Numeric Tensor Framework:
Toward a New Paradigm in Technical Computing

by

Adam P. Harrison

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Signal and Image Processing

Department of Electrical and Computer Engineering

University of Alberta

Abstract

Technical computing is a cornerstone of modern scientific practice. Within technical computing, the matrix-vector (MV) framework, composed of MV algebra and MV software, dominates the discipline in representing and manipulating linear mappings applied to vectors. Indeed, prominent technical computing packages, *e.g.*, MATLAB, revolve around the MV framework. Applying Thomas S. Kuhn's theory of paradigms, the MV framework *is* technical computing's paradigm. One may then reasonably ask whether the MV paradigm imposes significant restrictions on technical computing's practice. This question may be answered by synthesising the literature on widespread and disparate research efforts on frameworks beyond the MV paradigm. Two categories of anomalous practice emerge, namely special linear mappings, *i.e.*, high-dimensional and entrywise linear mappings, and mappings beyond linear, *i.e.*, polynomial and multilinear mappings. To tackle these anomalies, a framework for numeric tensors (NTs), *i.e.*, high-dimensional data invested with arithmetic operations, proves well-equipped. The proposed NT framework uses an NT algebra that exploits and extends the storied Einstein notation, offering unmatched capabilities, *e.g.*, N-dimensional operators, associativity, commutativity, entrywise products, and linear invertibility, complemented by distinct ease-of-use. This expressiveness is comprehensively supported by innovative NT software, embodied by open-source C++ and MATLAB libraries. Novelties include a lattice data structure, which can execute or invert any NT product, of any dimensions, using optimised algorithms. Regarding sparse NT computations, which are essential to address the curse of dimensionality, the software takes new approaches for data storage, rearrangement, and multiplication. Moreover, the software performs competitively on representative benchmarks, matching or surpassing leading competitors, including the MATLAB Tensor Toolbox, NumPy, FTensor, and Blitz++, while providing a more general set of arithmetic operations. To illustrate these contributions, two original problems from computer vision are solved using the NT framework. The selected exemplars, concerning image segmentation and depth-map estimation, involve high-dimensional differential operators, linking them to the partial-differential equations found in countless other disciplines. Returning to Kuhn, the contributions of this thesis, literature review included, help make a case that technical computing is experiencing a revisionary period. As such, the NT framework, with its expressive algebra and innovative software, represents a timely and significant contribution to the evolution of technical computing's paradigm.

Preface

Chapter 6 highlights three exemplars of the numeric tensor framework. Portions of the chapter text and figures have been previously published as follows:

- A. P. Harrison, N. Birkbeck, and M. Sofka, "IntellEditS: Intelligent Learning-Based Editor of Segmentations," in *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2013*, ser. Lecture Notes in Computer Science, K. Mori, I. Sakuma, Y. Sato, C. Barillot, and N. Navab, Eds. Springer Berlin Heidelberg, 2013, vol. 8151, pp. 235-242;

- A. P. Harrison and D. Joseph, "Maximum Likelihood Estimation of Depth Maps Using Photometric Stereo," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1368-1380, 2012;

- A. P. Harrison and D. Joseph, "Depth-Map and Albedo Estimation with Superior Information-Theoretic Performance," in *Image Processing: Machine Vision Applications VIII*, ser. Proceedings of the SPIE, E. Y. Lam and K. S. Niel, Eds. SPIE, 2015, vol. 9405, pp. 94050C-94050C15.

For all published contributions, I was responsible for the manuscript composition, writing, and editing, as well as the software development, data collection, and data analysis. For the IntellEditS paper, N. Birkbeck contributed toward software development, and manuscript composition and editing. M. Sofka was the supervisory author and contributed to concept formation, manuscript composition and editing, and data analysis. For the depth-map estimation papers, D. Joseph was the supervisory author and contributed to concept formation, manuscript composition and editing, and data analysis.

Acknowledgements

I am indebted to many people, but foremost among them is my supervisor, Dr. Dileepan Joseph. With my MSc included, Dil and I have been working together, off and on, for 8 years running, a total that I think sneaked up on both of us. Dil is never daunted by any scientific question and approaches its resolution with creativity and aplomb. In tackling a problem, Dil typically begins from first principles, giving him a deep and firm base of understanding that he can use to launch into new explorations. Dil has allowed me to join these explorations, providing tutelage, encouragement, and, when necessary, prodding. To Dil I owe the idea of framing the numeric tensor framework using Kuhn's theory of paradigms, which, apart from its insightful value, has been great fun articulating. On an interpersonal level, Dil *always* treats all his students and employees with great respect, drawing upon an astonishing reserve of patience and goodwill. These are just a few of Dil's great qualities that I hope to emulate going forward. Thank you Dil for making me a fellow traveller in your research and for being such a great role model.

I have also been blessed with some other fantastic mentors. In particular, I've relied on Dr. Martin Jägersand and Dr. Dana Cobzaş for advice, encouragement, and the occasional ski. Martin and Dana also ran several reading groups in computer vision and medical imaging, and I cannot emphasise enough how grateful I am, as it gave me much valued exposure to topics outside the confines of my PhD work. I also want to thank Dr. Pierre Boulanger for being incredibly generous with his time, offering me advice and insight, despite his own busy schedule, regarding my PhD work and on research opportunities post-PhD.

I must also mention Dr. Neil Birkbeck, who has been another role model, although he would probably scoff at being labelled as such. Neil, who is also my very good friend, urged me to apply for an internship in medical imaging analysis at Siemens Corporation, Corporate Technology, where he was working at the time. The experience altered the course of my professional life. I appreciated learning from his example, relished our many conversations on subjects ranging from fixed-gear bicycles to discriminative learning, and also enjoyed trying to keep up with him and his wife, Leslie, on the bike (but never on the skateboard).

While at Siemens, I worked under Dr. Michal Sofka, in the Image Analytics & Informatics group. Michal gave me the opportunity to conduct medical imaging research, opening the door to exciting new career prospects and teaching me a ton along the way. He also trusted me enough to explore my own solutions to some of the complex problems being tackled at Siemens. For that I am indebted, and I look forward to finally grabbing that beer in Prague. I am also grateful to all the other research scientists and interns in the Princeton office who helped make my time there such a blast.

Within the Electronic Imaging Lab at the University of Alberta, I want to thank Orit, Ali, Kamal, Cindy, Jing, Erika, and Maikon for all the coffee, laughs, and tasty potluck contributions during my time there. Many of you have helped mentor me throughout the years, particularly when I was completely new to graduate studies. For those who have graduated and/or moved on, it's been wonderful to see you all find success in your varied and exciting careers.

Outside of academia, my friends have helped make Edmonton and the great province of Alberta a cherished part of my life. In particular, I want to stress how important all the shared outdoors pursuits have been to my time here. Finishing my PhD and moving on from Edmonton is bittersweet because it means we'll have to say good-bye. To all my friends, you have taken me to vistas, mountainsides, locales, and occasionally levels of exposure I never dreamed of reaching. And thank you for keeping the friendship alive while I was cloistered in a lab, library, or coffeeshop these last few, I mean numerous, months. To many more adventures!

I want to also thank my parents for all their support. Dad, thank you for sharing the drive from Ottawa to Edmonton with me, and for all the planned and unplanned camping experiences we've had. Thank you also for instilling a love of history, which has kept me grounded throughout the years, despite our select discussions of who borrowed what book from whom. Mom, thank you for your constant belief and confidence in my own research abilities, and for always being ready with much-needed advice drawn from your own experiences as a highly successful academic researcher. It was you who got me started on this track, and yes, I do promise to always think about knowledge translation going forward. Thank you also to my grandmother, my siblings and their spouses, and my nieces and nephews. We may be living apart, but you are always in my heart. It was visits with you that gave me much-needed boosts of energy and motivation to keep chipping away at my PhD.

I will end by acknowledging Kara, my stalwart and beautiful companion. You've been my rock. It was your support, encouragement, and well-placed kicks in the rear, to either keep working or to take a break and go for a run, that got me through this last long push. In so many ways, you set an example for me to live by. Thank you for that, and for so much more.

Contents

List of Figures

List of Tables

List of Abbreviations

ALS	alternating least squares
APL	A Programming Language
BEP	base edge probability
CO	coordinate
CP	canonical-polyadic
CRTP	curiously reoccurring template pattern
CSC	compressed sparse-column
CSCNA	compressed sparse-column no-accumulator
CSE	computational science and engineering
CSL	C++ Standard Library
CSR	compressed sparse-row
CSRNA	compressed sparse-row no-accumulator
CTF	Cyclops Tensor Framework
DCSC	doubly-compressed sparse-column
DCSR	doubly-compressed sparse-row
DSEL	domain-specific embedded language
DSL	domain-specific language
EMV	extended matrix-vector
FD	finite-difference
GLS	generalised least-squares
GP	generic programming

GUI	graphical user interface
HOSVD	high-order SVD
IID	independent and identically distributed
LCO	linearised coordinate
LI	linearised index
LML	linearised maximum likelihood
LSD	least-significant digit
MDT	multiplication datatype
ML	maximum likelihood
MP	Moore-Penrose
MSD	most-significant digit
MTT	MATLAB Tensor Toolbox
MV	matrix-vector
NML	nonlinear maximum likelihood
NNZ	number of non-zeros
NT	numeric tensor
OOP	object-oriented programming
PDE	partial-differential equation
PITAC	President's Information Technology Advisory Committee
PSCP	parametric subclass pattern
RP	radix permutation
RW	random walker
SIAM	Society for Industrial and Applied Mathematics
SNLS	separable nonlinear least squares
SOP	simple outer-product
SSE	sum-squared error

SSKC	Steeb and Shi's Kronecker classes
SVD	singular value decomposition
TCE	Tensor Contraction Engine
TMP	template metaprogramming

Chapter 1

A Paradigm Examined

The advancement of technical computing's role in science and engineering can follow many different avenues, including improving the capacities of current modes of theory and practice. But any serious examination of the state of technical computing should also be willing to investigate the fundamental and perhaps unspoken assumptions, rules, and methodologies employed in the discipline. Doing so requires stepping back and leveraging the insights of those who offer serious commentary on the nature of science and its development. This helps to bring into relief the structure of technical computing. A proper study of the discipline's structure entails an examination of anomalies, or limitations, aiding the investigator to hazard how the field could, and arguably should, progress.

1.1 Scientific Paradigms

The focus of this work is on critically examining some fundamentals of technical computing, with the purpose of outlining promising work and ideas that run counter to the established mode of theory and practice. To do so, we draw upon the views of Thomas S. Kuhn and his notion of paradigms, using them to discuss technical computing and how it fits within the larger scientific and engineering world.

1.1.1 Kuhn's Thesis

Thomas S. Kuhn's book, *The Structure of Scientific Revolutions* [1], hereafter called *Structure*, has arguably had the greatest influence out of any other tract on the current understanding of scientific progress. Even without accepting Kuhn's views on incommensurability and world change, as has been done by his critics [2,3], his seminal work provides invaluable analysis on how science is practised. Most importantly, Kuhn's explanation of paradigms illuminates how scientific communities elevate certain questions as priorities to solve, while overlooking or even disregarding others.

Kuhn also differentiated between two modes of science. The typical mode he called *normal science*, which essentially sets out to solve, clarify, or improve solutions to scientific questions the community has deemed important. The atypical mode he called *extraordinary science*; however, we prefer the term *revisionary science*. This mode can be considered a period of transition triggered by some form of scientific tension or crisis, which causes practitioners to question some of their previously accepted assumptions and tenets. As this crisis is resolved, new scientific questions begin to supplant the old ones. These new questions are accompanied by new theories, techniques, instruments, and experimental methods.

According to Kuhn's early writings, periods of normal science are defined by what type of exemplar problems are used to illustrate the theory and practice of the discipline. Kuhn viewed exemplars as the fundamental means by which scientific communities coalesce around a set of questions to solve. When a crisis occurs, it is resolved through a scientific revolution, whereby new exemplars are chosen as representative of the discipline's mode of practice. While these views enjoyed longstanding impact, Kuhn later amended his thinking of scientific practice and change for several reasons. These are worth outlining as they make clear in what sense we use Kuhn's terminology for our own purposes.

For one, as Kuhn himself admits, his use of the word "paradigm" in *Structure* was ambiguous and engendered confusion [4]. In addition to using the word paradigm in the same sense as *exemplar*, Kuhn employed another more general meaning of paradigm, which he defined as "disciplinary matrix" [4]. A disciplinary matrix includes all the techniques, instruments, theories, questions, and exemplars that a scientific community employs in its practice. We believe this latter definition is more in line with current popular understanding of the meaning of paradigm. As a result, despite Kuhn's later preference, we intend the reader to understand disciplinary matrix when we refer to paradigm.

In addition to changes surrounding the definitions of exemplar vs. paradigm, the role of the former underwent an important amendment. Kuhn deemphasised the role of exemplars by revising his definition of scientific revolutions to mean changes in lexicon or taxonomy [5]. By taxonomy, Kuhn refers to how a scientific discipline categorises objects, theories, and phenomena. Thus, taxonomy, and not exemplars, assumed the mantle of defining the bounds and limits of a scientific discipline. With a different taxonomy in place, crucial aspects of scientific practice, including what questions are important to answer, fall into place. Exemplars then play a diminished, albeit still foundational, role within a paradigm.

A final important amendment by Kuhn is related to the concept of scientific change expressed as revolution. For one, both the graphic terminology and dramatic historical examples Kuhn used to illustrate his point can obscure his oft-repeated assertion that paradigm shifts may be slow moving processes confined to a small community of specialists. As well, Kuhn's later writings outlined another important response to scientific crises—that of scientific specialisation [5]. In this type of response, only a subset of the scientific discipline in question adopts a new taxonomy. Even so, a new lexicon is still adopted by a group of scientists, meaning that regardless of how a crisis is resolved, taxonomic changes, along with corresponding paradigm changes, will result. Thus, for both notational and definitional reasons, we prefer paradigm shift over revolution, where the former can refer to the results of both revolutions and specialisations.

Kuhn's reexamination of scientific structure and change were crucial in addressing logical and historical problems in *Structure*. Nevertheless, the implications of Kuhn's arguments to practicing scientists have remained remarkably constant. By illuminating the social aspects of scientific practice, Kuhn's work can guide practitioners toward critically examining their own disciplines. In particular, Kuhn's works emphasise that one should examine the questions a discipline deems appropriate and also what it considers anomalous based on its taxonomy. This focus is the result of Kuhn's insight that paradigm shifts are not necessarily predicated on solving existing unanswered questions, rather they are predicated on presenting new important questions to consider, which often arise by re-categorising anomalies in the old lexicon as kernels in the new one. As well, a new paradigm must offer enough promise that these new questions can be answered. Importantly, since Kuhn disavowed the cumulative view of scientific progress, this means that individual practitioners must choose to adopt a different paradigm, elevating the influence of persuasion toward effecting change. Thus, practitioners must also be convinced that these new questions are significant and worth pursuing.

Consequently, if one wishes to advocate for a change of theory and practice, one must identify the prevailing paradigm or disciplinary matrix, and just as importantly its anomalies. These anomalies must be shown to have serious impact, and their resolution must be shown to promise significant benefit. Finally, a new disciplinary matrix, or components thereof, must be shown to have great promise in tackling these anomalies.

1.1.2 Third Pillar of Science

Computing continues to exert an increasingly greater role in science and engineering, which has led to the creation of new fields and terms and also the redefinition of old ones. Some of these terms include numerical methods, numerical analysis, applied mathematics, computational science and engineering (CSE), and of course, the subject of this work, technical computing. These terms are used in different manners and their exact definitions and boundaries are not settled. To avoid confusion, we offer our own views on these fields and their place in today's science and engineering community.

The ubiquity of computers in science and engineering has progressed to a point where a new field called CSE has emerged at the forefront of scientific policy. In a 2005 US Report to the Executive [6], the President's Information Technology Advisory Committee (PITAC) defined CSE as "a rapidly growing multidisciplinary field that uses advanced computing capabilities to understand and solve complex problems." The PITAC report was motivated by the increasing importance of CSE in all realms of science, spurring the committee to label CSE as the "third pillar of science," alongside theory and experimentation.

The PITAC's definition of CSE is very broad, as the committee included algorithms and

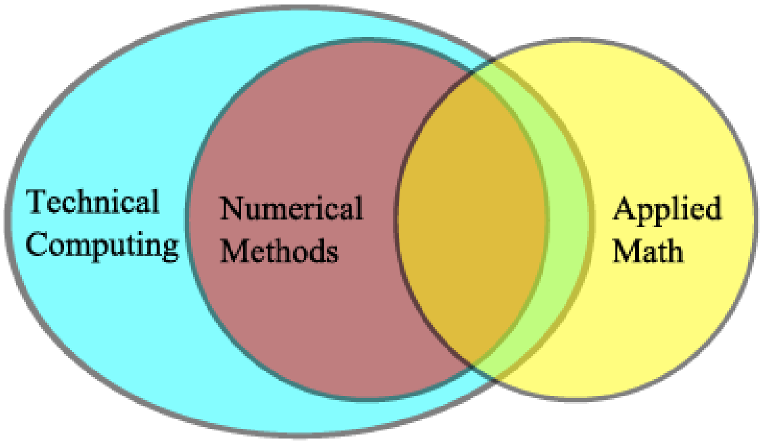


Figure 1.1: Venn diagram of technical computing. Technical computing employs the concepts, theory, and techniques of applied mathematics and numerical methods.

modelling, advanced computer infrastructure, and the information technology expertise to manage and optimise said infrastructure [6]. We define technical computing as embodying the first element of the PITAC's definition, *i.e.*, algorithms and modelling for the purposes of CSE. We use the term technical computing instead of scientific computing as we feel industrial applications deserve their place alongside their scientific counterparts, and the term *technical* encompasses both types of applications. This reflects the large contribution of industrial giants, like IBM, toward the development of computing [7] and also the fact that industrial and engineering applications are major drivers in advancing technical computing.

Examining the algorithmic and modelling purview of technical computing more closely, we view the discipline as encompassing all of numerical methods and parts of applied mathematics. The former focuses on developing and applying algorithms in a environment that is inherently faced with approximation errors, finite resources, and noisy data input. This describes the environment and challenges that technical computing must operate under. Equally important to algorithms, especially in a scientific or engineering setting, one must have the mathematical tools and methods to model and solve problems. Thus, the tools and methods of applied mathematics represent an integral aspect of technical computing. Technical computing also encompasses other aspects of computing not found in applied math or numerical methods, such as programmatic control structures, graphical user interfaces (GUIs), optimisation techniques, and memory management. The general makeup of technical computing is depicted in Figure 1.1.

The prominence of computing in science and engineering has not been welcomed in all quarters. For instance, when CSE was still an emerging trend, the influential applied mathematician Clifford Truesdell offered a viewpoint encapsulated by the title, "The Computer: Ruin of Science and Threat to Mankind" [8]. While Truesdell's essay can verge into the polemic (in one paragraph, he applies a metaphor comparing the coming predominance of computers in science to the Third Reich), he does provide a valuable and prescient critique of CSE. In particular, Truesdell warns that computers present a great temptation toward blindly applying and accepting computational techniques to problems. Several of these warnings are echoed in more recent publications. For instance, a 2010 article in *Nature* provides a modern account of problems arising due to the use of CSE by scientists lacking in requisite expertise [9]. Many of these dangers are accepted and recognised by proponents of CSE, who themselves emphasise the importance of both computational and domain expertise in order to avoid the dangerous pitfalls Truesdell mentions [6, 10, 11, 12].

As the importance of CSE increases, there is an impetus to advance the tools and techniques used in the discipline. Since technical computing represents a major component of a push toward better and more intelligent use of computing in science and engineering, the continued effort must include examining how problems are modelled and solved numerically. This aligns well with the PITAC's observation that:

> "...our preoccupation with peak performance and computing hardware, vital though they are, masks the deeply troubling reality that the most serious technical problems in computational science lie in software, usability, and trained personnel." [6].

If one takes concerns like Truesdell's seriously, then one accepts that technical computing practitioners should be continually on guard against the blind application of their art. Thus, critical examinations of the field should be considered important activities in the advancement of technical computing's role in science and engineering. Kuhn's commentaries on scientific progress can provide valuable insights for such an examination. But first, it is essential to address whether Kuhn's views apply to computing. This is important, as Kuhn's writings typically focused on the physical sciences [13], and do not specifically discuss computing.

Mirroring Kuhn himself, authors in computing have used the word paradigm in different and ambiguous ways. In some instances, authors use the word paradigm somewhat casually, without relating their use to Kuhn's conceptions of exemplars, world-view, or taxonomy. For instance, programming styles and practices have been labelled paradigms with little to no reference to whether any of the criteria of being a paradigm are met, *cf.* Tsai [14] and Colburn [15, 16]. However, Floyd provides a laudable exception to this, using Kuhn's writings to examine how "programming paradigms" propagate and affect how practitioners view computer science problems [17].

Outside of programming practice, authors have identified several paradigms within computing. For instance, Tedre and Sutinen have outlined the stored-program paradigm as the major defining characteristic of modern computing [18]. Denning and Freeman offer their modern and more general notion of computing's paradigm based on information processing [19], which aligns well with Colburn's take on computer science progressing from data processing to a more information-oriented approach [15]. Finally, Tedre points out that the master-apprentice relationship common in computer science epitomises the use of exemplar as a tool for knowledge translation [20].

Authors have also used paradigms to frame the debate on whether computer science is a branch of mathematics [15, 16, 21]. This framing is informative as it affects one's viewpoint on the role of computer science and which questions to pursue. For instance, the pursuit of formal verification tools looms large to those that subscribe to the mathematical paradigm of computer science [15, 16, 21, 22], but it is largely a neglected question to those that do not. Moreover, the debates between practitioners within the mathematical or non-mathematical paradigm have oftentimes been acrimonious, *e.g.*, the caustic back-and-forth between Fetzer and the formal verification community [23], indicating that paradigms in computer science, just as with other disciplines, come imbued with the passions and investments of its adherents.

Apart from the identification of paradigms in computing, it is also useful to point out instances where new developments were met with resistance, but ultimately led to a fundamental change in practice. Doing so suggests that progress in computing is not always cumulative, just as with the scientific disciplines of Kuhn's focus. For an early example, in the 1940s most computing machines were electro-mechanical and the dominant voices in the field opposed the first fully-electronic computer [18]. As evidenced by the architecture of modern computers, eventually practitioners completely accepted the fully-electronic architecture. A later example in terms of software can be found in Backus' reminiscences of the resistance or indifference of machine-code programmers toward the development of algebraic programming languages like FORTRAN [24]. Contrast this with today's programming practices, where it is now common to program microcontrollers without resorting to assembly languages.

In addition, it is important to highlight that progress in computing, just as with the larger scientific world, can often pivot on the human element and may rely on elements of persuasion. For instance, when discussing FORTRAN, Knuth and Pardo describe the arrival of "a language description that was carefully written and beautifully typeset, neatly bound with a glossy cover", as a major first in the history of programming and an important agent of change [25]. Narrowing the focus to technical computing, Parlett asserts that Wilkinson's undeniable impact to technical computing was due in part on his ability to make error analysis accessible and interesting [26]. These examples indicate that Kuhn's insights are capable of shining a light on the structure of computing-focused disciplines, which includes technical computing.

1.2 History and Structure

Kuhn's insights into scientific progress derived from his chosen practice of examining scientific history and deriving conclusions from this study [5]. When investigating technical computing, its history also provides insights. In particular, the development of technical computing helps illuminate two features that have served as cornerstones within technical computing's disciplinary matrix—algebra and software. This recognition leads to the argument that matrix-vector (MV) algebra and software constitute a paradigm in technical computing today.

1.2.1 Algebra and Software

Formalism and constructivism take on prominent roles within practical applications of mathematics. Technical computing is no exception, and we identify algebra and software as the technical computing embodiments of formalism and constructivism, respectively. The veracity of this description can be revealed through a brief examination of technical computing's history.

Unlike more modern times, where computers enjoy ubiquity in innumerable end uses, from mission-critical to social-media applications, the early history of computing is essentially synonymous with the early history of technical computing [27]. The US Army's demand for faster ballistic calculations spurred the first fully-electronic Turing-complete computer [18]. The need to numerically solve physical problems that did not lend themselves to closed-form solutions, *e.g.*, turbulent flow, local weather predictions, and most (in)famously physical problems related to the atomic bomb, encouraged further development of computing [28].

The notion of computability was a topic borne out of the constructivist school of mathematics. Those within this school only accept mathematical objects that they can logically construct [29]. Even though their work focused on logic and commenced prior to the advent of computers, emerging from the constructivist camp were hugely influential figures, such as Gödel, Turing, and Church, who contributed to efforts in defining computatability and formalising the concept of algorithms [29, 30]. These discussions on the nature of algorithms and computability were particularly influential in the realm of computer science.

Despite the impact of heroic constructivist figures to the development of computer science, today constructivists remain small in number [29, 30, 31]. Gurevich argues that constructivists created their own type of pure mathematics, losing sight of the important questions of efficiency and feasibility [31]. Or in other words, constructivists were not "sufficiently constructive" [31]. This echoes arguments of those who hold the engineering view of computer science, as opposed to the mathematical or scientific view, that practical and tangible results are the reasons for the discipline's extraordinary impact [32]. Eriksson *et al.* have deemed computer science and numerical analysis the inheritors of the constructivist tradition [30].

These discussions on the impact of mathematical constructivism on early computing reflect two different recurrent and symbiotic threads in computing. First, is the need to formalise computing algorithms for the purposes of "expressiveness, program readability, and algorithm provability" [33]. Second, one must have a means to constructively perform the algorithm using finite computations that are feasible and practical [31]. These two needs can be called formalism and constructivism.

May, the mathematical historian, has described the two needs of formalism and constructivism as being part of a larger mathematical tradition, stating that "there are two great traditions in mathematics: the scientific tradition and the technological tradition, mathematical science and mathematical technology" [34]. May argues that the development of mathematical technology has been ignored by scholars. May continues by contending that the computer has now turned mathematical technology into a recognised discipline. Ulam echoes this sentiment, maintaining that the development of computers was made possible by the merging of formal logical systems and technological development [28].

A major milestone in the evolution of programming was the first algebraic compiler. Programming now had a much more mathematical flavour. This was met with indifference at the time, but the efforts culminated in the development of the FORTRAN language [24]. FORTRAN was developed to skirt the common trade off at the time between easy coding and efficient programs. Its notation was designed to be mathematical and independent of machines. Interestingly, it is the first to allow variable names longer than one letter, breaking mathematical convention [25], but illustrating the unavoidable fact that mathematical technology often has needs separate from those of mathematical science. Control statements, which predated FORTRAN, are another prominent example of this. However, despite making "programming an activity akin in rigour and beauty to that of proving mathematical theorems" [35], these so-called algebraic languages only gained acceptance after demonstrating performance comparable to the status quo of low-level assembly languages [24].

Thus, formalism and constructivism each play their own crucial role in the progress of computing technology. Today, in the larger computer science world that has broken free from the confines of technical computing, formalisms, *i.e.*, programming languages and coding idioms, are still the subject of passionate debate, as the choice affects practical aspects such as programming efficiency, computational efficiency, readability, and maintainability. Of course, constructivism continues to play a dominant role in computer science, as witnessed, for instance, by the huge efforts undertaken to optimise compiled programs, parallelise areas of bottleneck, and vectorise machine code. The modern state of computer science synchronises well with Denning and Freeman's view of computing's paradigm consisting of "expressions that do work" [19].

Returning to technical computing, while these considerations also apply, the discipline must also meet the needs of scientific and engineering problems. As mentioned in Section 1.1.2, technical computing comprises both modelling and algorithms, which are addressed by formalism and constructivism, respectively. Thus, technical computing provides its own unique context for the interplay between formalism and constructivism. Reflecting this unique interplay, within technical computing algebra and software take on the role of formalism and constructivism, respectively. A technical computing algebra should align well with the scientific notation used to model the problem(s) under question. The algebra should also be conceptually efficient, *i.e.*, requires minimum time and effort for a practitioner to conceptualise a problem and express its solution. However, it should also foster programming efficiency, *i.e.*, requires minimum coding time, effort, and chance of error. As well, the supporting software must be able execute the operations demanded by the algebra with large-scale and numeric data. Issues like efficiency, numerical stability, and sensitivity affect the feasibility and reliability of the implementation.

The crosstalk between formalism and constructivism is so potent that Åhlander *et al.* advocate choosing technical computing mathematical abstractions through the lens of software engineering considerations [36]. This argument is echoed by Eriksson *et al.*, at Chalmers University of Technology, and Estep, at Colorado State University, who argue that constructivism and formalism inform each other [30,37,38]. These researchers use the metaphor "body and soul" to describe the makeup of *applied math*—formalism represents the soul while constructivism serves as the body. We adopt the same metaphor in describing the makeup of technical computing, where formalism and constructivism manifest as algebra and software, respectively.

Establishing the role of algebra and software provides a valuable stepping stone from Kuhn's work to technical computing. This is important because Kuhn never specifically addressed computing [13]. In *Structure*, Kuhn emphasised the role of theories, instrumentation, laws, and applications within a paradigm [1]. From this nomenclature, an analogy can be made from Kuhn's terminology of theory and instrumentation to Eriksson *et al.*'s terminology of formalism and constructivism, respectively. The analogy can be extended to also describe algebra and software within technical computing.

While allowing us to link Kuhn's terminology to technical computing, this analogy does reveal an important difference from *Structure*. Kuhn mostly focused on theoretical traditions of science and seemed to view theory as playing a primary role within a paradigm [13]. However, this may not be an accurate view for instrumental-heavy domains like technical computing. In particular, we do not view software as subordinate to algebra within the discipline. Non-theoretical innovations, *e.g.*, improved algorithms, play an enormous role in the progression of the discipline. If we use Kuhn's terminology, this means that within technical computing we explicitly place instrumentation, *e.g.*, software, on an equal footing with theory, *e.g.*, algebra. This explicit designation is an important distinction from the description of science found within *Structure*.

1.2.2 Matrix-Vector Paradigm

When discussing the partnership between algebra and software, the combined solution may be called a framework. Since, practitioners use a framework to cast scientific problems in a mathematical language and perform computations within that context, if technical computing has a paradigm then it will be identified by answering whether practitioners have predominantly chosen one framework. With this in mind, an examination of the impact of MV algebra and software, from technical computing's formative years to its present state, reveals the crucial role the MV framework exerts.

Underscoring the formative role of the MV framework, the publication of von Neumann and Goldstine's 1947 paper [39], "Numerical Inverting of Matrices of High Order", is often credited as the starting point of modern numerical analysis [26]. As Goldstine put it, "One of the first and most likely topics to be discussed [during the early years of technical computing] was the solution of large systems of linear equations, since they arise almost everywhere in numerical work" [40].

Many of the initial pushes were toward fast and efficient softwares for MV computations. One of the first efforts to produce reliable and reusable mathematical software routines was Bell Telephone Laboratories' *Numerical Mathematics Program Library* [41]. Its initial submission focused on a key MV problem, providing code to produce eigenvalues of nonsymmetric matrices [42]. In the mid 1970's LINPACK and EISPACK were arguably the successors of this work [43], offering reliable and fast MV computations that formed the core to many technical computing applications. Equally impactful, the low-level BLAS subroutines for MV multiplications have been the bedrock for innumerable computing routines. In the 1990's, LAPACK emerged onto the scene, effectively supplanting LINPACK and EISPACK [44].

Along with pushes toward better MV software, practitioners were also consciously thinking of the benefits of integrating mathematical algebra into programming practice [25]. As early as the 1950s, pioneers of the programming art were pushing for this. In fact, Backus and Herrick even advocated for matrix-multiplication-like operations in programming languages, but interestingly chose to express it using summations: "to go a step further [the programmer] would like to write $\sum a_{ij} \cdot b_{jk}$ instead of the fairly involved set of instructions corresponding to this expression" [45]. Modern languages that support operator overloading, *e.g.*, C++, have been able to provide an environment that supports MV algebra, with underlying routines hidden to the software user.

Backus and Herrick's forward thinking even went so far as to express the wish that, one day, "a programmer might not be considered too unreasonable if he were willing only to produce the formulas for the numerical solution of his problem and perhaps a plan showing how the data was to be moved from one storage hierarchy to another and then demand that the machine produce the results for his problem" [45]. Arguably, the advent of very high-level languages for technical computing, such as MATLAB, have come close to fulfilling this vision. Such modern computational packages have embedded MV capabilities within their key datatypes. This allows implementations to be programmed using the natural language of a wide variety of problems, expediting the process of transforming theory into practice, as the coding process resembles the symbolic formulation to a great degree. As well, given sound underlying routines, it allows scientists and engineers to employ technical-computing techniques without being experts in the highly specialised field of MV computations. This reduces implementation errors. The influence of MATLAB has grown to the point where it can boast of over one million users [46], leading some authors to maintain it is the most popular computing environment for technical computing [47, 48].

As MATLAB, and technical-computing softwares like it, continue to solidify their place in the field, the importance of the MV framework has only increased. In fact, the abilities and popularity of the MV framework have even led authors to reframe well known and established techniques. For instance, the Society for Industrial and Applied Mathematics (SIAM) has published books that reframe the Fast Fourier Transform [49] and graph algorithms [50] using MV algebra. Much of the motivation behind recasting these techniques lies in the ubiquity and expressive power of MV formalism. However, constructivist considerations also apply. For instance, employing the MV framework to model and solve graph algorithm problems was only made practical by the advent of efficient sparse matrix computations and data structures [51]. Thus, both the algebraic and software components of the MV framework are major players in technical computing today.

In viewing the influence of the MV framework and the evolution of its use in technical computing, it is also striking to consider how much has remained constant. In his 1967 survey of the role of linear algebra in technical computing, George E. Forsythe makes several statements that could have been said today. His views on the predominant use of MV algebra for modelling scientific and engineering equations still rings true. As well, his statement that the "the amount of literature on matrix computations is staggering" has only become more accurate. Most importantly, Forsythe's survey outlined the crucial (and sometimes unrecognised) role MV software fills in its partnership with MV algebra [52].

Due to the influence of the MV framework, it is valid to ask whether it defines technical computing's paradigm. For certain, the MV framework remains a linchpin in the disciplinary matrix of technical computing. However, the MV framework plays an even greater role than that. Since it is the dominant means by which practitioners model and solve problems, the MV framework is key in defining what sort of scientific questions technical computing can tackle, *i.e.*, linear mappings applied to vectors[1]. In doing so, it also defines the anomalies technical computing faces today, *i.e.*, phenomena not easy or impossible to incorporate within the MV framework. As such, the MV framework plays a crucial role in defining the taxonomy of technical computing. Thus, even though technical computing contains components outside the MV framework, *e.g.*, programmatic control structures, GUIs, and data input/output, we label the current paradigm in technical computing as the *MV paradigm*.

[1]One apparent exception are the well-worn linear algebra factorisations, which are solutions to quadratic equations, *e.g.*, QR, LU, and SVD. However, the purpose of these factorisations is still to characterise or manipulate linear mappings.

1.3 Anomaly Categories

Kuhn's view was that any scientific paradigm faces anomalies. Indeed, the MV paradigm faces a number of them, which we consider significant. These can be identified by first starting from the MV paradigm's bastion of strength. As Rauhala notes, "the tools of linear algebra have been centered in solving for a linear system... where the parameters... and observed values are only one-dimensional vectors" [53]. In short, with the exception of bilinear or quadratic forms, MV algebra is largely used to solve equations that arise from a basic linear mapping:

$$\mathbf{y} = \mathbf{A}\mathbf{x}, \tag{1.1}$$

where both $\mathbf{x}$ and $\mathbf{y}$ are one-dimensional vectors. Nonetheless, the MV paradigm's narrow but powerful focus works against it when one confronts equations that arise from other mappings. Key areas outside its strengths include certain types of linear mappings, *e.g.*, N-degree (N-dimensional)[2] and entrywise mappings, and mappings beyond the linear type, *e.g.*, polynomial and multilinear mappings.

1.3.1 Special Linear Mappings

MV algebra's natural strengths include its ability to model linear relationships. However, important types of linear relationships fall outside its natural scope. These include N-degree and entrywise mappings. Both categories of linear mappings play significant roles in technical computing.

For instance, high-degree structures naturally arise when representing in the discrete domain physical phenomena that naturally occur over a space of two or more dimensions [54]. MV algebra cannot naturally represent third-degree or higher phenomena. Even when the phenomena in question is only second-degree, representing all possible linear mappings of such phenomena requires a fourth-degree structure. Image processing and computer vision provide excellent examples of this, as practitioners are often interested in solving linear systems applied to second or third-degree data, *e.g.*, depth-map estimation [55] or image segmentation [56].

As Åhlander notes, "Multidimensional arrays are used extensively in scientific computing" [54]. Even introductory books on numerical methods, *cf.* Chapter 10 of Heath [47], Chapter 29 of Chapra and Canale [57], and Chapter 9 of Schilling and Harris [58], discuss how to construct linear mappings of second-degree partial-differential equation (PDE) domains. Hence there is a need for methods that can handle N-degree data, even when one is only interested in working with linear mappings.

In order to continue working within the MV paradigm, practitioners employ vectorisation operations, which flatten an N-degree structure into a first-degree one. Prominent

[2]Our reasoning for this terminology is given in Section 2.1.6.

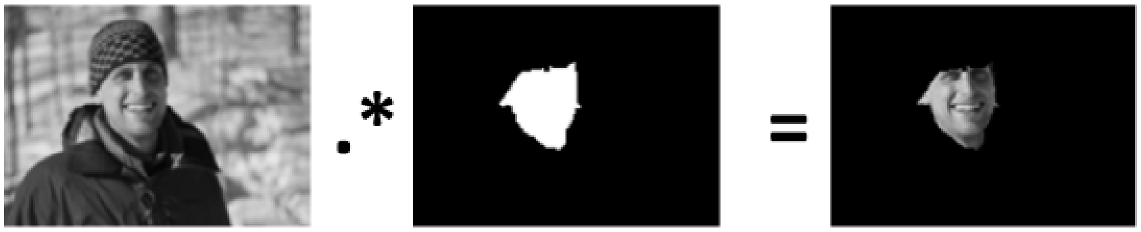


Figure 1.2: Entrywise nature of image masking. Image masking represents one of many common programmatic operations that are inherently entrywise. The mask shown is generated using Grady's random-walker method [56].

uses of such operations include casting the high-degree linear PDEs mentioned above into the MV paradigm. Vectorization possesses strong links with Kronecker algebra [59], which impart tensor product operations to MV algebra. While the Kronecker product can be used to describe bilinear operators, it can also be used to construct a linear operator acting upon a flattened second-degree tensor, as highlighted by several authors [53, 60, 61, 62]. When working within the MV paradigm, flattening N-degree data is necessary because "The technique of transferring to the vector domain in order to operate on the two-dimensional data... is the only way in Kronecker algebra to represent all the possible linear operations of an image" [60]. In addition to images, mappings of any other N-degree structure can be included within this statement, *e.g.*, linear high-degree PDEs and high-degree differentials.

While some authors [63] consider Kronecker algebra part of the MV paradigm, the former is not very well known outside of statistics and econometrics [64]. Should one consider Kronecker algebra part of the MV paradigm, it is typically used to work with second-degree, and not N-degree, data. Even in this case, data must be flattened into a different form. The need for algebra and software that can naturally work with N-degree linear mappings has led authors [53, 60, 61, 65, 66, 67, 68] to develop and advocate for tools and techniques outside of the MV paradigm.

Alongside the issue of N-degree linear mappings, another important category of linear mappings not traditionally supported by the MV paradigm are entrywise operations. Such operations hold a particularly crucial role in imaging processing and computer vision, as the data is made up of individual pixels composing an aggregate whole. This means that entrywise operations are often executed, either alone or in combination with image, row, or column-wise operations. For example, as Fig. 1.2 illustrates, applying an image mask is a purely entrywise operation.

Entrywise products are well used in many other fields and in technical computing in general. For instance, MATLAB supports entrywise multiplication with its `.*` operation. However, the operation is not formalised within MV algebra. Entrywise operations can also be found in tensor calculus [69] or applications that use tensor notation [68, 70]. n-ary inner products, which Chapter 3 will show has links with entrywise products, have appeared in

Table 1.1: Frequency of entrywise and N-degree operations in MATLAB toolboxes. An analysis scanned 30 MATLAB toolboxes (2014a release) for arithmetic operations that fall outside of the traditional MV paradigm. Fully commented lines were excluded from the analysis.

Operation	Frequency
All lines	4 469 192
Arithmetic lines	1 633 594
`repmat`	5 246
`(:)`	14 899
`.*`	12 280
`.\` or `./`	10 466
`diag` or `spdiag`	8 205
`permute` or `ipermute`	726
`reshape`	3 993
`kron`	279
Subtotal	56 094
% Arithmetic lines	3.4%

works on analytical dynamics [69], geophysics [71], and signal processing [72].

Due to their utility and prevalence, entrywise operations have been introduced as extensions to MV algebra [73]. Such specialised operations are often included with descriptions of Kronecker algebra, which is discussed above. Thus, whether these operations are part of the MV paradigm is questionable. In any case, such operations do not all generalise to N-degree structures. In addition, apart from `.*`, which is constructively similar but not identical to the Hadamard product, these operations are not supported in major technical computing packages [73].

Multiplying vectors or matrices by a diagonal matrix is another means to incorporate entrywise products within the MV paradigm. While common, diagonal matrix multiplication can only effect a specific type of entrywise product, *i.e.*, entrywise multiplication of a vector, and relies on embedding a first-degree structure into a second-degree one. Thus, the structure of first-degree data must be altered in order to fit entrywise products within the MV paradigm.

To further underscore the role of N-degree and entrywise mappings in technical computing today, one can investigate how frequently arithmetic operations outside of the standard view of the MV paradigm are used in a technical computing context. A good sample of this can be found by examining the MATLAB toolboxes for occurrences of N-degree and entrywise operations. Table 1.1 presents such an analysis, using the MATLAB 2014a release. As the table demonstrates, these operations play a significant role, being involved in 3.4% of all arithmetic statements. Consequently, there are important and well-used arithmetic operations in technical computing that the MV paradigm *does not* support. The role of these

Table 1.2: Techniques to solve various categories of equations. For the most part, the technical computing paradigm is in agreement on how practitioners should formulate and solve the different cases. The exception to this rule are multivariate polynomial equations.

	Univariate	Multivariate
Linear	$ax = b$ Trivial	$\mathbf{Ax} = \mathbf{b}$ Gaussian elimination*
Polynomial	$a_0 + a_1x \ldots + a_nx^n = 0$ Companion matrix	No consensus
Nonlinear and Non-Polynomial	$f(x) = 0$ Root-finding methods	$\mathbf{f}(\mathbf{x}) = \mathbf{0}$ Newton's method*

*and derivatives

operations is further magnified when considering the considerable scientific and industrial end-user base of MATLAB. As a consequence, Table 1.1 supports the notion that N-degree and entrywise linear mappings represent significant anomalies to the MV paradigm.

1.3.2 Beyond Linear Mappings

There is a large swath of mathematical relations beyond linear mappings. Some of these mappings possess a global structure that can be used in solving equations or in data decomposition. We consider polynomial systems and tensor decomposition in particular, which can be seen as occupying the next level in complexity beyond linear equations.

Polynomial Systems

When considering the categories of equations that are tackled by technical computing, they are typically classified by whether the equations are univariate or multivariate and also whether the equations are linear or nonlinear. Table 1.2 illustrates this visually, depicting the categories of equations and also the common approaches toward their solution.

The categorisation of equations into 'linear' and 'nonlinear' is quite broad, reminding us of the well known but unattributed quote: "Classification of mathematical problems as linear and nonlinear is like classification of the Universe as bananas and non-bananas." Polynomial equations, in particular, are an important subclass of nonlinear equations.

As shown in Table 1.2, univariate polynomial equations can boast of their own specialised means of solution, which are classically performed using companion matrices. Stetter argues that this exception is the one case where "algebra and numerical analysis joined ranks to develop efficient and reliable black-box software for the—necessarily approximate—solution of this task" [74]. However, when moving to the multivariate case, there remains no standard way to formulate and solve systems of polynomial equations *numerically*. The numeric caveat to this statement is important to emphasise, as computational algebra has developed powerful Gröbner basis methods [75] toward characterising and solving polynomial equations. Yet, as Stetter notes, the methods of computational algebra are not equipped by themselves to robustly and numerically solve systems of polynomial equations whose inputs are inexact. Nonetheless, polynomial equations possess their own global structure, which is something computational algebra does tackle head-on [74].

In a technical computing context, numerical systems of polynomial equations are tackled in different ways. A common approach is to simply apply standard nonlinear methods, determining a local solution given a good-enough initial guess. Solving polynomial equations this way is epistemologically equivalent to lumping polynomial equations together with more general nonlinear equations. For engineers and scientists wishing to consider the special structure of polynomial equations, there is still no consensus. Practical applications may be solved using one or more of Newton's methods, homotopy, exclusion, eliminant, or Gröbner methods [76]. This state has led researchers like Stetter [74], examining the success of numerical linear algebra, to push for numeric variants of computational algebraic methods of solving polynomial equations. In fact, the Defense Advanced Research Projects Agency in the US has included the following in their list of mathematical challenges [77]: "Beyond Convex Optimization: Can linear algebra be replaced by algebraic geometry in a systematic way?" To us, we read systematic to include numeric and stable implementations for technical computing applications.

Since the only class of polynomials, outside of linear ones, that MV algebra can model are quadratic or bilinear forms, *i.e.*, $\mathbf{xAx} = b$ or $\mathbf{xAy} = b$, the current paradigm is not equipped to naturally handle this important problem. The challenges the MV paradigm faces with polynomials are further underscored by the known connection between homogenous polynomials and symmetric tensors [78, 79, 80, 81], which suggest structure outside of matrices and vectors are needed for this problem. Moreover, as Stetter notes, the numeric constructivism for solving such problems requires considerable further development [74].

In Stetter's analysis of the field he remarks that, "most surprisingly—this challenge of pioneering a 'numerical nonlinear algebra' remained practically unnoticed by the many young mathematicians hungry for success, even by those working in the immediate neighborhood of the glaring white spot" [74]. We disagree in one respect. This is *not* surprising when considering the field through Kuhn's viewpoints. In fact, Stetter's observation fits Kuhn's model very well, where phenomena deemed anomalous in a paradigm are not included in the questions "worth asking," lending weight to the argument of polynomial systems as anomalies in the current technical computing paradigm.

Tensor Decomposition

Matrices, which are second-degree constructs, enjoy a legion of mature and well-known decompositions. However, when considering high-degree data, the decomposition options are not so clear cut and are much more difficult problems [82]. The field that focuses on such problems is called tensor decomposition. Despite having its origins in the 1920s, tensor decomposition is a topic that has only received widespread attention in recent years [83]. While the term tensor is used, in actuality the field is concerned with the decomposition of general high-degree data, which may or may not satisfy the conditions of being a tensor in the mathematical or physical sense [84].

Tensor decomposition has applications in many domains, including signal processing, graph analysis, neuroscience, computational biology, computer graphics, and computer vision, amongst others [82, 83]. Much of the benefits and promise of tensor decomposition ties in with its ability to provide a natural context for multilinear mappings [80,83,84]. Additionally, tensor decomposition can be viewed as a high-degree generalisation of singular value decomposition (SVD) [81]. Reflecting on the huge impact of SVD gives a sense of the promise behind tensor decomposition. For this reason, tensor decomposition is increasingly solidifying its place within technical computing. Underscoring this, the latest edition of Golub and Van Loan's very influential book, *Matrix Computations*, now includes sections devoted to this topic [85].

However, like systems of polynomial equations, there are many fundamental and open questions within tensor decomposition. For instance, there are several different decomposition options. The two most prominent options are canonical-polyadic (CP) and Tucker decomposition, which provide rank-R and orthogonal decompositions, respectively [83]. In contrast, the SVD provides both qualities, suggesting that a generalisation to higher-degrees is not clear cut. Moreover, there is also no universally reliable means to compute CP and low-rank Tucker decompositions. Instead, current approaches are typically variations of optimisation approaches, *e.g.*, the alternating least squares (ALS) algorithm, whose final solution depends on initial conditions and other factors.

Because tensor decomposition is concerned with N-degree data, the MV framework is not in general equipped to represent such data, mappings, or any factorizations applied to them. As Richard A. Harshman, one of the pioneers of the field, put it, "A single two-way array (*i.e.*, a matrix) cannot *directly* represent a three-way 'cubical' array of data, nor can any sum of matrix products *directly* produce a three-way latent structural object" [86]. Hence, from the MV paradigm's vantage point, tensor decomposition remains anomalous. Unless other algebras and softwares gain widespread acceptance beyond researchers focused on tensor decomposition, the consequences of this field being anomalous will begin to loom large over the discipline of technical computing.

1.4 Summary

Technical computing plays a crucial role in science and engineering today, to the point where some label CSE the third pillar of science. Since technical computing plays such a large role within CSE, it is critical that its capabilities are continually examined. By using the terminology and insights of Kuhn's seminal work, *Structure*, we argue that the dominant MV algebra and software, which we call the MV *framework*, represents a paradigm in technical computing.

One of the key observations of Kuhn's work is that paradigms possess a taxonomy, by which certain phenomena act as the linchpin of the scientific theory and practice and other phenomena act as anomalies. Identifying both sets of phenomena is key in outlining the strengths, but also limitations, of the current paradigm. This chapter outlined several anomalies facing the current MV paradigm. Broadly speaking, they can be categorised as specialised linear mappings, *i.e.*, N-degree and entrywise products, and specific nonlinear mappings, *i.e.*, polynomial and multilinear mappings. Both types of anomalies are playing an increasingly important role in CSE.

Kuhn also pointed out that as practitioners increasingly strive to pursue scientific questions touching upon a paradigm's anomalies, they will employ workarounds commensurate with the existing taxonomy or they will develop techniques and tools that imply changes to the taxonomy. These efforts will often be disparate. In the context of technical computing, this means alterations related to the dominant algebra and software. As we will discuss in Chapter 2, this is indeed the case in technical computing, as researchers and practitioners have developed or argued for a host of different algebra and software to tackle the anomalies outlined in this chapter.

Chapter 2

A Growing Crowd

Milgram *et al.* famously demonstrated how the size of a crowd looking up from a street corner influences the likelihood that passersby will join in [87]. Given that Kuhn's writings emphasise the social aspect of scientific progress, we use Milgram *et al.*'s crowd as metaphor for those engaged in revisionary science. Like the crowd at the street corner, as greater numbers of researchers and practitioners look beyond an established paradigm, the likelihood that other scientists and engineers will join this effort will only increase.

Focusing on technical computing, there is a healthy body of researchers employing and often advocating for scientific algebra or software that lie beyond the matrix-vector (MV) paradigm. These researchers come from backgrounds that include numerical analysis [54, 67], applied mathematics [88, 89, 90], computational chemistry [91, 92, 93, 94, 95], image processing [60], computer vision [96, 97, 98], systems and control theory [65, 99], signal processing [66, 72, 80, 100], physics [101], geodesy [53, 61, 62], econometrics [64, 102, 103, 104], database and information systems [105], psychometrics [86] and statistics [84, 106, 107]. Exemplars represent crucial illustrative and persuasive elements within this body of work.

In our view, there is a growing crowd. By reviewing its efforts and arguments, we articulate the salient characteristics that alternative algebras, software, and related exemplars, encapsulate. In doing so, numeric tensors (NTs) are identified as an integral concept that binds many of these efforts together. This insight moulds and informs our own efforts at contributing to this growing crowd.

2.1 Algebraic Characteristics

Practitioners have introduced or advocated for various algebras outside of traditional MV algebra. How far researchers venture outside of MV algebra can range from scientists who only use other algebras when their need (or taste) dictates, to others, like Papastavridis, who contend that "[matrix algebra] is not a daily working tool for the exploration and mastery of new and rugged terrain" [69].

Kronecker algebra represents a major player in technical computing formalisms different than traditional MV algebra. The algebra furnishes MV formalism with a direct or tensor product and typically operates in tandem with vectorising operations, which flatten matrices into vectors [59]. Practitioners have employed Kronecker algebra in signal processing, image processing, mechanics, quantum theory, statistics and econometrics [63, 108, 109]. Apart from formalising the concepts of direct products, practitioners extend MV algebra in other ways. In particular, notions of entrywise products have been applied in the form of Hadamard and Khatri-Rao products. Like Liu and Trenkler [73], we group these entrywise matrix products together with Kronecker algebra, using our own label of extended matrix-vector (EMV) algebra.

Other algebras depart more completely from MV formalism. The n-mode product notation, which works directly with N-degree constructs, represents a popular example and is often used in the tensor decomposition field [81, 83, 110]. Eldén and Savas introduced a similar notation [111], which has also seen use, *cf.* Lim's chapter [84] in the *Handbook of Linear Algebra*. For operations common in tensor decomposition, *e.g.*, multiplying a tensor with a matrix, the algebra supplies dedicated notation that uses numerals to designate which NT index is undergoing an inner product. Less-prevalent conventions exist to express general tensor-times-tensor products [97, 98, 112], again using numerals, *e.g.*, in Bader and Kolda's convention [112] $\{1, 3; 4, 2\}$ would specify that the first and third indices must be contracted together with the fourth and second indices of the first and second operands, respectively. We label this augmented version n-mode$^+$ notation.

Practitioners from other fields have introduced algebras with similar characteristics. R-matrix notation [53, 61, 62], which was articulated for geodesy applications, represents one prominent example. Predating n-mode product notation, its algebra expresses products in a similar manner using numerical designations. Suzuki and Shimizu's array algebra [65], introduced for the purposes of systems and control, provides another example that offers a similar set of characteristics and abilities as n-mode product notation.

Einstein notation represents the final high-degree algebra. Enjoying deep roots in physics, the notation can boast of an enviable pedigree with a long and fruitful history [69]. The notation can be grouped within the larger category of index notations, which include Tait's array algebra [113] and Antzoulatos and Sawchuck's hypermatrix algebra [60], which apart from aesthetics are very close to Einstein notation. We omit modern index-free notations for tensor calculus from consideration, as our focus is on technical computing algebras for numeric high-degree data. For interested readers, Papastavridis provides an excellent and lively summary of the virtues of indicial notation over index-free notation for concrete science and engineering applications [69].

The algebras advocated by different researchers embody various algebraic characteristics that have proven beneficial. These are summarised in Table 2.1. In describing an algebra, authors often cite its capabilities vis-à-vis the characteristics in Table 2.1. The rationale and benefits behind such algebraic characteristics are outlined below underneath their specific subheadings. With these algebras introduced, the arguments advocating the benefits of

Table 2.1: Comparing the capabilities of formalisms alternative to MV algebra. A '✓✓' and '✓' designate supported and partially supported capabilities, respectively.

Formalism	N-Degree	Associative	Commutative	Entry-Wise	Linearly Invertible
EMV algebra [73,108]	✓	✓	✓	✓✓	✓✓
n-mode$^+$ [112]	✓✓	✓	✓		
R-Matrix Notation [114]	✓	✓	✓		✓✓
Array Algebra [65]	✓✓	✓	✓		✓
Einstein Notation [69]	✓✓	✓✓	✓✓		
NT Algebra	✓✓	✓✓	✓✓	✓✓	✓✓

the characteristics outlined in Table 2.1 can be summarised and some unifying concepts discussed.

2.1.1 N-degree

MV algebra is unable to naturally represent N-degree linear mappings and multilinear mappings outside of bilinear forms. The introduction of Kronecker products and vectorisation operations blurs this boundary somewhat as, in principle, these two operations provide a means to incorporate multilinear and N-degree linear mappings into the MV framework. Nonetheless, proponents of other algebras, such as R-matrix notation [61,62], array algebra [65], Einstein notation [60,86], and n-mode notation [81,84], have argued for a more complete departure from MV formalism toward an algebra that can more naturally represent N-degree data structures and inner and outer products between them.

Many of these arguments rest upon the complexities involved when casting high-degree problems into the MV domain. For instance, both Kronecker products and vectorisation operations require settling on a lexicographic order. Including lexicographic considerations within an algebra elevates a consideration normally confined to constructivism into the realm of formalism. This can increase the burdens upon an algebra. For instance, while there are only two ways to lexicographically vectorise a matrix, *i.e.*, row or column-major, in the general N-degree case there are $N!$ vectorising options.

Even when data structures are confined to second-degree, mappings between them can produce important lexicographic confusion. As an extreme example, Eisele and Mason felt it necessary to describe 24 different means to represent the direct product of two matrices [115]. The Kronecker product itself has two different definitions—the common representation seen in Van Loan's paper [108] as well as Regalia and Mitra's representation [116]. In addition, there are competing views on how to best lexicographically lay out the partial derivatives of vector or matrix-valued functions [117].

Authors from signal processing [66], systems and control [65,113], image processing [60], and numerical methods [54,67,68] have explicitly made a case against casting N-degree problems into the MV domain, describing some of the requisite identities, rearrangements, and manipulations as "troublesome" [65], "cumbersome" [60], and "awkward" [54]. Harshman and Hong [118], after summarising the many conventions and complications in tensor decomposition that accrue from casting multilinear models into the MV framework, note such complications can be avoided "if one is willing to abandon matrix notation" [118]. Thus, for applications and fields that express mappings outside of MV's natural purview, there is an acute need for algebras that can innately work with N-degree data and mappings.

As noted, EMV algebra, with its intimate connections with the Kronecker product and vectorisations, can represent N-degree data and mappings, but not in a natural manner. R-matrix notation limits itself to matrices multiplied onto high-degree constructs, similar to the multilinear product described by Lim [84]. For this reason, Rauhala, who invented R-matrix notation, has seen cause to employ a more natural N-degree algebra when modelling high-order derivatives [114]. All the other featured algebras are designed to naturally accommodate N-degree data structures and mappings, allowing them to straightforwardly represent multilinear and high-degree linear mappings.

2.1.2 Associative

Associativity is a fundamental characteristic that eases algebraic manipulation. The associativity of matrix products is a classic example, providing MV algebra with group-like algebraic properties. As Meyer puts it, without associativity using MV algebra would be "unbearable" [119]. Constructive reasons also intrude, as associativity allows an implementation to choose how exactly to break up expressions into separate binary operations. Thus, a software implementation is free to choose the most efficient option.

EMV algebra exhibits some associative capabilities. For instance, a Kronecker product is associative with other Kronecker products. However, by remaining within the MV framework, how the algebra uses indices is tightly governed, *i.e.*, column and row indices of the left and right operand, respectively, always undergo an inner product within a standard matrix product. As a result, mixtures of different products are not always associative. As an example, matrix products and Kronecker products do not associate. This specific instance was highlighted by Snay [61] in his advocation of R-matrix algebra, which is associative for that case. Snay argues that key efficiencies are better revealed with an associative framework. Nonetheless, R-matrix notation is not completely associative in other situations.

