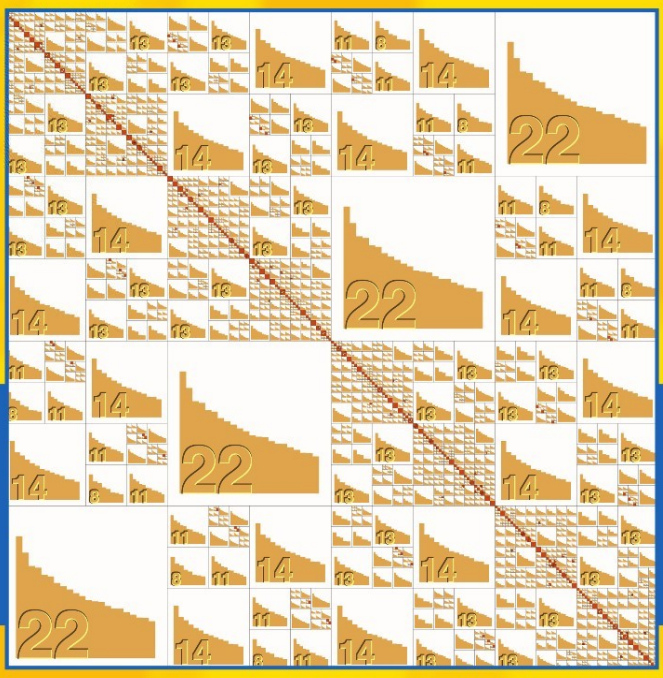
Wolfgang Hackbusch



Hierarchische Matrizen

Algorithmen und Analysis

Springer

Hierarchische Matrizen

W. Hackbusch

Hierarchische Matrizen

Algorithmen und Analysis

Springer

Wolfgang Hackbusch
Max-Planck-Institut für Mathematik in den
Naturwissenschaften
Abteilung Wissenschaftliches
Rechnen
Inselstr. 22-26
04103 Leipzig
Deutschland
wh@mis.mpg.de

ISBN 978-3-642-00221-2 e-ISBN 978-3-642-00222-9
DOI 10.1007/978-3-642-00222-9
Springer Dordrecht Heidelberg London New York

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Mathematics Subject Classification (2000): 65F05, 65F30, 65F50, 65F10, 15A09, 15A24, 15A99, 15A18, 47A56, 68P05, 65N22, 65N38, 65R20, 45B05

Einbandentwurf: WMXDesign GmbH, Heidelberg

Printed on acid-free paper

Springer ist Teil der Fachverlagsgruppe Springer Science+Business Media (www.springer.com)

Meiner Frau Ingrid gewidmet

Vorwort

Operationen mit großen Matrizen werden in der Numerik weitgehend vermieden. Stattdessen wird in der Regel versucht, alle Algorithmen auf Matrix-*Vektor*-Multiplikationen zurückzuführen. Der Grund ist der hohe Aufwand von z.B. $\mathcal{O}(n^3)$ Gleitkommaoperationen für eine Multiplikation von $n \times n$-Matrizen. Beginnend mit dem Strassen-Algorithmus wurde versucht, den Aufwand auf $\mathcal{O}(n^\gamma)$ mit $\gamma < 3$ zu reduzieren. Diese Bemühungen können aber nicht zum Ziel führen, da die theoretische untere Schranke $\gamma \geq 2$ besteht, und selbst quadratischer Aufwand bei großskaligen Matrizen inakzeptabel ist.

Dass die Matrixoperationen im hier beschriebenen $\mathcal{H}$-Matrix-Format trotzdem mit fast linearem Aufwand $\mathcal{O}(n \log^* n)$ ausführbar sind, ist kein Widerspruch zu der vorherigen Aussage, da oben die exakte Berechnung unterstellt wird, während die $\mathcal{H}$-Matrix-Operationen Approximationen enthalten. Die Approximationsfehler sind jedoch akzeptabel, da großskalige Matrizen von Diskretisierungen stammen, die ohnehin Diskretisierungsfehler enthalten. Die mit der $\mathcal{H}$-Matrix-Technik ermöglichten Operationen sind nicht nur die Matrixaddition und -multiplikation, sondern auch die Matrixinversion und die LU- oder Cholesky-Zerlegung.

Verwendet man die $\mathcal{H}$-Matrix-Technik zur Lösung von linearen Gleichungssystemen, so nimmt sie eine Stellung zwischen direkten Verfahren und Iterationsverfahren ein. Auf der einen Seite kann die approximative Inverse oder LU-Zerlegung mit wählbarer Genauigkeit bestimmt und damit das Gleichungssystem gelöst werden. Auf der anderen Seite reicht eine Inverse oder LU-Zerlegung mit mäßiger Genauigkeit, um eine schnelle Iteration zu konstruieren.

Hat man die Matrixoperationen zur Verfügung, lässt sich eine wesentlich größere Problemklasse behandeln, als dass mit der Beschränkung auf Matrix-Vektor-Multiplikationen möglich ist. Hierzu gehören die Berechnung von matrixwertigen Funktionen, zum Beispiel der Matrix-Exponentialfunktion, und die Lösung von Matrixgleichungen, etwa der Riccati-Gleichung.

Die approximative Durchführung der Operationen kann nur erfolgreich sein, wenn der Aufwand für eine Genauigkeit ε nur schwach mit $\varepsilon \to 0$ wächst.

Für die Inversen von Diskretisierungsmatrizen elliptischer Randwertprobleme und für zugehörige Randintegralgleichungen lässt zeigen, dass der Aufwand nur logarithmisch von ε abhängt. Für allgemeine, große Matrizen ist diese Aussage falsch, d.h. die $\mathcal{H}$-Matrix-Technik ist nicht für allgemeine Matrizen anwendbar. Trotzdem zeigen numerische Tests ein sehr robustes Verhalten. Für die praktische Durchführung ist es zudem sehr wichtig, dass die $\mathcal{H}$-Matrix-Technik in hohem Maße blackbox-artig ist.

Die Tatsache, dass nicht alle Matrizen (mit verbessertem Aufwand) dargestellt werden können, entspricht nicht dem klassischen Verständnis der Linearen Algebra. Dort ist man es gewohnt, dass Verfahren möglichst auf sämtliche Elemente eines endlichdimensionalen Vektorraumes anwendbar sind. Andererseits ist der Approximationsgedanke der Ursprung der Analysis. Da die Objekte wie zum Beispiel Funktionen im Allgemeinen unendlich viele Daten enthalten, verwendet man seit Anbeginn der Analysis Approximation aufgrund von partiellen Informationen (Beispiel Interpolation), wobei man in Kauf nimmt, dass die Approximation nur unter geeigneten Glattheitsvoraussetzungen möglich ist.

Die $\mathcal{H}$-Matrix-Technik beruht auf drei verschiedenen Komponenten. Die *erste, analytische Komponente* ist die lokale, separable Approximation der Greenschen Funktion beziehungsweise der Integralkernfunktion. Derartige Techniken sind in der Vergangenheit in verschiedenen Versionen bei der Behandlung diskreter Integraloperatoren angewandt worden: bei der Paneel-Clusterungstechnik, bei Multipolentwicklungen, aber auch bei der Matrixkompression von Wavelet-Matrizen. Nur mit Hilfe dieser Techniken kann die Matrix-Vektor-Multiplikation mit der an sich vollen Matrix in fast linearem Aufwand durchgeführt werden. Die *zweite Komponente* gehört der linearen Algebra an. Techniken der Singulärwertzerlegung und der QR-Zerlegung spielen eine wichtige Rolle bei der Organisation der lokalen Matrixdaten. Die *dritte Komponente* betrifft die diskreten Strukturen. Die ersten beiden Komponenten werden auf Teilmatrizen angewandt. Die geeignete Partition der Matrix in Teilmatrizen der richtigen Größe ist für die Matrixoperationen der entscheidende Schritt. Die diskreten Strukturen stellen sich in Form vom Bäumen, den Cluster- und Blockclusterbäumen dar.

Ziel dieser Monographie ist die umfassende Einführung in die Technik hierarchischer Matrizen. Da diese insbesondere für die großskaligen Matrizen aus dem Umfeld von Randwertaufgaben entwickelt worden sind, muss auch knapp auf die Diskretisierung von Randwertaufgaben und auf die Randintegralmethode eingegangen werden. Den oben aufgezählten, sehr unterschiedlichen Komponenten entsprechend widmen sich die ersten Kapitel den verschiedenen Fragen zur Analysis, zur Linearen Algebra und den Strukturen, die dann die Grundlage der Algorithmen bilden. Um die Kapitel einerseits nicht zu umfangreich werden zu lassen und andererseits hinreichend Hintergrundmaterial bereitzustellen, enthält das Buch fünf Anhänge. Hier findet der Leser den benötigten Hintergrund zu Themen aus dem Hauptteil.

