurch die Obergrenze für die Dimension der Normalgleichungsmatrix durch die Summe der Arbeitsspeicher aller beteiligten Computer vorgegeben ist. Als Nachteile können angeführt werden, dass die Aufstellung der Designmatrix nicht vollständig parallel ablaufen kann, wodurch die Effizienz dieses Verfahrens reduziert und der Aufwand für die Implementierung aufgrund der komplexen Organisation der Daten hoch ist.

**Zeilenweiser Zugriff:** Zur Parallelisierung des blockzeilenweisen Zugriffs wird die Normalgleichungsmatrix mit (26) und (27) als Summe der Matrizenprodukte

$$\boldsymbol{N} = \sum_{\alpha=1}^{\eta} \boldsymbol{A}_\alpha^T \boldsymbol{A}_\alpha \tag{29}$$

dargestellt, wobei $\eta$ die Anzahl der verfügbaren Prozessoren bezeichnet. Jeder Prozessor berechnet auf diesem Weg einen Summand $\boldsymbol{N}^{(\alpha)}$ der Normalgleichungsmatrix. Wie in Abbildung 30 dargestellt erfolgt sowohl die Aufstellung der Designmatrix als auch der Normalgleichungsmatrix mit (27) auf diesem Weg parallel. Sind al-

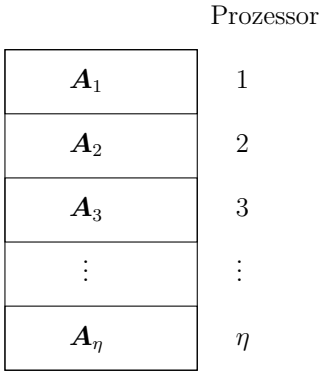



Abbildung 30: Blockzeilenweise Aufteilung der Designmatrix auf $\eta$ Prozessoren

le Summanden $\boldsymbol{N}^{(\alpha)}$ aufgestellt, so können sie in logarithmischer Zeit mit einem baumbasierten Algorithmus aufsummiert werden. Dieser Algorithmus wird z. B. durch die Routine `MPI_Reduce` des MPI Standards (siehe Anhang A.2) bereitgestellt. Vorteile der Parallelisierung mit blockzeilenweisem Zugriff sind die gute Skalierung, d. h. das lineare Verhältnis zwischen der Anzahl der Prozessoren und der Rechenzeit, und die einfache Implementierung des Verfahrens. Der Beitrag $\boldsymbol{N}^{(\alpha)}$ jedes Prozessors $P_\alpha$ zur Normalgleichungsmatrix $\boldsymbol{N}$ erfordert genausoviel Speicherplatz wie die Normalgleichungsmatrix selbst, was der entscheidende Nachteil dieses Verfahrens ist, da die Dimension der Normalgleichungsmatrix so durch die Größe des Arbeitsspeiches der einzelnen Computer begrenzt wird. Grundsätzlich ist die Kombination von blockzeilenweisem Zugriff mit blockspaltenweisem Zugriff möglich, deren Implementierung eine komplexe Aufgabenstellung ist.

### 4.1.3 Cholesky–Faktorisierung

Mit der Cholesky–Faktorisierung wird eine positiv definite, symmetrische Matrix $\boldsymbol{N}$ in das Produkt

$$\boldsymbol{N} = \boldsymbol{R}^T \boldsymbol{R} \tag{30}$$

einer oberen Dreiecksmatrix $\boldsymbol{R}$ mit ihrer Transponierten zerlegt. Der Algorithmus von Cholesky kann in Form von drei ineinandergeschachtelten Schleifen ausgedrückt werden. Er ist in die drei Schritte *Update*, *Cholesky* und *Substitution* zerlegbar. Für die Berechnung jedes Elementes von $\boldsymbol{R}$ muss ein Updateschritt gefolgt von einem Cholesky–Schritt oder einem Substitutionsschritt durchgeführt werden. Die Diagonalelemente $r_{ii}$ werden mit einem Updateschritt gefolgt von einem Cholesky–Schritt berechnet und die Nebendiagonalglieder werden mit einem Updateschritt gefolgt von einem Substitutionsschritt berechnet (Algorithmus 4.1), siehe z. B. Koch (1999, S. 31). Die drei Schritte der Cholesky–Zerlegung sind nun im einzelnen aufgeführt.

- Cholesky
  Wurzel aus einem Skalar ziehen. Dies wird nur auf das Diagonalglied angewendet.

- Update
  Skalarprodukt zweier Vektoren und Subtraktion zweier Skalare.
  Gemeint sind die Spaltenvektoren über der Spalte $i$ und über der Spalte $j$ (Abbildung 31). Hierbei kann die BLAS Routine `ddot` für das Skalarprodukt verwendet werden.

- Substitution
  Division durch einen Skalar. Es wird durch das Diagonalelement $r_{ii}$ geteilt.

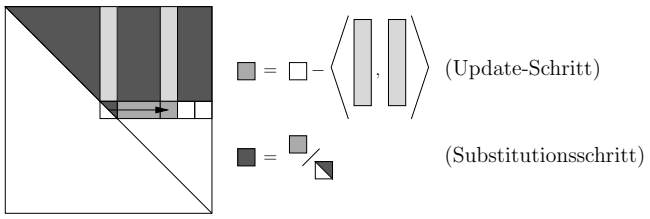



Abbildung 31: Updateschritt beim Cholesky–Verfahren

**Algorithmus 4.1 (Algorithmus von Cholesky)**

```
1  for  i = 1:n
2       for  j = i:n
3       {
4            for  k = 1:i−1
5                 r_ij = r_ij − r_ki r_kj  // Update
6
7            if( i == j )
8                 r_ii = √r_ii            // Cholesky
9            else
10                r_ij = r_ij/r_ii        // Substitution
11      }
```

**Blockweise Formulierung der Cholesky–Faktorisierung:** Durch den Übergang von Skalaren auf Blöcke innerhalb einer Matrix können die meisten Algorithmen analog zu ihrer skalaren Form auch als Blockalgorithmus ausgedrückt werden. Operationen mit Skalaren werden bei Blockalgorithmen durch ihre verwandten Matrizenoperationen ausgedrückt. Zum Beispiel kann aus einer Division durch einen Skalar eine Multiplikation mit der Inversen eines Blocks werden. Numerisch effizienter kann die Division durch Vorwärts– und Rückwärtseinsetzen mit der Cholesky–Faktormatrix einer Matrix durchgeführt werden. Ebenso kann die Multiplikation zweier Skalare durch die Multiplikation zweier Matrizen ersetzt werden. Vom Standpunkt der Optimierung ist die Verwendung von Blockalgorithmen immer zu empfehlen, weil hierbei Level-3 BLAS Routinen eingesetzt werden können, mit denen wie in Anhang A gezeigt, eine deutlich höhere Rechengeschwindigkeit erreicht werden kann. Der skalare Cholesky–Algorithmus (4.1) soll nun als Blockalgorithmus formuliert werden. Dazu wird die zu zerlegende symmetrische, positiv definite Matrix $\boldsymbol{N} \in \mathbb{R}^{n\times n}$ als $\eta \times \eta$ Blockmatrix $(\boldsymbol{N}_{\alpha\beta})$ mit $\boldsymbol{N}_{\alpha\beta} \in \mathbb{R}^{n_\alpha \times n_\beta}$ und $\sum_{\alpha=1}^{\eta} n_\alpha = n$ dargestellt. Eine genaue Herleitung und Analyse der Optimierungsmöglichkeiten der Cholesky–Faktorisierung findet sich in Golub und Van Loan (1996, Kap. 4.2.) oder auch in Schuh (2000, Kap. 2.4.1.4). Der Block–Cholesky–Algorithmus sieht folgendermaßen aus:

**Algorithmus 4.2 (Block–Cholesky)**

```
1  for  α = 1:η
2       for  β = α:η
3       {
4            for  γ = 1:α−1
5                 R_αβ = R_αβ − R_γα^T R_γβ        // Updateschritt
6
7            if( α == β )
8                 R_αα = cholesky(R_αα)            // Choleskyschritt
9            else
10                R_αα R_αβ^neu = R_αβ^alt         // Substitutionsschritt durch Rückwärtseinsetzen
11      }
```

### 4.1.4 Lösung eines Gleichungssystems mit dem Cholesky–Verfahren

Die Zerlegung nach (30) zur Lösung eines Gleichungssystems bezeichnet man als Cholesky–Verfahren. Zur Lösung des Gleichungssystems

$$\boldsymbol{N}\boldsymbol{x} = \boldsymbol{n} \quad \text{mit} \quad \boldsymbol{N} \in \mathbb{R}^{n\times n}, \quad \boldsymbol{x},\, \boldsymbol{n} \in \mathbb{R}^{n\times 1} \quad \text{und} \quad \boldsymbol{N} = \boldsymbol{N}^T \tag{31}$$

mit der symmetrischen, positiv definiten Matrix $\boldsymbol{N}$ ist es nicht notwendig, die Inverse $\boldsymbol{N}^{-1}$ zu bilden. Durch die Cholesky–Zerlegung der Matrix $\boldsymbol{N}$ wird das Gleichungssystem

$$\boldsymbol{R}^T\boldsymbol{R}\boldsymbol{x} = \boldsymbol{n} \tag{32}$$

erhalten, bei dem das Produkt aus der oberen Dreiecksmatrix und dem unbekannten Vektor $\boldsymbol{x}$ durch den unbekannten Hilfsvektor

$$\boldsymbol{c} = \boldsymbol{R}\boldsymbol{x} \tag{33}$$

ersetzt wird. Daraus resultiert das Gleichungssystem

$$\boldsymbol{R}^T\boldsymbol{c} = \boldsymbol{n} \tag{34}$$

mit einer unteren Dreiecksmatrix als Systemmatrix. Der unbekannte Vektor $\boldsymbol{c}$ kann in (34) durch Vorwärtseinsetzen mit (35) berechnet werden (siehe auch Abbildung 40(a)).

$$c_i = \left( n_i - \sum_{k=1}^{i-1} r_{ki}\, c_k \right) / r_{ii} \qquad i \in \{1, n\} \tag{35}$$

Nachdem der Hilfsvektor $\boldsymbol{c}$ nun bekannt ist, dient er als rechte Seite im Gleichungssystem $\boldsymbol{R}\boldsymbol{x} = \boldsymbol{c}$, dessen Systemmatrix die obere Dreiecksmatrix $\boldsymbol{R}$ ist. Dieses Gleichungssystem ist durch Rückwärtseinsetzen mit (36) lösbar (siehe auch Abbildung 42).

$$x_i = \left( c_i - \sum_{k=i+1}^{n} r_{ik}\, x_k \right) / r_{ii} \qquad i \in \{n, 1\} \tag{36}$$

Zur Lösung eines Gleichungssystems mit mehreren rechten Seiten ist die Cholesky–Zerlegung nur einmal vorzunehmen. Das Vorwärts- und Rückwärtseinsetzen erfordert $\frac{1}{2}\,n^2 + n$ kombinierte Additionen und Multiplikationen. Daher ist das wiederholte Lösen eines Gleichungssystems mit verschiedenen rechten Seiten über die Cholesky–Zerlegung eine effiziente Methode.

**Optimierung:** Das Vorwärts- und Rückwärtseinsetzen mit (35) und (36) sind Grundbausteine der linearen Algebra. Aus diesem Grund gibt es dafür in den BLAS die Level 3 Routine `dtrsm`. Von Vorteil für die Optimierung ist, dass diese Operation blockweise und mit mehreren rechten Seiten durchgeführt werden kann. Optimierte BLAS Routinen sind daher in der Lage, das Vorwärts- und Rückwärtseinsetzen effizient durchzuführen. Es ist empfehlenswert, `dtrsm` für alle vollbesetzten Gleichungssysteme mit einer Dreiecksmatrix zu verwenden.

### 4.1.5 Rekursive Berechnung der Inversen aus der Cholesky–Matrix

Um Aussagen über die Genauigkeit der geschätzten Parameter zu erhalten, ist die Inverse der Normalgleichungsmatrix zu bilden. Zur Berechnung der Inversen wird ein Algorithmus verwendet, der es bei dünnbesetzten Matrizen erlaubt, entstehende Füllelemente vollständig unberücksichtigt zu lassen. Dies ist zulässig, weil gezeigt werden kann, dass zur Berechnung der inversen Elemente an den Stellen der Nichtnullelemente die Füllelemente nicht benötigt werden. Durch die Vernachlässigung der Füllelemente wird die sogenannte *unvollständige Inverse* erhalten. Dieser Algorithmus wurde bereits von Schuh (1996) und Auzinger (1997) im Programm `pcgma` für die Berechnung der unvollständigen Inversen der Kitematrix verwendet und ist von Hanson (1978) vorgestellt worden.
Zur Berechnung der Inversen $\boldsymbol{A}^{-1} = \boldsymbol{B}$ einer positiv definiten, symmetrischen Matrix $\boldsymbol{A}$ teilt man beide Matrizen in jeweils zwei symmetrische, aber nicht unbedingt gleichgroße Diagonalblöcke, und zwei Nebendiagonalblöcke auf.

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{12}^T & \boldsymbol{A}_{22} \end{pmatrix} \qquad \boldsymbol{B} = \begin{pmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{12}^T & \boldsymbol{B}_{22} \end{pmatrix} \tag{37}$$

Die einzelnen Blöcke der Matrix $\boldsymbol{B}$ lassen sich durch die folgenden Formeln ausdrücken, siehe z. B. Koch (1999, S. 33):

$$\begin{aligned} \boldsymbol{B}_{22} &= (\boldsymbol{A}_{22} - \boldsymbol{A}_{12}^T(\boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12}))^{-1} \\ \boldsymbol{B}_{12} &= -(\boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12})\boldsymbol{B}_{22} \\ \boldsymbol{B}_{11} &= \boldsymbol{A}_{11}^{-1} - \boldsymbol{B}_{12}(\boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12})^T \end{aligned} \tag{38}$$

Im Folgenden ist wichtig, dass die Abhängigkeiten hierbei von unten nach oben verlaufen, und daher die Reihenfolge der Berechnung $\boldsymbol{B}_{22}, \boldsymbol{B}_{12}, \boldsymbol{B}_{11}$ sein muss. Abbildung 32 zeigt, dass die Blöcke der Inversen von $\boldsymbol{A}$ von unten nach oben berechnet werden. Da die Inverse der Matrix $\boldsymbol{A}$ aus (37) nicht aus der Matrix selbst, sondern

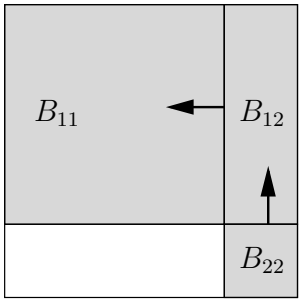



Abbildung 32: Inversion durch Partitionierung

aus ihrer Cholesky–Zerlegung $\boldsymbol{R}$ berechnet werden soll, muss diese zunächst mit Algorithmus 4.1 zerlegt werden, woraus (39) erhalten wird.

$$\boldsymbol{R} = \begin{pmatrix} \boldsymbol{R}_{11} & \boldsymbol{R}_{12} \\ \boldsymbol{0} & \boldsymbol{R}_{22} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_{11}^{\frac{1}{2}} & \boldsymbol{A}_{11}^{-\frac{1}{2}}\boldsymbol{A}_{12} \\ \boldsymbol{0} & (\boldsymbol{A}_{22} - \boldsymbol{A}_{12}^{T}\boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12})^{\frac{1}{2}} \end{pmatrix} \tag{39}$$

Durch Multiplikation jeder Zeile von $\boldsymbol{R}$ mit ihrem eigenen Diagonalelement entsteht eine Matrix, aus der die Inverse von $\boldsymbol{A}$ berechnet werden kann, weil der letzte Block bereits der inverse Block von $\boldsymbol{B}_{22}$ aus (37) ist. Diese Matrix ist identisch mit der oberen Diagonalmatrix aus dem Gauß'schen Algorithmus und wird im Folgenden mit $\boldsymbol{G}$ bezeichnet.

$$\boldsymbol{G} = \begin{pmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{0} & \boldsymbol{B}_{22}^{-1} \end{pmatrix} \tag{40}$$

Nach Anwendung des Gauß'schen Algorithmus bzw. Umformung der Cholesky–Matrix $\boldsymbol{R}$ in die Gauß'sche Matrix $\boldsymbol{G}$ ist das letzte Diagonalelement gleich dem reziproken Wert des letzten Diagonalelementes der Inversen $\boldsymbol{B} = \boldsymbol{A}^{-1}$. Die Inverse wird mit (38) rekursiv aus $\boldsymbol{G}$ berechnet, indem zunächst die Partitionierung

$$\boldsymbol{B}_{22} = (1/g_{nn}), \quad \boldsymbol{A}_{21} = (g_{n-1,n}), \quad \boldsymbol{A}_{11} = (g_{n-1,n-1}) \tag{41}$$

gewählt wird. Die Blockbezeichnung $\boldsymbol{A}_{ij}$ benennt hierbei nicht Elemente aus der Matrix $\boldsymbol{A}$, sondern die Partitionierung der in diesem Schritt zu berechnenden Teilinversen nach (38). Im nächsten Rekursionsschritt wird die Dimension von $\boldsymbol{B}_{22}$ um eins erhöht, so dass gilt

$$\boldsymbol{B}_{22} = \begin{pmatrix} b_{n-1,n-1} & b_{n-1,n} \\ b_{n,n-1} & b_{nn} \end{pmatrix}, \quad \boldsymbol{A}_{21} = (g_{n-2,n-1},\ g_{n-2,n}), \quad \boldsymbol{A}_{11} = g_{n-2,n-2}. \tag{42}$$

Auf diesem Wege wird die gesamte Inverse $\boldsymbol{B}$ rekursiv, zeilenweise von unten nach oben mit (38) berechnet. Der neu berechnete Block $\boldsymbol{B}_{22}$ wird als volle, symmetrische Matrix gespeichert, damit die Multiplikation mit $\boldsymbol{A}_{21}$ bequemer gerechnet werden kann. Abbildung 33 zeigt die Schritte der Rekursion, der gesamte Rechenverlauf zur Berechnung der Inversen aus der nach Cholesky reduzierten Matrix ist in Algorithmus 4.3 zusammengefasst.

**Algorithmus 4.3 (Rekursive Berechnung der Inversen aus einer Cholesky–Matrix)**

1 $\boldsymbol{A}^{-1}(n,n) = \frac{1}{\boldsymbol{R}(n,n)^2}$

2

3 **for** i = n−1:−1:1

4 $\quad \boldsymbol{A}^{-1}(i,i+1:n) = \dfrac{-1}{\boldsymbol{R}(i,i)}\ \boldsymbol{R}(i,i+1:n)\ \ \boldsymbol{A}^{-1}(i+1:n,i+1:n)$

5 $\quad \boldsymbol{A}^{-1}(i+1:n,i) = \left(\boldsymbol{A}^{-1}(i,i+1:n)\right)^T$

6 $\quad \boldsymbol{A}^{-1}(i,i) = \dfrac{1}{\boldsymbol{R}(i,i)^2} - \boldsymbol{A}^{-1}(i,i+1:n)\left(\dfrac{1}{\boldsymbol{R}(i,i)}\ \boldsymbol{R}(i,i+1:n)\right)^T$

7 **end**

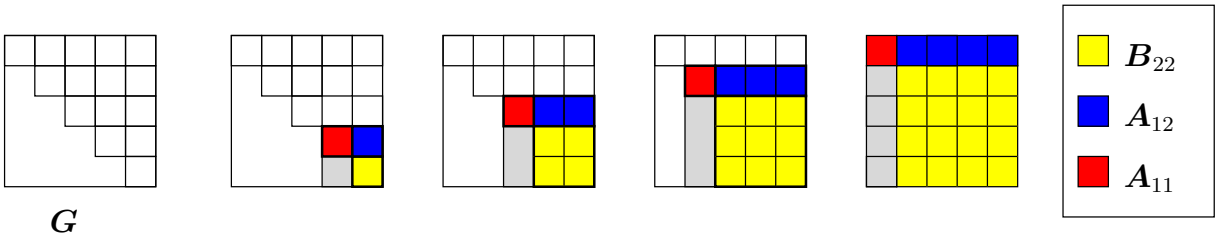



Abbildung 33: Rekursive Berechnung der Inversen

## 4.2 Zur Entstehung von Füllelementen bei der Cholesky–Zerlegung

Bei der Kombination von regelmäßigen und nicht regelmäßigen Daten entsteht bei ordnungsweiser Nummerierung der unbekannten Parameter ein Schachbrettmuster, was bereits in Abbildung 11 auf Seite 24 gezeigt wurde. Bei der Cholesky–Zerlegung dieser Matrix entstehen die in Abbildung 12(b) gezeigten Streifen und bei der Inversion der in Abbildung 12(c) gezeigte vollbesetzte Diagonalblock. Wird ein Speicherschema verwendet, das die schwache Besetzung der Matrix ausnutzt, so kann das Ergebnis der Berechnung in beiden Fällen aufgrund der Füllelemente nicht mehr mit diesem Speicherschema gespeichert sondern muss i. d. R. voll gespeichert werden, weshalb die Berechnung unter Umständen nicht mehr durchführbar ist. Das Freie Kite–Nummerierungsschema bringt die unbekannten Parameter, wie schon erwähnt in eine Reihenfolge, welche die Kitestruktur in der Normalgleichungsmatrix erzeugt, wodurch eine Cholesky–Faktorisierung ohne die Entstehung von Füllelementen möglich ist. Die Füllelemente bei der Inversion können im Gegensatz dazu nicht durch die Umsortierung mit dem Freien Kite–Nummerierungsschema verhindert werden, ihre Berechnung wird stattdessen, wie später gezeigt werden wird, umgangen.
Der Algorithmus von Cholesky kann in verschiedensten Varianten notiert werden. Es wird unterschieden zwischen zeilenorientiert und spaltenorientiert bzw. blockweisem und elementweisem (*skalaren*) Algorithmus. Da die Cholesky–Matrix $\boldsymbol{R}$ der Zerlegung (30) jedoch eindeutig ist, müssen alle Varianten zum gleichen Ergebnis führen. Eine Variante des Algorithmus von Cholesky ist Algorithmus 4.1. Ein Element $n_{ij}$ der Normalgleichungsmatrix $\boldsymbol{N}$ wird, wie bereits in Abschnitt 4.1.3 erläutert, durch einen Updateschritt gefolgt von entweder einem Choleskyschritt (Diagonalelement) oder einem Substitutionsschritt (Nebendiagonalelement) in das entsprechende Element $r_{ij}$ der Matrix $\boldsymbol{R}$ der Choleskyfaktoren transformiert. Da eine Null nur durch Addition oder Subtraktion einer Zahl in eine von Null verschiedene Zahl transformiert werden kann, nicht aber durch Multiplikation oder Wurzelziehen, kann ein Füllelement nur beim Update-Schritt entstehen. Abbildung 34 zeigt die Choleskyzerlegung der ersten Zeile. Da der Updateschritt die Fortpflanzung der Information der bereits transformierten Zeilen nach unten ist, werden durch ihn in der ersten Zeile noch keine Operationen durchgeführt.

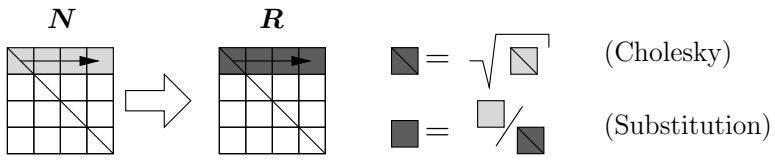



Abbildung 34: Choleskyzerlegung der ersten Zeile

In der ersten Zeile der Cholesky–Matrix können daher keine Füllelemente entstehen. Erst ab der zweiten Zeile enthält der Updateschritt Operationen, durch die Füllelemente entstehen können. In den Zeilen 4 und 5 von Algorithmus 4.1 ist abzulesen, dass der Updateschritt zum Element $r_{ij}$ das Skalarprodukt der Teilspalte über dem Element $r_{ij}$ und der Teilspalte über dem Element $r_{ii}$ bildet und von dem Ausgangselement $n_{ij}$ abzieht. Dieser Vorgang ist in Abbildung 35 dargestellt.
Der Updateschritt erzeugt daher nur ein Füllelement, wenn das ursprüngliche Element ein Nullelement ist und die Teilspalte $\boldsymbol{R}(1 : i-1, i)$ und die Teilspalte $\boldsymbol{R}(1 : i-1, j)$ linear abhängig sind. Die Entstehung und Fortpflanzung der Füllelemente über mehrere Schritte hinweg ist nicht intuitiv erfassbar, da die Matrix sich nach jedem Schritt verändert. Festzuhalten ist jedoch, dass Füllelemente sich nach unten fortsetzen können und zwar solange, wie in der gleichen Zeile Nichtnullelemente über dem Diagonalelement vorhanden sind. Ein besonders anschauliches Beispiel liefert die Blockdiagonalstruktur mit kleineren Nebendiagonalelementen, die in Abbildung

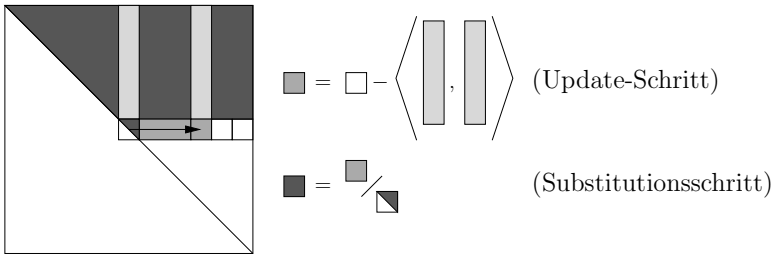



Abbildung 35: Skalarprodukte von Teilspalten innerhalb des Algorithmus von Cholesky

11 als Schachbrettmuster bereits im Zusammenhang der Datenkombination gezeigt wurde. Die Entstehung eines Blocks von Füllelementen unter einem Nebendiagonalblock ist auf den größeren Diagonalblock neben ihm zurückzuführen. Abbildung 36 veranschaulicht die Entstehung der Füllelemente unter den Nebendiagonalblöcken. Fünf weitere Beispiele für die Entstehung von Füllelementen durch die Cholesky–Zerlegung sind in Abbildung 37 gegeben. In der ersten Zeile sind die Ausgangsmatrizen gezeigt und in der zweiten Zeile die entsprechenden Cholesky–Faktorisierungen.