The algebra of n-mode$^+$ products is also not easily associative, as associating may require switching the numerical designation of inner-product indices. This is highlighted by Suzuki and Shimizu, whose array algebra follows similar laws as n-mode$^+$ products. The authors outline the notational updates required to associate, stating that "the associative law for array multiplication is not so simple as that for the matrix multiplication" [65]. The requisite notational updates involved in associating complicates further when more than one operand is involved, as the ramifications of switching numeric index designations cascades throughout the entire algebraic expression. For this reason, n-mode$^+$ and array algebra are only given one checkmark for associativity.

The variation of n-mode product notation used by Brazell *et al.* [120] is unique in that it is associative. However, associativity comes at the cost of restricting the types of products that can be expressed, *e.g.*, inner product indices must be positioned at the end and beginning of the left and right operands, respectively. In contrast, like MV algebra, associativity is a bedrock for Einstein notation, to the point where practitioners will prohibit certain types of operations if they do break associativity [54]. In addition, Einstein notation can remain associative without imposing restrictions on the types of products it can express.

2.1.3 Commutative

Unlike associativity, commutativity is a characteristic not enjoyed by MV algebra. The restrictions imposed by non-commutativity are often highlighted by champions of Einstein notation, which is completely commutative. To provide a particularly pugnacious quote, Papastavridis writes that "whatever cosmetic or aesthetic advantages that [MV] notation may have, they are far outweighed by the merciless straightjacket of *noncommutativity* [emphasis original]" [69]. With a more positive viewpoint of MV algebra (and more in line with ours), Åhlander [54] and Harshman [86] also both highlight commutativity in their promotion of Einstein notation.

In fact, commutativity has proven important enough that practitioners of non-commutative algebras have developed commutation operators or expounded on instances where their formalism does commute. As an example of a commutation operator, Kronecker products can commute, provided one pre- and post-multiplies the expression with a "perfect shuffle" matrix [108]. For an example of showcasing isolated instances, both Eldén and Savas [111] and Blaha [62] highlight the limited cases where their algebras do commute, drawing attention to ensuing algebraic benefits. Nonetheless, their algebras do not commute in general.

When citing the benefits of commutativity, authors often highlight the accompanying algebraic conveniences. Yet, there are also important reasons related to how we write or typeset mathematical notation. Because notation is arranged on a line, a mathematical symbol can only be adjoined by other symbols on its left or right. Hence, non-commutativity is often acceptable for MV notation, as a matrix can only pre- or post-multiply anyway. However, for an algebra designed to express N-degree relationships, *e.g.*, a trilinear mapping, not every operator can adjoin its operand. Commutativity, combined with associativity, allows practitioners to rearrange, group, and substitute algebraic elements as needed.

This can also aid computations, as it is easier to modify the order of execution, allowing for more efficient options to be discovered and chosen. For a compelling example of these benefits, the Tensor Contraction Engine (TCE) uses associativity and commutativity to optimise the order of execution of Einstein-notation expressions involving tens to hundreds of terms [91].

In general most of the described algebras do not commute easily. For n-mode$^+$ products, R-matrix notation, and Suzuki and Shimizu's array algebra, commuting is similar to associating, in that the numerical inner product designations must be updated, and if necessary cascaded throughout the entire expression in question. And like associating, this hinders the commuting action. These difficulties are also reflected in Bader and Kolda's careful explanation of n-mode$^+$ algebra's commutation rules [112] outside of the core n-mode product. In contrast, Einstein notation is unique in its complete support for commutativity.

2.1.4 Entrywise

Supporting the notion of entrywise products represents another important thrust in the development of algebraic characteristics not embodied by MV formalism. To this end, EMV algebra has served as an important standard bearer in this push. By providing a formal framework for entrywise products, EMV algebra satisfies important expressive needs. As Liu and Trenkler note, entrywise matrix products have been receiving increased attention in several fields [73], reflecting the usefulness of these types of operations.

In fact, there is a reoccurring need for entrywise products or operations in general. For instance, in his monograph on linear structures, Magnus devotes space to Hadamard products, diagonal selection matrices, and other related operations [121]. Tensor decomposition also has need for entrywise operations, particulary in its canonical-polyadic (CP) decompositions or derivatives thereof. This need is significant enough for practitioners to use expressions outside of n-mode products. For instance, in Kolda and Bader's review [83], the CP decomposition is expressed using either matricisations and Khatri-Rao products or by using explicit summation symbols.

Tensor calculus occasionally requires entrywise products [69], requiring various workarounds from conventional Einstein notation. The more recent field of computational chemistry also needs entrywise extensions to Einstein notation [94,95]. Some conventions are to distinguish, by a parenthesis [69], an underline [68], an assignment [54, 60, 67, 94, 95], or a subscripted parenthesis [70], the indices whose summation is suppressed.

The repeated manifestation of entrywise products within these varied algebras indicate that these types of operations fill an important need. Formalising entrywise products are thus an important initiative in advancing technical computing's expressive capabilities.

2.1.5 Linearly Invertible

Inversion, either explicit or implicit through the solution of equations, represents a bedrock of MV algebra. In fact, the notion of determinants was studied and applied to the solution of linear equations long before Arthur Cayley formalised matrix multiplication [119]. Reflecting this, linear inversion is a major facet within technical computing. As mentioned in Section 1.2.2, the starting point of modern numerical analysis is marked as the publication of von Neumann and Goldstine's paper [39], "Numerical Inverting of Matrices of High Order". This prominent focus on inversion is also shared by EMV algebra. As van Loan notes, one of the most well-known uses of the Kronecker product lies in solving the Sylvester matrix equation [108]. As well, Lev-Ari's exposition of the benefits of certain EMV algebra operations rests on efficiently solving a set of entrywise linear equations [72].

Proponents of other N-degree algebras have also emphasised aspects regarding inversion. For instance, Snay's description of R-matrix notation concentrates on the efficiencies gained in solving a specific high-degree linear equation vs. Kronecker algebra [61]. Suzuki and Shimizu's exposition of array algebra also focuses on solving high-degree linear mappings, articulating some important qualitative differences compared to MV equations [65]. In the context of tensor decomposition, Brazell *et al.* [120] and Braman [122] have also discussed inversion and decompositions, respectively, of high-degree linear mappings. Iterative methods to solve the fourth-degree linear mappings discussed by Brazell *et al.* were also earlier discussed by Otto [68] and Åhlander and Otto [67] in the context of Einstein notation. Proponents of Einstein notation, like Harshman, have noted that inversion is "an area for further development" [86], calling for further consolidation and articulation.

These developments suggest that linear inversion occupies a prominent role in the minds of practitioners of these disparate algebras. For this reason, should an algebra lack clear notation and rules for solving equations, its widespread adoption would likely be hampered.

2.1.6 Unification – NT Algebra

Table 2.1 summarises the capabilities of several different algebras outside of MV formalism. In reviewing the literature on alternative algebras, a recurring thread epitomises much of the work, namely the need to describe N-degree data and mappings. Even the nominally independent characteristics of associativity and commutativity are affected by this, as these algebraic features become even more important once one enters into an N-degree context. The need for explicit description of entrywise products is also made more acute in an N-degree context as one is less able to resort to the simple diagonal-matrix strategies seen in the MV paradigm. Finally, linear invertibility also fits within this argument, as it is the complexities surrounding N-degree linear inversion that motivates select discussion on alternative algebras. Thus, N-degree data provides an important linking concept between the needs of the many different featured formalisms.

Unsurprisingly, there is no consensus on what name to give N-degree data. When advancing the case for the numerical and symbolic analysis of high-degree structures, authors use terms such as arrays [61, 62, 123], tensors [68, 80, 83, 112, 124], multidimensional arrays [54], multi-way or N-way arrays [83, 112, 124], multidimensional matrices [88, 99], and hypermatrices [60, 65].

If there were no historical context in play, we would opt to use none of the above terms, for the following reasons. Matrices are well known to engineers and scientists, which lends familiarity to the terms multidimensional matrices and hypermatrices. However, matrices imply rigid non-commutative rules, which argues against their use to describe N-degree data and operations. Moreover, due the multi-definitional use of dimensionality, where in some works it indicates the number of indices needed to specify a component and in others it refers to the size or range of indices, using the adjective multidimensional to describe N-degree data is problematic. To avoid this confusion, we opt to use "degree" for the former definition and "dimensionality" for the latter, aligning ourselves with tensor calculus terminology[1]. The adjective multi-way is not well-recognised outside of psychometric, chemometric, and certain tensor decomposition literature. While tensors can be represented by N-degree data structures, not all such data structures are tensors [54]. In addition, N-degree data structures are perfectly capable of describing physical phenomena outside of tensors. With no adjective, array is a good candidate, as it avoids much of the definitional baggage associated with other terms. However, in the context of programming, a discipline highly relevant to technical computing, the term array is widely used to describe a data structure that is not necessarily numerical. For these reasons, our ideal term would be multi-index arrays, to describe N-degree data.

Nonetheless, today the term "tensor" is ubiquitously used to describe N-degree data, regardless whether it is or is not a *geometric* tensor [84]. As this is more or less a *fait accompli*, we opt not to quarrel with this convention. However, we use the term numeric tensor (NT) to describe such data, emphasising that the salient characteristic consists of *numeric* N-degree data invested with a set of arithmetic operations, and de-emphasising any geometric considerations.

While the concept of an NT, as the fundamental data structure, can link together different research efforts associated with alternative algebras, an even stronger link can be forged by a formalism able to meet all their algebraic needs, meaning satisfying all the characteristics in Table 2.1. There are likely several different avenues to pursue toward developing such an algebra. In Chapter 3 we detail an NT algebra built off of Einstein notation that acts as a powerful formalism beyond MV algebra. As will be discussed in the chapter, there are good reasons to base an NT algebra off of Einstein notation. The last row of Table 2.1 illustrates NT algebra's expressive capabilities compared to the featured algebras. Because it embodies all the characteristics outlined in Table 2.1, the NT algebra we develop provides a foundation for an NT framework aspiring to tackle head-on certain of the MV paradigm's anomalies.

2.2 Software Characteristics

As discussed in Chapter 1, software acts as a partner to algebra in a technical computing framework. Thus, should an NT algebra be considered a desirable development, it is also important to articulate desired characteristics for NT software. Effective guidance can

[1] "Order" is another accepted term to describe the number of indices. However, when discussing sparse NTs, "order" will frequently be used to discuss how non-zero data is arranged, similar to the row- or column-major order found in MV software. For this reason, we avoid "order" as well.

be found by examining current efforts to develop software libraries tailored to support expressions and computations demanded by algebras different from MV algebra. As well, by surveying current software efforts, insight can be gained on how well NT-algebraic needs are met already. Together with an NT algebra, a compatible NT software would define an *NT framework*.

We focus on libraries designed for numeric computations involving high-degree data. As a result, we omit discussion of A Programming Language (APL) and derivatives, which use generalised and possibly heterogeneous multidimensional arrays as their central datatype. While APL has influenced subsequent technical computing languages, *e.g.*, MATLAB [46], the language allows arrays to be collections of any datatype, and even other arrays, meaning its focus is not numeric calculations.

Of the software libraries featured in this survey, most support N-degree data and operations. An important exception is Steeb and Shi's Kronecker classes (SSKC), which is a C++ library implementing Kronecker algebra operations.

Many of the N-degree libraries focus specifically on minimising abstraction penalties as much as possible. A prominent example is Blitz++ [125], a trailblazing library that has pushed the abstraction capabilities of the C++ language itself [126]. Although Blitz++ is designed to be a general array manipulation library, it provides partial support for Einstein notation. Following up on Blitz++'s success, FTensor [101] further optimised Einstein notation expressions in C++. Like Blitz++, FTensor has also been cited as a driving force behind C++'s abstraction capabilities [126]. LTensor [127] is a similar library to FTensor, but offers better support for large-dimension data than FTensor.

Other software libraries also support Einstein notation, but place more focus on techniques for large-dimension problems, particularly partial-differential equations (PDEs). Two prominent examples are the EinSum [54,67] and POOMA [128] libraries, both implemented in C++. Reflecting their focus on large-dimension problems, both libraries offer support for efficient description of stencil operators, *e.g.*, finite differences.

The field of computational chemistry provides an additional important thrust in supporting large-dimension Einstein-notation calculations. Due to the massive computational burden of the chemistry calculations, researchers have developed solutions for massively-parallel and distributed-memory environments, which include the TCE [91,92,93] and the Cyclops Tensor Framework (CTF) [95]. The TCE is also able to automatically generate optimised low-level FORTRAN loops given highly complex Einstein-notation expressions with many operands, revealing another important thrust of high-degree software research. Libtensor represents another example within computational chemistry, which focuses on optimised block NT computations on shared-memory architectures [94].

Researchers have also designed softwares for dynamic languages. A good example is NumPy [129], which is a general purpose technical computing library implemented in Python that also provides excellent support for Einstein notation. The MATLAB Ten-

Table 2.2: Comparing the capabilities of alternative software solutions for high-degree algebra. A '✓✓' and '✓' designate supported and partially supported capabilities, respectively.

	Comprehensiveness*	Sparse Support	Programming Efficiency	Computational Efficiency
SSKC [130]	✓		✓	✓
Blitz++ [125]	✓	✓	✓	✓✓
FTensor [101]	✓		✓	✓✓
LTensor [127]	✓		✓	✓✓
EinSum [54, 67]	✓	✓	✓	
POOMA [128]	✓	✓	✓	✓✓
TCE [91]	✓		✓	✓✓
CTF [95]	✓		✓	✓✓
Libtensor [94]	✓	✓	✓	✓✓
NumPy [129]	✓		✓✓	✓
MTT [112, 124]	✓	✓✓	✓✓	✓
LibNT	✓✓	✓✓	✓	✓✓
NTToolbox	✓✓	✓✓	✓✓	✓

*w.r.t. support for N-degree data, entrywise products, and linear inversion

sor Toolbox (MTT) [112, 124], a highly influential library, is a MATLAB library providing comprehensive support for tensor decomposition operations. As part of this, MTT supplies routines implementing n-mode$^+$ notation and Khatri-Rao products.

To support NT algebra, we focus on four software characteristics, illustrated in Table 2.2. It should be noted that the table does not include two qualities that are regularly emphasised in computational chemistry: massively-parallel computations and support for symmetric and anti-symmetric NTs. As our aim is to first break ground on general-purpose software supporting the NT-algebraic needs of Table 2.1, we do not investigate parallel and symmetric computations within this thesis. Nonetheless, parallel and symmetric computations are an important aspect of future work that we discuss in more detail in Chapter 7.

Returning to the four characteristics of Table 2.2, the capabilities of these software libraries with regard to these characteristics can differ markedly from each other. An explanation of the importance of these capabilities to NT software are outlined below, along with summaries of the functionality of the featured libraries based on these factors.

2.2.1 Comprehensiveness

Comprehensiveness refers to how well a software library satisfies the needs of algebraic operations for high-degree data, which are outlined in Table 2.1. Out of these, the focus is on support for N-degree data and operations, entrywise operations, and linear invertibility. Associativity and commutativity remain important, but those characteristics are more specific to algebra, lying somewhat orthogonal to software considerations.

In using this criterion, we acknowledge that we are often assessing software libraries on capabilities that can lie outside designer intent. For instance, SSKC, which is designed to support Kronecker algebra, should not be expected to fully support features outside of its formalist scope. It should also be noted that many libraries can boast of varied and important capabilities not listed in Table 2.1. As such, judging that a software does not comprehensively support NT algebra is not necessarily a criticism of the implementation *per se*. Instead, our examination of the literature concludes that the capabilities in Table 2.1 are important in tackling MV paradigm anomalies, and we are surveying the field to examine in what way these capabilities are already met by the state of the art.

When considering N-degree data and operations, software library designers have emphasised the design choices necessary to offer complete support. For instance, when outlining their complete N-degree support, the authors of MTT emphasise that algorithms and data structures often must differ from their common MV counterparts. The authors of EinSum do the same, discussing how C++ expression templates allow one to support the more numerous mapping options in N-degree space. NumPy's design has also been carefully tailored to support N-degree data, operations, and indexing [131]. The TCE, CTF, and Libtensor packages also offer N-degree support, which is crucial for computational chemistry calculations.

Other libraries focus more heavily on offering zero or close-to-zero abstraction penalties, sometimes at the cost of only providing partial support for N-degree operations. For instance, the designers of both FTensor and LTensor manually coded the different index matchings between binary addition, subtraction, and multiplication operations, limiting the libraries to general second and fourth-degree operations, respectively. Blitz++ and POOMA accept some degree of abstraction penalties, but still only allow up to 11- and 7-order data, respectively. These limits suffice for many applications. Finally, by focusing on Kronecker algebra, SSKCs do not support N-degree operations.

In addition to N-degree operations, entrywise operations have also been the focus of certain software offerings. The EinSum authors devote space to discussing how entrywise operations can be rigourously supported in Einstein notation by using a summation suppression scheme. NumPy, CTF, TCE, and Libtensor also offer support for entrywise products using a similar scheme. MTT provides partial support for entrywise products by furnishing operations to matricise high-degree data, which can then be supplied to its Khatri-Rao product routines.

Inversion also enjoys varied support among constructivisms. SSKC offers inversion by virtue of remaining within the MV framework. NumPy is unique in providing linear solving routines directly in high-degree space. However, NumPy does not allow full freedom for how indices can be matched together in the expression to invert. MTT offers implicit support for inversion through its matricisation routines. Inversion does not occupy a place in the other featured software libraries.

2.2.2 Sparse Support

The "curse of dimensionality", coined by Bellman [132], refers to the fact that as the number of dimensions increases, the volume of a hypercube grows exponentially. Of relevance to NT

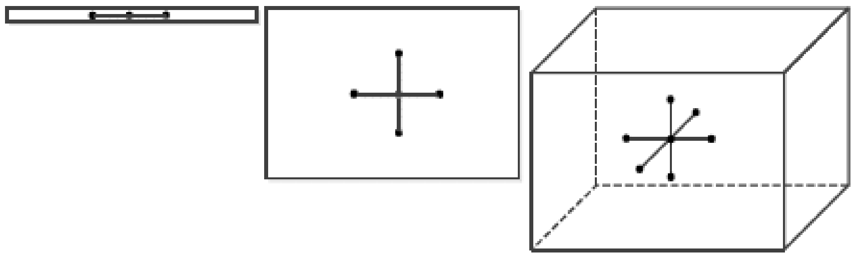


Figure 2.1: Sparsity pattern of the $O(h^2)$ Laplacian stencil. If the range of each index is n, the fill factor for first, second, and third-degree domains is $3/n$, $5/n^2$, and $7/n^3$, respectively.

data and operations, this means that working with dense data quickly becomes infeasible as the degree increases. Provided the underlying data has mostly zero-valued entries, using a sparse representation is one way to overcome this problem.

Even if the underlying data is not sparse, often the operations performed on it are sparse. For instance, a dense linear mapping of an $n \times n$ image to another $n \times n$ image would require an operator having n^4 elements. However, in many cases mappings are local in nature, forestalling the need for dense representation. For instance, finite-difference (FD) approximations to partial derivatives are typically only defined using the immediate neighbours of the location in question.

A prominent example is the Laplacian operator, which is typically approximated by an $O(h^2)$ FD scheme. As Figure 2.1 demonstrates, if N is the domain degree, the Laplacian stencil consumes $1 + 2N$ non-zero entries. Thus, since the number of non-zeros increases linearly, the sparsity of such operators increases exponentially with degree. These considerations are also just as applicable, if not more so, for mappings beyond linear. For instance, polynomial systems are often extraordinarily sparse [74]. This means that proper and reliable sparse support is highly important for high-degree operations.

Despite the importance of sparsity for NT data, it is not commonly supported. MTT represents an important exception, as its developers have devoted considerable effort toward advancing the understanding of how to treat sparse NT data [124]. As the authors explain, many of the strategies appropriate for sparse MV data no longer hold in high-degree contexts. These observations emphasise that computational techniques designed for the second-degree environment of MV constructivisms do not always generalise.

Libtensor's block-tensor data representation is able to avoid explicitly storing non-zero blocks. However, because metadata for each nonzero block still must be stored, this representation is not designed for highly-sparse NTs. Blitz++, POOMA, and EinSum have also contributed to the state of sparse high-degree data by providing means to implicitly describe banded high-degree operators, *e.g.*, the Laplacian stencil illustrated in Fig. 2.1. Apart from these instances, most software libraries focus predominantly on dense calculations.

2.2.3 Programming Efficiency

In his take on modern programming, John Ousterhout, the creator of the Tcl scripting language, argues that programming tasks can be roughly divided into creating computationally-heavy algorithmic components or writing applications that plug and glue said components together [133]. For tasks in the latter camp, any heavy computational burden is already offloaded to pre-existing components, leading Ousterhout to argue that programming efficiency, *i.e.*, how quickly can programmers create quality software, is paramount. Ousterhout argues that much of programming tasks today can be characterised as gluing and systems integration rather than algorithmic or data-structure development.

As a field practised and advanced by both end users and computing experts, technical computing has a foot firmly planted in both camps. Nonetheless, much of the field consists of plugging together gold-standard algorithms, *e.g.*, MATLAB programs essentially glue together components from LAPACK and other highly-optimised algorithms. This is particularly true for scientists and engineers within the application domain, leading authors to elevate programming efficiency to a primary consideration within technical computing [134, 135]. This emphasis on programming efficiency can be seen as part of a larger trend in general-purpose programming [133, 136]. When discussing an NT framework, the unavoidable operational complexity that accompanies working with N-degree data, *i.e.*, the different ways that indices between operands can match-up, places its own impetus on programming efficiency, as it becomes even more incumbent to minimise implementation barriers.

Programming efficiency can be realised in different ways. One important means is the structure of the language chosen for development. More specifically, dynamic or scripting languages can offer an environment particulary amenable to programming efficiency. Such languages are often characterised by interpreters, dynamic type-checking, automatic memory management, and/or command-line interfaces. In the realm of general-purpose programming, the frequent priority placed on programming efficiency has been cited as the driving force behind the increasing popularity of such languages [133, 136]. The argument that dynamic languages lead to programming efficiency is strengthened by case studies [133] and empirical evidence [137]. In the realm of technical computing, authors have echoed these assertions, arguing that dynamic languages, accompanied by mature graphical user interface (GUI) tools and interactive environments, lead to programming efficiency [135, 138]. These authors include the original author of NumPy [134]. The importance of programming efficiency to technical computing is also supported by analytics on language popularity. For instance RedMonk's 2014 Q1 analysis [139] and the TIOBE's software popularity index [140] both demonstrate MATLAB's increasing popularity over the years, now placing in the top 20 for both metrics.

The demands of technical computing impose another means toward programming efficiency that is not always encountered in general-purpose programming. Namely, in technical computing how faithfully a software supports an algebra significantly impacts programming efficiency. This is an important factor, as it affects how easily a solution on paper can be translated to an implementation on a computer. When general-purpose computing does encounter similar needs, it is framed as requiring a domain-specific language (DSL). In short, a DSL is a programming language that offers notation and expressive power for a specific problem domain [141]. As van Deursen *et al.* note, "DSLs allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain. Consequently, domain experts themselves can understand, validate, modify, and often even develop DSL programs" [141]. This description is especially apt for technical computing, where the direct support of MV algebra in packages like MATLAB acts as a DSL, affording scientists and engineers greater opportunity to develop their own computational solutions and easing programming burdens.

For these reasons, in examining programming efficiency of the featured software libraries, the assessment in Table 2.2 is based upon whether the software provides a dynamic-language interface and also whether it offers a DSL for its associated algebra. Using this criteria, all of the featured constructivisms do offer some form of a DSL. Nonetheless, the translation effort between a software library's associated algebra and its offered DSL can vary. For instance, MTT and SSKC offer free-function interfaces, which differ significantly from their modelled algebras and introduce obstacles toward composing larger expressions. When considering whether the featured software libraries offer a dynamic language interface, only two languages, NumPy and MTT, meet that criterion. All other libraries require using a compiled language.

2.2.4 Computational Efficiency

By its nature, technical computing relies upon computationally intensive numerical methods. For this reason, technical computing can be susceptible to debilitating bottlenecks, making computational efficiency a primary concern. As well, because of their potentially large sizes and complex mapping possibilities, NTs possess their own particular brand of computational complexity. Important factors in computational efficiency include whether a software library employs gold-standard algorithms and whether it minimises computational overhead. Oftentimes the latter factor is at cross purposes with programming efficiency, while the former is not.

As noted in Section 2.2.3, technical computing's practice involves both the development of computationally-intensive algorithms and the gluing together of such components. As advocates of scripting in technical computing argue [134, 135, 138], as long as highly-optimised implementations of algorithmically-intensive components are used, gluing together components using dynamic languages does not always significantly impact computational efficiency. For this reason, a major factor in computational efficiency is whether a constructivism employs gold-standard algorithms and routines, such as LAPACK and BLAS.

While heavily-optimised algorithmic components can be leveraged by all manners of software implementations, the overhead in gluing together said routines can significantly differ. Even advocates of scripting languages in technical computing do admit that, come deployment, the demands of computational efficiency can make it necessary to use compiled languages to reduce overhead [138]. Despite efforts at reducing this overhead, *e.g.*, the vectorisation techniques in MATLAB, avoiding constructs like explicit loops or other heavily interpreted statements is not always possible. Yet, overhead is not only a feature of scripting languages. Even when one uses compiled languages, design choices can seriously drive down computational efficiency. For instance, in C++ dynamic polymorphism, or classes with virtual functions, can produce highly significant bottlenecks [142]. As a result, regardless of the implementation language, minimising overhead remains an important aspect of computational efficiency, and constitutes our second criterion in assessing computational efficiency. It should be noted that a dynamic language implementation, which cannot avoid this overhead, represents one of our programming efficiency criteria. As a result, computational efficiency can conflict with programming efficiency, reflecting the commonly encountered tension between programming and computational efficiency in more general-purpose programming.

When considering whether software libraries are built upon well-optimised algorithms, most of the featured examples use such components. Important exceptions include FTensor and LTensor. However, these libraries are primarily designed for a series of small-dimension tensor operations. For such scenarios, their lazy and in-place evaluation schemes can produce excellent computational efficiency. Thus, they are not considered inefficient, with the important caveat that their efforts at computational efficiency do not always apply to more general situations. On the other hand, EinSum and SSKC, which are meant to be used with large-dimension data structures, do not use optimised linear algebra numerical methods. Depending on the number of operands within an expression, NumPy will also forego using optimised numerical routines. All other featured constructivisms use highly optimised numerical methods.

Apart from gold-standard algorithms, the other criterion is level of overhead. As both MTT and NumPy are dynamic-language implementations they are unable to avoid impacts on runtime performance for tasks that cannot be offloaded to compiled-language components. Examining compiled-language implementations, EinSum does come with overhead through its use of dynamic polymorphism to resolve data access calls during inner/outer product multiplications. As multiplication can involve frequent data access, the overhead caused by numerous virtual table lookups can seriously impact performance [142].

The TCE, CTF, and Libtensor libraries for computational chemistry also come with relatively high-levels of overhead, but this is due to their need to distribute massively large computations across different nodes and cores. Thus, for the dimensionalities they are designed for, their overhead adds to, rather than detracts from, computational efficiency.

For most compiled-language libraries, great efforts were made to reduce or even eliminate abstraction penalties. In fact, the authors of both FTensor and Blitz++ have pioneered template metaprogramming (TMP) techniques for the express purpose of reducing or even eliminating abstraction overhead. These efforts have impacted the larger C++ programming community [126], attesting to the importance of computational efficiency when implementing NT software.

2.2.5 Unification – NT Software

From surveying the state of software libraries outside the MV paradigm, it is clear that both programming efficiency and computational efficiency each play important roles. As a result, in order to ease the acceptance of NT frameworks distinct from the MV paradigm, supporting software should not neglect the needs of those seeking high programming efficiency or those wishing for high computational efficiency. However, fully satisfying one criterion often means that some aspect of the other is not fully realised. Thus, to satisfy both needs, an NT software suite should be built off of a common core of routines, and offer a choice of interfaces that either bias toward programming or computational efficiency.

Of course, an interface should still attempt to simultaneously satisfy both needs as much as possible. Along those lines, the C++ language provides an excellent foundation for technical computing software. As Abraham and Gurtovoy outline, C++ provides an excellent host language to implement the constructivism of all manners of formalisms [126]. It inherently supports object-oriented programming (OOP) with static-type checking, meaning operations and relationships can be explicitly controlled and defined in a type-safe and efficient manner. In addition, C++ supports 48 different operators [126], all of which can be overloaded. Finally, these features are complemented by excellent generic programming (GP) capabilities [126, 143].

The low-overhead abstraction capabilities of C++ make it an excellent environment for the development of domain-specific embedded languages (DSELs), which are DSLs built on top of a standard host language. These abstraction abilities also engender high levels of programming efficiency, as a formalism can be faithfully reproduced in a programming environment, allowing a practitioner to conceptualise and program her solution ideas with minimal translational burdens. These reasons have allowed existing C++ DSELs for MV and high-degree data structures to flourish. These include Armadillo [144] and Eigen [145] for the former and Blitz++ [125, 142] and FTensor [101, 146] for the latter.

Despite these enviable abstraction capabilities, the compiled and static-typing nature of C++ means that programming efficiency cannot always reach the level found within dynamic-language implementations. Fortunately, any core C++ algorithmic routines can be interfaced to dynamic languages. For instance, MATLAB provides its MEX interface and Python enjoys a myriad of its own C++ interfaces, *e.g.*, Boost.Python [147]. A constructivism offering both compiled-language and dynamic-language interfaces allows practitioners themselves to use their own programming vs. computational efficiency cost-benefit analysis. As well, should a practitioner wish to translate a dynamic-language prototype to a native-language deployment, having similar styles of interfaces helps to reduce programming effort. Finally, since both interfaces would use the same algorithmic routines, worries related to differences in numerical stability, convergence, or performance are minimised.

These arguments have shaped the development of LibNT and NTToolbox, which represent our own NT software offerings for C++ and MATLAB, respectively. Both libraries are designed to support the NT algebra we develop, which as Section 2.1.6 explained is an extended form of Einstein notation. However, they each place a different emphasis on programmatic vs. computational efficiency. For instance, primarily focusing on computational efficiency, LibNT provides a native DSEL interface to NT algebra, incorporating innovative high-performance algorithms and data structures for N-degree, entrywise, and inversion operations. As well, LibNT places heavy emphasis on sparse data structures and computations appropriate for high-degree data. NTToolbox, on the other hand, offers a MATLAB interface to LibNT's fundamental computing routines. Like LibNT, NTToolbox offers a DSEL for NT algebra, but implemented within the interpreted environment of MATLAB.

By placing different emphases on computational and programmatic efficiency, LibNT and NTToolbox complement each other. As such, they represent an effective embodiment of the key principles identified as important for NT software, making them a productive contributor to an NT framework. These NT software libraries will be discussed in detail in Chapters 4 and 5.

2.3 Exemplar Categories

Kuhn noted that anomalies in a paradigm can act as catalyst in galvanizing practitioners into performing revisionary science [1]. The successful treatment, or more frequently the promise of successful treatment, of these anomalies serves as motivation toward the adoption of tools, techniques, and theories outside the accepted paradigm. These anomalies may then serve as exemplars for a new paradigm. As such, exemplars are concrete embodiments of certain principles, enabling them to define and shape scientific practice. Kuhn emphasised that the successful redefinition of an anomaly to an exemplar is dependant on whether this taxonomic change fosters enough new problems, pregnant with potential, for a scientific discipline to pursue. The persuasive role of new and promising questions is reflective of the role and desire of scientists to act as investigators.

Unsurprisingly, for those advocating technical computing algebra and software outside the MV paradigm, exemplars have served as crucial persuasive tools. Like the identified anomalies, the two broad categories that these exemplars often fall into are either special linear mappings or mappings beyond linear. NTs are a key element of both exemplar categories.

2.3.1 Special Linear Mappings

Kuhn observed that anomalies are a constant reality for any paradigm. Anomalies may arise due to the discovery of new facts, but just as often as not, "the problems that are responsible for the anomalous data are not necessarily new problems that arose after consensus but may have been present all along" [148]. Or in Kuhn's words, in some cases anomalies "without apparent fundamental importance may evoke a crisis if the applications that it inhibits have a particular practical importance" [1]. This description applies very well to the problem of high-degree linear and entrywise mappings. Using flattening operations for the former and diagonal matrices for the latter, it is in principle possible to frame such linear mappings using the MV paradigm. However, such an approach can be burdened with significant complexities, suffer from inefficiencies, and may even hide solution avenues. The detriments in overlooking the anomalous nature of such mappings move in lockstep with the significance of their applications.

Researchers have used exemplars to highlight certain benefits that alternative algebra and software offer for special linear mappings. Practical concerns represent a prominent theme behind much of this work. For instance, Suzuki and Shimizu [65] and Antzoulatos and Sawchuck [60] argue that their natural N-degree algebras eases the implementation of high-degree linear operators in systems-and-control and image processing applications, respectively. In terms of entrywise operations, Liu and Trenkler [73] showcase several instances where linear regression and covariance problems gain from the explicit entrywise products of EMV algebra. Of particular note, the authors show how explicit entrywise product operations can reveal more efficient solutions. As another example, researchers within computational chemistry have used major quantum chemistry models to highlight how fully associative and commutative N-degree algebras can help derive massive gains in computational efficiency [91]. Moreover, the pursuit of solutions to quantum chemistry exemplar problems have driven innovations in symmetric representation [94, 95], amongst other advancements within high-degree software.

Many of the same practical concerns drive work on differential operators applied to high-degree domains. These are an important set of exemplars, as they are a core component of high-degree PDEs, which in turn act as the core elements of myriad scientific models. In addition, as explained in Section 2.2.2, when manifested as FD operators, or other approximations, differential operators exhibit high sparsity, making them practical even in high-degree contexts. This has been well-recognised in the literature. For instance, Åhlander and Otto maintain that Einstein notation avoids significant problems arising when using the MV paradigm to implement and program high-degree finite-difference operators [67]. Differential operators are also specifically highlighted by the developers of POOMA [128] as a motivating reason for prospective users to use their library.

Seeking a rigourous means to express derivatives of high-degree functions also motivates much work on alternative frameworks. In the MV paradigm, such situations are referred to as vector or matrix-valued functions, and the involved derivatives are referred to as Jacobians or Hessians, where the former is actually a linear map. Systems and control provides many motivating examples, *e.g.*, "Modern control theory is based on the vector-matrix theory... Hence it is often necessary to differentiate vectors (matrices) with respect to vectors (matrices)" [65]. Suzuki and Shimizu cite the difficulties of performing differential calculus in the MV paradigm to motivate the development of their array algebra [65]. Milov also discusses this need for the purposes of sensitivity analysis [99], calling for greater rigour and formalisation, which he develops in his multidimensional matrix concepts. The need for derivatives of matrix-valued functions is also acute for many of the statistical methods used by econometricians [63, 117], which partly motivates the field's use of Kronecker algebra.

Rauhala also cites derivatives of high-degree functions as crucial for optimising nonlinear functions in geodesy and photogrammetry [114]. In fact, optimisation problems in general often require derivatives. Algebraic complications and confusion can increase drastically with each increase in degree, making the expression of derivatives a particularly prominent concern.

Another recurring discussion involves the frequent need to perform decompositions to make intractable problems tractable. Researchers note that these situations often arise amongst high-degree linear mappings, where the curse of dimensionality can quickly balloon a problem's computational and memory demands. In these cases, researchers argue that algebra and software designed to handle high-degree data and mappings can reveal decomposition strategies that would otherwise be well hidden. For example, Rauhala, Snay, and Blaha motivate R-matrix notation by demonstrating how certain high-degree linear and least squares systems can be decomposed into separable mappings, stressing that seeking out such opportunities can produce massive computational savings [53, 61, 62, 114]. The need to actively seek out separable high-degree solutions was more recently highlighted by Belykin and Mohlenkamp [89], where the authors advocate working with approximate, but separable, linear mappings. Separability is also a frequent motivator for Kronecker algebra. In fact, the Sylvester equation, which is one of Kronecker algebra's most common exemplars [108], can be framed as a separable linear system. Specific applications of these cited works have included interpolation and signal processing transforms [53, 114], regression [61], and physics-based mappings [89].

Thus, chosen exemplars in the literature often emphasise one or more of the challenges in implementing special linear mappings, the need for rigourous derivative expressions, and the importance in executing separability. These characteristics have proven themselves significant enough to spur pursuits of alternative frameworks in technical computing. What is important to note is that NTs play a significant role in all such motivations. To be more specific, these topics are concerned with seeking out optimal means to express, manipulate, and compute with mappings applied to NTs.

To this end, we showcase important exemplars, drawn from the author's computer vision work, that are driven by high-degree differential mappings. The exemplars are based on two computer vision problems, called interactive image segmentation and depth-map estimation. As the differential mappings exhibit anisotropic properties, entrywise products also come into play, providing an excellent venue to illustrate both the challenges involved from remaining with the MV paradigm and also the benefits accrued from employing an NT framework. Using differential mappings links our exposition to PDEs, connecting the exemplars to the work of practitioners in many other fields. Depending on the exemplar, optimisation is involved, meaning derivatives of high-degree functions come into play. Additionally, we highlight separability, but of the separable nonlinear least squares (SNLS) kind [149], an important type of separability that has not yet received as much attention from advocates of high-degree frameworks. Finally, sparsity plays a crucial role in all the exemplars, along with other concepts such as linear NT inversion. Thus, interactive image segmentation and depth-map estimation serve as an excellent source of exemplars for special linear mappings. Chapter 6 is devoted to these computer vision exemplars, outlining how an NT framework is particularly well suited to them and similar exemplars involving high-degree differential operators.

2.3.2 Beyond Linear Mappings

Kuhn wrote that "To be accepted as a paradigm a theory must seem better than its competitors, but it need not, and in fact never does, explain all the facts with which it can be confronted" [1]. Continuing on this theme, Kuhn emphasised that a successful paradigm must be "sufficiently open-ended to leave all sorts of problems for the redefined group of practitioners to solve" [1]. In other words, paradigms do not only offer a set of solved exemplars, they also offer up a body of exemplar *problems*.

For instance, the MV paradigm, by its very nature, helps delineate what type of expressions are desirable to compute. This defined the questions investigated early on in technical computing, *e.g.*, how to stably and numerically invert a matrix [39], and continues to define research pursuits today. However, the MV paradigm is ill equipped to naturally represent major open problems in technical computing that involve mappings beyond linear. Here, we focus on decomposing high-degree data and numerical approaches to solve and manipulate multilinear and polynomial systems.

The first problem is the focus of the tensor decomposition field. As the field is explicitly concerned with NT data, the need for alternative frameworks is acute. Since non-symmetric NTs naturally describe multilinear mappings, the field is indeed concerned with mappings beyond linear and its literature often elucidates this link. Symmetric NTs, in turn, are bijectively related to homogeneous polynomials, which are also beyond linear.

From the outset, exemplars have driven the pursuit of decompositions of high-degree data, as it was researchers from the application-driven fields of psychometrics and chemometrics that pioneered much of the theory and algorithms [83]. To tackle these exemplars, researchers have developed algebra and software to naturally represent NT data. These efforts occupy a valued place within the field. Requiring capabilities outside the MV paradigm, researchers have proposed n-mode notation and variants [83, 110, 111], n-mode$^+$ notation [97, 98, 112], EMV algebra [83, 118], and Einstein notation [86] as formalisms. Tensor decomposition researchers have also trail-blazed new approaches to compute with high-degree data. For instance, practitioners have emphasised that sparse computations for high-degree data must take a different approach than those found within the MV paradigm [124, 150, 151, 152]. Similar arguments have been articulated for symmetric representations of high-degree data [153].

As such, the problem of decomposing high-degree data drives a great deal of work beyond the MV paradigm. Like the singular value decomposition (SVD) decomposition, tensor decomposition impacts countless fields. Interest in the field has exploded in recent years, and currently tensor decomposition plays an enormous role within technical computing [83]. Reflecting this importance, throughout our description of NT algebra and software we frequently focus on CP tensor decomposition as an exemplar, highlighting its unique algebraic and software challenges.

In contrast to the popular focus on tensor decomposition, the question of solving and manipulating multilinear and polynomial systems does not enjoy the same degree of attention within technical computing. We view this as an important gap within technical computing, as in principle an NT framework can represent multilinear and polynomial mappings naturally, expanding the bounds of what sort of operations one could compute or *wishes* to compute. Like Stetter [74], we believe a numerical approach to solving polynomial systems to be a "no man's land", or we would say major anomaly, in technical computing's state of the art. We also include multilinear systems within this anomaly. Considering that technical computing caters both to its own specialists but also to domain experts wishing to apply its techniques and tools, the problem cannot be judged fully resolved until any new knowledge is framed in a form palatable for the general science and engineering community. By naturally expressing and computing both polynomial and multilinear operations, an NT framework cannot but help increase the scope of technical computing investigations. This motivation is distinct from the objective of solving existing problems better.

While we view this as an important exemplar to pursue, it falls outside the scope of this thesis. However, we return to this argument in the conclusion, where we argue that future work should articulate the links between high-degree frameworks and systems of multilinear and polynomial equations. Because they can naturally represent such equations, we make the case that high-degree frameworks, like the NT framework, can play a crucial role in expanding the bounds of technical computing to include these crucial exemplar problems. Should this happen, the taxonomy of technical computing would be irrecoverably altered.

2.4 Summary

A growing crowd of researchers are investigating algebras and software beyond the MV paradigm, encapsulating a variety of desired algebraic and computational capabilities and characteristics. Exemplars feature prominently in many of these efforts, serving as important illustrative and persuasive applications. From this survey of algebras, software, and exemplars, NTs are identified as a recurring theme. The viewpoints, techniques, and tools offered by the reviewed research informs our own contributions to this growing crowd.

The algebras surveyed in this chapter satisfy one or more of the following attributes: N-degree, associativity, commutativity, entrywise, and linearly invertible. Each of these play important roles in applications and research pursuits. Broadly speaking, the demand to express N-degree data and mappings amplifies the need for these algebraic considerations. We argued that an algebra alternative to MV algebra should embody all such characteristics, increasing its impact and relevance to the disparate fields advocating alternative formalisms. Chapter 3 details the NT algebra we develop, discussing its expressive capabilities.

After identifying NTs as a unifying concept for alternative algebras, the chapter then investigated the characteristics important for any NT software. As a partner to an NT algebra in a technical computing framework, an NT software library must support all formalist characteristics outlined above. As well, to help circumvent the "curse of dimensionality", often present in an NT context, efficient representation and computations of sparse data is of particular importance. Finally, both programming and computational efficiency represent crucial, if occasionally conflicting, considerations. This assay of constructivist developments shapes our own NT software work. Outlined in Chapters 4 and 5, our own NT software, called LibNT and NTToolbox, is designed to meet the computational demands of a framework beyond the MV paradigm.

Kuhn maintains, as also evidenced by the work of those advancing alternatives to the MV paradigm, that exemplars are fundamental in articulating the promise lurking beyond the boundaries of a scientific discipline. As this chapter emphasised, two exemplar categories epitomise pushes for technical computing frameworks different than the MV one. We focus These exemplar categories are further articulated within this thesis. For the first category, *i.e.*, special linear mappings, we identify two prominent computer vision problems to showcase how an NT framework can meet demands frequently raised by such mappings. These exemplars are pursued in Chapter 6. For the second category, *i.e.*, mappings beyond linear, we frequently highlight CP tensor decomposition throughout our elucidation of NT algebra and software. Finally, articulating the links between NT frameworks and polynomial/multlinear systems of equations remains an important aspect of future work that we discuss further in Chapter 7.

Returning to our analogy of Milgram *et al.*'s famous experiment, the crowd of researchers we noticed, who were looking at algebras, software, and exemplars beyond the MV paradigm, was large enough to motivate our own investigations. Examining their work helps us craft our own contributions, meaning we now consider ourselves part of this growing crowd. NTs serve as a unifying concept for much of these research efforts. For this reason, articulating the features, capabilities, considerations, and benefits of NTs serves as a focal point for our own work. In the following chapters, we will focus on our own contributions, which can be summarised as introducing a specific NT framework, composed of an algebra and software, and using it to advance selected exemplars.

Chapter 3

Extending Einstein Notation

We begin this chapter by making the case that exploiting Einstein notation leads to a solid foundation for an numeric tensor (NT) algebra. This argument is made concrete through the introduction of an NT algebra notation that extends Einstein notation. Afterwards, we outline the advantages of NT algebra in tackling three exemplars, contrasting it with other high-degree algebras.

While the algebra outlined in this chapter differs considerably from matrix-vector (MV) algebra, taught to most computer engineers and scientists, the formalism does not discard the concepts of matrices and vectors. In fact, MV algebra occupies a valued place within NT algebra, reflecting our belief that the considerable strengths of the former should be leveraged in the appropriate circumstances. Moreover, NT algebra builds upon a heavy body of prior work. For instance, NT algebra is deeply rooted in index or Einstein notation, which is excellently outlined by Papastavridis [69]. Many of the notations and identities, such as entrywise products, that make NT algebra unique from typical Einstein notation has already been outlined by Joseph [123] and to a lesser extent by Barr [70]. However, the notation of vector and matrix NTs and concepts surrounding the solution of linear NT equations still require more development. As well, important aspects, including differentiation, nonlinear functions, and factoring out common terms, have not received treatment within an NT algebra context.

This chapter both reviews already developed material and fills in some of the aforementioned gaps. In performing the former, the chapter also provides many of its own justifications and explanations.

3.1 Einstein Notation

Section 2.1 identified five algebraic characteristics considered important for algebraic work beyond the MV paradigm. The section argued that an algebra alternative to MV formalism should embody all these characteristics. Since no prospective algebra in the literature meets this criterion, none of them qualify, motivating the development of the NT algebra we develop. Nonetheless, an NT algebra should use aspects of these alternative algebras as foundation, allowing it to leverage their considerable beneficial qualities. Along those lines, the NT algebra we outline is built off of Einstein notation, which already enjoys enviable expressive capacities.

Several of Einstein notation's qualities serve to make it an excellent foundation for an NT algebra. As Åhlander argues, the Einstein summation of Einstein notation offers a great deal of versatility, as it can represent any combination of inner and outer products [54], which means it can represent all manner of linear, bilinear, or multilinear operations. Because of Einstein notation's associative and commutative products, accompanying this capability is a fundamental ease of algebraic manipulation. Moreover, Einstein notation is amendable to extensions that allow it to fully support entrywise products and inversions.

In contrast, it is not so clear how to extend some of the other featured formalisms without fundamentally altering their makeup. For instance, by remaining in the MV framework, there is no natural means to generally commute products or express N-degree mappings using extended matrix-vector (EMV) algebra. Similarly, it is not clear how to extend some of the other featured algebras without altering them entirely. As an example, developing a fully associative and commutative framework for n-mode$^+$ notation, R-matrix notation, or array algebra would require abandoning the very means by which such formalisms specify inner-product indices. This can restrict their applications.

The arguments supporting Einstein notation are strengthened by considerations outside of those listed in Table 2.1. For one, Einstein notation enjoys a compactness in notation. For example, in Einstein notation an NT's degree is implicity specified by the number of indices. In comparison, n-mode$^+$ product notation requires additional notational markers, or tasks the reader with remembering the degree of each NT in question. Additionally, Einstein notation expresses products with an almost spartan amount of symbology. This contrasts with some other algebras. As an example, Lev-Ari's [72] demonstration of the usefulness of EMV algebra relies on 5 and 4 separate operators and identities, respectively. These symbolic burdens are magnified by notational differences in the literature. For instance Steeb and Hardy [109], Liu and Trenkler [73], and Lev-Ari [72] each provide different notations for the Hadamard product. This situation is mirrored in other operations, *e.g.*, Magnus [121] and Lev-Ari [72] employ differing notations for diagonal selection. Compounding this problem, extra symbols required by other algebras are not always alphanumeric. Because standard keyboards do not support many symbols outside of alphanumeric ones, notational compactness eases the transitional burden of implementing a solution on a computer. Since software plays such an important role in technical computing, this consideration looms large.

Apart from its minimal symbology, authors have also noted that Einstein notation aligns well with thinking in terms of computation. For instance, Pollock argues that Einstein notation strongly resembles the programming practice of indexed loops [154]. This is echoed by Vetter [66]. McCullagh outlines the merits of this computational focus for statistics applications [107]. Åhlander argues that Einstein notation enjoys both a compactness and a resemblance to the natural language of the problem, minimising programming errors [54]. This strong link likely explains why even in 1954 Backus and Herrick used an index notation as an example of how programmers could one day employ an algebra directly within code [45].

Einstein notation's appeal is broad enough that practitioners of some of the other featured algebras have advocated for its use. For instance, Pollock, a major figure in the econometrics field, asserts that identities seen in Kronecker algebra can be difficult, arguing that an index-based notation can better reveal key results [64, 154]. As another example, Vetter, who published papers describing and employing Kronecker algebra [155, 156], has also advocated for an index-based notation in certain contexts [66]. In the field of tensor decomposition, Harshman, one of the field's pioneers [83], published work advocating for Einstein notation [86] over the commonly used stretch or slice-wise matricisations seen in his field. In their explanation of n-mode notation, even de Lauthauwer *et al.* concede that Einstein notation is the most "versatile" [81] means to express multilinear operations.

The utility of Einstein notation has led practitioners from disparate fields, such as computer graphics [70], statistics [107], econometrics [154], psychometrics [86], numerical methods [54, 67, 68], computational chemistry [91, 92, 93, 94, 95], image processing [60], systems and control [113], and signal processing [66] to promote its use within their own fields. As such, Einstein notation enjoys a broad appeal that is not always enjoyed by other algebras.

Considered together with its excellent algebraic characteristics, such as associativity, commutativity, and compactness, the virtues of Einstein notation conspire to make an excellent case toward its use for applications beyond the natural purview of MV algebra. Nonetheless, as Table 2.1 makes clear, Einstein notation lacks key conventions on how it expresses entrywise products and inversions. By extending Einstein notation, one can formalise a powerful and versatile NT algebra.

3.2 Extensions

While the NT algebra we describe does build off of Einstein notation, not every concept in the notation's arsenal is employed. The origins of Einstein notation lie in tensor calculus, which means that its notation can be overly complex for many applications outside of physics or dynamics. Traditional Einstein notation expresses summation using an inner product by repeated indices—but one must lie in the upper index or contravariant position, and one must lie in the lower or covariant position. While covariant and contravariant indices are fundamental in tensor calculus, not all applications of NTs require the concept of contravariance and covariance. Thus, this work uses a simpler single-index-type notation, similar to that advocated by champions of Cartesian tensors [70, 157, 158, 159]. By default all indices lie in the lower position.

Elements in this convention are called NTs. Scalar NTs, N-degree collections of scalar values, represent the simplest elements in the algebra. Restricting attention for now to the scalar case, an NT $\underset{\mathbf{d}}{a_{\mathbf{i}}}$ is indexed by a sequence of natural numbers, $\mathbf{i} = \{i_1, i_2, \ldots i_N\}$. The range of each index is specified by the dimension sequence $\mathbf{d} = \{d_1, d_2, \ldots d_N\}$. The degree of a is N, while the dimensionality of a is the product of each of its index dimensions, $\prod_{k=1}^{N} d_k$, which can also be denoted $\dim(a)$.

Context often allows an array to be expressed as $a_{\mathbf{i}}$, suppressing dimensions. Depending on context, indices can be represented by the sequence $\mathbf{i}$, as in $a_{\mathbf{i}}$, or by individual indices, as in a_{ijk}. Operations are specified based on how operand indices match up. For instance, NTs can be added or subtracted entrywise provided matching indices have the same range:

$$\underset{M\times P\times R}{c_{ijk}} = \underset{P\times M\times R}{a_{kij}} + \underset{N\times P\times R}{b_{jki}} . \tag{3.1}$$

The set of all NTs of a certain dimension sequence can constitute a vector space with respect to entrywise addition and scalar multiplication [78].

The following subsections are dedicated to outlining some of the salient aspects of NT algebra, which include multiplication, unary operations, linear inversion, differentiation, and finally vector and matrix NTs.

3.2.1 Multiplication

NT algebra can represent inner, outer, and also entrywise products across arbitrary indices. This section will first outline the more well-known inner and outer product notation. Afterwards, the notation and some implications for the accommodation of entrywise products are discussed, including how to express ternary or higher inner products, which can be called n-ary inner products. The subsection concludes with a discussion of factoring NT products over addition and subtraction.

Restricting attention to first-degree NTs for now, an inner product, y, between two NTs a_i and x_i, each of dimension n, is the summation of the products of all corresponding elements:

$$y = \sum_{i=1}^{n} a_i x_i. \tag{3.2}$$

Einstein notation uses a repeated index to represent an inner product, where the dimensionality is understood from the context:

$$y = a_i x_i. \tag{3.3}$$

Einstein notation also provides a concise convention for outer products. An outer product, y_{ij}, of a_i and x_j, with dimensions $\{M, N\}$, is defined as the ordered product of all possible combinations of elements within the two NTs[1]:

$$y_{ij} = \begin{pmatrix} a_1x_1 & a_1x_2 & \dots & a_1x_\mathrm{N} \\ a_2x_1 & a_2x_2 & \dots & a_2x_\mathrm{N} \\ \vdots & \vdots & \ddots & \vdots \\ a_\mathrm{M}x_1 & a_\mathrm{M}x_2 & \dots & a_\mathrm{M}x_\mathrm{N} \end{pmatrix}. \tag{3.4}$$

Einstein notation simply uses differing indices to represent an outer product, where again the dimensionality is understood from the context:

$$y_{ij} = a_ix_j. \tag{3.5}$$

The analogues in classic MV algebra of the inner product and outer product of two vectors, **a** and **b**, are well known:

$$c = \mathbf{a}^\mathsf{T}\mathbf{b}, \tag{3.6}$$

$$\mathbf{C} = \mathbf{a}\mathbf{b}^\mathsf{T}. \tag{3.7}$$

For general N-degree NTs, Einstein notation can express any combination of outer and inner products. Thus, any linear mapping, $L : \mathbb{R}^{\dim(x)} \to \mathbb{R}^{\dim(y)}$ can be represented by the product:

$$y_\mathrm{i} = a_\mathrm{ij}x_\mathrm{j}. \tag{3.8}$$

Einstein notation can be readily extended to support entrywise, *i.e.*, element-wise, products. This operation can occasionally be found in tensor calculus, where it is described as summation suppression [69]. Some conventions are to distinguish, by a parenthesis [69,123], an underline [68], or a subscripted parenthesis [70], the indices whose summation is suppressed.

Using the underline convention, the entrywise product of two first-degree NTs is:

$$y_i = a_{\underline{i}}x_{\underline{i}}. \tag{3.9}$$

Underlines corresponding to *executed* entrywise products are understood to be removed after a product grouping has been executed. Product groupings are separated by addition, subtraction, or parentheses, *e.g.*, the following expressions are equivalent:

$$y_i = a_{\underline{i}}x_{\underline{i}}b_{\underline{i}} = (a_{\underline{i}}x_{\underline{i}})b_{\underline{i}}. \tag{3.10}$$

Another convention to support entrywise products is based on assignment [54, 95], whereby all repeated indices are assumed to be entrywise products unless they do *not* appear on the left-hand side of an assignment. This implicit scheme mean products are ambiguous without assignments, making it burdensome to perform substitutions or to parse long expressions. For this reason, we prefer explicitly designating entrywise products.

[1](3.4) displays the elements of y_{ij} using the standard row/column matrix convention, but this is an arbitrary layout choice that is not a part of NT algebra.

There are both symbolic and computational advantages of incorporating an entrywise product. Without it, artificial constructs must be used. For example, consider the following expression which incorporates an entrywise product between two first-degree NTs:

$$c_i = a_{\underline{i}} b_{\underline{i}} + a_i. \tag{3.11}$$

Without an entrywise product notation, the expression in (3.11) would have to be represented by composing one of the operands into a diagonal second-degree NT:

$$c_i = \tilde{a}_{ij} b_j + a_i, \tag{3.12}$$

where $\tilde{a}_{ij}$ is a second-degree NT with entries in its main diagonal consisting of elements of a_i:

$$\tilde{a}_{ij} = \begin{pmatrix} a_1 & 0 & \dots & 0 \\ 0 & a_2 & \dots & \vdots \\ \vdots & \dots & \ddots & 0 \\ 0 & \dots & 0 & a_n \end{pmatrix}. \tag{3.13}$$

A well-known example of such a construction in MV algebra is the diagonal matrix seen in singular value decomposition (SVD) computations.

One key problem with the formulation in (3.12) is that two different constructions of the same underlying values are used. In addition, in (3.11) all elements are indexed by i, which correctly represents that in the expression all elements correspond to the same index. However, in (3.12), a different index is used for b, which is an artificial distinction introduced in order to represent entrywise products using a combination of inner and outer products. Finally, transforming a first-degree NT into an 'equivalent' second-degree NT in order to provide entrywise operations is an unnatural representation of the underlying data.

In addition, there is an important computational rationale in using entrywise products. Assuming the dimension of a and b is n, the entrywise product in (3.9) should consume $O(n)$ arithmetic operations. In contrast, using the construction in (3.13) to execute the product is $O(n^2)$ in computational complexity if dense NTs are used. If a sparse implementation is used then the computational complexity would remain $O(n)$ in either case. However, sparse representations are accompanied by overhead, which adds to the computational cost even if the order of complexity remains $O(n)$. These computational differences are magnified when high-degree NTs are considered.

Entrywise products hold an important relationship to summations over more than two indices, such as ternary inner products:

$$a_i b_i c_i \equiv \sum_{i=1}^{n} a_i b_i c_i. \tag{3.14}$$

While not always framed as n-ary inner products, operations like (3.14) are key operations for canonical-polyadic (CP) decomposition, *e.g.*, equation (3.1) of Kolda and Bader [83]; linear estimation problems, *e.g.*, example (4.3) of Liu and Trenkler [73]; differential geometry, *e.g.*, equation (3) of Bolton [71]; support vector machines, *e.g.*, equation (14) of Burges [160]; and tensor optimisation methods, *e.g.*, equation (3.6) of Schnabel and Frank [161]. Traditional tensor calculus also encounters this need [69].

Such products are challenging to work with, as they break associativity [123]. For instance, the following two expressions are not equivalent to each other, nor are they equivalent to (3.14):

$$a_i(b_i c_i) \neq (a_i b_i)c_i. \tag{3.15}$$

For this reason, many Einstein-notation conventions forbid a summation over more than one index or make special notational exceptions [54]. The assignment convention for implicit entrywise products [54] also allows n-ary inner products, but not in a way that enables them to be broken into a sequence of binary operations, which is often desirable for subsequent implementations on a computer. This convention would also make parentheses meaningless in (3.15), which despite Åhlander's assertions [54], does break associativity.