Das letzte Kapitel gibt einen kleinen Einstieg in die Behandlung großskaliger Tensoren. Hier werden nur Anwendungen angesprochen, die direkt mit hierarchischen Matrizen verbunden sind. Andererseits besteht die herausfordernde Aufgabe, die im Umfeld der hierarchischen Matrizen entwickelten Techniken auf Tensoren zu übertragen. Da der Speicher- und Rechenaufwand bei großskaligen Tensoren den Anwender vor noch größere Schwierigkeiten als bei Matrizen stellt, sind geeignete Darstellungsmethoden und Algorithmen zur Approximation der Operationen dringend erforderlich. Die in diesem Bereich erzielten Resultate passen thematisch nicht in diese Monographie und gäben Stoff für ein weiteres Buch.

In zahlreichen Hinweisen wird verbal auf Details der Implementierung eingegangen. Eine konkrete Beschreibung der in C geschriebenen Algorithmen wird hier vermieden, da hierfür die Lecture Notes [26] und ein zukünftiges, hieraus hervorzugehendes Buchprojekt heranzuziehen sind. Letzteres ist auch ein besserer Platz für konkrete numerische Beispiele und Vergleiche.

Lesehinweis: Unterkapitel, die mit einem Stern* versehen sind, können beim ersten Lesen übersprungen werden, da sie eher der thematischen Vertiefung dienen.

Algorithmen sind in einer Notation angegeben, die sich an Algol und Pascal anlehnt. Man beachte, dass die geschweiften Klammern Kommentare enthalten. Der Einfachheit halber werden die Parameter nicht mit einer Typenbeschreibung versehen, sondern verbal erklärt. Für die (wenigen) Hilfsvariablen wird ebenfalls keine explizite Spezifikation gegeben, da sie sich aus dem Zusammenhang ergibt.

Der Autor entwickelte die Technik der hierarchischen Matrizen Ende der 90er Jahre. Die erste Arbeit [69] hierzu erschien 1999. Ein wesentlicher Schritt war die Implementierung der Methode in der Dissertation [50] von Dr. L. Grasedyck, die 2001 verteidigt wurde. Die Grundlagen zu dieser Monographie wurde durch Manuskripte zu Vorlesungen an der Universität Leipzig (Sommersemester 2004 und Wintersemester 2006/7) und an der Christian-Albrechts-Universität zu Kiel (Sommersemester 2004) gelegt. Das Material wurde entscheidend anreichert durch die Beiträge der Drs. S. Börm, M. Bebendorf, R. Kriemann und B. Khoromskij. Insgesamt sind zum Thema der hierarchischen Matrizen und seinem Umfeld zwei Dissertationen und eine Habilitationsarbeit an der Christian-Albrechts-Universität zu Kiel und vier Dissertationen und eine Habilitationsarbeit an der Universität Leipzig geschrieben worden. Die neuartigen Möglichkeiten, die die $\mathcal{H}$-Matrix-Technik bietet, haben zu einem Programmpaket $\mathcal{H}$-Lib^{pro} geführt, das für kommerzielle Zwecke zur Verfügung steht (siehe http://www.scai.fraunhofer.de/hlibpro.html).

Neben den oben Genannten danke ich Dr. L. Banjai, Dr. Wendy Kreß, Prof. Sabine Le Borne, Dr. Maike Löhndorf, Prof. J.M. Melenk, Prof. S. Sauter, sowie weiteren Mitarbeitern und Gästen des Leipziger Max-Planck-Institutes, die wesentlich zum Inhalt dieses Buches beigetragen haben. Herrn Dr. Ronald Kriemann gilt mein Dank für die Gestaltung des Umschlagbildes.

Dem Springer-Verlag danke ich für die unkomplizierte Kooperation bei der Fertigstellung dieses Werkes.

November 2008 *Wolfgang Hackbusch*

Inhaltsverzeichnis

1 Einleitung 1
1.1 Was ist die zu lösende Aufgabe, wo liegen die Schwierigkeiten? 1
1.1.1 Aufgabenbeispiele 1
1.1.2 Größenordnung der Dimension 3
1.1.3 Exakte oder näherungsweise Berechnung 3
1.2 Komplexität der Algorithmen 3
1.2.1 Komplexität 3
1.2.2 Warum braucht man (fast) lineare Komplexität für großskalige Probleme? 5
1.3 Zugrundeliegende Strukturen und Implementierungsdarstellungen 6
1.3.1 Vektor- und Matrixnotation 6
1.3.2 Implementierungsdarstellungen 7
1.3.3 Darstellungen und Operationen 12
1.4 In welchen Fällen ist lineare Komplexität erreichbar? 13
1.4.1 Familie der Diagonalmatrizen 13
1.4.2 Anwendung der schnellen Fourier-Transformation 13
1.4.3 Schwierigkeiten in anderen Fällen 14
1.5 Wo entstehen großskalige Probleme? 15
1.5.1 Diskretisierung elliptischer Differentialgleichungen 15
1.5.2 Integralgleichungen und ihre Diskretisierung 17
1.6 Angeordnete bzw. nicht angeordnete Indexmengen 21
1.6.1 Indexmengen 21
1.6.2 Vektoren $x \in \mathbb{R}^I$ 22
1.6.3 Matrizen $A \in \mathbb{R}^{I \times I}$ 22
1.6.4 Anordnung bzw. Nichtanordnung bei hierarchischen Matrizen 23
1.7 Übersicht über die weiteren Kapitel 23
1.7.1 Lokale Rang-k-Matrizen 23
1.7.2 Hierarchie und Matrixoperationen 24

2 Rang-k-Matrizen ... 25
2.1 Allgemeines ... 26
2.2 Darstellung und Kosten ... 26
2.3 Operationen und ihre Kosten ... 28
2.4 Bestapproximation durch Rang-k-Matrizen ... 30
2.5 Bestapproximation von Rang-ℓ-Matrizen durch Rang-k-Matrizen ... 33
2.6 Rang-k-Matrix-Addition mit anschließender Kürzung ... 35
2.6.1 Formatierte Addition ... 35
2.6.2 Formatierte Agglomeration ... 36
2.6.3 Mehr als zwei Terme ... 36
2.6.4 Stufenweise ausgeführte Agglomeration ... 38
2.7 Varianten der Rang-k-Matrixdarstellungen ... 39
2.7.1 AKB-Darstellung ... 39
2.7.2 SVD-Darstellung ... 41

3 Einführendes Beispiel ... 43
3.1 Das Modellformat $\mathcal{H}_p$ ... 43
3.2 Zahl der Blöcke ... 44
3.3 Speicheraufwand ... 45
3.4 Matrix-Vektor-Multiplikation ... 45
3.5 Matrix-Addition ... 45
3.6 Matrix-Matrix-Multiplikation ... 46
3.7 Matrixinversion ... 48
3.8 LU-Zerlegung ... 49
3.8.1 Vorwärtssubstitution ... 49
3.8.2 Rückwärtssubstitution ... 50
3.8.3 Aufwand der LU-Zerlegung ... 50
3.9 Weitere Eigenschaften der Modellmatrizen und Semiseparabilität * ... 51

4 Separable Entwicklung und ihr Bezug zu Niedrigrangmatrizen ... 55
4.1 Grundbegriffe ... 56
4.1.1 Separable Entwicklungen ... 56
4.1.2 Exponentielle Konvergenz ... 57
4.1.3 Zulässigkeitsbedingungen an X, Y ... 59
4.2 Separable Polynom-Entwicklungen ... 60
4.2.1 Taylor-Entwicklung ... 60
4.2.2 Interpolation ... 62
4.2.3 Exponentielle Fehlerabschätzung ... 63
4.2.4 Asymptotisch glatte Kerne ... 64
4.2.5 Taylor-Fehlerabschätzung ... 65
4.2.6 Interpolationsfehler für $d = 1$ ... 66
4.2.7 Verschärfte Fehlerabschätzung ... 68

4.2.8 Interpolationsfehler für $d > 1$ 69
4.3 Weitere separable Entwicklungen 70
4.3.1 Andere Interpolationsverfahren * 70
4.3.2 Transformationen * 70
4.3.3 Stückweise separable Entwicklung * 71
4.3.4 Kerne, die von $x - y$ abhängen 72
4.3.5 L-harmonische Funktionen * 72
4.3.6 Separable Entwicklungen mittels Kreuzapproximation * 73
4.3.7 Die optimale separable Entwicklung 73
4.4 Diskretisierung von Integraloperatoren mit separablen Kernfunktionen 74
4.4.1 Einführung: Separable Entwicklung und Galerkin-Diskretisierung 74
4.4.2 Separable Entwicklung und allgemeine Diskretisierungen * 76
4.5 Approximationsfehler * 78
4.5.1 Operatornormen 78
4.5.2 Matrixnormen 79
4.5.3 Sachgerechte Normen 81