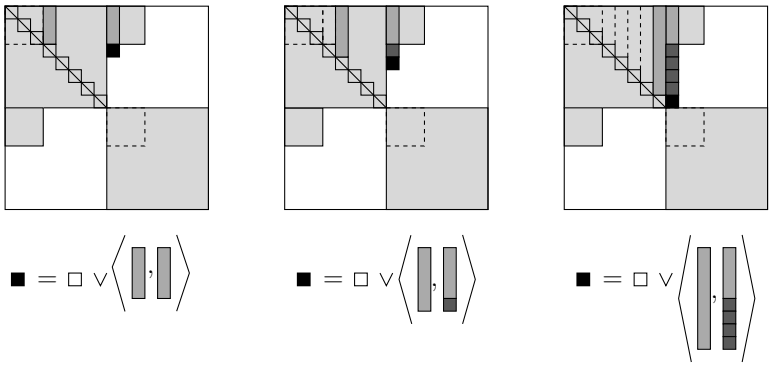
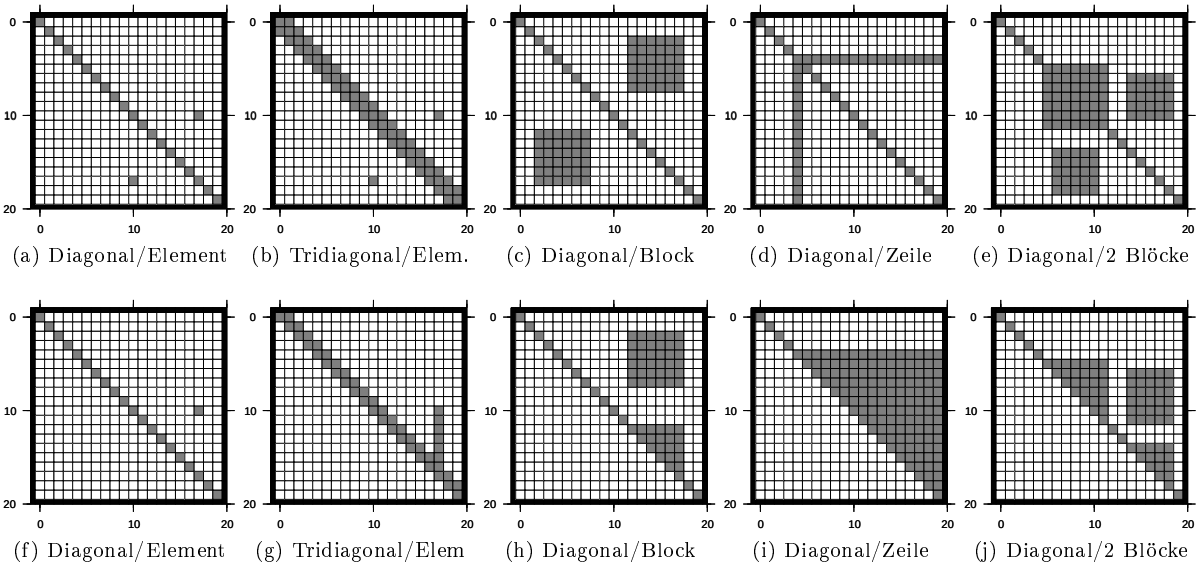


Abbildung 36: Entstehung von Füllelementen beim Schachbrettmuster aus der Datenkombination

## 4.3 Schwach besetzte Normalgleichungen mit FKN

### 4.3.1 Parallele Aufstellung der Kitematrix

Die Kitematrix setzt sich, wie in Abbildung 18 gezeigt, aus dem Beitrag des regelmäßigen Datensatzes und dem Beitrag des nicht regelmäßigen Datensatzes zusammen. Der Beitrag des nicht regelmäßigen Datensatzes soll für die vorliegende Aufgabenstellung als vollbesetzte Normalgleichungsmatrix vorliegen. Der Beitrag des regelmäßigen Datensatzes (Abbildung 18(a)) muss dagegen aus der Designmatrix berechnet werden und soll im Folgenden mit $\boldsymbol{K}$ bezeichnet werden. Das Datenvolumen der hochauflösenden, aber schwach besetzen Normalgleichungsmatrix $\boldsymbol{K}$ ist im Vergleich zu einer ebenso hoch auflösenden, vollbesetzten Normalgleichungsmatrix sehr gering, wie aus Tabelle 7 zu entnehmen ist. Die Angaben dieser Tabelle sind aus der Ausgabe von Algorithmus 3.5, welcher die Positionen der Nichtnullelemente berechnet, abgeleitet. Für die Aufstellung der Kitematrix hat die Beschränkung der Methode des blockzeilenweisen Zugriffs aus (29), dass die Dimension der Normalgleichungsmatrix durch den Arbeitsspeicher der einzelnen Computer begrenzt ist, daher wenig Gewicht. Aufgrund der schwachen Besetztheit der Kitematrix ist der rechentechnische Aufwand bei ihrer Aufstellung entsprechend den Angaben in Tabelle 7 kleiner. Im Gegensatz dazu wird der rechentechnische Aufwand für die Aufstellung der Designmatrix durch die schwache Besetztheit der Kitematrix nicht kleiner, denn in der Kitematrix sind stets alle Hauptdiagonalelemente ungleich Null, wodurch alle Zeilen der Designmatrix benötigt werden, wie Abbildung 38 veranschaulicht. Zur Beschleunigung der Aufstellung der Kitematrix wird daher die Parallelisierung mit blockzeilenweisem Zugriff auf die Designmatrix verwendet. Zusätzlich zur Aufteilung der Designmatrix auf die Prozessoren wird die Designmatrix auf den einzelnen Prozessoren weiter in Blöcke aufgeteilt, um den



(a) Diagonal/Element (b) Tridiagonal/Elem. (c) Diagonal/Block (d) Diagonal/Zeile (e) Diagonal/2 Blöcke

(f) Diagonal/Element (g) Tridiagonal/Elem (h) Diagonal/Block (i) Diagonal/Zeile (j) Diagonal/2 Blöcke

Abbildung 37: Beispiele für die Entstehung von Füllelementen bei der Choleskyzerlegung

| **Maximale Auflösung [ Grad / Ordnung ]** | **Anzahl Parameter [-]** | **Normalgleichungs-matrix [ Gigabyte ]** | **Kitematrix [ Megabyte ]** | **Prozent der Ngl [%]** |
|---|---|---|---|---|
| 60 | 3717 | 0.1 | 0.3 | 0.29% |
| 90 | 8277 | 0.5 | 1.0 | 0.19% |
| 120 | 14637 | 1.6 | 2.3 | 0.14% |
| 150 | 22797 | 3.9 | 4.5 | 0.11% |
| 180 | 32757 | 8.0 | 7.7 | 0.09% |
| 210 | 44517 | 14.8 | 12.1 | 0.08% |
| 240 | 58077 | 25.1 | 18.0 | 0.07% |
| 270 | 73437 | 40.2 | 25.6 | 0.06% |
| 300 | 90597 | 61.2 | 35.0 | 0.06% |
| 330 | 109557 | 89.4 | 46.5 | 0.05% |
| 360 | 130317 | 126.5 | 60.3 | 0.05% |

Tabelle 7: Datenvolumen des SGG Beitrags zur Kitematrix

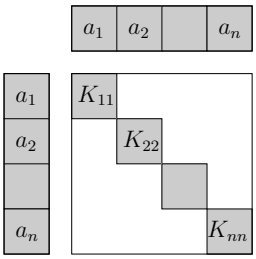



Abbildung 38: Für das Update der Diagonalelemente der Kitematrix $\boldsymbol{K}$ wird jeweils eine vollständige Zeile $\boldsymbol{a}$ der Designmatrix $\boldsymbol{A}$ benötigt. Zur Aufstellung der Kitematrix wird deshalb die komplette Designmatrix benötigt.

Speicherplatzbedarf zu reduzieren. Ausgehend von einer Designmatrix

$$\boldsymbol{A} \in \mathbb{R}^{m \times n} \tag{43}$$

mit $m$ Messungen und $n$ unbekannten Werten soll nun die Kitematrix mit $\mathcal{P}$ Prozessoren parallel berechnet werden. Dazu wird zunächst die Designmatrix $\boldsymbol{A}$ in $\eta$ Blöcke

$$\boldsymbol{A} = \left(\boldsymbol{A}_\alpha\right), \quad \boldsymbol{A}_\alpha \in \mathbb{R}^{b \times n}, \quad \alpha \in 1, \ldots, \eta \tag{44}$$

aufgeteilt, wobei die Blockgröße $b$ so gewählt wird, dass die Blöcke $\boldsymbol{A}_\alpha$ in den Arbeitsspeicher der einzelnen Computer passen. Anschliessend werden die $\eta$ Blöcke auf die $\mathcal{P}$ Prozessoren verteilt, wodurch jeder Prozessor

$$r = \frac{\eta}{\mathcal{P}} \tag{45}$$

Blöcke erhält, so dass jeder Prozessor $p$ einen Summand der Kitematrix

$$\boldsymbol{K}^{(p)} \quad \text{mit} \quad \boldsymbol{K} = \sum_{p=0}^{\mathcal{P}-1} \boldsymbol{K}^{(p)} \tag{46}$$

berechnet, welcher anschliessend über das Netzwerk an den Prozessor mit dem MPI Rang 0 (siehe Anhang A.2) verschickt und dort zur Kitematrix addiert wird. Die Berechnung der Summanden $\boldsymbol{K}^{(p)}$ aus (46) auf den einzelnen Prozessoren wird jedoch nicht durch ein einfaches Matrizenprodukt wie in (27) berechnet, sondern mit dem blockspaltenweisen Zugriff aus (24) kombiniert, um nur die Blöcke der Kitematrix

$$\boldsymbol{K}^{(\alpha)} = \left(\boldsymbol{K}_{\gamma\delta}\right) \quad \text{mit} \quad \boldsymbol{K}_{\gamma\delta} \in \mathbb{R}^{n_\gamma \times n_\delta} \quad \text{und} \quad n = \sum_{\gamma=1}^{\mu} n_\gamma \tag{47}$$

berechnen zu können. Dazu wird jede Blockzeile $\boldsymbol{B} = \boldsymbol{A}_\alpha$ der Designmatrix in $\mu$ Blockspalten

$$\boldsymbol{B} = \left(\boldsymbol{B}_\gamma\right) \quad \text{mit} \quad \boldsymbol{B}_\gamma \in \mathbb{R}^{b \times n_\gamma} \quad \text{mit} \quad n = \sum_{\gamma=1}^{\mu} n_\gamma \quad \text{und} \quad \gamma \in \{1, \ldots, \mu\} \tag{48}$$

aufgeteilt, woraufhin die Diagonalblöcke der Kitematrix nach (24) mit

$$\boldsymbol{K}_{\gamma\gamma} = \boldsymbol{B}_\gamma^T \boldsymbol{B}_\gamma \quad \text{mit} \quad \gamma \in \{1, \ldots, \mu\} \tag{49}$$

und die Subblöcke, deren Koordinatenpaare $(\gamma, \delta)$ in dem Vektor $\boldsymbol{\gamma\delta}$ gespeichert sind, mit

$$\boldsymbol{K}_{\gamma\delta} = \boldsymbol{B}_\gamma^T \boldsymbol{B}_\delta \quad \text{mit} \quad (\gamma, \delta) \in \boldsymbol{\gamma\delta} \tag{50}$$

berechnet werden.

### 4.3.2 Block–Cholesky–Zerlegung der Kitematrix

Für die Cholesky–Zerlegung der Kitematrix wird der Block–Cholesky–Algrithmus (Algorithmus 4.2) verwendet. Allerdings kann der Algorithmus aufgrund der Struktur der Kitematrix vereinfacht werden.

Die Vereinfachungen des Algorithmus von Cholesky können nur für alle Blöcke $\boldsymbol{K}_{\alpha\beta}$ mit $\alpha \in \{1, \ldots, \tau\}$ und $\beta \in \{1, \ldots, \eta\}$ diesseits der Reduktionsgrenze $\tau$ getroffen werden. Abbildung 31 zeigt, wie das Skalarprodukt der Spalte über dem zu reduzierenden Element und der Spalte über dem Diagonalelement der gleichen Zeile zu bilden ist. Die Definition der Kitematrix in Abbildung 23 beinhaltet, dass alle Subblöcke in einer eigenen Blockspalte liegen. Alle Blockspalten über einer zu reduzierenden Spalte sind daher orthogonal. Durch diese Orthogonalität entfällt der Updateschritt aus Zeile 5 in Algorithmus 4.2 für den Fall $\alpha \neq \beta$. Weiterhin entfällt der Updateschritt für alle Diagonalblöcke, über denen kein Subblock liegt. Dies sind alle Diagonalblöcke $\boldsymbol{K}_{\alpha\alpha}$ mit $\alpha \in \{1, \ldots, \tau\}$ bis zur Reduktionsgrenze $\tau$.

Zusammenfassend können zur Cholesky–Reduktion einer Kitematrix folgende Vereinfachungen getroffen werden; bis zur Reduktionsgrenze $\tau$ mit $\alpha \in \{1, \ldots, \tau\}$ und $\beta \in \{1, \ldots, \eta\}$ gilt:

- die drei Schritte der Cholesky–Reduktion beziehen sich nur auf die Blöcke $\boldsymbol{K}_{\alpha\alpha}$ (Cholesky), $\boldsymbol{K}_{\alpha\beta}$ (Update) und $\boldsymbol{K}_{\beta\beta}$ (Substitution) in dieser Reihenfolge,

- die Reduktion der drei Blöcke $\boldsymbol{K}_{\alpha\alpha}, \boldsymbol{K}_{\alpha\beta}, \boldsymbol{K}_{\beta\beta}$ ist unabhängig von der Reduktion der Blöcke $\boldsymbol{K}_{\gamma\gamma}$, $\boldsymbol{K}_{\gamma,\delta}$, $\boldsymbol{K}_{\delta\delta}$ mit $\gamma \in \{1, \ldots, \tau\}, \gamma \neq \alpha$ und $\delta \in \{1, \ldots, \eta\}, \eta \neq \beta$.

Algorithmus 4.4 ist durch Anbringen dieser Vereinfachungen aus dem Block-Cholesky–Algorithmus 4.2 entstanden. Da aufgrund der Definition der Kitematrix in Abschnitt 3.2 bis zur Reduktionsgrenze $\tau$ keine Subblöcke vorkommen dürfen, müssen in diesem Bereich keine Substitutions– und Updateschrittte durchgeführt werden. Die Cholesky–Schritte für die Diagonalblöcke $\boldsymbol{R}_{\alpha\alpha}$ mit $\alpha \in \{1, \ldots, \tau\}$ aus Zeile 8 von Algorithmus 4.2 können daher in eine eigene Schleife ausgelagert werden. Die Substitutions– und Updateschritte jenseits der Reduktionsgrenze $\tau$ werden anschließend ausgeführt, wobei jeder Blockindex $\alpha \in \{1, \ldots, \tau\}$ und $\beta \in \{\tau + 1, \ldots, \eta\}$ nur einmal vorkommt. Die verschachtelten Schleifen über $\alpha$ und $\beta$ aus den Zeilen 1 und 2 können daher in eine separate Schleife über den Vektor der Blockkoordinaten $\boldsymbol{\alpha\beta} = [(\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_s, \beta_s)]$ ersetzt werden. Der an die Kitematrix angepasste Algorithmus 4.4 besteht aus zwei separaten Schleifen und einer Cholesky–Zerlegung der vollen Matrix $\boldsymbol{D}$. In der ersten Schleife über $\alpha$ werden alle Diagonalblöcke bis zur Reduktionsgrenze nach Cholesky reduziert. In der zweiten Schleife über den Vektor $\boldsymbol{\alpha\beta}$ wird jeweils ein Subblock durch einen Substitutionsschritt berechnet und anschließend der entsprechende Diagonalblock unter diesem Subblock durch einen Updateschritt aktualisiert. Am Ende des Algorithmus wird der Teil der Kitematrix $(\boldsymbol{K}_{\alpha\beta})$ mit $\alpha > \tau$ und $\beta > \tau$, der in der Matrix $\boldsymbol{D}$ zusammenhängend gespeichert ist, in einem Schritt mit einer Cholesky–Zerlegung faktorisiert (siehe Abbildung 39).

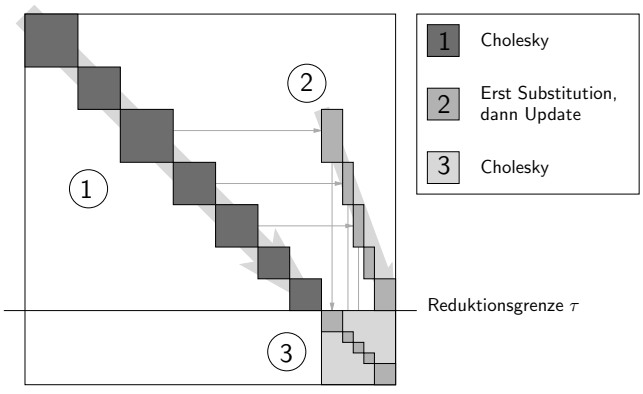



Abbildung 39: Illustration der Cholesky–Zerlegung der Kitematrix

**Algorithmus 4.4 (Kite–Cholesky)**

```
1  for  α = 1:τ
2       R_αα = cholesky( R_αα )          // Cholesky Schritt
3
4  for  (α,β) ∈ αβ
5  {
6       R_αα R_αβ^neu = R_αβ^alt         // Substitutionsschritt
7       R_ββ = R_ββ − R_αβ^T R_αβ         // Updateschritt in D
8  }
9
10 D = cholesky( D )
```

**Implementierung:** Wie bereits in Kapitel 3.4 erläutert ist die Kitematrix in zwei Vektoren von Objektklassen gespeichert, was mit der Implementierung von Algorithmus 4.4 harmoniert. Jede Objektklasse repräsentiert einen Block der Kitematrix. Im ersten Vektor, `block` genannt, werden alle Diagonalblöcke und im zweiten Vektor alle Nebendiagonalblöcke gespeichert. Die Objektklassen, `NeqSubBlock`, die die Subblöcke repräsentieren, besitzen einen Verweis auf die Objektklasse, die den Diagonalblock aus derselben Blockzeile enthält. Der Verweis dient dazu, den korrespondierenden Diagonalblock bei Durchlauf der Subblöcke im Substitutionsschritt in Zeile 6 von Algorithmus 4.4 zu adressieren. Bei der programmtechnischen Umsetzung des Algorithmus wird erst der Vektor `block` durchlaufen und für jede der Objektklassen `NeqBlock` bis auf die letzte die Methode `NeqBlock::chol` aufgerufen. Anschließend wird über den Vektor `subblock` iteriert und für jede der Objektklassen `NeqSubBlock` die Methoden `NeqSubBlock::backSubstitution` und `NeqSubBlock::update` aufgerufen. Zum Schluss wird die Methode `NeqBlock::chol` des letzten Blocks des Vektors `block` der Diagonalblöcke aufgerufen.

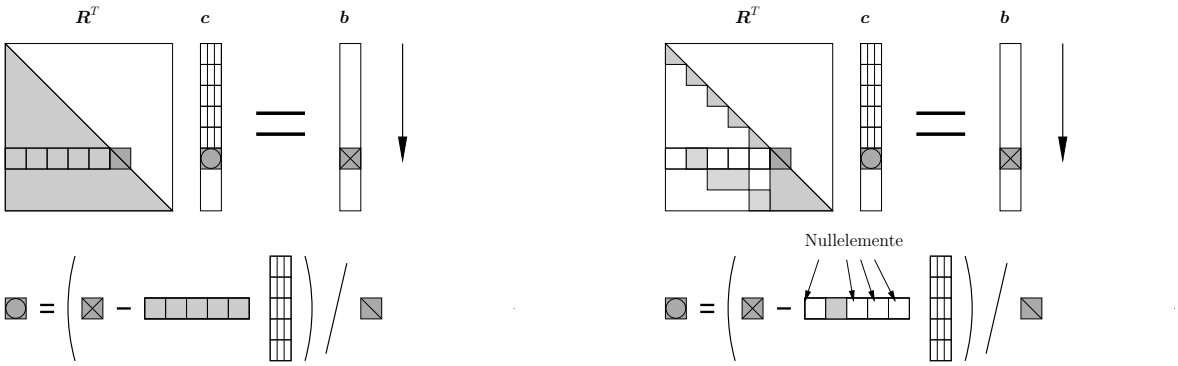
**Optimierung:** Die Objektklassen `NeqBlock` und `NeqSubBlock` führen jeweils die Operationen Cholesky–Zerlegung, Rückwartseinsetzen und Matrizenmultiplikation mit einzelnen Blöcken der Kitematrix aus. Dies wird intern mit den entsprechenden BLAS/LAPACK Routinen durchgeführt. Diese teilen die Blöcke intern wieder in kleinere Blöcke auf, so dass die verwendete Hardware optimal ausgenutzt wird. Die Hauptrechenarbeit wird daher mit hardwareoptimierten Bibliotheken durchgeführt, so dass die Kitematrix nicht nur mit einer Sparsetechnik gespeichert, sondern auch hochoptimiert berechnet wird. Die verwendeteten BLAS-Routinen sind `dtrsm` für das Rückwärtseinsetzen und `dgemm` bzw. `dsyrk` für die Matrizenmultiplikation. Die Cholesky–Zerlegung wird mit der LAPACK Routine `dpotrf` durchgeführt.

### 4.3.3 Cholesky–Verfahren mit der Kitematrix

Das Gleichungssystem

$$\boldsymbol{K}\boldsymbol{x} = \boldsymbol{b} \qquad \boldsymbol{x} \text{ unbekannt}, \quad \boldsymbol{K} \in \mathbb{R}^{n\times n}, \quad \boldsymbol{b} \in \mathbb{R}^{n\times 1}, \tag{51}$$

bei dem die Kitematrix die Systemmatrix ist, wird in `pcgma` mit dem Cholesky–Verfahren aus Abschnitt 4.1.4 gelöst. Dazu wird die Kitematrix zu Beginn des Programms einmal nach Cholesky zerlegt und gespeichert. Für die Kitematrix konnte die BLAS Routine `dtrsm` für das Vorwärts– und Rückwärtseinsetzen jedoch nicht eingesetzt werden, da es sich um eine schwach besetzte Matrix handelt. Stattdessen ist eine speziell an die Kitematrix angepasste Implementierung zum Einsatz gekommen, die nun erläutert wird.
Die zentrale Operation beim Vorwärtseinsetzen ist die Skalarmultiplikation zweier Vektoren (Summe in (35)). Um das $i$-te Element $c_i$ des unbekannten Vektors $\boldsymbol{c}$ zu berechnen, sind dies die Vektoren mit den Elementen $r_{i1}, \ldots, r_{i\,i-1}$ der $i$-ten Zeile von $\boldsymbol{R}^T$ und den bereits berechneten Elementen des Vektors $\boldsymbol{c}$ an den Stellen $1 \ldots, i-1$ (siehe Abbildung 40(a)). Durch die schwache Besetztheit der Kitematrix vereinfacht sich dieses Skalarprodukt. Der Vektor mit den Elementen $r_{i1}, \ldots, r_{i\,i-1}$ der $i$-ten Zeile von $\boldsymbol{R}^T$ hat nur sehr wenig Elemente, weil er nur an den Stellen Nichtnullelemente hat, die in der Kitematrix Teil eines Diagonalblocks oder eines Subblocks sind. Aufgrund der Definition der Kitematrix kann eine Zeile nur durch maximal zwei Blöcke, nämlich einen Diagonalblock und einen Subblock gehen (siehe Abschnitt 3.2). Dieser Zusammenhang wird in Abbildung 40 gezeigt. Dargestellt sind die verwendeten Elemente für die Berechnung des $i$-ten Elementes des unbekannten Vektors $\boldsymbol{c}$; für eine vollbesetzte untere Dreiecksmatrix $\boldsymbol{R}$ (Abbildung 40(a)) und für die Cholesky–Matrix einer Kitematrix (Abbildung 40(b)). Bei der schwach besetzen Cholesky–Matrix der Kitematrix müssen bei dem Skalarprodukt nicht alle Elemente berücksichtigt werden. Die Kitematrix wird im Computer mit dem speziell für




(a) Vollbesetze Matrix (b) Kitematrix

Abbildung 40: Benutzte Elemente beim Vorwärtseinsetzen

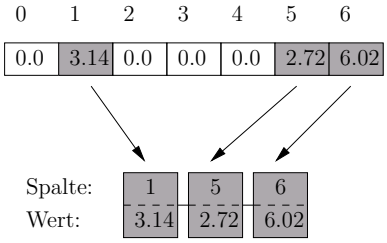



Abbildung 41: Speicherung einzelner Zeilen/Spalten der Kitematrix als Vektor von Nichtnullelementen

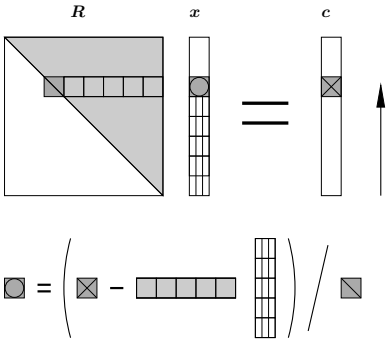



Abbildung 42: Benutzte Elemente beim Rückwärtseinsetzen

diese Matrix entwickelten Speicherschema (siehe Abschnitt 3.4) repräsentiert. Ein Zugriff auf eine vollständige Zeile einschließlich der Nullelemente ist nicht möglich.
Um das Vorwärts- und Rückwärtseinsetzen aus (35) und (36) mit der Kitematrix verwenden zu können, werden die Zeilen als Vektoren von Nichtnullelementen dargestellt. Ein Nichtnullelement einer Zeile der Kitematrix besteht aus dem Spaltenindex und dem dazugehörigen Wert an dieser Stelle (siehe Abbildung 41). Wir erhalten dadurch den Vektor $\boldsymbol{z}$ der Werte der Nichtnullelemente und den Vektor $\boldsymbol{j}$ der entsprechenden Spaltenindizes. Die Anzahl der Nichtnullelemente einer Zeile wird mit $q$ bezeichnet, daher gilt $\boldsymbol{z}, \boldsymbol{j} \in \mathbb{R}^{1\times q}$. Zu beachten ist, dass der Vektor $\boldsymbol{z}$ der oberen Diagonalmatrix $\boldsymbol{R}$ das Diagonalelement an erster Stelle hat (Abbildung 42). Beschreibt der Vektor $\boldsymbol{z}$ jedoch eine Zeile der unteren Diagonalmatrix $\boldsymbol{R}^T$, so ist das Diagonalelement an der letzten Stelle (Abbildung 40). Nun können (35) und (36) mit geringen Modifikationen mit der Kitematrix eingesetzt werden. Die Summen $\sum_{k=1}^{i-1}$ bzw. $\sum_{k=i+1}^{n}$ werden ersetzt durch die Summen $\sum_{k=1}^{q-1}$ bzw. $\sum_{k=2}^{q}$. Der Index der bereits berechneten Elemente $c_{j_k}$ wird dem Indexvektor $\boldsymbol{j}$ an der Stelle $k$ entnommen, wodurch ist eine zweifache Indizierung notwendig ist. Daher werden nur die Nichtnullelemente der Kitematrix benötigt. Das Vorwärts– und Rückwärtseinsetzen mit der Kitematrix ergibt sich daher zu

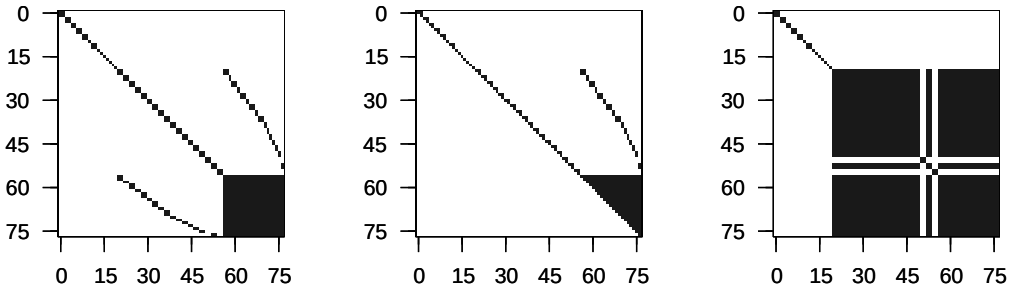
$$c_i = \left(\sum_{k=1}^{q-1} z_k\, c_{j_k}\right)/z_q \quad i \in \{1, n\} \qquad \text{Vorwärtseinsetzen} \tag{52}$$

$$x_i = \left(\sum_{k=2}^{q} z_k\, c_{j_k}\right)/z_1 \quad i \in \{n, 1\} \qquad \text{Rückwärtseinsetzen}. \tag{53}$$

**Implementierung:** Im Programm liefern die Funktion `getRow()` und `getCol()` der Klasse `Kite` den Vektor der Nichtnullelemente einer beliebigen Zeile oder Spalte der Kitematrix. Da im Speicher nur die Matrix $\boldsymbol{R}$ und nicht die Matrix $\boldsymbol{R}^T$ vorliegt, muss eine Zeile der Matrix $\boldsymbol{R}^T$ mit dem Befehl `getCol()` erzeugt werden. Die Spalte $i$ der Matrix $\boldsymbol{R}$ ist die Zeile $i$ der Matrix $\boldsymbol{R}^T$.