Explicit entrywise products can express n-ary inner products and, combined with a simple association identity, can retain associativity [123]. In short, given a ternary inner product, such as (3.14), isolating operands can be accomplished using:

$$a_i b_i c_i = (a_{\underline{i}} b_{\underline{i}})c_i = a_i(b_{\underline{i}} c_{\underline{i}}) = (a_{\underline{i}} c_{\underline{i}})b_i. \tag{3.16}$$

With the entrywise product defined, an NT product can consist of any combination of N-degree inner, entrywise, and outer products.

As mentioned, Einstein notation, and thus NT algebra, is strongly associated with computation [107]. Supporting this, the computational complexity of any product of dense NTs, *e.g.*,

$$y_i = a_{i\underline{j}k} x_{\underline{j}\ell i}, \tag{3.17}$$

can be read by simply multiplying the ranges of each distinct index. If the range of each index in (3.17) is N, then the complexity of the product is N^4. This can be extended to more complicated expressions involving many operands, further illustrating NT algebra's affinity for computation.

Like MV algebra, NT multiplication distributes over addition. NT algebra expressions can also be decomposed using common factors, provided the indices in the common array are multiplied in identical manners, *e.g.*, such as the following expressions:

$$a_{i\underline{j}k} x_{k\underline{j}} + b_{\ell i\underline{j}} x_{\ell\underline{j}} = (a_{i\underline{j}k} + b_{ki\underline{j}})x_{k\underline{j}}, \tag{3.18}$$

where the index ℓ has been changed to k on the right hand side, since both are dummy indices whose symbols can be changed.

If the indices of the common array are not multiplied in the same manner in each subexpression, this can be due to one or a combination of three possibilities. Table 3.1

Table 3.1: Factoring out common NT terms. To factor out common terms between expressions, the type of NT product must be identical. When they are not identical common terms can still be factored using three rules.

Scenario	Solution	Example
An index undergoes an *outer* product in one subexpression and an *inner* product in another.	Change the *outer* product to an *inner* product by multiplying by a Kronecker delta array (*e.g.*, $\delta_{i'i}$).	$a_j x_i + b_{ijk} x_k$ $= a_j x_k \delta_{ki} + b_{ijk} x_k$ $= a_j \delta_{ki} x_k + b_{ijk} x_k$ $= (a_j \delta_{ki} + b_{ijk}) x_k.$
An index undergoes an *entrywise* product in one subexpression and an *inner* product in another.	Change the *entrywise* product to an *inner* product by multiplying by a Kronecker delta array (*e.g.*, $\delta_{i'i}$).	$a_{\underline{i}} x_{\underline{i}} + b_{ij} x_j$ $= (a_{\underline{j}} x_{\underline{j}}) \delta_{ji} + b_{ij} x_j$ $= (a_{\underline{j}} \delta_{\underline{j}i}) x_j + b_{ij} x_j$ $= (a_{\underline{j}} \delta_{\underline{j}i} + b_{ij}) x_j.$
An index undergoes an *outer* product in one subexpression and an *entrywise* product in another.	Change the *outer* product to an *entrywise* product by multiplying by a unit array (*e.g.*, 1_i).	$a_j x_i + b_{j\underline{i}} x_{\underline{i}}$ $= a_j x_{\underline{i}} 1_{\underline{i}} + b_{j\underline{i}} x_{\underline{i}}$ $= a_j 1_{\underline{i}} x_{\underline{i}} + b_{j\underline{i}} x_{\underline{i}}$ $= (a_j 1_{\underline{i}} + b_{j\underline{i}}) x_{\underline{i}}.$

outlines each scenario and the approach to factor such expressions. All of the profiled scenarios rely on the Kronecker delta NT, *i.e.*, δ_{ij}, which is defined as:

$$\delta_{ii'} = \begin{cases} 1, & i = i' \\ 0, & \text{otherwise} \end{cases} . \tag{3.19}$$

Just as with identity matrices in MV algebra, Kronecker delta NTs serve a fundamental purpose in NT algebra. The first scenario has also been described within the MV paradigm by Brewer [162], who also provides an efficient means to solve an analogous system of equations involving second-degree parameters and independent variables.

3.2.2 Unary Operations

As with MV algebra and tensor calculus, NT algebra accommodates unary operations. One common such operation is contraction, which is the summation of corresponding elements across two NT indices:

$$b = a_{ii} \equiv a_{11} + a_{22} + \ldots a_{nn}. \tag{3.20}$$

This operation can also be expressed as a binary operation using the Kronecker delta NT:

$$b = a_{ii} = a_{ii'} \delta_{ii'}. \tag{3.21}$$

In MV algebra, the equivalent to contraction is the trace operation, expressed as `trace(A)` in MATLAB. However, unlike trace, contraction can be applied to arbitrary indices of NTs of arbitrary degree.

While contraction also holds an important place in tensor calculus, the ability to represent entrywise products provides NT algebra with an additional unary operation. Joseph coined the term *attraction* to describe the operation [123]. Restricting attention to a second-degree NT, attraction is defined as selecting the diagonal elements from two indices in the NT. In MATLAB notation, this corresponds to the operation `diag(A)`. Despite its usefulness, this is not a well-recognised operation within standard MV algebra, but it has been treated within EMV [72, 102, 103, 121]. This extension expresses attraction using a matrix product between a very large and sparse selection matrix and a vectorised form of the matrix or by using a non-algebraic operation similar to `diag(A)`.

Using the entrywise-product underline symbol, the unary operation of attraction can be represented as:

$$b_i = a_{\underline{ii}}. \tag{3.22}$$

The attraction operation generalises to an N-degree NT, and can also be easily combined with the more familiar unary contraction operation:

$$b_{jk} = a_{iji\underline{kk}}. \tag{3.23}$$

As with the contraction operator, attraction can be expressed as a binary operation:

$$b_i = a_{\underline{ii}} = a_{\underline{i}i'}\delta_{\underline{i}i'}. \tag{3.24}$$

Explicitly accommodating entrywise operations avoids having to use the non-intuitive selection matrices seen in EMV algebra and does not require flattening second-degree data. Moreover, unlike the EMV algebra approach, which is difficult to generalise to higher degrees, attraction can be applied across NTs of arbitrary degree.

The operations of contraction and attraction can also be used to provide a relationship between outer products and inner or entrywise products [123]. An outer product followed by contraction is equivalent to an inner product, while following an outer product with attraction is equivalent to an entrywise product. This can be expressed using binary operations, which has already been demonstrated above in the explanation on how to factor out common terms.

3.2.3 Solution of Linear Equations

An NT product can represent a single linear mapping or, if entrywise products are involved, it can represent a sequence of linear mappings. Like MV algebra, inverting these linear mappings is a fundamental concept. Unlike MV algebra, there may be multiple inverses for an arbitrary NT, as an NT on its own does not define a unique linear mapping [65, 86, 123].

If one wishes to denote the inverse of an NT, one must also specify which linear mapping it represents. Omitting entrywise products from consideration, an NT represents a linear mapping by grouping one subsequence of indices as dummy indices, and the remaining as

Table 3.2: Linear mappings and inverse representations using NT algebra. All examples assume no collinearity in the linear mappings.

Example	NT Product	Solving for x	Inverse Representation
1	$a_{ijk\ell}x_{k\ell} = y_{ij}$	$x_{k\ell} = a^{-1}_{ijk\ell}y_{ij}$	$a^{-1}_{ijk\ell} = \delta_{ii'}\delta_{jj'}/a_{i'j'k\ell}$ $a^{-1}_{ijk\ell} = \delta_{kk'}\delta_{\ell\ell'}/a_{ijk'\ell'}$
2	$a_{ijk\ell}x_{j\ell} = y_{ik}$	$x_{j\ell} = a^{-1}_{ijk\ell}y_{ik}$	$a^{-1}_{ijk\ell} = \delta_{ii'}\delta_{kk'}/a_{i'jk'\ell}$ $a^{-1}_{ijk\ell} = \delta_{jj'}\delta_{\ell\ell'}/a_{ij'k\ell'}$
3	$a_{ijk}x_{jk} = y_i$	$x_{jk} = a^{-1}_{ijk}y_i$	$a^{-1}_{ijk} = \delta_{i'i}/a_{i'jk}$ $a^{-1}_{ijk} = \delta_{jj'}\delta_{kk'}/a_{ij'k'}$
4	$a_{\underline{ij}}x_{\underline{ij}} = y_{ij}$	$x_{ij} = a^{-1}_{\underline{ij}}y_{\underline{ij}}$	$a^{-1}_{ij} = 1_{\underline{ij}}/a_{\underline{ij}}$
5	$a_{ij\underline{k}}x_{j\underline{k}} = y_{ik}$	$x_{jk} = a^{-1}_{ij\underline{k}}y_{i\underline{k}}$	$a^{-1}_{ijk} = \delta_{ii'}1_{\underline{k}}/a_{i'j\underline{k}}$ $a^{-1}_{ijk} = \delta_{jj'}1_{\underline{k}}/a_{ij'\underline{k}}$

free indices. Like matrix inverses, a composition of an NT and one of its inverses results in an identity mapping. This remains true if the roles of dummy and free indices are reversed.

This is illustrated by the following expressions:

$$a^{-1}_{ij'}a_{ij}x_j = \delta_{j'j}x_j = x_{j'}, \tag{3.25}$$
$$a^{-1}_{i'j}a_{ij}x_i = \delta_{i'i}x_i = x_{i'}. \tag{3.26}$$

Here, the identity mapping is represented by the Kronecker delta NT. In (3.25) and (3.26), the dummy indices switch, differing from MV algebra, which typically restricts second, or column indices, to act as summation indices[2]. As a result, there are two possible identities, *i.e.*, $\delta_{j'j}$ and $\delta_{i'i}$, even though there is only one possible way to divide two indices into two proper subsequences.

In the high-degree case, there can be more than one way to create two proper subsequences, so inverses must be represented by specifying a corresponding identity mapping. However, like the second-degree case, there are two possible, and equivalent, identity mappings for each specified inverse. These notational issues can be avoided if the inverse is specified in the context of a solution of equations, which removes all ambiguity.

Table 3.2 illustrates this, providing several high-degree examples. The first two examples

[2]An exception is post-multiplication, but even in this case it is commonly performed using a transposed matrix, *e.g.*, SVD or Lyapunov equations.

illustrate different mappings and inverse representations originating from the same fourth-degree NT. As is also demonstrated, the context of a solution of equations implicitly specifies the division between dummy and free indices. The third example demonstrates that an inverse may be executed across an asymmetric sequence of indices. For this to happen, the dimensionality of the two subsequences must be identical and, like the symmetric case, there must be no collinearity. The fourth example introduces the identity mapping for entrywise products—an NT of all ones, *i.e.*, 1_{ij}. As the example also illustrates, should a linear mapping only consist of entrywise products there is only one possible identity mapping. The fifth example demonstrates a scenario involving all three types of products.

NT algebra also supports Moore-Penrose (MP) inverses for the solution of over and under-determined sets of linear equations [63]. Like NT inverses, when possible it is better to represent the MP inverse as part of an expression. Assuming one of the subsequences is fully-ranked, this can be easily done, as in MV algebra, by replacing the -1 superscript with a $+$ symbol.

The MP inverse can also be represented in isolation. Unlike NT inverses, the roles of dummy and free indices are not so trivially reversed because they are entangled with matters of under or over-determinedness. Instead of the two possibilities seen in Table 3.2, the Kronecker delta arrays should only correspond to the fully-ranked indices. In cases where neither subsequence is fully-ranked, the MP inverse can be realised using singular value decomposition [63], but a differing representation would be required.

3.2.4 Symbolic Differentiation

Differentiating expressions is an important concept for work involving high-degree data. For instance, gradients are a common element within tensor decomposition algorithms [83]. Matrix calculus [63], the setting of much of EMV algebra, is concerned with derivatives of vector- or matrix-valued functions. Differentiation using NT algebra closely aligns with Einstein notation conventions, except that entrywise products can come into play and coordinate basis vectors are often not necessary since the data is typically numeric rather than geometric.

Consider a first-degree NT, x_i. The derivative of each element of x_i with respect to *every* element in the same NT can be expressed as:

$$\frac{\partial x_i}{\partial x_{i'}} = \delta_{ii'}. \tag{3.27}$$

Note that under this convention, the index label, *i.e.*, i', must be specified in the partial derivative. This convention readily generalises to high-degree NTs, as in the following example:

$$\frac{\partial x_{ij}}{\partial x_{i'j'}} = \delta_{ii'}\delta_{jj'}. \tag{3.28}$$

Table 3.3: Partial derivatives of NT expressions.

Expression	Derivative
$y_{k\ell ij} = a_{k\ell}x_{ij}$	$\dfrac{\partial y_{k\ell ij}}{\partial x_{i'j'}} = a_{k\ell}\delta_{ii'}\delta_{jj'}$
$y_{kj} = a_{ki}x_{ij}$	$\dfrac{\partial y_{kj}}{\partial x_{i'j'}} = a_{ki'}\delta_{jj'}$
$y_{ij} = a_{\underline{ij}}x_{\underline{ij}}$	$\dfrac{\partial y_{ij}}{\partial x_{i'j'}} = a_{\underline{ij}}\delta_{\underline{i}i'}\delta_{\underline{j}j'}$
$y = a_{ij}x_{ij}$	$\dfrac{\partial y}{\partial x_{i'j'}} = a_{i'j'}$

Such a convention is useful when taking derivatives of expressions that include an outer, entrywise, or inner product of an array. Table 3.3 provides examples of partial derivatives of high-degree NT expressions. In the examples involving an entrywise product, corresponding indices in the Kronecker delta array are also underlined.

Many of the expressions in Table 3.3 are unavailable in traditional matrix algebra, particularly those incorporating high-degree NTs or entrywise products. This can be problematic for many formulations, such as Taylor series, that require partial-derivatives. Even when working with lower-degree data, such as vector-valued functions, MV algebra cannot naturally represent any Taylor polynomials beyond the linear term.

For certain expressions, such as the last row of Table 3.3, the use of a different index, *i.e.*, i' instead of i, may seem needless. However, their importance becomes clear when they are incorporated into larger expressions. A good example of this is computing the sum-squared error (SSE) of a linear regression estimate. Using a first-degree system for simplicity's sake, this can be expressed as:

$$SSE = (b_i - a_{ij}x_j)(b_i - a_{ij}x_j). \tag{3.29}$$

Using the product rule in calculus, the partial derivative of the SSE with respect to x_ℓ is:

$$\begin{aligned}\frac{\partial SSE}{\partial x_\ell} &= (a_{ij}\delta_{j\ell})(b_i - a_{ij}x_j) + (b_i - a_{ij}x_j)(a_{ij}\delta_{j\ell}), &(3.30)\\ &= 2(a_{ij}\delta_{j\ell})(b_i - a_{ij}x_j), &(3.31)\\ &= 2a_{i\ell}(b_i - a_{ij}x_j), &(3.32)\end{aligned}$$

The expressions in (3.63) and (3.66) correspond to the well-known MV algebra expressions:

$$\begin{aligned}SSE &= (\mathbf{b} - \mathbf{A}\mathbf{x})^\mathsf{T}(\mathbf{b} - \mathbf{A}\mathbf{x}), &(3.33)\\ \frac{\partial SSE}{\partial \mathbf{x}} &= 2\mathbf{A}^\mathsf{T}(\mathbf{b} - \mathbf{A}\mathbf{x}). &(3.34)\end{aligned}$$

By explicitly denoting the partial derivative index, which is different than the original expression index, products between identically-valued NTs, *i.e.*, a_{ij}, can be correctly evaluated. This bookkeeping is particularly important when working with expressions incorporating N-degree NTs, as properly matching up indices in these cases is not trivial and conventional MV algebra does not provide its own version.

3.2.5 Nonlinear Functions

Nonlinear functions serve important roles in scientific computing. For this reason, MV algebra supports notation for nonlinear functions, *e.g.*, matrix-valued functions. Analogous notation is also important for NT algebra. This work will assume nonlinear functions act on NTs entrywise, *e.g.*,

$$f(x_k) = \sqrt{x_k}, \tag{3.35}$$

where the square root operation acts entrywise on x_k.

Taking the derivative of nonlinear NT functions is very similar to taking the derivative of linear NT functions. To see this, first consider the partial differentiation of a first-degree NT with itself,

$$\frac{\partial x_i}{\partial x_{i'}}. \tag{3.36}$$

In Section 3.2.4, this derivative was evaluated as

$$\delta_{ii'}. \tag{3.37}$$

In actuality, the derivative could be evaluated as

$$1_{\underline{i}}\delta_{\underline{i}i'}, \tag{3.38}$$

which is equivalent to (3.37). When nonlinear functions come into play, a generalisation of the formulation in (3.38) can be used, *e.g.*,

$$\frac{\partial\sqrt{x_i}}{\partial x_{i'}} = \frac{1}{2}(x_{\underline{i}})^{-\frac{1}{2}}\delta_{\underline{i}i'}, \tag{3.39}$$

where $1_{\underline{i}}$ has been replaced by $\frac{1}{2}(x_{\underline{i}})^{-\frac{1}{2}}$. As can be seen, due to the entrywise nature of the non-linear functions treated here, entrywise products play a key role in differentiation.

Table 3.4 illustrates examples on how to evaluate the partial derivative of larger NT expressions that incorporate non-linear functions and linear inner, entrywise, or outer products. As can be seen by comparison the rules are very similar to their linear analogues.

3.2.6 Vector and Matrix NTs

So far attention has been focused on scalar NTs. However, equally valid are vector and matrix NTs, which are denoted using lower-case and upper-case boldface roman letters, respectively. Examples of each are given below:

$$\mathbf{a}_{ij}^{N}, \tag{3.40}$$

$$\mathbf{A}_{ij}^{M\times N}, \tag{3.41}$$

Table 3.4: Linear and nonlinear NT expressions and their partial derivatives.

Scenario	Expression	Partial Derivative
Inner Product	$a_{ij}x_j$	$\dfrac{\partial a_{ij}x_j}{\partial x_{j'}} = a_{ij}(1_{\underline{j}}\delta_{\underline{j}j'}) = a_{ij}\delta_{jj'} = a_{ij'}$
	$a_{ij}\sqrt{x_j}$	$\dfrac{\partial a_{ij}\sqrt{x_j}}{\partial x_{j'}} = \dfrac{1}{2}a_{ij}(x_{\underline{j}}^{-\frac{1}{2}}\delta_{\underline{j}j'}) = \dfrac{1}{2}a_{i\underline{j'}}x_{\underline{j'}}^{-\frac{1}{2}}$
Entrywise Product	$a_{i\underline{j}}x_{\underline{j}}$	$\dfrac{\partial a_{i\underline{j}}x_{\underline{j}}}{\partial x_{j'}} = a_{i\underline{j}}1_{\underline{j}}\delta_{\underline{j}j'} = a_{i\underline{j}}\delta_{\underline{j}j'}$
	$a_{i\underline{j}}\sqrt{x_{\underline{j}}}$	$\dfrac{\partial a_{i\underline{j}}\sqrt{x_{\underline{j}}}}{\partial x_{j'}} = \dfrac{1}{2}a_{i\underline{j}}x_{\underline{j}}^{-\frac{1}{2}}\delta_{\underline{j}j'}$
Outer Product	a_ix_j	$\dfrac{\partial a_ix_j}{\partial x_{j'}} = a_i1_{\underline{j}}\delta_{\underline{j}j'} = a_i\delta_{jj'}$
	$a_i\sqrt{x_j}$	$\dfrac{\partial a_i\sqrt{x_j}}{\partial x_{j'}} = \dfrac{1}{2}a_ix_{\underline{j}}^{-\frac{1}{2}}\delta_{\underline{j}j'}$

where the dimensions of the row and column indices, not to be confused with the NT indices, are given as superscripts. The convention of column vectors are used for this work, but it is equally possible to use a row-vector convention. For the purposes of this work, MV NTs are kept homogenous, meaning each MV within the NT has the same dimensions. Context often allows the superscripts to be dropped.

Such NTs follow the same rules as their scalar cousins, with the caveat that arithmetic operations performed on the elements must follow standard MV algebra rules. For instance, in the following product,

$$\mathbf{C}_{ij\ell}^{M\times N} = \mathbf{A}_{ik\underline{\ell}}^{M\times P}\mathbf{B}_{kj\underline{\ell}}^{P\times N}, \tag{3.42}$$

in addition to the constraint that the k and ℓ indices in each operand hold the same range, the product between matrices of $\mathbf{A}_{ik\ell}$ and $\mathbf{B}_{kj\ell}$ must be legal, *i.e.*, the number of columns of $\mathbf{A}_{ik\ell}$ must equal the number of rows of $\mathbf{B}_{kj\ell}$. Unless the matrix NTs are square, the commuted version,

$$\mathbf{C}_{ij\ell} = \mathbf{B}_{kj\underline{\ell}}^{P\times N}\mathbf{A}_{ik\underline{\ell}}^{M\times P}, \tag{3.43}$$

is not legal, and typically not equivalent, meaning that matrix NTs do not share the same care-free commutativity of scalar NTs. Analogous implications apply with vector NTs or mixtures of the two.

Being able to express MV NTs means that using NT algebra does in no way bar the use of MV algebra when suitable. Thus, when appropriate the benefits of MV algebra, *e.g.*, minimal bookkeeping, notational simplicity, and other conveniences, can be enjoyed.

To really be able to use MV algebra, NT algebra, or a mixture of the two whenever one wishes, a mapping between scalar NTs and MV NTs is needed. The key to these mappings are vector basis arrays, denoted $\underset{N}{\mathbf{e}_i^N}$, which is an ordered sequence of the standard-basis vectors for $\Re^N$:

$$\underset{N}{\mathbf{e}_i^N} = \left\{ \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{pmatrix}, \ldots \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \right\}. \tag{3.44}$$

The underset and superscript is usually suppressed for $\mathbf{e}_i$.

Focusing momentarily on vectors, one can map any scalar NT to a vector NT using $\mathbf{e}_i$:

$$\mathbf{a}_{ij} = a_{ijk}\mathbf{e}_k, \tag{3.45}$$

$$= \begin{pmatrix} a_{ij1} & \ldots & a_{ijN} \end{pmatrix}^{\mathsf{T}}. \tag{3.46}$$

The reverse mapping is expressed as,

$$a_{ijk} = \mathbf{e}_k^{\mathsf{T}}\mathbf{a}_{ij}. \tag{3.47}$$

The basis vectors can also map more than one index into a vector and back again:

$$\mathbf{a}_i = a_{ijk}\mathbf{e}_{jk}, \tag{3.48}$$

$$a_{ijk} = \mathbf{e}_{jk}^{\mathsf{T}}\mathbf{a}_i. \tag{3.49}$$

If M and N represent the range of j and k respectively, then the basis vectors are composed of the standard basis for $\Re^P$, where $P = M \times N$. Individual vectors within $\mathbf{e}_{jk}$ are ordered using any valid, but fixed, lexicographical ordering of j and k. Each vector within $\mathbf{a}_i$ is a 'flattened' representation of a_{ijk}.

Vector NTs enjoy important notational conveniences, such as the capability to concatenate. For example, in MV algebra, one can combine two vectors into a third using the simple expression $\mathbf{c} = \left(\mathbf{a}^{\mathsf{T}}\ \mathbf{b}^{\mathsf{T}}\right)^{\mathsf{T}}$. Two vector-valued NTs can be similarly combined,

$$\mathbf{c}_{ij} = \begin{pmatrix} \mathbf{a}_{ij}^{\mathsf{T}} & \mathbf{b}_{ij}^{\mathsf{T}} \end{pmatrix}^{\mathsf{T}}, \tag{3.50}$$

$$= \begin{pmatrix} a_{ij1} & \ldots & a_{ijM} & b_{ij1} & \ldots & b_{ijN} \end{pmatrix}^{\mathsf{T}}, \tag{3.51}$$

where M and N denote the size of the vectors in $\mathbf{a}_{ij}$ and $\mathbf{b}_{ij}$ respectively.

Mappings between scalar NTs and their matrix-valued cousins follow the same process as vector NTs, except that two basis vector NTs are needed, one for row indices and one for the column indices:

$$\mathbf{A}_{ij} = a_{ijk\ell}\mathbf{e}_k\mathbf{e}_\ell^{\mathsf{T}}, \tag{3.52}$$

$$\mathbf{A}_{ij} = \begin{pmatrix} a_{ij11} & \ldots & a_{ij1N} \\ \vdots & \ddots & \vdots \\ a_{ijM1} & \ldots & a_{ijMN} \end{pmatrix}. \tag{3.53}$$

The reverse mapping is expressed as,

$$a_{ijk\ell} = \mathbf{e}_k^{\mathsf{T}} \mathbf{A}_{ij} \mathbf{e}_\ell. \tag{3.54}$$

Analogous to (3.48) and (3.49), basis NTs can also map more than one index to the rows or columns. As well, matrix NTs can also be concatenated together provided matrix dimensions appropriately match, *e.g.*,

$$\mathbf{E}_{ij} = \begin{pmatrix} \mathbf{A}_{ij}^{M\times N} & \mathbf{B}_{ij}^{M\times P} \\ \mathbf{C}_{ij}^{Q\times N} & \mathbf{D}_{ij}^{Q\times P} \end{pmatrix}. \tag{3.55}$$

3.3 Selected Exemplars

The extensions to Einstein notation discussed in the previous section proffers algebraic capabilities key to several exemplars. In Chapter 6, the merits of NT algebra vis-à-vis prominent computer vision exemplars, falling under the category of special linear mappings, are examined in detail. To broaden the scope of this discussion beyond computer vision, this section discusses three additional important exemplars.

Section 3.3.1 begins this exposition by focusing on tensor decomposition, an important exemplar falling within the mappings beyond linear category and a topic that has emerged as a major research focus in technical computing [83]. This is followed by Section 3.3.2, which highlights the difficulties surrounding parameter estimation involving entrywise products, and the opportunities for NT algebra to stake a claim for such exemplars. Finally, Section 3.3.3 discusses separable nonlinear least squares (SNLS), a major technique in optimisation that involves high-degree data, and one that is featured in the author's own computer vision work [163, 164].

3.3.1 Tensor Decomposition

Tensor decomposition provides instructive instances where entrywise products, n-ary inner products, linear inversion, and differentiation all play a role. This can be seen by considering the representation and the computation of high-degree versions of the SVD.

As mentioned, entrywise products and ternary inner products arise often in MV algebra, but in the form of diagonal matrices. The SVD is probably the most well-known example, which is expressed commonly as

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathsf{T}}, \tag{3.56}$$

where $\boldsymbol{\Sigma}$ is understood to be diagonal. A ternary inner product can express the SVD without embedding the singular values within a matrix,

$$a_{ik} = \sigma_j u_{ij} v_{kj}. \tag{3.57}$$

The benefits of framing SVD using NT algebra manifest most strongly when generalising (3.57) to higher degrees, where it takes on the form of CP decomposition, one of the two major forms of tensor decompositions [83]. For instance, the expression in (3.57) can be easily extended to third-degree data:

$$a_{ik\ell} = \sigma_j u_{ij} v_{kj} w_{\ell j}. \tag{3.58}$$

In tensor decomposition literature, (3.58) is often expressed by flattening high-degree data into matrices in order to use Khatri-Rao matrix products or by using explicit summation symbols [83]. The former approach removes the data from its natural domain, while the latter approach does not lend itself well to algebraic manipulation. As will be outlined, with the entrywise product and the association identity in hand, n-ary products are readily manipulated algebraically, allowing practitioners to explore efficiencies or derive solutions to NT problems. Moreover, unlike the Khatri-Rao matricisation approach, these benefits can be extended to expressions other than (3.58).

The alternating least squares (ALS) algorithm, a workhorse in tensor decomposition [83], showcases how n-ary inner products can be manipulated within NT algebra. In the third-degree case, the ALS algorithm aims to compute u_{ij}, v_{kj}, $w_{\ell j}$, and σ_j. It does so by alternately solving for each of the factors while keeping the others constant. For instance, to solve for u_{ij}, one can easily isolate the factor,

$$a_{ik\ell} = u_{ij}(v_{k\underline{j}} w_{\ell \underline{j}}), \tag{3.59}$$

where σ_j has been absorbed within the factors. Solving for u_{ij} is straightforward:

$$u_{ij} = (v_{k\underline{j}} w_{\ell \underline{j}})^+ a_{ik\ell}. \tag{3.60}$$

Similarly, linear least-squares solutions for the other factors can be formulated:

$$v_{kj} = (u_{i\underline{j}} w_{\ell \underline{j}})^+ a_{ik\ell}, \tag{3.61}$$

$$w_{\ell j} = (u_{i\underline{j}} v_{k \underline{j}})^+ a_{ik\ell}. \tag{3.62}$$

The ALS algorithm alternates between executing the three linear least-squares solutions above. Once the algorithm converges, the σ_j factor can be computed by normalizing the three second-degree factors [83].

While this is algebraically correct, there are more efficient means to arrive at the same solution. The NT differentiation notation, along with the associated bookkeeping, can reveal such efficiencies. To see this, the third-degree ALS algorithm in CP decomposition can be framed as minimising the following expression:

$$SSE = (a_{ik\ell} - u_{ij} v_{kj} w_{\ell j})(a_{ik\ell} - u_{ij} v_{kj} w_{\ell j}), \tag{3.63}$$

$$= (a_{ik\ell} - u_{ij}(v_{k\underline{j}} w_{\ell \underline{j}}))(a_{ik\ell} - u_{ij}(v_{k\underline{j}} w_{\ell \underline{j}})). \tag{3.64}$$

As mentioned, the ALS algorithm minimises (3.64) by alternating between solving for one of the three NT factors while keeping the remaining constant, *e.g.*, solving for u_{ij} takes the following form:

$$\frac{\partial SSE}{\partial u_{i'j'}} = -2(\delta_{ii'}\delta_{jj'}(v_{k\underline{j}}w_{k\underline{j}}))(a_{ik\ell} - u_{ij}(v_{k\underline{j}}w_{\ell\underline{j}})), \tag{3.65}$$

$$= -2v_{k\underline{j'}}w_{\ell\underline{j'}}(a_{i'k\ell} - u_{i'j}(v_{k\underline{j}}w_{\ell\underline{j}})). \tag{3.66}$$

Setting (3.66) to zero and solving for $u_{i'j}$ leads to,

$$v_{k\underline{j'}}w_{\ell\underline{j'}}(v_{k\underline{j}}w_{\ell\underline{j}})u_{i'j} = v_{k\underline{j'}}w_{\ell\underline{j'}}a_{i'k\ell}, \tag{3.67}$$

$$((v_{k\underline{j'}}v_{k\underline{j}})(w_{\ell\underline{j'}}w_{\ell\underline{j}}))u_{i'j} = v_{k\underline{j'}}a_{i'k\ell}w_{\ell\underline{j'}}, \tag{3.68}$$

$$u_{i'j} = ((v_{k\underline{j'}}v_{k\underline{j}})(w_{\ell\underline{j'}}w_{\ell\underline{j}}))^{+}(v_{k\underline{j'}}a_{i'k\ell}w_{\ell\underline{j'}}), \tag{3.69}$$

where the rearrangement on the left-hand side in (3.68) produces the commonly used identity to efficiently calculate the pseudo-inverse of the Khatri-Rao product of two or more matrices as a natural matter of course. See Kolda and Bader for more details on the application of this identity to CP decomposition [83]. This result is produced by using the associativity and commutativity of NT algebra to rearrange operands so that inner products are executed with higher priority than entrywise or outer products. This can be contrasted with Lev-Ari's 3-page proof using Khatri-Rao products [72] that involves the auxiliary operations of vectorisation, selection matrices, diagonal extractions, and requisite identities involving these operations. The contrast between the two approaches is dealt with in more detail in the next subsection.

3.3.2 Parameter Estimation

As Liu and Trenkler [73] note, entrywise products have been receiving increased attention in several fields. As MV algebra is not inherently equipped to accommodate entrywise products, these types of operations are introduced within EMV algebra. Despite notable applications, EMV algebra has not enjoyed widespread adoption in technical computing packages [73]. Some of these extended operations include generalised Kronecker products [116], Katri-Rao products [162], and Hadamard products [73]. However, each operation requires a dedicated symbol, which complicates the algebra and makes it difficult to manipulate. Algebraically, commuting or associating these products often involves nonintuitive identities.

The following example provides a good comparison between the respective strengths of EMV algebra and NT algebra. It has similar attributes to the ALS algorithm described in the previous subsection, but its motivation stems from special linear mappings and not mappings beyond linear.

As described by Liu and Trenkler [73], Lev-Ari [72] outlines a scenario in antenna signal processing where scattering coefficients must be estimated from observations. This problem can be expressed using NT algebra and a ternary inner product:

$$h_{ij} = a_{ik}x_k b_{jk}, \tag{3.70}$$

where x_k represents the set of coefficients to estimate. Assuming the system is fully ranked for the k index, this can be solved using the association identity of (3.16):

$$(a_{i\underline{k}}b_{j\underline{k}})x_k = h_{ij}, \tag{3.71}$$

$$x_k = (a_{i\underline{k}}b_{j\underline{k}})^{+}h_{ij}. \tag{3.72}$$

This pseudo-inverse is very similar to those found in the ALS technique explained in Section 3.3.1. Like the ALS technique, efficiencies can be gained. In Section 3.3.1 these efficiencies were gained by differentiating the SSE. They can also be revealed by expanding the expression for the MP inverse:

$$x_k = (a_{i\underline{k}}b_{j\underline{k}}a_{i\underline{\ell}}b_{j\underline{\ell}})^{-1}(a_{i\underline{\ell}}b_{j\underline{\ell}}h_{ij}), \tag{3.73}$$

$$x_k = ((a_{i\underline{k}}a_{i\underline{\ell}})(b_{j\underline{k}}b_{j\underline{\ell}}))^{-1}(a_{i\underline{\ell}}h_{ij}b_{j\underline{\ell}}). \tag{3.74}$$

As mentioned in the previous subsection, Lev-Ari arrives at an identical solution, but does so using EMV algebra operations [72]. This involves using 5 and 4 separate operators and identities respectively. As well, the first-degree data is represented by embedding it within a matrix. Having to remember these extra operations, and their accompanying identities places additional burdens on the researcher. Making matters worse, there seems to be no agreement on symbology. Lev-Ari [72] and Liu and Trenkler [73] use different symbols to represent the same operations in their exposition of the same solution.

When the problem is formulated using NT algebra, the only identity required is the association identity—all other operations flow naturally from the algebra, including rearranging operands. To contrast the two approaches, the solution provided by Lev-Ari [72] is given below. Here, $\otimes$, $\odot$, and $\circ$ represent the Kronecker, Khatri-Rao, and Hadamard matrix products, respectively. The two other symbols used are the vec and vecd operations, which serve as flattening and diagonal selection operations, respectively. The definitions of these operations are given by Liu and Trenkler [73]. Even without knowing the definitions, the following derivation provides insight into the complexity of EMV compared to NT algebra:

$$\mathbf{AXB}^{\mathsf{T}} = \mathbf{H}, \tag{3.75}$$

$$\mathrm{vec}(\mathbf{AXB}^{\mathsf{T}}) = \mathrm{vec}(\mathbf{H}), \tag{3.76}$$

$$(\mathbf{B} \otimes \mathbf{A})\mathrm{vec}(\mathbf{X}) = \mathrm{vec}(\mathbf{H}), \tag{3.77}$$

$$(\mathbf{B} \odot \mathbf{A})\mathrm{vecd}(\mathbf{X}) = \mathrm{vec}(\mathbf{H}), \tag{3.78}$$

$$\mathrm{vecd}(\mathbf{X}) = [(\mathbf{B} \odot \mathbf{A})^{\mathsf{T}}(\mathbf{B} \odot \mathbf{A})]^{-1}(\mathbf{B} \odot \mathbf{A})^{\mathsf{T}}\mathrm{vec}(\mathbf{H}), \tag{3.79}$$

$$\mathrm{vecd}(\mathbf{X}) = [(\mathbf{B}^{\mathsf{T}} \circ \mathbf{B})(\mathbf{A}^{\mathsf{T}} \circ \mathbf{A})]^{-1}(\mathbf{B} \odot \mathbf{A})^{\mathsf{T}}\mathrm{vec}(\mathbf{H}), \tag{3.80}$$

$$\mathrm{vecd}(\mathbf{X}) = [(\mathbf{B}^{\mathsf{T}} \circ \mathbf{B})(\mathbf{A}^{\mathsf{T}} \circ \mathbf{A})]^{-1}\mathrm{vecd}(\mathbf{A}^{\mathsf{T}}\mathbf{HB}). \tag{3.81}$$

This example demonstrates that NT algebra can provide a simpler and more natural formalism even when only second-degree NTs and first-degree parameters come into play. Moreover, the EMV algebra approach is less efficient, as it requires matrix products to be computed, followed by diagonal selection, which involves extra computation that is later discarded. NT algebra can also naturally accommodate high-degree data, a characteristic not shared by EMV algebra.

3.3.3 Separable Nonlinear Least Squares

SNLS problems are a category of problems that describes models where one set of parameters, $\mathbf{x}$, expresses a linear relationship provided another set of parameters, $\mathbf{z}$, is known. More formally, the model can be expressed as:

$$\mathbf{y} = \mathbf{A}(\mathbf{z})\mathbf{x}, \tag{3.82}$$

where $\mathbf{A}(\mathbf{z})$ is typically over-determined. A least-squares formulation can be used to determine the parameters that best fit the model:

$$\min_{\mathbf{x},\mathbf{z}} ||\mathbf{r}(\mathbf{x},\mathbf{z})||^2, \tag{3.83}$$

$$\mathbf{r}(\mathbf{x},\mathbf{z}) = \mathbf{y} - \mathbf{A}(\mathbf{z})\mathbf{x}. \tag{3.84}$$

However, if $\mathbf{z}$ is known, $\mathbf{x}$ can be calculated using $\mathbf{A}(\mathbf{z})^{+}\mathbf{y}$. This fact can be used to reduce the number of parameters to be estimated. First, denote the projection operator onto the column space of $\mathbf{A}(\mathbf{z})$ by $\mathbf{P}(\mathbf{z}) = \mathbf{A}(\mathbf{z})\mathbf{A}(\mathbf{z})^{+}$. The complement of $\mathbf{P}(\mathbf{z})$ is $\mathbf{P}(\mathbf{z})^{\perp} = \mathbf{I} - \mathbf{P}(\mathbf{z})$. Using these formulations, Golub and Pereyra [165] proved in 1973 that the following residual is equivalent to minimising (3.83):

$$\min_{\mathbf{z}} ||\mathbf{r}_2(\mathbf{z})||^2, \tag{3.85}$$

$$\mathbf{r}_2(\mathbf{z}) = \mathbf{P}(\mathbf{z})^{\perp}\mathbf{y}. \tag{3.86}$$

Thus, the problem is reduced to finding only the nonlinear parameters $\mathbf{z}$ and determining the linear parameters $\mathbf{x}$ after the fact. As Golub and Pereyra subsequently note in a retrospective, this observation can lead to faster convergence to solutions and better abilities to avoid local minima [149]. For this reason, the SNLS technique has been applied to legions of applications, including the author's work on image alignment [163] and depth-map estimation [164].

In order to solve for $\mathbf{z}$, many nonlinear optimisation methods require the Jacobian of (3.86), which involves the derivative of the projection operator, which will produce a third-degree NT dependant on $\mathbf{z}$. This is not trivial to formulate, as the projection operator is partly composed of an inverse. Golub and Pereyra demonstrated how to formulate this derivative [165], but used a matrix notation that hid the third-degree data from the reader, relying on the reader to understand exactly where the third-degree data comes into play and on how to multiply it with first and second-degree data. Golub and Pereyra freely acknowledge the context-specific nature of their notation and justify its use based on relative simplicity. However, it is instructive to formulate the problem using NT algebra. This has the added advantage of being easily extensible to high-degree parameters, *e.g.*, those encountered in electronic imaging or other high-degree domains. As Chapter 6 will outline, generalising the SNLS approach to high-degree domains is particularly useful for depth-map estimation, amongst other applications.

To start, one must express the Jacobian of (3.86) using NT algebra. Let a_{ij}, a^{+}_{ij}, z_ℓ, y_i, p_{ij}, and $p^{\perp}_{ij}$ denote the NT equivalents of $\mathbf{A}$, $\mathbf{A}^{+}$, $\mathbf{z}$, $\mathbf{y}$, $\mathbf{P}$, and $\mathbf{P}^{\perp}$, respectively, where dependencies on z_ℓ have been dropped for simplicity. Note that the projection operators are symmetric and idempotent, *i.e.*, $p_{ij} = p_{ji}$ and $p_{ij} = p_{ik}p_{kj}$.

The crux of formulating the derivative lies in a clever reformulation that avoids explicitly taking the derivative of an inverse. First, note the following relationship:

$$\frac{\partial p^{\perp}_{ij}}{\partial z_\ell} = -\frac{\partial p_{ij}}{\partial z_\ell}, \tag{3.87}$$

which takes advantage of the fact that the derivative of the constant-valued Kronecker delta NT, or identity matrix when using the MV paradigm, is zero. Using the idempotence of projection operators and the product rule, the partial derivative of p_{ij} can be expressed as

$$\frac{\partial p_{ij}}{\partial z_\ell} = \frac{\partial p_{ik}p_{kj}}{\partial z_\ell}, \tag{3.88}$$

$$= \frac{\partial p_{ik}}{\partial z_\ell}p_{kj} + p_{ik}\frac{\partial p_{kj}}{\partial z_\ell}. \tag{3.89}$$

On its own this does not seem helpful. But when expressing the Jacobian of a_{ij} a relationship to (3.89) can be derived. Again, using the properties of the projection operator, note that $a_{ij} = p_{ik}a_{kj}$. Thus, the partial derivative of a_{ij} with respect to z_ℓ leads to the following set of expressions:

$$\frac{\partial a_{ij}}{\partial z_\ell} = \frac{\partial p_{ik}a_{kj}}{\partial z_\ell}, \tag{3.90}$$

$$= \frac{\partial p_{ik}}{\partial z_\ell}a_{kj} + p_{ik}\frac{\partial a_{kj}}{\partial z_\ell}. \tag{3.91}$$

Rearranging (3.91) leads to

$$\frac{\partial p_{ik}}{\partial z_\ell}a_{kj} = \frac{\partial a_{ij}}{\partial z_\ell} - p_{ik}\frac{\partial a_{kj}}{\partial z_\ell}, \tag{3.92}$$

$$= p^{\perp}_{ik}\frac{\partial a_{kj}}{\partial z_\ell}. \tag{3.93}$$

The expression in (3.93) can be multiplied on both sides with a^{+}_{jm}, resulting in

$$\frac{\partial p_{ik}}{\partial z_\ell}a_{kj}a^{+}_{jm} = p^{\perp}_{ik}\frac{\partial a_{kj}}{\partial z_\ell}a^{+}_{jm}, \tag{3.94}$$

$$\frac{\partial p_{ik}}{\partial z_\ell}p_{km} = p^{\perp}_{ik}\frac{\partial a_{kj}}{\partial z_\ell}a^{+}_{jm}. \tag{3.95}$$

With the above expression in hand, one can turn to formulating the derivative of p_{ij}.

The first term of (3.89) corresponds to (3.95). The second term of (3.89) can also be formulated, but by observing that the symmetry of p_{ij} and its derivative means that it is simply a permuted version of the first term, where the positions of i and j have been reversed. This leads to a complete expression for the partial derivative of the projection operator:

$$\frac{\partial p_{ij}}{\partial z_\ell} = p^{\perp}_{ik}\frac{\partial a_{km}}{\partial z_\ell}a^{+}_{mj} + p^{\perp}_{jk}\frac{\partial a_{km}}{\partial z_\ell}a^{+}_{mi}. \tag{3.96}$$

With (3.96) in hand, the partial derivative of the reduced residuals can be reformulated, allowing the separable representation to be employed within gradient-based optimisation algorithms. In Golub and Pereyra's exposition, the symmetry is expressed using a transpose operator that the reader must know only applies to the "slices" of the partial derivative of a_{km}. Unlike the context-specific notation used in the seminal paper [165], the derivation of this result here can also be extended to incorporate high-degree operands and entrywise products, which will prove useful for the depth-map estimation problem discussed in Chapter 6.

3.4 Summary

The need to perform arithmetic upon high-degree data is a wide-ranging need, arising in myriad scientific disciplines. However, existing algebras for high-degree data do not satisfy the disparate requirements expressed by this body of work. Viewing a universal formalism as a positive development, we outline an NT algebra, grounded in Einstein notation, whose capabilities imbue it with a unifying capacity for work upon high-degree data.

NT algebra supports commutative and associative inner, entrywise, and outer products across arbitrary indices of N-degree data. With an association identity in hand, NT algebra also provides an associative framework for n-ary inner products, an operation crucial to many applications, but not typically dealt with head-on by previous algebras. Completing the picture, NT algebra also supports linear inversion of NT equations. As outlined by Chapter 2, these capabilities are necessary features for a universal algebra for high-degree data.

These capabilities are rounded out by unary operations, including the entrywise analogue of contraction called attraction. Differentiation, non-linear functions, and vector and matrix NTs are also supported, yielding an algebra with singular expressibility. Three exemplars, drawn from CP tensor decomposition, linear parameter estimation, and the SNLS formulation, showcase the abilities of NT algebra within varied and impactful settings.

NT algebra represents an important step forward toward the prospect of a universal technical computing framework for high-degree data—one that would unite divergent work beyond the MV paradigm. Yet, these algebraic innovations are orphaned without supporting software. NT algebra, and other high-degree algebras, may describe operations we would like to perform, but NT software, and kindred contributions, must aim to expand the scope of what operations we *can* perform. Since computational approaches for high-degree data are still being established, developing software for NT algebra requires its own innovations. The rich topic of NT software is explored in the following two chapters.

Chapter 4

A Dense Foundation

To perform numeric tensor (NT) operations for high-degree applications, software libraries must be available for the formalism of NT algebra. This is the first of two chapters which overview the NT software designed to provide this functionality. As highlighted in Section 2.2.5, this software is composed of two components. The first, called LibNT, is a C++ library that encompasses the high-performance algorithmic kernels needed for NT computations. In addition, LibNT is designed to support NT algebra programmatically in a compiled-language environment. The second component, called NTToolbox is a MATLAB interface to LibNT's fast NT algorithms, offering a flexible interpreted language environment for NT computations. Both libraries are open source and shared online[1]. Reflecting their great translational importance, these libraries are licensed using the highly permissive three-clause BSD license [166]. Moreover, the libraries are documented using Doxygen [167] and are accompanied by extensive unit tests.

We first provide an outline on how LibNT and NTToolbox support and implement NT algebra. Following this we delve into the design choices and implementation details of the dense algorithms for NT computations. Finally we provide benchmarks comparing the performance of LibNT and NTToolbox's dense algorithms to those of other high-degree software packages. Since the high-performance algorithms are implemented using LibNT's C++ code, for simplicity when discussing these kernels we will refer to them as being part of LibNT, with the understanding that NTToolbox offers an interface to these routines.

4.1 NT Software Overview

The following sections provide an overview of LibNT and NTToolbox. Section 4.1.1 begins by discussing the design principles guiding the development of these libraries. Section 4.1.2 then discusses how to constructively represent any NT product. This is followed by Section 4.1.3 which discusses how LibNT and NTToolbox resolve and check NT algebra expressions.

[1]https://github.com/extragoya/LibNT

4.1.1 Design Principles

In its outline of software for high-degree data, Section 2.2 explained that important qualities are comprehensive support for NT algebra, support for sparse data, and programming and computational efficiency. Both LibNT and NTToolbox offer full comprehensive support of NT algebra and sparse NTs. However, completely satisfying programmatic and computational efficiency simultaneously is not always possible. Thus, when faced between the choice of the two, LibNT and NTToolbox take opposite paths, leaning toward computational and programming efficiency, respectively. Yet, when considered together, the two libraries combine to offer a suite of NT software that allows users to prioritise for programmatic or computational efficiency as their needs demand.

The design principles used to implement LibNT and NTToolbox follow from their respective emphases on computational or programmatic efficiency. These are detailed below.

LibNT

When faced with the choice between programmatic efficiency and computational efficiency, LibNT is designed to lean toward the latter consideration. Nonetheless, LibNT is still meant to provide an easy-to-use programmatic environment for NT algebra. Because C++ is a language with enormous breadth, there are many design choices toward implementing a library for NT computations. In LibNT's case, we choose to employ both generic programming (GP) and object-oriented programming (OOP), which are two programming idioms that can sometimes conflict, as they are typically connected with static and dynamic polymorphism, respectively [126]. The former is associated with templates and fast running times while the latter is a much more rigid idiom, where functions are constrained to only operate on a certain set of classes or sub-classes.

For numerical libraries run-time speed is often paramount. As LibNT falls into this category, it cannot afford the virtual look-up times of traditional OOP inheritance and when possible it errs on using the compile-time resolution capabilities of GP. Nonetheless, NTs are accompanied by specific algebraic and computational structures, whose need for operations, inheritance structures, and overload sets make them amenable to certain OOP concepts.

A programming idiom that offers an excellent mix of OOP and GP functionality is the curiously reoccurring template pattern (CRTP) [126]. A more descriptive, yet less-used, name for this idiom is the parametric subclass pattern (PSCP), a term coined by the makers of Boost's Phoenix library [168]. The static inheritance used in the PSCP is often contrasted with the virtual functions seen in OOP, as it offers much (but not all) of the same dispatching capabilities but without the same costs to efficiency. As inheritance is the cornerstone of the PSCP, the idiom provides an explicit structure, which is desirable for implementing NT classes with specific operations. For these reasons, all NT classes within LibNT are designed to follow the PSCP idiom, mirroring the route of other fast and successful numerical libraries [144, 145].

To further decrease running time as much as possible, LibNT resolves the grammar of NT algebra during compile time using template metaprogramming (TMP). In effect TMP can commandeer standard compilers to check NT expressions and perform static dispatches. This allows C++ to act as a domain-specific embedded language (DSEL) for NT algebra. These techniques have helped other numerical libraries with complex grammars match the run-time speeds of hand-written code seen in FORTRAN-style scientific computing routines [101, 126]. Importantly, TMP allows programming efficiency to remain high without sacrificing computational efficiency, allowing users to program directly using NT algebra.

In addition to these advanced programming techniques, LibNT also employs many new features of the C++11 standard. For instance, unlike many other NT libraries, there is no upper limit to NT degree. Implementing a library general enough to handle arbitrary degrees during *compile time*, with the accompanying arbitrary number of indices in the NT algebra grammar, required using variadic templates, which are a feature unique to the C++11 standard. This and other features of the standard has allowed C++ to enjoy a generalisability unique among statically-typed languages, which serves it well for supporting NT algebra.

While LibNT is kept as self-contained as possible, it does rely on other code bases. For instance, its TMP capabilities were implemented using Boost's metaprogramming library [169]. As well, LibNT is heavily reliant on matrix-computation routines in order to implement certain NT operations. For these cases, LibNT uses the Eigen [145] library. To ease barriers to adoption, LibNT follows a header-only principle in its own C++ code base. This also extends to the libraries it depends on, as only the header-only components of both Boost and Eigen are mandatory.

NTToolbox

While C++ is a powerful, portable, and accessible language, it is not always the most convenient choice for prototyping and other scientific applications. Acting as an environment for NT computations within the MATLAB environment, NTToolbox is designed to embody programming efficiency.

NTToolbox uses MATLAB's OOP capabilities to represent NTs and their operations. Unlike LibNT, all aspects of an NT, including its degree, datatype, and its sparsity are resolved dynamically at runtime. To ease programming burdens, all NT expressions are supported using string processing. Correctness is checked at runtime. Thus, as much as possible, the burdens to program using NT algebra are removed. Nonetheless, these features mean dispatching is performed dynamically, which adds to runtime costs.

Even so, these abstraction penalties are often very small compared to the costs of actually performing NT computations. To keep computational efficiency as high as possible, NTToolbox always relies heavily on MEX interfaces to LibNT's C++ algorithmic kernels.

$$a_{ijgh}b_{ghrs} \Leftrightarrow \mathbf{AB}$$

$$\mathbf{A} = a_{ijgh}\mathbf{e}_{ij}\mathbf{e}_{ij}^{\mathsf{T}} \qquad \mathbf{B} = b_{ghrs}\mathbf{e}_{gh}\mathbf{e}_{rs}^{\mathsf{T}}$$

$$\mathbf{A} = \begin{bmatrix} a_{1111} & \cdots & a_{11\mathrm{N}1} & a_{1112} & \cdots & a_{11\mathrm{NN}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{\mathrm{N}111} & \cdots & a_{\mathrm{N1N1}} & a_{\mathrm{N}112} & \cdots & a_{\mathrm{N1NN}} \\ a_{1211} & \cdots & a_{12\mathrm{N}1} & a_{1212} & \cdots & a_{12\mathrm{NN}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{\mathrm{NN}11} & \cdots & a_{\mathrm{NNN}1} & a_{\mathrm{NN}12} & \cdots & a_{\mathrm{NNNN}} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} b_{1111} & \cdots & b_{11\mathrm{N}1} & b_{1112} & \cdots & b_{11\mathrm{NN}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ b_{\mathrm{N}111} & \cdots & b_{\mathrm{N1N1}} & b_{\mathrm{N}112} & \cdots & b_{\mathrm{N1NN}} \\ b_{1211} & \cdots & b_{12\mathrm{N}1} & b_{1212} & \cdots & b_{12\mathrm{NN}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ b_{\mathrm{NN}11} & \cdots & b_{\mathrm{NNN}1} & b_{\mathrm{NN}12} & \cdots & b_{\mathrm{NNNN}} \end{bmatrix}$$

Figure 4.1: Mapping inner and outer products to a matrix product. This figure depicts one possible mapping of the NT product $a_{ijgh}b_{ghrs}$ to the product of two matrices. End users need not know the exact mapping. Here the range is assumed to be N for all indices of a and b. Also depicted are the algebraic mappings to matrix NTs as described in Section 3.2.6.

Thus, even though NTToolbox leans toward programming efficiency, it still offers a highly computationally efficient environment for NT computations.

4.1.2 Lattice Products

With the general design principles elucidated, the computational strategies behind the NT libraries can be explored. For instance, executing or inverting NT products is a core aspect of any library aiming to support NT computations. NT products also epitomise many of the challenges inherent in NT computations, as they can consist of any combination of inner, entrywise, and outer products executed across operands of arbitrary degree. Moreover, to be effective, an NT software should execute NT products with high efficiency. This subsection outlines a constructive platform, called the lattice data structure, that provides a means to quickly execute or invert any NT product.

A binary NT product can be boiled down to a sequence of linear mappings over multiple indices. Entrywise products define multiple linear mappings, which in turn are defined by the makeup of inner and outer products. Matrix products are the typical symbolic representations of linear mappings [170].

Any inner/outer product combination can be mapped to a matrix product [112]. Figure 4.1 depicts one possible mapping of an NT product consisting only of inner and outer products. As the figure demonstrates, the indices of a and b undergoing an inner product are mapped to the columns and rows of the matrices **A** and **B**, respectively. The remaining indices undergoing an outer product are mapped to the rows and columns of **A** and **B**, respectively. Mapping inner and outer products to matrix products is a powerful approach, as it provides a ready means to exploit established and gold-standard algorithms for matrix computations.

An analogous mapping when entrywise products come into play is not as well recognised. In such scenarios, any NT product can be mapped to a sequence of matrix products, which Joseph calls a lattice [123]. Indices undergoing an entrywise product are mapped to

$$\begin{gathered} c_{ik\ell} = a_{ij\underline{\ell}} b_{jk\underline{\ell}} \\ \Updownarrow \\ \mathbf{C}_\ell = \mathbf{A}_{\underline{\ell}} \mathbf{B}_{\underline{\ell}} \\ \Updownarrow \\ \mathcal{C} = \mathcal{A} * \mathcal{B} \end{gathered}$$

$$\mathbf{A}_\ell = a_{ij\ell}\mathbf{e}_i\mathbf{e}_j^\mathsf{T} \quad \mathbf{B}_\ell = b_{jk\ell}\mathbf{e}_j\mathbf{e}_k^\mathsf{T} \qquad\qquad c_{ik\ell} = \mathbf{e}_i^\mathsf{T}\mathbf{C}_\ell\mathbf{e}_k$$

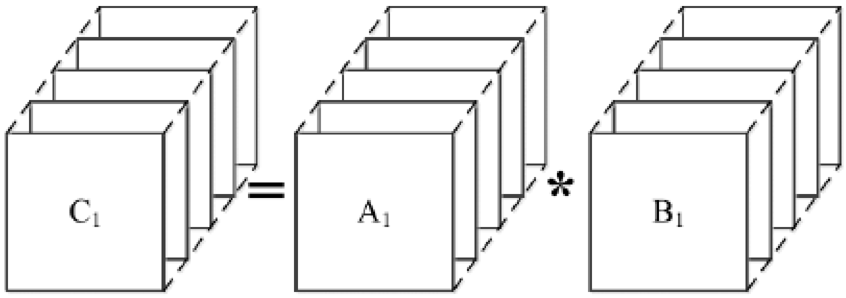


Figure 4.2: Mapping inner, entrywise, and outer products to a lattice product. This figure depicts one possible mapping of the product of two third-degree NTs: $a_{ij\underline{\ell}}b_{jk\underline{\ell}}$. Each set of inner and outer products must also use a mapping, analogous to the one used in Figure 4.1. Also depicted are the mappings to matrix NTs using the convention described in Section 3.2.6.

correspond to the depth or tabs of the lattice. The number of tabs must be the same for each operand and the same multiplication rules used in matrix-vector (MV) algebra apply. Figure 4.2 depicts the mapping of a product of two NTs into a sequence of matrix products.

Figure 4.2 also demonstrates that lattice mappings can be framed as mapping scalar NTs to matrix NTs. Nonetheless, lattices differ from general matrix NTs, in that they do not possess arbitrary NT indices that allow a variety of operations. Instead, they possess rows, columns, and tabs, which follow tightly constrained behaviour. For instance, their third index can only be multiplied entrywise. For this reason, lattices are denoted symbolically using calligraphic script, which helps differentiate them from the more free-wheeling matrix NTs. This also helps emphasise that lattices are entities designed for a purely constructivist role, being implemented in software as a datatype different than NTs. Examples of binary lattice operations are summarised in Table 4.1.