5 Matrixpartition 83
5.1 Einleitung 83
5.1.1 Ziele 83
5.1.2 Eindimensionales Modellbeispiel 84
5.2 Zulässige Blöcke 85
5.2.1 Metrik der Cluster 85
5.2.2 Zulässigkeit 87
5.2.3 Verallgemeinerte Zulässigkeit 89
5.2.4 Erläuterung am Beispiel aus §5.1.2 90
5.3 Clusterbaum $T(I)$ 91
5.3.1 Definitionen 91
5.3.2 Beispiel 92
5.3.3 Blockzerlegung eines Vektors 93
5.3.4 Speicherkosten für $T(I)$ 94
5.4 Konstruktion des Clusterbaums $T(I)$ 96
5.4.1 Notwendige Daten 96
5.4.2 Geometriebasierte Konstruktion mittels Minimalquader 97
5.4.3 Kardinalitätsbasierte Konstruktion 101
5.4.4 Implementierung und Aufwand 101
5.4.5 Auswertung der Zulässigkeitsbedingung 102
5.5 Blockclusterbaum $T(I \times J)$ 104
5.5.1 Definition des stufentreuen Blockclusterbaums 104
5.5.2 Verallgemeinerung der Definition 105
5.5.3 Alternative Konstruktion von $T(I \times J)$ aus $T(I)$ und $T(J)$ 107

5.5.4 Matrixpartition . . . 108
5.5.5 Beispiele . . . 111
5.6 Alternative Clusterbaumkonstruktionen und Partitionen . . . 112

6 Definition und Eigenschaften der hierarchischen Matrizen . 113
6.1 Menge $\mathcal{H}(k, P)$ der hierarchischen Matrizen . . . 113
6.2 Elementare Eigenschaften . . . 115
6.3 Schwachbesetztheit und Speicherbedarf . . . 116
6.3.1 Definition . . . 116
6.3.2 Speicherbedarf einer hierarchischen Matrix . . . 118
6.4 Abschätzung von C_{sp} * . . . 120
6.4.1 Erster Zugang . . . 120
6.4.2 Abschätzung zu Konstruktion (5.30) . . . 123
6.4.3 Anmerkung zu Konstruktion (5.34) . . . 127
6.5 Fehlerabschätzungen . . . 128
6.5.1 Frobenius-Norm . . . 128
6.5.2 Vorbereitende Lemmata . . . 128
6.5.3 Spektralnorm . . . 135
6.5.4 Norm $|||\cdot|||$ . . . 136
6.6 Adaptive Rangbestimmung . . . 138
6.7 Rekompressionstechniken . . . 140
6.7.1 Kompression durch $\mathcal{T}_\varepsilon^{\mathcal{H}}$ . . . 140
6.7.2 Vergröberung der Blöcke . . . 141
6.8 Modifikationen des $\mathcal{H}$-Matrixformates . . . 141
6.8.1 $\mathcal{H}$-Matrizen mit Gleichungsnebenbedingungen . . . 141
6.8.2 Positive Definitheit . . . 143
6.8.3 Positivität von Matrizen . . . 144
6.8.4 Orthogonalität von Matrizen . . . 146

7 Formatierte Matrixoperationen für hierarchische Matrizen . 147
7.1 Matrix-Vektor-Multiplikation . . . 148
7.2 Kürzungen und Konvertierungen . . . 148
7.2.1 Kürzungen $\mathcal{T}_{k\leftarrow\ell}^{\mathcal{R}}$, $\mathcal{T}_k^{\mathcal{R}}$ und $\mathcal{T}_{k\leftarrow\ell}^{\mathcal{H}}$ . . . 148
7.2.2 Agglomeration . . . 150
7.2.3 Konvertierung $\mathcal{T}_k^{\mathcal{R}\leftarrow\mathcal{H}}$ . . . 150
7.2.4 Konvertierung $\mathcal{T}_{P'\leftarrow P}^{\mathcal{H}\leftarrow\mathcal{H}}$ . . . 152
7.2.5 Konvertierung $\mathcal{T}_{P'\leftarrow P}^{\mathcal{H}\leftarrow\mathcal{H}}$ bei unterschiedlichen Blockclusterbäumen * . . . 152
7.3 Addition . . . 154
7.4 Matrix-Matrix-Multiplikation . . . 155
7.4.1 Komplikationen bei der Matrix-Matrix-Multiplikation . . . 155
7.4.2 Algorithmus im konsistenten Fall . . . 157
7.4.3 Algorithmus im stufentreuen Fall . . . 167
7.5 Matrix-Inversion . . . 170
7.5.1 Rekursiver Algorithmus . . . 170

7.5.2 Alternativer Algorithmus mittels Gebietszerlegung 172
7.5.3 Newton-Verfahren 172
7.6 LU- bzw. Cholesky-Zerlegung 173
7.6.1 Format der Dreiecksmatrizen 173
7.6.2 Auflösung von $LUx = b$ 174
7.6.3 Matrixwertige Lösung von $LX = Z$ und $XU = Z$ 176
7.6.4 Erzeugung der LU- bzw. Cholesky-Zerlegung 177
7.7 Hadamard-Produkt 178
7.8 Aufwand der Algorithmen 179
7.8.1 Matrix-Vektor-Multiplikation 179
7.8.2 Matrix-Addition 179
7.8.3 Matrix-Matrix-Multiplikation 180
7.8.4 Matrix-Inversion 188
7.8.5 LU- bzw. Cholesky-Zerlegung 189

8 $\mathcal{H}^2$-Matrizen 191
8.1 Erster Schritt: $M|_b \in \mathcal{V}_b \otimes \mathcal{W}_b$ 191
8.2 Zweiter Schritt: $M|_{\tau\times\sigma} \in \mathcal{V}_\tau \otimes \mathcal{W}_\sigma$ 195
8.3 Definition der $\mathcal{H}^2$-Matrizen 197
8.3.1 Definition 197
8.3.2 Transformationen 197
8.3.3 Speicherbedarf 199
8.3.4 Projektion auf $\mathcal{H}^2$-Format 200
8.4 Hinreichende Bedingungen für geschachtelte Basen 202
8.4.1 Allgemeiner Fall 202
8.4.2 Beispiel: Approximation von Integraloperatoren durch Interpolation 203
8.5 Matrix-Vektor-Multiplikation mit $\mathcal{H}^2$-Matrizen 204
8.5.1 Vorwärtstransformation 204
8.5.2 Multiplikationsphase 205
8.5.3 Rücktransformation 206
8.5.4 Gesamtalgorithmus 206
8.6 $\mathcal{H}^2$-Matrizen mit linearem Aufwand 207
8.7 Adaptive Bestimmung der $\mathcal{H}^2$-Räume $\mathcal{V}_\tau$ und $\mathcal{W}_\sigma$ 209
8.8 Matrix-Matrix-Multiplikation von $\mathcal{H}^2$-Matrizen 213
8.8.1 Multiplikation bei gegebenem $\mathcal{H}^2$-Format 213
8.8.2 Multiplikation bei gesuchtem $\mathcal{H}^2$-Format 214
8.9 Numerisches Beispiel 215

9 Verschiedene Ergänzungen 217
9.1 Konstruktion schneller Iterationsverfahren 217
9.2 Modifizierte Clusterbäume für schwach besetzte Matrizen 219
9.2.1 Problembeschreibung 219
9.2.2 Finite-Element-Matrizen 220
9.2.3 Separierbarkeit der Matrix 222

9.2.4 Konstruktion des Clusterbaums 224
9.2.5 Anwendung auf Invertierung 226
9.2.6 Zulässigkeitsbedingung 226
9.2.7 LU-Zerlegung 227
9.2.8 $\mathcal{H}$-Matrixeigenschaften der LU-Faktoren 228
9.2.9 Geometriefreie Konstruktion der Partition 231
9.3 Schwache Zulässigkeit 232
9.3.1 Definition und Abschätzungen 232
9.3.2 Beispiel $k(x, y) = \log|x - y|$ 234
9.3.3 Zusammenhang mit der Matrixfamilie $\mathcal{M}_{k,\tau}$ 235
9.4 Kreuzapproximation 238
9.4.1 Basisverfahren und theoretische Aussagen 238
9.4.2 Praktische Durchführung der Kreuzapproximation 239
9.4.3 Adaptive Kreuzapproximation 241
9.4.4 Erzeugung separabler Entwicklungen mittels Kreuzapproximation 243
9.4.5 Die hybride Kreuzapproximation 245
9.5 Kriterien für Approximierbarkeit in $\mathcal{H}(k, P)$ 246
9.6 Änderung der Matrizen bei Gitterverfeinerung 249

10 Anwendungen auf diskretisierte Integraloperatoren 251
10.1 Typische Integraloperatoren für elliptische Randwertaufgaben 251
10.1.1 Randwertproblem und Fundamentallösung 252
10.1.2 Einfach-Schicht-Potential für das Dirichlet-Problem 252
10.1.3 Direkte Methode, Doppelschicht-Operator 253
10.1.4 Hypersingulärer Operator 254
10.1.5 Calderón-Projektion 254
10.2 Newton-Potential 255
10.3 Randelementdiskretisierung und Erzeugung der Systemmatrix in hierarchischer Form 255
10.4 Helmholtz-Gleichung für hohe Frequenzen 257
10.5 Allgemeine Fredholm-Integraloperatoren 258
10.6 Anwendungen auf Volterra-Integraloperatoren 258
10.6.1 Diskretisierungen von Volterra-Integraloperatoren 258
10.6.2 Implementierung als Standard-$\mathcal{H}$-Matrix 260
10.6.3 Niedrigrangdarstellung von Profilmatrizen 261
10.6.4 Matrix-Vektor-Multiplikation 262
10.7 Faltungsintegrale 265