### 4.3.4 Unvollständige Inversion aus der nach Cholesky reduzierten Kitematrix

Die Entstehung von Füllelementen kann, wie schon erwähnt, nicht durch Umsortierung mit dem Freien Kite–Nummerierungsschema umgangen werden, der Teil der Kitematrix, der die Korrelationssektoren *Semi-Semi* und *Semi-Full* enthält, wird nach der Inversion voll besetzt sein (Abbildung 43). Der Algorithmus zur Inversion aus Abschnitt 4.1.5 kann jedoch an die Kitematrix angepasst werden, so dass nur die Elemente der Inversen Kitematrix berechnet werden, die innerhalb der Kitestruktur liegen, denn die ausserhalb der Kitestruktur entstehenden Füllelemente tragen, wie Hanson (1978) gezeigt hat, nicht zur Berechnung dieser Elemente bei. Die entstehende Matrix wird unvollständige Inverse genannt. Im Folgenden wird der an die Kitestruktur angepasste Algorithmus




(a) Kombinierte Kitematrix (b) Cholesky zerlegte Kitematrix (c) Invertierte Kitematrix

Abbildung 43: In der Kitestruktur entstehen bei der Inversion im Gegensatz zur Cholesky–Zerlegung Füllelemente

vorgestellt. Da nur die Elemente der inversen Kitematrix berechnet werden sollen, die auch in der Kitematrix ungleich Null sind, soll nun untersucht werden, welche Elemente dazu gebraucht werden. Der Korrelationssektor *Full-Full* der Kitematrix wird immer als volle Matrix gespeichert, da hier alle Parameter als korreliert betrachtet werden. Aus diesem Grund kann die rekursive Inversion in diesem ganzen Block ohne Beachtung von Füllelementen von unten nach oben durchgeführt werden. Am Übergang zu dem Korrelationssektor *Semi-Full* bzw.. *Semi-Semi*, soll die nachfolgende Betrachtung ansetzen. Es ist nun die erste Zeile der inversen Kitematrix über dem Korrelationssektor *Full-Full* zu berechnen. Abbildung 44 zeigt die Ausgangssituation. Die Matrix

wird in diesem Beispiel weiterhin mit $\boldsymbol{A}$, deren Inverse mit $\boldsymbol{B}$ und die Partitionen der aktuell zu berechnenden Teilinversen nach (38) mit $\boldsymbol{A}_{ij}$ bezeichnet. Dabei soll angenommen werden, dass drei Elemente des Blocks $\boldsymbol{A}_{12}$

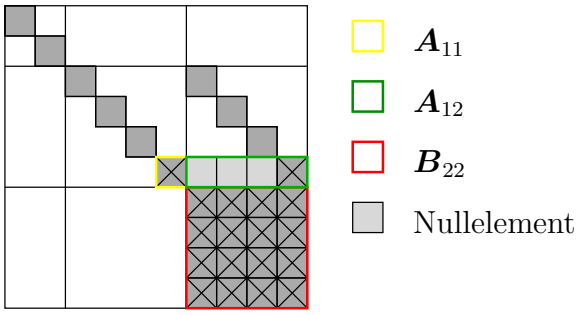



Abbildung 44: Ausgangssituation eines Rekursionsschrittes in der Struktur der Kitematrix

gleich Null sind. Es soll nun darauf geachtet werden, welche Elemente des Blocks $\boldsymbol{B}_{22}$ aufgrund des Auftretens dieser Nullelemente nicht benötigt werden, weil mit ihnen wirkungslose Multiplikationen berechnet werden. Wie man in Abbildung 45 sieht, besteht der Block $\boldsymbol{B}_{12}$ nach der Inversion nur noch aus Nichtnullelementen. Dabei

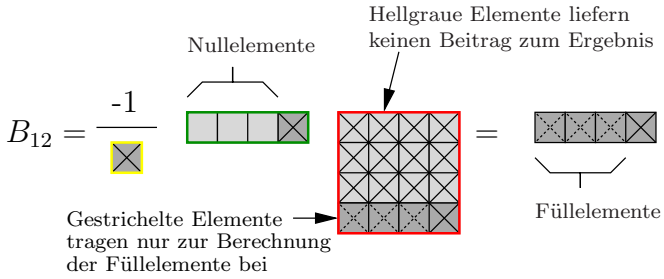



Abbildung 45: Berechnung des Blocks $\boldsymbol{B}_{12}$ innerhalb eines Schrittes der rekursiven Inversion

sind drei Füllelemente entstanden. Aufgrund der Nullelemente an den Positionen 1-3 in Block $\boldsymbol{A}_{12}$ tragen die Zeilen 1-3 des Blocks $\boldsymbol{B}_{22}$ nicht zum Ergebnis von Block $\boldsymbol{B}_{12}$ bei. Die nicht benötigten Elemente sind hellgrau hervorgehoben. Auf die Berechnung der gestrichelt eingezeichneten Füllelemente in Block $\boldsymbol{B}_{12}$ kann nur dann verzichtet werden, wenn sie auch zur Berechnung von Block $\boldsymbol{B}_{11}$ nicht beitragen (Abbildung 46). In der Be-

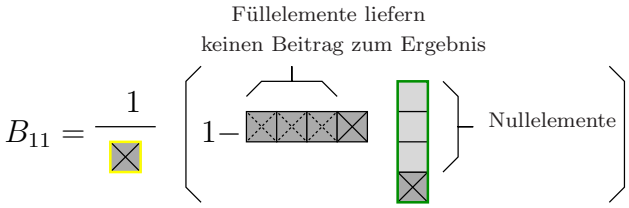



Abbildung 46: Berechnung des Blocks $\boldsymbol{B}_{11}$ innerhalb eines Schrittes der rekursiven Inversion

rechnung des Blocks $\boldsymbol{B}_{11}$ wird der neu berechnete Block $\boldsymbol{B}_{12}$ mit dem transponierten Block $\boldsymbol{A}_{12}$ multipliziert. Man sieht, dass die neu berechneten Füllelemente in diesem Schritt mit den ursprünglichen Nullelementen an der gleichen Stelle multipliziert werden und daher nichts zu dem Ergebnis von Block $\boldsymbol{B}_{11}$ beitragen. Auf die Berechnung der Füllelemente kann daher vollständig verzichtet werden. Dieser Zusammenhang wird durch die folgende Regel ausgedrückt.

**Regel 4.1 (Vernachlässigung von Füllelementen)** *Enthält eine Zeile bei der rekursiven Inversion durch Partitionierung Nullelemente außerhalb der Diagonalen, so kann auf die Berechnung der Füllelemente an diesen Stellen verzichtet werden, weil diese keinen Einfluß auf die Berechnung der anderen inversen Elemente haben. Liegt ein Nullelement in Spalte $j$ der aktuellen Zeile vor, so wird die Zeile und die Spalte $j$ des Blocks $\boldsymbol{B}_{22}$*

*nicht benötigt. Anders ausgedrückt heisst das, wenn in der aktuell berechneten Zeile nur ein Nichtnullelement an Position $j$ vorliegt, so wird zur Berechnung der ganzen inversen Zeile nur das Element der Zeile und Spalte $j$ aus $\boldsymbol{B}_{22}$ benötigt.* ■

Ersetzt man die Skalare durch Blöcke, so kann angegeben werden, welche Blöcke benötigt werden, um eine vollständige Blockzeile der unvollständigen inversen Kitematrix zu berechnen. Laut Definition der Struktur der Kitematrix (Abschnitt 3.2) gibt es in jeder Blockzeile der Kitematrix nur einen Diagonalblock und maximal einen Nebendiagonalblock. In Blockkoordinaten ausgedrückt bedeutet das, dass für die Berechnung der Blöcke $\boldsymbol{K}_{\alpha\alpha}^{(-1)}$ und $\boldsymbol{K}_{\alpha\beta}^{(-1)}$ der inversen Kitematrix nur der Block $\boldsymbol{K}_{\beta\beta}^{(-1)}$ aus dem Korrelationssektor *Full-Full* und die beiden Blöcke $\boldsymbol{R}_{\alpha\alpha}$ und $\boldsymbol{R}_{\alpha\beta}$ aus der reduzierten Kitematrix $\boldsymbol{R}$ benötigt werden. Alle anderen Blöcke sind unabhängig davon. Daher konnte der Algorithmus zur Berechnung der unvollständigen, inversen Kitematrix gut auf das neue Speicherschema, bei dem ein Vektor von Diagonalblöcken und ein Vektor von Nebendiagonalblöcken gespeichert wird, angewendet werden. In Algorithmus 4.5 gelten die Bezeichnungen für die Kitematrix, die in Abschnitt 3.2 eingeführt worden sind. Der Korrelationssektor *Full-Full* der Kitematrix wird in einer zusammenhängenden Matrix gespeichert und mit $\boldsymbol{D}$ bezeichnet. Sowohl dieser als auch die Diagonalblöcke, für die es keinen Nebendiagonalblock gibt, sind innerhalb der Inversion vollkommen unabhängig vom Rest der Kitematrix. Für sie kann daher die vollständige, rekursive Inversion separat berechnet werden. In Anlehnung an die gleichnamige LAPACK Routine wird dieser Schritt dpotri genannt. Der Vektor $\boldsymbol{\alpha}$ enthält alle Blockkoordinaten $\alpha$ dieser Blöcke. Der Vektor $\boldsymbol{\alpha\beta}$ enthält die Blockkoordinaten $\alpha_i, \beta_i$ aller Nebendiagonalblöcke.

**Algorithmus 4.5 (Unvollständige Inversion der Kitematrix)**

1
2 $\boldsymbol{K}_{D}^{(-1)} = \mathrm{dpotri}(\boldsymbol{D})$ // *Inverse aus der Choleskymatrix*
3
4 **for** $\alpha \in \boldsymbol{\alpha}$ // *Diagonalblöcke ohne Subblock*
5 {
6 $\boldsymbol{K}_{\alpha\alpha} = \mathrm{dpotri}(\ \boldsymbol{R}_{\alpha\alpha}\ )$
7 }
8
9 **for** $(\alpha, \beta) \in \boldsymbol{\alpha\beta}$ // *Diagonalblöcke mit Subblock*
10 {
11 $\boldsymbol{K}_{\alpha\beta}^{(-1)} = -\boldsymbol{R}_{\alpha\alpha}^{-1}\boldsymbol{R}_{\alpha\beta}\ \boldsymbol{K}_{\beta\beta}$
12 $\boldsymbol{K}_{\alpha\alpha}^{(-1)} = \boldsymbol{R}_{\alpha\alpha}^{-1}\ (\boldsymbol{R}_{\alpha\alpha}^{-1})^T - \boldsymbol{K}_{\alpha\beta}\ \boldsymbol{R}_{\alpha\beta}^{T}\ (\boldsymbol{R}_{\alpha\alpha}^{-1})^T$
13 }

**Implementierung:** Die Technik, die vollständige Kitematrix in den zwei Vektoren `block` und `subblock` zu speichern, eignet sich auch für diesen Algorithmus. Der Zugriff auf die Subblöcke und die korrespondierenden Diagonalblöcke ist realisierbar, indem der Vektor `subblock` durchlaufen und der in der Objektklasse `NeqSubBlock` enthaltene Verweis genutzt wird, um auf den entsprechenden Diagonalblock zuzugreifen. Die Koordinaten für den Zugriff auf den korrespondierenden Ausschnitt aus dem Block $\boldsymbol{D}$ ergeben sich aus den Startkoordinaten des aktuellen Subblocks, dessen Anzahl von Spalten und den Startkoordinaten des Blocks $\boldsymbol{D}$, der das letzte Element des Vektors `block` bildet.

**Optimierung:** Die Speicherung der einzelnen Blöcke als Matrizen erlaubt auch hier den Einsatz von hochoptimierbaren BLAS/LAPACK Routinen. So wird die Inversion des letzten Blocks und der alleinstehenden Diagonalblöcke $\boldsymbol{D}$ in einem Stück mit der LAPACK Routine `dpotri` durchgeführt. Diese Routine berechnet die Inverse einer Matrix mit Hilfe ihrer Cholesky–Matrix. Multiplikationen von Dreiecksmatrizen können mit BLAS Routinen `dtrmm` und von normalen Matrizen mit `dgemm` berechnet werden. Die Inverse einer Dreiecksmatrix ermittelt die LAPACK Routine `dtrtri` effizient, noch effizienter ist der Einsatz der BLAS Routine `dtrsv` für das Rückwärtseinsetzen anstelle der Multiplikation mit einer inversen Dreiecksmatrix bzw. das Vorwärtseinsetzen anstelle der Multiplikation mit ihrer Transponierten.

# 5 Lösungsverfahren

Bei der Kombination eines hochauflösenden, regelmäßigen Datensatzes mit einem niedrig auflösenden, nicht regelmässigen Datensatz nutzt das Freie Kite–Nummerierungsschema die schwache Besetztheit der Normalgleichungsmatrix des hochauflösenden Datensatzes, welche aus den in Kapitel 2.1 erwähnten Orthogonalitäten resultiert, aus. Diese Orthogonalitäten können ganz oder nur näherungsweise erfüllt sein, wodurch das Freie Kite–Nummerierungsschema bei verschiedenen Lösungsverfahren Anwendung findet.

## 5.1 Strenge Kombination von regelmäßigen und unregelmäßigen Daten

Sind die Orthogonalitäten aus Kapitel 2.1 erfüllt, so enthält die Kitematrix exakt dieselben Einträge, wie die Normalgleichungsmatrix, bis auf die durch das Freie Kite–Nummerierungschema geänderte Reihenfolge der Parameter. Zur Berechnung der strengen Kombination beider Datentypen kann daher die Normalgleichungsmatrix in (21) durch die Kitematrix ersetzt werden, woraus (51) erhalten wird, welches mit dem an die Kitematrix angepassten Cholesky–Verfahren aus Abschnitt 4.3.3 gelöst werden kann. Da beim Cholesky–Verfahren mit der Kitematrix keine Füllelemente entstehen, kann die sphärische harmonische Analyse wie in Kapitel 2.2 gezeigt, bis zu einem deutlich höheren maximalen Entwicklungsgrad durchgeführt werden. Zur Berechnung der Varianzen und Kovarianzen der unbekannten Parameter muss die inverse Kitematrix gebildet werden, wobei, wie in Abbildung 43(c) gezeigt auch bei der Kitematrix viele Füllelemente entstehen. Diese Füllelemente können jedoch mit Algorithmus 4.5 umgangen werden, wodurch die unvollständige Inverse erhalten wird. Aus dieser Matrix können die Varianzen und Kovarianzen der unbekannten Parameter abgeleitet werden.

## 5.2 Das Verfahren PCGMA

Sind die Orthogonalitäten aus Abschnitt 2.1 im Gegensatz zu der Annahme aus dem vorhergehenden Abschnitt nur näherungsweise erfüllt, so stimmen die Elemente der Kitematrix zwar mit denen der Normalgleichungsmatrix überein, bilden aber nur eine Teilmenge von ihnen. Durch das Austauschen der Kitematrix mit der Normalgleichungsmatrix werden vorhandene Korrelationen ignoriert und daher nur eine Näherungslösung von (21) erhalten. Die Kitematrix ist in diesem Fall jedoch eine Approximation der Normalgleichungsmatrix, weil sie, wie Abbildung 7 zu entnehmen ist, deren diagonaldominanten Teil enthält und nur kleinere Elemente vernachlässigt. Sie eignet sich daher als Vorkonditionierungsmatrix für das CG–Verfahren. Durch die Möglichkeit der Freien Kite–Nummerierung, die unbekannten Parameter in verschiedene Korrelationszonen einzuteilen, können zusätzliche Korrelationen bei der Aufstellung der Kitematrix berücksichtigt werden, wodurch die Approximation der Normalgleichungsmatrix und somit die Konvergenz des Verfahrens verbessert wird. Wie oben erwähnt, wurde das Verfahren PCGMA neu implementiert und durch das Freie Kite–Nummerierungsschema erweitert. Das resultierende parallele Programm `pcgma` läuft im produktiven Einsatz sowohl auf dem Cluster des Instituts für Theoretische Geodäsie, als auch auf dem Supercomputer JUMP des Forschungszentrums Jülich. Der Algorithmus der Konjugierten Gradienten und die Erweiterungen, die das Verfahren PCGMA beinhaltet werden im Folgenden kurz besprochen. Sie können ausführlicher in Schuh (1996) nachgeschlagen werden. Am Ende dieses Abschnitts wird der Algorithmus des Verfahrens PCGMA dargestellt und der nachfolgende Abschnitt wird die Erweiterung des Verfahrens durch das Freie Kite–Nummerierungsschema behandeln.

### 5.2.1 Methode der Konjugierten Gradienten

Schuh (1996) hat die Methode der konjugierten Gradienten (*conjugate gradients, CG*) für die Kombination von fast regelmäßigen Datensätzen mit nicht regelmäßigen Datensätzen niedriger Auflösung vorgeschlagen. Durch dieses Verfahren konnte der durch die Dimension der Normalgleichungsmatrix bedingte hohe Rechenaufwand und die aufwändige Speicherung in langsamen Massenspeichern, oder Netzwerken mit vielen Computern umgangen werden. Das CG–Verfahren ist ein schlankes System, dessen dominierende Kosten eine Matrix–Vektor Multiplikation pro Iteration ist. Den CG–Algorithmus zur Lösung eines Gleichungssystems $\boldsymbol{N}\boldsymbol{x} = \boldsymbol{n}$ zeigt Algorithmus 5.1.

**Algorithmus 5.1 (Methode der Konjugierten Gradienten)**

| | | |
|---|---|---|
| Eingabe | $\boldsymbol{N}$ | Normalgleichungsmatrix ($\boldsymbol{N} = \boldsymbol{A}^T\boldsymbol{A}$) |
| | $\boldsymbol{x}_0$ | Näherungslösung |
| | $\boldsymbol{n}$ | Rechte Seite ($\boldsymbol{n} = \boldsymbol{A}^T\boldsymbol{\ell}$) |
| Ausgabe | $\boldsymbol{r}^T\boldsymbol{r}$ | Quadratsumme der Residuen des Normalgleichungssystems |
| | $\boldsymbol{x}$ | Lösungsvektor |

Initialisierung

$$\boldsymbol{r}_{(0)} = \boldsymbol{N}\boldsymbol{x}_0 - \boldsymbol{n}$$
$$\boldsymbol{p}_{(0)} = -\boldsymbol{r}_{(0)}$$

Iterationsschritte $i = 0, 1, \ldots, I$

$$\left.\begin{aligned} e &= \frac{\boldsymbol{r}_{(i)}^T\boldsymbol{r}_{(i)}}{\boldsymbol{r}_{(i-1)}^T\boldsymbol{r}_{(i-1)}} \\ \boldsymbol{p}_{(i)} &= -\boldsymbol{r}_{(i)} + e\,\boldsymbol{p}_{(i-1)} \end{aligned}\right\} i > 0$$
$$\boldsymbol{h} = \boldsymbol{N}\boldsymbol{p}_{(i)}$$
$$q = \frac{\boldsymbol{r}_{(i)}^T\boldsymbol{r}_{(i)}}{\boldsymbol{p}_{(i)}^T\boldsymbol{h}}$$
$$\boldsymbol{x}_{(i+1)} = \boldsymbol{x}_{(i)} + q\,\boldsymbol{p}_{(i)}$$
$$\boldsymbol{r}_{(i+1)} = \boldsymbol{r}_{(i)} + q\,\boldsymbol{h}$$

### 5.2.2 Lösung eines Normalgleichungssystems

Wird CG dazu benutzt, ein Normalgleichungssystem zu lösen, so braucht die Normalgleichungsmatrix $\boldsymbol{N}$ nicht explizit aufgestellt zu werden, denn die Multiplikation eines Vektors mit der Normalgleichungsmatrix kann durch zwei Matrix–Vektor Multiplikationen mit der Designmatrix ersetzt werden. In Algorithmus 5.1 wird dazu die erste Zeile

$$\boldsymbol{r}_{(0)} = \boldsymbol{N}\boldsymbol{x}_0 - \boldsymbol{n}$$

geändert in

$$\begin{aligned} \boldsymbol{v}_{(0)} &= \boldsymbol{A}\boldsymbol{x}_\mathbf{0} - \boldsymbol{\ell} && \text{mit} \quad \boldsymbol{A} \in \mathbb{R}^{m\times n} \quad \text{und} \quad \boldsymbol{\ell} \in \mathbb{R}^{m\times 1} \\ \boldsymbol{r}_{(0)} &= \boldsymbol{A}^T\boldsymbol{v}_{(0)} && \text{mit} \quad \boldsymbol{r} \in \mathbb{R}^{n\times 1} \end{aligned} \tag{54}$$

und die fünfte Zeile

$$\boldsymbol{h} = \boldsymbol{N}\boldsymbol{p}_{(i)}$$

geändert in

$$\begin{aligned} \boldsymbol{g} &= \boldsymbol{A}\boldsymbol{p}_{(i)} \\ \boldsymbol{h} &= \boldsymbol{A}^T\boldsymbol{g}\,. \end{aligned} \tag{55}$$

Dadurch wird $\boldsymbol{N}$ nicht mehr für den CG–Algorithmus benötigt und die Anzahl der notwendigen Operationen reduziert. Der CG–Algorithmus ist daher mit $mn$ Additionen und $mn$ Multiplikationen deutlich schneller als das Aufstellen der Normalgleichungsmatrix, welches $mn^2$ Additionen und $mn^2$ Multiplikationen erfordert. Bei der GOCE Mission wird $m$ ungefähr 46 Millionen (6 Monate Beobachtungszeitraum) und $n$ ungefähr 58 000 sein (Grad 240). Die Aufstellung der Normalgleichungsmatrix ist in diesem Fall 58 000 mal aufwändiger, d. h. wenn die Multiplikation $\boldsymbol{A}\boldsymbol{x}$ beispielsweise 10 Minuten dauert, dann würden bei gleicher Rechengeschwindigkeit für $\boldsymbol{A}^T\boldsymbol{A}$ ca. 400 Tage vergehen.

### 5.2.3 Lösung großer Systeme

Bei großen Ausgleichungsproblemen kann die Designmatrix oft nur mit hohem Aufwand gespeichert werden, so dass es schneller ist, sie in jedem Iterationsschritt neu aufzustellen. Die Kosten für eine Iteration werden in diesem Fall nicht mehr von zwei Matrix–Vektor Multiplikationen dominiert, sondern von der Zeit für die Aufstellung der Designmatrix. Die Multiplikation mit $\boldsymbol{A}$ erfordert zeilenweisen Zugriff und die Multiplikation mit $\boldsymbol{A}^T$ erfordert spaltenweisen Zugriff, so dass die Designmatrix zweimal pro Iteration berechnet werden muss. Durch Umstellung der Berechnungsreihenfolge auf blockzeilenweisen Zugriff auf die Designmatrix entsprechend (26) und (27) ist es jedoch möglich, sie nur einmal pro Iteration aufstellen zu müssen. Dazu werden die Matrix $\boldsymbol{A}$ und der Vektor $\boldsymbol{g}$ aus (55) entlang der Dimension $m$ in $S$ Blöcke von Zeilen aufgeteilt:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_1 \\ \boldsymbol{A}_2 \\ \dots \\ \boldsymbol{A}_S \end{bmatrix} \qquad \boldsymbol{g} = \begin{bmatrix} \boldsymbol{g}_1 \\ \boldsymbol{g}_2 \\ \dots \\ \boldsymbol{g}_S \end{bmatrix} \tag{56}$$

Die einzelnen Blöcke $\boldsymbol{g}_s$ ergeben sich nun mit Hilfe des Blocks $\boldsymbol{A}_s$ und des Vektors $\boldsymbol{p}_{(i)}$ zu

$$\boldsymbol{g}_s = \boldsymbol{A}_s \boldsymbol{p}_{(i)} \,. \tag{57}$$

Durch diese Einteilung ist $\boldsymbol{h}$ als Summe darstellbar, wobei für die Berechnung jedes einzelnen Summanden jeweils nur der Block $s$ von $\boldsymbol{A}$ und $\boldsymbol{g}$ notwendig ist. In (58) ist daher nur zeilenweiser Zugriff auf die Designmatrix notwendig.

$$\boldsymbol{h} = \sum_{s=1}^{S} \boldsymbol{A}_s^T \left( \boldsymbol{A}_s \boldsymbol{p}_{(i)} \right) \tag{58}$$

Analog dazu wird aus (54)

$$\boldsymbol{r}_{(0)} = \sum_{s=1}^{S} \boldsymbol{A}_s^T \left( \boldsymbol{A}_s \boldsymbol{x}_0 - \boldsymbol{\ell}_s \right) \,. \tag{59}$$

Diese Technik der Aufteilung der Designmatrix und aller anderen Vektoren des CG–Algorithmus, die dem $\mathbb{R}^{m \times 1}$ angehören, kann sowohl dazu benutzt werden, Designmatrizen zu verarbeiten, die am Stück nicht speicherbar sind, als auch zur Parallelisierung. Bei der Parallelisierung werden die $S$ Blöcke der Designmatrix auf verschiedene Computer verteilt und die Ergebnisvektoren aus (54) und (55) auf einem Computer gesammelt und addiert.

### 5.2.4 Vorkonditionierung

Vorkonditionierung ist eine Technik zur Verbesserung der Konditionszahl $\kappa$ der Systemmatrix. Die Konditionszahl

$$\kappa(\boldsymbol{A}) = \frac{\lambda_{\max}}{\lambda_{\min}} \tag{60}$$

einer symmetrischen, positiv definiten Matrix $\boldsymbol{A}$ ist definiert als das Verhältnis zwischen dem größten und dem kleinsten Eigenwert. Je größer die Konditionszahl der Normalgleichungsmatrix ist desto schlechter konvergiert die CG–Methode. Dies wird durch den Spektralradius

$$\rho_{\mathrm{CG}} = \left( \frac{1 - \sqrt{\kappa(\boldsymbol{A})}}{1 + \sqrt{\kappa(\boldsymbol{A})}} \right)^2 \tag{61}$$

der Methode der Konjugierten Gradienten ausgedrückt. Im Verfahren PCGMA ist die Systemmatrix die Normalgleichungsmatrix $\boldsymbol{N}$. Das überbestimmte Gleichungssystem

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{\ell} + \boldsymbol{v} \tag{62}$$

nach der Methode der kleinsten Quadrate mit dem CG–Verfahren zu lösen hat den Nachteil, dass die Konditionszahl der Normalgleichungsmatrix

$$\kappa(\boldsymbol{N}) = \kappa(\boldsymbol{A})^2$$

das Quadrat der Konditionszahl der Designmatrix ist, wodurch ist eine schlechte Konvergenz zu erwarten ist. Untersuchungen des numerischen Verhaltens der Kombination von SST und SGG Daten haben dies bestätigt. Im PCGMA Verfahren wird daher immer mit Vorkonditionierung gearbeitet. Die Kite Struktur, die sich durch die strenge Kombination von regelmäßigen und nicht regelmäßigen Daten ergibt, hat sich als sehr gute Vorkonditionierungsmatrix erwiesen. Die Vorkonditionierung basiert auf der Annahme, dass die hochauflösenden SGG Daten näherungsweise regelmäßig sind, so dass durch Vernachlässigung der Korrelationen zwischen den Koeffizienten ungleicher Ordnung eine blockdiagonale Struktur in der SGG Normalgleichungsmatrix entsteht. Durch Kombination dieser mit der kleineren, aber vollbesetzten SST Normalgleichungsmatrix entsteht die Kitestruktur entsprechend Abbildung 18. Wie in Kapitel 3 gezeigt, kann die Kitematrix effizient gespeichert werden und alle notwendigen Operationen für die Vorkonditionierung mit ihr durchgeführt werden. Dazu wird eine repräsentative Matrix $\boldsymbol{K}$ gesucht, die der Normalgleichungsmatrix möglichst ähnlich ist, so dass die Konditionszahl $\kappa(\boldsymbol{K}^{-1}\boldsymbol{N})$ kleiner als $\kappa(\boldsymbol{N})$ ist, wobei $\boldsymbol{K}$ leicht zu speichern und zu berechnen sein muss. Die Eigenschaften treffen auf die Kitematrix zu. Dazu wird das zu lösende Gleichungssystem in ein alternatives Gleichungssystem transformiert, dessen Systemmatrix eine kleinere Konditionszahl hat.

$$\boldsymbol{N}\boldsymbol{x} = \boldsymbol{n} \quad \text{wird ersetzt durch} \quad \boldsymbol{K}^{-1}\boldsymbol{N}\boldsymbol{x} = \boldsymbol{K}^{-1}\boldsymbol{n}\,. \tag{63}$$

Die Herleitung der Formeln für die Vorkonditionierung kann in Schuh (1996) nachgelesen werden. Zur Durchführung der Vorkonditionierung wird in Algorithmus 5.1 der Residuenvektor $\boldsymbol{\rho}$ des transformierten Gleichungssystems aus (63) eingeführt, der sich aus der Multiplikation des Residuenvektors mit der inversen Kitematrix zu

$$\boldsymbol{\rho} = \boldsymbol{K}^{-1}\boldsymbol{r} \tag{64}$$

ergeben würde. Die Multiplikation mit der inversen Kitematrix wird nicht direkt ausgeführt, sondern durch Vorwärts– und Rückwärtseinsetzen mit der nach Cholesky zerlegten Kitematrix und dem Residuenvektor als rechte Seite berechnet. Die Quadratsumme der Residuen $\boldsymbol{r}^T\boldsymbol{\rho}$ wird dann durch das Skalarprodukt aus dem Residuenvektor mit dem transformierten Residuenvektor gebildet. Die frühere Relaxationsrichtung $\boldsymbol{p}$ wird ersetzt durch die neue Relaxationsrichtung

$$\boldsymbol{\Pi} = \boldsymbol{K}^{-1}\boldsymbol{p}\,. \tag{65}$$

Da die Relaxationsrichtung aber auch direkt im transformierten System durch

$$\boldsymbol{\Pi}_{(i)} = -\boldsymbol{\rho}_{(i)} + e\,\boldsymbol{\Pi}_{(i-1)} \tag{66}$$

berechnet werden kann (siehe Algorithmus 5.2), ist das Vorwärts– und Rückwärtseinsetzen mit der Kitematrix nur einmal pro Iteration notwendig.