As Table 4.1 illustrates, in addition to providing an equivalent constructivist representation for NT multiplication, lattices play an equivalent role when it comes to solving linear NT equations. Since fast and efficient matrix computations enjoy a long pedigree, mapping any NT product into a lattice product allows the seamless integration of gold-standard MV algorithms. Moreover, tabs can be multiplied independently, providing a parallel framework for any general entrywise product. Thus, the lattice forms a core datatype for both LibNT

Table 4.1: Lattice operations and prerequisites. Dimension prerequisites are specified by the dimension sequences.

Operation					Requirements on each tab of $\mathcal{A}$
$\underset{M\times N\times P}{\mathcal{C}}$	$=$	$\underset{M\times N\times P}{\mathcal{A}}$	$+$	$\underset{M\times N\times P}{\mathcal{B}}$	
$\underset{M\times N\times P}{\mathcal{A}}$	$=$	$\underset{M\times N\times P}{\mathcal{A}}$	$+$	$\underset{M\times N\times P}{\mathcal{B}}$	
$\underset{M\times Q\times P}{\mathcal{C}}$	$=$	$\underset{M\times N\times P}{\mathcal{A}}$	$*$	$\underset{N\times Q\times P}{\mathcal{B}}$	
$\underset{M\times N\times P}{\mathcal{C}}$	$=$	$(\underset{M\times M\times P}{\mathcal{A}})^{-1}$	$*$	$\underset{M\times N\times P}{\mathcal{B}}$	Non-singular
$\underset{Q\times N\times P}{\mathcal{C}}$	$=$	$(\underset{M\times Q\times P}{\mathcal{A}})^{+}$	$*$	$\underset{M\times N\times P}{\mathcal{B}}$	Fully column-ranked

and NTToolbox, acting as cornerstone for NT computations.

For dense NTs, mapping to a lattice is accomplished by permuting data to keep memory accesses across inner products contiguous. Once done, fast dense matrix routines can execute or invert the product. Both libraries use the Eigen library's [145] routines for this purpose. Performing the same operation efficiently in the sparse case is more complex, which is a topic explored in Chapter 5.

4.1.3 Supporting NT Algebra

To seamlessly support NT algebra, NT software should emulate as close as possible the written notation. The capabilities of LibNT and NTToolbox in supporting NT algebra can be best showcased using an exemplar. With an exemplar presented, the mechanics by which the libraries support the unary, binary, and assignment operations of NT algebra can be explained.

The alternating least squares (ALS) algorithm, featured in Section 3.3.1 for canonical-polyadic (CP) tensor decomposition, provides an excellent exemplar to demonstrate LibNT and NTToolbox's capabilities. For instance, (3.69) combines entrywise products and pseudo-inverses together. The expression in question is repeated here:

$$u_{i'm} = ((v_{k\underline{j'}} v_{k\underline{m}})(w_{\ell\underline{j'}} w_{\ell\underline{m}}))^{+}(v_{k\underline{j'}} a_{i'k\ell} w_{\ell\underline{j'}}). \tag{4.1}$$

Focusing on LibNT for now and supposing the dimensions of $a_{ik\ell}$ are $n_1 \times n_2 \times n_3$ and the number of factors in the CP decomposition r, Figure 4.3 depicts how (3.69) would be expressed.

In a practical application the code snippet of Figure 4.3 would be a component of a larger routine. Even so, the example illustrates several important facets. For one, since LibNT uses TMP to check the validity of NT expressions, the matchings between NT indices are examined at compile-time. Since TMP can only differentiate between types, and not values, each declared index must be a different type. The `NTINDEX` macro fulfills this purpose.

In terms of algebraic expressions, LibNT faithfully represents NT algebra, with small but important variations. First, exclamation points, instead of underlines, designate entrywise

```
DenseNT<double,3> a(n1,n2,n3); //could also be sparse
DenseNT<double,2> y, u, v(n2,r), w(n3,r);
NTINDEX i; NTINDEX j;
NTINDEX k; NTINDEX l;
//initialise values of a, v, w
y(j,m)=(v(k,!j)*v(k,!m))*(w(l,!j)*w(l,!m));
u(i,m)=y(j,m)|~(w(k,!j)*a(i,k,l)*v(l,!j));
```

Figure 4.3: LibNT code for executing CP tensor decomposition operations. The code snippet provides a programmatic implementation of (3.69). Syntax errors are detected at compilation.

```
DenseNT a(n1,n2,n3); //could also be sparse
DenseNT a(n1, n2, n3), v(n2,r), w(n3,r), y, u;
//initialise values of a, v, w
y('j,m')=(v('k,!j')*v('k,!m'))*(w('l,!j')*w('l,!m'));
u('i,m')=y('j,m')\~(w('k,!j')*a('i,k,l')*v('l,!j'));
```

Figure 4.4: NTToolbox code for executing CP tensor decomposition operations. The code snippet provides a programmatic implementation of (3.69). Syntax errors are detected at runtime.

products. Second, similar to MATLAB's backslash operator, the | operator, which is the closest C++-supported operator to the former, is used to perform an inversion or pseudo-inversion depending on the dimensionality and determinedness of the linear mapping being inverted.

The final important alteration concerns n-ary inner products. Unlike NT algebra, LibNT only supports binary operations. Thus, to execute an n-ary inner product, the association identity should be used to break it up into binary operations. However, as discussed in Section 3.2.1, NT algebra uses an implicit convention whereby surrounding an *executed* entrywise product with parentheses removes the corresponding underlines, enabling the expression of n-ary inner products. Since the C++ language does not provide a means to detect when expressions are surrounded by parentheses, an explicit convention is used instead, whereby the ˜ operator removes any underlines from any *executed* entrywise products within a parenthesis. The last line of Figure 4.3 depicts this operation. The expression resolution capabilities of LibNT are intelligent enough to also handle situations where an index may be underlined but not used in an entrywise product, such as:

$$\mathtt{x}(\mathtt{j},\mathtt{k}) = \tilde{}(\mathtt{a}(!\mathtt{i},!!\mathtt{j},!\mathtt{k}) * \mathtt{b}(!\mathtt{i},!!\mathtt{j})) * \mathtt{c}(\mathtt{i},!\mathtt{j},!\mathtt{k}),$$

where the exclamation point for k is not removed within the parenthesis because it was not involved in the entrywise product executed therein.

As Figure 4.4 demonstrates, NTToolbox provides similar functionality, except in the MATLAB interpreted environment. As demonstrated, the code is very similar to LibNT's version, except that string processing is used to resolve NT expressions. Using string processing means that unlike LibNT, no declaration of NT indices is needed within NTToolbox `a(!i,j,j,k,i)`

id predicate

	!i	j	j	k	i
!i	·	✗	✗	✗	✓
j	·	·	✓	✗	✗
j	·	·	·	✗	✗
k	·	·	·	·	✗
i	·	·	·	·	·

(a)

full predicate

	!i	j	j	k	i
!i	·	✗	✗	✗	✗
j	·	·	✓	✗	✗
j	·	·	·	✗	✗
k	·	·	·	·	✗
i	·	·	·	·	·

(b)

`a(!i,j,k)*x(j,l,i)`

id predicate

	!i	j	k
j	✗	✓	✗
l	✗	✗	✗
i	✓	✗	✗

(c)

full predicate

	!i	j	k
j	✗	✓	✗
l	✗	✗	✗
i	✗	✗	✗

(d)

Figure 4.5: Example auto-sequence and cross-sequence checks of NT expressions. (a) and (b) depict an auto-sequence check while (c) an (d) depict a cross-sequence check. Example originating NT expressions for each is also illustrated. The left column depicts the checks using the id predicate while the right uses the full predicate. A ✓ denotes a match, while an ✗ denotes no match.

. In addition, the \ operator is used to perform the solution of linear equations.

While important to consider, the slight alterations to NT algebra necessary for LibNT and NTToolbox's functionality do not seriously alter their ability to replicate the formalism. Expression resolution for both libraries is powered by an identical set of rules. How these rules are implemented do differ. In LibNT's case, TMP resolves expressions. As TMP is a stateless and functional language, recursion is fundamental to LibNT's grammar checking. On the other hand, NTToolbox uses the comparatively simpler approach of string processing, which can be performed procedurally at runtime. These distinctions will not be dwelled upon here. Regardless of any differences in implementation, the two libraries rely on the same set of rules for resolving NT expressions. These are detailed below. Any code snippets will be drawn from LibNT.

Unary Operations

Whenever an NT datatype is subscripted by NT indices, the first step is to always check and perform any expressed unary operations. Apart from resolving any unary operations, this also acts as the first level of checking NT expression legality for any subsequent binary operations. An *auto-sequence* check, where the indices of an operand are matched against each other, powers the grammar checking rules of unary operations.

Figure 4.5(a) and (b) demonstrate an auto-sequence check using two different types of predicates. The first, called the *id predicate*, only examines the index letter when determining a match, ignoring any exclamation marks. The second, called the *full predicate*, considers both the index letter and number of exclamations when declaring a match.

These two types of auto-sequence checks are performed to first confirm that if two indices share the same letter, they also also share the same number of exclamation marks. If this rule is violated, the code will not compile in LibNT's case and C++11's `std::assert` is used to provide a meaningful compilation error message. In NTToolbox's case, a runtime error will trigger if the code is run.

After the expression legality is confirmed, a second check observes if a contraction and/or attraction is expressed. If so, any corresponding unary expressions are executed. For instance the following expression,

`a(!i,j,j,k,!i)`,

will execute a contraction and attraction across the j and i indices, respectively. This will produce a second-degree NT indexed by i and k. Since a new index sequence has been calculated, the expression can be chained together with other expressions to create a larger NT equation, *e.g.*,

$$\mathtt{a}(!\mathtt{i},\mathtt{j},\mathtt{j},\mathtt{k},!\mathtt{i}) * \mathtt{b}(\mathtt{i},\mathtt{l}),$$

Binary Operations

Binary operations between two NTs can involve any manner of matching between operand indices. A *cross-sequence check* between index sequences of two operands is used to resolve binary operations. Figure 4.5(c) and (d) depict an example cross-sequence check using the id and full predicates, respectively.

Binary expressions fall into two major categories—simpler entrywise operations, such as addition and subtraction, and more involved operations, such as multiplication and solution of equations. The former category can be denoted as abelian operations, which is a term drawn from abstract algebra that refers to operations, such as addition and subtraction, that are commutative but not distributive.

For these types of operations, the expression legality is checked by ensuring that every index is matched with another index. In addition, each index must have the same number of exclamation points. Legality is verified using a cross-sequence check with the full predicate. Some legal and illegal LibNT expressions are given below.

```
a(i,j,k) + b(j,k,i);  \\will compile,
a(i,j,k) + b(j,!k,i);  \\will not compile,
a(i,j,!k) + b(j,k,i);  \\will not compile,
a(!i,j,k) + b(!i,k,l);  \\will not compile,
a(i,j,k,l) + b(j,k,i);  \\will not compile.
```

Instances that do not compile under LibNT will report a runtime error under NTToolbox. Since abelian operations can also express a re-ordering of index sequences between the two operands, expression resolution also computes the re-ordering of the right-operand indices with respect to the left operand's.

The rules for multiplication and the solution of equations differ from abelian operations. In these cases, expression checking must treat entrywise product indices different than indices without exclamation marks. A cross-sequence check uses the full predicate to verify that the binary expression follows two rules: a standard index can have none or one match whereas an entrywise index must match up once. Standard indices that have a match correspond to inner-product indices, otherwise they are outer-product indices.

Provided the expression passed the check, the re-ordering of indices used in inner, outer, and entrywise products is calculated for each operand. These are then used to flatten each NT into a lattice. After execution of the multiplication or solution of equations, the resulting lattice is expanded back into an NT and returned along with a new index sequence. For instance, the following expression:

$$\mathtt{a(i,!j,k) * b(k,l,!j)},$$

will return an NT parameterised by an index sequence of `i,l,!j`.

Assignment

Assignment in NT expressions may involve a re-ordering of indices. Take for instance the following expression:

$$\mathtt{a(i,j,k) = b(j,k,i);},$$

which involves a re-ordering of all indices. Thus, in executing assignment, the matching between index sequences on each side of the equal sign must be examined.

Similar to abelian operations, a cross-sequence check examines expression legality by ensuring that every index of the left operand is matched with an index of the right operand. However, in assignment's case only the id predicate is used, meaning the occurrence of exclamation marks is ignored in matching indices. Some invalid and valid assignment expressions are listed below:

```
a(i,j,k) = b(j,k,i); \\will compile,
a(i,j,k) = b(j,!k,i);\\will compile,
a(i,j,!k) = b(j,k,i);\\will compile,
a(!i,j,k) = b(!i,k,l);\\will not compile,
a(i,j,k,l) = b(j,k,i);\\will not compile.
```

Like abelian operations, a key component of the matching consists of calculating the ordering of the right-operand indices when matched to the left-operand indices.

4.2 Dense Algorithms

Dense NTs are represented using a contiguous array of data elements, which also allows random-access. A fixed lexicographical order was chosen where data elements are arrayed with the most significant index as the last index, followed by the second-last index and so on. For a second-degree NT this lexicographical order is equivalent to column-major order in matrices. The speed of multiple random accesses is dependant on the contiguity of the data elements being accessed, which is largely dependant on which indices vary between consecutive data accesses. Lattice multiplication and the solution of linear NT equations are performed using Eigen's routines [145].

This section details some additional considerations that help speed up the execution of dense NT computations. Reflecting LibNT's focus on speed, these improvements have been implemented therein. Several, but not all, of the same enhancements could be extended to NTToolbox, but this is an aspect of future work.

First, Section 4.2.1 explains how LibNT performs abelian operations without creating extra temporaries. This is followed by Section 4.2.2, which outlines how LibNT permutes dense data based on an index rearrangement or shuffle. Section 4.2.3 discusses how index re-ordering calculations can be avoided. Finally, Section 4.2.4 briefly outlines instances where an NT product need not be executed using a lattice product.

4.2.1 Abelian Operations

C++ is an eager programming language, which means that expressions are evaluated immediately. As a result, n-ary expressions composed of many operands may produce a large number of temporary objects. For instance, in an eager language the following expression,

$$\mathtt{y}(\mathtt{i}) = \mathtt{a}(\mathtt{i}) + \mathtt{b}(\mathtt{i}) + \mathtt{c}(\mathtt{i}), \tag{4.2}$$

produces a first temporary, the result of `a(i)+b(i)`, which is then added to `c(i)`, creating another temporary. This results in two separate loop traversals over the index range of `i`, in addition to any added memory consumed by the temporaries.

If one was coding (4.2) by hand, the most efficient implementation would be to loop through all NTs simultaneously, and perform a ternary addition across each of the three operands, assigning the result to `y(i)`. This not only saves on memory costs, but it also means that only one traversal through the elements is needed rather than the two traversals needed in the eager scheme. This latter benefit also enjoys the added advantage of reducing cache misses should the NTs in question be small enough.

Expression templates provide a means to realise the benefits of hand-coding expressions like (4.2). By combining static dispatch with lazy evaluation, expression templates often produce code that is comparable to or even faster than hand-coded solutions [101,127,142]. While expression templates mean one can avoid hand-coding every possible n-ary operation, using them still requires coding every possible *binary* expression. When using indicial notation, *e.g.*, Einstein or NT algebra, this means coding every possible index matching for every type of arithmetic operation. For N-degree operands, there are $N!$ possible index matchings just for abelian operations alone, a quickly unworkable number. This perhaps explains why FTensor and LTensor, two expression-template libraries for high-degree data, only support general NTs up to degree two [101] and four [127][2], respectively. Blitz++, which supports high-degree Einstein notation, evaluates expressions eagerly.

While it is possible to use C++11's incredibly flexible `std::function` object to perform lazy evaluation, this would be accompanied by virtual function calls, which are to be avoided for repeated calls in numerical libraries [142]. For this reason, LibNT does not evaluate abelian, or any other operations, lazily. This means LibNT would perform two loop traversals to evaluate (4.2). Nonetheless, even though it eschews lazy evaluation, LibNT still manages to avoid creating unnecessary temporaries. For instance in evaluating (4.2), LibNT first creates a temporary NT that holds the result of `a(i)+b(i)`. Afterwards, when `c(i)` is added to the temporary NT, LibNT will recognise its temporary nature and change the `+` operation to a `+=` one. Upon assignment to `y(i)` `std::move` semantics are employed, avoiding a needless copy. Thus, LibNT avoids creating any extra temporary memory locations when performing abelian operations.

This process is invisible to the user and is dispatched statically during compilation, meaning this overhead does not impact runtimes. In terms of execution time, avoiding additional memory provides significant boosts in runtime speed, especially when NTs are of large dimensionality.

4.2.2 Dense Permutations

Permutation of data elements based on an index shuffle represents a key operation in LibNT that underlies many dense arithmetic operations. For N-degree data, these permutations can be considered generalisations of the familiar transposition operation in the MV framework. One operation that frequently requires permutations is assignment. For instance, the following expression illustrates a third-degree assignment requiring a permutation,

$$b_{jik} = a_{ijk}. \tag{4.3}$$

Since the contents of a_{ijk} are being written to a new NT, (4.3) is best executed using an out-of-place permutation, provided a_{ijk} is not a temporary object.

On the other hand, if a_{ijk} is designated as temporary, *e.g.*, an rvalue reference in C++11, then an in-place permutation could also be a good option. This can occur frequently, as the execution of NT arithmetic expressions, especially n-ary ones with many operands, will produce temporary objects that often must be subsequently permutated before being assigned or used in subsequent operations.

[2]Although we were unable to compile certain third or fourth-degree expressions with the LTensor code base.

In the MV paradigm, permutations are limited to the familiar two-index transposition. Even the best in-place transposition algorithms cannot match the out-of-place versions in running time [171], relegating in-place transposition to only those situations where memory use is absolutely paramount. However, when working within the N-degree context of the NT framework, pinning down the merits of in-place vs. out-of-place permutations is not so black-and-white, as running times are affected by the NT degree, absolute sizes of index ranges, relative sizes of index ranges, and the specific permutation in question.

Very limited work on in-place high-degree permutations can be found in the literature. For instance, Ding [172] discussed an in-place permutation algorithm for NTs, but the posted results only considered one particular permutation of third-degree NTs. For this reason, the relative speeds between the two options remains very much unclear and more experiments are needed to gain a better picture of the merits of in-place vs. out-of-place dense permutations.

To help fill in this picture, an in-place permutation algorithm was implemented in LibNT. The routine is based on the vacancy-tracking cycles algorithm of Ding [172] along with the improvement suggested in Sec. III A of Jie *et al.*'s work [173]. In the simplest version, a bit array is used to keep track of which data locations have been touched or not. Various authors have published means to avoid the storage of this bit array [171, 172, 174], but when implemented none of these were able to match the running times when the bit array is included. For this reason, LibNT takes the bit-array approach, along with its added memory consumption.

Experiments performed permutations on third, fourth, and fifth-degree NTs. The running times of in-place vs. out-of-place permutations for different dimensionalities are plotted in Figure 4.6. For the specific dimensionalities, degrees, and permutations in question, the results indicate that in-place permutation can post running times competitive with its out-of-place version. Thus, LibNT opts for in-place permutations of temporary dense NTs of third-degree or higher, because runtime compares favourably with the out-of-place versions and there is no extra memory consumption.

These results only shed light on a sliver on the different permutation possibilities LibNT may encounter, and more work is needed to characterise the relative merits of each permutation type. This should include approaches found within computational chemistry, where tuned [93] and multi-threaded [95] approaches to dense permutations have been shown to yield significant performance gains. Similar strategies should be explored for the NT software.

4.2.3 Avoiding Index Calculations

LibNT can support any manner of matching between indices for any of the supported arithmetic operations or assignments. For instance, in the expression,

$$\mathtt{c}(\mathtt{i},\mathtt{j},\mathtt{k}) = \mathtt{a}(\mathtt{k},\mathtt{i},\mathtt{j}) + \mathtt{b}(\mathtt{j},\mathtt{k},\mathtt{i}),$$

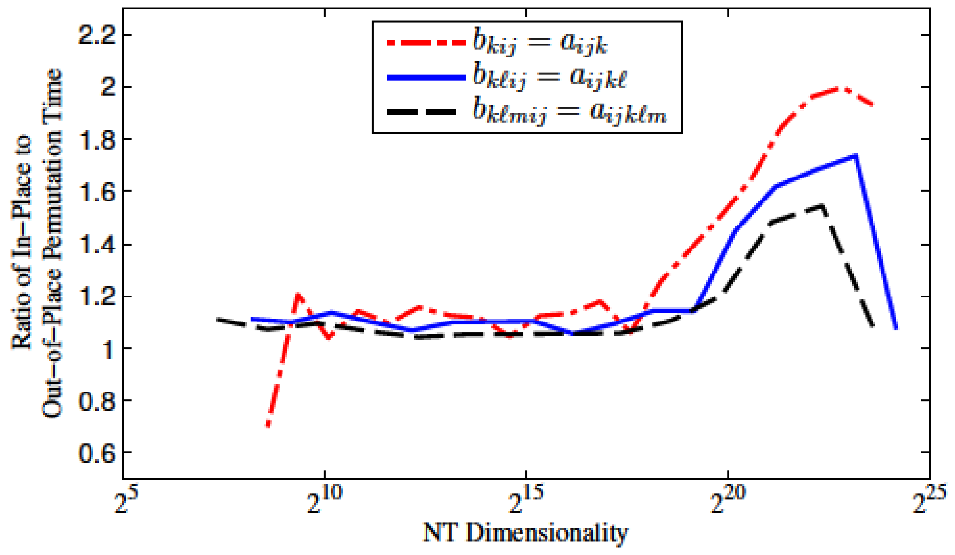


Figure 4.6: Comparison of out-of-place vs. in-place permutation times. The ratio of in-place to out-of-place permutations of the same NT is plotted for increasing dimensionalities, where the executed permutation is indicated in the legend. 10 trials were performed at different dimensionalities with trend lines indicating median values. Dimensionalities were $dim = n \times 2n \times 3n$, $dim = 3n \times 2n \times 3n \times n$, and $dim = 2n \times n \times 2n \times 3n \times n$ for the third, fourth, and fifth-degree NT results, respectively. The x-axis plots the *dim* values.

each element of `b(j,k,i)` must be added to a matching element in `a(k,i,j)` whose location is based on how operand indices match up. A similar process is involved with the resulting assignment. This necessary matching involves calculating equivalent index locations, which may consume significant computational cycles relative to the arithmetic or assignment operation in question.

Nevertheless, some NT calculations do not express an index shuffle, such as the following:

$$\mathtt{c(i,j,k)} = \mathtt{a(i,j,k)} + \mathtt{b(i,j,k)}.$$

LibNT is intelligent enough to recognise such situations at *compile* time and will statically dispatch to specialised and simpler operations when needed. For instance, if the NTs are dense the addition operation will simply add the elements of `b` to `a` without any index calculations. Dispatching at compile-time can provide significant boosts for low-dimensionality dense-NT calculations where the impact of any runtime overhead is disproportionately large.

Similarly, when mapping NTs to lattices, LibNT will statically detect cases where data permutations can be avoided. In these cases LibNT will wrap the existing NT data with a lattice data structure and use it directly within a lattice product or inversion.

4.2.4 Special Products

Computational and/or memory costs accompany the process of mapping NTs to lattices. For dense NTs, the mapping can involve a significant amount of data permutation. The associated costs are usually justified as the rearrangement of data elements ensures high memory locality during the execution of an inner product.

However, when only outer or entrywise products are involved the costs of the lattice mapping become harder to justify. For instance, for dense NTs a pure outer product can simply be computed by multiplying every element in one NT with every other element in the second NT. Thus, if only outer products are involved NT indices need not be matched together and there is no need to place data elements in some specific order. In fact, while not addressed here, the same conclusion applies for sparse NTs, except only the non-zeros of the two NTs need be multiplied.

When entrywise products come into play, with or without outer products, dense NTs can be multiplied by simply using the random-access ability of contiguous data to multiply NT elements with the same entrywise product indices. Thus, when an NT product expresses no inner products, there is no need to map dense NTs to lattices. LibNT detects these instances at compile time and statically dispatches to a multiplication routine specialised to only handle entrywise and outer products.

4.3 Dense Performance

To gauge the effectiveness of LibNT and NTToolbox, we measure their performance against other leading high-degree libraries. We focus on publicly accessible implementations that have a focus on arithmetic calculations with high-degree data [101, 112, 124, 125, 127, 175]. We also do not compare against computational chemistry packages designed for massively parallel systems, *e.g.*, the Tensor Contraction Engine (TCE) [91, 92, 93] and the Cyclops Tensor Framework (CTF) [95], or many-core shared-memory solutions, *e.g.*, Libtensor [94].

It should be noted that all these libraries, benchmarked or not, offer rich functionality outside that of pure high-degree arithmetic, much of which is trailblazing. The fact that these libraries focus on different domains demonstrates the widespread utility of high-performance NT arithmetic. As such we view LibNT and NTToolbox as complementary to these efforts.

The topic of small-dimensionality computations is explored first, a focus of NT computational work that places heavy emphasis on reducing runtime abstraction penalties [101, 125, 127]. This is followed by more generalised benchmarks incorporating entrywise products needed in CP tensor decomposition. Since the data involved can range from small- to large-dimensionality NTs, the latter benchmarks measure performance across a wide spectrum of conditions. Details on the test platform and compiler can be found in Appendix B.

4.3.1 Small-Dimensionality Benchmarks

Fast small-dimensionality NT computations are important for general relativity [101], computer-graphics [70], and computational mechanics [127]. In small-dimensionality settings, reducing abstraction penalties is paramount, as their relative impact on performance can be high.

It is for these reasons that the developers of libraries like Blitz++ [125], FTensor [101], and LTensor [127] have expended considerable effort toward using C++ TMP to support Einstein notation with little to no runtime abstraction penalties. Since LibNT also aims to support its NT algebra with minimal runtime abstraction penalties, comparison to leading small-dimensionality libraries are illuminating. Once the capabilities of these libraries are explained, two benchmarks measure their comparative performance.

FTensor is a powerful library primarily focused on physics computations and has been lauded for its ability to match hand-crafted code speeds while still providing a flexible Einstein-notation interface [126]. To achieve these impressive running times, FTensor evaluates every operation lazily. Uniquely FTensor requires users to specify dimensionality at compile time. This latter aspect allows FTensor to take advantage of loop unrolling optimisations. Unfortunately, this also means that one can quickly reach the template instantiation depth of the compiler, restricting FTensor's use to only small dimensionality NTs. As well, FTensor uses expression templates for each different matching of indices that it supports. Since the number of possible index matchings increases factorially with degree, only general NTs up to second degree are supported. These limitations are not actually negatives for the physics applications FTensor was designed for, but they do restrict its applicability to other types of problems.

Providing a more general interface, LTensor relaxes some of the restrictions of FTensor, by providing dynamically sized NTs in addition to statically-sized versions. Like FTensor, LTensor evaluates expressions lazily and uses expression templates to account for every possible index matching. As the authors note, this required considerable amount of "tedious work" [127]. Despite the authors' claims to support up to fourth-degree NTs, during testing we were unable to successfully compile expressions that included NTs greater than second degree.

Acting as the most general of the three libraries, Blitz++ is considered an exemplar in efficient dense array calculations, trail-blazing many of the TMP and expression template techniques that now see prevalent use [126]. Blitz++ supports Einstein notation, except that it does not allow indices on the left-hand side of expressions and requires a free function call to perform inner products. Unlike FTensor and LTensor, Blitz++ supports entrywise products. As well, Blitz++ evaluates all Einstein notation expressions eagerly, avoiding the limitations on NT degree seen in FTensor and LTensor.

All three of the detailed libraries implement multiplication by accessing NT elements without permuting the data. This is in contrast to LibNT, which physically permutes data into lattice or matrix form. Since extremely fast small-dimensionality computations are an important driver for high-degree computations, it is important to measure LibNT's performance compared to these libraries. The primary overlapping functionality of the cited libraries with LibNT are addition/subtraction and inner/outer products. For this reason, benchmarks only consisted of those operations. To isolate differences in operation

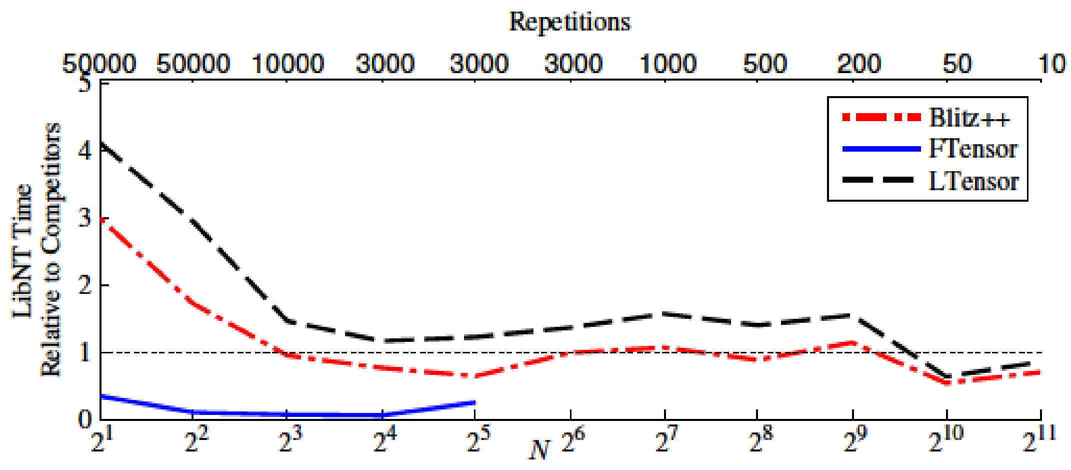


Figure 4.7: Benchmarks for small-dimensionality addition and subtraction. Tests executed the expression in (4.4). All dimensionalities were always $N \times N$. The ratio of running times of LibNT to each library is plotted, where the bottom axis denotes N and the top x-axis displays the number of trials for each N.

performance, each of these operation categories were tested independently, minimising the effect of confounding factors. Since FTensor only supports general NTs up to second degree and we were also unable to compile third or fourth-degree LTensor code, benchmarks only consist of operations on second-degree data. As well, since Blitz++ does not support Einstein notation on the left-hand side of equations, operations were chosen that did *not* involve a permutation during assignment.

The first benchmark focuses on addition/subtraction of second-degree NTs. Specifically, given three operands, x_{ij}, y_{ij}, and n_{ij}, the following expression is calculated:

$$n_{ij} = y_{ij} - x_{ij} - y_{ij} - x_{ji} + y_{ji} - x_{ij} - y_{ji} - x_{ji}, \tag{4.4}$$

where (4.4) includes all possible matching of indices between y_{ij} and x_{ij}, highlighting performance when data must be accessed in a non-contiguous manner. This expression plays to many of the strengths of FTensor and LTensor, as these libraries use manually programmed expression templates for each of the different index matchings, avoiding any runtime calculations of permuted index offsets. The n-ary expression of (4.4) is also highly amenable to the lazy evaluation scheme of the two libraries, which can calculate the entire expression using only one loop traversal and no temporaries.

Tests executed (4.4) $N \times N$ dense NTs with increasing values of N. Since running times for a single execution of (4.4) at the lowest dimensionalities can be smaller than the resolution of C++'s timers, the running time for repeated executions was measured. Figure 4.7 illustrates the running times of the three libraries compared to LibNT for values of N ranging from 2 to 2048. Unsurprisingly, LTensor and Blitz++ highly outperform LibNT at lower dimensionalities, as the index matching calculations that LibNT executes are relatively more expensive at such low dimensionalities. Surprisingly, FTensor did not perform well at any dimensionality. This may stem from the test platform and compiler we used, which may not be favourable to FTensor's complete static unrolling of loops. However, without a thorough investigation of FTensor's complete code base and assembly output this speculation cannot be confirmed. Since FTensor uses statically defined dimensionalities, the template instantiation depth limit was reached fairly early on at $N = 32$.

At higher dimensionalities the computational costs begin to be dominated by the actual arithmetic operations, mitigating the cost of LibNT's calculation of permuted index offsets. Somewhat surprisingly, despite its eager evaluation scheme Blitz++ is able to almost match LTensor's performance throughout the entire benchmark. Blitz++'s performance becomes less surprising when considering the considerable efforts and expertise the developers applied in optimising the library for different platforms, including issues related to memory alignment and vectorised code. This is also reflected in the need to configure and build Blitz++, optimising the library for different platforms. This is in contrast to the header-only principle of LibNT, LTensor, and FTensor.

In terms of LibNT's performance, the most important conclusions to draw are that despite using a benchmark highly favouring the other libraries, LibNT can still perform competitively. In fact, at dimensionalities of 8×8, LibNT's performance begins to become competitive with that of Blitz++ and LTensor. Eventually, LibNT matches or exceeds the other two libraries in runtime performance. LibNT realises this performance, despite offering a more general-purpose environment for calculations with high-degree data with a greater set of operations and better support for large-scale operations.

The second benchmark focuses on inner and outer products by evaluating the following expressions with dense NTs:

$$n_{ik} = y_{ij} x_{jk}, \tag{4.5}$$

$$n_{ik} = y_{ij} x_{kj}, \tag{4.6}$$

$$n_{ik} = y_{ji} x_{jk}, \tag{4.7}$$

$$n_{ik} = y_{ji} x_{kj}, \tag{4.8}$$

where the different index placements test the performance of the libraries in performing inner products in concert with index permutations. Like the previous benchmark, NT dimensionalities were $N \times N$, and performance tested speed for increasing values of N. Figure 4.8 depicts the results of this benchmark. As the figure demonstrates, all three competitor libraries outperform LibNT at small NT dimensionalities, with the exception of Blitz++ at the very smallest dimensionality. As expected, FTensor and LTensor performed best with small NTs, reflecting the advantage of avoiding permutations and simply accessing data non-contiguously at low dimensionalities. However, at higher dimensionalities these libraries are quickly outmatched by LibNT. This performance dropoff is likely the result of greater numbers of cache misses from non-contiguous data access during inner products. As Blitz++ also accesses data non-contiguously its performance closely matches that of

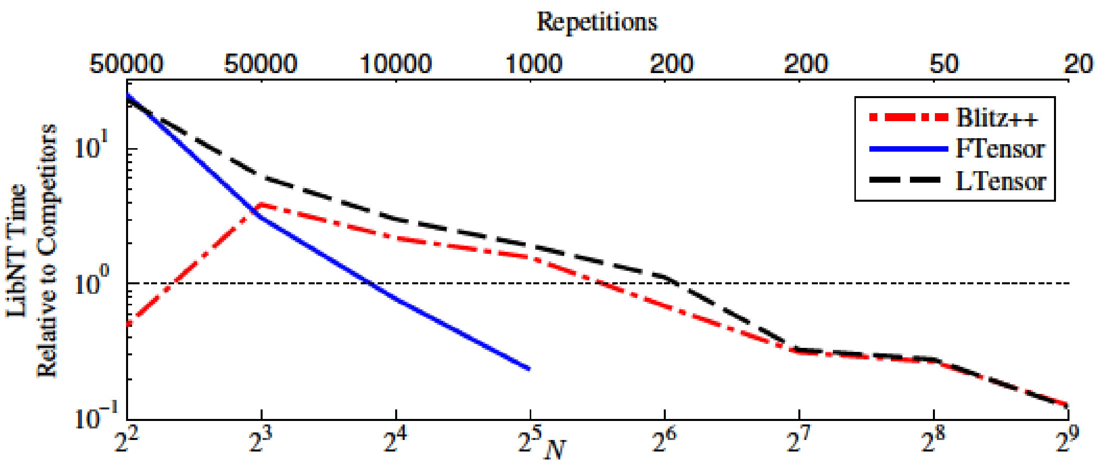


Figure 4.8: Benchmarks for small-dimensionality inner and outer products. Tests executed the expressions from (4.5) to (4.8). Dimensionality of all NTs is always $N \times N$. The ratio of running times of LibNT to each library is plotted, where the bottom axis denotes N and the top x-axis displays the number of trials for each N.

LTensor, except at low dimensionalities where Blitz++'s calculations of index permutation offsets takes a greater toll on execution speed.

In general, permuting data beforehand helps increase locality for the repeated element accesses involved in an inner product, but as seen in Figure 4.8 at low dimensionalities the cost of permutation will outweigh any benefits. Hence, for the very lowest of dimensionalities, accessing data non-contiguously is best. Nonetheless, by the time dimensionalities have reached 32×32 to 64×64, the benefits of permutation begin to bear fruit. At the highest dimensionalities of 512×512 LibNT outperforms both LTensor and Blitz++ by almost a factor of 8. These results indicate that dimensionalities need only reach 32×32 before LibNT becomes competitive, and very soon afterwards LibNT far outpaces the other libraries. Consequently, dimensionalities need not reach high magnitudes before LibNT can begin to compete with or outperform these leading libraries.

4.3.2 Generalised Benchmarks

While blazing fast small-dimensionality computations remain important, in actuality practical calculations with high-degree data can run the gamut from small- to large-dimensionalities. For many of these cases, in particular large-dimensionality computations, runtime abstraction penalties can be less of a concern. Moreover, many applications, *e.g.*, tensor decomposition, require operations outside of the second-degree inner and outer products of FTensor and LTensor. Thus, it is important to characterise performance of more generalised operations that veer into the domain of large-scale computations and entrywise products. We begin by outlining three leadings libraries designed to support generalised NT operations and contrast them with LibNT and NTToolbox. Afterwards, we detail a benchmark, drawn from CP decomposition, that measures performance in executing generalised NT operations.

In terms of libraries that can support generalised large-scale operations, Blitz++, profiled in the previous subsection for small-dimensionality computations, is a prominent example [125]. As noted, Blitz++'s Einstein-notation functionality supports dense inner and entrywise products, with the latter executed by default. Free-function calls are required to perform inner products. The library does not support assignments or inversions using Einstein notation.

Moving to interpreted solutions, NumPy [175] offers similar Einstein notation functionality as Blitz++, but for the Python environment. NumPy offers a free-function Einstein-notation interface that supports entrywise products using the assignment convention discussed in Section 3.2.1. It also provides full support for additions and subtractions, but only limited support for linear inversion. As well, NumPy strives to avoid allocation of temporary memory, implementing n-ary expressions using one large computation. However, this can produce its own slowdowns, and in our experience NumPy code runs faster when code is manually broken line-by-line into binary expressions, which can often be inconvenient and verbose.

Unlike LibNT and NTToolbox, NumPy and Blitz++ do not map NT products to lattice products. Instead they access data without permutes. As outlined in Section 4.3.1, this strategy can be fast for small-dimensionality NTs. However, by foregoing the use of gold-standard matrix-multiplication routines and not ensuring data contiguity, performance with large-dimensionality NTs may suffer as a result.

The MATLAB Tensor Toolbox (MTT) [112, 124] is another interpreted solution, designed to support n-mode^{+} notation in the MATLAB environment. As such, it does offer routines for general inner and outer products across NT indices. However, these come in the form of binary free-function calls, meaning multi-operand expressions must be realised using nested function calls or must be broken into separate code lines, which can be verbose and difficult to parse. Nonetheless, the MTT maps its NT products to matrix products, leveraging MATLAB's LAPACK routines and resulting in fast execution times for inner and outer products. However, the MTT only provides limited support for entrywise products, in the form of pure entrywise products and Khatri-Rao products of flattened NTs. The provided Khatri-Rao implementations are fast, but rely on specialised implementations that do not generalise. As well, the MTT does not support linear inversion and general NT addition and subtraction.

Thus, in general NumPy and, to a lesser extent, Blitz++ provide a greater and more flexible set of NT algebraic operations than the MTT, but without exploiting gold-standard matrix computation algorithms. The MTT, on the other hand, relies heavily on tried-and-tested matrix computation algorithms, but only offers a limited set of operations outside of inner and outer NT products. The NT software we describe, with lattice products forming its computational backbone, aims to provide the MTT's high computational efficiency while offering an environment for general-purpose NT arithmetic even more complete than NumPy.

A good way to gauge success in this regard is to measure performance in executing functionality corresponding to the MTT's specialised code. Ideally, performance should be as good, or better, than the MTT. However, this performance would be realised using the general-purpose lattice data structure, which can execute NT products outside of any specialised routine. To this end, the following computations, found within the third-degree CP decomposition, provide excellent benchmarks:

$$a_{ik\ell}u_{i\underline{j}}v_{k\underline{j}}, \tag{4.9}$$

$$a_{ik\ell}u_{i\underline{j}}w_{\ell\underline{j}}, \tag{4.10}$$

$$a_{ik\ell}v_{k\underline{j}}w_{\ell\underline{j}}. \tag{4.11}$$

Like LibNT and NTToolbox, Blitz++ and NumPy can execute (4.9)-(4.11) algebraically. The MTT, on the other hand, employs its specialised and non-algebraic `mttkrp` routine to execute each of (4.9)-(4.11). Efficient means to execute this routine, which stands for matricised tensor times Khatri-Rao product, is a common topic within the tensor decomposition field [176, 177].

Comparing the performance of Blitz++ and LibNT in executing this benchmark allows the impact of using lattices to be measured. The potential confounding issues of runtime abstraction penalties should not be an issue, since both libraries use similar TMP strategies to support their algebras. The running time to perform (4.9)-(4.11) was measured using dimensions of $N \times N \times N$ for $a_{ik\ell}$ and $N \times 5$ for each second-degree NT factor. This was performed for values of N ranging from 50 to 300 and repeated 10 times for each value of N.

Figure 4.9(a) depicts the result of this benchmark. As the figure illustrates, at all dimensionalities LibNT outperforms Blitz++. As N increases, the performance gap increases, to the point of LibNT running over 10 times faster than Blitz++, demonstrating the merits of using the lattice as the core computational data structure for NT products.

Using the same experimental setup, but with N values ranging as high as 500, experiments measured the performance of NTToolbox vs the MTT and NumPy in executing (4.9)-(4.11). To ensure good speed for NumPy, the code was broken into one binary expression per line. Figure 4.9(b) depicts these results. As the figure illustrates, at low values of N, NumPy's implementation, which does not permute data, is able to match MTT's hand-crafted `mttkrp` routine. However, as N increases, the merits of exploiting fast matrix-multiplication algorithms begin to tell, with the MTT eventually performing over 3 seconds faster.

Focusing on NTToolbox, it matches or exceeds the performance of the MTT at almost all values of N. The exception is at low values of N, where profiling reveals the time spent is dominated by MATLAB's built-in string-matching functions for parsing NT algebra.

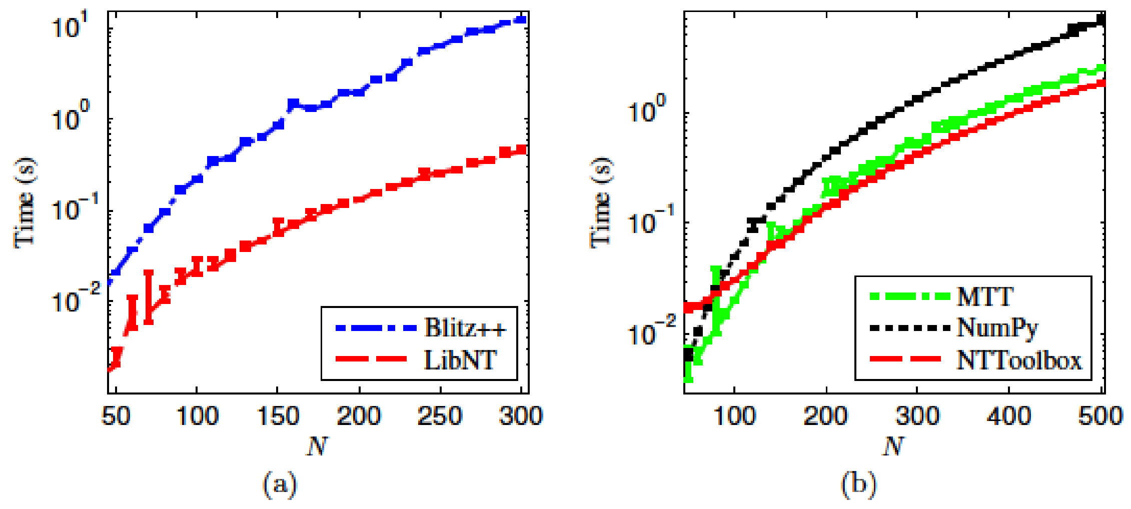


Figure 4.9: Benchmarks for multiplications needed for dense CP tensor decomposition. Tests executed the expressions in (4.9)-(4.11). Trend lines represent median of 10 trials and error bars designate quartiles. (a) depicts performance of LibNT vs Blitz++ while (b) depicts NTToolbox vs the MTT and NumPy.

Supporting this assessment, LibNT, which has almost no runtime abstraction penalties and uses the same lattice-product algorithms, executes at the highest speed at low values of N. However, once runtime becomes dominated by the actual NT products within the `mttkrp` operation, the advantages of the lattice product begin to manifest, eventually outperforming the MTT by roughly 35%. Importantly, unlike the MTT's hand-crafted functions, these capabilities can be extended to any NT algebraic operation.

These results demonstrate that armed with the lattice data structure, LibNT and NTToolbox can efficiently execute a wide-range of NT algebraic expressions, while matching the performance of more specialised and non-generalisable routines. Thus, these results bolster the case for using the lattice as a core computational platform for NT products.

4.4 Summary

A powerful and effective software forms the body of any effective framework within technical computing. Reflecting this, NT software is a central thrust within the development of the NT framework. This chapter is the first of two devoted to NT software, providing a dense foundation for computing NT algebra operations. The LibNT and NTToolbox libraries are introduced, which are open-source libraries for NT algebra implemented in C++ and MATLAB, respectively.

We first provide a general overview of the NT software libraries. To start, we outline the design principles animating LibNT and NTToolbox, which both offer a environment where users can program directly using NT algebra. LibNT is designed to be as computationally efficient as possible, reflected in its use of GP, TMP, and the PSCP idiom to minimise the runtime overhead needed to resolve NT algebraic expressions. NTToolbox, on the other hand, is designed to be as programmatically efficient as possible, resolving NT algebraic expressions entirely at runtime.

Even though both libraries balance programming and computational efficiency differently, they use the same core kernels. The most prominent of which is the lattice data structure, which provides a computational platform to efficiently execute or invert any combination of inner, entrywise, and outer products between N-degree NTs. Importantly, the lattice data structure provides a ready means to interface to tried-and-tested MV computational algorithms. Apart from the lattice data structure, both libraries also share the same rules for checking NT algebra grammar, providing a consistent and almost seamless interface to NT algebra.

These general principles are made concrete through a suite of dense NT algorithms. We detail strategies to reduce superfluous memory consumption in n-ary expressions and shed light on the merits of in-place vs. out-of-place dense permutations. As well, we outline how LibNT can discover efficiencies at compile time to avoid unnecessary index calculations and needless permutations.

Two benchmarks demonstrate the excellent performance of LibNT and NTToolbox. First, small-dimensionality benchmarks test LibNT's success in reducing abstraction penalties compared to the leading Blitz++, FTensor, and LTensor C++ libraries. Results highlight LibNT's competitiveness to these libraries, which is realised despite offering a greater set of arithmetic operations. Second, generalised benchmarks illustrate the merits of the lattice data structure in executing small- to large-scale NT products. These tests reveal that LibNT and NTToolbox, which rely on the lattice data structure, offer a highly general environment for NT computations at speeds competitive to or exceeding that of Blitz++, the MTT, and NumPy.

The principles, algorithms, and data structures outlined in this chapter form a solid foundation for an NT software implementation. Embodied by the open-source implementations of LibNT and NTToolbox, NT software serves as an integral component of the NT framework. Expanding the scope of the NT software further can take several forms. In the next chapter we focus on one such important elaboration, discussing the topic of sparse NT computations.

Chapter 5

Exploiting Sparsity

Efficient sparse-matrix data structures and algorithms have had an unquestionably massive impact on the field of technical computing. This is due to an increasingly greater need to work with large datasets. As Section 2.2.2 highlights, “the curse of dimensionality” magnifies this need within numeric tensor (NT) contexts. Thus, just as was done with matrix-vector (MV) computations, sparse NT data structures and algorithms can provide practitioners with the tools to tackle otherwise intractable large-scale problems. For these reasons, effective sparse data structures and algorithms form an essential component of the NT framework. While many of the principles behind sparse-matrix computations can be employed in a sparse NT context, core data structures and algorithms still must be tailored for this unique and demanding setting.

Apart from work into low-parametric representations [89, 178, 179, 180] and implicit representations of high-degree operators [54, 67, 128], researchers have also focused on how to explicitly represent sparse NTs [124, 150, 151, 152, 176, 177, 181]. Tensor decomposition drives a great deal of this work [124, 150, 151, 152, 176, 177], but applications involving high-degree linear operators [54, 67, 128] and high-degree partial derivatives [181] also see need for sparse NTs. Additionally, the known link between symmetric NTs and polynomial equations [78] introduces further impetus for sparse NT computations. Considering the high-sparsity of real-world polynomial equations [74], if NTs are used to manipulate such equations, efficient sparse algorithms will be needed (likely along with symmetric-specific optimisations [153]).

The topic of data structures and algorithms for general sparse NT operations has been broached previously by Bader and Kolda [124] as part of their MATLAB Tensor Toolbox (MTT). Yet, since the MTT does not provide high-performance kernels of its own [153], there is considerable opportunity to further develop the state of sparse NT computations. As Bader and Kolda themselves note, continued development of sparse-tensor computations is welcome [124].

With this vision in mind, we detail a set of core kernels for sparse computations for the NT framework. Sharing Bader and Kolda’s [124] design philosophy of not favouring any particular index over another, we outline a flexible data structure that is related to, but different from, the one seen in the MTT. This flexibility comes at the cost of heavily relying on sorts and permutes. For this reason, we describe high-performance rearrangement algorithms specifically tailored for sparse NTs, benefiting all sparse NT computations as a whole. Finally, we outline a multiplication poly-algorithm that can effectively compute the products between any NTs exhibiting any manner of hyper-sparsity. Other aspects of sparse NT computations are detailed in Appendix A. Detailed benchmarks demonstrate the high performance of these algorithms, which are embodied within LibNT. NTToolbox offers a MATLAB interface to LibNT's C++ algorithms. As such we will discuss the kernels as belonging to LibNT, with the understanding that NTToolbox also has access to them. Details on the test platform and compiler can be found in Appendix B.

5.1 Data Representation

This section first outlines considerations for sparse NT representation, highlighting the data format used in LibNT. Afterwards, comparative results demonstrate the data format's effectiveness.

5.1.1 Concepts

Sparse MV computations rely on compressed formats [182,183], *e.g.*, the compressed sparse-column (CSC) format, which allows efficient column-centred operations at the expense of inefficient row-centred operations. The compressed sparse-row (CSR) format is identical, with the roles of rows and columns switched. However, Bader and Kolda [124] convincingly argue against compressed formats for NTs, pointing out that it requires categorising an index, or set of indices, differently from others, which becomes less meaningful as the degree of an NT increases.

These arguments are bolstered by considering the operational complexity of NT computations, *i.e.*, the enormous number of ways that NT indices can match up differently for the same arithmetic operation. For instance, enumerating all possible inner/outer product possibilities between two N-degree NTs requires calculating all possible *partial* permutations of the N indices. Partial permutations [184] can be calculated using,

$$P = \sum_{i=0}^{N} i! \binom{N}{i}^2 , \tag{5.1}$$

which grows factorially with degree. The same figure can be used to calculate the number of entrywise/outer products.

When moving to the inner/inter case, enumerating all products entails an even larger sum, as both types of products require a matching of indices. This number can be expressed

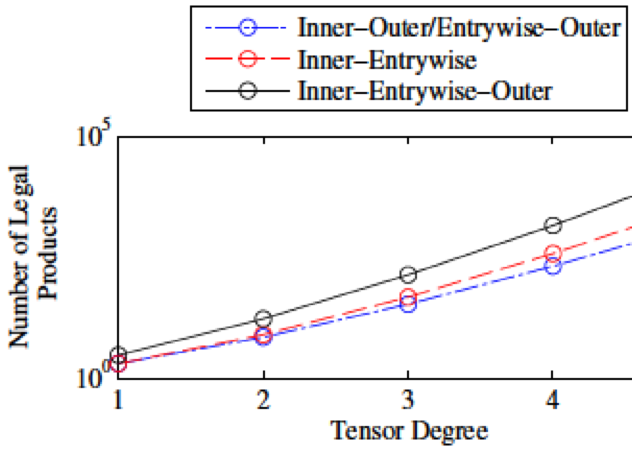


Figure 5.1: Number of possible NT products vs. degree. This figure graphs the number of different possible multiplications between two N-degree tensors, assuming index ranges match up appropriately for inner products. The rate of growth is factorial.

as,

$$P = \sum_{i=0}^{N} i! \binom{n}{i}^2 (N-i)!\,, \tag{5.2}$$

where i would represent the number of indices undergoing an inner (entrywise) product and the trailing factorial would account for the possible entrywise (inner) product permutations of the remaining indices. Considering all three products together means needing to enumerate all possible *partial* permutations of any inter (entrywise) product indices leftover after the first inner (entrywise) product partial permutation has been accounted for,

$$P = \sum_{i=0}^{N} \sum_{j=0}^{i} i! \binom{n}{i}^2 j! \binom{N-i}{j}^2 . \tag{5.3}$$

This sum is much larger than even the inner/entrywise inter case.

Figure 5.1 visually illustrates the scale of this rate of growth, with the number of possibilities of inner/outer, inner/entrywise, and inner/entrywise/outer products at 5 degrees totalling 1546, 3840, and 19 091 respectively. Similar conclusions are drawn when considering addition and subtraction, as the number of possible such operations also increases factorially with degree,

$$P = N!\,. \tag{5.4}$$

This operational complexity indicates that *general-purpose* data structures for sparse NTs should be as flexible as possible, meaning they should not favour any particular index over another. As each different index matching requires a particular lexicographical order, data structures should be amenable to changing lexicographical orders. Even so, several authors have adapted the compressed approach to work with NTs [150,151,152,181]. Typically,

Table 5.1: The CO and LCO sparse formats. Example zero-based indices, from a $4 \times 4 \times 4$ sparse NT with a lexicographical order of $\{0, 1, 2\}$, illustrates the two formats. The CO format uses an expanded list of index values whereas the LCO format uses an LI scheme.

CO Indices:	$\{1,0,0\}$	$\{2,0,1\}$	$\{0,1,1\}$	$\{3,2,2\}$	$\{1,0,3\}$	$\{2,2,3\}$
LCO Indices:	1	18	20	43	49	58

these approaches embed CSC- or CSR-type structures within a high-degree scheme [150,181] or flatten an NT into a compressed-format matrix [151,152]. These solutions would struggle to accommodate the operational complexity inherent in general-purpose computations. Compression schemes would need to be re-computed or there would have to be different code implementations depending on what tasks are performed on what indices. These two approaches become less viable with each increase in degree. This is not to say that compressed formats for NTs have no place, but *general-purpose* NT computations should rely on different data structures.

Bader and Kolda make the case for concurrent and simple lists of non-zero data and index values. When an operation demands a different lexicographical order a re-sort or permute is required. Thus, no indices are favoured over others in terms of operational efficiency. This comes at the cost of relying heavily on rearrangements, *i.e.*, sorts or permutes. As well, since non-zero data is not indexed in any way, non-compressed formats rely heavily on comparisons between index values.

The coordinate (CO) and linearised coordinate (LCO) sparse formats are the two main non-compressed choices that store their non-zeros using straightforward lists. The CO format stores expanded indices, *i.e.*, for an N-degree tensor each of the N non-zero indices. In contrast, the LCO format stores linearised indices (LIs), *i.e.*, N indices represented by a single integer value. For instance, the zero-based LIs for a third-degree NT, a_{ijk}, can be calculated using

$$LI = i + n_i(j + n_j k), \tag{5.5}$$

where $n_{(\cdot)}$ denotes the range of the corresponding index. Such a lexicographical order places greatest significance on the third index, followed by the second and first indices, which we designate numerically as $\{0, 1, 2\}$. Any permutation of the $\{0, 1, 2\}$ sequence is also valid, and this scheme is trivially extended to higher degrees. Table 5.1 illustrates the differences between the two formats. Of note is that the sparse formats are identical for first-degree NTs.

Both formats rely on a lexicographical order to arrange non-zero values. For the CO format, $\{0, 1, 2\}$ indicates that when comparing values, the third index must be considered first, followed by the second and first indices. Altering the lexicographical order requires changing the sequence in which expanded indices are compared. In contrast, for the LCO format, the lexicographical order governs the linearisation scheme used to compute index values. Altering the lexicographical order requires recomputing the LIs. A straightforward integer comparison then suffices to compare indices.

Bader and Kolda opt for the CO sparse format for their MTT library [124]. The CO format has also been used by Parkhill and Head-Gordon for sparse computational chemistry calculations [185]. While Bader and Kolda do not specifically discuss the LCO format, they do mention concerns with linearisation schemes in general, arguing that LIs may overflow integer datatypes. However, their justification uses an example consisting of 32-bit precision integers [124], which is increasingly out-dated considering the current prevalence of 64-bit computers. As well, for cases where LIs do exceed standard integer limits, high-precision libraries [186, 187] offer fast and efficient very-large integer datatypes. For this reason, we do not view integer overflow issues a deciding factor in choosing between CO and LCO formats.

Nevertheless, we place importance on certain other factors. For instance, compared to the CO format, the LCO format is more memory efficient for NT degrees greater than one. Moreover, there is an increased cost of fundamental operations when using the CO format. For instance, comparison operations in the CO format requires up to N individual numerical comparisons for an N-degree tensor. Apart from the increased complexity, the increased memory requirements degrade locality between consecutive non-zero indices, resulting in more cache misses, which can be the deciding factor in sorting performance [188]. This also impacts arithmetic operations. These considerations all add up to the CO format placing greater demands on memory bandwidth, which is often the limiting factor in modern computer architectures [189].

On the other hand, for the LCO format changing the lexicographical order necessitates recomputing LI values. In contrast, the CO format needs to only alter the sequence in which expanded indices are compared. Thus, putting memory storage requirements aside, choosing between the two can come down to comparing the impact of increased comparison, read, and write costs of the CO format vs. the $O(nnz)$ LI re-computation step of the LCO format. This can be judged using benchmark tests.

5.1.2 Results

Since rearrangements of non-zero data forms a linchpin of both data formats, their comparative sorting performance can reveal advantages and disadvantages. As recomputing LIs must often be performed prior to rearranging LCO data, this cost must also considered. If executed naively, recomputing LIs can be very expensive as it requires integer division. However, using fast division libraries, *e.g.*, libdivide [190], can help mitigate this cost. Appendix A.1 details some additional strategies to reduce the cost of re-computing LIs.