11 Anwendungen auf Finite-Element-Matrizen 267
11.1 Inverse der Massematrix 267
11.2 Der Green-Operator und seine Galerkin-Diskretisierung 271
11.2.1 Das elliptische Problem 271
11.2.2 Die Green-Funktion 272
11.2.3 Der Green-Operator $\mathcal{G}$ 272

11.2.4 Galerkin-Diskretisierung von $\mathcal{G}$ und der Zusammenhang mit A^{-1} . . . 273
11.2.5 Folgerungen aus separabler Approximation der Greenschen Funktion . . . 275
11.3 Analysis der Greenschen Funktion . . . 279
11.3.1 L-harmonische Funktionen und innere Regularität . . . 280
11.3.2 Approximation durch endlich-dimensionale Unterräume 282
11.3.3 Hauptresultat . . . 284
11.3.4 Anwendung auf die Randelementmethode . . . 289
11.3.5 FEM-BEM-Kopplung . . . 290

12 Inversion mit partieller Auswertung . . . 291
12.1 Baum der Gebietszerlegung und zugehörige Spurabbildungen . . 292
12.2 Diskrete Variante - Übersicht . . . 294
12.3 Details . . . 295
12.3.1 Finite-Element-Diskretisierung und Matrixformulierung 295
12.3.2 Zerlegung der Indexmenge . . . 297
12.3.3 Die Abbildung Φ_ω . . . 298
12.3.4 Natürliche Randbedingung . . . 298
12.3.5 Zusammenhang der Matrizen . . . 299
12.3.6 Die Abbildung Ψ_ω . . . 299
12.3.7 Konstruktion von Φ_ω aus Ψ_{ω_1} und Ψ_{ω_2} . . . 300
12.3.8 Konstruktion von Ψ_ω aus Ψ_{ω_1} und Ψ_{ω_2} . . . 303
12.4 Basisalgorithmus . . . 304
12.4.1 Definitionsphase . . . 304
12.4.2 Auswertungsphase . . . 305
12.4.3 Homogene Differentialgleichung . . . 306
12.5 Verwendung hierarchischer Matrizen . . . 306
12.6 Partielle Auswertung . . . 308
12.6.1 Basisverfahren . . . 309
12.6.2 Realisierung mit hierarchischen Matrizen . . . 310
12.6.3 Vergröberung des Ansatzraumes für die rechte Seite . . . 311
12.6.4 Berechnung von Funktionalen . . . 311

13 Matrixfunktionen . . . 313
13.1 Definitionen . . . 313
13.1.1 Funktionserweiterung mittels Diagonalmatrizen . . . 314
13.1.2 Potenzreihen . . . 315
13.1.3 Cauchy-Integraldarstellung . . . 316
13.1.4 Spezialfälle . . . 317
13.2 Konstruktionen spezieller Funktionen . . . 317
13.2.1 Approximation von Matrixfunktionen . . . 317
13.2.2 Matrix-Exponentialfunktion . . . 319
13.2.3 Inverse Funktion $1/z$ . . . 324
13.2.4 Anwendung von Newton-artigen Verfahren . . . 328

13.3 $\mathcal{H}$-Matrix-Approximation . . . 328
13.3.1 Matrix-Exponentialfunktion . . . 328
13.3.2 Approximation nichtglatter Matrixfunktionen . . . 328

14 Matrixgleichungen . . . 329
14.1 Ljapunow- und Sylvester-Gleichung . . . 330
14.1.1 Definition und Lösbarkeit . . . 330
14.1.2 Andere Lösungsverfahren . . . 331
14.2 Riccati-Gleichung . . . 332
14.2.1 Definition und Eigenschaften . . . 332
14.2.2 Lösung mittels der Signumfunktion . . . 333
14.3 Newton-artige Verfahren zur Lösung nichtlinearer Matrixgleichungen . . . 334
14.3.1 Beispiel der Quadratwurzel einer Matrix . . . 334
14.3.2 Einfluss der Kürzung bei Fixpunktiterationen . . . 335

15 Tensorprodukte . . . 339
15.1 Tensor-Vektorraum . . . 339
15.1.1 Notationen . . . 339
15.1.2 Hilbert-Raum-Struktur . . . 341
15.1.3 Datenkomplexität . . . 341
15.2 Approximation im Tensorraum . . . 341
15.2.1 k-Term-Darstellung . . . 341
15.2.2 k-Term-Approximation . . . 342
15.2.3 Darstellung mit Tensorprodukten von Unterräumen . . . 343
15.3 Kronecker-Produkte von Matrizen . . . 343
15.3.1 Definitionen . . . 343
15.3.2 Anwendung auf die Exponentialfunktion . . . 345
15.3.3 Hierarchische Kronecker-Tensorproduktdarstellung . . . 346
15.4 Der Fall $d = 2$ . . . 346
15.4.1 Tensoren . . . 346
15.4.2 Kronecker-Matrixprodukte . . . 348
15.4.3 Komplexitätsbetrachtungen . . . 349
15.4.4 HKT-Darstellung . . . 350
15.5 Der Fall $d > 2$ . . . 351
15.5.1 Spezielle Eigenschaften . . . 351
15.5.2 Inverse eines separablen Differentialoperators . . . 353

A Graphen und Bäume . . . 355
A.1 Graphen . . . 355
A.2 Bäume . . . 356
A.3 Teilbäume . . . 358
A.4 Bäume zu Mengenzerlegungen . . . 359

B Polynome . . . 363
B.1 Multiindizes . . . 363
B.1.1 Notation . . . 363
B.1.2 Formelsammlung . . . 363
B.2 Polynomapproximation . . . 364
B.3 Polynominterpolation . . . 366
B.3.1 Eindimensionale Interpolation . . . 366
B.3.2 Tensorprodukt-Interpolation . . . 369

C Lineare Algebra, Funktionalanalysis, Singulärwertzerlegung . . . 371
C.1 Matrixnormen . . . 371
C.2 Singulärwertzerlegung von Matrizen . . . 373
C.3 Hilbert-Räume, L^2-Operatoren . . . 377
C.4 Singulärwertzerlegung kompakter Operatoren . . . 379
C.4.1 Singulärwertzerlegung . . . 379
C.4.2 Hilbert-Schmidt-Operatoren . . . 381
C.5 Abbildungen zu Galerkin-Unterräumen . . . 383
C.5.1 Orthogonale Projektion . . . 383
C.5.2 Unterraumbasis, Prolongation, Restriktion, Massematrix 383
C.5.3 Norm $|||\cdot|||$ . . . 385
C.5.4 Bilinearformen, Diskretisierung . . . 388

D Sinc-Interpolation und -Quadratur . . . 391
D.1 Elementare Funktionen . . . 391
D.2 Interpolation . . . 392
D.2.1 Definitionen . . . 392
D.2.2 Stabilität der Sinc-Interpolation . . . 393
D.2.3 Abschätzungen im Streifen $\mathfrak{D}_d$ . . . 394
D.2.4 Abschätzungen durch $\mathrm{e}^{-CN/\log N}$ . . . 397
D.2.5 Approximation der Ableitung . . . 399
D.2.6 Meromorphes f . . . 399
D.2.7 Andere Singularitäten . . . 400
D.3 Separable Sinc-Entwicklungen . . . 400
D.3.1 Direkte Interpolation . . . 400
D.3.2 Transformation und Skalierung . . . 401
D.3.3 Eine spezielle Transformation . . . 403
D.3.4 Beispiel $1/(x+y)$ . . . 404
D.3.5 Beispiel $\log(x+y)$ . . . 408
D.4 Sinc-Quadratur . . . 409
D.4.1 Quadraturverfahren und Analyse . . . 409
D.4.2 Separable Entwicklungen mittels Quadratur . . . 411
D.4.3 Beispiel: Integrand $\exp(-rt)$ . . . 412
D.4.4 Beispiel: Integrand $\exp(-r^2t^2)$ . . . 416

E Asymptotisch glatte Funktionen 419
E.1 Beispiel $|x-y|^{-a}$ 419
E.1.1 Richtungsableitungen 419
E.1.2 Gemischte Ableitungen 423
E.1.3 Analytizität 424
E.2 Asymptotische Glattheit weiterer Funktionen 425
E.3 Allgemeine Eigenschaften asymptotisch glatter Funktionen 427
E.3.1 Hilfsabschätzungen 428
E.3.2 Abschätzung für Richtungsableitungen 429
E.3.3 Aussagen für beschränkte Gebiete 430
E.3.4 Produkte asymptotisch glatter Funktionen 431

Literaturverzeichnis 435

Notationen 443

Sachverzeichnis 447

1

Einleitung

1.1 Was ist die zu lösende Aufgabe, wo liegen die Schwierigkeiten?

1.1.1 Aufgabenbeispiele

Aufgaben der linearen Algebra treten bei verschiedensten Anwendungen auf. Im Zusammenhang mit Vektoren (z.B.[1] im $\mathbb{R}^n$) sind die Vektorraum-Operationen

$$\begin{aligned} x, y \in \mathbb{R}^n &\quad\mapsto\quad x + y \in \mathbb{R}^n, \\ \lambda \in \mathbb{R},\, x \in \mathbb{R}^n &\quad\mapsto\quad \lambda x \in \mathbb{R}^n \end{aligned}$$

gefordert, was im Allgemeinen die geringsten Mühen bereitet.