### 5.2.5 Dekorrelation der Beobachtungen durch digitale Filterung

Aufgrund der Charakteristik des Messprozesses werden die Beobachtungsdaten der Satellitengradiometrie hoch korreliert sein. Die unbekannten Parameter können daher nicht in dem linearen Modell

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{\ell} + \boldsymbol{v} \quad \text{mit} \quad \Sigma\{\boldsymbol{\ell}\} = \boldsymbol{\sigma}^2\boldsymbol{I} \tag{67}$$

geschätzt werden, sondern in dem Modell

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{\ell} + \boldsymbol{v} \quad \text{mit} \quad \Sigma\{\boldsymbol{\ell}\} = \boldsymbol{\sigma}^2\boldsymbol{P}^{-1}\,. \tag{68}$$

Daher ergeben sich die Normalgleichungen zu

$$\boldsymbol{A}^T\boldsymbol{P}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{P}\boldsymbol{\ell}\,. \tag{69}$$

Vom mathematischen Standpunkt wäre die Gewichtung der Beobachtungen leicht in den CG–Algorithmus integrierbar. Dazu wird die Gewichtsmatrix zerlegt in die zwei Dreiecksmatrizen

$$\boldsymbol{P} = \boldsymbol{G}\boldsymbol{G}^T\,. \tag{70}$$

Mit

$$\boldsymbol{G}^T\boldsymbol{A} = \bar{\boldsymbol{A}} \quad \text{und} \quad \boldsymbol{G}^T\boldsymbol{\ell} = \bar{\boldsymbol{\ell}} \tag{71}$$

wird dann das Modell mit unkorrelierten Beobachtungen

$$\bar{\boldsymbol{A}}\boldsymbol{x} = \bar{\boldsymbol{\ell}} + \bar{\boldsymbol{v}} \quad \text{mit} \quad \Sigma\{\boldsymbol{\ell}\} = \boldsymbol{\sigma}^2\boldsymbol{I} \tag{72}$$

erhalten, siehe z. B. Koch (1999, S. 154). Innerhalb des CG–Algorithmus 5.1 würde somit aus (55)

$$\begin{aligned} \boldsymbol{g} &= \boldsymbol{G}^T\boldsymbol{A}\boldsymbol{p}_{(i)} = \bar{\boldsymbol{A}}\boldsymbol{p} \\ \boldsymbol{h} &= \boldsymbol{A}^T\boldsymbol{G}\boldsymbol{g} \quad = \bar{\boldsymbol{A}}^T\boldsymbol{g}\,. \end{aligned} \tag{73}$$

Die Dimension der Gewichtsmatrix ist jedoch $m \times m$, wobei $m$ die Anzahl der Messungen ist. Bei $m = 18$ Millionen wäre die Gewichtsmatrix mit 2000 Terabyte nicht speicherbar. Ein weiterer Nachteil dieses Verfahrens ist, dass für die Berechnung der ersten Zeile der dekorrelierten Designmatrix $\bar{\boldsymbol{A}}$ die gesamte Designmatrix $\boldsymbol{A}$ benötigt wird, was mit der in Abschnitt 5.2.3 vorgestellten Technik zur Verarbeitung großer Systeme nicht vereinbar ist. Innerhalb des PCGMA Verfahrens wird die Dekorrelation der Beobachtungen daher durch einen ARMA Filter dargestellt. Die Berechnung der Designmatrix $\bar{\boldsymbol{A}}$ zu den dekorrelierten Beobachtungen $\bar{\boldsymbol{\ell}}$ stellt sich dann als ein Produkt mit der Filtermatrix $\boldsymbol{F}$ dar.

$$\bar{\boldsymbol{A}} = \boldsymbol{F}\boldsymbol{A} \qquad\qquad \bar{\boldsymbol{\ell}} = \boldsymbol{F}\boldsymbol{\ell} \tag{74}$$

Die Filterung kann so implementiert werden, dass nur zeilenweiser Zugriff auf die Designmatrix notwendig ist (Schuh 1996, Schuh 2003). Die Filterung der Designmatrix ist so blockweise durchführbar

$$\bar{\boldsymbol{A}}_s = \text{filter}(\boldsymbol{A}_s) \quad \bar{\boldsymbol{\ell}}_s = \text{filter}(\boldsymbol{\ell}_s) \tag{75}$$

und daher mit der Technik zur Verarbeitung großer Systeme aus Abschnitt 5.2.3 vereinbar. Durch die Endlichkeit der Messreihe wird die regelmässige Struktur der Korrelationsmatrix durch Randeffekte gestört. Die strenge Berücksichtigung dieser Randeffekte erfordert entweder die Lösung von sehr großen Toeplitz–Systemen oder eine Approximation mit zirkulierenden Systemen. Wird diese Approximation angenommen, so hat dies zur Folge, dass die Randelemente nicht korrekt berechnet werden, da bei ARMA–Filtern auch über die Startelemente verfügt werden muss, und dies weitgehend willkürlich geschieht, führen wir eine Aufwärmphase am Anfang der Beobachtungen ein, damit diese unkorrekten Korrelationen nicht das System stören. Der Verzicht auf diese Messungen ist aufgrund der großen Überbestimmung bei der Satellitengradiometrie unkritisch, verlangt jedoch Beachtung bei der Parallelverarbeitung, weil jeder Prozessor seine eigene Aufwärmphase benötigt und die Beobachtungen daher überlappend verteilt werden müssen. Untersuchungen über den Einfluss des Aufwärmverhaltens bei der Parallelverarbeitung von SGG Daten sind von Plank (2004) angestellt worden. Ist die Überlappung ausreichend lang, so sind keine Unterschiede mehr zwischen der Parallelverarbeitung und der seriellen Verarbeitung zu erkennen.
In Algorithmus 5.2 wird die Filterung nicht explizit erwähnt, weil die Aufstellung der Designmatrix und ihre Filterung in einem kombinierten Verarbeitungsschritt erfolgen weshalb keine Änderung am Algorithmus notwendig ist. Die Zeit für die Aufstellung der Designmatrix setzt sich zusammen aus den Schritten

- Berechnung der zweiten Ableitungen des Potentials,
- Transformation der Designmatrix in ein anderes Bezugssystem,
- und Filterung,

welche ungefähr gleichlang dauern.

### 5.2.6 Kombination von Normalgleichungen und Beobachtungsgleichungen

In Kapitel 2.3 wurde die gemeinsame Ausgleichung von regelmäßigen und unregelmäßigen Daten vorgestellt. Eine typische Anwendung des PCGMA Verfahrens ist die gemeinsame Ausgleichung von SGG und SST Daten. Zur Aufstellung des Vorkonditionierers wird die Annahme der Regelmäßigkeit für die SGG Daten getroffen. Dies gilt jedoch nur für die Vorkonditionierung. PCGMA berechnet die strenge Kombination der voll korrelierten SGG Daten mit den SST Daten. Die SST Daten sollen dabei als Normalgleichungssystem vorliegen. Die SGG Normalgleichungsmatrix ist aufgrund ihrer Größe nicht berechenbar und erscheint daher im PCGMA Algorithmus (Alg. 5.2) nur indirekt. Die Kombination der beiden Beobachtungstypen wird auf die Addition des

Residuenvektors der Normalgleichungssysteme zurückgeführt. Die Residuenvektoren $\boldsymbol{r}_{\text{sgg}}$ und $\boldsymbol{r}_{\text{sst}}$ ergeben sich im Initialisierungsschritt auf unterschiedliche Weise

$$\boldsymbol{r}_{\text{sgg}} = \sum_{s=1}^{S} \left[ \boldsymbol{A}_s \left( \boldsymbol{A}_s \boldsymbol{x} - \boldsymbol{\ell}_{\text{sgg}} \right) \right] \qquad \boldsymbol{r}_{\text{sst}} = \boldsymbol{N}_{\text{sst}} \boldsymbol{x} - \boldsymbol{n}_{\text{sst}} \,. \tag{76}$$

Die Gewichtung zwischen den Datentypen wird über den Gewichtungsparameter $\alpha$ realisiert

$$\boldsymbol{r} = \boldsymbol{r}_{\text{sgg}} + \alpha \boldsymbol{r}_{\text{sst}} \,. \tag{77}$$

Zu beachten ist jedoch die unterschiedliche Dimension der Systeme. Da die Satellitengradiometrie die hohen Frequenzen des Schwerefeldes besser *sehen* kann, ist geplant, die SGG Daten bis Grad 240 aufzulösen. Das Satellite-to-Satellite Verfahren eignet sich gut zur Bestimmung der niedrigeren Grade, daher werden die SST Daten bis maximal Grad 100-150 aufgelöst. Es gilt daher $n_{\text{sgg}} \gg n_{\text{sst}}$. Durch die Umnummerierung der unbekannten Parameter in das Freie Kite–Nummerierungsschema befinden sich die Koeffizienten, die von beiden Beobachtungstypen geschätzt werden, immer in der Korrelationszone *Full*, die am Ende des Parametervektors angeordnet ist (siehe Kapitel 2.3.2). Der SGG Residuenvektor kann daher in die Korrelationszonen *Independent*, *Semi* und *Full* aufgeteilt werden. Der SST Residuenvektor hat nur Elemente in der Korrelationszone *Full*.

$$\boldsymbol{r}_{\text{sgg}} = \begin{bmatrix} \boldsymbol{r}_{\text{independent}} \\ \boldsymbol{r}_{\text{semi}} \\ \boldsymbol{r}_{\text{full}} \end{bmatrix} \qquad \boldsymbol{r}_{\text{sst}} = \begin{bmatrix} - \\ - \\ \boldsymbol{r}_{\text{full}} \end{bmatrix} \tag{78}$$

Zur gemeinsamen Ausgleichung wird daher (77) unter Berücksichtigung von (78) durch Erweiterung von (59) in den CG–Algorithmus integriert.

$$\boldsymbol{r}_{(0)} = \sum_{s=1}^{S} \Big( \boldsymbol{A}_s^T ( \boldsymbol{A}_s \left| \begin{matrix} \boldsymbol{x}_{\text{independent}} \\ \boldsymbol{x}_{\text{semi}} \\ \boldsymbol{x}_{\text{full}} \end{matrix} \right| - \boldsymbol{\ell}_s ) \Big) + \alpha \ \left( \boldsymbol{N}_{\text{sst}} \, \boldsymbol{x}_{\text{full}} - \boldsymbol{n}_{\text{sst}} \right) \tag{79}$$

Analog dazu wird (58) erweitert zu

$$\boldsymbol{h} = \sum_{s=1}^{S} \Big( \boldsymbol{A}_s^T ( \boldsymbol{A}_s \left| \begin{matrix} \boldsymbol{p}_{\text{independent}} \\ \boldsymbol{p}_{\text{semi}} \\ \boldsymbol{p}_{\text{full}} \end{matrix} \right| - \boldsymbol{\ell}_s ) \Big) + \alpha \, \boldsymbol{N}_{\text{sst}} \, \boldsymbol{p}_{\text{full}} \,. \tag{80}$$

Der vollständige Algorithmus PCGMA mit den in den Abschnitten 5.2.1 bis 5.2.6 dargestellten Erweiterungen ist in Algorithmus 5.2 gezeigt.

**Algorithmus 5.2 (PCGMA)**

| | | |
|---|---|---|
| Eingabe | $\boldsymbol{A}_s$ | $s \in 1, \ldots, S$ Designmatrix mit ausschließlich zeilenweisem Zugriff |
| | $\ell$ | Beobachtungen |
| | $\boldsymbol{N}_{\text{sst}}$ | Normalgleichungsmatrix (z. B. SST) |
| | $\boldsymbol{n}_{\text{sst}}$ | Rechte Seite |
| | $\boldsymbol{x}_0$ | Näherungslösung |
| | $\alpha$ | Gewichtungsfaktor zwischen den Normalgleichungen |
| | $I$ | Anzahl Iterationen |
| Ausgabe | $\boldsymbol{x}$ | Lösungsvektor |
| | $\boldsymbol{r}^T\rho$ | Quadratsumme der Residuen |
| | $\boldsymbol{v}^T\boldsymbol{v}$ | Quadratsumme der Verbesserungen der Beobachtungen |

Initialisierung

$$\boldsymbol{K} = \text{buildKitePreconditioner}(\boldsymbol{\ell})$$

$$\boldsymbol{r}_{(0)} = \sum_{s=1}^{S}\Big(\boldsymbol{A}_s^T \underbrace{(\boldsymbol{A}_s \left|\begin{matrix}\boldsymbol{x}_{\text{independent}}\\ \boldsymbol{x}_{\text{semi}}\\ \boldsymbol{x}_{\text{full}}\end{matrix}\right| - \boldsymbol{\ell}_s)}_{\boldsymbol{v}_s^{(0)}}\Big) + \alpha\ (\boldsymbol{N}_{\text{sst}}\,\boldsymbol{x}_{\text{full}} - \boldsymbol{n}_{\text{sst}})$$

$$\boldsymbol{v}_{(0)} = \left[\boldsymbol{v}_1^T, \boldsymbol{v}_2^T, \ldots, \boldsymbol{v}_S^T\right]^T$$

$$\boldsymbol{\rho}_{(0)} = \text{solve}(\boldsymbol{K}, \boldsymbol{r}_{(0)})$$

$$\boldsymbol{\Pi}_{(0)} = -\boldsymbol{\rho}_{(0)}$$

Iterationsschritte $i = 0, 1, \ldots, I$

$$\left.\begin{aligned} e &= \frac{\boldsymbol{r}_{(i)}^T\boldsymbol{\rho}_{(i)}}{\boldsymbol{r}_{(i-1)}^T\boldsymbol{\rho}_{(i-1)}} \\ \boldsymbol{\Pi}_{(i)} &= -\boldsymbol{\rho}_{(i)} + e\,\boldsymbol{\Pi}_{(i-1)} \end{aligned}\right\} i > 0$$

$$\boldsymbol{h} = \sum_{s=1}^{S}\Big(\boldsymbol{A}_s^T \underbrace{(\boldsymbol{A}_s \left|\begin{matrix}\boldsymbol{\Pi}_{\text{independent}}\\ \boldsymbol{\Pi}_{\text{semi}}\\ \boldsymbol{\Pi}_{\text{full}}\end{matrix}\right|)}_{\boldsymbol{g}_s}\Big) + \alpha\,\boldsymbol{N}_{\text{sst}}\,\boldsymbol{\Pi}_{\text{full}}$$

$$\boldsymbol{g}_{(i)} = \left(\boldsymbol{g}_1^T, \boldsymbol{g}_2^T, \ldots, \boldsymbol{g}_S^T\right)^T$$

$$q = \frac{\boldsymbol{r}_{(i)}^T\boldsymbol{\rho}_{(i)}}{\boldsymbol{\Pi}_{(i)}^T\boldsymbol{h}}$$

$$\boldsymbol{x}_{(i+1)} = \boldsymbol{x}_{(i)} + q\,\boldsymbol{\Pi}_{(i)}$$

$$\boldsymbol{r}_{(i+1)} = \boldsymbol{r}_{(i)} + q\,\boldsymbol{h}$$

$$\boldsymbol{\rho}_{(i+1)} = \text{solve}(\boldsymbol{K}, \boldsymbol{r}_{(i+1)})$$

$$\boldsymbol{v}_{(i+1)} = \boldsymbol{v}_{(i)} + q\,\boldsymbol{g}_{(i)}$$

## 5.3 Erweiterung des Verfahrens PCGMA durch FKN

Mit dem Freien Kite–Nummerierungsschema ist die Möglichkeit geschaffen worden, die unbekannten Parameter bei der sphärischen harmonischen Analyse so umzusortieren, dass die blockdiagonale Struktur der Normalgleichungsmatrix zu einem regelmäßigen, hoch auflösenden Datensatz an eine vollbesetzte, niedriger auflösende Normalgleichungsmatrix beliebigen Typs angepasst werden kann. Dabei entsteht die Kitematrix mit vier Korrelationssektoren (Abbildung 16 auf Seite 27). Die Struktur der Kitematrix sieht vor, dass der Korrelationssektor *Full-Full* vollbesetzt ist, ohne dass bei der Cholesky–Zerlegung Füllelemente im sonstigen Bereich entstehen. Die bisherige Strategie zur Erstellung der Kitematrix war zunächst, den blockdiagonalen Beitrag der regelmäßigen Daten aufzustellen, welcher dann die in Abbildung 47 gezeigte Struktur besaß. Der Korrelationssektor *Full-Full* beinhaltet zwar nur eine blockdiagonale Struktur, aber die Algorithmen der Kitematrix behandeln diesen wie

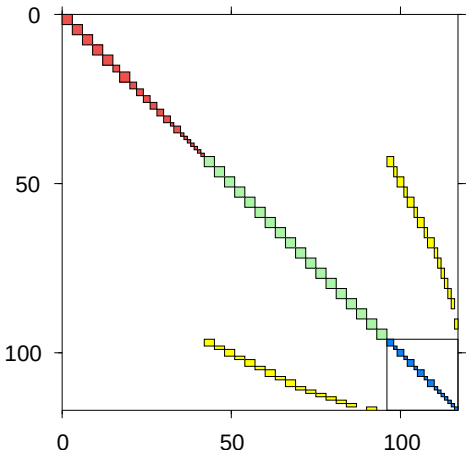


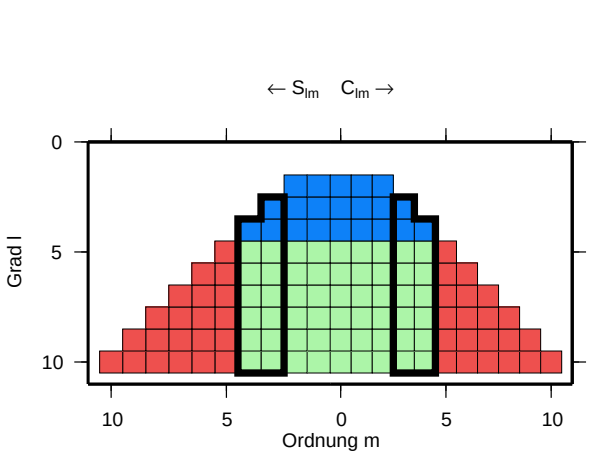

Abbildung 47: Anpassung eines hochauflösenden Modells an ein niedriger auflösendes Modell: Gemeinsame Koeffizienten werden am Schluss angeordnet und liegen im Korrelationssektor *Full-Full*

eine vollbesetzte Matrix, so dass eine vollbesetzte Normalgleichungsmatrix beliebigen Typs zu diesem Block addiert werden kann, vorausgesetzt, die Reihenfolge der Parameter stimmt überein. Diese Voraussetzung wird durch FKN erfüllt.

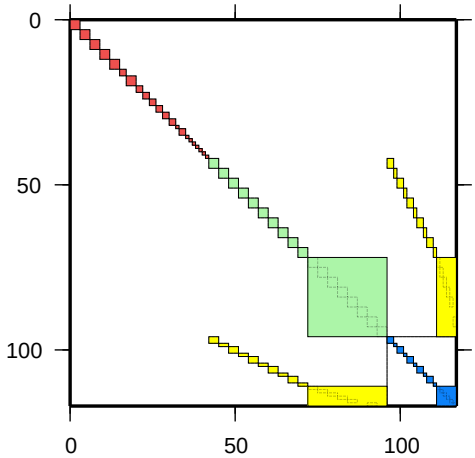
### 5.3.1 Berücksichtigung von zusätzlichen Korrelationen

Die Vorkonditionierung mit der Kitematrix in PCGMA beruht auf der Annahme, dass die Beobachtungen der Satellitengradiometrie näherungsweise einen regelmäßigen Datensatz und daher eine näherungsweise blockdiagonale Normalgleichungsmatrix bilden. Zur Aufstellung der Kitematrix werden die Korrelationen außerhalb der Blockdiagonalstruktur vernachlässigt. Ein neuer Denkansatz ist nun, nicht alle Korrelationen zu vernachlässigen, sondern auch bestimmte Bereiche ausserhalb der Blockdiagonalstruktur in die Kitematrix aufzunehmen, um die Normalgleichungsmatrix, wie später erläutert wird, besser zu approximieren. Berücksichtigung von Korrelationen zwischen benachbarten Ordnungen wäre mit der Kitestruktur verträglich, weil die Parameter ordnungsweise sortiert sind und so nur größere Diagonalblöcke entstünden.

Abbildung 48 zeigt dies am Beispiel der Kombination von voll korrelierten Daten bis Grad und Ordnung 4 (blau dargestellt) mit regelmässigen Daten bis Grad und Ordnung 10 (rot dargestellt). Die in Abbildung 48(a) schwarz umrandeten Parameter der Ordnungen 3 und 4 sollen zusätzlich als voll korreliert angesehen angesehen werden. Die daraus resultierenden Änderungen in der Kitematrix bei nicht geänderter Reihenfolge der Parameter sind in




(a) Änderungen im Koeffizientendreieck

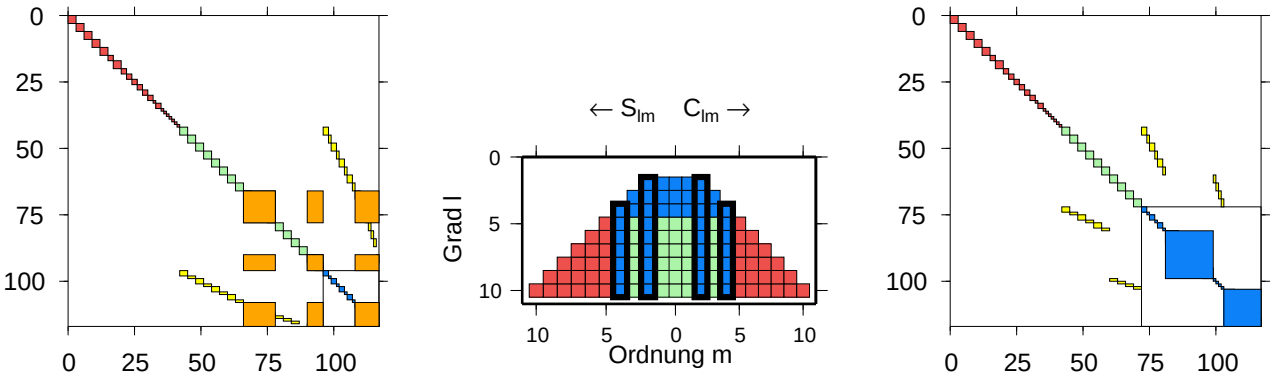



(b) Änderungen in der Kitematrix ohne FKN

Abbildung 48: Änderungen in der Kitematrix durch Berücksichtigung zusätzlicher Korrelationen zwischen ausgesuchten Ordnungen

Abbildung 48(b) farbig dargestellt, die Umrisse der ursprünglichen Form der Kitematrix sind durch gestrichelte Linien innerhalb der gelben und grünen Blöcke gekennzeichnet. Die acht Diagonalblöcke zu den Ordnungen 3 und 4 (vier pro Ordnung) verschmelzen zu einem großen Block. Falls Nebendiagonalelemente zu diesen Ordnungen bestehen, so wachsen die entsprechenden Blöcke ebenfalls zu einem großen Nebendiagonalblock zusammen.

Sollen im Gegensatz dazu Korrelationen zwischen den nicht benachbarten Ordnungen 2 und 4 und alle Korrelationen innerhalb dieser Ordnungen berücksichtigt werden, wie in Abbildung 49(b) durch schwarze Umrandung hervorgehoben, so entstehen mehrere Nebendiagonalblöcke in einer Zeile (Abbildung 49(a)) und die Bedingungen für eine Kitematrix (Abbildung 23, S. 36) sind verletzt. Dadurch entstehen bei der Cholesky–Zerlegung Füllelemente (siehe Abschnitt 4.2) und das Verfahren der Datenkombination mit der Kitematrix funktioniert nicht mehr.




(a) Bedingungen für die Kitematrix verletzt

(b) Koeffizientendreieck mit vollständig korrelierten Ordnungen 2 und 4

(c) Kitematrixstruktur durch Umsortierung mit FKN erhalten

Abbildung 49: Korrelationen zwischen nicht benachbarten Ordnungen zerstören die Struktur der Kitematrix, wenn sie nicht in den Korrelationssektor *Full-Full* umsortiert werden

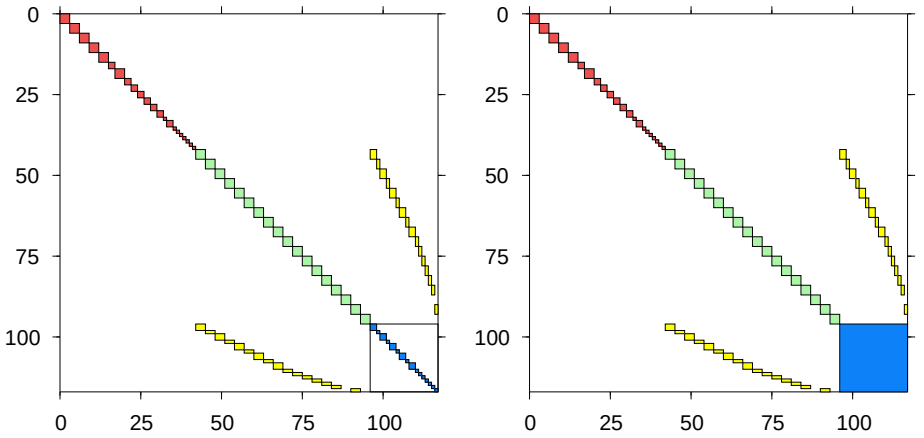
Das Freie Kite–Nummerierungsschema erlaubt es hingegen, für die Korrelationszone *Full* für jede Ordnung den minimalen und den maximalen Grad anzugeben. So ist es möglich, wie in Abbildung 49(b) gezeigt, die Korrelationszone *Full* auf die Ordnungen 2 und 4 bis zum maximalen Grad auszudehnen. Der FKN Algorithmus (Alg. 3.3) sortiert die unbekannten Parameter anschließend so, dass alle Parameter zu den Ordnungen 2 und 4 in der blau dargestellten Korrelationszone *Full* enthalten sind. Dadurch wird zwar der Korrelationssektor *Full-Full* vergrößert, aber die Kitestruktur der Normalgleichungsmatrix bleibt erhalten (Abbildung 49(c)).

### 5.3.2 Berechnung von zusätzlichen Normalgleichungselementen

Nur durch Umsortieren der unbekannten Parameter sind zusätzliche Korrelationen jedoch noch nicht berücksichtigt. Dazu müssen die entsprechenden Normalgleichungselemente aufgestellt werden. Da der letzte Block in der Kitematrix, welcher dem Korrelationssektor *Full-Full* entspricht, immer als vollständige Matrix behandelt wird, können, vom algorithmischen Standpunkt aus betrachtet, beliebige Elemente in diesem Block aufgestellt werden. Es entstehen in keinem Fall zusätzliche Füllelemente.