To compare the two formats, tests measured the running-time to sort the indices of a fourth-degree sparse NT stored in the CO and LCO formats. As integer division operations are extraordinarily fast when divisors are a power of two, index ranges were chosen to

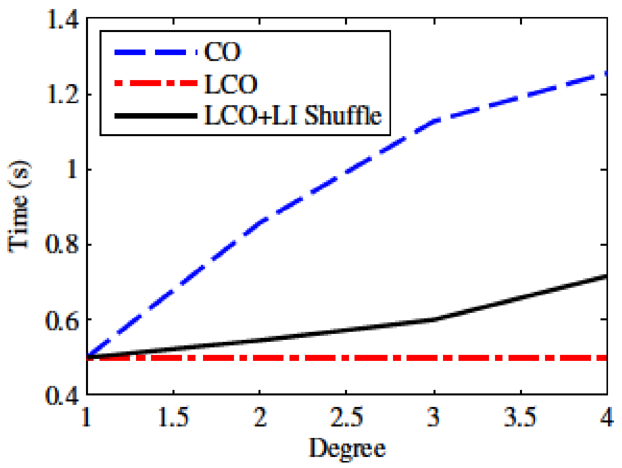


Figure 5.2: Sorting times of the LCO and CO sparse formats. A fourth-degree $N \times N \times N \times N$ NT, with $N = 2^{10} - 1$ and $5N^2$ non-zeros was represented in the two formats and sorted. Experiments also measured the time taken to sort and re-compute LI values when the NT was flattened to lesser degrees. All tests used the same introspective sorting algorithm [191]. Experiments were run 10 times and median values are shown.

be $2^{10} - 1$ to avoid providing the LCO format with an unfair advantage. To maximise memory locality, the CO format did not store expanded indices in separate arrays (as done by the MTT), rather the format used a single large array and packed the expanded indices consecutively one after each other in groups of four. To judge the impact of NT degrees, the same indices were also "flattened" into first-, second-, and third-degree LCO and CO formats. Thus the impact of increasing NT degree, with its increased demands on memory bandwidth and LI computations, was measured under identical conditions. All tests used the same introspective sorting algorithm [191], which was also tailored to sort CO indices.

Figure 5.2 outlines the results of this experiment. As the table demonstrates, recomputing LI values comes with a non-trivial running-time cost, which increases with degree. However, the cost of sorting the CO format increases at a much greater rate, meaning that even with an LI recomputation step included, sorting LCO indices is still much faster than sorting second-degree or higher CO indices. These results indicate that the LCO format is better able to manage the demands of increasing NT degrees. Coupled with the fact that the LCO format uses much less memory at high degrees, these performance metrics lead us to prefer the LCO sparse format over the CO format.

5.2 Rearrangement Algorithms

As noted, one of the major consequences of opting for a non-compressed sparse format is a heavy demand on rearranging non-zero data. The operational complexity of NT operations makes it likely that a sparse NT's lexicographical order would be altered at some point in its lifetime, making rearrangements a frequent first step in many NT operations. Consequently, fast and efficient sparse NT computations can hinge on the algorithmic choices made for these operations.

When discussing rearrangements there are two primary categories. The first category is sorting *unsorted* non-zero data into a desired lexicographical order. For instance, the creation of a sparse NT or insertion of non-zero data could produce an unsorted NT. The second category is re-sorting already sorted data into a different lexicographical order. In the MV paradigm such tasks are called transposition. In an NT context we call such tasks permutations. Despite residing in simple LCO lists, sorted non-zeros possess structure, which can be exploited, calling for dedicated routines different than general-purpose sorting.

5.2.1 Sorting

In a sparse NT context, general-purpose sorting algorithms are well-equipped to tackle sorting NTs. However, unlike many general sorting tasks, when sorting LIs, another array, *i.e.*, the data array, must be sorted alongside it. LibNT's approach, ensuring fast and optimised execution, is use existing and effective sorting algorithms, but adapt them to swap or move the data array based on how the LI array is sorted. Since sorting plays such an integral role for the sparse LCO format, testing the capabilities of different algorithms is an important investment toward developing efficient software. While there is an almost overwhelming amount of literature on sorting techniques, benchmarks and tests involving cases where a second array is sorted based on the first are almost nonexistent. As the demands on memory bandwidth are different than typical sorting applications, it is important to gauge how sorting algorithms fare under a sparse NT setting.

We tested and adapted several different algorithms. The leading comparison-based sorting algorithms that were tested include the highly prominent introspective sort [191] and Timsort [192] algorithms. Integer sorting algorithms were also tested, including a least-significant digit (LSD) radix sort [193] and an in-place version of most-significant digit (MSD) radix sort [194] that uses no extra memory. All of the tested algorithms were adapted to sort the LCO data array alongside any movements of the LCO LI array. In addition, effort was taken to optimise their implementations, including using hybrid and adaptive approaches, in order to achieve fast running times. More details on the considered algorithms can be found in Appendix A.2.

Two different types of tests were performed. The first type of test, depicted in Figure 5.3(a), measures the sorting time on a randomly-generated NT matching the fill-factor of a fourth-degree Laplacian operator, *e.g.*, one that can act on an image. The Laplacian operator's fill factor decreases quadratically with dimensionality, providing a highly-sparse test setting. While it is important to measure performance in highly-sparse settings, it is also worthwhile to test under settings where sparsity does not grow quadratically with dimensionality. Along those lines, Figure 5.3(b) depicts sorting times of fourth-degree NTs with 5% fill factors.

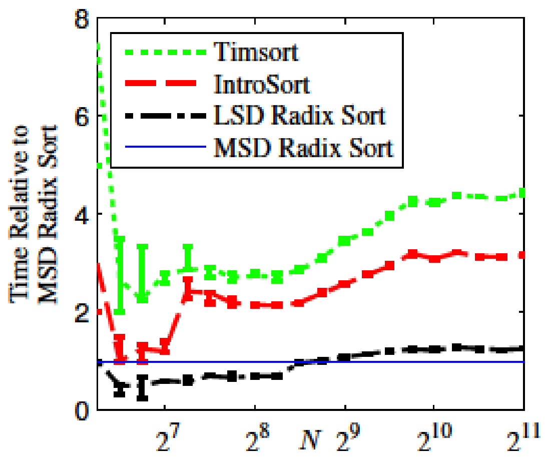
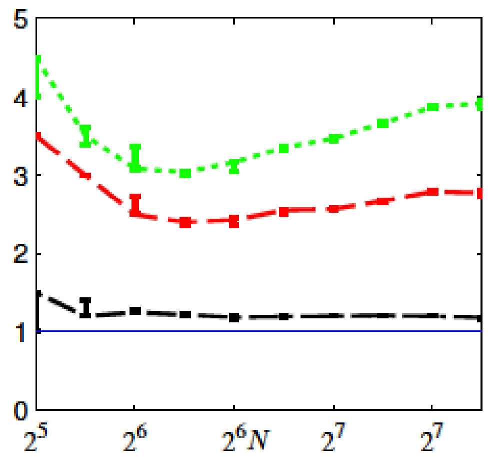


Figure 5.3: Comparison of different sorting algorithms. Benchmarks measured the performance to sort sparse fourth-degree NTs. (a) and (b) depict results from randomly generated $N \times N \times N \times N$ sparse NTs, with a fill factor of a fourth-degree Laplacian and a 5% fill factor, respectively. Tests were run on increasing values of N and repeated 10 times. Trend lines represent median values and error bars represent quartiles.



As the figure makes clear, both radix sorts beat out the two comparison sorts in a highly-sparse setting, posting 2 to 4 times faster speeds for most of the range of dimensionalities. These results are more striking when considering that the benchmark setup is directly unfavourable to radix sorts. Since a fourth-degree Laplacian's fill factor decreases quadratically, the maximum magnitude of the LI values increases at an quadratic rate compared to the number of non-zeros (NNZ). Since the number of $\mathcal{O}(nnz)$ passes radix sort must perform is proportional to the magnitude of the LIs, these results indicate that radix sort can perform extremely well even in the demanding setting of highly-sparse NTs. Similar results were produced when the algorithms were tested on sparse NTs with 5% fill factor, with the radix sorts outperforming their comparison counterparts by highly significant margins.

Apart from illustrating the high-performance of radix sorts, these results demonstrate the significant impact of algorithm choice in sorting sparse NTs. Depending on the choice of algorithm, sorting times vary by factors of roughly 2 to 4, which is of high consequence when considering the importance of rearrangement operations to sparse NT computations. In terms of whether the LSD or MSD variant is preferable, the latter generally outperformed the former, particularly at very-large values of N. Moreover, the MSD version used here is inplace. For these reasons, LibNT uses MSD radix sort as its sort routine for sparse NTs.

5.2.2 Permuting

Apart from sorting, permuting already sorted data into a new lexicographical order is another major rearrangement task. Due to the operational complexity highlighted in Section 5.1, data permutation is a frequent requirement within NT computations, especially if the same NT is used for a series of different operations. While permutation operations are tasked with the same goal as sorting, *i.e.*, rearranging data into a desired lexicographical order, their starting points differ. By taking advantage of the existing structure of already sorted non-zero data, faster means to permutation can be realised. These speedups can be quantified using benchmarks.

Optimising Rearrangement

When permuting data the first step is to recompute the LIs into the new lexicographical order. The scale of the subsequent rearrangement task hinges on the relationship between the starting and ending lexicographical orders. For instance, intuitively it should be simpler to permute sparse NT data from the $\{0, 1, 2, 3, 4\}$ lexicographical order to the $\{1, 0, 2, 3, 4\}$ lexicographical order than it would be to permute it to the $\{4, 3, 2, 1, 0\}$ lexicographical order. This intuition stems from the fact that regardless of the starting-ending lexicographical orders, new LI values will always be arrayed in *sorted sublists*. The characteristics of these sublists can be exploited, taking advantage of any efficiencies that a specific permutation may offer up.

A permutation essentially divides NT indices into two sets—those that require rearranging and those that do not. Figure 5.4 illustrates how this can be determined, with a third-degree NT a_{ijk} and an example permutation. The top of the figure illustrates the bipartite graph of the starting and ending arrangements of i,j,k. Because the i index crosses an index that *originally had* a higher significance, *i.e.*, j, the new LIs must be rearranged according to the i index. We call such indices rearrangement indices. The other two indices do not meet this criterion and thus, the new LIs do not need to be rearranged according to j and k. These indices we call resting indices.

Categorising the indices this way breaks the permutation task into a recursive hierarchy of steps. For instance, working from highest-to-lowest significance of the *new* LIs in Figure 5.4's example, k is a resting index. As a result, rearrangements need not consider k, meaning regions where k is constant can each be independently sorted. Each of these independent regions can then be *stably* sorted based solely on the i index, which is a rearrangement index. Stability means the original relative ordering is used to break ties between equal values. The next resting index j is also the final index, so there is no more work to do. However, if j was not the final index, then the process would have to continue, where each sub-region where j is constant would be independently sorted. This process can be generalised to arbitrary degrees and starting/ending lexicographical orders. An important aspect to note is that the final index is always a resting index.

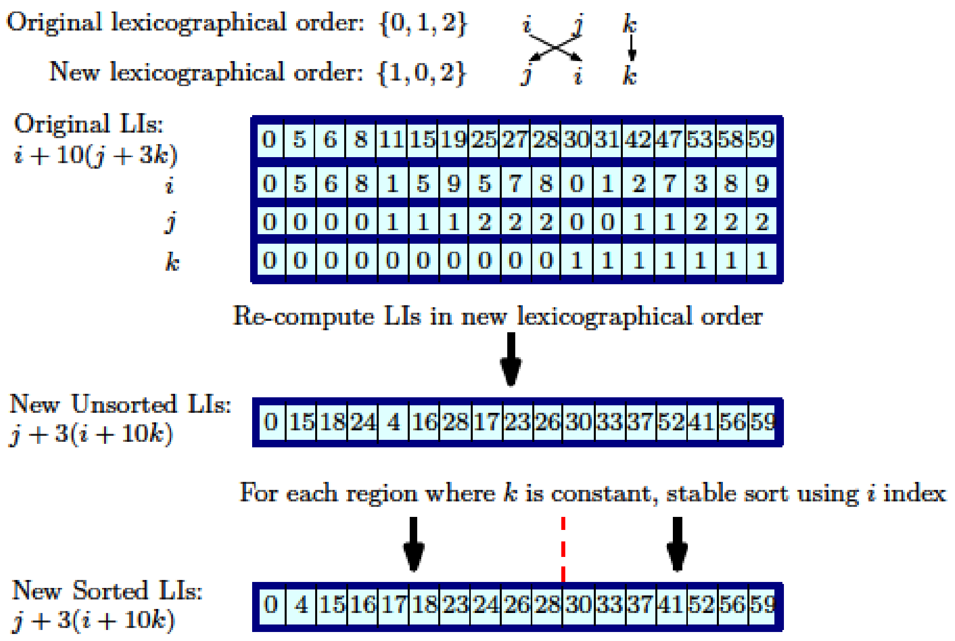

Original lexicographical order: $\{0,1,2\}$
New lexicographical order: $\{1,0,2\}$
i j k
j i k
Original LIs: $i+10(j+3k)$
0 5 6 8 11 15 19 25 27 28 30 31 42 47 53 58 59
i
0 5 6 8 1 5 9 5 7 8 0 1 2 7 3 8 9
j
0 0 0 0 1 1 1 2 2 2 0 0 1 1 2 2 2
k
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
Re-compute LIs in new lexicographical order
New Unsorted LIs: $j+3(i+10k)$
0 15 18 24 4 16 28 17 23 26 30 33 37 52 41 56 59
For each region where k is constant, stable sort using i index
New Sorted LIs: $j+3(i+10k)$
0 4 15 16 17 18 23 24 26 28 30 33 37 41 52 56 59

Figure 5.4: Illustration of sparse NT permutation. Using a $10 \times 3 \times 2$ third-degree NT, a_{ijk}, the figure depicts the original and new LIs corresponding to starting and ending lexicographical orders of $\{0,1,2\}$ and $\{1,0,2\}$, respectively. In this particular case, the index of highest significance in the new LIs, *i.e.*, k, is a resting index, so regions where k is constant can be sorted independently. Each such independent region must be stably sorted based on the index of second-highest significance, *i.e.*, the rearrangement index i. The final resting index j can be ignored.

Returning to Figure 5.4's example, identifying regions in the new LIs where k is constant can be done by integer dividing LIs by $n_j n_i = 30$. Separating the LI values into independent parts benefits all sorting algorithms. For comparison sorts, the asymptotic bounds may be lowered. However, for radix sorts, within each region of constant k, each LI can be examined modulo 30, reducing the maximum possible integer magnitude to accommodate. Depending on the radix digit size, this can reduce the key length, thereby reducing the number of passes a radix sort need perform. Moreover, when sorting each independent region of constant k, the LIs modulo 30 need only be stably sorted using the rearrangement index i. Thus each LI modulo 30 can be reduced even further by integer dividing by $n_j = 3$. In the general case, this aggressive shaving off of irrelevant portions of the LIs can drastically reduce the key length for radix sorts, significantly reducing the number of passes the sort must perform.

As Section 5.2.1 demonstrated, radix sort is well-equipped to sort sparse LCO data. However, key modifications are needed to tailor it to take advantage of the distinct nature of sparse NT permutations. LibNT includes such an algorithm, called radix permutation (RP). Given a starting and ending lexicographical order, a preprocessing step determines which indices are rearrangement or resting indices. The RP routine then employs a stable, but not inplace, variant of the MSD radix sort algorithm. Since shaving off irrelevant portions of the LIs relies on integer division, libdivide [190] is used to to minimise slowdowns. Nonetheless, even when using a fast integer-division library, shaving off LIs comes with a computational cost, which can only be justified if the number of radix sort passes can be reduced. This is typically the case when both the NNZ and magnitude of reductions in LI magnitude are large. Taking this into account, LibNT's RP algorithm is adaptive and will switch to the standard MSD radix sort of Section 5.2.1 depending on conditions.

There are theoretically interesting implications of permuting sparse NT data this way. As Sedgewick explains, radix sorts are often sublinear in the information content of the keys being sorted [193], meaning they can often arrange data without examining every bit. However, this is only an average-case result based on random conditions. Yet, in the context of sparse-NT permutations, by always having *at least* one index a resting index, it is always possible to permute without examining every bit in the LIs. Whether these theoretical gains translate to practical ones is a matter revealed by benchmarks.

Results

Tests measured the permutation performance of RP using fourth-degree NTs sharing the same characteristics as those used in Section 5.2.1. The performance was compared against the MSD radix sort algorithm described in Section 5.2.1. While other algorithms were also tested, including those well suited to sorting already sorted sublists, *e.g.*, natural mergesort, only the MSD radix sort algorithm proved competitive to RP. Figure 5.5 outlines the performance of these algorithms using all $4! - 1 = 23$ permutations of a fourth-degree NT.

As the figure demonstrates, at the median case, RP performs almost identically with

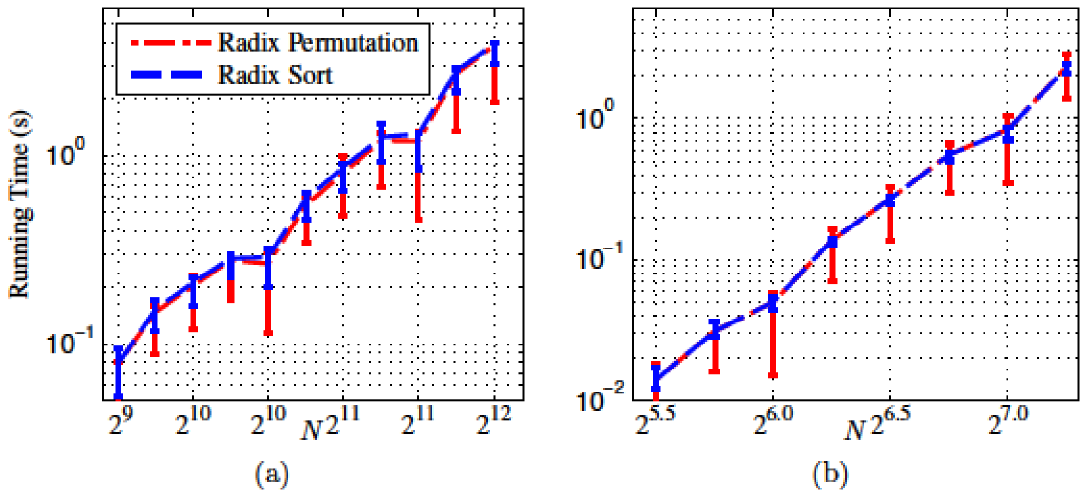


Figure 5.5: Sparse NT permutation benchmark results. Using the same scheme as Figure 5.3, (a) and (b) depict the time taken to permute $N \times N \times N \times N$ fourth-degree Laplacian and 5% fill factor NTs, respectively. All 23 possible permutations were performed, with trend lines displaying median running times and error bars representing minimum and maximum values.

MSD radix sort, indicating that certain permutations do not provide much opportunity for further optimisation. On the other hand, certain permutations *do* provide such opportunities, with the RP algorithm running at significantly faster speeds. It should be noted that RP occasionally ran slightly slower than MSD radix sort depending on the permutation and sparsity characteristics in question. Future work can isolate and remove these instances.

Even so, considering that the MSD radix sort already represents one of the fastest means to *sort* LCO data, the improvements demonstrated by RP attest to the value of using specialised *permutation* algorithms. The highly demanding setting of NT computations makes it necessary to exploit whatever efficiencies are offered. The RP algorithm is a successful embodiment of this principle. It is expected that these gains would only increase with higher degrees, larger fill factors, and greater NNZs. Moreover, since rearrangements are a frequent requirement in sparse NT computations, the improvements garnered here will enhance the speed and usability of all sparse NT computations.

5.2.3 Lazy Approach

The above two subsections detailed the careful development of algorithmic approaches to both sorting and permuting, with the aim of minimising their impact on sparse NT arithmetic operations. However, the best means to minimise the impact of sorting and permuting is to avoid performing them altogether when possible. To this end, LibNT takes a lazy approach, *e.g.*, only opting to sort unsorted data when necessary. As some operations do not require sorted data, *e.g.*, a pure outer product [124], a lazy sorting approach can avoid costly and superfluous rearrangements.

Table 5.2: Possible lexicographical orders after an assignment. This table illustrates possible output lexicographical orders stemming from a sparse $a_{kij} = b_{ijk}$ assignment operation. The result depends on the input lexicographical order and the index permutation from $\{i, j, k\}$ to $\{k, i, j\}$. Only three of the 3! possible input-output lexicographical orders are shown.

b_{ijk} lexicographical order	a_{kij} lexicographical order
$\{0, 1, 2\}$	$\{1, 2, 0\}$
$\{2, 1, 0\}$	$\{0, 2, 1\}$
$\{1, 2, 0\}$	$\{2, 0, 1\}$

In a similar vein to sorting, a flexible approach to permuting can produce gains in sparse computation time. For instance, one can reduce needless computations by being agnostic about lexicographical order, *i.e.*, not restricting LIs to a certain lexicographical order. Under this scheme, sparse NT operands can manifest any legal lexicographical order. This does add to bookkeeping as arithmetic operations must take into account both of the operands' lexicographical orders and also how their NT indices match up. Table 5.2 illustrates this for an example assignment operation. By allowing any legal lexicographical order, simple copies and an appropriate designation of the new lexicographical order suffice for any sparse NT assignments. This applies even for assignments, like that of Table 5.2, nominally involving a permutation. The extra bookkeeping involved is justified by the avoidance of permutations and LI re-computations. Similar considerations apply for other arithmetic operations like addition and subtraction. Appendix A.3 discusses how LibNT minimises sorting and permutation for such operations.

5.3 Multiplication

Multiplying two sparse NTs together epitomises the unique demands of sparse NT computations. As outlined in Section 4.1.2 any NT multiplication, involving inner, entrywise, and outer products, can be represented as a lattice product, *i.e.*, a sequence of matrix products. Thus, sparse-matrix products play a fundamental role in executing sparse-NT products.

However, sparse NTs often exhibit hyper-sparsity. Typically raised in an MV context, hyper-sparsity refers to matrices where the numbers of rows and columns exceed the NNZ [195, 196]. Applied to an NT context, this meaning implies a dimension that exceeds the NNZ. While comparatively rare in linear algebra, graph algorithm applications, which see uses for NTs [197], encounter hyper-sparsity frequently [195, 196].

In addition, even when NTs are not hyper-sparse on their own, they can exhibit hyper-sparsity when they are mapped to lattices during multiplication. For instance, as Figure 5.6(a) demonstrates, flattening a purely diagonal NT can produce index-sparse matrices, meaning both row and column ranges exceed the NNZ. In addition, as Figure 5.6(b) and (c) demonstrate, flattening operations can also produce row- and column-sparse matrices, often manifesting as very-tall and very-wide matrices, respectively, in which only one of the

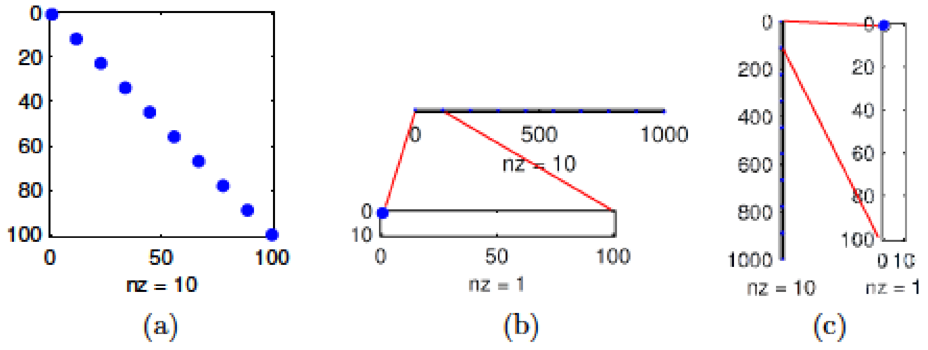


Figure 5.6: Hyper-sparsity of a matricised NT. A fourth-degree diagonal NT $a_{ijk\ell}$, meaning all-zero except for when $i = j = k = \ell$, can produce hyper-sparse matrices when flattened: (a) depicts the hyper-sparsity patten when a $10 \times 10 \times 10 \times 10$ diagonal NT is matricised so that two of its indices are mapped to rows while the remaining are mapped to columns; (b) depicts the same NT, except that is it matricised by mapping one index to rows, while the remaining are mapped to columns; (c) depicts the same as (b) except that the role of rows and columns are reversed. The two figures of (b) and (c) depict column- and row-sparsity, respectively, while (a) depicts index-sparsity.

dimensions exceeds the NNZ. Thus, any routine executing sparse NT products must be able to handle hyper-sparsity.

Hyper-sparsity is not well-handled by typical techniques found within the MV paradigm. We address these difficulties by developing an effective multiplication poly-algorithm designed to handle operands exhibiting any manner of hyper-sparsity. We first overview the poly-algorithm, followed by a description of the specific approaches used to handle the different types of hyper-sparsity that NT operands manifest.

5.3.1 Poly-Algorithm

To establish the workings and motivation behind the sparse NT multiplication poly-algorithm, we first provide an overview of its operation. In addition, we introduce notation necessary to understand the poly-algorithm and the different sub-algorithms it dispatches to. Following this we provide details on the synthetic dataset used to characterise the sub-algorithms used by the poly-algorithm.

Overview

Sparse-NT multiplication can be executed in three steps:

1. map sparse-NT LCO data to lattice form using a permute or sort;
2. convert each LCO lattice tab to an appropriate multiplication datatype (MDT), *e.g.*, CSC, and multiply;
3. map lattice back to a sparse NT.

Table 5.3: Multiplication possibilities based on sparse characteristics of operands. Table entries indicate the sub-algorithm LibNT employs, along with the subsection number describing it, for each sparse-characteristic combination.

A \ B	Sparse	Row-Sparse	Column-Sparse	Index-Sparse
Sparse	CSC/CSR (Section 5.3.2)	CSC (Section 5.3.2)	CSC (Section 5.3.2)	CSC (Section 5.3.2)
Column-Sparse	CSR (Section 5.3.2)	DCSC/DCSR (Section 5.3.3)	DCSC (Section 5.3.3)	DCSC (Section 5.3.3)
Row-Sparse	CSR (Section 5.3.2)	DCSR (Section 5.3.3)	CSCNA/ CSRNA (Section 5.3.4)	CSCNA (Section 5.3.4)
Index-Sparse	CSR (Section 5.3.2)	DCSR (Section 5.3.3)	CSRNA (Section 5.3.4)	SOP (Section 5.3.5)

This is similar to the scheme used by the MTT. However, the MTT uses the same MDT and algorithm regardless of whether operands are hyper-sparse or not. To handle hyper-sparsity, the MTT excises all-zero columns and rows before performing standard CSC multiplication. This excision is an expensive operation requiring extra sorts and additional book keeping, but is necessary because the CSC algorithm is not equipped to handle hyper-sparsity [195, 196].

An alternative is to avoid the cost of excising rows and columns and instead employ algorithms that can naturally handle hyper-sparsity. These will naturally require their own respective MDTs. This is not a disadvantage in sparse NT contexts where conversions to the MDT must happen anyway. Thus, there is considerable freedom in choosing the multiplication algorithm and its MDT. This freedom is not enjoyed within the MV paradigm, in which matrices, once constructed, are typically locked into a single datatype and lexicographical order. The extra flexibility means that a poly-algorithm can be an effective approach, as it there is no detriment, and many benefits, in dispatching to specific multiplication sub-algorithms and MDTs depending on whether the flattened NTs are sparse, row-sparse, column-sparse, or index-sparse.

Table 5.3 outlines the 16 different possible sparsity combinations. In addition, it details the algorithmic choices used by LibNT. Two considerations animated these choices. The primary consideration was on ensuring memory use and running time were not dependant on any hyper-sparse dimension sizes. As Buluç and Gilbert [195, 196] warn, algorithms, *e.g.*, the CSC, whose running time and memory use depend on the dimensionalities of the matrix can consume inordinate amounts of memory or exhibit impractical running times under hyper-sparse conditions. With the first consideration satisfied, the second goal was to gain the fastest running time and/or the lowest memory use. Unlike MV computations, performance metrics of sparse-NT multiplication must include the cost of converting to the MDT. The specific algorithms are discussed in detail within Sections 5.3.2 to 5.3.5.

Table 5.4: Notation used to describe multiplication poly-algorithm.

First Operand	$\mathbf{A}$	Second Operand	$\mathbf{B}$
Rows and Columns of $\mathbf{A}$	m and k	Rows and Columns of $\mathbf{B}$	k and n
Number of Columns of $\mathbf{A}$ with one or more non-zeros	$nzc_{\mathbf{A}}$	Number of Rows of $\mathbf{B}$ with one or more non-zeros	$nzr_{\mathbf{B}}$

Notation

In describing the different multiplication sub-algorithms, we will use a set of common notation outlined in Table 5.4. Since the presence of entrywise products only means that the same type of product is repeated, it does not change the basic approach of sparse-NT multiplication. Thus, for simplicity only inner/outer products will be considered, explaining why the first and second operands of Table 5.4 are matrices, and not lattices. Apart from the notation of Table 5.4, we will use f to refer to the number of floating-point operations in a multiplication, which is the same for all algorithms. $\mathbf{C}$ will denote the matrix product of $\mathbf{A}$ and $\mathbf{B}$. To make the exposition simpler, we will focus mostly on column-by-column versions of the algorithms, *e.g.*, CSC. As such, $f(i)$ will denote the number of floating-point operations to compute the ith column of $\mathbf{C}$ and $nnz_{\mathbf{C}}(i)$ will denote the resulting NNZ of the column. LibNT tests for hyper-sparsity by measuring the ratio of NNZ to the dimension in question. For example, the row-sparsity of $\mathbf{A}$ can be tested by measuring whether $m/nnz_A > 1$.

Dataset

Datasets used for testing can consist of real-world examples or synthetic datasets, the latter of which are parameterised and/or randomly generated. While authors have used real-world networks represented as sparse NTs to characterise decomposition techniques [198, 199], these datasets do not consist of many examples. Thus, to characterise sparse-multiplication algorithms under different conditions, *e.g.*, NNZs, hyper-sparsities, and dimension sizes, this work uses a synthetic dataset.

The dataset consists of a third-degree NT generalisation of the R-MAT recursive graph model [200], which can control for dimension size, fill factor, and fill pattern. In the original R-MAT model, the recursive base edge probabilitys (BEPs) are specified for each quadrant. To generalise to a third-degree R-TENSOR, the BEPs must be specified for each octant. Table 5.5 outlines the probabilities used for this work. The symbol r_{ijk} will be used to denote R-TENSORs. R-TENSORs can manifest column- or row-sparsity depending on their fill factor and how they are multiplied. Moreover, R-TENSORs can also present complete index-sparsity.

Table 5.5: Synthetic dataset used to test the sparse multiplication poly-algorithm. Tests used a dataset consisting of the R-TENSOR model, whose BEP are given with their octant specified in parentheses. To add variability into experiment trials, the probabilities were adjusted by additive values drawn from a uniform distribution of $[-.1\ .1]$ and renormalised so that they all sum to 1.

Octant:	$(1,1,1)$	$(1,1,2)$	$(1,2,1)$	$(1,2,2)$	$(2,1,1)$	$(2,1,2)$	$(2,2,1)$	$(2,2,2)$
BEP:	.3	.5/6	.5/6	.5/6	.5/6	.5/6	.5/6	.2

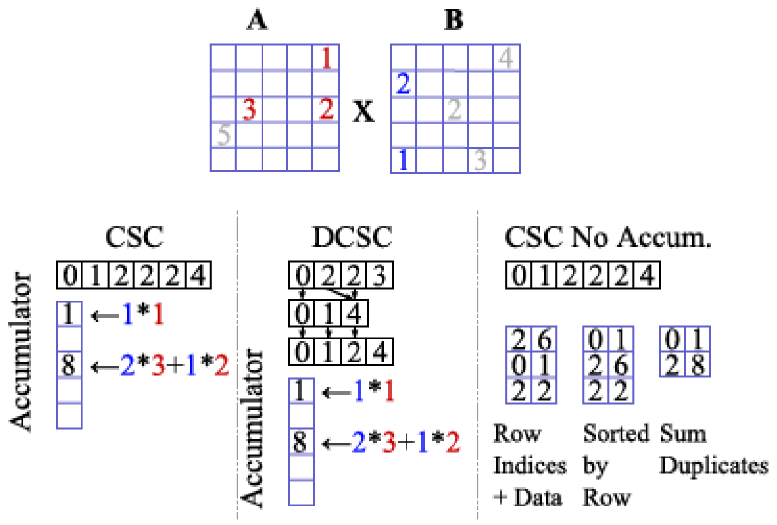


Figure 5.7: Three different column-by-column MDTs and algorithms. A column-by-column algorithm only need represent the left operand, *i.e.*, **A**, using the MDT. Underneath each MDT label, the dense arrays used to index columns of **A** are depicted. Red and blue entries in **A** and **B**, respectively, correspond to entries needed to compute the first column of the resulting matrix. The data structures used to collect entries of the first column are also illustrated.

5.3.2 Regular Sparsity

The standard column-by-column CSC algorithm is a tried-and-tested algorithm that enjoys widespread use. To ensure fast execution, the CSC algorithm relies on a dense, size k, singly-compressed index array to quickly access columns. Moreover, a dense, size m, accumulator array is used to quickly collect non-zeros as each column of **C** is constructed. Similar considerations apply for the CSR algorithm, with the roles of columns and rows reversed. Davis [182] and Buluç *et al.* [183] provide excellent descriptions of these algorithms.

Figure 5.7 includes a visual depiction of the how the CSC algorithm would compute the first column of **C** during its column-by-column run, illustrating the dense index and accumulator arrays. Dense indexing assures fast indexing of columns, while the accumulator ensures contributions to non-zero entries can be quickly added together. The salient characteristics of the CSC algorithm, along with other column-by-column algorithms, can be found in Table 5.6. The final summation term in the runtime originates from the need to sort each column of **C** after each is constructed. Importantly, both runtime and memory

Table 5.6: Asymptotic characteristics of the column-by-column multiplication algorithms. Runtimes include the cost to convert to the MDT from column-major LCO data. Memory use only includes temporary data structures used for multiplication.

Algorithm	Sparse Accumulator	Column Indexing	Runtime	Memory Use
CSC	yes	singly-compressed	$\mathcal{O}(m + k + nnz_{\mathbf{A}} + nnz_{\mathbf{B}} + f + \sum_i^n nnz_{\mathbf{C}}(i) \log nnz_{\mathbf{C}}(i))$	$\mathcal{O}(m + k)$
DCSC	yes	doubly-compressed	$\mathcal{O}(m + nnz_{\mathbf{A}} + nnz_{\mathbf{B}} + f + \sum_i^n nnz_{\mathbf{C}}(i) \log nnz_{\mathbf{C}}(i))$	$\mathcal{O}(m + nzc_{\mathbf{A}})$
CSCNA	no	singly-compressed	$\mathcal{O}(k + nnz_{\mathbf{A}} + nnz_{\mathbf{B}} + \sum_i^n f(i) \log f(i))$	$\mathcal{O}(\max f(i) + k)$

use are dependant upon the dimensionality of $\mathbf{A}$, precluding its use when $\mathbf{A}$ exhibits any hyper-sparsity.

Since the traditional CSC and CSR algorithms are tried-and-tested for standard sparse contexts, LibNT opts for these algorithms when *both* operands present no hyper-sparsity. LibNT gains additional efficiency by converting only one of the matrices to compressed form. For instance, the CSC algorithm only requires fast indexing of the columns of $\mathbf{A}$, meaning it suffices if $\mathbf{B}$ is simply stored in column-major LCO format.

Unlike the MV context, the poly-algorithm can choose between the two formats on-the-fly. The runtime the CSC and CSR algorithms is almost identical, except for the final summation term, where the latter must sort each row of $\mathbf{C}$ instead of each column. Assuming somewhat uniform distribution of non-zeros across rows and columns, the runtime for the sort should be smaller if the task is broken into a greater number of pieces. Thus, LibNT opts for the CSC format when $n > m$, otherwise it chooses CSR.

Finally, by avoiding converting one of the matrices to compressed form, the applicability of the standard algorithms can be extended to greater numbers of cases. For instance, as long as $\mathbf{A}$ has no hyper-sparsity, the standard CSC algorithm can be applied, regardless whether $\mathbf{B}$ is row-, column-, or index-sparse. Thus, the CSC and CSR algorithms can be employed beyond the sparse-sparse case, explaining the first column and row of Table 5.3.

5.3.3 Wide Times Tall

Hyper-sparsity challenges the CSC/CSR algorithms in two manners. The first corresponds to cases where using singly-compressed index arrays are no longer tenable. For instance, should two $N \times N \times N$ cubic R-TENSORS be multiplied using

$$r^{(1)}_{ijk} r^{(2)}_{\ell jk}, \tag{5.6}$$

the left and right operands would be mapped to very-wide and very-tall matrices, respectively. In this case, the CSC and CSR datatypes would require a size N^2 dense array to index the columns and rows of $\mathbf{A}$ and $\mathbf{B}$, respectively. Consequently, when $nnz << N^2$ using the CSC or CSR datatypes is prohibitive or even intractable.

A solution is offered by Buluç and Gilbert [195], who introduced the doubly-compressed sparse-column (DCSC) and doubly-compressed sparse-row (DCSR) formats. Originally used as a means to tackle the fully index-sparse case, these data structures are just as applicable when faced with column-sparsity or row-sparsity. Figure 5.7 depicts the DCSC format, where two-stage indexing replaces the simpler CSC column indexing. Apart from this change, the same column-by-column approach to multiplication can be employed. As Table 5.6 highlights, the DCSC format's doubly-compressed scheme removes runtime and memory-use dependance on k. Thus, the column-by-column DCSC multiplication algorithm can be executed even when $\mathbf{A}$ is column-sparse. Similarly, the row-by-row DCSR algorithm can handle cases when $\mathbf{B}$ exhibits row sparsity. Yet, despite enjoying identical asymptotic running times as their CSC/CSR counterparts, both options come at the cost of additional memory accesses, which can produce a decisive degree of slowdown.

Nonetheless, under the right situations, opting for the doubly-compressed data structures over the singly-compressed versions can avoid possibly catastrophic levels of memory use. As well, even when k fits comfortably within memory capacities, the doubly-compressed scheme can still produce highly significant speedups. To demonstrate this, experiments used the CSC and DCSC algorithms to compute (5.6). The tests used cubic R-TENSORs generated with NNZs ranging from $4e5$ to $1e6$ in increments of $1e5$. Column- and row-sparsity of $\mathbf{A}$ and $\mathbf{B}$ respectively, ranged from 1, *i.e.*, no hyper-sparsity, to 500, *i.e.*, 1 non-zero per 500 columns or rows, in log10-scale increments. This was performed 3 times for each NNZ/hyper-sparsity combination. Finally, two different types of runs were performed. The first run used two different R-TENSORs within (5.6) and the second used the same R-TENSOR for each operand.

The results are depicted in Figure 5.8(a) using a scatter plot based on hyper-sparsity. As the figure demonstrates, the performance of the DCSC algorithm compared to the CSC algorithm is highly dependant on hyper-sparsity, *i.e.*, the column- and row-sparsity of $\mathbf{A}$ and $\mathbf{B}$ respectively. A hyper-sparsity value of 10 separates the point at which the DCSC algorithm outperforms the latter. As hyper-sparsity increases, the DCSC algorithm's running time is on average roughly 20 times faster than the CSC approach, demonstrating a tremendous amount of speedup.

For the purposes of LibNT's poly-algorithm, the library opts for the DCSC or DCSR algorithms whenever column-sparsity and row-sparsity of $\mathbf{A}$ and $\mathbf{B}$ exceeds 3. While lower than the threshold indicated by Figure 5.8(a), this satisfies the primary consideration of keeping memory use reasonably close to the NNZs. LibNT uses the same criteria explained in Section 5.3.2 to choose between the DCSC and DCSR variants. As well, and indicated in Table 5.3, LibNT uses the DCSC algorithm whenever $\mathbf{A}$ is column-sparse and for all cases of $\mathbf{B}$, except when the latter presents no hyper-sparsity. The reverse holds true for the DCSR algorithm.

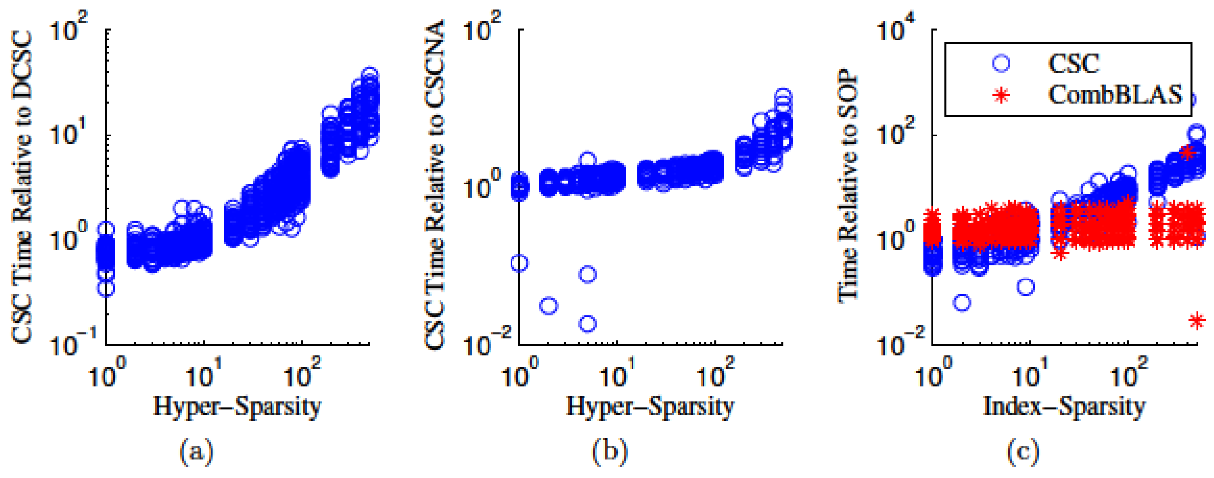


Figure 5.8: Benchmarks of the sub-algorithms for sparse NT multiplication. (a) graphs the ratio of running-times of the CSC vs. the DCSC algorithm under different levels of column- and row-sparsity of **A** and **B**, respectively. (b) graphs the ratio of the CSC vs. the CSCNA algorithms under different levels of row- and column-sparsity of **A** and **B**, respectively. (c) graphs the ratio of the CSC and CombBLAS algorithms to LibNT's SOP algorithm under different levels of *index-sparsity*.

5.3.4 Tall Times Wide

Section 5.3.3 outlined a multiplication strategy to handle cases when using the dense CSC and CSR indexing arrays becomes untenable, *i.e.*, when multiplying a very-wide matrix with a very-tall one. In the opposite scenario, *i.e.*, multiplying a very-tall matrix with a very-wide one, the dense indexing of the CSC and CSR data structures poses no problems and it is the accumulator array that can become untenable. For example, this situation would manifest should two R-TENSORs be multiplied using

$$r^{(1)}_{ijk} r^{(2)}_{\ell mk}. \tag{5.7}$$

In this situation, the standard CSC and CSR algorithms can be modified to forego the accumulator array, resulting in the compressed sparse-column no-accumulator (CSCNA) and compressed sparse-row no-accumulator (CSRNA) algorithms, respectively. As Figure 5.7 illustrates, in the column-by-column case, jettisoning the accumulator array means that as each column of **A** is constructed the non-zeros are not collected and summed together in one step. Instead for each column i, $f(i)$ values are computed and stored in a simple LCO list. These $f(i)$ values must then be sorted and any data values with the same row index are then summed together. As Table 5.6 indicates, this results in increased sorting burdens, but comes at the benefit of not having memory use and runtime be dependant on the potentially huge number of rows of **A**.

As with the DCSC algorithm, improvements can be garnered even when hyper-sparse dimensions fit comfortably in memory. To test this, the R-TENSOR experiments in Section 5.3.3 were repeated, except that (5.7) was computed. As well, NNZs ranged from

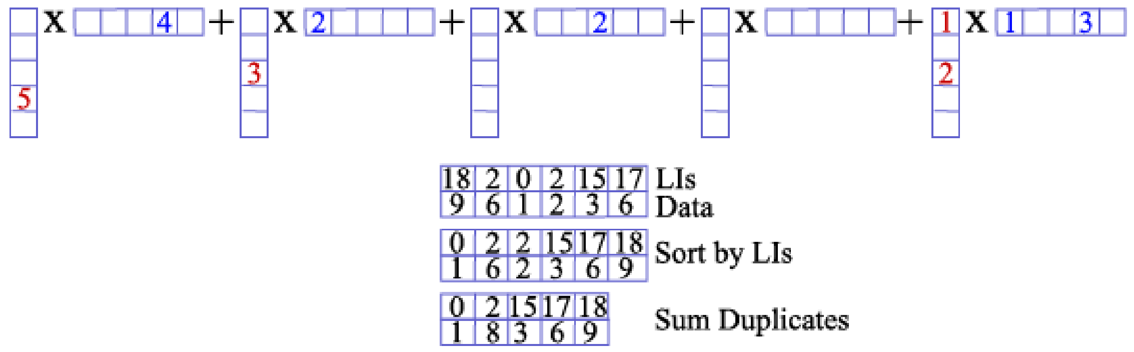


Figure 5.9: The outer-product multiplication algorithm. Using the same example matrix as Figure 5.7, the top of the figure demonstrates how the outer-product algorithm would multiply each column of **A** with each row of **B**. Below, the SOP algorithm used by LibNT to sum each of the k rank-1 matrices is illustrated.

$1e5$ to $3e5$ in increments of $5e4$. Apart from this change, all other test settings were kept identical. Figure 5.8(b) depicts the results of this test, graphing the ratio of running times of the CSC algorithm to the CSCNA algorithm under different levels of row- and column-sparsity of **A** and **B**, respectively. As the figure demonstrates, apart from some outliers at low-levels of hyper-sparsity, the CSCNA algorithm is able to match or exceed the CSC algorithm. At hyper-sparsity values of roughly 3 or higher the CSCNA algorithm begins to exhibit faster execution speeds than the CSC algorithm, eventually running on average 6 times faster. Nonetheless, in isolated instances the CSCNA algorithm performed considerably worse. Characterising when these situations occur is an important area for further investigation. Even so, as the CSCNA algorithm posts excellent performance for the far majority of trials and avoids having memory use and runtime depend on m, LibNT opts for the CSCNA approach whenever row- and column-sparsity of **A** and **B**, respectively, is greater than 3.

As before, LibNT uses the same criteria explained in Section 5.3.2 to choose between the CSCNA and CSRNA variants. LibNT also opts for the CSCNA algorithm whenever **A** is row-sparse and **B** is index-sparse while the CSRNA is chosen whenever **B** is column-sparse and **A** is index-sparse.

5.3.5 Index Sparsity

The final case to consider is when both **A** and **B** are index-sparse. Buluç and Gilbert [195, 196] have demonstrated that the outer-product approach is a fast and memory efficient option for this scenario. Under this approach, **A** and **B** must be sorted in different lexicographical orders—column- and row-major, respectively. As the top of Figure 5.9 demonstrates, each column of **A** is multiplied with each row of **B**, producing k $m \times n$ rank-1 matrices. To produce the final result, these rank-1 matrices must be summed together. Under an index-sparse setting, both $nzc_{\mathbf{A}}$ and $nzr_{\mathbf{B}}$ are each less than k. As well, not all non-zero columns of $\mathbf{A}$ have a matching non-zero row of $\mathbf{B}$. Thus, the number of rank-1 matrices to sum together is always less than k and often less than $\min(nzc_{\mathbf{A}}, nzr_{\mathbf{B}})$.

Different outer-product algorithms will differ in how the intermediate rank-1 matrices are summed together. As part of their CombBLAS library, Buluç and Gilbert use their DCSC and DCSR formats in combination with a heap-like data structure to merge the rank-1 matrices [196]. Designed to act as a component of their large-scale parallel matrix multiplication algorithm, Buluç and Gilbert's motivating problem has different demands than those stemming from sparse NTs. For instance, the authors correctly did not account for the time to construct doubly-compressed matrices in their performance metrics, because their parallel algorithm amortises these costs across sub-tasks. In contrast, the conversion costs to the MDT must be considered in a sparse-NT setting.

As a result, approaches with minimal conversion costs should also be considered. As the bottom of Figure 5.9 illustrates, one approach, which we call the simple outer-product (SOP) algorithm, is to simply concatenate all the intermediate rank-1 matrices together into a length f LCO data-structure. This LCO array can then be sorted based on the LIs, with duplicate entries being summed together. The downside is the $\mathcal{O}(f)$ memory use and a $\mathcal{O}(f \log f)$ sort, which dominates the complexity. However, unlike in regular sparse settings, f can often be small compared to the NNZ of $\mathbf{A}$ and $\mathbf{B}$. Moreover, the ratio of f to the $nnz_{\mathbf{C}}$ of the final result can also be close to 1, meaning strategies to efficiently add duplicate entries do not always justify their overhead.

These conclusions are borne out when multiplying R-TENSORs using a very similar setup as the experiments in Section 5.3.3. However, instead of cubic R-TENSORs, $M \times N \times N$ NTs are used instead, where $M = N^2$. Thus, when flattening the R-TENSORs to compute (5.6), the resulting matrices exhibit square dimension sizes of $N^2 \times N^2$, providing appropriate conditions to vary the row- and column-sparsity together. Additionally, the NNZs varied from $5e4$ to $1e5$ in increments of $1e4$ and the time taken for the CSC, SOP, and CombBLAS' [201] C++ index-sparse algorithm, including setup costs, was measured. All other conditions were kept the same.

The results of this test are depicted in Figure 5.8(c). As the figure demonstrates, the SOP algorithm outperformed the CSC algorithm at levels of index-sparsity greater than 6, with the performance gap increasing to roughly 35 times faster execution at the highest levels of index-sparsity. This emphasises the value of using specialised algorithms in index-sparse scenarios. Compared to the CombBLAS algorithm, the SOP outperformed it on average by a factor of 2 at all levels of index-sparsity, demonstrating the value of a simplified approach in sparse NT settings. However, in several instances the CombBLAS outperformed the SOP algorithm, indicating that certain scenarios call for a more sophisticated merging approach. Further work should focus on identifying these scenarios *a priori*. Even so, the SOP executed the fastest, by significant margins, on almost all test instances.

As a result, LibNT opts for the SOP algorithm whenever both $\mathbf{A}$ and $\mathbf{B}$ are index-sparse.

For the purposes of satisfying the primary consideration of avoiding highly excessive memory use, the SOP algorithm is applied whenever both row- and column-sparsity exceed 3.

5.4 Comparative Performance

So far, this work has highlighted the comparative performance of different algorithmic choices. What has not been discussed is the impact of using such high-performance kernels in a general sparse-NT setting. To unearth some of the significance of using this work's techniques, we compare the performance of the MTT with NTToolbox's MATLAB routines, with the latter consisting of MATLAB classes employing MEX interfaces to LibNT's C++ algorithms.

At the time of writing, the MTT represents the only other software library supporting general-purpose sparse-NT operations. Reflecting the designers' intent, the MTT's focus on tensor decomposition endows it with a suite of routines and data structures that go beyond basic arithmetic operations. With regard to sparse arithmetic operations, it does not support generalised entrywise products, solutions of equations, addition-subtraction with arbitrary index matchings, and operations between dense and sparse data.

In comparing runtime performance, the question should be addressed whether it is fair to compare the MTT's code with NTToolbox, as the latter relies on LibNT's C++ code. First, the developers of the MTT made successful efforts to offload functionality onto MATLAB's built-in and compiled subroutines, minimising the number of commands executed by MATLAB's interpreter. Even so, minimisation is not the same as elimination, and interpreted commands could still hamper performance to varying degrees. However, this does not invalidate comparisons with NTToolbox, as it touches upon the points brought up by Ousterhout [133], and discussed in Sections 2.2.3 and 2.2.4, on programming vs. computational efficiency. As Ousterhout argued, programming tasks are typically either computationally heavy algorithmic development or plugging and gluing these optimised algorithms together. Along these lines, if the MATLAB interpreter is causing significant slowdowns, then this is an argument for offloading certain tasks to a compiled-language implementation. This is the approach taken by NTToolbox.

Two different types of tests were performed. The first benchmarked the performance of LibNT in performing NT products crucial for canonical-polyadic (CP) decomposition. The MTT's specialised routines to compute these products form a core of its sparse CP decomposition routines. The first benchmark touches upon a highly important application, but does not reveal any insights into the impact of LibNT's strategies for handling hyper-sparsity. A second benchmark focuses specifically on this matter, with an emphasis on hyper-sparse multiplication combined with permutations.

5.4.1 Sparse Tensor Decomposition

While the MTT does not explicitly support entrywise multiplication across arbitrary indices, specific instances of these types of products still play an important role in tensor decomposition. For instance the Khatri-Rao product is a crucial kernel in the classic alternating least squares (ALS) algorithm for CP decomposition [83] and remains an important operation in more recent CP decomposition strategies [202].

In the language of NT formalism, the Khatri-Rao product is expressed as

$$a_{ijk} = u_{i\underline{k}} v_{j\underline{k}} \,. \tag{5.8}$$

As also detailed in the dense benchmarks of Section 4.3.2, the Khatri-Rao product is important enough for tensor decomposition applications that the MTT offers dedicated routines to multiply the product in (5.8) with another tensor, without forming the Khatri-Rao product explicitly. In NT algebra, this is expressed as

$$\hat{w}_{\ell k} = x_{ij\ell} u_{i\underline{k}} v_{j\underline{k}} \,, \tag{5.9}$$

and in the MTT library a specific function, `mttkrp`, executes the entirety of (5.9) given an N-degree tensor and $N-1$ appropriately sized matrices.

Since the MTT supplies a specific function designed to efficiently execute (5.9), it is worthwhile to measure how NTToolbox fares using its general NT product routines. To recap the ALS algorithm for the reader, $x_{ij\ell}$ represents the sparse NT to be decomposed, whereas u_{ik} and v_{jk} and $w_{\ell k}$ represent the current estimates of the three dense factor matrices the algorithm is attempting to compute. The hat accent on $\hat{w}_{\ell k}$ in (5.9) reflects the fact that further processing is required before obtaining the updated $w_{\ell k}$ estimate. The alternating aspect of ALS comes from the fact that the algorithm cycles through each factor matrix, estimating it, while leaving the other two constant. Thus, for each iteration, the algorithm computes (5.9) plus the following two calculations:

$$\hat{v}_{jk} = x_{ij\ell} u_{i\underline{k}} w_{\ell\underline{k}} \,, \tag{5.10}$$

$$\hat{u}_{ik} = x_{ij\ell} v_{j\underline{k}} w_{\ell\underline{k}} \,. \tag{5.11}$$

Based on profiling, computing these three equations consumes roughly 75% of the running time when calling MTT's CP ALS algorithm.

To compare the performance of NTToolbox vs. the MTT in computing these Khatri-Rao-type products, $N \times N \times N$ sparse NTs were randomly generated using the MTT's `create_problem` function. This function constructs a randomly generated sparse NT corresponding to a desired fill factor and number of CP factors. Once the sparse NT was generated, the experiment used MTT's `create_guess` function to generate three initial dense factor matrices. Then the time required to compute (5.9), (5.10), and (5.11) was measured. This was repeated for different tensor sizes, fill factors, and number of CP factors.

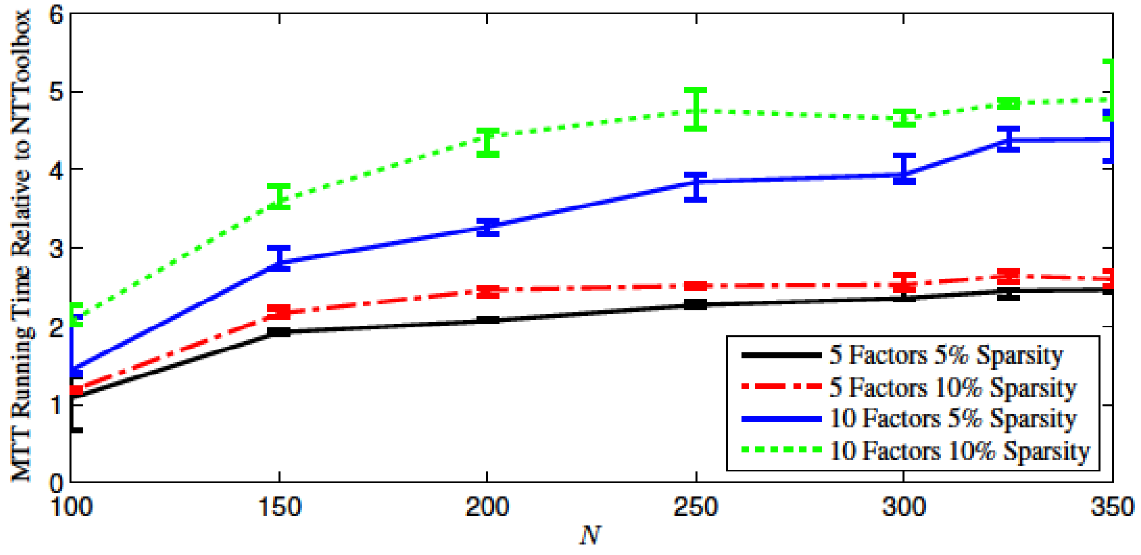


Figure 5.10: Benchmarks for performing sparse Khatri-Rao-type products. Third-degree sparse tensors with cubic sizes were randomly generated using the MTT's `create_problem` function. The ratio of the MTT's running time to NTToolbox's in computing (5.9), (5.10), and (5.11) is graphed for different dimensionalities and the indicated sparsity fill factor and CP factors. For each tensor size, fill factor, and CP factor number, 10 trails were performed. Trend lines depict median ratios and error bars represent quartiles.

The results of this experiment are graphed in Figure 5.10. As can be seen, NTToolbox produced much faster running times than the MTT. For all experiment conditions, NTToolbox's relative performance improved as the size of the CP problem increased. As well, as the number of CP factors and fill factor increased, the performance gap between NTToolbox and the MTT grew to almost a factor of 5. Considering that NTToolbox executed its computations using its own basic functions, and not a specialised function like the MTT, these results demonstrate that the sparse design principles outlined in this chapter can significantly benefit tensor decomposition applications.

5.4.2 Hyper-Sparse Multiplication

The CP decomposition example highlighted performance improvements on an important high-degree exemplar. However, the hyper-sparse poly-algorithm did not come into play. As well, any sparse permutations were of a third-degree NT, which does give the RP algorithm the best opportunity to showcase its performance improvements. To help measure the impact of these innovations, a second benchmark focuses specifically on hyper-sparse multiplication.