Im Zusammenhang mit Matrizen (quadratischen $n \times n$-Matrizen wie rechteckigen $n \times m$-Matrizen) kann schon die *Abspeicherung* ein Problem darstellen, wenn n^2 bzw. nm groß sind.

Unter den Operationen mit Matrizen ist die häufigste die *Matrix-Vektor-Multiplikation*

$$A \in \mathbb{R}^{n\times m},\, x \in \mathbb{R}^m \quad\mapsto\quad Ax \in \mathbb{R}^n, \tag{1.1}$$

die der Realisierung der mit A beschriebenen linearen Abbildung entspricht.

Da Matrizen einen Ring beschreiben, möchte man auch die Ringoperationen durchführen:

$$A, B \in \mathbb{R}^{n\times m} \quad\mapsto\quad A + B \in \mathbb{R}^{n\times m}, \tag{1.2}$$

$$A \in \mathbb{R}^{n\times m},\, B \in \mathbb{R}^{m\times p} \quad\mapsto\quad A \cdot B \in \mathbb{R}^{n\times p} \tag{1.3}$$

[1] Für den zugrundeliegenden Körper nehmen wir im Folgenden stets $\mathbb{R}$ an. Gelegentlich ist auch der Körper $\mathbb{C}$ der komplexen Zahlen notwendig, aber die Verallgemeinerung auf $\mathbb{C}$ bringt keine prinzipiell neuen Probleme, sodass eine Beschränkung auf $\mathbb{R}$ Sinn macht. Völlig andersartige Körper wie z.B. $\mathbb{Z}_p$ werden explizit ausgeschlossen, da man hierin exakt rechnen müsste, während im Folgenden Approximationen angewandt werden sollen.

W. Hackbusch, *Hierarchische Matrizen*, DOI 10.1007/978-3-642-00222-9_1,

(die Skalarmultiplikation $A \in \mathbb{R}^{n\times m}, \lambda \in \mathbb{R} \mapsto \lambda A \in \mathbb{R}^{n\times m}$ gehört auch dazu, ist aber nicht so bemerkenswert). Schließlich möchte man für reguläre Matrizen die Inverse berechnen:

$$A \in \mathbb{R}^{n\times n} \text{ regulär} \quad \mapsto \quad A^{-1} \in \mathbb{R}^{n\times n}. \tag{1.4}$$

Daneben gibt es weitere Aufgaben wie die *LU-Zerlegung*

$$\begin{array}{l} A \in \mathbb{R}^{n\times n} \text{ regulär} \quad \mapsto \quad A = LU, \\ L \text{ untere normierte und } U \text{ obere Dreiecksmatrix,} \end{array} \tag{1.5a}$$

die eventuell um die Pivotisierung ergänzt werden muss, sowie die *Cholesky-Zerlegung*[2]

$$\begin{array}{l} A \in \mathbb{R}^{n\times n} \text{ positiv definit} \quad \mapsto \quad A = LL^\top, \\ L \text{ untere Dreiecksmatrix mit } L_{ii} > 0 \end{array} \tag{1.5b}$$

(vgl. Stoer [129, Abschnitte 4.1 und 4.3]).

Im Falle eines Gleichungssystems $Ax = b$ möchte man nicht die inverse Matrix berechnen, sondern nur das spezielle Bild $x = A^{-1}b$ ermitteln:

$$A \in \mathbb{R}^{n\times n} \text{ regulär}, b \in \mathbb{R}^n \quad \mapsto \quad x \in \mathbb{R}^n \text{ Lösung von } Ax = b. \tag{1.6}$$

Mitunter sind auch Funktionen von Matrizen von Interesse. Das bekannteste Beispiel ist die *Matrix-Exponentialfunktion*

$$A \in \mathbb{R}^{n\times n} \quad \mapsto \quad \exp(A) := \sum_{\nu=0}^{\infty} \frac{1}{\nu!} A^\nu. \tag{1.7}$$

Schließlich gibt es noch (lineare oder nichtlineare) Gleichungssysteme für Matrizen. Eine bekannte lineare Gleichung in X ist die *Ljapunow*[3]*-Gleichung*

$$AX + XA^\top = B \qquad (A, B \in \mathbb{R}^{n\times n} \text{ gegeben}, X \in \mathbb{R}^{n\times n} \text{ gesucht}). \tag{1.8}$$

Die *Riccati*[4]*-Gleichung*

$$AX + XA^\top + XCX = B \quad (A,B,C \in \mathbb{R}^{n\times n} \text{ gegeben}, X \in \mathbb{R}^{n\times n} \text{ gesucht}) \tag{1.9}$$

ist quadratisch in X und daher nichtlinear.

---

[2] André Louis Cholesky (geb. 15.10.1875 in Montguyon bei Bordeaux, gest. am 31.8.1918) setzte dieses Verfahren für Kleinste-Quadrate-Probleme ein, die bei geodätischen Vermessungen auftraten. Sein Verfahren wurde posthum von Benoît [16] veröffentlicht. Eine Beschreibung des Verfahrens findet sich auf handschriftlichen Notizen von Cholesky datiert auf den 2.12.1910.

[3] Aleksandr Michailowitsch Ljapunow, geboren am 6. Juni 1857 in Yaroslavl, gestorben am 3. Nov. 1918 in Odessa.

[4] Jacopo Francesco Riccati, geboren am 28. Mai 1676 in Venedig, gestorben am 15. April 1754 in Treviso (Venezianische Republik).

1.1.2 Größenordnung der Dimension

Solange n eine feste, kleine Zahl ist, stellen die genannten Aufgaben (wenn sie nicht schlecht konditioniert sind) keine Schwierigkeit dar. Es gibt aber eine große Klassen von Problemen der Linearen Algebra, die aus der Diskretisierung partieller Differentialgleichungen oder verwandter Integralgleichungen resultieren. Vor der Diskretisierung sind die Aufgaben unendlich-dimensional. Der durch die Diskretisierung induzierte Fehler (der sogenannte *Diskretisierungsfehler*) geht gegen null, wenn die Dimension n gegen ∞ geht. Daher wird man versuchen, n so groß wie möglich zu wählen, und das heißt, so groß, dass der Rechner-Speicherplatz ausreicht und das Resultat der Rechnung in einer Zeitspanne geliefert wird, die akzeptabel ist. Man hat sich die Größenordnung $n = 10^6$ und mehr vorzustellen, und eventuell benötigt der Rechner mehrere Tage für die Ausführung.

1.1.3 Exakte oder näherungsweise Berechnung

Die oben aufgezählten Aufgaben fragen nach dem exakten Resultat der jeweiligen Operation. Legt man den Körper $\mathbb{R}$ zugrunde, so ist offensichtlich, dass Rundungsfehler in der Größenordnung der relativen Maschinengenauigkeit *eps* unvermeidbar sind. Viel entscheidender ist aber, dass wegen der in §1.1.2 angesprochenen Diskretisierung auch bei exakt ausgeführter linearer Algebra die Resultate nur Näherungen gegenüber der Lösung der eigentlichen partiellen Differentialgleichung darstellen. Wichtig ist in diesem Zusammenhang, dass der Diskretisierungsfehler $\varepsilon_{\text{Diskr}}$ im Allgemeinen um Größenordnungen größer als *eps* ist. Ein zusätzlicher Fehler infolge einer inexakten Auswertung der algebraischen Operationen in der Größenordnung von $\varepsilon_{\text{Diskr}}$ ist durchaus akzeptabel.

Die Tatsache, dass man beispielsweise ein Gleichungssystem (1.6) nicht exakt, sondern nur näherungsweise berechnen möchte, erlaubt iterative Verfahren, deren Iterationsfehler mit der Anzahl der Iterationsschritte und damit mit dem Rechenaufwand verbunden ist (vgl. Hackbusch [66]).

1.2 Komplexität der Algorithmen

Es reicht nicht aus zu wissen, dass ein Algorithmus prinzipiell durchführbar ist, sondern man muss auch sicherstellen, dass die Speicherkapazität und Rechengeschwindigkeit der zur Zeit verfügbaren Rechner ausreichen. Für diesen Zweck ist die Komplexität eines Algorithmus zu quantifizieren.