Die einfachste Methode, diese Technik anzuwenden, soll nun für die Kombination von hochauflösenden, regelmäßigen Daten mit niedrigauflösenden, nicht regelmäßigen Daten wie in Beispiel 2.2 auf Seite 21 erläutert werden. Auch für den regelmäßigen Datensatz werden alle Normalgleichungselemente aus dem Korrelationssektor *Full-Full* aufgestellt, so dass der letzte Block der Kitematrix eine Summe von zwei vollbesetzten Normalgleichungsmatrizen der Dimension 2601 × 2601 ist. Damit ist die Annahme, dass nur Parameter des hochauflösenden Datensatzes mit gleicher Ordnung, gleicher Paraität des Grades und gleicher trigonometrischer Funktion miteinander korreliert sind, für alle Parameter bis $\ell_{\max_n}$ ersetzt durch die Annahme, dass alle Parameter bis $\ell_{\max_n}$ miteinander korreliert sind. Der Beitrag des regelmäßigen Datensatzes zur Kitematrix entspricht daher Abbildung 50(b).

Diese Strategie ist bei ersten Einsätzen des Kite Schemas zunächst verfolgt worden, um die tatsächliche Normalgleichungsmatrix möglichst gut zu approximieren. Es stellte sich jedoch heraus, dass dieser Ansatz oft zu einer nicht positiv definiten Kitematrix führte, weshalb vielfach auf die Blockdiagonalstruktur (Abbildung 50(a)) zurückgegriffen wurde.

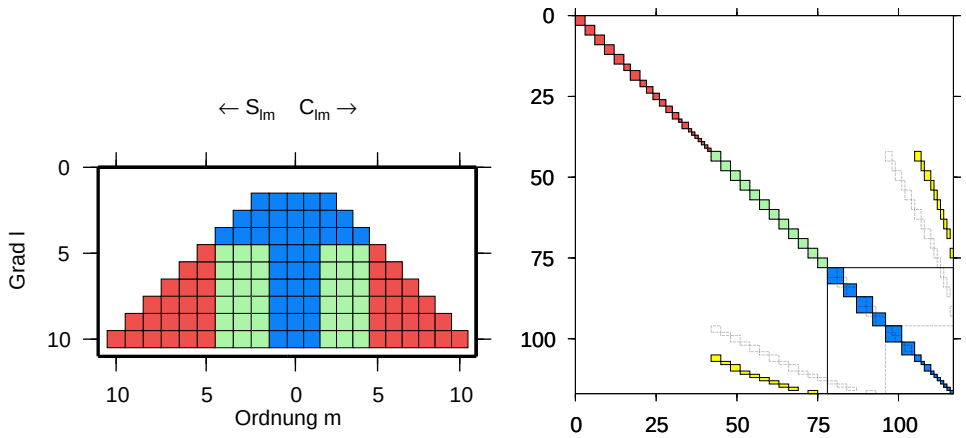



(a) Strenge Kombination (b) Neue Annahmen für die Vorkonditionierung

Abbildung 50: Durch die Annahme, dass bis Grad $\ell_{\max_n}$ auch die Ordnungen zu den regelmäßigen Daten korreliert sind, wird der letzte Block der Kitematrix vollbesetzt

### 5.3.3 Das Zuordnungsproblem

Die Vorkonditionierung beim CG–Verfahren hat, wie oben schon erwähnt, keinen Einfluss auf das Endergebnis, sondern nur auf die Konvergenzgeschwindigkeit. Die bei der sphärischen harmonischen Analyse mit Satellitengradiometerdaten schlecht bestimmbaren zonalen und nahe zonalen Koeffizienten ändern sich zwischen den einzelnen Iterationsschritten sehr stark und verlangsamen daher die Konvergenzgeschwindigkeit. Die Idee, das Schwingen dieser Koeffizienten durch Hinzunahme von Korrelationen zu stabilisieren, führte auf die in Abbildung 51 gezeigte Form der Korrelationszonen im Koeffizientendreieck. Durch diese *Pfeilform* wird die Korrelationszone *Full* für ausgesuchte Ordnungen bis zum maximalen Grad $\ell_{\max_h}$ ausgedehnt. Dadurch werden alle Parameter, die zu diesen Ordnungen gehören aus der Korrelationszone *Semi* in die Korrelationszone *Full* verschoben. Dadurch werden die zu diesen Ordnungen gehörenden Blöcke in den Korrelationssektoren *Semi-Semi* und *Semi-Full* der Kitematrix eliminiert. Im Gegenzug dazu wächst jedoch der vollbesetzte Korrelationssektor *Full-Full* der Kitematrix. Eventuell vorhandene Korrelationen zwischen allen zonalen Koeffizienten $C_{\ell 0}$ und nahe zonalen Koeffizienten werden dadurch voll berücksichtigt. Abbildung 51(b) zeigt, die Veränderung der Kitematrix durch die Verwendung der Pfeilform zur Vorkonditionierung auf. Die gestrichelten Linien stellen die Kitematrix dar, bei der die nahe zonalen und zonalen Koeffizienten als nicht korreliert angesehen werden. Farbig dargestellt ist die Kitematrix unter Berücksichtigung dieser Korrelationen.




(a) Ausdehnung der Korrelationszone *Full* für die Ordnungen 0 bis 10 bis zum maximalen Entwicklungsgrad

(b) Änderung in der Kitematrix durch die zusätzlichen Korrelationen

Abbildung 51: Pfeilform zu Stabilisierung der zonalen Koeffizienten und dadurch zur Verbesserung der Konvergenzeigenschaften

Mit der Forderung die Korrelationszone *Full* des Koeffizientendreiecks und damit den Korrelationssektor *Full-Full* in der Kitematrix frei variieren zu können, muss eine eindeutige Zuordnung zwischen den Elementen der zusätzlichen Normalgleichungen aus nicht regelmäßigen Datensätzen und den Elementen des Korrelationssektors *Full-Full* realisiert werden. Es reicht nun nicht mehr aus, beide Matrizen in die gleiche Reihenfolge umzusortieren, da sowohl der Fall eintreten kann, dass die zusätzliche Normalgleichungsmatrix grösser ist als der Korrelationssektor *Full-Full* als auch der umgekehrte Fall. Damit ist nicht mehr garantiert, dass alle Parameter in der Zielmatrix lückenlos aufeinander folgen. Dies ist jedoch die Grundvoraussetzung für die Verwendung eines Indexvektors zur Umsortierung. In einem Indexvektor $\boldsymbol{I}$ zur Umsortierung von Vektor $\boldsymbol{A}$ nach Vektor $\boldsymbol{B}$ steht an der Stelle $i$ der Index des korrespondierenden Elementes in $\boldsymbol{B}$, so dass gilt

$$\boldsymbol{A}(i) = \boldsymbol{B}(\boldsymbol{I}(i))\,. \tag{81}$$

Somit entsteht eine direkte Indizierung in $\boldsymbol{B}$ durch die Zahl, die in dem Indexvektor an der Stelle $i$ steht. Die Indizierung in $\boldsymbol{A}$ ist jedoch indirekt, da die Elemente des Indexvektors nur durch ihre Position $i$ mit den entsprechenden Elementen aus $\boldsymbol{A}$ verknüpft sind. Werden im Indexvektor zwei Elemente vertauscht, so funktioniert die Zuordnung nicht mehr. Ebenso funktioniert die Zuordnung nicht, wenn in $\boldsymbol{A}$ Elemente auftauchen, die keine Entsprechung in $\boldsymbol{B}$ haben, da der Indexvektor genauso lang sein muss, wie $\boldsymbol{A}$. Abbildung 52 verdeutlicht diesen Zusammenhang. Die Lösung für dieses Problem ist eine Indextabelle, die für Paare von Werten, die in

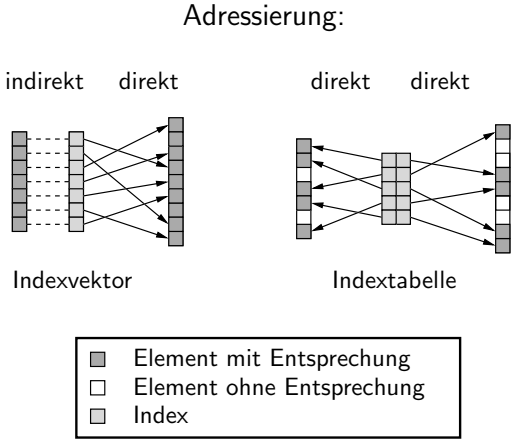



Abbildung 52: Vergleich zwischen Indexvektor und Indextabelle

beiden Vektoren $\boldsymbol{A}$ und $\boldsymbol{B}$ auftauchen jeweils den entsprechenden Index in diesem Vektor speichern. Somit ist eine eindeutige Zuordnung aller korrespondierenden Elemente in beide Richtungen möglich, unabhängig von der Größe der Vektoren. Die einzige Einschränkung ist, dass die Länge der Indextabelle kleiner gleich der Länge des kürzeren Vektors $\boldsymbol{A}$ oder $\boldsymbol{B}$ ist.

$$|I| \leq \min(|A|, |B|)$$

Diese Strategie ist in `pcgma` umgesetzt, so dass mehr Parameter in der zusätzlichen Normalgleichungsmatrix stehen können als in der Korrelationszone *FULL* der FKN Nummerierung vorgesehen sind. Ebenso können in der Korrelationszone *FULL* nun Parameter vorkommen, die nicht in den zusätzlichen Normalgleichungen enthalten sind. Fehlende Informationen werden durch Nullinformationen ergänzt.
Numerische Tests haben gezeigt, dass die Kitematrix mit dem pfeilförmigen Korrelationssektor positiv definit wird und die Konvergenz des CG–Verfahrens damit gegenüber der Vorkonditionierung, bei der nur der blockdiagonale Beitrag der regelmäßigen Daten berücksichtigt wird, (Abbildung 47) erheblich beschleunigt wird.

## 5.4 Umsetzung des Verfahrens PCGMA

Das Verfahren PCGMA ist bereits von Auzinger (1997) und Auzinger und Schuh (1998) implementiert und danach von Plank (2002) parallelisiert worden. Durch zahlreiche Änderungen, die mit der Zeit in das Programm eingeflossen sind, und durch die starre Implementierung der Vorkonditionierung mit der Kitematrix wurde der Quellcode unübersichtlich und schlecht erweiterbar. So konnte das neu entwickelte Kite–Nummerierungsschema

nicht mehr in das alte Programm integriert werden und es wurde daher unumgänglich, das gesamte Verfahren PCGMA neu zu implementieren.
Die Implementierung erfolgte mit der objektorientierten Programmiersprache C++ und es wurde darauf geachtet, das Programm modular und erweiterbar zu gestalten. Der folgende Abschnitt gibt einen Überblick über die Architektur und führt einige implementierungsspezifische Details aus.

### 5.4.1 Überblick über die Architektur des Programms

Zentrale Aufgabenbereiche des Verfahrens PCGMA sind im Programm `pcgma` in Modulen organisiert. In der objektorientierten Programmierung werden Module *Klassen* genannt. Funktionen können Elemente einer Klasse sein, die dann als *Methoden* bezeichnet werden. Eine Klasse dient als ein vom Rest des Programmes getrennter Datenbereich. Die Daten in einer Klasse werden *Attribute* genannt. Nur die Methoden haben Zugriff auf die Attribute der Klasse. Diese Art der Trennung von Daten wird in der objektorientierten Programmierung als *Kapselung* bezeichnet.
Für das Programm `pcgma` sind ca. 90 Klassen entwickelt worden, die aufgrund ihrer Verwendung im ESA Projekt GOCE *Goce Development Kit (GDK)* genannt worden sind. Die zentralen Klassen des GDK sind

- CG
- Distributor
- Observations
- Kite
- Kernel
- SGGparallel
- SSTequations

Das Programm `pcgma` ist nur ein kurzes Steuerungsprogramm, dass die entsprechenden Klassen instanziiert und die Interaktion zwischen ihnen initiiert. Es besteht aus einer *Master* Funktion und einer *Client* Funktion. Die Master Funktion wird aufgerufen, wenn der MPI Rang 0 ist, ansonsten wird die Client Funktion aufgerufen (zur Funktionsweise von MPI siehe Anhang A.2).

**Vorstellung der wichtigsten Klassen:** Die einzelnen Klassen sollen nun jeweils mit ihren wichtigsten Methoden und Attributen in Tabellenform kurz vorgestellt werden.

| Klasse | *CG* |
|---|---|
| Beschreibung | Die Klasse CG führt den Algorithmus der konjugierten Gradienten aus. Zur Durchführung aller Operationen mit der Kitematrix wird die Klasse *Kite* benutzt. Sämtliche Operationen mit der Designmatrix werden durch die Klasse *SGGparallel* ausgeführt, die diese Arbeit auf die vorhandenen Rechenknoten verteilt. Die notwendigen Operationen mit der Normalgleichungsmatrix zu den SST Beobachtungen sind nicht rechenintensiv und werden daher von der Klasse *SSTequations* seriell ausgeführt. |
| Methoden | `relaxation` |
| Attribute | *SGGparallel*<br>*SSTequations*<br>*Kite* |

| Klasse | *Distributor* |
|---|---|
| Beschreibung | Die Klasse Distributor ist für das Einlesen der Daten und für das Verteilen der richtigen Datenpakete an die verfügbaren Computer im Cluster zuständig. Ein Datenpaket stellt jeweils ein Objekt der Klasse *Observations* dar. |
| Methoden | `getObservations` |
| Attribute | *Observations* |

| Klasse | *Observations* | |
|---|---|---|
| Beschreibung | *Observations* stellt ein Paket von Daten für einen Prozessor im Cluster dar. Die Grösse des Pakets ist frei bestimmbar und wird von der Klasse *Distributor* berechnet. Sie richtet sich nach der Anzahl der verfügbaren Prozessoren. | |
| Methoden | `Send` | Versendet sich selbst über MPI |
| | `Recv` | Empfängt sich selbst über MPI |
| Attribute | vector<double> Txx, Tyy, Tzz | Gradiometermessungen |
| | vector<double> x, y, z | Satellitenpositionen |
| | vector<double> time | Messzeitpunkte |
| | vector<*Matrix*> rotations | Rotationsmatrizen |

| Klasse | *Kite* | |
|---|---|---|
| Beschreibung | Die Klasse *Kite* beinhaltet alle Operationen, die mit der Kitematrix durchgeführt werden müssen. Die Speicherung und die Algorithmen sind in Kapitel 3 ausführlich beschrieben. Da für die Aufstellung der Kitematrix die gesamte Designmatrix notwendig ist, wird dieser Schritt parallel gerechnet. Objekte vom Typ *Kite* erstellt die Klasse *Kernel* mit den Beobachtungen aus einem Datenpaket der Klasse *Observations*. Diese Objekte können mit den Methoden `Send` und `Recv` zwischen einzelnen Rechenknoten versendet werden. Dort werden sie anschliessend mit Hilfe des Operators `+=` addiert. Diese Koordinationsaufgabe wird von der Klasse *SGGparallel* durchgeführt. | |
| Methoden | `save` | Laden einer Kitematrix von der Festplatte |
| | `load` | Speichern einer Kitematrix |
| | `Send` | Senden über MPI |
| | `Recv` | Empfangen über MPI |
| | `assemble` | Aufstellen der Kitematrix aus einem Block der Designmatrix |
| | `operator+=` | Addieren einer anderen Kitematrix zu dieser Instanz |
| | `addSST` | SST Normalgleichungsteil zum Korellationsblock *Full* addieren |
| | `chol` | Cholesky–Zerlegung der Kitematrix nach Algorithmus 4.4 |
| | `invert` | Unvollständige Inverse der Kitematrix nach Algorithmus 4.5 |
| | `solve` | Lösung eines Gleichungssystems mit der Kitematrix |
| Attribute | vector<*NeqBlock*> | |
| | vector<*NeqSubblock*> | |

| Klasse | *Kernel* | |
|---|---|---|
| Beschreibung | Der Kernel repräsentiert die Operationen mit der Designmatrix, inklusive der blockweisen Aufstellung. Die Aufstellung der Designmatrix besteht im Wesentlichen aus drei großen Schritten<br>1. Berechnung der zweiten Ableitungen des Potentials und der Legendrepolynome<br>2. Rotation der Designmatrix in das Zielsystem<br>3. Filterung der Designmatrix zur Dekorrelation der Beobachtungen<br>Diese drei Schritte benötigen ungefähr die gleiche Rechenzeit. Der *Kernel* führt die Rechenschritte aus, die in Abbildung 53 rot hervorgehoben sind. Dabei können beliebig viele Instanzen des Kernels gleichzeitig laufen. | |
| Methoden | `buildKitePreconditioner` | Stellt einen Summand der Kitematrix auf. |
| | `processObservationsInitial` | Aufstellung und Anwendung der gesamten Designmatrix |
| | `processObservations` | Aufstellung und Anwendung der gesamten Designmatrix |
| Attribute | | |

| Klasse | *SGGparallel* | |
|---|---|---|
| Beschreibung | *SGGparallel* steuert die parallele Berechnung und Anwendung der Designmatrix, die durch die Beobachtungen der Satellitengradiometrie entsteht. Nur auf dem Master existiert eine Instanz der Klasse *SGGparallel*. Diese veranlasst die Berechnung und Anwendung der Designmatrix auf den Clientprozessoren durch die Klasse *Kernel*. Die Ergebnisse der einzelnen *Kernel* Objekte werden über MPI gesammelt und aufsummiert, bevor sie an die Klasse *CG* weitergegeben werden. | |
| Methoden | `buildKitePreconditioner` | veranlasst parallele Berechnung der Kitematrix |
| | `processObservationsInitial` | veranlasst parallele Anwendung der Designmatrix |
| | `processObservations` | veranlasst parallele Anwendung der Designmatrix |
| Attribute | *Distributor* | Verteilen der Beobachtungen auf die Clientprozessoren |

| Klasse | *SSTequations* | |
|---|---|---|
| Beschreibung | *SSTequations* ist zuständig für alle Operationen mit dem SST Normalgleichungssystem. Dies beinhaltet das Lesen und das Multiplizieren mit der Normalgleichungsmatrix, um den Summanden zum Teil des Residuenvektors zu berechnen, der in der Korrelationszone *Full* liegt. | |
| Methoden | `multiplyInitial` | Berechnung des Residuenvektors im Initialisierungsschritt |
| | `multiply` | Berechnung des Residuenvektors |
| Attribute | | |

### 5.4.2 Programmablauf und Parallelisierung

Nach der Vorstellung der einzelnen Bausteine sollen diese jetzt zu einem Programm zusammengestellt werden. Die Programmabschnitte, die die Hauptrechenzeit verbrauchen werden identifiziert und die gewählte Technik zur Parallelisierung dieser Programmabschnitte wird vorgestellt.
Abbildung 53 zeigt zunächst eine Prinzipskizze des seriellen Programmablaufs von `pcgma`. Diese Darstellung zeigt den Programmablauf aus algorithmischer Sichtweise.

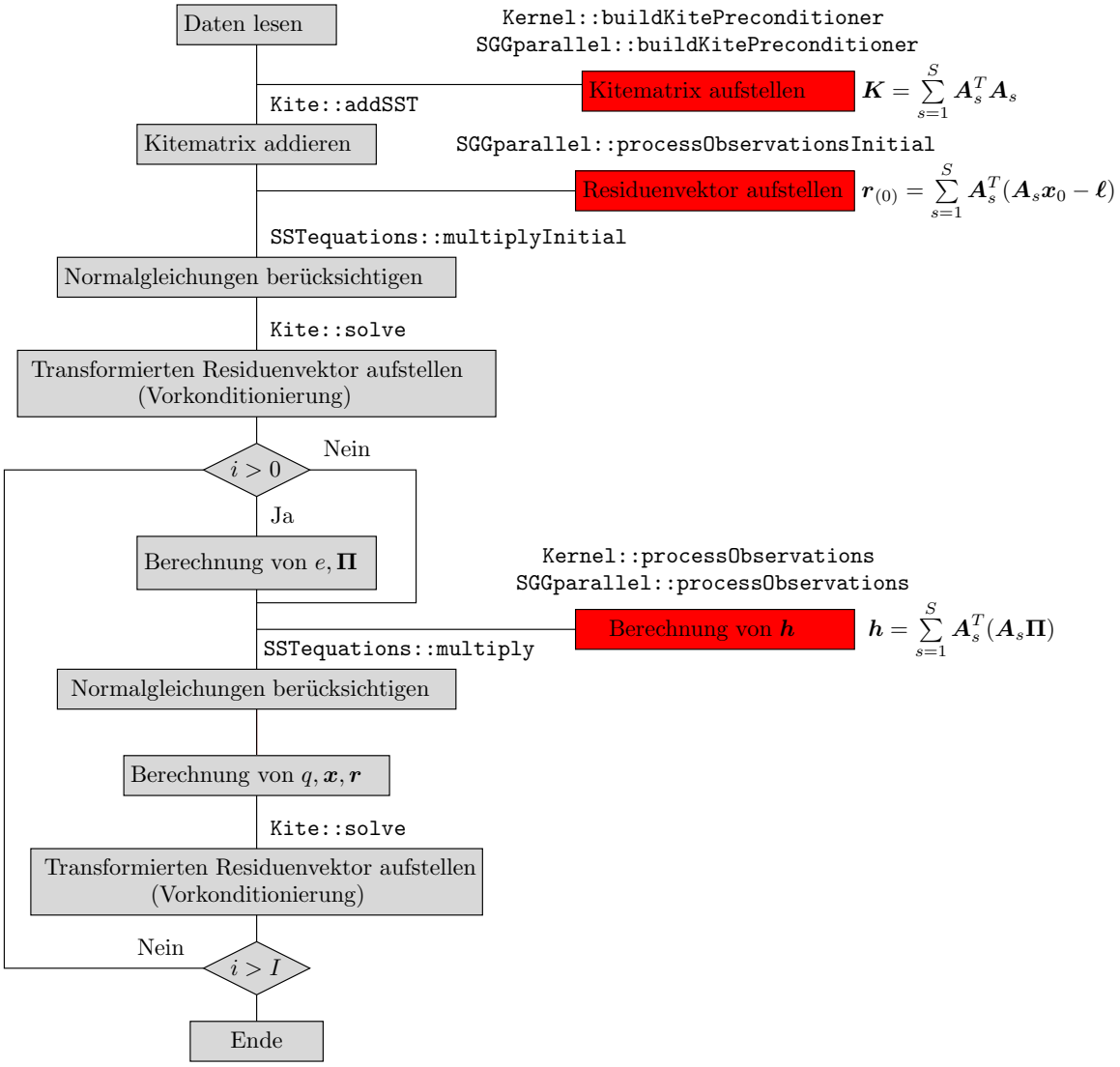



Abbildung 53: Flussdiagramm für das Programm `pcgma`. Rot dargestellt sind besonders zeitintensive Rechenschritte.

# 6 Numerische Untersuchungen zum Einsatz von FKN bei der Vorkonditionierung

## 6.1 Verwendete Datensätze

Die Wirkung der neuen Vorkonditionierungsstrategie soll an einer Kombination von Satellitengradiometriedaten (SGG) mit Satellit–zu–Satellit Beobachtungen (SST) gezeigt werden. Die verwendeteten Datensätze sind simulierte GOCE Daten, welche auf der Grundlage des bekannten Schwerefeldmodells EGM96 berechnet worden sind. Die SGG Daten liegen als beobachtete Schweregradienten $T_{xx}, T_{yy}, T_{zz}$ in den drei Koordinatenrichtungen $x, y, z$ eines lokalen Koordinatensystems vor. Diese stellen die Diagonalglieder der Matrix der zweiten Ableitungen des Gravitationspotentials

$$T = \begin{pmatrix} T_{xx} & T_{xy} & T_{xz} \\ T_{yx} & T_{yy} & T_{yz} \\ T_{zx} & T_{zy} & T_{zz} \end{pmatrix} , \tag{82}$$

des sogenannten Schweretensors dar. Sie werden im Beobachtungsvektor $\boldsymbol{\ell}$ zusammengefasst. Zur Simulation von fehlerfreien Gradienten wird der lineare Zusammenhang

$$\boldsymbol{\ell} = \boldsymbol{A}\boldsymbol{x} \tag{83}$$

zwischen den Potentialkoeffizienten $C_{\ell m}$ und $S_{\ell m}$, welche im Vektor $\boldsymbol{x}$ zusammengefasst sind und den Schweregradienten $T_{xx}, T_{yy}, T_{zz}$ , welche im Vektor $\boldsymbol{\ell}$ zusammengefasst sind, genutzt. Die Designmatrix ist nur von der Bahn des Satelliten abhängig und wird für einen angenommenen GOCE Orbit aufgestellt. Die Multiplikation der Designmatrix mit den Potentialkoeffizienten des EGM96, zusammengefasst im Vektor $\boldsymbol{x}$ ergibt die fehlerfreien, simulierten GOCE–Beobachtungen. Diese fehlerfreien Beobachtungen sind mit einem Rauschen verfälscht worden, das die erwartete Fehlercharakteristik des Messinstruments widerspiegelt. Die auf diesem Wege entstandenen Testdaten besitzen die in der nachfolgenden Tabelle beschriebenen Eckdaten.

| **Verwendete SGG Testdaten** | | | |
|---|---|---|---|
| Positionen | 2 546 134 | Unbekannte Parameter | 22 797 |
| Messungen | 7 638 402 | Signalgehalt | bis Grad und Ordnung 150 |
| Messzeitraum | 29,5 Tage | Schwerefeldmodell | EGM96 |
| Erdumläufe | 471 | Fehlercharakteristik | zu erwartendes Messrauschen |
| Liegt vor als | $T_{xx}, T_{yy}, T_{zz}$ | | |

Zur Bestimmung der niedrigen Frequenzen des Erdschwerefeldes werden die SGG Daten mit SST Daten kombiniert, welche ebenfalls auf Basis des EGM96 simuliert worden sind. Die SST Daten liegen bereits als Normalgleichungssystem bis Grad und Ordnung 50 vor und sind im Gegensatz zu den SGG Daten fehlerfrei. Die nachfolgende Tabelle beschreibt die SST Daten.

| **Verwendete SST Testdaten** | | | |
|---|---|---|---|
| Positionen | 2 546 134 | Unbekannte Parameter | 2597 |
| Messungen | 2 546 134 | Signalgehalt | bis Grad und Ordnung 50 |
| Messzeitraum | 29,5 Tage | Schwerefeldmodell | EGM96 |
| Anzahl Umläufe | 471 | Fehlercharakteristik | fehlerfrei |
| Liegt vor als | $\boldsymbol{N} = \boldsymbol{A}^T\boldsymbol{A}, \quad \boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{\ell}$ | | |

## 6.2 Testszenario

Die verwendeten Daten sind nicht optimal in dem Sinne, dass sie eine Lösung mit der von GOCE angestrebten Genauigkeit erzielen können, denn es handelt sich um Testdaten eines frühen Entwicklungsstadiums, bei denen das Fehlerverhalten des Messinstrumentes schlecht simuliert ist. Trotzdem sind sie geeignet die Wirkung der neuen Vorkonditionierung zu zeigen. Dazu soll zunächst ein Ergebnis präsentiert werden, das in diesem Zusammenhang als Solllösung dienen kann, da es mit dem besten Vorkonditionierer und ausreichend vielen Iterationen berechnet worden ist. Es stellt daher das beste Ergebnis dar, welches mit diesen Daten und der angegebenen Methode erreichbar ist. Für die Auswertung der Daten bis zum maximalen Entwicklungsgrad von 150 sind 192 Prozessoren verwendet worden, wie die erste Zeile des Logfiles belegt:

```
1 ATTENTION: 0031-408  192 tasks allocated by LoadLeveler, continuing... .
```

Die Auswertezeit betrug 2 Stunden und 22 Minuten was den letzten Zeilen des Logfiles zu entnehmen ist:

```
1 INFO:  Writing the final solution in CTM_EGM_I
2 INFO:  Creating CTM_COM_RWP_I
3 INFO:  Saving the filtered residuals in CTM_COM_RES
4 INFO:  Time elapsed: 8534.81 seconds.
5 INFO:  Finished.
```