Three fourth-degree $N \times N \times N \times N$ NTs were created using the MTT's `sptenrand` routine, with a 5% fill factor. These three NTs were then multiplied using the following

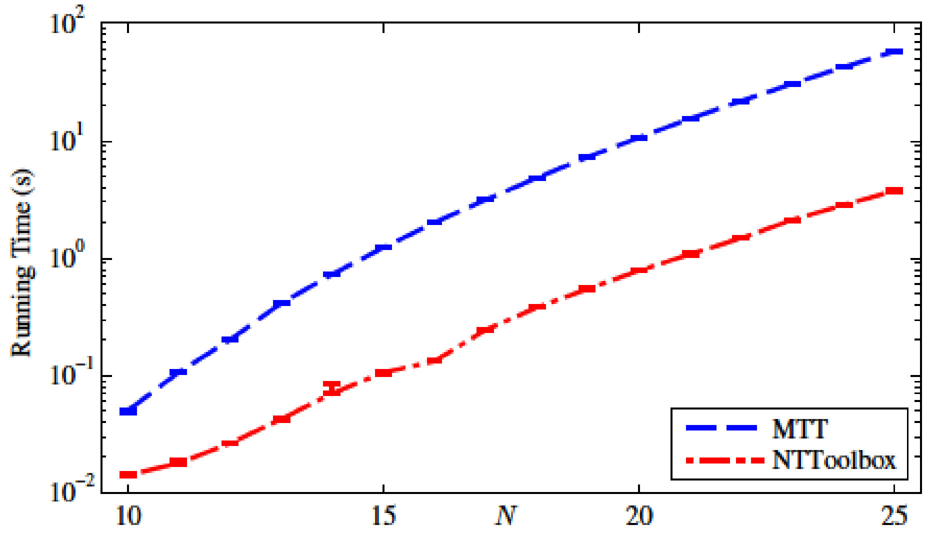


Figure 5.11: Benchmarks for hyper-sparse products. The performance of NTToolbox and the MTT in executing (5.12), an expression incorporating permutations and column- and row-sparse matrix multiplications, was measured using increasing sizes of N. Trend lines depict median values and error bars represent quartiles.

expression:

$$(a_{ijk\ell}b_{pmnk})c_{nplr}, \tag{5.12}$$

which requires both permutations and the multiplication of highly column- and row-sparse matrices[1]. As such, (5.12) is well-suited to highlight the impact of the permutation algorithm and multiplication poly-algorithm discussed in this work. The running time to execute (5.12) was measured across different values of N.

As Figure 5.11 demonstrates, NTToolbox outperforms the MTT by very-wide margins, running at minimum 4.5 times faster. At the highest value of N this margin increases, with NTToolbox completing the task at roughly 3.8 s while the MTT requires roughly 58 s. These performance gaps are large enough to mean the difference between practical and non-practical running-times.

These results highlight the importance of high-performance kernels even under prototyping settings. As well, they confirm that the principles animating the RP algorithm and the multiplication poly-algorithm can accelerate sparse NT computations. For this reason, they represent important advancements to the state of the art.

5.5 Summary

As means to overcome the curse of dimensionality, sparse computations play a pivotal role for high-degree computations. For this reason, sparse data structures and algorithms play a

[1]To execute using the MTT, (5.12) was translated into n-mode$^+$ notation.

central role within the NT software of the NT framework. Along with the MTT [112, 124], LibNT and NTToolbox stand alone in offering a general-purpose environment for sparse NT computations. The NT software is differentiated by several novel high-performance kernels, implemented as C++ routines within LibNT. NTToolbox interfaces to these algorithms through MATLAB. While researchers have published high-performance arithmetic algorithms tackling specific tasks [176, 177, 181], LibNT's high-performance routines are tailored for general-purpose sparse NT arithmetic, meaning addition, subtraction, multiplication, and the solution of equations incorporating dense, sparse, or dense/spase NT mixtures.

We focus on three core aspects of LibNT's sparse functionality. First, like Bader and Kolda [124], we believe a general library should not place an *a priori* precedence on certain NT indices over others. However, we argue for the LCO format over Bader and Kolda's CO format, based on its faster sorting performance and lower memory consumption.

Secondly, we emphasise the importance of high-performance rearrangement algorithms when using list-like data structures like the CO and LCO formats. Such algorithms are necessary to realise a high-performance sparse NT library. We outline the impact of using radix sort, which is specialised to sort integer datatypes, over more general-purpose sorting algorithms. As well, we are the first to outline how the rearrangement task faced by permutations can be optimised by exploiting the structure inherent in permuted data. The resulting algorithm can rearrange data faster than the best sorting option, underscoring the importance of employing routines tailored for sparse NTs.

Finally, we address the topic of sparse-times-sparse NT multiplication, which faces difficulties distinct from those found within the MV paradigm. In particular, a general NT library could encounter any combination of sparse, index-sparse, column-sparse, or row-sparse data, which all demand their own specialised approaches. Apart from highlighting this unique characteristic, we also outlined a multiplication poly-algorithm that can choose appropriate algorithms accordingly. This poly-algorithm ensures that excessive memory use is avoided, a potentially catastrophic event. Moreover, the poly-algorithm produces highly-significant reductions in running time over the common CSC/CSR approach. While the MTT can also handle hyper-sparsity, its one-size-fits-all algorithm does not exploit the very distinct features of the different sparsity types.

Compared to the MTT, the outlined kernels contributed to considerable improvements in running time on benchmarks focusing on arithmetic operations stemming from CP decomposition and hyper-sparse multiplication. These performance gains demonstrate the value of the outlined high-performance kernels, which are tailored to the unique challenges of sparse NT computations. All of the discussed high-performance kernels are accessible through LibNT and NTToolbox, acting as a backbone to the NT framework. However, as with any innovations regarding NT computations, we view these sparse kernels as a contribution to the general field of high-degree software. As such, the data structures and algorithms described here, or variants thereof, are also well suited to any other package incorporating sparse NTs. Thus, these kernels represent an important contribution to the burgeoning field of NT software.

In concert with NT algebra and the dense software foundation, these sparse innovations help complete the NT framework. With the NT framework described, exemplar problems can be tackled, illuminating aspects of technical computing practice beyond the MV paradigm. This is the focus of the following chapter.

Chapter 6

Differential Operators

Special linear mappings, typified by high-degree linear mappings and entrywise products, are an exemplar category not well-handled by the matrix-vector (MV) paradigm. In contrast, a numeric tensor (NT) framework easily expresses and computes such mappings. Differential operators are a recurring source of special linear mappings. Applied to gridded data, *e.g.*, an image, often as part of a partial-differential equation (PDE), differential operators commonly take the form of finite-difference (FD) operators. The need for effective solutions for high-degree differential operators has driven a great deal of work on high-degree algebra and software. This includes the EinSum [54,67] and POOMA [128] software libraries, which provide implicit representations of common high-degree FD operations. Complementary to these efforts, the NT software uses explicit sparse representations, which are necessary in order to construct linear systems which must be later solved. The widespread need for high-degree differential operators means they are an excellent source of exemplars that showcase the merits of the NT framework.

This chapter illustrates three differential-operator exemplars taken from two computer vision problems, called interactive image segmentation and depth-map estimation. Together, these exemplars involve high-degree mappings, entrywise products, linear inversion, non-linear functions, separable representation, and differentiation, capturing many of the features highlighted in current pushes for frameworks beyond the MV paradigm. Moreover, the chapter emphasises aspects not easily implemented using the MV paradigm. As the exemplars can be framed as PDEs, the demonstrated benefits are relevant to many fields outside of computer vision.

The selected exemplars have also been important aspects of the author's work in computer vision, having been featured in three different publications [55,164,203]. The inherent challenges of working on such exemplars has motivated much of the NT framework's development. This influence goes both ways, as the NT framework has impacted the approach used to tackle these problems, *e.g.*, the author's latest work on depth-map estimation, which incorporates a high-degree generalisation of separable nonlinear least squares (SNLS). This aligns with Kuhn's assertion that the need to address anomalies stemming from important problems motivates the pursuit of models, tools, and techniques beyond a scientific paradigm. Moreover, when these anomalies are viewed as key components of new exemplars, they can begin to demarcate the possibilities and boundaries of an alternative paradigm.

The chapter begins with a discussion on the importance and challenges of FD operators in high-degree domains, such as image processing and computer vision. Afterwards, the chapter outlines how the NT framework can benefit random walker (RW) segmentation, a prominent image segmentation algorithm. This is followed up by a discussion on how the NT framework's capabilities can aid depth-map estimation problems.

6.1 Finite-Difference Operators

Differential operators are a bedrock of scientific and engineering modelling. The most notable manifestation of differential operators is within PDEs, whose importance to scientific and engineering practice is hard to overstate. PDEs are just as crucial for image processing and computer vision, where techniques, such as depth-map estimation [55], image-segmentation [56,204], denoising [205], and image enhancement [204,206,207], are frequently cast as PDEs.

In digital imaging, which is typically characterised by gridded pixels, differential operators frequently take on the form of FD operators. Apart from PDEs, FD operators are also useful for computing image gradients, *e.g.*, for the purposes of edge-detection [208], optical flow [209], or object-recognition features [210]. Whether used for PDEs or other purposes, as Figure 6.1 emphasises, FD operators must meet the challenges inherent within high-degree data.

For instance, in digital imaging, it is often of interest to explicitly formulate the FD operator acting upon an image. If applied as part of a PDE, the FD operator can be used as part of a design matrix. If this is done using MV algebra, the digital image must be flattened into a vector. The FD operator must then be created using the same lexicographical order, which typically involves using a large offset to index rows or columns. This process is burdensome and error prone [67]. Digital imaging is not unique in needing to work with FD operators applied to high-degree domains, as witnessed by the discussion of flattening schemes in prominent texts on numerical methods and technical computing [47,57,58].

In one degree, on the other hand, FD matrix operators are relatively easy to construct. For example, should $O(h^2)$ central differencing be required, the FD matrix can be constructed using

$$\mathbf{D} = \frac{1}{2}\left\{ \begin{pmatrix} \mathbf{0} & 0 & 0 \\ -\mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \\ 0 & 0 & \mathbf{0} \end{pmatrix} + \begin{pmatrix} -3 & 4 & -1 & \mathbf{0} & 0 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 0 & 0 & \mathbf{0} & -1 & 4 & -3 \end{pmatrix} \right\}, \quad (6.1)$$

where the first and last rows of $\mathbf{D}$ are filled using $O(h^2)$ forward and backward-differencing operators. Each $\mathbf{0}$ is a sub-matrix of appropriate dimensions.

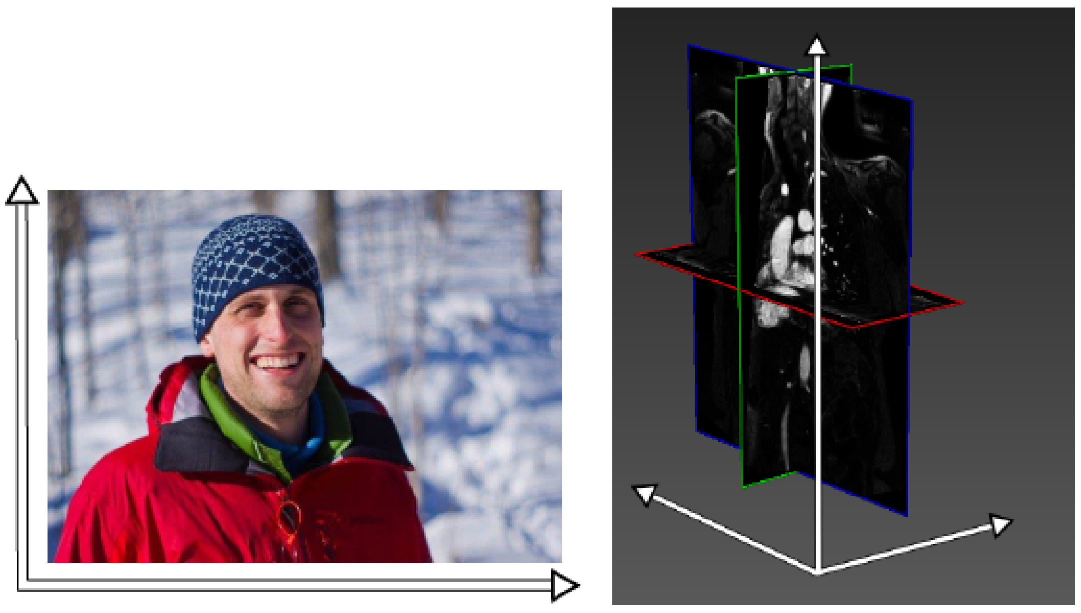

(a) (b)

Figure 6.1: The high-degree nature of imaging poses challenges for FD operators. Natural images are 2D or 3D if colour is taken into account, *e.g.*, (a). Medical images are often 3D, *e.g.*, the CT scan in (b). When a time series is taken, medical images can even be 4D. Volume rendered using the Medical Imaging Interaction Toolkit [211].

Unlike MV algebra, NT algebra can just as easily express FD operators for higher degrees. First, the $\mathbf{D}$ operator should be converted to a second-degree NT, *i.e.*, $d_{ii'} = \mathbf{e}_i^{\mathsf{T}}\mathbf{D}\mathbf{e}_{i'}$, designed to operate on first-degree data. In NT algebra, this operator can be easily applied to any of the indices of an NT. For instance, if a 3D image is represented using a third-degree NT, I_{ijk}, then the three possible gradients can be represented as

$$I^y_{ijk} = d^y_{ii'}I_{i'jk}, \tag{6.2}$$
$$I^x_{ijk} = d^x_{jj'}I_{ij'k}, \tag{6.3}$$
$$I^z_{ijk} = d^z_{kk'}I_{ijk'}, \tag{6.4}$$

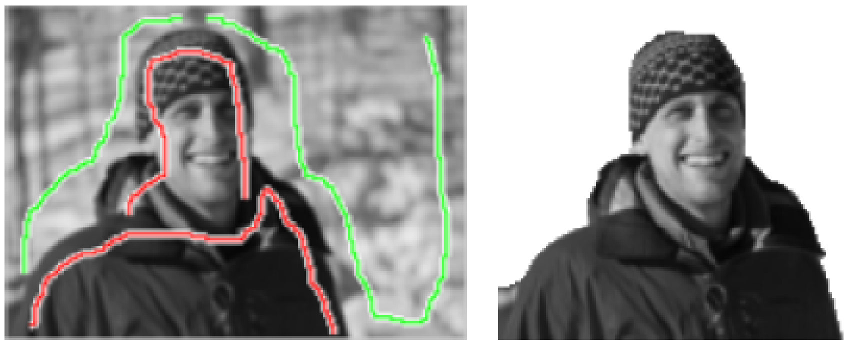
where the ranges of indices undergoing an inner product are assumed to match.

It may also be desired to combine gradient values together, *e.g.*,

$$\begin{aligned} f_{ijk} &= I^y_{ijk} + I^x_{ijk} + I^z_{ijk}, &(6.5)\\ &= d^y_{ii'}I_{i'jk} + d^x_{jj'}I_{ij'k} + d^z_{kk'}I_{ijk'}. &(6.6)\end{aligned}$$

While (6.6) is an unlikely formulation, versions very similar to it, *e.g.*, incorporating second-order derivatives, are common when problems are formulated as Poisson or Laplacian PDEs. To isolate I_{ijk}, the associativity and distributivity of NT products over addition can be used. For instance, using the factoring rules of Table 3.1, (6.6) can be reexpressed as

$$f_{ijk} = d^y_{ii'}\delta_{jj'}\delta_{kk'}I_{i'j'k'} + d^x_{jj'}\delta_{ii'}\delta_{kk'}I_{i'j'k'} + d^z_{kk'}\delta_{ii'}\delta_{jj'}I_{i'j'k'}, \tag{6.7}$$



(a) (b)

Figure 6.2: Foreground and background labelling for 2D RW image segmentation. Interactive segmentation requires that the user designates a set of foreground and background seed labels. In this case, the user has painted (a) with red and green brush strokes to label selected pixels as foreground and background, respectively. Interactive segmentation algorithms then propagate seed labels to the entire image, resulting in a masked image, *e.g.*, (b). The segmentation in this example was performed using the RW algorithm, executed using NTToolbox.

where the right-hand side can be collected together to form a design NT:

$$a_{ijki'j'k'} = d^y_{ii'}\delta_{jj'}\delta_{kk'} + d^x_{jj'}\delta_{ii'}\delta_{kk'} + d^z_{kk'}\delta_{ii'}\delta_{jj'}, \tag{6.8}$$

forming a high-degree linear system to solve:

$$f_{ijk} = a_{ijki'j'k'} I_{i'j'k'}. \tag{6.9}$$

Doing the same in the MV paradigm requires flattening operations based on the lexicographical order. Moreover, as Sections 6.2 and 6.3 will demonstrate, FD operators for computer-vision problems must often work in concert with entrywise products, non-linear functions, and linear inversion of high-degree equations, posing additional challenges. These can be met by the NT framework.

6.2 Random Walker Image Segmentation

Image segmentation is one of the fundamental aims of computer vision and medical-imaging analysis [212]. Roughly speaking, the goal of image segmentation is to isolate region(s) of interest from a 2D or 3D image. *Interactive* image segmentation is a major approach to this problem, where the result is guided by user input. Figure 6.2(a) illustrates a common manifestation of interactive image segmentation where virtual brush strokes designate sets of seed points. These seed points specify foreground and background regions. Based on these seed points, the segmentation algorithm propagates the seed-point labelling to all unseeded pixels or voxels in the image. Figure 6.2(b) depicts an example of how the seed-point labelling in Figure 6.2(a) could be propagated. More than two label types are also possible.

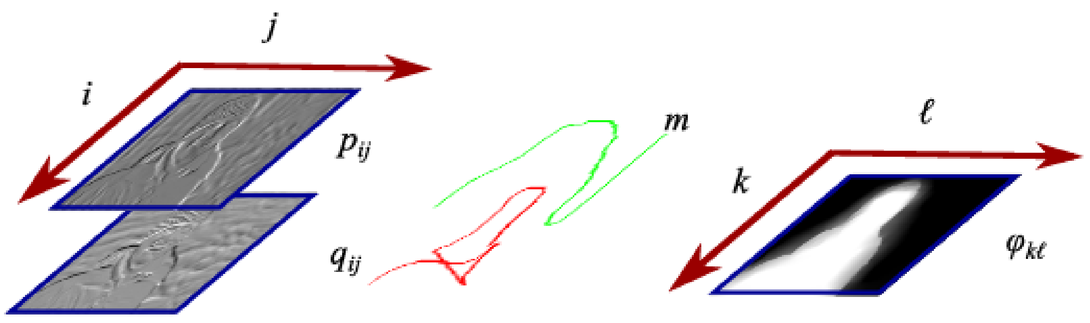


Figure 6.3: At least four indices are required for 2D RW segmentation. Affinity between neighbouring pixels is calculated using image gradients, p_{ij} and q_{ij}. Based on the seed points and these affinities, a labelling potential, $\phi_{k\ell}$, is calculated. This potential can be thresholded to produce a segmentation mask. Depending on how the seed-point constraints are incorporated, a fifth index, *i.e.*, m, could index the seeded pixels. Another index which is not shown, would index the unseeded pixels.

The RW algorithm [56] is one of the most popular and influential approaches to interactive image segmentation. Pertinent to this discussion, the RW algorithm relies heavily on high-degree differential operators, the challenges of which the NT framework can address. We omit explaining the RW algorithm using the MV paradigm, as such an exposition can be found in Grady's original paper [56].

Section 6.2.1 begins by describing the RW formulation using NT algebra. This is followed by Section 6.2.2, which outlines how the RW can be solved. Afterwards, Section 6.2.3 outlines some important extensions to the RW algorithm that are easily accommodated by the NT framework but which challenge the MV paradigm even further. Finally, Section 6.2.4 contrasts the computational and programming efficiency of MV paradigm vs. NT framework implementations of the RW algorithm.

6.2.1 Formulation

As Grady explains in his original paper, the RW algorithm can be motivated through several analogies, including diffusion equations and circuit analysis [56]. Regardless of how it is motivated, the formulation boils down to determining a pixel labelling that minimises differences across neighbouring and similar pixels. As Figure 6.3 illustrates, in the 2D segmentation case the RW algorithm must contend with at least four indices. Moreover, as will be shown, entrywise products play an important role. Thus, the MV paradigm is not naturally equipped to model this image-segmentation task.

To see this, consider first that the RW algorithm attempts to determine a labelling for all pixels or voxels in an image. In a 2D setting $\phi_{k\ell}$ designates the labelling, which is a soft floating-point valued labelling often called the potential. In the case of Figure 6.2(a) user-defined virtual brush strokes designate sets of seed points, S_f and S_b, that are constrained to be in the foreground and background, respectively. To propagate the seed-point labelling to the rest of the image, a natural approach is to make it easier to propagate labels when neighbouring pixels are alike and harder when neighbouring pixels are dissimilar.

There are many ways to compute pixel similarity, but the most common is to use intensity differences between neighbours, *e.g.*, Grady's use of a Gaussian kernel [56]. Using a 4-connected grid, intensity differences can be computed using the 2D analogues of the gradient calculations in (6.2)-(6.4). The x and y gradients can be represented as p_{ij} and q_{ij} respectively. These gradients can be incorporated into a Gaussian kernel to produce two sets of similarity weights,

$$w^x_{ij} = \exp\left(\frac{-p_{ij}}{\beta}\right), \tag{6.10}$$

$$w^y_{ij} = \exp\left(\frac{-q_{ij}}{\beta}\right), \tag{6.11}$$

where β is an algorithm parameter.

Each weight value is used to penalise any differences in potential values across neighbouring pixels, where penalty values are greatest for neighbouring pixels of similar intensity. The goal then is to minimise the penalties. This task can be formulated as determining the labelling potential that minimises the following energy functional:

$$\begin{aligned} f(\phi) &= (d^y_{ii'}\phi_{i'j})(d^y_{ii'}\phi_{i'j})w^y_{ij} + (d^x_{jj'}\phi_{ij'})(d^x_{jj'}\phi_{ij'})w^x_{ij}, && (6.12)\\ &= (d^y_{ii'}\phi_{i'j})(d^y_{\underline{i}i'}\phi_{i'\underline{j}}w^y_{\underline{ij}}) + (d^x_{jj'}\phi_{ij'})(d^x_{\underline{j}j'}\phi_{\underline{ij'}}w^x_{\underline{ij}}), && (6.13)\\ &\text{s.t.} \begin{cases} \phi_{ij} = 1\,,\ \{i,j\} \in S_\mathrm{f} \\ \phi_{ij} = 0\,,\ \{i,j\} \in S_\mathrm{f} \end{cases}, && (6.14) \end{aligned}$$

where d^x_{ij} and d^y_{ij} denote one-dimensional FD operators with appropriate dimensions for the x and y directions, respectively. This formulation can be understood as minimising the weighted squared difference between pixel labels.

The formulation in (6.12) expresses a ternary inner product between the FD operators and the weightings, resulting in an anisotropic Laplacian operator. The subsequent expression in (6.13) uses the association identity to isolate one of the ϕ operands. In the original formulation, Grady described these products as a weighted inner product, which resulted in a combinatorial Laplacian matrix. To describe the energy term of (6.12) Grady predominantly used MV notation, but resorted to a mix of diagonal matrices, implicit vectorisations, and isolated instances of Einstein notation. In contrast, (6.12) expresses all operations algebraically. Additionally, (6.12) can be easily extended to 3 or more dimensions.

6.2.2 Solution

With the energy functional constructed, solving for the energy potential requires finding its minimum. This requires taking the derivative of $f(\phi)$ with respect to $\phi_{k'\ell'}$,

$$\frac{1}{2}\frac{\partial f(\phi)}{\partial \phi_{k'\ell'}} = (d^y_{ii'}\delta_{i'k'}\delta_{j\ell'})(d^y_{\underline{i}i'}\phi_{i'\underline{j}}w^y_{\underline{ij}}) + (d^x_{jj'}\delta_{ik'}\delta_{j'\ell'})(d^x_{\underline{j}j'}\phi_{\underline{i}j'}w^x_{\underline{ij}})\,, \tag{6.15}$$

$$= d^y_{ik'}(d^y_{\underline{i}i'}w^y_{\underline{ij}}\delta_{\underline{j}\ell'})\phi_{i'j} + d^x_{j\ell'}(d^x_{\underline{j}j'}w^x_{\underline{ij}}\delta_{\underline{i}k'})\phi_{ij'}\,, \tag{6.16}$$

$$= d^y_{ik'}(d^y_{\underline{i}k}w^y_{\underline{i\ell}}\delta_{\underline{\ell}\ell'})\phi_{k\ell} + d^x_{j\ell'}(d^x_{\underline{j}\ell}w^x_{\underline{kj}}\delta_{\underline{k}k'})\phi_{k\ell}\,, \tag{6.17}$$

$$= a_{k'\ell'k\ell}\phi_{k\ell}\,, \tag{6.18}$$

where the association identity was used in (6.16) to isolate ϕ. In (6.17), the dummy indices for ϕ were changed to k and ℓ, aligning with Figure 6.3's conventions. The NT $a_{k'\ell'k\ell}$, defined by (6.17) and (6.18), forms the combinatorial Laplacian operator discussed by Grady. The next step consists of setting the product in (6.18) to zero, forming a Laplacian-like elliptic PDE, and solving the resulting equation for $\phi_{k\ell}$. However, such a problem is underdetermined without constraints. The seed values fill this role, acting as non-boundary Dirichlet constraints.

There are several means to incorporate the seed-value constraints. One approach, used by Grady [56], reduces the problem size by decomposing the labelling potentials into unseeded and seeded components. For instance, a sequence of seed labels can be placed into their respective pixel coordinates via a selection NT:

$$\phi^{\mathrm{s}}_{k\ell} = \pi^{\mathrm{s}}_{k\ell m}s_m, \tag{6.19}$$

where s_m denotes the list of known seed labels that are either 0 or 1 valued, $\phi^{\mathrm{s}}_{k\ell}$ is a second-degree NT of all-zero values except for 1-valued seed points, and $\pi^{\mathrm{s}}_{k\ell m}$ represents a selection NT that places seed labels into their matching image coordinates. A similar formulation is possible for unseeded labels:

$$\phi^{\mathrm{u}}_{k\ell} = \pi^{\mathrm{u}}_{k\ell n}u_n, \tag{6.20}$$

where u_m denotes the list of unseeded labels whose values must be determined and $\pi^{\mathrm{u}}_{k\ell n}$ is the corresponding selection NT. Thus, the complete labelling can be expressed as the sum of seeded and unseeded labels:

$$\phi_{k\ell} = \phi^{\mathrm{s}}_{k\ell} + \phi^{\mathrm{u}}_{k\ell}. \tag{6.21}$$

Since the unseeded labels are the unknowns, they must be solved for. The derivative of $\phi_{k\ell}$ with respect to u_n is simply

$$\frac{\partial \phi_{k\ell}}{\partial u_{n'}} = \pi^{\mathrm{u}}_{k\ell n'}. \tag{6.22}$$

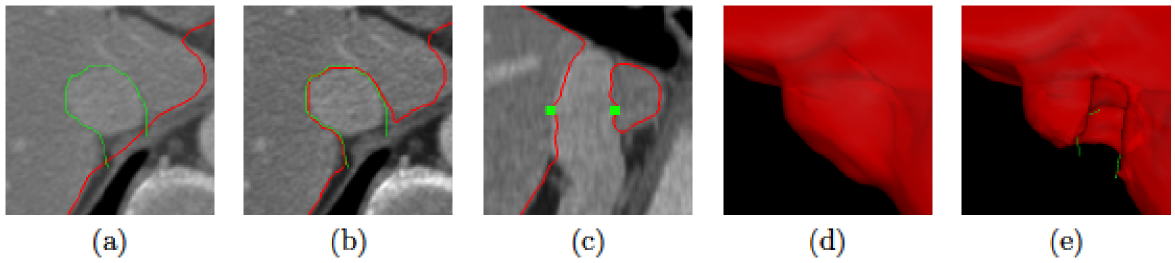


Figure 6.4: Medical-imaging application of using the RW method with a data prior. The IntellEditS tool [203] used data prior terms, based on interactive discriminative learning, to intelligently propagate a user's 2D edits to the larger 3D volume of a medical scan. For instance, in (a) the user wishes to correct a segmentation to remove the vena cava from a liver mesh (red). The user chooses an interaction plane and draws a corrected boundary (green). (b) The mesh is updated in the plane. (c) Using the RW and a data prior based on the learned features of the edited region, the mesh boundary is accurately propagated in 3D. (d)-(e) The 3D propagation can also be viewed by comparing before and after surfaces in (d) and (e), respectively. Figures taken from Harrison *et al.* [203].

The expression in (6.22) can be multiplied with (6.18) to form an expression of the derivative of $f(\phi)$ with respect to u_n:

$$\frac{\partial f(\phi)}{\partial u_{n'}} = \pi^{\mathrm{u}}_{k'\ell'n'} a_{k'\ell'k\ell} (\pi^{\mathrm{s}}_{k\ell m} s_m + \pi^{\mathrm{u}}_{k\ell n} u_n). \tag{6.23}$$

Setting the above to zero leads to a final expression, which can be solved for u_n:

$$\pi^{\mathrm{u}}_{k'\ell'n'} a_{k'\ell'k\ell} \pi^{\mathrm{u}}_{k\ell n} u_n = -\pi^{\mathrm{u}}_{k'\ell'n'} a_{k'\ell'k\ell} \pi^{\mathrm{s}}_{k\ell m} s_m. \tag{6.24}$$

The selection NTs can be seen as selecting certain sub-regions of the $a_{k'\ell'k\ell}$ NT.

6.2.3 Extensions

Extending the energy functional in (6.12) to 3 or more dimensions, which involves adding terms incorporating FD differences in the additional directions, is notationally straightforward under the NT framework. Additionally, the NT framework is particularly well-equipped to model an important refinement to the RW formulation, which adds a prior term that biases the solution based on some pre-existing knowledge. This was first discussed by Grady [213], leading to several other important works employing the RW-with-prior formulation [203, 214, 215]. This includes a tool developed by the author called intelligent editor of segmentations (IntellEditS) that uses the RW method coupled with two data-prior terms to edit pre-existing medical-imaging segmentations [203]. Figure 6.4 visually demonstrates the functionality of the tool.

The data prior's influence on the final solution is controlled by a set of weights. Notationally, this alters the formulation in (6.12) to include an additional additive term,

$$f(\phi) = (d^y_{ii'}\phi_{i'j})(d^y_{\underline{i}i'}\phi_{i'\underline{j}}w^y_{\underline{ij}}) + (d^x_{jj'}\phi_{ij'})(d^x_{\underline{j}j'}\phi_{\underline{i}j'}w^x_{\underline{ij}}) + \gamma_{ij}(\phi_{ij} - y_{ij})(\phi_{ij} - y_{ij}), \tag{6.25}$$

$$\text{s. t.} \begin{cases} \phi_{ij} = 1\,,\ \{i,j\} \in S_f \\ \phi_{ij} = 0\,,\ \{i,j\} \in S_b \end{cases}, \tag{6.26}$$

where y_{ij} represents the data prior terms and γ_{ij} represents the weighting that controls the data-prior's influence. The energy functional in (6.25) can be understood as penalizing both pixel labels that deviate from the data prior and pixel labels that deviate from their neighbours. For the IntellEditS tool, data prior terms were calculated based on learning the features that discriminate between foreground and background seed values [203].

Like the original RW method, the energy functional can be minimised by taking the derivative with respect to ϕ_{ij},

$$\frac{1}{2}\frac{\partial f(\phi)}{\partial \phi_{k'\ell'}} = d^y_{ik'}(d^y_{\underline{i}k}w^y_{\underline{i\ell}}\delta_{\underline{\ell}\ell'})\phi_{k\ell} + d^x_{j\ell'}(d^x_{\underline{j}\ell}w^x_{\underline{kj}}\delta_{\underline{k}k'})\phi_{k\ell} + \gamma_{ij}\delta_{ik'}\delta_{j\ell'}(\phi_{ij} - y_{ij}), \tag{6.27}$$

$$\begin{aligned} &= d^y_{ik'}(d^y_{\underline{i}k}w^y_{\underline{i\ell}}\delta_{\underline{\ell}\ell'})\phi_{k\ell} + 2d^x_{j\ell'}(d^x_{\underline{j}\ell}w^x_{\underline{kj}}\delta_{\underline{k}k'})\phi_{k\ell} \\ &\quad + (\gamma_{\underline{k\ell}}\delta_{\underline{k}k'}\delta_{\underline{\ell}\ell'})\phi_{k\ell} - 2\gamma_{ij}\delta_{ik'}\delta_{j\ell'}y_{ij}, \end{aligned} \tag{6.28}$$

$$= a_{k'\ell'k\ell}\phi_{k\ell} + f_{k'\ell'}, \tag{6.29}$$

where ϕ's dummy indices for the data-prior term in (6.28) were changed to k and ℓ. As (6.28) demonstrates, the data-prior adds weighted Kronecker delta NTs to the final $a_{k'\ell'k\ell}$ NT. In addition, a constant term, $f_{k'\ell'}$, is included within the formulation, which, when (6.29) is set to zero, turns the original Laplacian elliptic PDE into a Poisson-like PDE. With a formulation for (6.29) in hand, solving for unseeded labels follows the same process as before. Grady's original exposition, using the MV paradigm, relied on diagonal matrices to express entrywise products [213]. In addition, the steps involved to construct the final formulation of (6.29) is left to the reader. In contrast, using NT formalism, the solution to (6.25) is algebraically constructed, making the translation to a programmed solution much more seamless.

6.2.4 Results

Despite having to deal with fourth-degree data, entrywise products, ternary inner products, and high-degree linear inversions, it is possible to implement the RW method using the MV paradigm. In fact, Grady's original RW implementation uses MATLAB's MV routines [56]. For this reason it is possible to compare the computational and programming efficiency of a standard MV-paradigm MATLAB implementation vs. that of an NTToolbox implementation. For simplicity, hereafter the standard MV-paradigm MATLAB implementation will be referred to as the standard MATLAB implementation. The comparison will focus on the original RW formulation, and not the extensions discussed in Section 6.2.3.

Focusing first on programming efficiency, to implement the RW algorithm using a MATLAB implementation requires several hand-crafted workarounds, which mirrors the algebraic workarounds encountered in Grady's original exposition [56]. These include needing to flatten fourth-degree and second-degree data into matrix and vector form, respectively. Moreover, entrywise products must be executed using diagonal matrices. These operations introduce programming difficulties that are not inherent to the problem-at-hand. Åhlander *et al.* have emphasised that such workarounds can be a significant source of error [54, 67], which largely stem from the need to manually bookkeep the mappings from high-degree to low-degree form.

Apart from adding additional opportunities for errors, the flattening scheme itself introduces an arbitrary element to the solution. The programmer must settle upon one of several flattening schemes and ensure that the implications of this are understood and communicated to anyone viewing the code. In addition, by shoehorning the RW algorithm into the MV paradigm, the programmer cannot use algebraic operations. Instead her only recourse is to use hand-crafted workarounds, which come predominantly in the form of MATLAB-specific operations. These do not generalise to other technical computing environments, introducing obstacles to knowledge translation and uptake.

The end result of these issues is that the resulting MATLAB implementation is opaque to someone not already well-versed in MATLAB workarounds and the implications of these actions. In contrast, using NT algebra to express the RW algorithm allows all steps to be algebraically described. In addition to its expositional value, explicitly describing all steps eases translating the RW concepts to code, as the NT software can mimic NT algebra. This avoids introducing difficulties that are not inherent to the RW algorithm. Most importantly, the solution on paper aligns with the programmatic solution, resulting in a highly transparent implementation. Thus, the NTToolbox solution enjoys highly significant gains in programming efficiency.

Switching focus to computational efficiency, the RW method comprises two main steps. The first step constructs a high-degree linear system to solve. The second step solves it. Focusing on the second step first, the time taken by the MATLAB and NTToolbox implementations to solve the RW linear system is illustrated by Figure 6.5(b). These performance metrics compare the performance of Eigen's open-source sparse solver [145], used by NTToolbox, vs. MATLAB's sparse solver. As such, they are not measuring the impact of the NT algebra and software innovations detailed in this thesis, but instead the choice of the third-party sparse solver. From Figure 6.5(b), it can be seen that MATLAB's sparse solver outperforms Eigen's by considerable margins, highlighting the effectiveness of the former's routines. Future work should link to or include faster third-party sparse solvers, possibly replicating MATLAB's poly-algorithmic approach. Because effective open-source solutions exist, *e.g.*, the routines used in Octave's sparse solver [216], the performance gap seen in Figure 6.5(b) can be closed.

In contrast to the solution step, setting up the RW linear system involves the NT operations outlined in this thesis. Figure 6.5(a) illustrates the performance of the MATLAB and

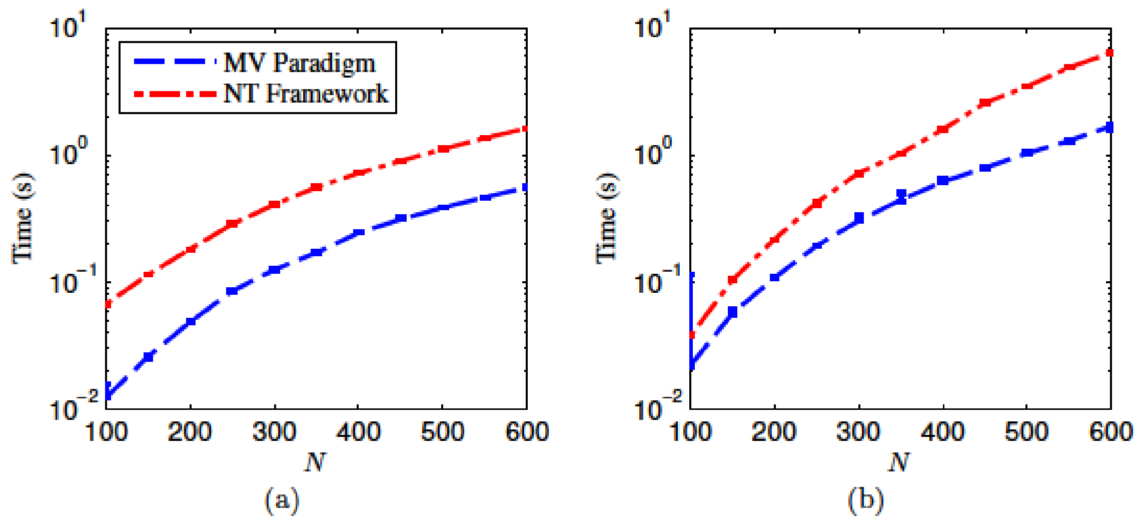


Figure 6.5: Runtime comparison of the MV paradigm and the NT framework in performing 2D RW image segmentation. The RW method was executed on randomly generated $N \times N$ images using foreground and background seed points corresponding to the first and last columns, respectively. Runtimes of a MATLAB MV paradigm implementation and of a NTToolbox implementation were measured across increasing values of N. Ten trials were executed for each value of N. Trend lines depict median values while error bars depict quartiles. The time taken to construct the RW linear system and then solve it are depicted in (a) and (b), respectively.

NTToolbox implementations. As can be seen, the MATLAB solution is significantly faster than NTToolbox. Profiling reveals that the gap between the two can be reduced by offloading some of NTToolbox's resolution of NT algebra to C++ MEX functions. Nonetheless, even with these changes, performance gaps will likely remain.

When assessing the performance results of Figure 6.5(a) one fact should be considered: using a standard MATLAB MV implementation forces programmers to use hand-crafted code in order to meet the high-degree and entrywise needs of the RW algorithm. The considerable disadvantages of hand-crafted code were documented above. Nonetheless, because abstraction penalties are avoided, hand-crafted code, if done well, typically results in highly efficient code. Indeed, the struggle to provide improved expressive capabilities while trying to match the performance of hand-crafted code is a reoccurring thread within technical computing. Two examples include the efforts of the developers of FORTRAN to match the performance of hand-crafted assembly code [24] and the later trail-blazing innovations of Landry's FTensor library, which is designed to provide support for Einstein notation while matching the performance of hand-crafted FORTRAN-style code [101]. Each step forward in programming efficiency typically results in a sacrifice in computational efficiency. So it is no surprise then that a hand-crafted MATLAB implementation performs faster than NTToolbox's algebraic solution. Likewise, a hand-crafted lower-level solution would be faster than a hand-crafted MATLAB implementation. Yet, this latter fact has not arrested

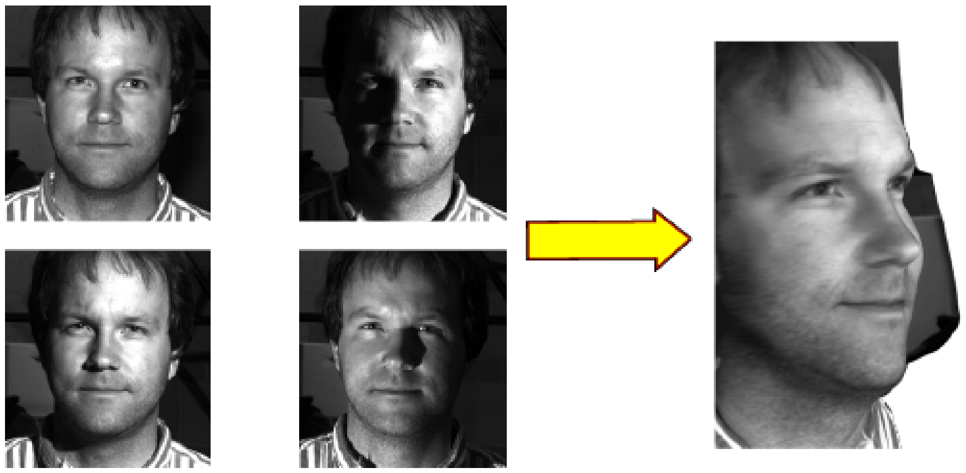


Figure 6.6: Illustration of depth-map & albedo estimation. Four images of a single-view image sequence are displayed. Each image corresponds to a distinct principal light direction, with its own shading characteristics. From the image sequence, an estimate of the depth of the subject's face, along with the albedo, can be produced. Image data obtained from the Extended Yale Face Database B [217].

MATLAB's rise in popularity as a technical computing platform.

Thus, these results reflect the frequent tension between programmatic and computational efficiency, which transcends the scope of NT computations. Should one accept that forcing practitioners to hand-craft technical computing code is a deleterious practice, then the faster speed of MATLAB's RW implementation compared to NTToolbox's should not be surprising or disturbing. A belief otherwise would not only undermine the rationale behind the NT software, but to varying extents, it would also undermine the rationale for FORTRAN all the way up to very high-level languages like MATLAB.

Moreover, the RW runtime is dominated not by the time needed to construct the linear system, but by the time needed to solve it. Interfacing to improved third-party sparse solvers will more closely align the overall performance of NTToolbox to that of a standard MATLAB implementation. Any remaining performance gaps will be a consequence of significantly higher, and frequently more impactful, programming efficiency.

6.3 Depth-Map & Albedo Estimation

Depth-map estimation is a seminal computer vision technique whose goal is to estimate the height field of an object from a sequence of 2D images captured from the same viewpoint, but each under a different light direction. Estimating depth this way cannot be done without also considering the object's reflectance characteristics. Assuming Lambertian reflectance, *i.e.*, a perfectly diffuse surface material, an object's reflectance is fully described by its albedo, or colour. Figure 6.6 visually illustrates depth-map & albedo estimation.

Closely related to depth-map & albedo estimation, and a method important for this

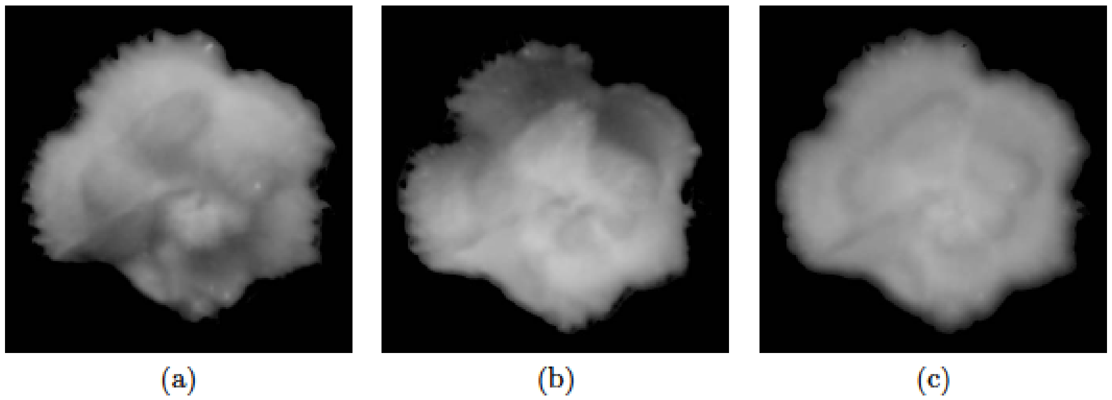


Figure 6.7: Using photometric stereo to generate unseen images from arbitrary light directions. Photometric stereo was performed on an image sequence of a microfossil specimen. The estimates were used to generate new images of the specimen. (a) and (b) depict new images under spot-light illumination, from a zenith angle of 30° and azimuths of 90° (north) and 270° (south) respectively. (c) depicts the same microfossil, illuminated with a ring light at a zenith angle of 30°. Figures taken from Harrison *et al.* [218].

exposition, is photometric stereo. Unlike the former, photometric stereo does not aim for an explicit estimate of an object's depth map, but rather a map of its surface normals. Photometric stereo is relevant to depth-map & albedo estimation, as if often acts as a first step for many of the latter's techniques. Amongst other applications, depth-map & albedo estimation and photometric stereo act as crucial components of the author's work on virtual reflected-light microscopy [218]. Both techniques can be used for image generation purposes, *i.e.*, generating new images of an object under arbitrary lighting conditions. The image-generation process of virtual reflected-light microscopy is illustrated by Figure 6.7.

Depth-map estimation is intimately connected to differential operations. As the problem is also characterised by high-degree mappings, entrywise products, linear inversions, and analytical derivatives, depth-map & albedo estimation serves as a compelling exemplar for the NT framework. We outline two exemplars stemming from depth-map & albedo estimation. Section 6.3.1 first outlines the traditional two-step approach to depth-map & albedo estimation, which requires accommodating five different indices. This is followed by Section 6.3.2, which outlines a direct, one-step, approach to depth-map & albedo estimation. This approach has superior information theoretic performance, but requires accommodating six different indices, further challenging the MV paradigm. When a separable representation for estimation is added to the mix, a seventh index comes into play.

6.3.1 Linearised Maximum Likelihood

Directly relating image observations to depth-map estimates requires a large-scale and non-linear generative model. As this poses many challenges, the traditional approach is to use a two-step process of first executing photometric stereo and then executing depth-map estimation from photometric stereo's surface-normal output. Classic two-step approaches enjoy a long pedigree and are recognised for their high efficiency. Yet, classic methods can fail catastrophically under noisy, real-world, conditions. Since there are good models of image noise, maximum likelihood (ML) approaches can in principle solve this problem.

However, it is not obvious how to obtain an ML estimate using the two-step approach, leading many practical methods to use non-robust heuristics to address image noise. Harrison and Joseph [55] solved this problem by modelling the propagation of noise distribution throughout the two steps. Since the method relies on linearised estimates of gradient noise, this two-step formulation can be called linearised maximum likelihood (LML) depth-map estimation.

The exposition will begin by explaining the initial photometric-stereo step. This is followed by an outline of the final depth-map estimation step. Finally, results will be given comparing an implementation using the MV paradigm vs. that of the NT framework. Effort will be taken to only highlight the details most salient to the NT framework. For details that are not related, such as the particulars of the image and gradient noise distributions, readers are encouraged to consult Harrison and Joseph [55].

Photometric Stereo

The first step in the LML method is to perform photometric stereo, which estimates the surface normals and the albedo of the object being imaged, but not the explicit depth. As this subsection will explain, the photometric-stereo model of image formation is used to perform this task. The output of photometric stereo can be used to generate images of the object under arbitrary lighting conditions. When the goal is to produce an ML estimate of the depth, a model of the noise distribution of surface gradients must be formulated. This model is expressed using high-degree matrix NTs and entrywise products.

When estimating surface normals and albedo, these parameters can be represented together as one vector called the weighted normals, $\boldsymbol{\eta} = (\eta^x, \eta^y, \eta^z)^\mathsf{T}$. This vector is composed of the unit-vector surface normals multiplied by the albedo, a gray-level intensity. Using this representation, Lambertian image formation for a known and single point light source, plus other important assumptions [55], is captured by a near-linear relationship, expressed as

$$I = u(\boldsymbol{\ell}^\mathsf{T}\boldsymbol{\eta})\boldsymbol{\ell}^\mathsf{T}\boldsymbol{\eta} + \epsilon, \tag{6.30}$$

where I is the pixel intensity and $\boldsymbol{\ell}$ represents the light direction. In essence the pixel intensity is directly related to the cosine of the angle between surface normals and the direction of the principle light source. The unit step function, $u(.)$, models attached shadows, setting image intensity to zero when the corresponding surface patch faces away from the light source. Image noise is represented by ϵ, which is modelled as an additive independent and identically distributed (IID) zero-mean Gaussian contribution, a reasonable assumption for

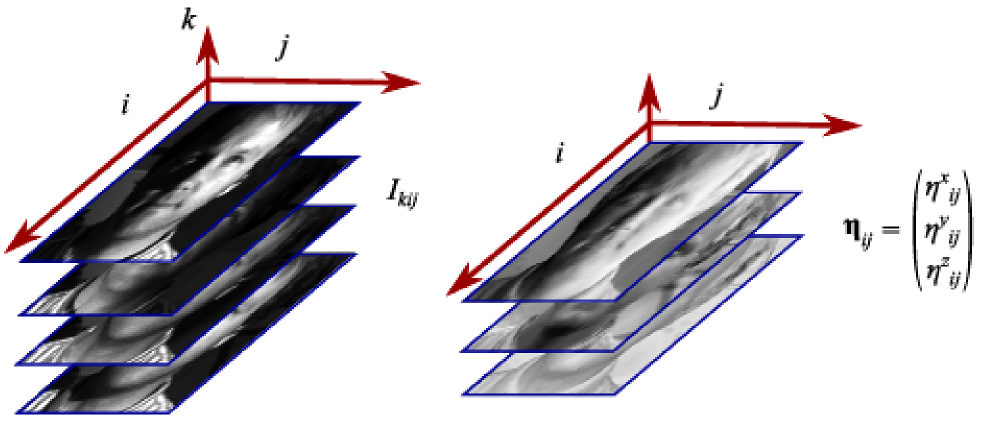

k
j
i
I_{kij}
j
i
$\boldsymbol{\eta}_{ij} = \begin{pmatrix} \eta^x_{ij} \\ \eta^y_{ij} \\ \eta^z_{ij} \end{pmatrix}$

Figure 6.8: The four indices of photometric stereo. In the image sequence, I_{kij}, each image corresponds to a light direction. The light directions are indexed by k. Light directions and weighted normals are represented by a vector NT, adding another index. Since photometric stereo is executed independently for each pixel location, *i.e.*, i and j, these last indices do not play an important role, practically limiting the number of indices to two.

image noise [219]. Since (6.30) only concerns one surface location, estimating the weighted normals can be executed independently for each pixel. Thus, each pixel location will be considered in isolation. This model of image formation can be called the *photometric-stereo model*.

Given N images, each illuminated under known directions, image formation of all observations can be expressed together as

$$I_k = u(\boldsymbol{\ell}_{\underline{k}}^{\mathsf{T}}\boldsymbol{\eta})\,\boldsymbol{\ell}_{\underline{k}}^{\mathsf{T}}\boldsymbol{\eta} + \epsilon_k, \tag{6.31}$$

where k indexes the different images and light directions. Figure 6.8 illustrates the indices needed for photometric stereo. A vector index is needed to represent the 3D light directions and surface normals. Technically, the pixel locations, i and j, also come into play. However, since photometric stereo can be executed independently for each pixel, for practical purposes there are are only two indices to contend with at one time.

Returning to the image formation model, if one can filter out the shadowed pixels the generative model reduces to a smaller linear one:

$$\tilde{I}_k = \tilde{\boldsymbol{\ell}_k}^{\mathsf{T}}\boldsymbol{\eta} + \epsilon_k, \tag{6.32}$$

where $\tilde{I}_k$ and $\tilde{\boldsymbol{\ell}}_k$ only constitute the non-shadowed pixels and light directions, respectively. The weighted normals can then be estimated by solving (6.32) in the least-squares sense for each pixel. More details can be found in Harrison and Joseph's paper, including a discussion of filtering tactics and how using the reduced linear regression form of (6.32) still keeps the result an ML estimate [55]. From this an estimate of the albedo, $\hat{\rho}$, can be obtained by taking the norm of $\hat{\boldsymbol{\eta}}$. With an estimate of the weighted normals in hand, images of the object from arbitrary light directions can be generated, *e.g.*, the virtual reflected-light microscopy images of Figure 6.7.

However, often the purpose is to explicitly estimate an object's depth map. While the surface normals do describe an object's shape, explicitly estimating the depth map requires computing the surface gradients. The surface-gradient estimates can be calculated from the weighted-normal estimates using the following relationship:

$$\hat{\boldsymbol{\beta}} = (\hat{p}, \hat{q})^{\mathsf{T}}, \tag{6.33}$$

$$= \left(-\frac{\hat{\eta}_x}{\hat{\eta}_z}, -\frac{\hat{\eta}_y}{\hat{\eta}_z}\right)^{\mathsf{T}}, \tag{6.34}$$

where p and q denote the surface gradients in the x and y direction, respectively.

To obtain an ML estimate, the noise distribution of the gradients must also be modelled. The error in the gradient estimates can be modelled based on the noise model of the original image observations. However, modelling this behaviour is challenging as its distribution is governed in part by the nonlinear operations within (6.34). Solving this problem, Harrison and Joseph demonstrated that one can model the gradient noise using an anisotropic Gaussian distribution, provided regularity assumptions and asymptotic approximations are used [55].

Harrison and Joseph also outlined how the covariance of the gradient estimates can be computed. These details will not be given here, but interested readers are encouraged to consult their paper [55]. Regardless of how the covariance is computed, since gradients are estimated independently for each pixel, they can be modelled as being independent from estimates stemming from other pixel locations. Thus, the covariance of the gradient estimates can be represented using a sequence of 2×2 matrices for each pixel location:

$$\boldsymbol{\Sigma}^{\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}} = \mathbf{M}. \tag{6.35}$$

When all pixels locations are considered together, the gradient covariance can be expressed as

$$\boldsymbol{\Sigma}^{\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}}_{iji'j'} = \mathbf{M}_{\underline{ij}}\delta_{\underline{i}i'}\delta_{\underline{j}j'}, \tag{6.36}$$

where i and j denote pixel locations and the use of δ NTs reflects that covariance between gradient estimates is nonzero only when they share the same pixel location. If the inverse of (6.36) need be obtained, it can be calculated relatively inexpensively using:

$${\boldsymbol{\Sigma}^{\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}}_{iji'j'}}^{-1} = \mathbf{M}_{\underline{ij}}^{-1}\delta_{\underline{i}i'}\delta_{\underline{j}j'}. \tag{6.37}$$

Depth-Map Estimation

With photometric stereo explained, depth-map estimation can be outlined, a setting where the NT framework can play a highly useful role. A depth map, $z_{k\ell}$, is related to surface gradients through the following formulation:

$$\boldsymbol{\beta}_{ij} = \left(\frac{\partial z_{ij}}{\partial x}, \frac{\partial z_{ij}}{\partial y}\right)^{\mathsf{T}}. \tag{6.38}$$

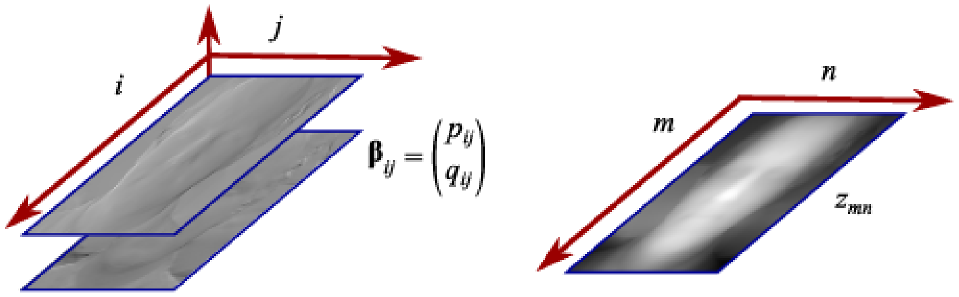

j
i
$\boldsymbol{\beta}_{ij} = \begin{pmatrix} p_{ij} \\ q_{ij} \end{pmatrix}$
n
m
z_{mn}

Figure 6.9: Five indices are required for LML depth-map estimation. Surface gradients for each pixel location are represented by a vector NT, $\boldsymbol{\beta}_{ij}$, constituting three indices when the vector index is included. Two other indices are needed to represent z_{mn}.

The relationship can be approximated using FD operators:

$$\boldsymbol{\beta}_{ij} = \left(d^{x}_{jn} z_{in}, d^{y}_{im} z_{mj}\right)^{\mathsf{T}}, \tag{6.39}$$

where as explained in Section 6.1, the $d^{x}_{jj'}$ and $d^{y}_{ii'}$ NTs represent FD operators in the x and y directions, respectively. Since FD operators use a neighbourhood of values in their calculations, an estimate for z_{mn} must be calculated for all pixel locations simultaneously. Figure 6.9 visually illustrates the indices needed for LML depth-map estimation.

When using an ML approach, estimating the depth map is framed as a regression problem:

$$\left(\begin{array}{c} d^{x}_{jn} z_{in} \\ d^{y}_{im} z_{mj} \end{array} \right) = \left(\begin{array}{c} p_{ij} \\ q_{ij} \end{array} \right) + \boldsymbol{\epsilon}^{\boldsymbol{\beta}}_{ij}, \tag{6.40}$$

which simplifies to

$$\mathbf{d}_{ijmn} z_{mn} = \boldsymbol{\beta}_{ij} + \boldsymbol{\epsilon}^{\boldsymbol{\beta}}_{ij}, \tag{6.41}$$

where

$$\mathbf{d}_{ijmn} = \left(\begin{array}{c} d^{x}_{jn} \delta_{im} \\ d^{x}_{im} \delta_{jn} \end{array} \right), \tag{6.42}$$

and $\boldsymbol{\epsilon}^{\boldsymbol{\beta}}_{ij} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}^{\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}}_{iji'j'})$ describes the gradient noise. In (6.41) and (6.42) z_{mn} is isolated using the factoring rules explained in Section 3.2.1. The expression in (6.41) incorporates the five indices outlined in Figure 6.9.