1.2.1 Komplexität

Sei $\Phi : x \in D \subset \mathbb{R}^n \mapsto y \in \mathbb{R}^m$ eine beliebige Abbildung. Ein Algorithmus $\mathcal{A}_\Phi$ zur Berechnung von $\Phi(x)$ ist eine Folge von *Elementaroperationen*,

die durch die Implementierung festgelegt werden. Unter der vereinfachenden Annahme, dass alle als Elementaroperationen zugelassenen Operationen die gleiche Rechenzeit benötigen, ist die Rechenzeit durch die Anzahl N_Φ der Elementaroperationen charakterisiert. Im Allgemeinen kann N_Φ vom Argument x abhängen, sodass man dann zur maximalen Anzahl $\sup_x N_\Phi(x)$ übergehen sollte.[5]

Anmerkung 1.2.1. Die Abbildung $\Phi : x \in D \subset \mathbb{R}^n \mapsto y \in \mathbb{R}^m$ sei in dem Sinne nichttrivial, dass $\Phi(x)$ von allen Eingabedaten (d.h. allen Komponenten x_i) abhängt und y im Allgemeinen verschiedene Komponenten y_i besitzt und nicht die Identität $\Phi_j(x) = x_i$ (für gewisse Paare (i, j) und alle $x \in D$) zutrifft. Dann gilt $N_\Phi \geq \max\{n/2, m\}$.[6]

Beweis. Die Elementaroperationen haben höchstens zwei Argumente. Zu jedem i muss es eine Elementaroperation geben, die x_i als Argument enthält, da sonst $\Phi(x)$ nicht von x_i abhängt. Damit müssen mindestens $n/2$ Elementaroperationen auftreten. Außerdem muss jedes y_j Resultat einer Elementaroperation sein, sodass ihre Anzahl $\geq m$ ist. ∎

Der vom Algorithmus in Anspruch genommene Speicher (Einheit ist der Speicherbedarf einer reellen Maschinenzahl) sei mit S_Φ bezeichnet. Dabei kann in günstigen Fällen S_Φ unabhängig von n, m sein (Beispiel: Summation aus Fußnote 6), ist in ungünstigen Situationen aber aufgrund von vielen simultan auftretenden Zwischenwerten wesentlich größer.

Zur Vereinfachung beschränken wir uns auf *einen* Dimensionsparameter n (d.h. $n = m$). Viele Algorithmen haben n als Parameter, d.h. die obige Abbildung Φ wird als $\Phi_n : x \in \mathbb{R}^n \mapsto y$ geschrieben, und man betrachtet die Familie aller $\{\Phi_n : n \in \mathbb{N}\}$. Alle Aufgaben aus §1.1.1 sind von dieser Art. Damit werden N_Φ und S_Φ Funktionen von $n \in \mathbb{N}$:

$$N_\Phi(n) := N_{\Phi_n}, \qquad S_\Phi(n) := S_{\Phi_n}.$$

Folglich lässt sich das Verhalten von N_Φ und S_Φ bezüglich $n \to \infty$ diskutieren. Wir benutzen das Landau[7]-Symbol $\mathcal{O}$: $N_\Phi(n) = \mathcal{O}(g(n))$ bedeutet, dass Konstanten K und n_0 existieren, sodass $N_\Phi(n) \leq Kg(n)$ für alle $n \geq n_0$.

Gemäß Anmerkung 1.2.1 ist $N_\Phi(n) = \mathcal{O}(n)$ die bestmögliche Abschätzung. In diesem Falle spricht man von *linearer Komplexität.*

Die Matrix-Vektor-Multiplikation $x \in \mathbb{R}^n \mapsto y := Ax \in \mathbb{R}^n$ wird im Allgemeinen mit Hilfe der Summation $y_j = \sum_{j=1}^n A_{jk}x_k$ berechnet

[5] Falls eine sinnvolle statistische Verteilung der Argumente x definiert werden kann, lässt sich auch ein Erwartungswert $E(N_\Phi(\cdot))$ definieren.

[6] $\max\{n/2, m\}$ kann sogar durch $\max\{n-1, m\}$ ersetzt werden (Beweishintergrund: Ein binärer Baum mit n Blättern muss außer den Blättern noch $n-1$ weitere Knoten enthalten; vgl. Lemma A.3.4). Ein Beispiel für die Anzahl $N_\Phi = n-1$ ist $\Phi(x) = \sum_{i=1}^n x_i =: y \in \mathbb{R}^1$ (d.h. $m = 1$).

[7] Edmund Georg Hermann Landau, geboren am 14. Februar 1877 in Berlin, gestorben am 19. Februar 1938 in Berlin.

und benötigt damit n^2 Multiplikationen und $n\,(n-1)$ Additionen. Also ist $N_\Phi(n) = 2n^2 - n = \mathcal{O}(n^2)$, d.h. dieser Algorithmus hat *quadratische Komplexität*.

Für die Gleichungsauflösung $b \in \mathbb{R}^n \mapsto x := A^{-1}b \in \mathbb{R}^n$ aus (1.6) lässt sich die Gauß[8]-Elimination verwenden. Sie besitzt *kubische Komplexität*: $N_\Phi(n) = \mathcal{O}(n^3)$. Allerdings lässt sich diese Aufgabe auch anders interpretieren. Sieht man neben $b \in \mathbb{R}^n$ die Matrix $A \in \mathbb{R}^{n\times n}$ als Eingabe an, ist $n' := n^2 + n$ die Zahl der Eingabedaten und N_Φ gleich $\mathcal{O}(n'^{\,3/2})$.

Bisher traten nur Beispiele für polynomielle Komplexität auf, d.h. $\mathcal{O}(n^p)$. In der diskreten Optimierung sind die meisten interessanten Probleme schwierig in dem Sinne, dass $N_\Phi(n)$ nicht polynomiell beschränkt ist und beispielsweise exponentiell in n wächst. Ein Beispiel aus der Linearen Algebra ist der nicht nachahmenswerte Versuch, $\det(A)$ für $A \in \mathbb{R}^{n\times n}$ mittels der Laplaceschen[9] Entwicklungsformel zu berechnen. Hierfür ermittelt man $N_\Phi(n) = \mathcal{O}(n!)$.

Die schnelle Fourier[10]-Transformation $\Phi_n : x \in \mathbb{R}^n \mapsto \hat{x} \in \mathbb{R}^n$ (siehe Übung 1.4.1) benötigt den Aufwand $\mathcal{O}(n \log n)$. Da der Logarithmus nur sehr schwach ansteigt, werden wir im Folgenden von *fast linearer Komplexität* sprechen, wenn $N_\Phi(n) = \mathcal{O}(n \log^q n)$ für ein von n unabhängiges $q \geq 0$ gilt.

1.2.2 Warum braucht man (fast) lineare Komplexität für großskalige Probleme?

Als *großskalige Probleme* werden solche bezeichnet, die man mit maximal möglichem n berechnen möchte. Ist $S_{\max}$ der maximal verfügbare Speicher, so kann man an der Berechnung von Φ_n mit $n_{\max} := \max\{n' : S_\Phi(n') \leq S_{\max}\}$ interessiert sein. Es wurde schon bemerkt, dass Diskretisierungen partieller Differentialgleichungen zu derartigen großskaligen Problemen führen.

Weiterhin nehmen wir an, dass der Speicherbedarf der Algorithmen $S_\Phi(n) = \mathcal{O}(n)$ beträgt.

Um zu verstehen, dass nur Algorithmen von (fast) linearer Komplexität akzeptabel sind, hat man die zeitliche Entwicklung der Rechnertechnik einzubeziehen. Es ist eine empirische Beobachtung, dass sich sowohl die Speicherkapazität als auch die Rechengeschwindigkeit pro Zeiteinheit um einen konstanten Faktor verbessern. Für die folgenden Überlegungen ist es nur wichtig, dass beide um den gleichen Faktor ansteigen. Danach gibt es z.B. eine Zeitspanne Δt, in der sich der Speicher verdoppelt ($S_{\max}(t + \Delta t) = 2S_{\max}(t)$) und der Zeitbedarf für eine Elementaroperation halbiert. Das oben definierte $n_{\max}$ ist damit auch zeitabhängig: wegen

[8] Johann Carl Friedrich Gauß, geboren am 30. April 1777 in Braunschweig, gestorben am 23. Feb. 1855 in Göttingen.

[9] Pierre-Simon Laplace, geboren am 23. März 1749 in Beaumont-en-Auge, Normandy, gestorben am 5. März 1827 in Paris.

[10] Jean Baptiste Joseph Fourier, geboren am 21. März 1769 in Auxerre (Bourgogne), gestorben am 16. Mai 1830 in Paris.

$S_\Phi(n) = \mathcal{O}(n)$ gilt $n_{\max}(t + \Delta t) = 2n_{\max}(t)$. Auf einem Rechner zur Zeit $t + \Delta t$ berechnet man die Aufgabe mit dem gestiegenen $n_{\max}$. Die Zahl der Operationen ist

$$N_\Phi(n_{\max}(t + \Delta t)) = N_\Phi(2n_{\max}(t)).$$

Es sei nun eine polynomielle Komplexität $\mathcal{O}(n^p)$ angenommen. Dann ist $N_\Phi(n_{\max}(t+\Delta t)) = N_\Phi(2n_{\max}(t)) \approx 2^p N_\Phi(n_{\max}(t))$, d.h. der Aufwand steigt um den Faktor 2^p. Da gleichzeitig die Rechengeschwindigkeit verdoppelt ist, steigt die benötigte Rechenzeit nur um $2^p/2 = 2^{p-1}$.