Um die Ergebnisse der Berechnungen in einer standardisierten Form zu erhalten, wurde am Institut für Theoretische Geodäsie eine Software entwickelt, die die Ausgabedateien von `pcgma` ohne Benutzerinteraktion in einem ca. 10–seitigen Technischen Bericht auswertet und darstellt. Das berechnete Schwerefeld wird mit einem vorher angegebenen Sollmodell verglichen, wobei die zu untersuchende Lösung mit *test* und das Sollmodell mit *true* bezeichnet wird. Ausgesuchte Darstellungen aus dem Technischen Bericht werden im Folgenden benutzt, um das Referenzergebnis für diesen Testdatensatz im Vergleich zum EGM96 darzustellen. Im nachfolgenden Abschnitt werden drei Vorkonditionierungsvarianten auf die gleiche Art miteinander verglichen, um die Effizienz der neuen Vorkonditionierung zu zeigen. Für die Darstellung der Ergebnisse in Abbildung 54 werden die Differenzen der

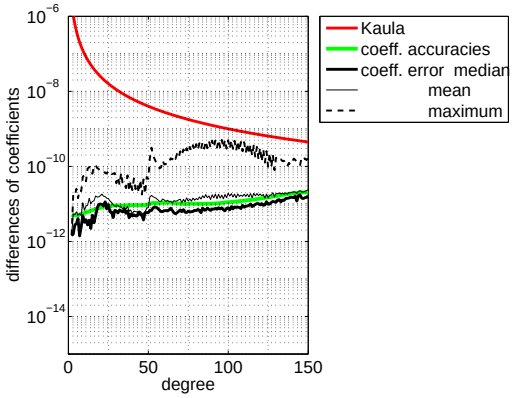



Abbildung 54: Vergleich der Koeffizienten mit denen des EGM96. Die Ergebnisse wurden erhalten durch Vorkonditionierung mit der Variante *Pfeil*

Koeffizienten

$$\Delta C_{\ell m} = |C_{\ell m}^{(\text{test})} - C_{\ell m}^{(\text{true})}|, \quad \Delta S_{\ell m} = |S_{\ell m}^{(\text{test})} - S_{\ell m}^{(\text{true})}| \tag{84}$$

gebildet und gradweise zusammengefasst, wobei der Mittelwert

$$\sigma_\ell^{\text{mean}} = \frac{1}{2\,\ell + 1} \sum_{m=0}^{\ell} (\Delta C_{\ell m} + \Delta S_{\ell m}) \,, \tag{85}$$

der Median

$$\sigma_\ell^{\text{median}} = \text{median}\,(\Delta C_{\ell m}, \Delta S_{\ell m}) \quad \text{mit} \quad m \in \{0, \ldots, \ell\} \tag{86}$$

und das Maximum

$$\sigma_\ell^{\text{max}} = \max\,(\Delta C_{\ell m}, \Delta S_{\ell m}) \quad \text{mit} \quad m \in \{0, \ldots, \ell\} \tag{87}$$

verwendet werden. Diese drei Kurven werden mit einer logarithmischen, einheitslosen Skala auf der y–Achse und mit dem harmonischen Grad auf der x–Achse dargestellt. Die in rot eingezeichnete Kurve basiert auf der sogenannten *rule of thumb* von Kaula (1966)

$$\frac{\sigma_\ell}{\sqrt{2\ell + 1}} \approx \frac{10^{-5}}{\ell^2} \,, \tag{88}$$

die abschätzt, welche Größenordnung die Gradvarianz

$$\sigma_\ell^2 = \sum_{m=0}^{\ell} (C_{\ell m}^2 + S_{\ell m}^2) \tag{89}$$

bei dem Grad $\ell$ annehmen wird. Dargestellt ist der Mittelwert der Varianz pro Grad

$$\sigma_\ell^{\text{kaula}} = \frac{\sigma_\ell}{\sqrt{2\,\ell+1}} = \frac{10^{-5}}{\ell^2} \tag{90}$$

der abschätzt, welche Grössenordnung die Koeffizienten eines Grades haben. In grün eingezeichnet sind die geschätzten Standardabweichungen $\sigma_\ell^{(\text{test})}$ der geschätzten Parameter, jeweils für den Grad $\ell$ mit

$$\sigma_\ell^{(\text{test})} = \text{median}\,(\sigma_{C_{\ell m}}, \sigma_{S_{\ell m}}) \quad \text{mit} \quad m \in \{0, \ldots, \ell\} \tag{91}$$

zusammengefasst. Bei der Berechnung mit `pcgma` gehen diese Werte aus der Diagonalen der unvollständigen Inversen der Kitematrix hervor. Sie stellen Näherungswerte dar, weil die Kitematrix eine Approximation der unbekannten Normalgleichungsmatrix ist. Wie Vergleiche mit der vollständigen Varianz–Kovarianzmatrix aus DNA gezeigt haben, stimmen diese jedoch gut mit den korrekten Standardabweichungen überein. Die Kurven der gradweisen Zusammenfassungen sollten unter der in grün eingezeichneten Kurve der geschätzten Standardabweichungen liegen, damit gesichert ist, dass die Abweichungen der Koeffizienten vom Sollmodell im Bereich der Standardabweichung liegen. Dies trifft im vorliegenden Szenario nur für das robuste Maß Median zu, der Mittelwert liegt von Grad 2 bis Grad 40 und über Grad 50 über der Fehlerkurve, was auf starke Unterschiede der Koeffizientendifferenzen innerhalb der Grade deutet. Dies bestätigt die Kurve der Maxima der Koeffizientendifferenzen, welche teilweise eine Größenordnung über der Fehlerkurve liegt. Welche Ordnungen innerhalb eines Grades diese Abweichungen aufweisen ist in Abbildung 55 abzulesen, welche die relativen Abweichungen

$$\left|\frac{\Delta C_{\ell m}}{\sigma_{C_{\ell m}}^{(\text{test})}}\right| \quad \text{bzw.} \quad \left|\frac{\Delta S_{\ell m}}{\sigma_{S_{\ell m}}^{(\text{test})}}\right| \tag{92}$$

in einem Koeffizientendreieck darstellt. Abbildung 54 fasst jeweils eine Zeile des Koeffizientendreiecks zu einem

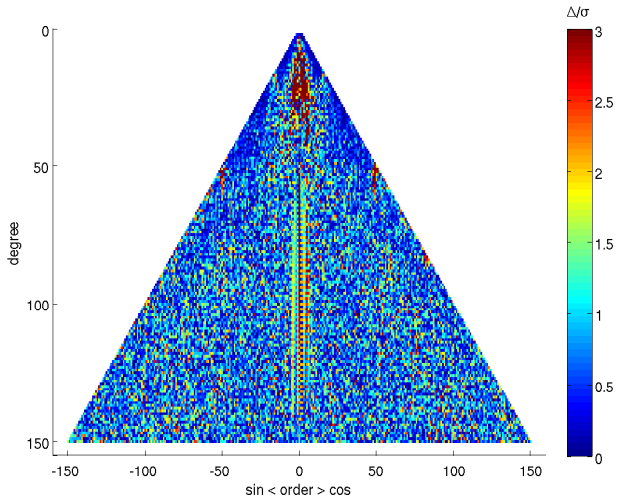



Abbildung 55: Relative Unterschiede zwischen den Koeffizienten nach (92)

Wert zusammen, weshalb Abbildung 55 Aufschluss über die Verteilung der Abweichungen innerhalb eines Grades geben kann. Hier fallen die Abweichungen der zonalen und nahe–zonalen Koeffizienten auf, die besonders bei den Graden zwischen 2 und 40 und über 50 auftreten, dort wo auch die Maxima in Abbildung 54 sehr gross sind.
Ein weiteres wichtiges Werkzeug im Technischen Bericht ist die Darstellung der Gradabweichungen zwischen den Modellen ausgedrückt als Geoidhöhen

$$\sigma_\ell(N) = R\,\sigma_\ell \qquad \text{mit} \qquad \sigma_\ell = \sqrt{\sum_{m=0}^{\ell} (\Delta C_{\ell m}^2 + \Delta S_{\ell m}^2)} \quad \text{und} \quad R = \text{Erdradius}\,, \tag{93}$$

sowohl gradweise, als auch kumulativ, bei den hohen Graden beginnend aufsummiert. Abbildung 56 zeigt dies an dem vorliegenden Beispiel. Auf der x–Achse sind die harmonischen Grade und auf der logarithmischen y–Achse

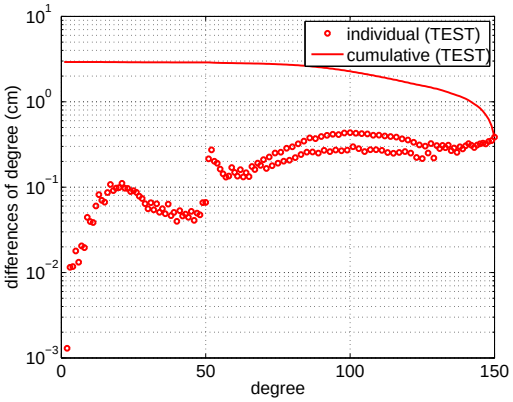



Abbildung 56: Graddifferenzen als gradweise und kumulative Geoidhöhen ausgedrückt

die Geoidhöhen in cm aufgetragen. Im vorliegenden Szenario ist das EGM96 bis auf 3 cm global reproduziert, die grössten Abweichungen liegen im SGG Bereich über Grad 70 und bei Grad 50 ist ein Sprung zu erkennen, der auf einen nicht optimalen Gewichtungsfaktor zwischen den SST Daten und den SGG Daten deutet. Dieser Sprung ist auch in Abbildung 54 zu erkennen und Simulationen haben gezeigt, dass dieser mit einem geeigneten Gewichtungsfaktor eliminiert werden kann.

Die Darstellung aus Abbildung 56 wird auch dazu genutzt, den Verlauf der Iteration zu analysieren. Dabei werden alle Iterationen in einem Diagramm zusammengefasst, indem für jede Iteration eine Kurve mit einer anderen Farbe gezeichnet wird. Abbildung 57 zeigt den Verlauf einer Berechnung mit 14 Iterationen. Die geographische

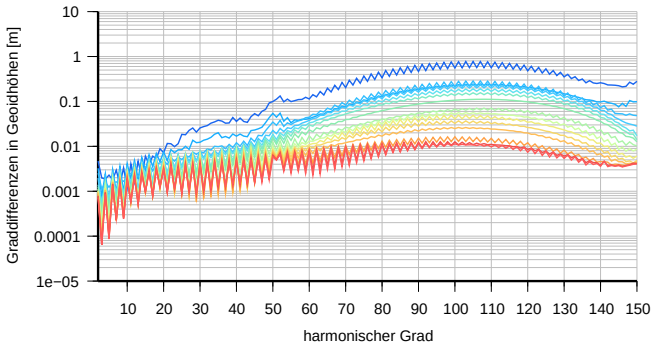
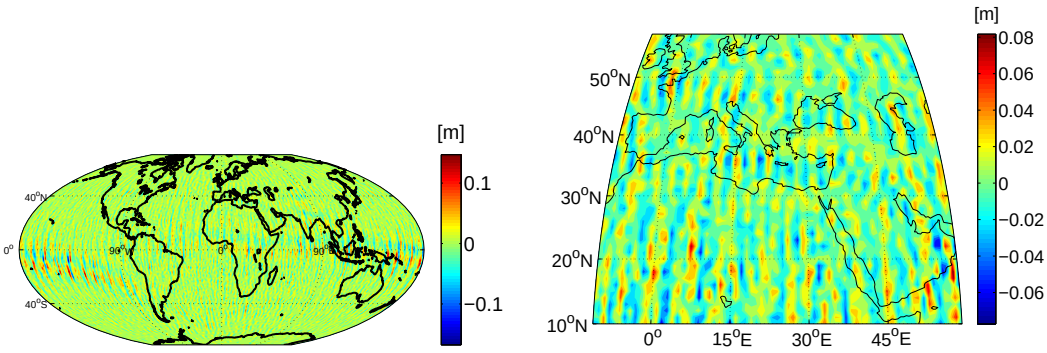



Abbildung 57: Analyse der Konvergenz durch Darstellung Koeffizientendifferenzen jeder einzelnen Iteration ausgedrückt in Geoidhöhen. Die Iterationen sind von oben nach unten aufgetragen.

Zuordnung von Berechnungsfehlern ist schließlich mit den in Abbildung 58 gezeigten Darstellungen der Koeffizientenunterschiede ausgedrückt als Geoidhöhen über einem Referenzellipsoid. Die globale Darstellung zeigt im vorliegenden Fall ein Muster über den gesamten Bereich des Äquators, welches auf systematisches Rauschen bei der Generierung der Testdaten zurückzuführen ist. Um die feinen Strukturen des Unterschiedes zwischen zwei Berechnungen besser erkennen zu können, wird die globale Darstellung der Geoidhöhen aus Abbildung 58(a) durch eine Darstellung eines Ausschnitts eines bestimmten Bereichs in Abbildung 58(b) ergänzt. Die Abweichungen sind für die lokale Darstellung mit einem Hochpassfilter gefiltert worden, um die Feinstrukturen des Schwerefeldes besser sehen zu können.

## 6.3 Vergleich von drei Varianten der Vorkonditionierung

Es wurden drei Berechnungen durchgeführt, die sich in der Wahl der Vorkonditionierungsmatrix unterscheiden. Die drei gewählten Varianten der Vorkonditionierung werden im Folgenden mit *diagonal*, *blockdiagonal* und *Pfeil* bezeichnet. Die Koeffizientendreiecke und der SGG Beitrag zur Kitematrix werden für jede Variante in den Abbildungen 59, 60 und 61 dargestellt.




(a) globale Darstellung (b) lokale Darstellung

Abbildung 58: Darstellung der Modellunterschiede in Geoidhöhen nach (93)

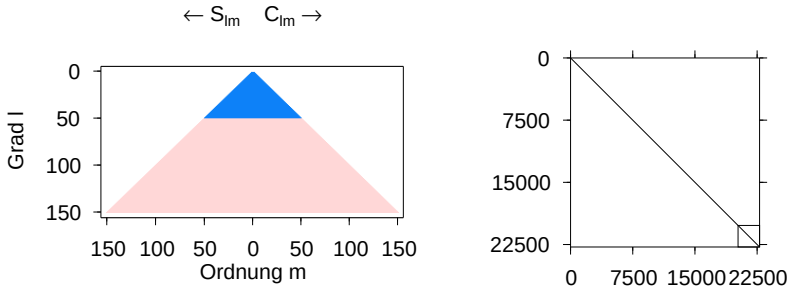



Abbildung 59: Variante *diagonal*: Für die Vorkonditionierung wird angenommen, dass sämtliche Parameter des SGG Datensatzes unkorreliert sind, daher werden für die Vorkonditionierung nur die Diagonalelemente der SGG Normalgleichungen verwendet.

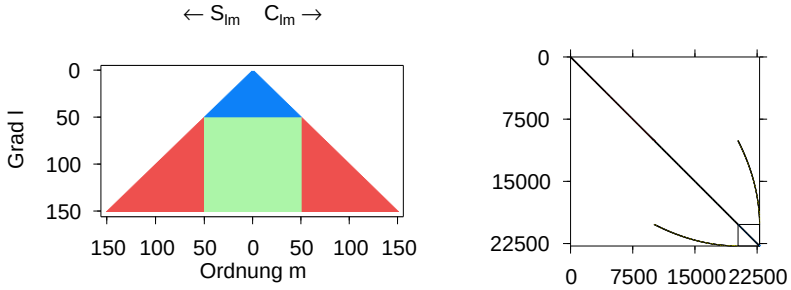



Abbildung 60: Variante *blockdiagonal*: Die SST Normalgleichungsmatrix wird zu dem unten links angedeuteten Korrelationssektor addiert.

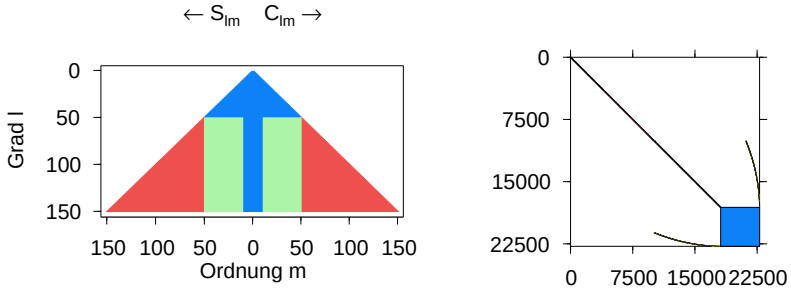
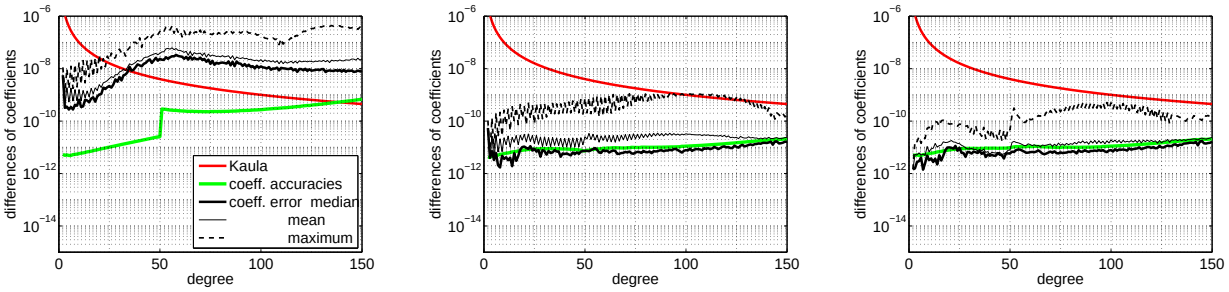



Abbildung 61: Variante *Pfeil*: Der Beitrag der SGG Beobachtungen zur Kitematrix enthält einen vollbesetzten Korrelationssektor *Full-Full* der größer ist als die SST Normalgleichungsmatrix, weil alle Korrelationen zwischen den Parametern der Ordnungen 0 bis 10 bis zum maximalen Grad 150 berücksichtigt werden.

## 6.4 Ergebnisse

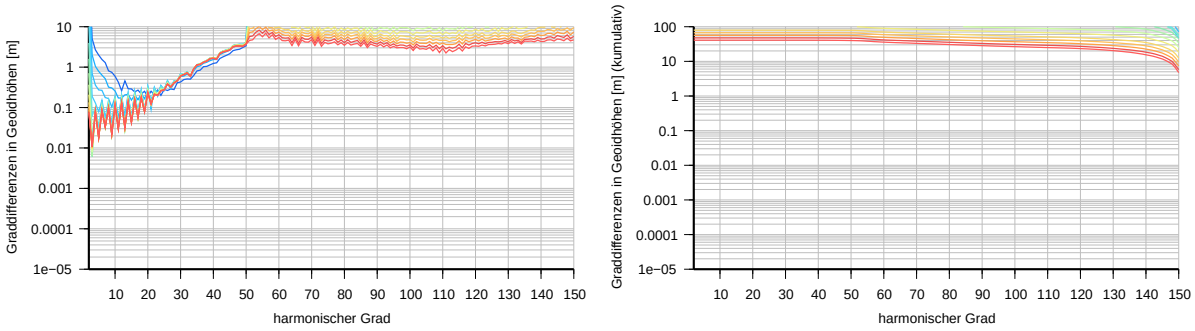
Die Ergebnisse der drei Varianten der Vorkonditionierung sollen nun miteinander verglichen werden. Dazu soll zunächst für jede Variante der gradweise Vergleich der Koeffizienten mit dem EGM96 gezeigt werden, um zu dokumentieren, wie nah die einzelnen Varianten nach 15 Iterationen an das Sollergebnis gekommen sind.




(a) Variante *diagonal* (b) Variante *blockdiagonal* (c) Variante *Pfeil*

Abbildung 62: Vergleich der drei Vorkonditionierungsvarianten mit dem EGM96

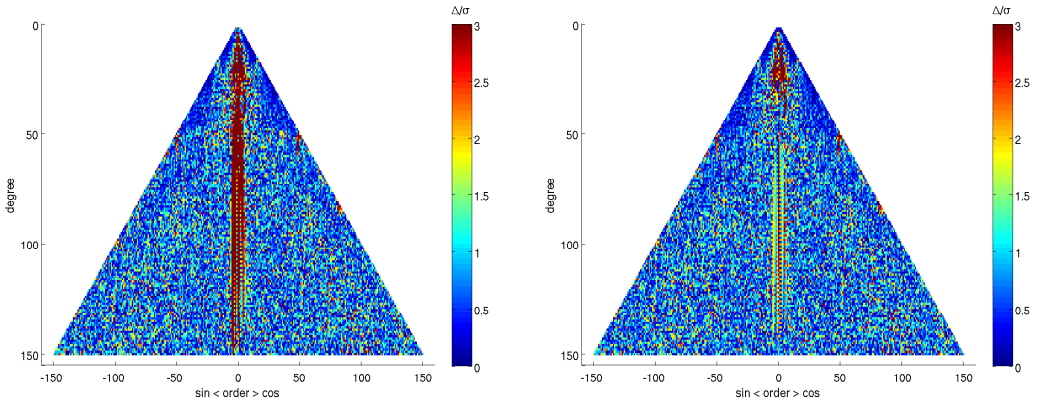
Aufgrund der schlechten Ergebnisse mit der Variante *diagonal*, welche auch durch die Dokumentation der Iterationsschritte in Abbildung 63 belegt wird, kann diese von den weiteren Betrachtungen ausgeschlossen werden. Die global erreichte Abweichung in Geoidhöhen vom EGM96 ist noch über 30 Meter.




(a) gradweise (b) kumulativ

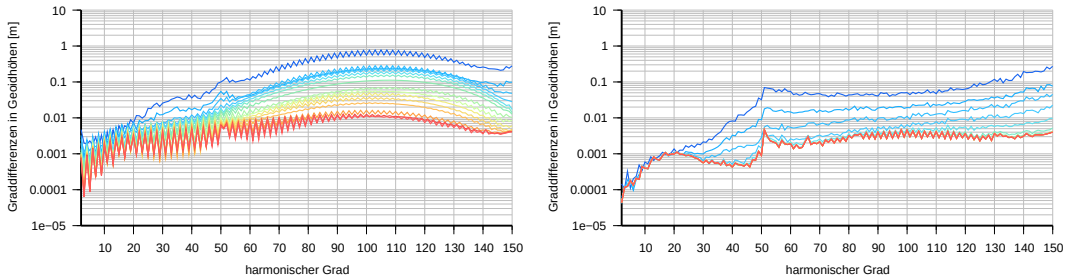
Abbildung 63: Konvergenzdarstellung von 15 Iterationen mit der Vorkonditionierungsvariante *diagonal*

Das Hinzunehmen der vier Diagonalblöcke pro Ordnung durch die Variante *blockdiagonal* verbessert die Konvergenz des Verfahrens deutlich, so ist z. B. das Ergebnis des Medians der Koeffizientendifferenzen pro Grad in Abbildung 62(b) mit dem Ergebnis aus der Variante *Pfeil* (Abbildung 62(c)) vergleichbar. Dies trifft jedoch nicht für den Mittelwert und das Maximum zu, denn der Mittelwert liegt durchgehend über der Kurve der Genauigkeiten und das Maximum ist teilweise so groß, dass es an die Kaula–Kurve herankommt. Dies legt die Vermutung nahe, dass innerhalb der Grade starke Abweichungen der Koeffizientenunterschiede bestehen. Diese Vermutung wird durch die Differenzen (92), welche Abbildung 64 in Koeffizientendreiecken darstellt, bestätigt, denn im Bereich der zonalen Koeffizienten in der Mitte des Dreiecks sind in allen Graden noch starke Abweichungen vom EGM96 zu erkennen. Diese Koeffizienten werden durch die Testdaten schwach bestimmt, weshalb das CG–Verfahren für die Festlegung dieser Koeffizienten noch weitere Iterationen braucht. An dieser Stelle setzt die neue Vorkonditionierungsstrategie ein, die es erstmals erlaubt, zusätzliche Korrelationen zwischen den unbekannten Parametern zu berücksichtigen. Wie Abbildung 61 zeigt, können jetzt die Korrelationen zwischen den schwach bestimmten, zonalen Koeffizienten in der Kitematrix berücksichtigt werden. Dieses Verfahren führt zu den in Abbildung 64(b) gezeigten, besseren Ergebnissen. Die Ergebnisse stimmen mit der Solllösung überein, wobei eine Übereinstimmung im Bereich der geschätzten Genauigkeit, wie der Verlauf in Abbildung 65(b) zeigt, statt nach 15 Iterationen bereits nach der 6. Iteration eintritt. Die Variante *blockdiagonal* wird zu demselben Ergebnis führen, jedoch wären noch weitere Iterationen notwendig, wie aus Abbildung 65(a) abzulesen ist.
Es ist gezeigt worden, dass die neue Vorkonditionierungsstrategie durch die Möglichkeiten der Freien Kite–Nummerierung in der Lage ist, auf die Charakteristik der auszuwertenden Daten einzugehen, wodurch eine




(a) Variante *blockdiagonal* (b) Variante *Pfeil*

Abbildung 64: Relative Unterschiede zwischen den Koeffizienten nach (92)




(a) Variante *blockdiagonal* (b) Variante *Pfeil*

Abbildung 65: Darstellung der Konvergenz bei 15 Iterationen. Die Iterationen sind von oben nach unten aufgetragen.

deutlich schnellere Konvergenz erzielt wird. Im vorliegenden Beispiel konnte die Hinzunahme von Korrelationen zwischen den Koeffizienten bis zum maximalen Entwicklungsgrad für die Ordnungen 0 bis 10 die Anzahl der notwendigen Iterationen auf ein Drittel reduzieren. Das Freie Kite–Nummerierungsschema kann nun in dem bestehenden Softwarepaket `pcgma` angewendet werden, um bei der Auswertung der GOCE Daten durch an die Daten angepasste Vorkonditionierung kurze Rechenzeiten zu ermöglichen.

# 7 Zusammenfassung und Ausblick

In dieser Arbeit wurde gezeigt, welche Probleme bei der sphärischen harmonischen Analyse auftreten, wenn hochauflösende regelmäßige Datensätze mit niedriger auflösenden, nicht regelmäßigen Daten in einer gemeinsamen Ausgleichung kombiniert werden. Die Auswirkung der Variation der Reihenfolge der unbekannten Parameter auf die Besetztheitsstruktur der Normalgleichungsmatrix des regelmäßigen Datensatzes und auf die aus der Kombination resultierende Normalgleichungsmatrix wurde analysiert und auftretende Speicherbeschränkungen wurden aufgezeigt. Die Arbeit von Schuh (1996) wurde fortgeführt, indem die Schwachpunkte des Kite–Nummerierungschemas behoben und das Freie Kite–Nummerierungsschema (FKN) entwickelt worden ist. Dieses erlaubt, bei der Kombination unterschiedlicher Datensätze für jede Ordnung jedes Datensatzes einen minimalen und einen maximalen Grad anzugeben, unter Beibehaltung der Kitestruktur der kombinierten Normalgleichungsmatrix. Zur Nutzung des freien Kite–Nummerierungsschemas wurde ein Speicherschema vorgestellt, das die Blöcke der Normalgleichungsmatrix in zwei Vektoren von Matrizen speichert. Die mit der Kitematrix notwendigen Operationen zum Lösen eines Normalgleichungssystems und zur Abschätzung der Varianzen der geschätzten Parameter wurden an die Struktur der Kitematrix und das gewählte Speicherschema angepasst. Mit dem Freien Kite–Nummerierungschema kann eine strenge Datenkombination von regelmäßigen und nicht regelmäßigen Datensätzen bei frei wählbarer Parametrisierung mit geringem Speicherplatzbedarf berechnet werden. Ein weiterer Einsatzbereich für das Freie Kite–Nummerierungsschema ist die Vorkonditionierung bei iterativen Verfahren. Das PCGMA Verfahren von Schuh musste neu implementiert werden, um das Freie Kite–Nummerierungschema integrieren zu können. Das dabei entstandene Programm `pcgma` wird zur Auswertung der GOCE Daten im Auftrag der ESA verwendet werden. Es wurde parallel implementiert und bereits mit bis zu 512 Prozessoren auf dem Supercomputer JUMP am Forschungszentrum Jülich erfolgreich eingesetzt.
Testrechnungen belegten den Erfolg des Freien Kite–Nummerierungsschemas bei der Vorkonditionierung und erlaubten die Reduktion der notwendigen Iterationen um 50-70 Prozent. Das Programm `pcgma` ermöglicht in Verbindung mit FKN eine Vielzahl von Untersuchungen zur datenangepassten Vorkonditionierung und ist somit ein wichtiges Werkzeug für die zukünftige Auswertung der GOCE Daten.