Since the gradient noise terms are not IID, (6.41) expresses a generalised least-squares (GLS) problem [220]. As explained by Harrison and Joseph [55], solving this large and sparse GLS problem directly is best accomplished using the least-squares normal equations:

$$\mathbf{d}^{\mathsf{T}}_{i'j'm'n'} (\boldsymbol{\Sigma}^{\boldsymbol{\beta}\boldsymbol{\beta}}_{i'j'ij})^{-1} \mathbf{d}_{ijmn} \hat{z}_{mn} = \mathbf{d}^{\mathsf{T}}_{i'j'm'n'} (\boldsymbol{\Sigma}^{\boldsymbol{\beta}\boldsymbol{\beta}}_{i'j'ij})^{-1} \boldsymbol{\beta}_{ij}, \tag{6.43}$$

which, when (6.37) is substituted in, equates to

$$\mathbf{d}^{\mathsf{T}}_{i'j'm'n'} (\mathbf{M}^{-1}_{\underline{i'j'}} \delta_{\underline{i'}i} \delta_{\underline{j'}j}) \mathbf{d}_{ijmn} \hat{z}_{mn} = \mathbf{d}^{\mathsf{T}}_{i'j'm'n'} (\mathbf{M}^{-1}_{\underline{i'j'}} \delta_{\underline{i'}i} \delta_{\underline{j'}j}) \boldsymbol{\beta}_{ij}, \tag{6.44}$$

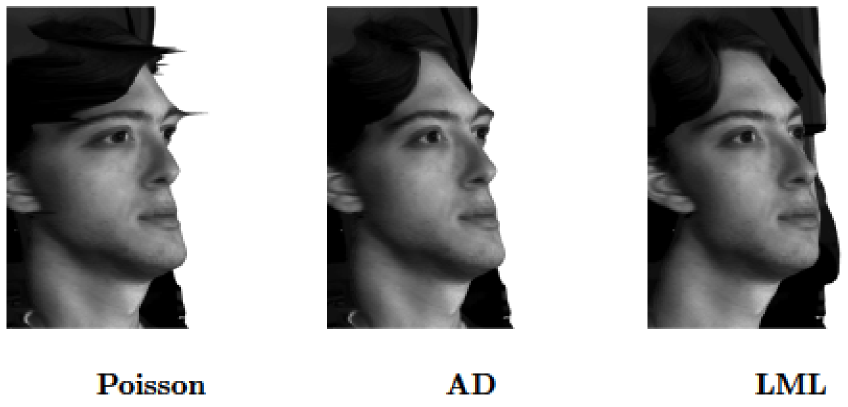


Figure 6.10: Visual improvements of the LML method over the state-of-the-art. The depth map estimates of the LML method is compared against the Poisson and anisotropic diffusion (AD) methods [221], which represent classic and more modern heuristic approaches, respectively. Image-sequence data used to estimate the depth-map & albedo corresponds to subject 24 from the Extended Yale Face Database B [217]. More details on the experiment setup can be found in Harrison and Joseph [55]

and simplifies to

$$\mathbf{d}^{\mathsf{T}}_{i'j'm'n'}\mathbf{M}^{-1}_{i'j'}\mathbf{d}_{i'j'mn}\hat{z}_{mn} = \mathbf{d}^{\mathsf{T}}_{i'j'm'n'}\mathbf{M}^{-1}_{i'j'}\boldsymbol{\beta}_{i'j'}. \tag{6.45}$$

The substitution of (6.37) into (6.44) introduces ternary inner products into the formulation. This also paves the way toward expressing the computationally simpler formulation of (6.45), as the δ NTs can be multiplied out, incorporating both FD operators into the ternary inner products.

The final formulation in (6.44) can be viewed as expressing a combinatorial Laplacian, which is identical in structure to, but different in value from, the RW formulation. Thus, depth-map estimation can be cast as an anisotropic Poisson elliptic PDE. As can be seen, LML depth-map estimation involves fourth-degree vector and matrix NTs, entrywise products, and ternary inner products. Even when resorting to flattening operations and using diagonal matrices to represent entrywise products, there is no obvious way to cast (6.45) into the MV paradigm, which is reflected by Harrison and Joseph's reliance on textual descriptions of the underlying algebraic operations [55]. For this reason, it is a problem well-suited to both NT algebra and NT software.

As Figure 6.10 illustrates, the effort taken to model the propagation of image noise throughout the two steps of LML estimation pays dividends. Significant visual improvements can be garnered and usable depth-map estimates are produced even when state-of-the-art methods fail. These qualitative observations are supported by extensive quantitative experiments [55].

Results

Like RW image segmentation, it is possible to implement LML estimation using standard MATLAB code. In fact, originally, the LML method was formulated using the MV paradigm [55] and the frustrations involved served as one of the motivations for our foray into NT algebra and software. As a result, just like with the RW algorithm, the programming and computational efficiency of the standard MATLAB implementation vs. an NTToolbox implementation can be gauged.

In our original exposition using MV algebra, steps had to be explained in words or by implicit flattening. Just like the RW method, these non-algebraic workarounds negatively impacted the programmatic implementation. Because the LML formulation is very similar to the RW formulation, the consequences to programming efficiency closely match. For instance, a standard MATLAB implementation required keeping track of lexicographical offsets and implementing ternary inner products by manually constructing a very-large and sparse diagonal matrix.

On the other hand, NT algebra can naturally formulate all steps. As a result, the NTToolbox implementation is able to mirror the solution on paper. Thus, the programmatic solution is expressed algebraically, providing a much more transparent implementation free of MATLAB-specific workarounds or other difficulties not inherent to the actual problem-at-hand.

Focusing on computational efficiency, like the RW algorithm, the LML method can be roughly divided into the two steps of first constructing a high-degree linear system and then solving it. Figure 6.11 illustrates the performance of the standard MATLAB and NTToolbox implementations for these two steps across increasing scales of data. Considering the time taken to solve the high-degree linear system first, as Figure 6.11(b) indicates, the MATLAB sparse solver proved much faster than Eigen's [145] sparse solver used by NTToolbox. As this mirrors the performance of the RW algorithm, this is not a surprising result. As mentioned in Section 6.2.4, interfacing to faster third-party sparse solvers will close this gap.

However, as Figure 6.11(b) illustrates, to construct the high-degree linear system, the NTToolbox implementation outperforms the standard MATLAB implementation by significant margins. The differences in performance are caused by the expression in (6.37), which calculates inverse entrywise covariance values. Because of the entrywise nature of (6.37), there are no fast MATLAB workarounds, forcing a standard implementation to resort to explicit `for` loops, which are a well-known bottleneck. In contrast, because the operation can be expressed algebraically using NT algebra, NTToolbox can offload the computation to LibNT's high-performance kernels, gaining considerable performance improvements.

Thus, unlike the RW algorithm, the LML method provides a scenario where even handcrafted MATLAB code offers no high-performance workarounds. This allows the NTToolbox code to construct the LML linear system much faster than the standard MATLAB imple-

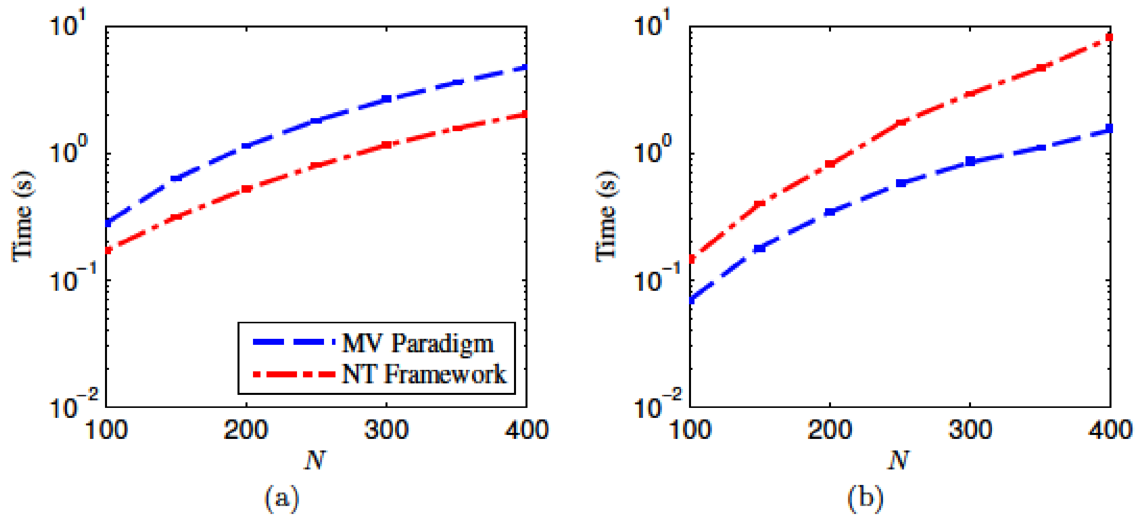


Figure 6.11: Runtime comparison of the MV paradigm and the NT framework in performing LML depth-map & albedo estimation. The LML method was executed on $N \times N$ images generated from Durou *et al.*'s vase surface [222]. Six images were used, each generated from its own light direction. Runtimes of a MATLAB MV paradigm implementation and of an NTToolbox implementation were measured across increasing values of N. Ten trials were executed for each value of N. Trend lines depict median values while error bars depict quartiles. The time taken to construct the LML linear system and then solve it are depicted in (a) and (b), respectively.

mentation. As these gains are accompanied by high programming efficiency, these results help demonstrate the power and utility of NT algebra and software.

6.3.2 Nonlinear Maximum Likelihood

Section 6.3.1 described how to produce an ML estimate using the two-step LML method. While this model demonstrates impressive robustness, even under very noisy conditions, improvements are possible. Realising these improvements is the topic of the author's followup work, embodied by a method called nonlinear maximum likelihood (NML) estimation [164]. This approach to depth-map & albedo estimation involves a large-scale and nonlinear formulation. The subsection begins by sketching the rationale for pursuing the NML approach. Afterwards, the process by which this problem can be solved, and the NT framework's role within the process, is explained.

Conceptual Overview

The LML method can be improved in several important ways. This subsection will highlight the possible improvements, motivating the use of a more parsimonious model of image formation called the depth-map & albedo model. The merits and challenges associated with this model are emphasised, underscoring the benefits of using the NT framework.

The motivation for pursuing improvements to the LML method stem from several issues. First, the two-step approach decouples albedo and depth-map estimates from each other. Ideally, reflectance should be not decoupled from depth-map estimation. Secondly, the LML method used asymptotic approximations to model gradient behaviour. These approximations are only valid under very stringent regularity conditions [223]. Moreover, even if the problem satisfies these conditions, sample sizes must typically be very large for these asymptotic approximations to take hold [223].

The third, and perhaps most important limitation, is due to overfitting. As the number of images approaches photometric stereo's three-image minimum, the propensities to overfit to image noise will increase. Moreover, it becomes less and less possible to filter out attached shadows. Once shadowed pixels are excluded in the linear regression sub-problem of (6.32), photometric stereo may no longer produce an ML estimate and may exhibit major errors. When the goal is to generate images of the object from arbitrary light directions, *e.g.*, for virtual reflected-light microscopy [218], overfitting will reduce accuracy. Moreover, these errors will propagate to the LML method's depth-map estimates, which rely directly on the photometric stereo output.

This last issue touches directly upon the concept of model parsimony. Simply put, the "best" model is not necessarily the one that maximises likelihood, but instead one that best balances the tradeoff between goodness of fit and complexity [224, 225]. Complexity can be thought of in many ways, but an important component is the number of parameters in the model. This is an information-theoretic approach to model selection, which can be seen as an extension to likelihood theory [224] and is exemplified most famously by the Akaike framework [226]. Incorporating parsimony within model selection helps address the issue of overfitting.

Thus, a more parsimonious model of image formation may well outperform the photometric-stereo model of (6.30), especially when observation counts are low. Such a model, called the *depth-map & albedo model*, is readily available. The model employs an augmented version of the vector NT $\boldsymbol{\beta}$ seen in the LML method,

$$\tilde{\boldsymbol{\beta}} = (p, q, 1)^{\mathsf{T}}, \tag{6.46}$$

$$= \left(\frac{\partial z}{\partial x}, \frac{\partial z}{\partial x}, 1\right)^{\mathsf{T}}. \tag{6.47}$$

Lambertian image formation can then be expressed as

$$I = \frac{\rho}{||\tilde{\boldsymbol{\beta}}||} u(\boldsymbol{\ell}^{\mathsf{T}}\tilde{\boldsymbol{\beta}})\boldsymbol{\ell}^{\mathsf{T}}\tilde{\boldsymbol{\beta}} + \epsilon, \tag{6.48}$$

which is composed of two parameters, ρ and z. The parameter count is a third less than the 3×1 weighted-normals vector, $\boldsymbol{\eta}$, seen in the photometric-stereo model of (6.30).

In practical applications, modelling the partial derivatives of the depth map requires employing FD operators. This connects neighbouring values of z together, meaning that

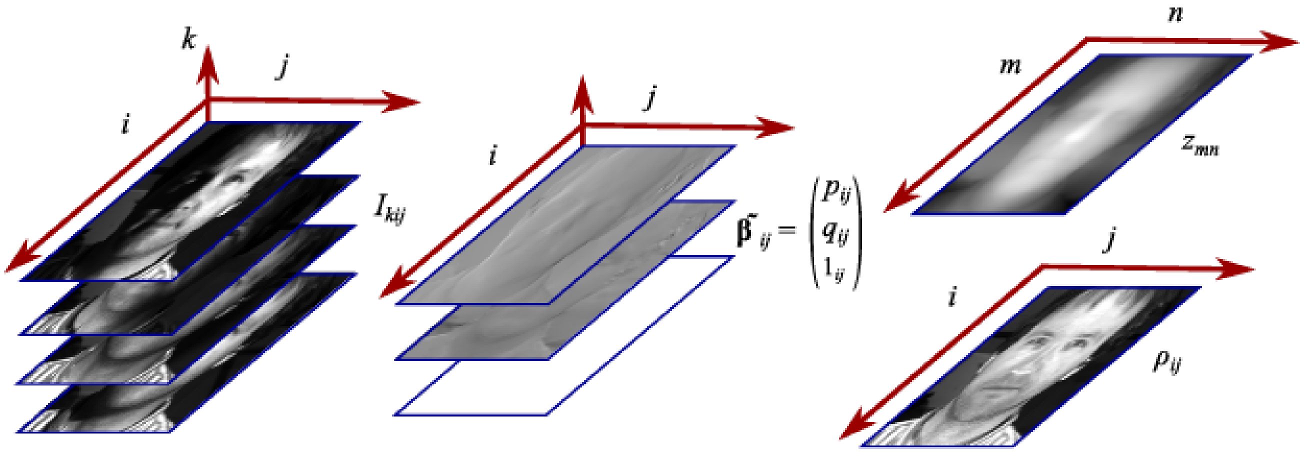

k
j
i
I_{kij}
j
i
$\tilde{\boldsymbol{\beta}}_{ij} = \begin{pmatrix} p_{ij} \\ q_{ij} \\ 1_{ij} \end{pmatrix}$
n
m
z_{mn}
j
i
ρ_{ij}

Figure 6.12: The six indices involved for NML estimation. Directly estimating the depth-map and albedo from image observations combines the indices seen in Figure 6.8 and Figure 6.9 into one formulation. As before k indexes each light direction and image observation and i and j index individual pixels. The vector needed to represent light directions and the $\tilde{\boldsymbol{\beta}}_{ij}$ vector NT accounts for an additional index. Finally, m and n index individual locations within the depth map.

one cannot consider instances of $\tilde{\boldsymbol{\beta}}$ in isolation. Instead, one must capture a neighbourhood of z values corresponding to the FD stencil:

$$\tilde{\boldsymbol{\beta}}_{ij} = \left(d^x_{jj'} z_{ij'}, d^y_{ii'} z_{i'j}, 1_{ij}\right)^{\mathsf{T}}, \tag{6.49}$$

where $d^x_{jj'}$ and $d^y_{ii'}$ are the same FD operators used in the previous sections. Thus, the challenge of the depth-map & albedo model is that all values of z must be considered at the same time, turning the estimation task into a large-scale nonlinear problem. When tied together with the sequence of image observations, the complete model can be expressed as

$$I_{kij} = \frac{\rho_{\underline{ij}}}{\sqrt{\tilde{\boldsymbol{\beta}}^{\mathsf{T}}_{\underline{ij}} \tilde{\boldsymbol{\beta}}_{\underline{ij}}}} u(\boldsymbol{\ell}^{\mathsf{T}}_{\underline{k}} \tilde{\boldsymbol{\beta}}_{\underline{ij}}) \boldsymbol{\ell}^{\mathsf{T}}_{\underline{k}} \tilde{\boldsymbol{\beta}}_{\underline{ij}} + \epsilon_{kij}, \tag{6.50}$$

$$= a(z_{mn})_{\underline{kij}} \rho_{\underline{ij}} + \epsilon_{kij}. \tag{6.51}$$

The challenges associated with the depth-map & albedo model are intensified by the large number of indices involved, which as revealed by (6.50) totals six, when vector indices and the dependance on z_{mn} are included. Figure 6.12 visually illustrates the indices involved.

Direct Estimation

Estimating the depth-map and albedo directly from image observations can be labelled nonlinear maximum likelihood (NML) estimation. The theoretical benefits of estimating depth maps directly has been recognised with varying levels of rigour in the literature [227, 228, 229, 230, 231], but Harrison and Joseph were the first to motivate the approach based on the information-theoretic principles of the Akaike framework [164]. Moreover, no method in the state of the art successfully coupled depth-map with albedo estimation. Harrison and Joseph demonstrated how this can be done, casting the problem as an SNLS problem. This subsection will outline the NML estimation process using the NT framework. As part of this outline, the complexities of managing the myriad indices will be stressed. The benefits of executing NML will be qualitatively demonstrated using visual examples.

Based on the IID Gaussian noise used in the generative model, the ML estimates of ρ_{ij} and z_{mn} correspond to the values that minimise the sum-squared error (SSE) of the residuals:

$$SSE = r_{kij} r_{kij}, \tag{6.52}$$

$$r_{kij} = I_{ijk} - a_{k\underline{ij}}\rho_{\underline{ij}}, \tag{6.53}$$

where a_{kij} depends on z_{mn}. This estimation task can be reduced by observing that the generative model in (6.51) expresses the fact that the formulation can be broken up into linear and nonlinear portions, where z_{mn} must be estimated nonlinearly. Given a z_{mn} estimate, the albedo estimate can always be recovered independently for each pixel location by solving an independent sequence of very small linear least-squares problems:

$$\hat{\rho}_{ij} = a^{+}_{k\underline{ij}} I_{k\underline{ij}}, \tag{6.54}$$

As such, this is an SNLS problem, and (6.54) can be substituted into (6.53) to produce a residual dependant upon only z_{mn}:

$$\begin{aligned} \tilde{r}_{kij} &= I_{kij} - a_{k\underline{ij}} a^{+}_{\ell\underline{ij}} I_{\ell\underline{ij}}, && (6.55)\\ &= I_{kij} - a^{p}_{k\ell\underline{ij}} I_{\ell\underline{ij}}, && (6.56)\\ &= (\delta_{k\ell} 1_{\underline{ij}} - a^{p}_{k\ell\underline{ij}}) I_{\ell\underline{ij}}, && (6.57)\\ &= a^{\perp}_{k\ell\underline{ij}} I_{\ell\underline{ij}}, && (6.58) \end{aligned}$$

where $a^{p}_{k\ell ij}$ and $a^{\perp}_{k\ell ij}$ denote the projection operator of a_{kij} and its complement, respectively. Note that the SNLS formulation throws an additional index, *i.e.*, ℓ in (6.58), into the mix. Eliminating ρ_{ij} from the residual reduces the estimation task to one that minimises an SSE that only depends on z_{mn}, *i.e.*,

$$SSE = \tilde{r}_{kij} \tilde{r}_{kij}. \tag{6.59}$$

Since the residual is nonlinear, minimising the SSE requires an initial estimate and a large-scale nonlinear least-squares routine. The former requirement can be satisfied with robust two-stage linear techniques, such as the LML method [55]. For the latter requirement, typical optimisation methods require partial derivatives of the residual with every free parameter. Using MV formalism would make this is a particularly onerous and error-prone task for several reasons. For one, the residual itself incorporates both high-degree data and entrywise products. Additionally, the residual incorporates a projection operator.

As explained in Section 3.3.3, even when the projection operator is a simple second-degree construct, its partial derivative is complex and involves third-degree NTs. In the NML case, the operator in (6.58) is a fourth-degree NT dependant on a second-degree NT, whose projection action involves entrywise products. Consequently, the NT framework is especially helpful in expressing and computing the partial derivatives of (6.58).

Formally, the partial derivative of (6.58) is

$$\frac{\partial \tilde{r}_{kij}}{\partial z_{mn}} = -\frac{\partial a^{p}_{k\ell\underline{ij}}}{\partial z_{mn}} I_{\ell\underline{ij}}, \tag{6.60}$$

where the partial derivative of the projection operator can be expressed using Golub and Pereyra's technique for differentiation of pseudo-inverses [165], which Section 3.3.3 outlined using NT algebra. Note that in this case, the formulation includes entrywise products and results in a sixth-degree NT:

$$\frac{\partial a^{p}_{k\ell\underline{ij}}}{\partial z_{mn}} = a^{\perp}_{kr\underline{ij}} \frac{\partial a_{r\underline{ij}}}{\partial z_{mn}} a^{+}_{\ell\underline{ij}} + a^{\perp}_{\ell r\underline{ij}} \frac{\partial a_{r\underline{ij}}}{\partial z_{mn}} a^{+}_{k\underline{ij}}. \tag{6.61}$$

The crux of (6.61) is calculating the partial derivative of a_{rij}. As (6.50) indicates, a_{rij} incorporates a unit-step function to model attached shadows. The derivative of a unit-step function is an impulse function, which is only non-zero when its argument is *exactly* zero. As this is unlikely, and for the sake of simplicity, when calculating the partial derivatives of a_{rij} the derivative of the unit step will not be considered. With this simplification in place, the first step toward formulating the partial derivative of a_{rij} is to formulate the partial derivative of $\tilde{\boldsymbol{\beta}}_{ij}$,

$$\frac{\partial \tilde{\boldsymbol{\beta}}_{ij}}{\partial z_{mn}} = \begin{pmatrix} -d^{x}_{jj'}\delta_{im}\delta_{j'n} \\ -d^{y}_{ii'}\delta_{i'm}\delta_{jn} \\ 0 \end{pmatrix}, \tag{6.62}$$

$$= \begin{pmatrix} -d^{x}_{jn}\delta_{im} \\ -d^{y}_{im}\delta_{jn} \\ 0 \end{pmatrix}, \tag{6.63}$$

$$= \dot{\boldsymbol{\beta}}_{ijmn}, \tag{6.64}$$

which is a fourth-degree vector NT.

The partial derivative of a_{rij} can then be formulated using the product rule,

$$\frac{\partial a_{r\underline{ij}}}{\partial z_{mn}} = -\frac{\tilde{\boldsymbol{\beta}}^{\mathsf{T}}_{\underline{ij}} \dot{\boldsymbol{\beta}}_{\underline{ij}mn}}{\left(\tilde{\boldsymbol{\beta}}^{\mathsf{T}}_{\underline{ij}} \tilde{\boldsymbol{\beta}}_{\underline{ij}}\right)^{3/2}} u(\boldsymbol{\ell}^{\mathsf{T}}_{\underline{r}} \tilde{\boldsymbol{\beta}}_{\underline{ij}}) \boldsymbol{\ell}^{\mathsf{T}}_{\underline{r}} \tilde{\boldsymbol{\beta}}_{\underline{ij}} + \frac{1_{\underline{ij}}}{\sqrt{\tilde{\boldsymbol{\beta}}^{\mathsf{T}}_{\underline{ij}} \tilde{\boldsymbol{\beta}}_{\underline{ij}}}} u(\boldsymbol{\ell}^{\mathsf{T}}_{\underline{r}} \tilde{\boldsymbol{\beta}}_{\underline{ij}}) \boldsymbol{\ell}^{\mathsf{T}}_{\underline{r}} \dot{\boldsymbol{\beta}}_{\underline{ij}mn}, \tag{6.65}$$

$$= \left(-\frac{\boldsymbol{\ell}^{\mathsf{T}}_{\underline{r}} \tilde{\boldsymbol{\beta}}_{\underline{ij}}}{\left(\tilde{\boldsymbol{\beta}}^{\mathsf{T}}_{\underline{ij}} \tilde{\boldsymbol{\beta}}_{\underline{ij}}\right)^{3/2}} \tilde{\boldsymbol{\beta}}^{\mathsf{T}}_{\underline{ij}} + \frac{1_{\underline{ij}}}{\sqrt{\tilde{\boldsymbol{\beta}}^{\mathsf{T}}_{\underline{ij}} \tilde{\boldsymbol{\beta}}_{\underline{ij}}}} \boldsymbol{\ell}^{\mathsf{T}}_{\underline{r}} \right) u(\boldsymbol{\ell}^{\mathsf{T}}_{\underline{r}} \tilde{\boldsymbol{\beta}}_{\underline{ij}}) \dot{\boldsymbol{\beta}}_{\underline{ij}mn} \tag{6.66}$$

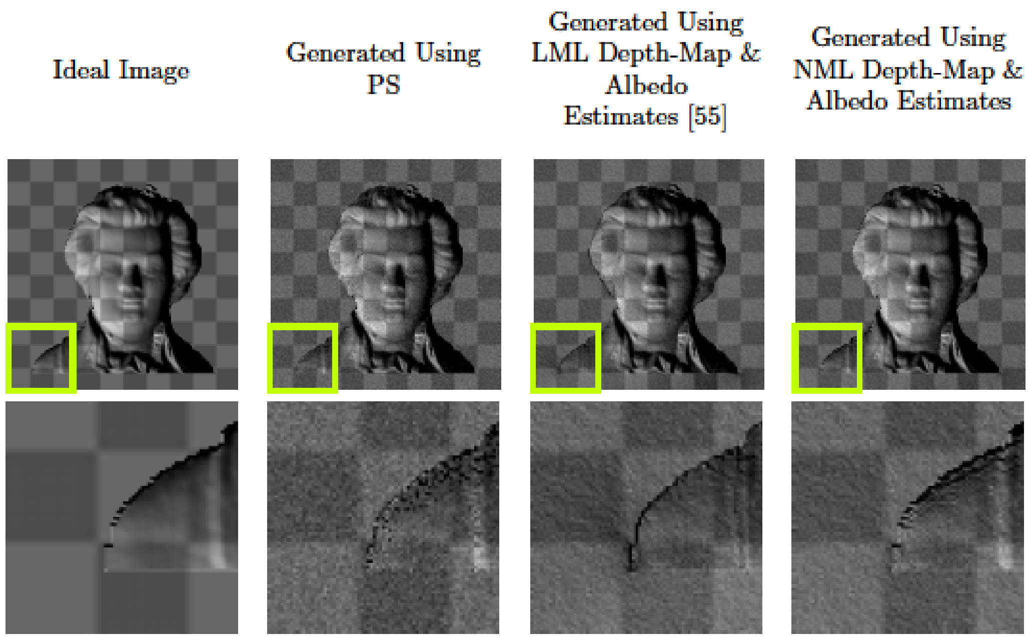

Ideal Image
Generated Using PS
Generated Using LML Depth-Map & Albedo Estimates [55]
Generated Using NML Depth-Map & Albedo Estimates

Figure 6.13: The benefits of the NML method for image generation. The output of the photometric-stereo, LML, and NML methods was used to generate an image from an *unseen* light direction under conditions with low observation counts. The generated images can be compared with the ideal and noiseless version, with the NML method exhibiting the best results. More details on the constructions of these figures can be found in Harrison and Joseph [164].

With an expression for the partial derivative of a_{rij} in hand, the partial derivative of the projection operator in (6.61) can be formulated. This can then be substituted into (6.60) to produce the partial derivative of the reduced residuals.

The NML method was able to successfully estimate depth map and albedos from simulated images. Using extensive quantitative experiments Harrison and Joseph demonstrated significant improvements in image generation and depth-map estimation [164]. These improvements can be visually demonstrated.

For instance, Figure 6.13 depicts an illustrative example involving image generation. The Lambertian image formation model was used to generate four noisy images of Zhang *et al.*'s Mozart surface [232] along with a checkerboard albedo. After images were generated, photometric stereo was then executed to estimate the weighted normals from the noisy images. As the second column of Figure 6.13 demonstrates, when generating an image originating from a light direction *not used* in the observations, photometric stereo produces a very noisy result that suffers from speckling. As well, image details, such as the border between Mozart and the background, are obscured. The fact that photometric stereo faces difficulties is not surprising, as the four observation images used in the example do not allow photometric stereo very much leeway to mitigate noise effects and detect attached shadows.

A natural question is whether the more parsimonious depth-map & albedo model can

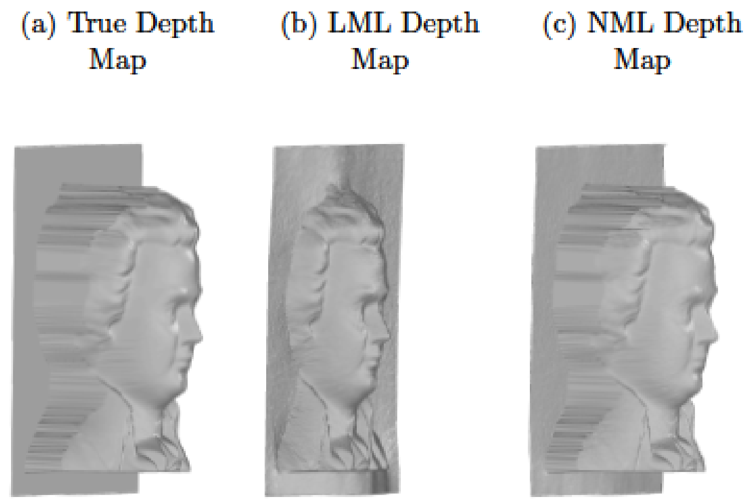


Figure 6.14: The benefits of the NML method for depth-map estimation. Using the experimental conditions of Figure 6.13, both LML and NML estimation were performed. Differences in depth-map quality between the LML and NML methods can be significant. Figures taken from Harrison and Joseph [164].

generate superior images. However, if the means to produce the depth-map estimate relies solely on the photometric-stereo output, image generation will face continued problems. As the third column of Figure 6.13 demonstrates, which depicts images generated using LML method, this is indeed the case. In general, while the LML method is able to generate images with less noise than photometric stereo, it is still unable to capture certain fine details. For instance, the extent of the raised ridge in the inset, corresponding to a wrinkle in Mozarts shirt, is made clear in the ideal image by lighter-coloured highlights. This is not captured by the LML method. On the other hand, the NML method can produce image reconstructions that capture these details, while still enjoying the benefits of a more parsimonious model, *i.e.*, exhibiting less susceptibility to noise. This illustrates the merits of the NML method for image generation tasks.

The NML method can also provide benefits to depth-map estimation. Figure 6.14(b) illustrates how the errors inherent in photometric stereo can affect LML depth-map estimation. As the figure demonstrates, the LML method suffers from localised noisy effects. Moreover, the depth map suffers from global distortions, including a flattened look. Thus, if the goal is depth-map estimation, a technique other than one based on photometric stereo should be adopted in these conditions. As Figure 6.14(c) demonstrates, the NML technique can fill this role, producing a depth-map estimate higher in quality than the LML method.

The NML method in Harrison and Joseph's publication [164] was implemented within the NT framework using the NTToolbox. The residuals and their partial derivatives were calculated using the NT framework and inputted into MATLAB's nonlinear least-squares routines. Implementing this using the MV paradigm was not even attempted, due to the plethora of indices, entrywise products, and complicated linear inversions involved. As such, the NT framework acted as a crucial tool in tackling this exemplar that lies beyond the MV paradigm.

6.4 Summary

Differential operators are an important concept within science and engineering. In computer vision contexts, differential operators manifest commonly as FD operators. The high-degree nature of FD operators acting upon images, combined with frequent needs to incorporate entrywise operators, differentiations, and linear inversions, contributes to the complexity of related computer vision tasks. Tackling this complexity head-on demands sophisticated tools.

As demonstrated by three exemplars arising from two computer vision exemplars, the NT framework excels in such settings. The RW algorithm represented one exemplar, whose formulation can be framed as a high-degree anisotropic Laplacian or Poisson elliptic PDE over a 2D or 3D domain. Formulating the fourth-degree anisotropic Laplacian operator requires entrywise products and ternary inner products. Important extensions to the RW algorithm, *e.g.*, adding a data prior, add further complications involving entrywise products. Unlike the MV paradigm, the NT framework can naturally model these challenging aspects of the RW algorithm.

Depth-map & albedo estimation represented the source of the other two exemplars. First, the LML method, which follows the traditional two-step approach, was outlined. Like the RW algorithm, the LML method can be modelled as an anisotropic elliptic PDE applied over a 2D domain, underscoring the frequent commonalities between computer vision work involving differential operators. The chapter also examined a recent enhancement, called the NML method, that eschews the two-step approach and instead nonlinearly estimates depth-map and albedos from image observations. This involved non-linear high-degree equations, partial derivatives, linear inversions, and separable representations. All told, this involves six natural indices to keep track of, with a seventh added when the separable representation is included. The NT framework's natural abilities to accommodate high-degree data, entrywise products, and linear inversions are integral toward a ready formulation and implementation of the NML method.

Both the RW and LML methods were programmed using a standard MV MATLAB implementation and an NTToolbox implementation. Comparisons between the two revealed that the NT software should incorporate a faster third-party linear sparse solver that can match the performance of MATLAB's equivalent routines. Once faster linear sparse solvers are included within the NT software suite, overall computational efficiency will be significantly improved. When comparing the computational efficiency of constructing the linear system, which incorporates the NT algebra and software innovations described in this thesis, performance metrics revealed that the NTToolbox was slower for the RW method but faster for the LML method. Moreover, regardless of any differences in computational efficiency, the NTToolbox implementations revealed considerable improvements in programming efficiency, as the hand-crafted and error-prone workarounds needed for the standard MATLAB implementation were avoided. Considering the heavy emphasis placed on programming efficiency within general-purpose [133,136] and technical computing [134,135] settings, these outcomes demonstrate the benefits of the NT software in tackling these exemplars.

The exemplars highlighted by this chapter serve as important vehicles to showcase and disseminate the merits of the NT framework for computer vision applications involving differential operators, of which there are legion. Moreover, the arguments outlined here are not limited to computer vision. As differential operators, and PDEs, play an enormous role within multitudinous other scientific and engineering fields, these exemplars can also act as persuasive agents in translating the gains realised by computer vision to other domains.

Chapter 7

Toward a New Paradigm

The impact of technical computing can be felt in almost every corner of modern scientific and engineering practice. Yet, technical computing is a scientific and engineering discipline in its own right, deserving of serious analysis regarding its structure, limitations, and progression. Such an analysis is especially topical today, where there is an increasing amount of research into decompositions, computations, equation solving, data analysis, and scientific problems that do not fit into the traditional matrix-vector (MV) mould that has so dominated technical computing's history and practice up to this point.

Considered together, these research thrusts signify that technical computing's prevailing mode of practice is being challenged by new research questions. This work is a serious articulation of this argument, using Kuhn's theory of paradigms as the kernel by which to understand the structure of technical computing and the increasing number of research efforts that are breaking the MV mould. This analysis informs our own contribution to this research movement, represented by the numeric tensor (NT) framework of this thesis.

Here, we summarise the contributions of this work, which encompass a highly diverse set of topics, ranging from the structure of technical computing to algebra to software and finally to exemplar problems. These contributions, which are significant to the ongoing evolution of technical computing, also lead to additional exciting research questions. We focus on many of these as promising avenues of future work. Stepping back, final remarks provide a reflection on the state of technical computing, arguing that the discipline is undergoing a revisionary period, as defined by Kuhn. We then conclude by offering our own speculations on how the discipline will evolve and share our own vision for technical computing's future.

7.1 Contributions

According to Kuhn, fully understanding the scope and limits of a domain's practice requires identifying the prevailing paradigm. Doing so reveals a taxonomy of exemplars and anomalies. Advocating for change requires developing components of a new disciplinary matrix that demonstrate enough promise to resolve current anomalies. These are the aims of this work, which provides a unique and timely interpretation of the discipline of technical computing and offers fundamental algebraic and software innovations, represented by the NT framework, for work beyond the prevailing paradigm. To highlight and emphasise the merits of the NT framework, we also discuss and advance several exemplar problems, with particular focus on problems drawn from computer vision.

7.1.1 Synthesis

There is a growing crowd of researchers developing, advocating for, or employing algebra and software beyond MV formalism and computations. We view these efforts as fundamental challenges to technical computing's prevailing mode of practice. To properly articulate this argument, it is incumbent to draw upon the views of those who have examined what exactly defines a fundamental change within a scientific domain. To the best of our knowledge, we are the first to apply Kuhn's theory of paradigms to the discipline of technical computing.

Concepts originating from Kuhn's writings, such as paradigm and terms that follow in its orbit, *e.g.*, revolution and crisis, are occasionally used carelessly. However, when used with care, Kuhn's writings offer a powerful lens by which to view the structure of a scientific domain. For our purposes, critically examining technical computing illuminates the discipline and also the current push for new algebra and software.

The structure of technical computing can be clarified by recognising that a paradigm defines the scientific questions a discipline investigates. Technical computing, made up of mathematical and computational ingredients, is a mix of the practical and analytical. These two aspects can be likened to the body and soul of technical computing. Algebra acts as its soul, serving as the language by which scientific problems are modelled and solutions to them are developed. As such, they are not just a component of the disciplinary matrix. Instead, algebras are integral in defining technical computing's exemplars and anomalies. Software acts as its body, defining what *can* be practically computed and whether prospective solutions are even possible. As such, software also defines the limits of technical-computing's practice. Thus, on their own neither body nor soul fully defines the taxonomy of scientific computing. However, together algebra and software can form a framework that does.

As we argue, the dominant framework within technical computing, currently the MV framework, defines its paradigm. These arguments are bolstered through our examination of the history of technical computing, where the MV framework plays a dominant role in the development of the discipline. Since the natural capabilities of the MV paradigm are well-defined, *i.e.*, linear mappings applied to vectors, its anomalies can be articulated. We focus on the two anomaly categories of special linear mappings, *e.g.*, high-degree linear mappings and entrywise products, and mappings beyond linear, *e.g.*, multilinear and polynomial mappings. Both these categories can be linked together by recognising that they each involve expressions and computations with high-degree data, *i.e.*, high-dimensional data.

Categorising anomalies together unifies significant prior work on high-degree algebras and softwares that previously have not engaged in much crosstalk. As Figure 7.1 illustrates,

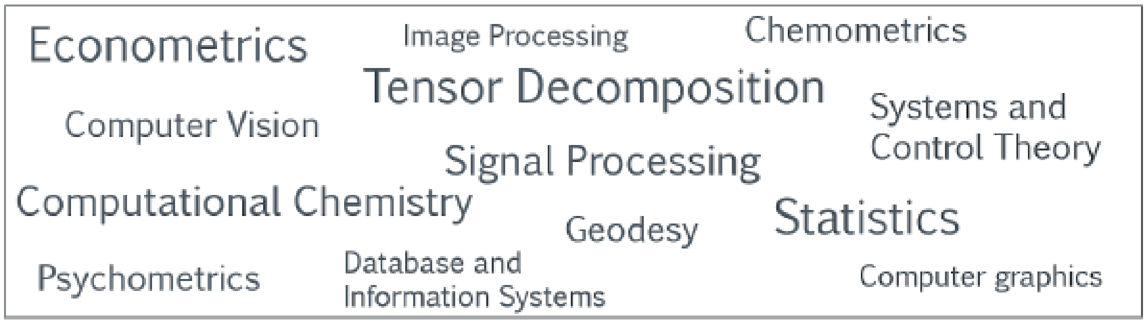


Figure 7.1: The growing crowd of work on high-degree algebras and softwares. Research efforts to address the MV paradigm's anomalies originates from a wide range of fields. In many cases, these are isolated and independent efforts.

these efforts stem from a diverse range of fields. We place these diverse efforts within a larger and combined context, in which we argue that these efforts all endeavour to address anomalies to the MV paradigm. We therefore link this varied body of work together and are the first to compile their algebraic and software characteristics together in one place. A technical computing framework able to unify work beyond the MV paradigm must capture these characteristics. This synthesis informs our subsequent efforts in developing and detailing the NT framework.

Finally Kuhn asserts that the pursuit of anomalies gives rise to new investigatory questions. Indeed, those pioneering frameworks beyond the MV paradigm have posed and investigated numerous algebraic and software questions. In developing the NT framework, we have also encountered new questions. These include algebraic ones, *e.g.*, what is the best way to represent the rich possibilities of arithmetic operations upon high-degree data without sacrificing ease of use? They also include new software questions, *e.g.*, how to efficiently execute or invert arbitrary products across high-degree data, how to best permute dense and sparse high-degree data, and what is the best means to manage hyper-sparsity inherent in sparse products? We offer answers to these questions, summarised in the sections below. However, like Kuhn, we emphasise the inherent value of raising these questions in the first place and welcome different answers, or, even better, additional questions.

7.1.2 Algebra

Sir Alfred North Whitehead, the nineteenth and twentieth-century mathematician and philosopher, offers a poignant argument for the importance of developing appropriate notation for any mathematical endeavour:

> "By relieving the brain of all necessary work, a good notation sets it free to concentrate on more advanced problems, and in effect, increases the mental power of the race" [233].

At the risk of saturating the reader with Whitehead's prose, it is worthwhile to consider his further elaboration:

> "It is a profoundly erroneous truism, repeated by all copy-books and by eminent people when they are making speeches, that we should cultivate the habit of thinking of what we are doing. The precise opposite is the case. Civilization advances by extending the number of important operations which we can perform without thinking about them. Operations of thought are like cavalry charges in a battle—they are strictly limited in number, they require fresh horses, and must only be made at decisive moments" [233].

Within technical computing, this is best exemplified today by MV algebra, which "takes the slog out of nailing equations" [234]. To be more specific it takes the slog out of nailing linear mappings applied to vectors. A notation offered up to model phenomena not fitting within this category, *e.g.*, special linear mappings and mappings beyond linear, should emulate MV algebra's superb abilities in setting the mind free from notational distractions.

The NT algebra we describe, which is built upon the rooted foundation of Einstein notation, sets itself apart from other high-degree algebras in this regard. For instance, it offers practitioners an unmatched set of operations for high-degree data. Compared to the state-of-the-art, only Joseph [123], Barr [70], and Åhlander [54] offer notation able to represent inner, entrywise, and outer products across arbitrary indices of N-degree data. By introducing an association identity, inverse notation, and attraction operator, Joseph's notation is even more comprehensive. NT algebra expands on Joseph's work by offering notation for factoring, pseudo-inverses, partial derivatives, nonlinear NT functions, and vector and matrix NTs. Thus NT algebra offers unmatched expressibility, providing practitioners with greater freedom to model and capture interactions involving high-degree data.

The universality of NT algebra eases use in another important manner. Without a universal algebra, practitioners must switch between notations depending on need. This is a common exercise, for instance, in the tensor decomposition field, where expositions frequently switch between n-mode and extended matrix-vector (EMV) notation. Switching between notations diverts practitioners from the problem at hand and superadds mental effort over and above any inherent difficulties of the problem. The comprehensive nature of NT algebra eliminates or at the very least minimises these interruptions.

Uniquely, NT algebra offers these capabilities without sacrificing ease of use. For example, the virtues of complete associativity and commutativity have long been recognised by proponents of Einstein notation. NT algebra retains these characteristics, except in the case of n-ary inner products where complete associativity is replaced by the association identity. Practitioners can group or isolate operands with minimal notational updates. Contrast this with EMV algebra, which requires vectorisation and specialised operators or identities to fully commute and associate. Or consider R-matrix, array algebra, and n-mode^{+} notation, which requires updating numeral inner-product designations. In contrast, with NT algebra in hand, practitioners can trivially perform operations that would normally be complex in other formalisms.

These highlighted aspects all focus on purely algebraic considerations. However, within technical computing, an algebra's affinity to software represents an additional and highly important factor affecting ease-of-use. NT algebra enjoys many advantages within this regard. For instance, NT algebra is only composed of alphanumeric characters, allowing it to be readily employed within programming environments. Moreover, NT algebra shares Einstein notation's well-recognised compatability with computation, where connections to indexed loops, the workhorse of computation, are highly apparent [54, 66, 107, 154].

Thus, we view NT algebra as living up to the spirit of Whitehead's criteria for a good notation. With the vision of an accepted and widespread technical computing framework beyond the MV paradigm in mind, the NT algebra represents an important contribution toward this aim.

7.1.3 Software

Within technical computing, an algebra can reveal avenues toward solving scientific problems. But reaching those solutions requires fast and stable software able to work with large-scale numeric data. As we identify, NT software should comprehensively support NT algebraic operations, should offer complete dense and sparse functionality, and should aim to be as programmatically and computationally efficient as possible.

The open-source NT software we develop meets these criteria. Embodied by LibNT and NTToolbox, built for C++ and MATLAB respectively, our NT software allows users to program directly using NT algebra. In this way, one of the most notable strengths of the MV paradigm is replicated—users can conceptualise and implement scientific solutions using the same language on computer as they do on paper.

The LibNT library, whose primary consideration is computational efficiency, supports NT algebra at compile-time using C++'s template metaprogramming (TMP), keeping runtime abstraction penalties to the absolute minimum. This allows LibNT to support NT operations that in other libraries produce runtime penalties. As such, it is the first to provide compile-time resolution of an index notation with no limit on degree. LibNT is also the only library to statically resolve all of inner, entrywise, and outer products combined with assignments, n-ary inner products, and linear inversion of equations.

NTToolbox, whose primary consideration is programming efficiency, relies on the same core algorithmic kernels as LibNT, but resolves NT algebra at runtime within the MATLAB environment. Hence, NTToolbox offers a highly programmatically-efficient environment for NT algebra, while ensuring major computations are off-loaded to LibNT's high-performance routines. NTToolbox is the first to allow users to program *directly* with an index notation of any kind within the MATLAB environment.

Both libraries use the lattice as a computational data structure, a distinction from other high-degree software. The lattice data structure provides a common constructive platform by which any combination of inner, entrywise, and outer products across NTs of arbitrary degree can be executed or inverted efficiently using optimised gold-standard algorithms.

We also outline efficiencies for dense algorithms, explaining strategies to minimise temporary memory allocation and shedding light on the merits of in-place vs. out-of-place dense high-degree permutations. Moreover, scenarios where index calculations and mappings to lattices can be skirted are explained, further adding to efficiency gains. Despite offering a more generalised set of arithmetic operations, the dense performance of the NT software is competitive with or exceeds the performance of leading high-degree libraries, including the MATLAB Tensor Toolbox (MTT), FTensor, LTensor, NumPy, and Blitz++.

These advancements within dense settings are complemented by considerable innovations for sparse NT computations. The linearised coordinate (LCO) datatype and its merits over the predominant coordinate (CO) datatype are outlined, demonstrating that it enables faster execution of sorting operations while also having a smaller memory footprint. Secondly, reflecting their importance toward realising a high-performance sparse NT library, we develop several innovations for rearranging non-zero data. For instance, we are the first to outline how permuted sparse data contains inherent structure, which can be exploited to reduce permutation times. An algorithm exploiting these underlying characteristics was developed, outperforming the fastest sorting options. Finally, we address how to implement sparse-times-sparse NT multiplication, an arithmetic operation that exemplifies the unique requirements of sparse NT computations. A novel multiplication poly-algorithm was developed that chooses algorithms tailored to the hyper-sparsity characteristics presented by the sparse NTs. Benchmarks illustrate significant performance improvements over the MTT, the current leader in sparse NT computations.

No other high-degree software library offers as extensive a set of fundamental arithmetic operations as LibNT and NTToolbox. Moreover, LibNT and NTToolbox both support operations on dense, sparse, or mixtures of dense and sparse NTs. This unmatched expressibility is realised while matching or exceeding the performance of other high-degree libraries.

By considerably increasing the performance and efficacy of core operations, *e.g.*, multiplication, the building blocks crucial for higher-level algorithms can be made solid. This will benefit any application involving high-degree computations, *e.g.*, the high-level tensor decomposition routines of the MTT or the high-degree exemplars discussed in this thesis. Thus, the NT software we develop significantly expands the boundaries of practical and efficient high-degree computations, making it an effective and powerful partner to NT algebra.

7.1.4 Exemplars

Exemplars are frequently a decisive means to convey scientific concepts and arguments. Work on high-degree algebra and software are no exception. When cast as anomalies in the state of the art, exemplar problems shed light on the limits of the MV paradigm. Moreover, they reveal opportunities for technical computing frameworks beyond the MV paradigm. We focus on the exemplar categories of special linear mappings and mappings beyond linear. Reflecting the importance of exemplars in highlighting and advancing frameworks beyond the MV paradigm, we use several to explain and demonstrate the merits of the NT framework.

For instance, we highlight the NT framework's capability to naturally capture the entrywise products, least-squares solutions, and opportunities for efficiencies in the alternating least squares (ALS) algorithm for canonical-polyadic (CP) tensor decomposition. The ALS algorithm is one of the most important algorithms within the tensor decomposition field [83], which falls under the mappings beyond linear category. In addition, we focus on the ability of the NT framework to capture similar efficiencies seen in a linear parameter estimation exemplar, taken from antenna research, involving entrywise products and ternary inner products. These are two prominent exemplars that have been tackled with n-mode$^+$ and EMV algebra in the literature. By highlighting NT algebra's ease of use and more comprehensive set of operations, we argue that NT algebra is the superior formalism for these exemplars. Moreover, using benchmarks based on arithmetic operations drawn from CP tensor decomposition, we demonstrate how LibNT and NTToolbox can provide efficiency and performance gains over competitor libraries, illustrating how NT software can advance this key exemplar.

Many of the current exemplars in the literature stem from tensor decomposition. While this is a crucial field, we believe other types of exemplars require further articulation before high-degree frameworks for technical computing gain widespread acceptance. For instance, this work highlights separable nonlinear least squares (SNLS), a hugely impactful [149] optimisation technique that relies on partial derivatives of a second-degree NT, producing a third-degree NT. Prior work on high-degree frameworks had not highlighted this exemplar.

We also focus heavily on high-degree differential operators, whose connection to partial-differential equations (PDEs) makes them integral to many scientific fields. One such field is computer vision, which frequently relies on high-degree differential operators. These often take the form of finite-difference (FD) operators when they act upon its regularly-gridded imaging data. Considering that entrywise products, ternary inner products, partial differentiation, and non-linear functions often play a role, computer vision problems can present formidable obstacles to the MV paradigm, making it fertile ground for the NT framework.

Inspired by the author's own computer vision work, we discuss three exemplars from two computer vision problems, demonstrating the NT framework's potential for this important

	High-Degree Mappings	Entry-wise Products	Ternary Inner Products	High-Degree Linear Inversion	High-Degree Sparsity	Symbolic Differentiation	Nonlinear NT Functions
RW Image Segmentation	✓	✓	✓	✓	✓	✓	
LML Depth-Map & Albedo Estimation	✓	✓	✓	✓	✓		
NML Depth-Map & Albedo Estimation	✓	✓		✓	✓	✓	✓

Table 7.1: The characteristics of the computer vision exemplars of Chapter 6. These exemplars all challenge the MV paradigm. Nonetheless, these challenges can be met by the NT framework.

and challenging field. As Table 7.1 outlines, these exemplars involve aspects that challenge the MV paradigm both algebraically and computationally. As we show, the NT framework is well-suited to tackle these exemplars.

The first exemplar consists of random walker (RW) image segmentation [56], which is a seminal interactive image segmentation technique that has heavily impacted the computer vision and medical-imaging analysis community [1]. This includes the author's own medical-imaging analysis work, which is an extension of the RW technique [203]. Regardless of its elaboration, the RW technique can be cast as a high-degree and anisotropic elliptic PDE. Formulating the solution to this problem requires fourth-degree NTs, high-degree linear inversion, ternary inner products, and partial derivatives, which cannot be represented naturally by the MV paradigm. However, as we show, the technique can be readily represented using NT algebra. Importantly, the problem involves sparsity, reinforcing the need for the sparse NT computations developed in this work. NTToolbox code implementing the RW algorithm showcased the ease by which NT algebraic novelties can be computed.

Depth-map and albedo estimation serves as the source for the two other computer vision exemplars. Estimating depth and albedo is a seminal computer vision technique, representing one of the core variants of the so-called Shape-From-X methods. In this work, we first discuss the linearised maximum likelihood (LML) technique, which is drawn from the author's work [55] and follows the classic two-step approach toward this problem. We formulate the technique using NT algebra, showcasing the ability of the formalism to handle the high-degree data and operators involved in the LML method. Like RW segmentation, differential operators combined with entrywise products, ternary inner products, and high-degree least-squares solutions result in an anisotropic elliptic PDE, emphasising the links between

[1]Grady has received the Edison Patent Award for developing the RW technique and his original paper has been cited over 1300 times to date, according to Google Scholar.

the two exemplars. The final exemplar is a refinement of the LML method, called nonlinear maximum likelihood (NML) estimation, that uses one single large-scale nonlinear step. The NT framework was used to develop and implement the NML method, which provides superior representations of depth-maps and albedos under low-observation counts [164]. As the problem requires keeping track of 6 different indices and incorporates nonlinear NT functions, the NML method is particularly unsuited for the MV paradigm. Adding to the complexity, NML estimation can be formulated as an SNLS problem, which requires yet another index in addition to a high-degree pseudo-inversion involving entrywise products. Despite these complications, the NT framework can readily tackle the problem both algebraically and computationally, serving as further illustration of its merits for computer vision applications.

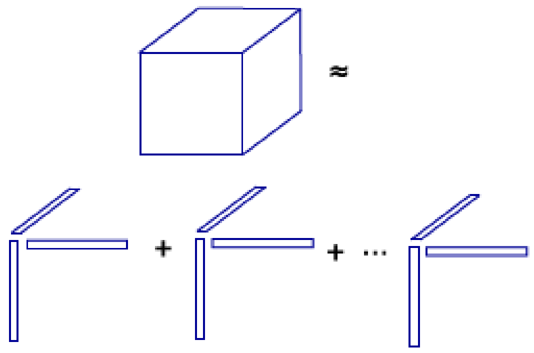
The exemplars we highlight reflect the diverse origins of work involving high-degree frameworks. As Figure 7.2 visually illustrates, these exemplars represent a considerable range of problems relevant to many fields. Prior to this work, these exemplars had not been discussed together. By using the NT framework to link these exemplars together, and by also expanding the stockpile of exemplars to include the ones listed in Table 7.1, this work consolidates diverse efforts under a unifying framework. Thus, the growing crowd of researchers pushing for and developing technical computing frameworks beyond the MV paradigm can be placed within a shared context, magnifying the strength of this important research movement.

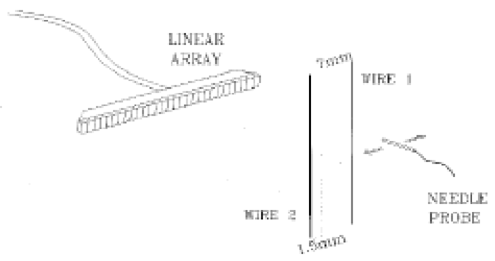
7.2 Future Work

Kuhn emphasised the value of research questions in animating and defining a discipline's practice. Technical computing work beyond the MV paradigm is no exception. In terms of the NT framework of this work, questions still remain on how to further enlarge the soul and body of the NT framework, *i.e.*, continuing to develop its algebra and software. Complementary to these efforts, the pursuit of additional notable exemplar problems beyond the MV paradigm will further establish the impact of the algebraic and software innovations of the NT framework.

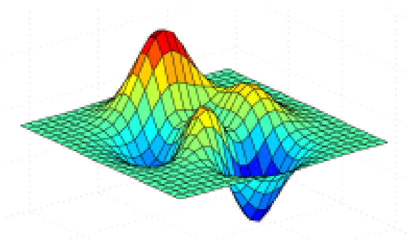
7.2.1 Algebra

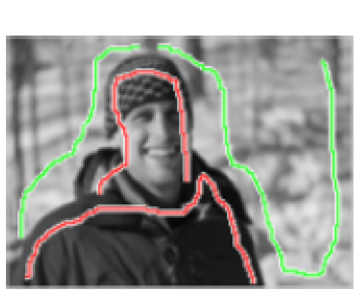
NT algebra employs a single type of index, *i.e.*, the subscripted index. Traditionally, Einstein notation employs two types of indices, *i.e.*, contravariant (superscripted) and covariant (subscripted) indices. The reasons for this lie in the geometric tensor origins of Einstein notation, where a change of basis within a curvilinear coordinate system requires the concepts of contravariance and covariance. Naturally, when employing NT algebra within curvilinear geometric contexts, contravariance and covariance will need to be accommodated. To accomplish this the implications of supporting dual-index types in concert with entrywise products, linear inversions, and n-ary inner products need to be better understood.


(a)



(b)


(c)


(d)

(e)

Figure 7.2: Variety of NT framework exemplars explored in this thesis. We showcase exemplars ranging from CP tensor decomposition, shown in (a), to linear parameter estimation (drawn from antenna signal processing) using entrywise products, shown in (b). The SNLS approach to discover the minimal SSE, *e.g.*, from the optimisation surface depicted in (c), was also highlighted. Finally, we devote considerable space detailing how the computer vision exemplars of image segmentation, shown in (d), and depth-map estimation, shown in (e), can be tackled using the NT framework. (b) was taken from Prada *et al.* [235].

Interestingly, practitioners from fields not concerned with geometric tensors *per se*, *e.g.*, statistics [107], econometrics [154], and numerical analysis [54], have also advocated for dual-index-type Einstein notation. As McCullagh [107] and Pollock [154] argue, it may be the case that partial derivatives are best expressed with dual-index types. On the other hand, the single-index-type variation has met our needs, so far, and those of other practitioners [70, 86, 157]. Researchers, possibly unaware of the single-index-type variant, even cite the dual-index types of Einstein notation as a reason to avoid the algebra [236]. This indicates that further work is needed to better understand the advantages and disadvantages of dual-index types in non-geometric situations. Articulating when exactly single- and dual-index types should be used would help further extend and disseminate NT algebra to a wider audience.

Another formalist aspect that should be explored lies in whether the rules for addition and subtraction can be relaxed. For instance, the following expression,

$$a_{ij} + b, \tag{7.1}$$

is easily interpreted as adding the scalar b to all elements within a_{ij}. However, strictly speaking this is not a legal expression within Einstein notation or NT algebra. Nonetheless, expressions like (7.1) are a frequent and useful practice within programming environments. For instance, it is common and expedient to add or subtract a scalar from a matrix within MATLAB, even though this operation is forbidden by MV algebra.