Hieraus ergibt sich der folgende Schluss. Falls $p > 1$, benötigt der neue Rechner eine um den Faktor $2^{p-1} > 1$ vergrößerte Rechenzeit. Nur falls $p = 1$, bleibt die Rechenzeit konstant. Damit "überleben" nur die Algorithmen mit linearer Komplexität (wieder gilt, dass fast lineare Komplexität tolerierbar ist; hierbei steigt die Rechenzeit nur um eine additive Größe).

Es ist offenbar, dass ein zu großer Speicherbedarf die Verwendung eines Algorithmus ausschließt. Dies gilt aber auch hinsichtlich der Rechendauer. Für große n liegen die Rechenzeiten bei linearer und quadratischer Komplexität um den Faktor $\mathcal{O}(n)$ auseinander. Schon bei $n = 500\,000$ – einer Dimension, die heute durchaus auftritt – ist dies der Quotient zwischen einer Minute und einem Jahr.

1.3 Zugrundeliegende Strukturen und Implementierungsdarstellungen

Es gibt viele Aufgaben, für deren Lösung man in allgemeinen Fall keinen befriedigenden Algorithmus kennt; trotzdem können Spezialfälle[11] gut lösbar sein. Für eine gute Implementierung ist es wesentlich, dass die zugrundeliegenden Strukturen ausgenutzt werden.

1.3.1 Vektor- und Matrixnotation

Zunächst wird eine mathematische Notation eingeführt. Anstelle von $x \in \mathbb{R}^n$ verwenden wir die präzisere Schreibweise

$$x = (x_i)_{i \in I} \in \mathbb{R}^I, \tag{1.10a}$$

wobei I die zugrundeliegende (nicht notwendigerweise angeordnete) endliche Indexmenge ist. Wenn die Schreibweise $\mathbb{R}^n$ verwendet wird, so ist sie ein Sammelbegriff für alle $\mathbb{R}^I$ mit $\#I = n$. Hierbei bezeichnet das Symbol $\#$ die Elementeanzahl (Kardinalität) einer Menge.

Analog zur Schreibweise $\mathbb{R}^I$ für die Vektormenge, schreiben wir

[11] Die meisten in der Praxis auftretenden und zu lösenden Aufgaben sind in dem Sinne Spezialfälle, als sie spezifische Strukturen enthalten. Spezialfälle sind daher eher die Regel als die Ausnahme!

$$M \in \mathbb{R}^{I \times J} \tag{1.10b}$$

für Matrizen $(M_{i,j})_{i \in I, j \in J}$. Die sonst übliche Schreibweise $\mathbb{R}^{n \times m}$ wird als Sammelbegriff für alle Mengen $\mathbb{R}^{I \times J}$ mit $\#I = n$ und $\#J = m$ benutzt.

1.3.2 Implementierungsdarstellungen

Auch wenn ein mathematisches Objekt eindeutig definiert ist, so erlaubt es doch viele Darstellungen für die Implementierung auf dem Rechner. Jede der Darstellungen kann für Spezialfälle besonders vorteilhaft sein. Umgekehrt findet man selten Darstellungen, die in jedem Fall optimal sind. Die gewählte Darstellung ist einerseits wichtig für die Abspeicherung der Daten und bestimmt den Speicheraufwand. Andererseits werden mathematische Objekte mittels Operationen verknüpft, was je nach Wahl der Darstellung zu mehr oder weniger Rechenaufwand führen kann. Im Folgenden geben wir einige Beispiele für Implementierungsdarstellungen von Vektoren (§§1.3.2.1-1.3.2.3) und Matrizen (§§1.3.2.4-1.3.2.12). Außerdem werden Begriffe eingeführt, die später benötigt werden.

1.3.2.1 Voller Vektor

Die naheliegende Darstellung eines Vektors $x \in \mathbb{R}^I$ ist ein Tupel $(x_i)_{i \in I}$ von `real`-Maschinenzahlen, wobei das Tupel entweder als `array` oder als Liste organisiert ist (auch die Information über I sollte verfügbar sein). Der damit verbundene Speicherplatz ist $S = \mathcal{O}(\#I)$. Diese Darstellung kann als `Voller_Vektor(`I`)` bezeichnet werden (falls $I = \{1, \ldots, n\}$, auch `Voller_Vektor(`n`)` genannt).

Anmerkung 1.3.1. Ein Skalarprodukt $(x, y) = \sum_{i \in I} x_i y_i$ zweier voll besetzter Vektoren $x, y \in \mathbb{R}^I$ kostet $2\#I - 1$ Operationen.

1.3.2.2 Schwach besetzter Vektor

Eine Alternative könnte ein schwach besetzter Vektor sein. Wenn nur ein kleiner Teil der Komponenten x_i von $x \in \mathbb{R}^I$ ungleich null ist, stellt dies eine Struktur dar, die man ausnutzen kann. Es wird eine Liste von Paaren

$$\big((i_1, x_{i_1}),\ (i_2, x_{i_2}), \ldots, \big(i_p, x_{i_p}\big)\big)$$

mit $0 \le p \le \#I$ abgespeichert, die alle Nicht-Null-Komponenten enthält. Falls die Indizes angeordnet sind, sollte zudem $i_1 < i_2 < \ldots < i_p$ vorausgesetzt werden. Dies definiert die Darstellung `Duenner_Vektor(`I`)`. In diesem Format lässt sich z.B. ein Einheitsvektor als das 1-Tupel $(i, 1)$ darstellen, hat also einen von n unabhängigen Speicheraufwand.

1.3.2.3 Blockvektor

Da die *Partition* einer Indexmenge auch später eine wichtige Rolle spielen wird, geben wir eine ausdrückliche Definition:

Definition 1.3.2 (Partition). *I sei eine endliche Indexmenge. P ist eine* Partition *von I, falls $P = \{I_1, \ldots, I_p\} \subset \mathcal{P}(I)\backslash\{\emptyset\}$ ($\mathcal{P}$: Potenzmenge) die Eigenschaften*

$$I_i \cap I_j = \emptyset \text{ für } i \neq j \text{ (Disjunktheit)}, \quad I = \bigcup_{j=1}^{p} I_j \text{ (Vollständigkeit)} \tag{1.11}$$

besitzt. Die angeordnete Menge $\{1, \ldots, p\}$ kann auch durch eine nicht angeordnete Indexmenge K ersetzt werden: $P = \{I_\iota : \iota \in K\}$.

Der *Vektorblock* von $x \in \mathbb{R}^I$ zu $\tau \in P$ ist

$$x|_\tau := (x_i)_{i\in\tau} \in \mathbb{R}^\tau. \tag{1.12}$$

Eine Darstellung von x als *Blockvektor*[12] ist durch $x = (x|_{I_j})_{j=1,\ldots,p}$ gegeben. Jeder Vektorblock kann eine der bisher beschriebenen Darstellungen haben.

Während $x \in \mathbb{R}^I \mapsto x|_{I'} \in \mathbb{R}^{I'}$ ($I' \subset I$) der Übergang zu einem Vektorblock ist, tritt auch der umgekehrte Prozess auf:

Definition 1.3.3 (Einbettung, Agglomeration). *a) Seien $\tau \subset I$ und $z \in \mathbb{R}^\tau$. Die Einbettung von z in $\mathbb{R}^I$ geschieht mittels*

$$x := z|^I \in \mathbb{R}^I, \text{ wobei } x_i := \begin{cases} z_i \text{ für } i \in \tau, \\ 0 \quad \text{sonst.} \end{cases} \tag{1.13}$$

b) Seien $I_1, I_2 \in P$ zwei verschiedene Indexblöcke einer Partition von I und $y \in \mathbb{R}^{I_1}$, $z \in \mathbb{R}^{I_2}$. Dann wird die Summe $x := y|^I + z|^I$ auch als Agglomeration *von y und z bezeichnet.*

Man beachte, dass im zweiten Fall I_1 und I_2 disjunkt sind. Die Agglomeration des Blockvektors durch seine Vektorblöcke schreibt sich nun als $x = \sum_j (x|_{I_J})|^I$.

1.3.2.4 Volle Matrix

Matrizen können viele verschiedene Strukturen besitzen, die auf verschiedenen Ebenen ausgenutzt werden können. Die primitivste Form, die keine Strukturen aufzeigt, ist die Darstellung als volle Matrix, die in der Form `Volle_Matrix(`I, J`) = array(`I`) of Voller_Vektor(`J`)` konstruiert werden könnte, d.h. jede der $\#I$ Zeilen ist als voller Vektor wie oben dargestellt. Der Speicheraufwand ist $S = \#I\#J$.

[12] Man beachte den Unterschied von *Vektorblock* und *Blockvektor*. Ein Vektorblock ist ein Teilvektor zur Indexmenge I_j, ein Blockvektor dagegen der Gesamtvektor mit einer Blockstruktur.

Anmerkung 1.3.4. a) Seien $M \in \mathbb{R}^{I\times J}$ eine volle Matrix und $x \in \mathbb{R}^J$. Die Matrix-Vektor-Multiplikation $M \cdot x$ kostet $\#I\,(2\#J-1)$ Operationen.
b) Seien $A \in \mathbb{R}^{I\times J}$ und $B \in \mathbb{R}^{J\times K}$ zwei volle Matrizen. Die Matrix-Matrix-Multiplikation $A \cdot B$ kostet $\#I\#K\,(2\#J-1)$ Operationen.