# A Numerische Optimierung

In diesem Abschnitt werden Methoden vorgestellt, um die Laufzeit von numerischen Algorithmen zu verkürzen. Die beiden Hauptsäulen sind dabei die Hardwareoptimierung durch optimierte Bibliotheken und die Verwendung von mehreren Computern parallel. Detailliertes Hintergrundwissen zu Optimierung von numerischen Algorithmen auf heutigen Prozessoren liefert Whaley (2004).

## A.1 Hardwareoptimierung

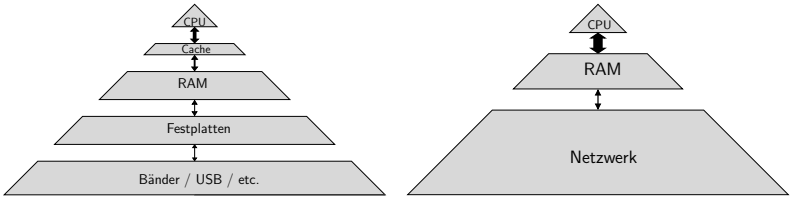
Unter dem Begriff Hardwareoptimierung wird in diesem Kapitel die Beschleunigung der Ausführungsgeschwindigkeit von Algorithmen der linearen Algebra verstanden. Es wird gezeigt, dass die Rechengeschwindigkeit moderner Prozessoren mit einem einfachen Programmieransatz oft nur zu einem Bruchteil ausgenutzt werden kann. Mit relativ geringem Aufwand können numerische Algorithmen jedoch so umstrukturiert werden, dass eine wesentlich höhere Effizienz erreicht wird.

### A.1.1 Optimierung von Speicherzugriffen

Computer besitzen im Allgemeinen eine hierarchische Speicherstruktur. Das obere Ende dieser Hierarchie bildet vereinfacht gesagt, der Prozessor. An den Prozessor angeschlossen ist eine kleine Menge sehr schnellen Speichers. Dieser ist wiederum mit einer größeren Menge von Speicher verbunden, der nicht ganz so schnell ist. Diese schnellen Speicher werden *caches* genannt, wobei der schnelle Speicher, der mit dem Prozessor verbunden ist *Level 1 Cache (L1)* und der größere *Level 2 Cache (L2)* genannt wird. An den L2 Cache ist der Hauptspeicher angeschlossen, der wiederum größer, aber auch langsamer ist. Der Sinn des Caches liegt darin, kürzlich verwendete Daten nicht jedesmal aus dem relativ langsamen Hauptspeicher laden zu müssen, sondern diese Daten in einem Zwischenspeicher vorzuhalten zu können, auf den fast ohne Verzögerung zugegriffen werden kann. Wenn eine Speicherstelle im Cache gefunden wird, bezeichnet man das als *Cache Hit*, ansonsten als *Cache Miss*. Tritt ein Cache Miss auf, so wird die gesuchte Speicherstelle aus dem Arbeitsspeicher in den Cache geladen, wodurch eine Verzögerung eintritt. Hierbei wird davon ausgegangen, dass das Programm auch weitere Speicherstellen aus diesem Bereich ansprechen wird.
Deswegen sind Caches so aufgebaut, dass immer eine feste Anzahl von Bytes gleichzeitig gelesen wird. Diese festgelegte Bytefolge nennt man *Cache-Line*. Bei einer Länge von 128 Byte passen in eine Cache Line Beispielsweise 16 Fließkommazahlen im double-Format. Mit Caches wird versucht, die mittlere Zugriffsgeschwindigkeit auf die Daten des Hauptspeichers zu verringern, basierend auf den Annahmen

- Die Werte werden mehrfach verwendet und
- wurde ein Wert im Speicher benutzt, so wird mit hoher Wahrscheinlichtkeit anschließend der nachfolgende Wert benutzt.

**Hierarchisches Speichermodell** Diese Strategie, die langsame Übertragungsgeschwindigkeit eines Speichermediums durch Zwischenschalten eines kleineren, schnellen Speichers zu verstecken (engl. to cache) lässt sich auf alle Hierarchieebenen eines Computers übertragen. In Abbildung 66(a) ist das Hierarchiemodell eines PC's dargestellt. Beispielsweise dient der Cache dazu, die langsame Geschwindigkeit des Arbeitsspeichers vor der CPU zu "verstecken"., indem er gelesene Werte zwischenspeichert, in der Annahme, dass sie ein zweites Mal verwendet werden. Auf dem Feld der parallelen Programmierung kann zum Beispiel der Arbeitsspeicher dazu dienen, kürzlich über das Netzwerk empfangene Daten zwischenzuspeichern, um sie bei mehrmaliger Verwendung schneller aus dem Arbeitsspeicher als aus dem Netzwerk laden zu können (Abbildung 66(b)). Das Prinzip




(a) Einzelner Computer (b) Computer im Netzwerk

Abbildung 66: Hierarchieebenen eines Computers

der Zwischenspeicherung basiert auf der Annahme, dass die Wahrscheinlichkeit für die Verwendung benachbarter Speicherplätze höher ist als die von weiter entfernten. Je eher die Annahmen zutreffen desto besser kann die Arbeitsgeschwindigkeit durch Zwischenspeicherung verbessert werden. Durch geeignete Organisation der Daten kann der Programmierer zur Erfüllung dieser Voraussetzungen beitragen. Diese Vorteile können besonders auf numerische Algorithmen übertragen werden.

**Profitieren von Cache–Speichern:** Die Technik der Ausnutzung von Cache–Speichern soll nun an einem Beispiel erläutert werden. Ziel dieser Technik ist es, zur Durchführung eines Algorithmus die Daten so zu organisieren, dass mehrfach verwendete Daten nahe beieinanderliegen, und dadurch möglichst viele Cache Hits erzielen werden. Weiterhin ist die Berechnungsreihenfolge so zu wählen, dass die Wiederverwendungsrate der Zahlen im Cache möglichst hoch ist. Die Matrizenmultiplikation eignet sich besonders für diese Umorganisation von Daten.

**Beispiel A.1 (Matrizenmultiplikation)** *Es sollen zwei Matrizen $\boldsymbol{A}$ und $\boldsymbol{B}$ der Dimension $n \times n$ nach Algorithmus (A.1) miteinander multipliziert werden und das Ergebnis zur bestehenden Matrix $\boldsymbol{C}$ addiert werden.*

$$\boldsymbol{C} = \boldsymbol{C} + \boldsymbol{A}\boldsymbol{B} \qquad \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{n \times n} \tag{94}$$

■

**Algorithmus A.1 (Matrizenmultiplikation)**

```
1  void multiply( Matrix &A, Matrix &B, Matrix &C )
2  {
3      for( int i = 0; i < C.rows(); ++i )
4          for( int j = 0; j < C.columns(); ++j )
5              for( int k = 0; k < A.columns(); ++k )
6                  C(i,j) += A(i,k) * B(k,j);
7  }
```

Bei den nachfolgenden Betrachtungen wird von einem vereinfachten Computermodell mit den Hierachieebenen Prozessor, Cache und Arbeitsspeicher ausgegangen. Der Prozessor kann drei Zahlen gleichzeitig in seinem internen Speicher, den *Registern* halten. Jeder Wert, der für eine Berechnung verwendet wird, muss vorher von dem Prozessor geladen werden. Dazu schaut er zunächst im Cache nach, ob diese Zahl dort vorhanden ist, und lädt sie mit geringem Zeitverzug von dort. Da die Berechnungsgeschwindigkeit heutiger Prozessoren höher als die Lesegeschwindigkeit von Caches ist, kann die Zeit $T_{\text{arith}}$ für die eigentliche Berechnung vernachlässigt werden. Mit $T_{\text{arith}}$ wird im Folgenden nur die Zeit bezeichnet, die benötigt wird, um eine Zahl aus dem Cache zu laden oder in ihn zu schreiben. Ist die Zahl jedoch nicht im Cache, so muss die Berechnung unterbrochen und die Zahl aus dem Arbeitsspeicher in den Cache geladen werden, wobei eine Verzögerung von $T_{\text{mem}}$ eintritt (siehe Abbildung 67). Der Zeitaufwand $T_{\text{mult}}$ für die Matrizenmultiplikation ist daher die Summe der Zeiten für die

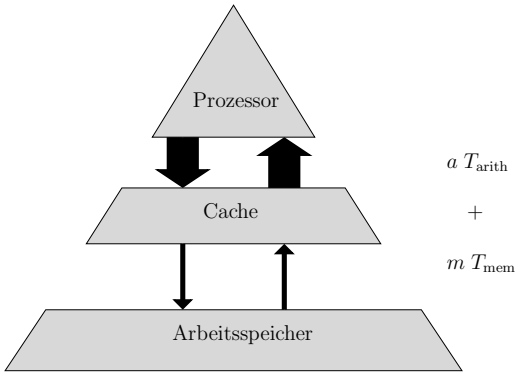



Abbildung 67: Vereinfachtes Speichermodell

Rechenoperationen $T_{\text{arith}}$ und die Speicherbewegungen $T_{\text{mem}}$

$$T_{\text{mult}} = a\,T_{\text{arith}} + m\,T_{\text{mem}}\,. \tag{95}$$

Die Anzahl der Zahlen, die zwischen Cache und Prozessor kommuniziert werden, wird mit $a$ und die Summe der Speicherbewegungen aus und in den Arbeitsspeicher mit $m$ bezeichnet. Es soll nun eine Aufteilung der Matrizenmultiplikation in kleinere Einheiten gefunden werden, bei der $m$ minimiert wird.
Im Fall der Matrizenmultiplikation von quadratischen Matrizen müssen $n^3$ Multiplikationen und $n^3$ Additionen durchgeführt werden. Um das Ergebnis zum vorherigen Wert der Matrix zu addieren, muss diese geladen und zum Schluß geschrieben werden. Da die Zugriffsgeschwindigkeit auf den Cache meistens eine Größenordnung schneller als die Zugriffsgeschwindigkeit auf den Arbeitsspeicher ist, soll in unserem Beispiel $T_{\text{arith}} = 0.1$ Sekunden und $T_{\text{mem}} = 1$ Sekunde sein.
Durch Variation der Cachegröße kann nun ein *Worst–Case* und ein *Best–Case* Speichermodell konstruiert werden. Für jedes Modell wird eine Tabelle angegeben, die $a$ und $m$ aufschlüsselt und $T_{\text{mult}}$ für die Matrizengröße $n \times n = 20 \times 20$ angibt.

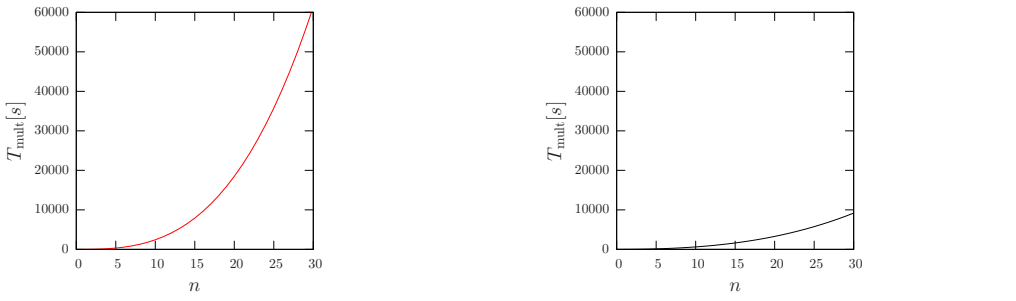
- Worst-Case: Der Cache wird nicht benutzt und deshalb muß der Prozessor vor jeder Multiplikation zwei Zahlen aus dem Arbeitsspeicher laden. In diesem Fall ist $a$ gleich $m$ und die Arbeitsgeschwindigkeit wird vom Arbeitsspeicher bestimmt.

| | $a$ | $m$ | $T_{\text{mult}} = 0.1\,[s]\,a + 1[s]\,m$ |
|---|---|---|---|
| Lesen aus $\boldsymbol{A}$ | $n^3$ | $n^3$ | |
| Lesen aus $\boldsymbol{B}$ | $n^3$ | $n^3$ | |
| Lesen aus $\boldsymbol{C}$ | $n^2$ | $n^2$ | |
| Schreiben in $\boldsymbol{C}$ | $n^2$ | $n^2$ | |
| $\sum$ | $2\,n^3 + 2\,n^2$ | $2\,n^3 + 2\,n^2$ | |
| $n \times n = 20 \times 20$ | 16800 | 16800 | 18480 |

- Best-Case: Der Cache ist so groß, dass alle Matrizen $\boldsymbol{A}$, $\boldsymbol{B}$ und $\boldsymbol{C}$ auf einmal in den Cache passen.

| | $a$ | $m$ | $T_{\text{mult}} = 0.1\,[s]\,a + 1[s]\,m$ |
|---|---|---|---|
| Lesen aus $\boldsymbol{A}$ | $n^3$ | $n^2$ | |
| Lesen aus $\boldsymbol{B}$ | $n^3$ | $n^2$ | |
| Lesen aus $\boldsymbol{C}$ | $n^2$ | $n^2$ | |
| Schreiben in $\boldsymbol{C}$ | $n^2$ | $n^2$ | |
| $\sum$ | $2\,n^3 + 2\,n^2$ | $4\,n^2$ | |
| $n \times n = 20 \times 20$ | 16800 | 1600 | 3280 |

Die Berechnugen dieser beiden Tabellen wurden nun für $n \in \{1, \ldots, 30\}$ durchgeführt und in Abbildung 68 dargestellt. Je größer die Matrizen werden, desto größer wird auch der Unterschied. In der Nutzung eines schnelleren Zwischenspeichers steckt daher ein sehr großes Potential.




(a) Worst-Case Modell (Ohne Cache) (b) Best-Case Modell (Optimale Cachnutzung)

Abbildung 68: Theoretische Ausführungszeit bei verschiedenen Dimensionen

Passen jedoch nicht alle drei Matrizen zusammen in den schnellen Cache, so empfiehlt es sich, die Matrizen in $N \times N$ Untermatrizen aufzuteilen, wie dies in Abbildung 69 gezeigt ist und die Blockgröße $n/N$ dabei so zu

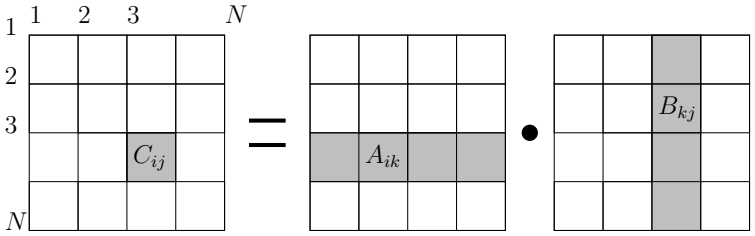



Abbildung 69: Aufteilung in Blockmatrizen

wählen, dass genau drei Untermatrizen gemeinsam in den Cache passen. Dadurch kann die Multiplikation der Untermatrizen mit optimaler Cacheausnutzung durchgeführt werden. Betrachtet man die $N \times N$ Blockmatrizen als Skalare, so wird der Algorithmus zur Matrizenmultiplikation mit dem *Worst–Case*–Speichermodell durchgeführt, die Multiplikation der einzelnen Blöcke findet jedoch mit dem *Best–Case*–Speichermodell statt. Durch Variation der Blockgröße kann man zwischen Speicherplatzverbrauch und guter Performance balancieren. Tabelle (8) zeigt die Anzahl der Speicherzugriffe in Abhängigkeit von der Anzahl der Blöcke $N$ pro Zeile. Für die Matrixdimension $n \times n = 20 \times 20$ und eine Einteilung in eine $N \times N = 4 \times 4$ Blockmatrix mit Blockgröße $n/N = 5$ ist die Anzahl der Speicherbewegungen und die Gesamtdauer für die Multiplikation dargestellt. Während im Best–Case Szenario eine Cachegröße von $3 \cdot (20 \times 20) = 1200$ Zahlen notwendig war, reicht für diese Version der Matrizenmultiplikation eine Cachegröße von $3 \cdot (5 \times 5) = 75$ Zahlen aus, weil nur drei $5 \times 5$ Blockmatrizen gleichzeitig im Speicher gehalten werden müssen. Trotzdem erhalten wir gegenüber dem Worst–Case Szenario eine Geschwindigkeitssteigerung von Faktor 3. Durch Veränderung der Blockgröße führen die Extremfälle $N = 1$

| | $a$ | $m$ | $T_{\text{mult}} = 0.1\,[s]\,a + 1[s]\,m$ |
|---|---|---|---|
| Lesen aus $\boldsymbol{A}$ | $n^3$ | $N^3(\frac{n}{N})^2 = Nn^2$ | |
| Lesen aus $\boldsymbol{B}$ | $n^3$ | $N^3(\frac{n}{N})^2 = Nn^2$ | |
| Lesen aus $\boldsymbol{C}$ | $n^2$ | $N^2(\frac{n}{N})^2 = n^2$ | |
| Schreiben in $\boldsymbol{C}$ | $n^2$ | $N^2(\frac{n}{N})^2 = n^2$ | |
| $\sum$ | $2\,n^3 + 2\,n^2$ | $2n^2(N+1)$ | |
| $n \times n = 20 \times 20$ und $N = 4$ | 16800 | 4800 | 6480 |

Tabelle 8: Block–Speichermodell

und $N = n$ auf die *Worst–Case*– und die *Best–Case*–Methoden.

$$2n^2(N+1) = \begin{cases} 4\,n^2 & \text{für } N = 1 \\ 2n^3 + 2n^2 & \text{für } N = n \end{cases} \tag{96}$$

Durch Abbildung 70 wird klar, dass das Einteilen in Blockmatrizen eine sehr effektive Technik ist, um schnelle Zwischenspeicher auszunutzen. Dargestellt sind der Best–Case, bei dem alle drei Matrizen in den Cache passen und der Worst–Case, bei dem der Cache unbenutzt bleibt. Die dazwischen liegenden Kurven zeigen Berechnungszeiten mit unterschiedlichen Blockgrößen von 5 bis 30. Bereits die kleine Blockgröße von 5 liegt deutlich näher an dem Best–Case, als am Worst–Case. Mit steigender Blockgröße konvergiert die Berechnungszeit gegen das Best–Case Szenario, jedoch erst sehr schnell und dann immer langsamer. Es gilt daher die Faustregel "Je größer die Blöcke sind, desto schneller, aber den größten Teil des Geschwindigkeitszuwachses erreicht man schon mit relativ kleinen Blockgrößen, d. h. mit wenig Cache.

Es wurde gezeigt, wie gut Matrizenmultiplikation sich optimieren läßt. Da sie eine Grundoperation der Linearen Algebra ist, können viele Operationen der Lineraren Algebra ebenfalls durch eine optimierte Funktion zur Matrizenmultiplikation beschleunigt werden. Der Schlüssel dazu liegt in der Verwendung von Blockalgorithmen, bei denen Untermatrizen wie Skalare behandelt werden und die Grundoperationen Addition und Multiplikation durch durch Matrixaddition und Matrixmultiplikation ersetzt werden. Matrixdivision kann als Multiplikation mit der inversen Matrix, oder als Abfolge von Vorwärts- und Rückwärtseinsetzen mit der nach Cholesky zerlegten Matrix ausgedrückt werden. Die Choleskyzerlegung selber entspricht dem Wurzelziehen bei Skalaren.

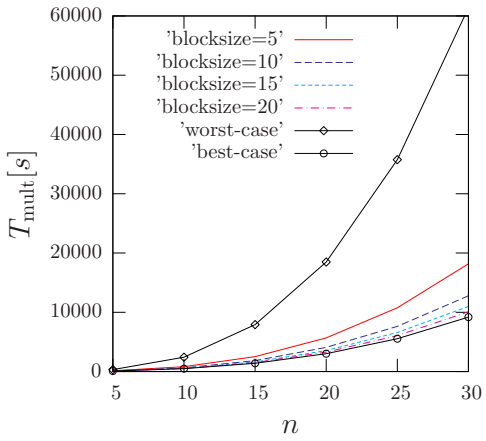



Abbildung 70: Ausführungszeit bei Verschiedenen Dimensionen und Blockgrößen

### A.1.2 BLAS

Weil die Bedeutung der Matrizenmultiplikation für die Lineare Algebra mit Computern so groß ist, wurden schon früh die *Basic Linear Algebra Subroutines (BLAS)* entwickelt (Lawson et al. 1979, Dongarra et al. 1988). Die BLAS enthalten Grundoperationen der Linearen Algebra, wie Vektor und Matrixmultiplikation und -Addition und haben sich als Standard für numerische Software etabliert. Sie sind in drei Abschnitte, die sogenannten *Level* eingeteilt. Diese Level können als ein Maß für das Optimierungspotential, das in den Grundoperationen steckt, interpretiert werden. Das höchste Optimierungspotential steckt in den Level 3 Routinen, weil sie Operationen enthalten, die zwei Matrizen als Operanden haben. Die Zugehörigkeit zu einem Level wird bei allen Routinen anhand der Operanden festgelegt. Grundsätzlich gilt, je niedriger der Level, desto weniger kann optimiert werden. Die Level lauten im einzelnen:

| | Operand 1 | Operand 2 |
|---|---|---|
| Level 1 | Vektor | Vektor |
| Level 2 | Vektor | Matrix |
| Level 3 | Matrix | Matrix |

Die BLAS haben immer dasselbe Interface, ihre Implementierung ist jedoch von System zu System unterschiedlich, weil die Hersteller von Workstations, wie SUN, SGI, oder HP sie speziell für ihre eigene Hardware handoptimieren. Ein Programm, das BLAS verwendet, kann auf jeder Art von Hardware optimal schnell sein, wenn die richtigen Bibliotheken beim Linken des Programmes verwendet werden. Es bedarf dazu keine Änderung am Quellcode. Die Vorteile der BLAS sind daher Performance und Portierbarkeit.

### A.1.3 LAPACK

Basierend auf den bestehenden Bibliotheken *LINPACK* und *EISPACK* wurde das Programpaket *LAPACK Linear Algebra Package* entwickelt (Anderson et al. 1999). LAPACK stellt ca. 300 Routinen für das Lösen von linearen Gleichungssystemen, Least-Squares Löser und Routinen für das Lösen von Eigenwert und Sinulärwertproblemen zur Verfügung. Seine Vorgänger LINPACK und EISPACK, die bereits in den 70er Jahren entwickelt wurden, basierten zwar ebenfalls schon auf den BLAS, sie benutzten jedoch nur Level 1/2 Routinen. LAPACK organisiert die Algorithmen, wenn möglich so um, dass Blockmatrix Operationen und somit Level 3 BLAS verwendet werden, um dem hierarchischen Speicheraufbau moderner Systeme Rechnung zu tragen. Da LAPACK Routinen wenn immer es möglich ist BLAS Level 3 aufrufen, wird durch optimierte BLAS Routinen auch die Leistung von LAPACK optimiert.

**Verwendung von BLAS und LAPACK** BLAS und LAPACK bieten die Vorteile Performance und Portierbarkeit für den Programmierer und den Anwender numerischer Software. Der Performancegewinn ist jedoch, wie Anfangs erwähnt, bei den BLAS Leveln unterschiedlich. Level 3 Operationen bieten einen Geschwindigkeitsvorteil von Faktor 10 und mehr, wohingegegen mit Level 2-1 nicht mehr als Faktor 2 zu erwarten ist (Whaley

et al. 2001, S. 8). Daher ist die oberste Direktive bei der Verwendung von BLAS, Blockalgorithmen zu formulieren, um Level 3 BLAS verwenden zu können. Bei der Implementierung von LAPACK ist diese Direktive bereits umgesetzt worden.

### A.1.4 ATLAS

Lange Zeit war es die Domäne der Workstation- und Supercomputerhersteller handoptimierte BLAS für ihre eigene Architektur anbieten zu können. Erst Ende der 90er Jahre wurden auch für Personalcomputer optimierte BLAS-Routinen verfügbar. Auf diesem Feld ist die Optimierung schwieriger, weil die Vielfalt der Prozessoren sehr hoch ist und deren Lebensdauer auf dem Markt sehr gering, so dass optimierte BLAS erst verfügbar wären, wenn der entsprechende Prozessor schon nicht mehr aktuell ist. Daher wurde von Whaley, Petitet und Dongarra (Whaley und Petitet 2005, Whaley et al. 2001) ein Projekt gegründet, das den Ansatz verfolgt BLAS nicht von Hand zu optimieren, sondern diese Aufgabe einer Software zu überlassen, die durch systematisches Ausprobieren verschiedener Systemparameter wie Cachegröße, Blockgröße usw. eine optimale Implementierung automatisch generiert. Diesen Ansatz bezeichneten sie mit *Automated Empirical Optimization of Software (AEOS)*. Der Grundgedanke dieses Ansatzes ist, mit der rasanten Prozessorentwicklung nach dem Moore'schen Gesetz mithalten zu können, damit auch die Leistungsfähigkeit neuer Prozessoren voll ausgeschöpft werden kann, bevor sie veraltet sind. Das Projekt, das sich speziell mit der Optimierung der BLAS beschäftigt heißt *Automatically Tuned Linear Algebra Software (ATLAS)* (Whaley et al. 2001, Project 2006). Die Beschleunigung der BLAS durch hinzulinken der ATLAS Bibliothek liegt im Bereich der handoptimierten Hersteller-BLAS, manchmal sogar darüber. ATLAS ist Open-Source und unter einer BSD Lizenz verfügbar (Project 2006).

### A.1.5 ATLAS-Alternativen

**Math Kernel Library** Seit einiger Zeit liefert Intel zu dem hauseigenen Compiler die *Math Kernel Library* [`www.intel.com`]. Sie ist jedoch auf bestimmte Architekturen beschränkt und ist für kommerziellen Gebrauch nicht frei verfügbar.

**Goto BLAS** Der Name Goto BLAS stammt von ihrem Entwickler Kazushige Goto, der diese Routinen an der Universität von Texas [`www.tacc.utexas.edu/~goto`] entwickelt. Diese Bibliothek verfolgt einen etwas anderen Ansatz, als zum Beispiel ATLAS und muß daher auf jede Architektur von Hand angepasst werden. Zur Zeit gelten die Goto BLAS jedoch als die schnellste BLAS Bibliothek. Sie sind zwar frei verfügbar, jedoch nicht auf Quellcodeebene.

### A.1.6 Wie schnell kann ein Computer rechnen?

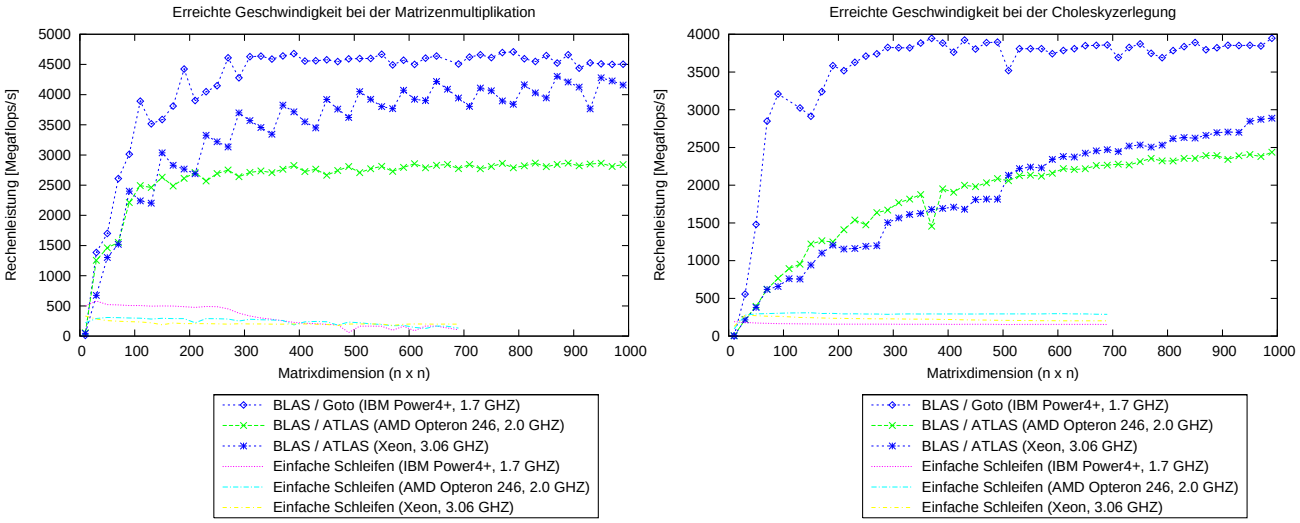
Im vorangegangenen Kapitel haben wir gesehen, dass die Arbeitsgeschwindigkeit von Computern maßgeblich durch die Optimierung von Speicherzugriffen verbessert werden kann. Wir haben auch gesehen, dass es durch Verwendung geeigneter Optimierungsbibliotheken wie z. B. ATLAS leicht ist, von dieser Technik zu profitieren. In diesem Kapitel soll nun anhand der Matrizenmultiplikation und der Choleskyzerlegung gezeigt werden, wie hoch der Profit durch diese Bibliotheken auf verschiedenen Architekturen in der Praxis ist.
Dazu wird die Geschwindigkeit in Megaflops (Millionen Fließkommaoperationen) pro Sekunde gemessen, die die Computer bei dieser Aufgabenstellung erreicht haben. Diese Erkenntnisse werden anschließend benutzt, um vorauszusagen, wie lange ein Computer für eine gegebene Problemstellung brauchen wird.