Beweis. Die obigen Berechnungen erfordern a) $\#I$ bzw. b) $\#I\#K$ Skalarprodukte von J-Vektoren. ∎

Übung 1.3.5. Das Produkt dreier Matrizen $A \in \mathbb{R}^{I\times J}$, $B \in \mathbb{R}^{J\times K}$ und $C \in \mathbb{R}^{K\times L}$ ist wegen der Kommutativität auf zwei Arten berechenbar: als $A \cdot (B \cdot C)$ oder $(A \cdot B) \cdot C$. Sind die Kosten gleich? Falls nicht, was ist billiger?

1.3.2.5 Schwach besetzte Matrix

Die Finite-Element-Diskretisierung von partiellen Differentialgleichungen liefert schwach besetzte Matrizen, das heißt, jede Zeile enthält nur wenige Nicht-Null-Einträge. Hier bietet sich das Format `array(`I`) of Duenner_Vektor(`J`)` an. Wenn, wie in diesem Beispiel, die Zahl der Nicht-Null-Einträge unabhängig von $\#J$ beschränkt ist, beträgt der Speicheraufwand $S = \mathcal{O}(\#I)$.

1.3.2.6 Bandmatrix

Spezielle schwach besetzte Matrix sind die Bandmatrizen, bei denen $I = J = \{1, 2, \ldots, n\}$ eine angeordnete Indexmenge ist und die Nicht-Null-Einträge auf die Positionen (i, j) mit $|i - j| \leq b$ beschränkt sind (b: Bandbreite). Pro Zeile sind $2b+1$ Zahlen (z.B. im Format `Voller_Vektor(`$2b+1$`)`) abzulegen, sodass der Speicheraufwand $S = n(2b + 1) = \mathcal{O}(bn)$ ist. Für $b = 1$ erhält man als Spezialfall die *tridiagonalen* Matrizen. Der Fall $b = 0$ wird nachfolgend behandelt.

1.3.2.7 Diagonalmatrix

Der angenehmste Fall ist die Diagonalmatrix $A = \mathrm{diag}\{a_{ii} : i \in I\}$. Hier enthält die Darstellung mittels `Voller_Vektor(`I`)` alle notwendigen Daten. Der Speicheraufwand ist $S = n$.

1.3.2.8 Toeplitz-Matrix

Seien $I = \{1, \ldots, n\} \subset \mathbb{Z}$ und $J = \{1, \ldots, m\} \subset \mathbb{Z}$. Bei einer Toeplitz[13]-Matrix A hängen die Einträge A_{ij} nur von der Differenz $i - j$ ab, d.h. in der Diagonalen sowie in jeder Nebendiagonalen stehen identische Werte:

[13] Otto Toeplitz, geboren am 1. August 1881 in Breslau, gestorben am 15. Februar 1940 in Jerusalem. Er war Ordinarius an den Universitäten Kiel (bis 1928) und Bonn (bis 1933).

$$A = \begin{bmatrix} a_0 & a_1 & \dots & a_{n-1} \\ a_{-1} & a_0 & a_1 & \dots \\ \vdots & \ddots & \ddots & \ddots \\ a_{1-n} & \dots & a_{-1} & a_0 \end{bmatrix} \qquad \text{(Toeplitz-Matrix für } n = m\text{)}.$$

Für $i \in I$, $j \in I$ variiert $i-j$ in der Differenzmenge $K := \{1-m, \dots, n-1\}$. Mit `Voller_Vektor(`K`)` werden die notwendigen Daten $(a_k)_{k\in K}$ bereitgestellt. Der Speicheraufwand beträgt $S = \#I + \#J - 1$.

1.3.2.9 Zirkulante Matrix

Ein Spezialfall einer quadratischen Toeplitz-Matrix ist die zirkulante Matrix. Hier hängt A_{ij} nur von der Differenz $i - j$ *modulo* n ab:

$$A = \begin{bmatrix} a_0 & a_1 & & a_{n-1} \\ a_{n-1} & a_0 & a_1 & \\ & \ddots & \ddots & \ddots \\ a_1 & & a_{n-1} & a_0 \end{bmatrix}. \tag{1.14}$$

Da die Werte $(a_0, \dots, a_{n-1})$ eine zirkulante Matrix vollständig charakterisieren, ist die Darstellung `Voller_Vektor(`n`)` möglich und benötigt den Speicheraufwand $S = n$.

1.3.2.10 Rang-k-Matrix

Falls eine Matrix $A \in \mathbb{R}^{I\times J}$ einen Rang $\le k$ besitzt, gibt es eine Faktorisierung

$$A = B \cdot C^\top \qquad \text{mit } B \in \mathbb{R}^{I\times\{1,\dots,k\}},\ C \in \mathbb{R}^{J\times\{1,\dots,k\}}, \tag{1.15}$$

die hilfreich ist, wenn $k \ll \min\{\#I, \#J\}$, d.h. wenn

$$B = \boxed{}, \qquad C^\top = \boxed{}$$

schmale Formate haben. Die Darstellung `Rang(`k`)Matrix(`I, J`)` besteht aus dem Paar von `Volle_Matrix(`I, K`)` und `Volle_Matrix(`J, K`)`, wobei $K = \{1, \dots, k\}$. Der Fall $k = 0$ ist als Null-Matrix zu interpretieren.

Der Speicheraufwand im Falle des Formates `Rang(`k`)Matrix(`I, J`)` ist $S = k\,(\#I + \#J)$.

Im Namen "Rang-k-Matrix" ist k eine freie Variable, sodass auch von einer Rang-ℓ-Matrix oder Rang-16-Matrix gesprochen werden kann.

1.3.2.11 Blockmatrix

Was die Abspeicherung betrifft, könnte eine Matrix aus $\mathbb{R}^{I\times J}$ als ein Vektor zur Indexmenge $I\times J$ behandelt werden. Die zusätzliche Produktstruktur von $I\times J$ wird jedoch in der folgenden Definition der Blockpartition berücksichtigt (vgl. Definition 1.3.2).

Definition 1.3.6 (Blockpartitionierung). *Seien I und J endliche Indexmengen. $P=\{b_1,\ldots,b_p\}\subset\mathcal{P}(I\times J)\backslash\{\emptyset\}$ ist eine* Blockpartition *von $I\times J$, falls*

$$\begin{array}{ll} \text{für alle } i\in\{1,\ldots,p\} \text{ ist } b_i = I'\times J' \text{ mit } I'\subset I, J'\subset J & (\textit{Produktstruktur}),\\ b_i\cap b_j=\emptyset \text{ für } i\neq j & (\textit{Disjunktheit}),\\ I\times J=\bigcup_{j=1}^{p} b_j & (\textit{Vollständigkeit}). \end{array}$$

Ein Beispiel einer Blockpartition sieht man in (3.3). Häufig wird eine wesentlich speziellere Blockpartition verwendet:

Anmerkung 1.3.7. Seien P_I und P_J Partitionen von I und J im Sinne von Definition 1.3.2. Dann definiert

$$P:=\{b=\tau\times\sigma:\tau\in P_I,\ \sigma\in P_J\}$$

die von P_I und P_J erzeugte *Tensor-Blockpartition* von $I\times J$.

b wird *Indexblock* (oder kürzer: Block) genannt. Die mittels einer Blockpartition strukturierte Matrix wird kurz als *Blockmatrix* bezeichnet. Der Name *Matrixblock* wird dagegen für die Untermatrix verwendet, die einem Block $b\in P$ entspricht. Wir führen hierfür die folgende Notation ein:

$$M|_b := (M_{ij})_{(i,j)\in b}\in\mathbb{R}^b \qquad \text{für ein } b\in P. \tag{1.16}$$

Die Blockmatrix kann demnach in der Form $M=(M|_b)_{b\in P}$ geschrieben werden. Jeder Matrixblock kann eine der bisher beschriebenen Darstellungen haben.

In völliger Analogie zu Definition 1.3.3 formulieren wir die

Definition 1.3.8 (Einbettung, Agglomeration). *Sei eine Blockpartition P von $I\times J$ gegeben. a) Seien $b\in P$ und $Z\in\mathbb{R}^b$. Die Einbettung von Z in $\mathbb{R}^{I\times J}$ geschieht mittels*

$$M:=Z|^{I\times J}\in\mathbb{R}^{I\times J},\ \textit{wobei } M_{i,j}:=\begin{cases} Z_{i,j} & \textit{für } (i,j)\in b,\\ 0 & \textit{sonst.}\end{cases} \tag{1.17}$$

b) Seien $b_1,b_2\in P$ zwei disjunkte Indexblöcke und $Y\in\mathbb{R}^{b_1}$, $Z\in\mathbb{R}^{b_2}$. Dann wird die Summe $M:=Y|^{I\times J}+Z|^{I\times J}$ auch als Agglomeration *von Y und Z bezeichnet.*