### A.1.7 Benchmark

Um die erreichte Rechengeschwindigkeit zu ermitteln, wurde durch den Autor das Programm `benchmark` erstellt, das sowohl die Choleskyzerlegung, als auch die Matrizenmultiplikation an Beispielmatrizen durchführt und die Zeit misst. Im Fall der Matrizenmultiplikation wurden die Zufallsmatrizen $\boldsymbol{A}$, $\boldsymbol{B}$ und $\boldsymbol{C} \in \mathbb{R}^{n \times n}$ erzeugt und mit ihnen die Operation $\boldsymbol{C} = \boldsymbol{C} + \boldsymbol{AB}$ jeweils einmal mit Algorithmus A.1 und einmal mit der optimierten BLAS Routine `dgemm` durchgeführt. Für die Choleskyzerlegung wurde eine symmetrische positiv definite Matrix $\boldsymbol{N} \in \mathbb{R}^{n \times n}$ erzeugt und am Platz nach Cholesky zerlegt. Dazu wurde jeweils einmal Algorithmus 4.1 und einmal die optimierte LAPACK Routine `dpotrf` benutzt. Für beide Testreihen wurde $n$ in 20er Stufen von 10 bis 990 erhöht. Dieser Test wurde auf drei ausgesuchten Systemen mit folgenden Prozessoren durchgeführt

- PC mit Intel Xeon 32 Bit-CPU mit 3.06 GHZ und 4 GB RAM

- PC mit AMD Opteron 246 64 Bit-CPU mit 2.0 GHZ 8 GB RAM
- IBM Power 4+ 64 Bit-CPU mit 1.7 GHZ mit 128 GB RAM
  Dieser Prozessor ist einer der 1312 Prozessoren des Supercomputers JUMP (JUMP 2006) des John von Neumann-Institus für Computing (NIC) am Forschungszentrum Jülich (NIC 2006).




(a) Algorithmus A.1 gegen optimierte BLAS Routine `dgemm`

(b) Algorithmus 4.1 gegen optimierte LAPACK Routine `dpotrf`

Abbildung 71: Erreichte Rechengeschwindigkeiten auf ausgesuchten Prozessoren mit optimierten und nicht optimierten Algorithmen

Die Ergebnisse zeigt Abbildung 71. Die Berechnungen mit den Algorithmen A.1 bzw. 4.1 werden in der Legende mit "einfache Schleifen" bezeichnet. Festzuhalten ist, dass die Blockgröße eine Mindestgrenze von ca $300 \times 300$ überschreiten muss, damit die optimierten Bibliotheken ihre Leistung voll entfalten können. Darüber bleibt die Performance dann auf einem gleichbleibenden Niveau. Im Bereich der Blockgröße zwischen ca. 50 und ca. 300 wird durch die optimierten Versionen der BLAS/LAPACK Routinen eine Leistungsexplosion erreicht. Die Beschleunigung ist dann je nach Computer / Bibliothek 10 bis 20-fach gegenüber den einfachen Schleifen.

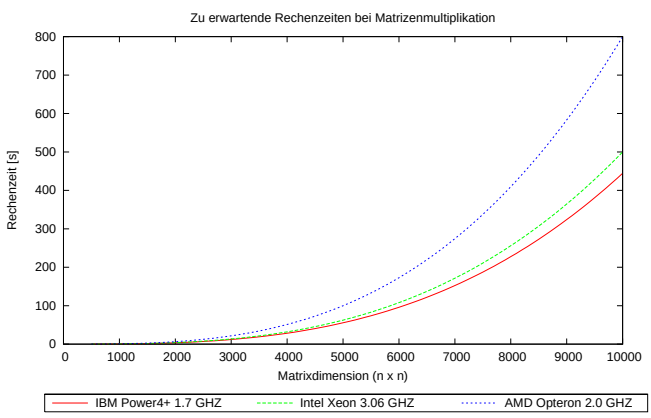
**Rechenzeitabschätzung** Mit Hilfe des Programms `benchmark` konnte die tatsächlich erreichte Rechengeschwindigkeit auf verschiedenen Prozessoren ermittelt werden. Um ein Gefühl dafür zu vermitteln, wie lange es bei gegebenen Matrizengrößen dauern wird, eine Multiplikation durchzuführen, dient die Abbildung 72. Sie soll helfen die Angabe von Rechengeschwindigkeit interpretierbar zu machen. Ermittelt wurden folgende Geschwindigkeiten für Matrizenmultiplikation durch Abgreifen aus Abbildung 71a.

- IBM Power 4+: 4500 MFlops/s
- Intel Xeon 3.06 GHZ: 4000 MFlops/s
- AMD Opteron 2.0 GHZ: 2500 MFlops/s

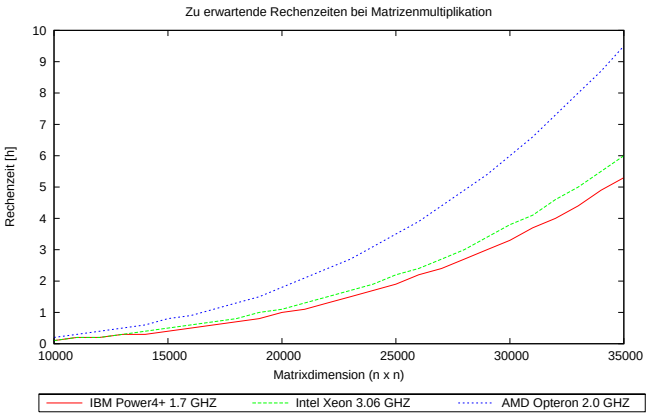
Abbildung 72 gibt für diese drei Rechengeschwindigkeiten die Rechenzeit für Matrizendimensionen $n \times n$ mit $n \in \{1000, 2000, \ldots, 10000\}$ an, dabei wurde zwischen kleinen Matrizen, deren Berechnung im Bereich Sekunden liegt und großen Matrizen, deren Berechnung im Bereich Stunden liegt, unterschieden.

## A.2 Paralleles Rechnen

Erst, wenn die numerische Software so weit optimiert ist, dass weitere Verbesserungen mit viel Aufwand erarbeitet werden müssten, sollte man darüber nachdenken, eine weitere Geschwindigkeitssteigerung durch paralleles Rechnen anzustreben. Wie wir im vorangegangenen Kapitel gesehen haben, ist bei numerischer Software, die viele Operationen mit Matrizen macht, durch Auswahl geeigneter Bibliotheken eine Geschwindigkeitssteigerung von Faktor 10 realistisch, so dass der erhebliche Mehraufwand durch Parallelisierung wohlüberlegt sein will. Ist die Laufzeit eines Programms jedoch so groß, dass Parallelisierung unumgänglich ist, so kann man unter verschiedenen Möglichkeiten auswählen. Die gängigsten Parallelisierungsverfahren sind:




(a) Kleine Matrizen unter $10000 \times 10000$




(b) Große Matrizen über $10000 \times 10000$

Abbildung 72: Zu erwartende Rechenzeiten für Matrizenmultiplikation auf ausgesuchten Prozessoren

- *Threads*
- Halbautomatische Parallelisierung mit *OpenMP*
- *Message Passing Interface MPI*

Die parallele Programmierung mit Threads ist sehr Betriebssystemnah und daher wenig portabel. Weiterhin ist sie nicht speziell für Numerik ausgelegt, sondern eine allgemeine Programmierschnittstelle. Mit Threads kann nur auf Computern parallelisiert werden, die einen gemeinsamen Hauptspeicher für mehrere Prozessoren haben, sogenannte *shared memory* Computer. Diese sind sehr teuer und der Anzahl der Prozessoren sind ökonomische Grenzen gesetzt.
Das letztgesagte gilt auch für die Parallelisierung mit OpenMP. Bei OpenMP fügt der Programmierer Präcompileranweisungen in den Quelltext ein um dem Compiler mitzuteilen, welche Schleifen er parallelisieren soll. Die Entscheidung, welche Teile des Programms zu parallelisieren sind, liegt daher in der Hand des Programmieres, die eigentliche Parallelisierung wird jedoch vom Compiler durchgeführt, daher der Begriff halbautomatische Parallelisierung. OpenMP funktioniert nur bei Schleifen über große Arrays, die sich leicht parallelisieren lassen. Der zu erwartende Geschwindigkeitsvorteil hängt start von der Komplexität der Schleifen ab und ist nur bei einfachen Konstrukten effektiv. OpenMp ist daher nur für die Parallelisierung von bestehender Software zu empfehlen.

### A.2.1 MPI

Einen anderen Ansatz verfolgt das Message Passing Interface MPI. MPI ist ein Programmierstandard, der Routinen definiert, die zur Kommunikation zwischen Prozessen dienen, wobei die Definition eines Prozesses ein Programm ist, dass gerade ausgeführt ist. Wird dasselbe Program zweimal ausgeführt, so spricht man von zwei Prozessen. MPI Programme sind portabel, weil sie betriebssystemspezifische Belange vor dem Benutzer verbergen. Das Prinzip von MPI soll nun anhand der Abbildung 73 erklärt werden. Dargestellt ist die parallele Ausführung des Programmes `pcgma` auf zwei Computern, von denen einer zwei Prozessoren hat und einer nur einen. Es werden vier Prozesse gestartet und zwar zwei auf jedem Computer. Jeder dieser Prozesse besitzt seinen eigenen Adressraum. Das bedeutet die Variable $x$ ist zwar in allen Prozessen vorhanden, aber eine Änderung ihres Inhaltes auf einem Prozess wird auf den anderen Prozessen nicht bemerkt. Sollen die Variablen $x$ auf allen Prozessen synchronisiert werden, so geschieht dies über expliziten Nachrichtenaustausch. Dabei wird unterschieden zwischen kollektiver Kommunikation, bei der an alle Prozesse einer Gruppe Nachrichten geschickt werden und Punkt-zu-Punkt Kommunikation, bei der Sender und der Empfänger jeweils ein einzelner Prozess ist. Um die Prozesse eindeutig zu identifizieren, werden sie in Gruppen eingeteilt und innerhalb dieser durchnummeriert. Diese Nummern werden auch als MPI Rang bezeichnet. MPI definiert Funktionen mit denen Nachrichten an alle Prozesse, oder mit Hilfe des Ranges an einzelne Prozesse geschickt werden können. Bei einer Punkt-zu-Punkt Verbindung muß dem Aufruf einer Sendefunktion eine Empfangsfunktion gegenüber stehen.
Aus der Sicht des Programmieres geschieht die Adressierung anderer Prozesse nur über ihren Rang. Es spielt dabei keine Rolle, auf welcher Maschine der andere Prozess läuft. Die tatsächliche Kommunikation geschieht

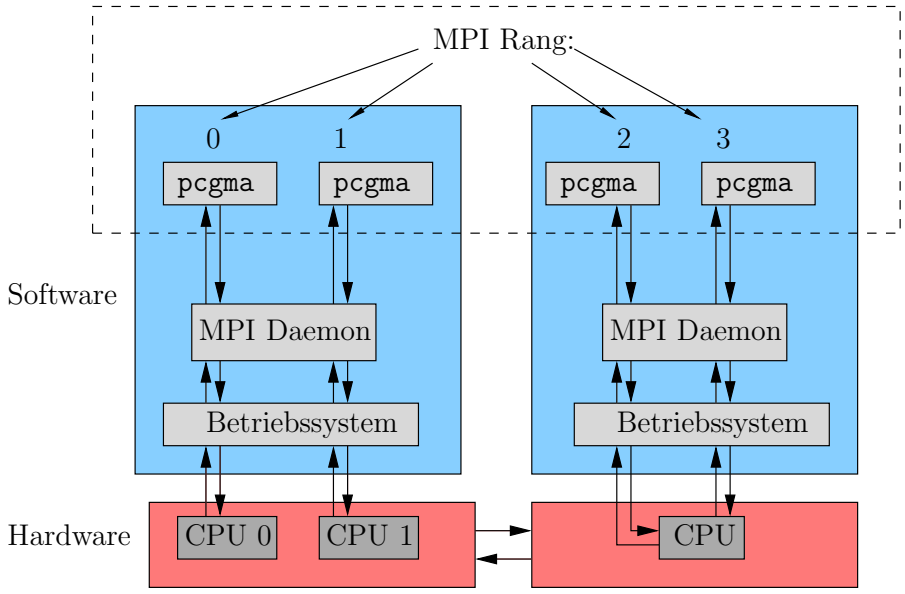



Abbildung 73: Funktionsweise von MPI

über den MPI-Daemon, von dem auf jeder Maschine eine Instanz läuft. Mit Daemon bezeichnet man ein Dienstprogramm das ohne Benutzerinteraktion im Hintergrund läuft. Diese Daemonen kommunizieren mit dem Betriebssystem und veranlassen das Senden der Nachrichten an andere Prozesse. Diese Vorgänge sind für den Programmierer jedoch völlig unsichtbar. In Abbildung 73 liegt nur der gestrichelte Bereich in der Hand des Programmierers. Alles darunter liegende wird von den MPI-Daemonen erledigt, es spielt daher für die Programmierung mit MPI keine Rolle, ob ein Prozessor auf einer entfernten Maschine läuft, die über ein Netzwerk verbunden ist, oder ob es nur ein lokaler Prozess auf der gleichen CPU ist.
Einige Beispielhafte MPI Funktionen sind

```
1  int MPI_Send(void *buf, int count, MPI_Datatype dtype, int dest, int tag, MPI_Comm comm)
2
3  EINGABE PARAMETER
4    buf    - Startadresse des Sendepuffers
5    count  - Anzahl der Elemente im Sendepuffer
6    dtyp   - Datentyp der Elemente im Sendepuffer
7    dest   - MPI Rang des Zielprozesses
8    tag    - Nachrichtennummer
9    comm   - Prozessgruppe
```

Sendet die durch Menge, Speicherort und Typ spezifizierten Daten an den Prozess mit dem angegebenen Zielrang innerhalb der angegebenen Prozessgruppe

```
1  int MPI_Recv(void *buf, int count, MPI_Datatype dtype, int src, int tag, MPI_Comm comm, MPI_Status *stat)
2
3  EINGABE PARAMETER
4    count  - Maximale Anzahl von Elementen im Empfangspuffer
5    dtype  - Datentyp der Elemente im Empfangspuffer
6    src    - MPI Rang des sendenten Prozesses
7    tag    - Nachrichtennummer
8    comm   - Prozessgruppe
```

Empfängt die von `MPI_Send()` gesendeten Daten, vom Prozess mit dem MPI Rang src und speichert den Fehlerstatus. Einen guten Einstieg in die Programmierung mit MPI bietet das Buch (Gropp et al. 1994), aber auch auf der Homepage des LAM/MPI Projektes findet man viele Informationen und Tutorials (LAM/MPI 2006).

## A.3 Fazit

Es wurde gezeigt, dass es bei Problemen, die viel Rechenzeit erfordern, unumgänglich ist, Programmbibliotheken zu verwenden, die speziell auf die Harware abgestimmt sind, denn diese Bibliotheken bieten bei Matrizenmultiplikation einen 10-20 fachen Geschwindigkeitsvorteil. Daher ist die Verwendung von Blockalgorithmen sehr wichtig bei der Implementierung von numerischen Algorithmen. Weiterhin wurde nachgewiesen, dass optimale Leistung auf verschiedenen Architekturen ab einer Blockgröße von ca. $300 \times 300$ erreicht wird. Die Verwendung von BLAS / ATLAS Routinen ist in numerischer Software zu empfehlen, weil dadurch leicht optimierte Bibiliotheken verwendet werden können. Die erstellte Software ist so ohne Änderungen am Quellcode auf andere Architekturen portierbar. Erst wenn die Hardwareoptimierung mit den vorgestellten Methoden und Bibliotheken durchgeführt wurde, sollte Parallelisierung in Betracht gezogen werden. MPI ist für numerische Programmierung sehr gut geeignet, weil es ermöglicht parallele, portable Programme zu erstellen, und dabei keine Betriebssystemspezifischen Kenntnisse von dem Programmierer erwartet.

# Literatur


Alkhatib, H. und W.-D. Schuh (2006) Integration of the Monte Carlo Covariance Estimation Strategy into Tailored Solution Procedures for Large-scaled Least Squares Problems. *J. Geodesy*, Issue: Online First.

Anderson, E., Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney und D. Sorensen (1999) *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 3. Ausgabe.

Auzinger, T. (1997) Numerical Investigations on High Degree Spherical Harmonic Analysis. Diplomarbeit, TU Graz.

Auzinger, T. und W.-D. Schuh (1998) High-degree spherical harmonic analysis combining gridded and random distributed data sets. *Phys. Chem. Earth*, 23:19–23.

Balmino, G. (1993) The spectra of the topography of the earth, venus and mars. *Geophys. Res. Lett.*, 20:1063–1066.

Bernhard Hoffmann-Wellenhof und Helmut Moritz (2005) *Physical Geodesy*. Springer.

Bosch, W. (1993) A rigorous least squares combination of low and high degree spherical harmonics. *Presented Paper on "IAG General Meeting"*, Beijing 1993.

Boxhammer, Ch. und W.-D. Schuh (2006) GOCE Gravity Field Modeling: Computational Aspects - Free Kite Numbering Scheme. Rummel, R., Ch. Reigber, M. Rothacher, G. Boedecker, U. Schreiber und J. Flury (Hrsg.), *Observation of the Earth System from Space*, Springer, Berlin - Heidelberg, 209–224.

Colombo, O.L. (1981) *Numerical methods for harmonic analysis on the sphere*. Reports of the Department of Geodetic Science. Ohio State University (OSU), Ohio. No. 310.

Dongarra, Jack J., Jeremy Du Croz, Sven Hammarling und Richard J. Hanson (1988) An Extended Set of FORTRAN Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 14(1):1–17.

ESA (1999) *The four candidate earth explorer core missions - gravity field and steady-state ocean circulation mission*. ESA Report SP-1233(1), Granada.

ESA (2002) From Eötvös to mGal+. Technischer Bericht, ESA-Project, ESA/ESTEC Contract No. 14287/00/NL/DC.

G.H. Golub und Charles F. Van Loan (1996) *Matrix Computations*. The Johns Hopkins University Press, third. Ausgabe.

Gropp, William, Ewing Lusk und Anthony Skjellum (1994) *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge, MA.

Gundlich, B., K.R. Koch und J. Kusche (2003) Gibbs sampler for computing and propagating large covariance matrices. *J. Geodesy*, 77:514–528.

Hanson, R.H. (1978) A posteriori error propagation. *Proceedings of the "2nd International Symposium on Problems Related to the Redefinition of North American Geodetic Networks"*, Arlington, Virginia (April 24-28, 1978), 427–445.

JUMP. (2006) , (JU)elich (M)ulti(P)rozessor (JUMP) Dokumentation. `http://jumpdoc.fz-juelich.de/`, 2006. Stand: 2006.

Kaula, W.M. (1966) *Theory of Satellite Geodesy*. Blaisdell Publ. Comp., Massachusetts - Toronto - London.

Koch, K.-R., J. Kusche, C. Boxhammer und B. Gundlich (2004) Parallel Gibbs Sampling for Computing and Propagating Large Covariance Matrices. *ZfV*, 129:32–427.

Koch, K.R. (1999) *Parameter Estimation and Hypothesis Testing in Linear Models*. 2nd Ed. Springer, Berlin - Heidelberg - New York.

Koch, K.R. (2005) Determining the Maximum Degree of Harmonic Coefficients in Geopotential Models by Monte Carlo Methods. *Studia Geophysica et Geodaetica*, 49:259–275.

LAM/MPI. (2006) , LAM/MPI Homepage, Parallel Computing. `http://www.lam-mpi.org`, 2006. Stand: 2006.

Lawson, C. L., R. J. Hanson, D. R. Kincaid und F. T. Krogh (1979) Algorithm 539: Basic Linear Algebra Subprograms for Fortran Usage [F1]. *ACM Transactions on Mathematical Software*, 5(3):324–325.

Lemoine, F.G., D.E. Smith, L. Kunz, R. Smith, E.C. Pavlis, N.K. Pavlis, S.M. Klosko, D.S. Chinn, M.H. Torrence, R.G. Williamson, C.M. Cox, K.E. Rachlin, Y.M. Wang, R.H. Rapp und R.S. Nerem (1996) The development of the NASA GSFC and NIMA joint geopotential model. *Proceedings of the "International Symposium on Gravity, Geoid, and Marine Geodesy"*, Tokyo, Japan.

Mayer-Gürr, T., K.H. Ilk, A. Eicker und M. Feuchtinger (2005) ITG-CHAMP01: A CHAMP Gravity Field Model from short kinematical arcs of a one-year observation period. *J. Geodesy*, 78:462–480.

NIC. (2006) , John von Neumann-Institut für Computing. `http://www.fz-juelich.de/nic/`, 2006. Stand:2006.

Pail, R., W-D. Schuh und Th. Wermuth (2005) GOCE Gravity Field Processing. In: C. Jekeli, L. Bastos, J. Fernandes (Hrsg.), *Gravity, geoid and space missions*, Band 129 der Reihe International Association of Geodesy symposia, 36–41.

Plank, G. (2002) Implementation of the pcgma-package on massive parallel systems. In: Sünkel, H. (Hrsg.), *ESA-Project "From Eötvös to mGal", Final-Report.* ESA/ESTEC Contract No. 14287/00/NL/DC, 183–216.

Plank, Gernot (2004) *Numerical solution strategies for the GOCE mission by using cluster technologies.* Dissertation, Institut für Navigation und Satellitengeodäsie der Technischen Universität Graz.

Project, ATLAS. (2006) . `http://math-atlas.sourceforge.net/\verb`, 2006. Stand: 2006.

Rapp, R.H., Y. Wang und N. Pavlis (1991) *The Ohio state 1991 geopotential and sea surface topography harmonic coefficient models.* Reports of the Department of Geodetic Science. Ohio State University (OSU), Ohio. No. 410.

Reigber, C., G. Balmino, P. Schwintzer, R. Biancale, A. Bode, J.-M. Lemoine, R. König, S. Loyer, H. Neumayer, J.-C. Marty, F. Barthelmes, F. Perosanz und S. Y. Zhu (2002) A high-quality global gravity field model from CHAMP GPS tracking data and accelerometry (EIGEN-1S). *Geophysical Rearch Letters*, 29:37–1.

Reigber, C., P. Schwintzer, K.-H. Neumayer, F. Barthelmes, R. König, C. Förste, G. Balmino, R. Biancale, J.-M. Lemoine, S. Loyer, S. Bruinsma, F. Perosanz und T. Fayard (2003) The CHAMP-only earth gravity field model EIGEN-2. *Advances in Space Research*, 31:1883–1888.

Reigber, C., H. Lühr, P. Schwintzer und J. Wickert (Hrsg.) (2004) *Earth Observation with CHAMP: Results from Three years in Orbit.* Springer, Berlin-Heidelberg.

Rummel, R. (1985) Satellitengradiometrie. *ZfV*, 6:242–257.

Rummel, R., F. Sansò, M. van Gelderen, M. Brovelli, R. Koop, F. Migliaccio, E. Schrama und F. Scerdote (1993) *Spherical harmonic analysis of satellite gradiometry.* Netherlands Geodetic Commission, New Series, 39.

Rummel, R., R. Koop und Th. Gruber (2004) High Level Processing Facility for GOCE: Products and Processing Strategy. In: *Proceedings of Second International GOCE User Workshop "GOCE, The Geoid and Oceanography".* ESA-ESRIN, Frascati.

Schuh, W.-D. (1996) *Tailored numerical solution strategies for the global determination of the earth's gravity field.* Mitteilungen der Geodätischen Institute der TU, Graz. Folge 81.

Schuh, W.-D. (2000) *Numerische Verfahren zur Geodätischen Optimierung.* TU Graz, 2000.

Schuh, W.-D. (2003) The processing of band-limited measurements; filtering techniques in the least squares context and in the presence of data gaps. Beutler, G., M.R. Drinkwater, R. Rummel und R. von Steiger (Hrsg.), *Earth Gravity Field from Space - From Sensors to Earth Sciences*, Band 108, Space Science Reviews, 67–78. ISSI Workshop, Bern (March 11-15,2002).

Schwarz, H.R. (1970) Die Methode der konjugierten Gradienten in der Ausgleichsrechnung. *ZfV*, 95:130–140.

Sneeuw, N. (2000) *A semi-analytical approach to gravity field analysis from satellite observations.* Reihe C, 527. Deutsche Geodätische Kommission, München.

Tapley, B., J. Ries, S. Bettadpur, D. Chambers, M. Cheng, F. Condi, B. Gunter, Z. Kang, P. Nagel, R. Pastor, T. Pekker, S. Poole und F. Wang (2005) GGM02 - An improved Earth gravity field model from GRACE. *J. Geodesy*, 79:467–478.

Whaley, R. Clint (2004) *Automated Empirical Optimization of High Performance Floating Point Kernels.* Dissertation, Florida State University, College of Art & Sciences, `http://www.cs.utsa.edu/ whaley/papers/diss.ps` (Stand 2006).

Whaley, R. Clint und Antoine Petitet (2005) Minimizing development and maintenance costs in supporting persistently optimized BLAS. *Software: Practice and Experience*, 35(2):101–121. `http://www.cs.utsa.edu/~whaley/papers/spercw04.ps`.

Whaley, R. Clint, Antoine Petitet und Jack J. Dongarra (2001) Automated Empirical Optimization of Software and the ATLAS Project. *Parallel Computing*, 27(1–2):3–35. Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 (`www.netlib.org/lapack/lawns/lawn147.ps`).

# Danksagung

Diese Arbeit entstand während meiner Tätigkeit am Institut für Theoretische Geodäsie der Rheinischen Friedrich–Wilhelms–Universität Bonn.
Ich möchte mich bei Herrn Prof. Dr. techn. W.–D. Schuh für die engagierte Betreuung und das Vertrauen bedanken, das er mir während dieser Zeit entgegengebracht hat. Durch die Freiheit, die umfangreiche technische Ausstattung, die Voraussetzung für meine Arbeit war, selbst bestimmen und aufbauen zu dürfen, gab er mir die Gelegenheit, wertvolle Erfahrungen zu sammeln, die diese Arbeit erst ermöglicht haben.
Ebenso möchte ich mich bei Herrn Prof. Dr.–Ing., Dr.–Ing. E.h. mult. K. R. Koch (em.) für die Übernahme des Koreferates und die stetige Hilfsbereitschaft bedanken.
Für die Übernahme des zweiten Koreferates bin ich Herrn Prof. Dr.–Ing. K. H. Ilk zu großem Dank verpflichtet.
Nicht zuletzt danke ich meiner Frau Sonja für das mehrmalige Korrekturlesen der gesamten Arbeit und das Verständnis für den hohen Zeitaufwand meiner wissenschaftlichen Arbeit.
Die Berechnungen wurden durch die Bereitstellung von Rechenzeit durch das John von Neumann–Institut für Computing (NIC) des Forschungszentrums Jülich innerhalb des Projekts 1827 ermöglicht.