Ideally, if allowed, expressions like (7.1) should be formalised within an improved NT algebra. This would open up other possibilities, *e.g.*,

$$a_{ij} + b_i + c_j, \tag{7.2}$$

which can be understood to be identical to

$$a_{ij} + b_i 1_j + c_j 1_i. \tag{7.3}$$

We call expressions like (7.1) and (7.2), including subtractive cases, outer addition. While the usefulness of outer addition is apparent, the implications of formalising it is not. These implications should be explored. Should it be prudent to incorporate outer addition within NT algebra, the notation would align even more closely to programming practice.

7.2.2 Software

As Schatz *et al.* state in their 2014 paper, "libraries for dense multilinear algebra (tensor computations) are in their infancy" [153]. We believe this assessment is just as apt for sparse NT computations. Both are discussed below. While our contributions have advanced the state of dense and sparse NT software, we believe there remains much important work.

For instance, one area that deserves further elucidation is determining the most efficient order of execution for n-ary NT products. Consider the following equivalent expressions, assuming all NTs are dense:

$$a_i b_i c_j = (a_i b_i) c_j, \tag{7.4}$$

$$= a_i (b_i c_j). \tag{7.5}$$

By executing the inner product first and eliminating the i index, the order of execution in the right-hand side of (7.4) is more efficient than the alternative in (7.5). For more complex expressions, the situation is not so trivially assessed, where the most efficient order of execution depends on the product type and dimensionalities of the operands in question. Yet, determining the most efficient execution order can have major consequences.

The ALS algorithm provides a persuasive example where the order of execution has major implications. A well-used identity reveals a more efficient means to calculate certain types of pseudo-inverses, significantly reducing the algorithm's computational complexity [83]. As outlined in Section 3.3.1, this identity is wholly driven by advantageously altering the order of NT products. A universal and rigorous process to determine efficient orders of execution would extend these benefits to arbitrary expressions.

In fact, researchers within computational chemistry have been developing solutions to this problem. The Tensor Contraction Engine (TCE) automatically generates low-level FORTRAN code based on an analysis of Einstein-notation expressions involving tens to hundreds of terms [91, 92, 93]. In addition to discovering effective pairings of operands, the TCE performs other optimisations that balance memory-use and computational efficiency. These innovations should be translated to the more general-purpose setting of the NT framework, which may not require the same level of sophistication. Should such a process be incorporated within NT software, n-ary product expressions could be automatically and seamlessly executed as efficiently as possible. Translating these benefits to the sparse case would be even more challenging, presenting further opportunities for research developments.

Moving to other directions of future work, to stay cutting edge, NT software should align with prevailing trends in technical computing. For one, just like in MV software, effective strategies to use vector operations, *i.e.*, single-instruction multiple-data operations, within high-degree computations is an important research thrust. As well, the increasing prevalence of multi-core, distributed, and heterogeneous computing environments provides impetus to develop mature parallel NT algorithms. The entrywise product, cast as different tabs within a lattice product, offers a natural avenue toward parallel implementations. However, parallel approaches to large-scale dense and hyper-sparse products, dense and sparse permutations, and other operations specific to high-degree settings should be developed. Insight into both these considerations can be found within computational chemistry libraries and associated publications [91, 92, 93, 94, 95]. Implementing some of these breakthroughs within a general-purpose NT setting will help disseminate these approaches to a larger technical computing audience.

The concept of implicit NTs should also be expanded. In the dense case, NT products that only involve entrywise and outer products can be readily represented implicitly, requiring significantly less memory than explicitly computing the products. As a result, implicit representations could help deal with notable problems within NT computations, such as the intermediate blowup problem within tensor decomposition [176,177,198]. However, implicit representations do come with runtime abstraction penalties. Future work should focus on strategies to minimise these abstraction penalties and in parallel should investigate when implicit representations should be favoured over explicit computations.

The topic of symmetric NTs is also replete with opportunities for future work. Work within computational chemistry [91,92,93,94,95] and tensor decomposition [153] has focused on dense symmetric and anti-symmetric NTs. The NT framework should adapt these approaches. Commensurate to these developments, effective representations of symmetric *sparse* NTs should be developed. Although there has been some prior work [185, 237], more work is required for this mostly untouched subject. Amongst other applications, efficiently handling symmetric sparse NTs could be an essential component for using NTs for polynomial equations. Successful strategies for symmetric sparse NTs would likely depart from both sparse-MV and dense-NT solutions, meaning new approaches would have be developed.

Sparse NT computations provide other promising avenues of future work. For instance, sparse tensor decompositions represent a highly significant research effort today. High-performance kernels, such as those outlined in this work, can play an important role in prototyping and developing such techniques. Merging the MTT's considerable suite of tensor-decomposition data structures and algorithms with LibNT's arithmetic routines would be an important direction for future work.

Finally, knowledge translation efforts need to be broadened. For instance, the TMP innovations of this work require further documentation and dissemination. TMP innovations and high-degree computations are of interest to practitioners both inside and outside the academic sphere. While top-down dissemination, *e.g.*, publications, should be an important focus, online and tutorial-like documentation should be a core component of any knowledge translation strategy to make an impact within non-academic communities *e.g.*, the open source software community. The latter should include ready-to-use sample programs based on high-degree exemplars, such as those of this work. These initiatives would foster the bottom-up dissemination of NT framework concepts, mirroring the successful approach of Blitz++ [125], NumPy [129], and FTensor [146] in impacting the wider technical computing discipline. Together, both top-down and bottom-up knowledge translation efforts will help ensure that high-degree algebraic and software innovations reach as broad an audience as possible, maximising the potential for long-lasting impact.

7.2.3 Exemplars

By touching on a varied set of applications, the exemplars we highlighted strengthen the case for the NT framework. The benefits flow both ways, as high-degree frameworks also play a crucial role in advancing important applications, *e.g.*, the NT framework's role in implementing NML depth-map estimation. Yet, essential work remains to be done to both translate the advantages of the NT framework and to make headway on important applications.

One promising avenue of future work, which would also highlight the merits of explicitly accommodating entrywise products, is to develop practical means to describe inter-variance. For instance, consider a linear regression problem, *e.g.*,

$$y_i = a_{ij} x_j + \epsilon_i, \tag{7.6}$$

where the additive error, ϵ_i, is assumed to be independent and identically distributed (IID), with a mean of zero and a variance of σ. Should this regression problem be solved using linear least-squares, the covariance of the estimate is well understood:

$$\mathcal{E}(\hat{x}_j \hat{x}_{j'}) = \sigma (a_{ij} a_{ij'})^{-1}. \tag{7.7}$$

However, if a_{ij} is large and sparse, calculating the covariance can be intractable as the inversion is costly and typically produces a dense result. Moreover, what is often of interest is not the covariance, but what Joseph [123] calls the *inter*-variance, *i.e.*,

$$\mathcal{E}(\hat{x}_{\underline{j}} \hat{x}_{\underline{j}}), \tag{7.8}$$

which describes the entrywise variability of the estimated x_j.

If a means to compute (7.8) can be established without completely inverting a large-and-sparse linear mapping, entrywise confidence intervals may be computed. As Figure 7.3 illustrates, this can be used to calculate the variance of large-scale estimates like depth maps. This problem is closely linked to work on computing the diagonal of matrix inverses [238, 239, 240, 241], where the matrix is sparse though its inverse is dense. When cast using NT algebra, entrywise products help make the expression of the inter-variance explicit. When considered together with corresponding sparsity computations, we view the NT framework as a promising platform with which to tackle this problem.

Like the differential operator exemplars of Chapter 6, inter-variance exemplars fall under the special linear-mappings category. However, the category of mappings beyond linear, especially multilinear and polynomial mappings, is an enormous topic whose challenges and potential are legion. Tensor decomposition, which falls in this category, is a highly important area of focus for the NT framework, which we have expanded upon within this thesis. We also view the field of numerical polynomial algebra, concerned with manipulating and solving polynomial equations, as another vital source of exemplars.

Currently, the means to symbolically solve polynomial equations are fairly well understood. Yet, reliable numerical approaches are still lacking [74]. Works within numerical polynomial algebra typically employ MV-based notation [74, 242, 243, 244]. Manipulating such representations, *e.g.*, polynomial multiplication or change of basis, is not trivial and is

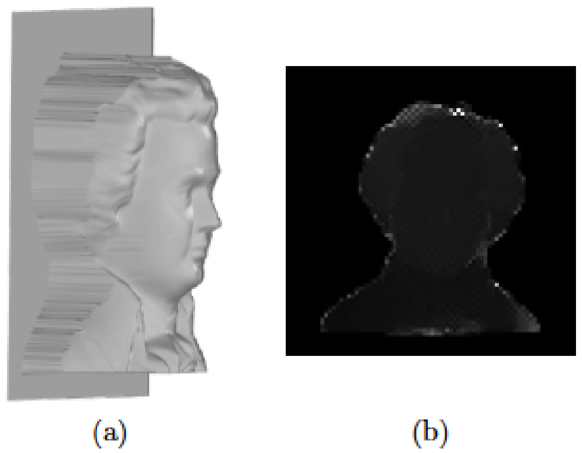


Figure 7.3: Estimating the variance of depth-map estimates. Noisy images of the Mozart surface, seen in (a), were generated and the LML method was used to estimate depth-maps from this noisy input. This was repeated 10 times, allowing a crude Monte-Carlo estimate of the depth-map variance to be computed, visually depicted in (b) where whiter pixels denote higher variance. Developing a means to calculate inter-variance would allow the variance of the depth-map estimates to be computed using ML principles without resorting to repeated trials, which may be impractical or inaccurate. The same may be said for other large-scale regression problems.

dependant on the monomial ordering of coefficients. Additionally, an MV approach ignores the links between multilinear forms and polynomial equations, which stems from whether one represents a polynomial in its apolar or polar form [245]. Multilinear forms and equations are most naturally described within an NT setting.

Because symmetric and non-symmetric NTs can naturally represent polynomial and multilinear equations, respectively [78], the NT framework, as with other tensor-based frameworks, can serve as numerical platforms to solve beyond linear-mapping exemplars. Using NTs as such a foundational concept links polynomial and multilinear systems together. Moreover, the numerical polynomial algebra field also becomes connected to the tensor decomposition field, benefiting from its tools and techniques. Since practical polynomial equations are typically highly sparse [74], the constructive innovations we and others [124, 153] detail are expected to play an important role.

Should NTs prove to be the best way to tackle numeric polynomial and multilinear systems, we view formalisms based off of Einstein notation, *e.g.*, NT algebra, as the most promising. While n-mode$^+$ notation, favoured by the MTT, could conceivably model systems of polynomial equations, that formalism is strongest when describing the so-called multilinear product or n-mode product. Many interactions between polynomials, *e.g.*, composition, addition, or weighted-sums of polynomial equations, stray from the multilinear product into areas where n-mode$^+$ notation is weaker. Exploring and articulating these capabilities would elevate the NT framework's impact toward even more far-reaching levels.

7.3 Final Remarks

We conclude this thesis by offering some thoughts on the current state of technical computing and some speculations on how the discipline's paradigm may evolve. As such, these represent our opinions and personal assessments. We start by making the case that technical computing is currently within a revisionary period, which is our preferred terminology over Kuhn's "crisis". We follow this up by speculating on how this revisionary period will be resolved.

7.3.1 Evidence of "Crisis"

Throughout this thesis we have emphasised the significance of the considerable body of work on algebra and software beyond the MV paradigm, demonstrating that this movement, hitherto unrecognised, is a substantial force within technical computing. Taking this analysis one step further, we argue that this body of work is evidence of revisionary science, meaning the current MV paradigm is beset with fundamental challenges. Moreover, these challenges imply a change of technical computing's taxonomy is underway, re-categorising anomalies and exemplars. In *Structure*, Kuhn describes this period as a crisis, which results in scientists stepping outside the bounds of normal science to begin practicing extraordinary science. Keeping with our preference of eschewing the dramatic aspects of Kuhn's language, we avoid the use of crisis and extraordinary science, opting for the terms *revisionary period* and *revisionary science*, respectively.

Kuhn states that a revisionary period is characterised by a "proliferation of competing articulations, the willingness to try anything, the expression of explicit discontent, the recourse to philosophy and to debate over fundamentals" [1]. We view this as an apt description of current work on high-degree algebra and software for technical computing and we will assess the individual components of Kuhn's statement against the evidence.

In terms of "competing articulations", Kuhn included attempts to extend the prevailing paradigm to address new phenomena, blurring the rules of its practice. Kuhn describes these efforts as often *ad hoc* responses to anomalies. We consider EMV algebra to be such an articulation, which as we have detailed is not a consolidated formalism, expressing competing notations and conventions for several different operators, including the Kronecker product itself. Other complications abound. For instance, Harshman and Hong devote an entire paper comparing two different approaches to use MV algebra to represent third-degree data and operators upon it [118]. As another example, Magnus discusses the rationale behind using his preferred EMV representation of derivatives over several competing versions [117]. A frequent source behind these complications is the "unpleasant ordering of indices" [106] needed in order to use matrices to represent high-degree constructs. Unsurprisingly, it seems that there is more than one way to blur the rules of the MV paradigm.

Kuhn also maintained that efforts to extend a prevailing paradigm can produce complexity, which may outpace any associated benefits. We subscribe to this view concerning

EMV algebra and so do others. For instance, after outlining competing methods to use MV notation to describe high-degree data, Harshman and Hong conclude by stating, "If one is willing to abandon matrix notation, all the complications discussed here can be avoided" [118]. Their preferred solution is Harshman's Einstein notation variant [86]. As we described earlier, adjectives used to describe EMV include "troublesome" [65], "cumbersome" [60], "unpleasant" [106], and "awkward" [54]. These statements certainly meet Kuhn's criterion of "expressions of explicit discontent".

As per Kuhn, the variety of proffered solutions toward representing high-degree data and operations suggests a "willingness to try" many approaches. These approaches range from the aforementioned EMV algebra, to n-mode$^+$ notation, R-matrix notation, array algebra, and Einstein notation. Even amongst these examples there are differing views, *e.g.*, the different flavours of Einstein notation offered by Tait [113], Antzoulatos and Sawchuck [60], and Harshman [86]. Or consider the proponents of single-index Einstein notation [70, 118, 157] vs. the dual-index variant [54, 67, 106, 107, 154]. Specific fields even resort to multiple formalisms. For example, within tensor decomposition, notation "differs widely over different papers and disciplines, and so does the terminology practitioners have employed" [110]. Notations used within the field include EMV algebra [83], n-mode notation [83], and Einstein notation [86]. What we call n-mode$^+$ notation is represented by independent efforts to extend the original n-mode notation to describe a greater set of operations [98, 112]. Researchers from other fields also see occasion to use two competing notations. Sometimes the need for these two notations are expressed by the same person, *e.g.*, Pollock [64, 154] and Vetter [66, 100, 155, 156] have both articulated the use of EMV algebra and Einstein notation for econometrics and signal processing, respectively.

The plethora of competing articulations stem from efforts to describe anomalous phenomena. As Kuhn maintained, these are often independent efforts where practitioners, discovering that the prevailing paradigm offers no guidance on investigating an important anomaly, resort to highly varied and creative solutions. This is no better epitomised than within the tensor decomposition field. As described by Kolda and Bader in their invaluable review [83], the history of the field is rife with elaborations that were forgotten, re-discovered, and independently developed by practitioners hailing from fields as distinct as psychometrics to biomedical imaging. Kolda and Bader even offer tables outlining how both CP and Tucker decomposition have each had five different names depending on the date and domain of the literature. The algebraic innovations within tensor decomposition have also been independently articulated within other fields. For example, n-mode notation shares many similarities with the R-matrix notation developed by geodesy researchers interested in describing and decomposing high-degree linear mappings. Many aspects of Bader and Kolda's n-mode$^+$ notation is shared by Suzuki and Shimizu's array algebra [65], which was articulated for signal-processing purposes not connected to tensor decomposition.

Competing articulations often represent serious "debates over fundamentals". We outlined some of the debate seen over high-degree algebras; yet, the debate is not confined to formalism, as the discussion enters the realm of software as well. For example, high-degree software seems to fall into two camps regarding dense computations. There are those, like LibNT, NTToolbox, and the MTT, that permute data before multiplying [91, 92, 93, 94, 95, 112] and those that do not [101, 125, 127, 175]. Sparse data formats use compressed indices [150, 151, 152, 181] or uncompressed indices [124, 185], the latter being what we favour. Algorithms to multiply sparse NTs must deal with hyper-sparsity. Approaches include excising all-zero rows and columns, *e.g.*, as with the MTT [124], or an approach like ours that applies a poly-algorithm based on the different sparsity properties of operands. All of these examples are fundamental differences in how to approach the challenging topic of computing with high-degree data. More than that, the issues being pursued are new questions that simply do not arise within the MV paradigm, offering further evidence of the revisionary nature of this work.

Questions concerning mappings beyond linear are also firmly embroiled in fundamental debates. For instance, tensor decomposition currently offers two main models of high-order SVD (HOSVD), *i.e.*, the Tucker and CP decomposition. These are complemented by a plethora of other types of decompositions [83]. Just as striking as tensor decomposition, those interested in solving polynomial systems of equations are currently faced with enormously varied and different options [76]. The differences between choices are much more than a matter of practicality. To pick two options, homotopy and Gröbner basis methods represent entirely different philosophical approaches to solving polynomial systems of equations.

Returning to Kuhn's analysis of revisionary periods, a "recourse to philosophy" is the one criterion that, this thesis aside, seems to be lacking. While McCullagh certainly waxes philosophical in the preface of his book advocating Einstein notation for statistics applications [107], he represents an isolated example. Perhaps this is just a product of scientific writing today, particularly in technical computing, which is less florid than examples found in the periods of science Kuhn studied. Nonetheless, by pursuing questions outside the MV paradigm, the body of work on high-degree algebra and software is expanding the bounds of technical computing, revising the anomalies and exemplars of its scientific practice. The attempts to articulate modifications to the MV paradigm, and the competing attempts to, in effect, articulate alternatives to the MV paradigm, align markedly well with Kuhn's descriptions of revisionary science. Accepting this argument precipitates an important question—how will this revisionary period be resolved? We speculate on this topic below.

7.3.2 Resolution of "Crisis"

The quality and quantity of articulations outside the boundaries of the MV paradigm indicate technical computing is experiencing a period of revisionary science. Based on Kuhn's writings, including his later contributions, a revisionary period can be resolved in three ways:

- the prevailing paradigm endures by satisfactorily dealing with crisis-provoking anomalies;
- a specialisation is established that incorporates its own new paradigm; or
- the entire discipline adopts a new paradigm.

For the topic at hand, we do not subscribe to the first outcome above, *i.e.*, we do not view efforts to extend the MV paradigm to describe high-degree data or mappings as sufficient. Hitching oneself to the MV paradigm is approaching the subject obliquely and fails to do justice to the demands and rewards of work on high-degree data. Moreover, today the subject of high-degree computations is an entrenched area of research within technical computing. Current trends, *e.g.*, the proliferation of literature focusing on or incorporating tensor decomposition, indicate only a continuing increase in importance. Returning to the analogy of Milgram *et al.*'s experiment [87], the crowd looking up at the street corner has grown too large to ignore.

Based on this argument, we expect the current revisionary period to be resolved via a paradigm shift—either for technical computing as a whole or for a specialisation within the discipline. Yet, the implications of a new paradigm within technical computing are not clear. Kuhn, whose writings focused heavily on physics, has little to say about computing [13]. Hence, it is worth pondering the implications of a new paradigm within technical computing. In *Structure*, Kuhn maintained that accepting a new paradigm requires rejecting the worldview of the old one. But he was careful to emphasise that this did not necessarily involve rejecting the practical basis of the old paradigm. For instance, while Newtonian mechanics is understood as an approximation even when used in non-relativistic contexts, its practical use is undeniable. Its inaccuracies are so minute that it is hard to imagine any other model usefully describing the motion of everyday macroscopic objects within the planet's environs.

Returning to technical computing, just like Newtonian mechanics in non-relativistic settings, it is hard to imagine a paradigm better suited than the MV paradigm within its natural context, *i.e.*, modeling and computing with linear mappings applied to vectors. Unlike the Newtonian mechanics analogy, we do not expect a new paradigm to reveal subtleties, however minute, regarding vector-based linear mappings that may be unaccounted for by the MV paradigm. In this sense a paradigm shift within technical computing may differ from Kuhn's examples in the degree of rejection involved.

We are prepared, however, to conjecture one type of rejection. The MV paradigm implies a certain range of technical computing's practice and limitations. A new paradigm would irrecoverably reject this conception and introduce its own language and taxonomy to describe its anomalies and exemplars. This may be what Kuhn meant as change of worldview, but he was typically ambiguous. Hacking argues Kuhn had a stronger viewpoint:

> "A cautious person may agree that after a revolution in her field, a scientist may view the world differently, have a different feeling for how it works, notice different phenomena, be puzzled by new difficulties, and interact with it in new ways. Kuhn wanted to say more than that" [13].

Regardless of Kuhn's intent, we view a paradigm shift within technical computing to align with Hacking's "cautious" summary.

With the features of a paradigm shift within technical computing established, the question remains on the scope of this change of worldview. As mentioned, either the MV paradigm is replaced within the whole of technical computing or a specialisation is established that incorporates its own high-degree paradigm. Which of these outcomes takes hold likely depends on which exemplars seize the field's imagination.

For instance, today arguably the two most prominent topics of work beyond the MV paradigm are special linear mappings and tensor decomposition. While incredibly important, both such topics are of interest only to researchers needing to explicitly describe high-degree data or to decompose multilinear interactions between such data, respectively. Yet, the demands of many technical computing applications do not extend beyond a notation and software to solve equations for low-degree data. If established techniques to solve systems of equations remain restricted to linear solvers and non-linear optimisation approaches, the MV paradigm would remain unassailable. Hence, should novel and significant exemplars be confined to special linear mappings and tensor decomposition, specialisation seems most likely, with the MV paradigm comfortably ensconced as a paradigm for those not needing to explicitly work with high-degree data.

The elevation of multilinear and polynomial systems of equations as a major technical computing exemplar would alter this outlook. As others have noted [74, 243], numeric approaches to this topic are a major gap within technical computing's body of knowledge. Effective means to manipulate and solve numeric multilinear and polynomial systems of equations would extend the scope of the entire discipline. This far-reaching impact suggests that adopting this question as an exemplar, and articulating promising avenues toward its resolution, would engender a paradigm shift affecting the entire technical computing discipline. Note that this does not mean the MV framework would be discarded, it would just be incorporated as part of a new paradigm. This may truly be revolutionary, an event that would be worthy of Kuhn's dramatic terminology.

But what of the NT framework of this thesis? Will its principles influence a new paradigm? We think they should. We have outlined some of the advantages of the NT framework over both the MV paradigm and other high-degree frameworks. These include a simplicity of use and comprehensiveness that allow it to bridge previously disconnected work on high-degree data. This unifying quality can give the NT framework a powerful edge. Yet, ultimately, the NT framework's place will be determined by whether its principles are widely adopted. As Kuhn argues, there is no purely objective criteria by which to judge paradigms or alternatives to them. Other persuasive elements like aesthetics and authority play their role too. For this reason, we are comfortable offering some of our own opinions in this regard.

For one, we unabashedly appeal to authority. NT algebra is rooted in Einstein notation, a formalism that enjoys undeniable pedigree and success. Impassioned and modern arguments, *e.g.*, those of Papastavridis [69] and McCullagh [107], have extolled its virtues, indicating that the notation has withstood the test of time. Einstein's name is not attached to the notation through historical happenstance. He helped develop it, and we happily appropriate some of his authority for our own purposes. This is not all that we appropriate, as the NT framework shares Einstein notation's aesthetic appeal. Using letters to designate indices allows them to take on greater purpose than simple placeholders designating the first, second, or nth index in high-degree data. Instead, indices are prominent actors within mathematical expressions, describing highly complex matters with deceptive ease, *e.g.*, how exactly two operands should interact arithmetically and how the result should interact with other operands. The expressed complexity only becomes apparent when one becomes embroiled in the messier numeral-based approaches seen in other high-degree algebras. These aesthetic and authoritative aspects are important facets of the NT framework.

Whether we have made a persuasive enough case for the NT framework's principles is up to the reader. What is harder to challenge is that there exists an increasing body of work on new algebraic and software approaches outside the MV paradigm, and that this body of work is making an ever expanding impact. As observers, it seems to us that we are witnessing a singular moment within technical computing. Even more exciting, current practitioners, researchers, and scientists are presented with an uncommon opportunity to shape the discipline as it moves forward from its current crossroads. We hope the reader will join us by sharing in the vision of a widely-accepted and universal technical computing framework beyond the MV paradigm.

References

[1] T. S. Kuhn, *The Structure of Scientific Revolutions: 50th Anniversary Edition.* University of Chicago Press, 2012.

[2] I. Scheffler, "Vision and Revolution: A Postscript on Kuhn," *Philosophy of Science*, vol. 39, no. 3, pp. 366–374, 1972.

[3] H. Sankey, *The Incommensurability Thesis*, ser. Avebury Series in Philosophy. Avebury, 1994.

[4] T. S. Kuhn, *The Essential Tension.* The University of Chicago Press, 1977, ch. Second Thoughts on Paradigms, pp. 293–319.

[5] K. B. Wray, *Kuhn's Evolutionary Social Epistemology.* Cambridge University Press, 2011.

[6] Presidents Information Technology Advisory Committee, "Computational Science: Ensuring America's Competetiveness," National Coordination Office for Information Technology Research and Development, Report to the President, 2005.

[7] C. C. Hurd, "Computer Development at IBM," in *A History of Computing in the Twentieth Century*, N. Metropolis, J. Howlett, and G.-C. Rota, Eds. New York, New York: Academic Press, Inc., 1980.

[8] C. Truesdell, "The Computer: Ruin of Science and Threat to Mankind," in *An Idiot's Fugitive Esays on Science: Methods, Criticism, Training, Circumstances.* Springer-Verlag, 1984.

[9] Z. Merali, "Computational science: ...Error," *Nature*, vol. 467, pp. 775–777, 2010.

[10] SIAM Working Group on CSE Education, "Graduate education in computational science and engineering," *SIAM Review*, vol. 43, no. 1, pp. 163–177, 2001.

[11] R. F. Boisvert, "Mathematical Software: Past, Present, and Future," *Mathematics and Computers in Simulation*, vol. 54, pp. 227–241, 2000.

[12] P. T. Boggs, "Mathematical Software: How to Sell Mathematics," in *Mathematics Tomorrow*, L. A. Steen, Ed. Springer-Verlag, 1981.

[13] I. Hacking, “Introductory essay,” in *The Structure of Scientific Revolutions.* The University of Chicago Press, 2012.

[14] W. T. Tsai, “Service-oriented system engineering: A new paradigm,” in *Proceedings of the IEEE International Workshop on Service-Oriented System Engineering (SOSE*, 2005, pp. 3–6.

[15] T. R. Colburn, *Philoshopy and Computer Science*, ser. Explorations in Philosophy, J. H. Fetzer, Ed. Armonk, New York: M. E. Sharpe, Inc., 2000.

[16] ——, “Methodology of computer science,” in *The Blackwell Guide to the Philosophy of Computing and Information*, ser. Blackwell Philosophy Guides, L. Floridi, Ed. Oxford, UK: Blackwell Publishing, 2004.

[17] R. W. Floyd, “The Paradigms of Programming,” *Communications of the ACM*, vol. 22, no. 8, pp. 455–460, 1979.

[18] M. Tedre and E. Sutinen, “Crossing the Newton-Maxwell Gap: Convergences and Contingencies,” *Spontaneous Generations: A Journal for the History and Philosophy of Science*, vol. 3, no. 1, pp. 195–212, 2009.

[19] P. J. Denning and P. A. Freeman, “The Profession of IT: Computing's Paradigm,” *Communications of the ACM*, vol. 52, no. 12, pp. 28–30, 2009.

[20] M. Tedre, “Computing as a Science: A Survey of Competing Viewpoints,” *Minds and Machines*, vol. 21, no. 3, pp. 361–387, 2011.

[21] J. H. Fetzer, “Philosophical aspects of program verification,” *Minds and Machines*, vol. 1, no. 2, pp. 197–216, 1991. [Online]. Available: http://dx.doi.org/10.1007/BF00361037

[22] B. Blum, “Formalism and prototyping in the software process,” in *Program Verification*, ser. Studies in Cognitive Systems, T. Colburn, J. Fetzer, and T. Rankin, Eds. Springer Netherlands, 1993, vol. 14, pp. 213–238.

[23] D. MacKenzie, *Mechanizing Proof: Computing, Risk, and Trust*, ser. Inside Technology, W. B. C. Wiebe E. Bijker and T. Pinch, Eds. Cambridge, Massachusetts: The MIT Press, 2001.

[24] J. Backus, “Programming in America in the 1950s,” in *A History of Computing in the Twentieth Century*, N. Metropolis, J. Howlett, and G.-C. Rota, Eds. New York, New York: Academic Press, Inc., 1980.

[25] D. E. Knuth and L. T. Pardo, “The Early Development of Programming Languages,” in *A History of Computing in the Twentieth Century*, N. Metropolis, J. Howlett, and G.-C. Rota, Eds. New York, New York: Academic Press, Inc., 1980.

[26] B. N. Parlett, "The Contribution of J.H. Wilkinson to Numerical Analysis," in *A The History of Scientific Computing*, ser. ACM Press History Series, S. G. Nash, Ed. New York, New York: ACM Press, 1990.

[27] D. Kelly, "A Software Chasm: Software Engineering and Scientific Computing," *Software, IEEE*, vol. 24, no. 6, pp. 120–119, Nov 2007.

[28] S. M. Ulam, "Von Neumann: The Interaction of Mathematics and Computing," in *A History of Computing in the Twentieth Century*, N. Metropolis, J. Howlett, and G.-C. Rota, Eds. New York, New York: Academic Press, Inc., 1980.

[29] A. Troelstra and D. van Dalen, *Constructivism in Mathematics: An Introduction*, ser. Studies in Logic and the Foundations of Mathematics, J. Barwise, D. Kaplan, H. Keisler, P. Suppes, and A. Troelstra, Eds. North-Holland, 1988, vol. 1.

[30] K. Eriksson, D. Estep, and C. Johnson, *Applied Mathematics: Body and Soul.* Springer, 2004, vol. 1.

[31] Y. Gurevich, "Platonism, constructivism, and computer proofs vs. proofs by hand." in *Current Trends in Theoretical Computer Science*, 2001, pp. 281–302.

[32] M. Tedre and E. Sutinen, "Three traditions of computing: what educators should know," *Computer Science Education*, vol. 18, no. 3, pp. 153–170, 2008.

[33] M. B. Wells, "Reflections on the Evolution of Algorithmic Language," in *A History of Computing in the Twentieth Century*, N. Metropolis, J. Howlett, and G.-C. Rota, Eds. New York, New York: Academic Press, Inc., 1980.

[34] K. O. May, "Historiography: A Perspective for Computer Scientists," in *A History of Computing in the Twentieth Century*, N. Metropolis, J. Howlett, and G.-C. Rota, Eds. New York, New York: Academic Press, Inc., 1980.

[35] P. Martin-Löf, "Constructive Mathematics and Computer Programming," *Royal Society of London Philosophical Transactions Series A*, vol. 312, pp. 501–518, 1984.

[36] K. Åhlander, M. Haveraaen, and H. Z. Munthe-Kaas, "On the Role of Mathematical Abstractions for Scientific Computing," in *The Architecture of Scientific Software*, ser. IFIP Advances in Information and Communication Technology, R. F. Boisvert and P. T. P. Tang, Eds. Spring, 2001, vol. 60.

[37] J. Hoffman, C. Johnson, and A. Logg, *Dreams of Calculus Perspectives on Mathematics Education.* Springer, 2004.

[38] K. Erikkson, D. Estep, P. Hansbo, and C. Johnson, *Computational Differential Equations.* Cambridge University Press and Studentlitteratur, 1996.

[39] J. von Neumann and H. H. Goldstine, "Numerical Inverting of Matrices of High Order," *Bulletin of the American Mathematical Society*, vol. 53, no. 11, pp. 1021–1099, 11 1947.

[40] H. H. Goldstine, "Remembrance of Things Past," in *A History of Scientific Computing*, S. G. Nash, Ed. ACM Press, 1990.

[41] W. M. Gentleman and J. F. Traub, "The Bell Laboratories Numerical Mathematics Program Library Project," in *Proceedings of the 1968 23rd ACM National Conference*, ser. ACM '68. New York, NY, USA: ACM, 1968, pp. 485–490.

[42] P. A. Businger, "NSEVB - Eigenvalues and Eigenvectors of Nonsymmetrlc Matrices," *Numerical Mathematics Computer Programs*, vol. 1, no. 1, 1968.

[43] J. J. Dongarra, "An interview with Jack J. Dongarra," Retrieved February 10, 2014, from http://history.siam.org/pdfs2/Dongarra_%20returned_SIAM_copy.pdf, 2004, conducted by Thomas Haigh for the SIAM History of Numerical Analysis and Scientific Computing Project.

[44] "LAPACK–Linear Algebra PACKage Release Notes," Retrieved February 10, 2014, from http://www.netlib.org/lapack, 2013.

[45] J. W. Backus and H. Herrick, "IBM 701 Speedcoding and other automatic programming systems," in *Proc. Symp. on Automatic Programming for Digital Computer*, 1954.

[46] C. Moler, "The Growth of MATLAB and The MathWorks over Two Decades," Retrieved April 8, 2014, from http://www.mathworks.com/tagteam/72887_92020v00Cleve_Growth_MATLAB_MathWorks_Two_Decades_Jan_2006.pdf, MathWorks, Tech. Rep., 2006.

[47] M. T. Heath, *Scientific Computing An Introductory Survey*, 2nd ed. McGraw Hill, 2002.

[48] S. J. Leon, "MATLAB," in *Handbook of Linear Algebra*, ser. Discrete Mathematics and its Applications, L. Hogben, Ed. Boca Raton, FL: Chapman and Hall/CRC, 2007.

[49] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1992.

[50] J. Kepner and J. Gilbert, Eds., *Graph Algorithms in the Language of Linear Algebra*. SIAM, 2011.

[51] J. Kepner, "Graphs and matrices," in *Graph Algorithms in the Language of Linear Algebra*, J. Kepner and J. Gilbert, Eds. SIAM, 2011.

[52] G. E. Forsythe, "Today's Computational Methods of Linear Algebra," *SIAM Review*, vol. 9, no. 3, pp. pp. 489–515, 1967.

[53] U. A. Rauhala, "Introduction to Array Algebra," *Photogrammetric Engineering & Remote Sensing*, vol. 46, no. 2, pp. 177–192, February 1980.

[54] K. Åhlander, "Einstein Summation for Multidimensional Arrays," *Computers and Mathematics with Applications*, vol. 44, pp. 1007–1017, 2002.

[55] A. P. Harrison and D. Joseph, "Maximum Likelihood Estimation of Depth Maps Using Photometric Stereo," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1368–1380, 2012.

[56] L. Grady, "Random Walks for Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 11, pp. 1768–1783, Nov. 2006.

[57] S. C. Chapra and R. Canale, *Numerical Methods for Engineers.* New York, NY, USA: McGraw-Hill, Inc., 2006.

[58] R. J. Schilling and S. L. Harris, *Applied Numerical Methods for Engineers Using MATLAB and C.* Pacific Grove, CA: Brooks/Cole, 2000.

[59] H. V. Henderson and S. R. Searle, "The vec-permutation matrix, the vec operator and Kronecker products: a review," *Linear and Multilinear Algebra*, vol. 9, no. 4, pp. 271–288, 1981.

[60] D. G. Antzoulatos and A. A. Sawchuk, "Hypermatrix Algebra: Theory," *CVGIP: Image Understanding*, vol. 57, pp. 24–41, January 1993.

[61] R. A. Snay, "Applicability of Array Algebra," *Reviews of Geophysics and Space Physics*, vol. 16, no. 3, pp. 459–464, 1978.

[62] G. Blaha, "A Few Basic Principles and Techniques of Array Algebra," *Bulletin God-sique*, vol. 51, no. 3, pp. 177–202, 1977.

[63] J. R. Magnus and H. Neudecker, *Matrix Differential Calculus with Applications in Statistics and Econometrics*, 3rd ed., ser. Wiley Series in Probability and Statistics. Chichester: John Wiley & Sons, 2007.

[64] D. Pollock, "Tensor products and matrix differential calculus," *Linear Algebra and its Applications*, vol. 67, pp. 169–193, 1985.

[65] M. Suzuki and K. Shimizu, "Analysis of distributed systems by array algebra," *International Journal of Systems Science*, vol. 21, no. 1, pp. 129–155, 1990.

[66] W. J. Vetter, "The Array Matrix Generalization for Signal Processing," in *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '86.*, vol. 11, 1986, pp. 297–300.

[67] K. Åhlander and K. Otto, "Software design for finite difference schemes based on index notation," *Future Gener. Comput. Syst.*, vol. 22, pp. 102–109, January 2006.

[68] K. Otto, "A unifying framework for preconditioners based on fast transforms," Department of Scientific Computing, Uppsala University, Tech. Rep., 1999.

[69] J. G. Papastavridis, *Tensor Calculus and Analytical Dynamics.* CRC Press LLC, 1999.

[70] A. H. Barr, "The Einstein Summation Notation: Introduction to Cartesian Tensors and Extensions to the Notation," California Institute of Technology, Tech. Rep., (retreived 4/01/2011).

[71] E. R. Bolton, "A simple notation for differential vector expressions in orthogonal curvilinear coordinates," *Geophysical Journal International*, vol. 115, pp. 654–666, 1993.

[72] H. Lev-Ari, "Efficient solution of linear matrix equations with applications to multistatic antenna array processing," *Communications in Information and Systems*, vol. 5, no. 1, pp. 123–130, 2005.

[73] S. Liu and G. Trenkler, "Hadamard, Khatri-Rao, Kronecker, and Other Matrix Products," *International Journal of Information and Systems Sciences*, vol. 4, no. 1, pp. 160–177, 2008.

[74] H. J. Stetter, *Numerical Polynomial Algebra.* Philadelphia: Society for Industrial and Applied Mathematics, 2004.

[75] D. A. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, 3rd ed., ser. Undergraduate Texts in Mathematics. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[76] A. Sommese and C. Wampler, *The Numerical Solution of Systems of Polynomials Arising in Engineering and Science.* World Scientific, 2005.

[77] "DARPA Mathematical Challenges," Defense Sciences Office, Defense Advanced Research Projects Agency, Tech. Rep. DARPA-BAA 08-65, 2008.

[78] P. Comon, G. H. Golub, L.-H. Lim, and B. Mourrain, "Symmetric tensors and symmetric tensor rank." *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 3, pp. 1254–1279, 2008. [Online]. Available: http://dx.doi.org/10.1137/060661569

[79] P. Comon and B. Mourrain, "Decomposition of quantics in sums of powers of linear forms," *Signal Processing*, vol. 53, pp. 93–107, 1996.

[80] L. de Lathauwer and B. D. Moor, "From matrix to tensor: Multilinear algebra and signal processing," in *Mathematics in Signal Processing IV*, ser. IMA Conference Series. Oxford, 1997.

[81] L. D. Lathauwer, B. D. Moor, and J. Vandewalle, "A Multilinear Singular Value Decomposition," *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.

[82] C. J. Hillar and L.-H. Lim, "Most Tensor Problems Are NP-Hard," *Journal of the ACM*, vol. 60, no. 6, pp. 1–39, 2013.

[83] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM REVIEW*, vol. 51, no. 3, pp. 455–500, 2009.

[84] L.-H. Lim, "Tensors and Hypermatices," in *Handbook of Linear Algebra*, 2nd ed., ser. Discrete Mathematics and Its Applications, L. Hogben, Ed. Boca Raton, FL: Chapman and Hall/CRC, 2013, ch. 15.

[85] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Baltimore, MD, USA: The Johns Hopkins University Press, 1996.

[86] R. A. Harshman, "An index formalism that generalises the capabilities of matrix notation and algebra to n-way arrays," *Journal of Chemometrics*, vol. 15, pp. 689–714, 2001.

[87] S. Milgram, L. Bickman, and L. Berkowitz, "Note on the Drawing Power of Crowds of Different Size," *Journal of Personality and Social Psychology*, vol. 13, no. 2, pp. 79–82, 1969.

[88] R. Brualdi and J. Csima, "Small Matrices of Large Dimension," *Linear Algebra and its Applications*, vol. 150, pp. 227–241, 1991.

[89] G. Beylkin and M. J. Mohlenkamp, "Algorithms for Numerical Analysis in High Dimensions," *SIAM Journal on Scientific Computing*, vol. 26, no. 6, pp. 2133–2159, 2005.

[90] A. Limache and P. R. Fredini, "LTensor: A high performance C++ Tensor Library based on Index Notation," Retrieved August 6, 2013, from http://code.google.com/p/ltensor/, 2013.

[91] G. Baumgartner, A. Auer, D. Bernholdt, A. Bibireata, V. Choppella, D. Cociorva, X. Gao, R. Harrison, S. Hirata, S. Krishnamoorthy, S. Krishnan, C. Lam, Q. Lu,

M. Nooijen, R. Pitzer, J. Ramanujam, P. Sadayappan, and A. Sibiryakov, "Synthesis of high-performance parallel programs for a class of ab initio quantum chemistry models," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 276–292, Feb 2005.

[92] E. A. Auer, G. Baumgartner, D. E. Bernholdt, A. Bibireata, D. Cociorva, X. Gao, R. Harrison, S. Krishnamoorthy, H. Krishnan, C. chung Lam, Q. Lu, M. Nooijen, R. Pitzer, J. Ramanujam, P. Sadayappan, and E. Sibiryakov, "Automatic code generation for many-body electronic structure methods: The tensor contraction engine," *Molecular Physics*, 2005.

[93] A. Hartono, Q. Lu, T. Henretty, S. Krishnamoorthy, H. Zhang, G. Baumgartner, D. E. Bernholdt, M. Nooijen, R. Pitzer, J. Ramanujam, and P. Sadayappan, "Performance optimization of tensor contraction expressions for many-body methods in quantum chemistry," *The Journal of Physical Chemistry A*, vol. 113, no. 45, pp. 12 715–12 723, 2009.

[94] E. Epifanovsky, M. Wormit, T. Kus, A. Landau, D. Zuev, K. Khistyaev, P. Manohar, I. Kaliman, A. Dreuw, and A. I. Krylov, "New implementation of high-level correlated methods using a general block tensor library for high-performance electronic structure calculations." *Journal of Computational Chemistry*, vol. 34, no. 26, pp. 2293–2309, 2013.

[95] E. Solomonik, D. Matthews, J. R. Hammond, J. F. Stanton, and J. Demmel, "A massively parallel tensor contraction framework for coupled-cluster computations," *Journal of Parallel and Distributed Computing*, vol. 74, no. 12, pp. 3176 – 3190, 2014, domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[96] M. A. O. Vasilescu and D. Terzopoulos, "A Tensor Algebraic Approach to Image Synthesis, Analysis and Recognition," in *Proceedings of the Sixth International Conference on 3-D Digital Imaging and Modeling*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 3–12.

[97] ——, "Multilinear projection for appearance-based recognition in the tensor framework," in *Proc. Eleventh IEEE International Conference on Computer Vision (ICCV'07)*, 2007, pp. 1–8.

[98] M. A. O. Vasilescu, "A multilinear (tensor) algebraic framework for computer graphics, computer vision, and machine learning," Ph.D. dissertation, University of Toronto, 2009.

[99] L. T. Milov, "Multidimensional Matrix Derivatives and Sensitivity Analysis of Control Systems," *Automation and Remote Control*, vol. 40, no. 9, pp. 1269–1277, 1979.

[100] W. Vetter and M. Porsani, "Extended matrix formulation for the marple algorithm," in *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '87.*, vol. 12, Apr 1987, pp. 340–343.

[101] W. Landry, "Implementating a high performance tensor library," *Scientific Programming*, vol. 11, no. 4, pp. 273–290, 2003.

[102] H. Neudecker and S. Liu, "Some statistical properties of Hadamard products of random matrices," *Statistical Papers*, vol. 42, pp. 475–487, 2001.

[103] ——, "Statistical properties of Hadamard products of random vectors," *Statistical Papers*, vol. 42, pp. 529–533, 2001.

[104] D. Pollock, "On Kronecker Products, Tensor Products And Matrix Differential Calculus," University of Leicester, Tech. Rep. Working Paper No. 11/34, 2011.

[105] R. De Virgilio and F. Milicchio, "Rfid data management and analysis via tensor calculus," in *Transactions on Large-Scale Data- and Knowledge-Centered Systems VII*, ser. Lecture Notes in Computer Science, A. Hameurlain, J. Kng, and R. Wagner, Eds. Springer Berlin Heidelberg, 2012, vol. 7720, pp. 1–30.

[106] A. Takemura, "Tensor analysis of anova decomposition," *Journal of the American Statistical Association*, vol. 78, no. 384, pp. 894–900, 1983.

[107] P. McCullagh, *Tensor Methods in Statistics*, ser. Monographs on Statistics and Applied Probability. London: Chapman and Hill, 1987.

[108] C. F. van Loan, "The ubiquitous Kronecker product," *Journal of Computational and Applied Mathematics*, vol. 123, pp. 85–100, November 2000.

[109] W.-H. Steeb and Y. Hardy, *Matrix Calculus and Kronecker Product*, 2nd ed. New Jersey: World Scientific, 2011.

[110] H. A. L. Kiers, "Towards a standardized notation and terminology in multiway analysis," *Journal of Chemometrics*, vol. 14, no. 105–122, 2000.

[111] L. Eldén and B. Savas, "A Newton-Grassmann Method for Computing the Best Multilinear Rank-($r_1\, r_2\, r_3$) Approximation of a Tensor," *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 2, pp. 248–271, Mar. 2009.

[112] B. W. Bader and T. G. Kolda, "Algorithm 862: MATLAB Tensor Classes for Fast Algorithm Prototyping," *ACM Transactions on Mathematical Software*, vol. 32, no. 4, pp. 635–653, 2006.

[113] G. R. Tait, "The Array-Matrix Concept—A New Approach to Multivariate Analysis," Ph.D. dissertation, McGill University, Montreal, Canada, 1971.

[114] U. A. Rauhala, "Array Algebra Expansion of Matrix and Tensor Calculus: Part 1," *SIAM Journal of Matrix Analysis and Applications*, vol. 24, no. 2, pp. 490–508, 2002.

[115] J. A. Eisele and R. M. Mason, *Applied Matrix and Tensor Analysis.* New York: Wiley-Interscience, 1970.

[116] P. A. Regalia and M. K. Sanjit, "Kronecker products, unitary matrices, and signal processing applications," *SIAM Review*, vol. 31, pp. 586–613, December 1989.

[117] J. R. Magnus, "On the concept of matrix derivative," *Journal of Multivariate Analysis*, vol. 101, no. 9, pp. 2200–2206, 2010.

[118] R. A. Harshman and S. Hong, "'Stretch' vs 'slice' methods for representing three-way structure via matrix notation," *Journal of Chemometrics*, vol. 16, pp. 198–205, 2002.

[119] C. D. Meyer, *Matrix Analysis and Applied Linear Algebra.* Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000.

[120] M. Brazell, N. Li, C. Navasca, and C. Tamon, "Solving Multilinear Systems via Tensor Inversion," *SIAM Journal on Matrix Analysis and Applications*, vol. 34, no. 2, pp. 542–570, 2013.

[121] J. R. Magnus, *Linear Structures.* London: Charles Griffin & Company Limited, 1988.

[122] K. Braman, "Third-order tensors as linear operators on a space of matrices," *Linear Algebra and its Applications*, vol. 433, pp. 1241–1253, 2010.

[123] D. Joseph, "Modelling and calibration of logarithmic CMOS image sensors," Ph.D. dissertation, University of Oxford, 2002.

[124] B. W. Bader and T. G. Kolda, "Efficient matlab computations with sparse and factored tensors," *SIAM Journal of Scientific Computing*, vol. 30, pp. 205–231, 2007.

[125] T. Veldhuizen, "Blitz++," Retrieved February 6, 2012, from http:://www.oonumerics.org/blitz/, 2005.

[126] D. Abrahams and A. Gurtovoy, *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond*, ser. C++ in Depth Series. Boston, USA: Addison-Wesley Professional, 2004.

[127] A. Limachea and P. R. Fredinib, "A tensor library for scientific computing," *Mecnica Computacional*, vol. XXVII, pp. 2907–2925, 2008.

[128] J. C. Cummings, J. A. Crotinger, S. W. Haney, W. F. Humphrey, S. R. Karmesin, J. V. Reynders, S. A. Smith, and T. J. Williams, "Rapid Application Development and

Enhanced Code Interoperability using the POOMA Framework," in *Object Oriented Methods for Interoperable Scientific and Engineering Computing: Proceedings of the 1998 SIAM Workshop*, M. E. Henderson, C. R. Anderson, and S. L. Lyons, Eds. SIAM, 1999.

[129] "NumPy," Retrieved March 18, 2014, from http://www.numpy.org/.

[130] W.-H. Steeb, *Matrix Calculus and Kronecker Product with Applications and C++ Programs.* World Scientific, 1997.

[131] T. E. Oliphant, *Guide to NumPy*, 2006.

[132] R. Bellman, *Adaptive Control Processes: A Guide Tour.* Princeton University Press, 1961.

[133] J. K. Ousterhout, "Scripting: Higher Level Programming for the 21st Century," *Computer*, vol. 31, no. 3, pp. 23–30, 1998.

[134] T. E. Oliphant, "Python for Scientific Computing," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 10–20, 2007.

[135] H. P. Langtangen, *Python Scripting for Computational Science*, 3rd ed., ser. Texts in Computational Science and Engineering. Berlin: Springer, 2008.

[136] L. D. Paulson, "Developers Shift to Dynamic Programming Languages," *Computer*, vol. 40, no. 2, pp. 12–15, 2007.

[137] L. Prechelt, "Are Scripting Languages Any Good? A Validation of Perl, Python, Rexx, and Tcl against C, C++, and Java," in *Information Repositories*, ser. Advances in Computers, M. Zelkowitz, Ed. Academic Press, 2003, vol. 57, pp. 205–270.

[138] D. Garey and S. Lang, "High Performance Development with Python," *Scientific Computing*, pp. 10–14, 2008.

[139] S. O'Grady, "The RedMonk Programming Language Rankings: January 2014," Retrieved May 4, 2014, from http://redmonk.com/sogrady/2014/01/22/language-rankings-1-14/, 2014.

[140] "TIOBE Index for MATLAB," Retrieved May 4, 2014 from http://www.tiobe.com/index.php/content/paperinfo/tpci/MATLAB.html, 2014.

[141] A. van Deursen, P. Klint, and J. Visser, "Domain-specific Languages: An Annotated Bibliography," *ACM SIGPLAN Notices*, vol. 35, no. 6, pp. 26–36, Jun. 2000.

[142] T. Veldhuizen, "Techniques for scientific c++," Retrieved Sept. 1, 2015, from https://www.cs.indiana.edu/cgi-bin/techreports/TRNNN.cgi?trnum=TR542, Indiana University, Tech. Rep. 542, 2000.

[143] A. Alexandrescu, *Modern C++ Design: Generic Programming and Design Patterns Applied*, ser. C++ in Depth Series. Pearson Education Inc., 2001.

[144] C. Sanderson, "Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments," NICTA, Australia, Tech. Rep., 2010.

[145] "The Eigen Library," Retrieved Sept. 1, 2015, from http://eigen.tuxfamily.org.

[146] W. Landry, "The FTensor Library," Retrieved February 6, 2012, from http://www.gps.caltech.edu/~walter/FTensor/, 2004.

[147] D. Abrahams and R. W. Grosse-Kunstleve, "Building Hybrid Systems with Boost.Python," Retrieved April 16, 2014, from http://www.boost.org/doc/libs/1_31_0/libs/python/doc/PyConDC_2003/bpl.html, Boost Consulting, Tech. Rep., 2003.

[148] J. Marcum, *Thomas Kuhn's Revolution: An Historical Philosophy of Science*, ser. Continuum Studies in American Philosophy Series. Continuum, 2005.

[149] G. Golub and V. Pereyra, "Separable nonlinear least squares: the variable projection method and its applications," *Inverse Problems*, vol. 19, pp. R1–R26, 2003.

[150] R.-G. Chang, T.-R. Chuang, and J. K. Lee, "Parallel Sparse Supports for Array Intrinsic Functions of Fortran 90," *The Journal of Supercomputing*, vol. 18, no. 3, pp. 305–339, 2001.

[151] C.-Y. Lin, J.-S. Liu, and Y.-C. Chung, "Efficient Representation Scheme for Multidimensional Array Operations," *IEEE Transactions on Computers*, vol. 51, no. 3, pp. 327–345, Mar. 2002.

[152] C.-Y. Lin, Y.-C. Chung, and J.-S. Liu, "Efficient Data Compression Methods for Multidimensional Sparse Array Operations Based on the EKMR Scheme," *IEEE Transactions on Computers*, vol. 52, no. 12, pp. 1640 – 1646, dec. 2003.

[153] M. D. Schatz, T.-M. Low, R. A. van de Geijn, and T. G. Kolda, "Exploiting Symmetry in Tensors for High Performance: Multiplication with Symmetric Tensors," *SIAM Journal on Scientific Computing*, vol. 36, no. 5, pp. C453–C479, September 2014.

[154] D. Pollock, "On Kronecker Products, Tensor Products And Matrix Differential Calculus," *International Journal of Computer Mathematics*, vol. 90, no. 11, pp. 2462–2476, 2013.

[155] W. J. Vetter, "Matrix Calculus Operations and Taylor Expansions," *SIAM Review*, vol. 15, no. 2, pp. 352–369, 1973.

[156] ——, "Vector structures and solutions of linear matrix equations," *Linear Algebra and its Applications*, vol. 10, no. 2, pp. 181–188, 1975.

[157] H. Jeffreys, *Cartesian Tensors*. London: Cambridge University Press, 1963.

[158] N. O. Myklestad, *Cartesian Tensors: The Mathematical Language of Engineering*, ser. University Series in Applied Mechanics. Princeton, New Jersey: D. Van Nostrand Company, Inc., 1967.

[159] A. M. Goodbody, *Cartesian Tensors: With Applications to Mechanics, Fluid Mechanics and Elasticity*, ser. Ellis Horwood Series in Mathematics and its Applications. West Sussex, England: Ellis Horwood Limited, 1982.

[160] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Min. Knowl. Discov.*, vol. 2, no. 2, pp. 121–167, Jun. 1998.

[161] R. B. Schnabel and P. D. Frank, "Tensor methods for nonlinear equations," *SIAM Journal on Numerical Analysis*, vol. 21, no. 5, pp. 815–843, 1984.

[162] J. Brewer, "Kronecker Products and Matrix Calculus in System Theory," *Circuits and Systems, IEEE Transactions on*, vol. 25, no. 9, pp. 772 – 781, sep 1978.

[163] A. P. Harrison and D. Joseph, "Translational photometric alignment of single-view image sequences," *Computer Vision and Image Understanding*, vol. 116, no. 6, pp. 765–776, 2012.

[164] ——, "Depth-Map and Albedo Estimation with Superior Information-Theoretic Performance," in *Image Processing: Machine Vision Applications VIII*, ser. Proceedings of the SPIE, E. Y. Lam and K. S. Niel, Eds. SPIE, 2015, vol. 9405, pp. 94 050C–94 050C–15.

[165] G. Golub and V. Pereyra, "The Differentation of Pseudo-Inverses and the Nonlinear Least Squares Problems Whose Variables Separate," *SIAM Journal on Numerical Analysis*, vol. 10, no. 2, pp. 413–432, 1973.

[166] "Open Source Initiative OSI - The BSD 3-Clause License," Retrieved May 25, 2012, from http://www.opensource.org/licenses/bsd-3-clause.

[167] "Doxygen," Retrieved April 13, 2012, from www.doxygen.org.

[168] J. de Guzman, D. Marsden, and T. Heller, "Phoenix 3.0," Retrieved February 15, 2012, from http://www.boost.org/doc/libs/1_48_0/libs/phoenix/doc/html/index.html.

[169] "The Boost Library," Retrieved February 6, 2012, from http://www.boost.org/.

[170] M. A. Akcoglu, P. F. Bartha, and D. M. Ha, *Analysis in Vector Spaces.* Wiley, 2009.

[171] F. Gustavson and T. Swirszcz, "In-place transposition of rectangular matrices," in *Applied Parallel Computing. State of the Art in Scientific Computing*, ser. Lecture Notes in Computer Science, B. Kgstrm, E. Elmroth, J. Dongarra, and J. Wasniewski, Eds. Springer Berlin Heidelberg, 2007, vol. 4699, pp. 560–569. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-75755-9_68

[172] C. H. Ding, "An Optimal Index Reshuffle Algorithm for Multidimensional Arrays and Its Applications for Parallel Architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 3, pp. 306–315, 2001.

[173] Y. Jie, W. Jian-ping, and W. Zheng-hua, "A High Efficient In-place Transposition Scheme for Multidimensional Arrays," in *2011 Fourth International Conference on Information and Computing*, vol. 1, 2010, pp. 158–161.

[174] N. Brenner, "Algorithm 467: Matrix transposition in place," *Communications of the ACM*, vol. 16, no. 11, pp. 692–694, 1973.

[175] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: A structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, March 2011.

[176] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, "GigaTensor: Scaling Tensor Analysis Up by 100 Times - Algorithms and Discoveries," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '12. New York, NY, USA: ACM, 2012, pp. 316–324.

[177] M. Baskaran, B. Meister, N. Vasilache, and R. Lethin, "Efficient and Scalable Computations with Sparse Tensors," in *High Performance Extreme Computing (HPEC), 2012 IEEE Conference on*, Sept 2012, pp. 1–6.

[178] I. V. Oseledets and E. E. Tyrtyshnikov, "Breaking the curse of dimensionality, or how to use svd in many dimensions," *SIAM J. Scientific Computing*, vol. 31, no. 5, pp. 3744–3759, 2009.

[179] I. V. Oseledets and S. V. Dolgov, "Solution of linear systems and matrix inversion in the tt-format," *SIAM J. Scientific Computing*, vol. 34, no. 5, 2012.

[180] B. N. Khoromskij, "Tensors-structured numerical methods in scientific computing: Survey on recent advances," *Chemometrics and Intelligent Laboratory Systems*, vol. 110, no. 1, pp. 1 – 19, 2012.

[181] G. Gundersen and T. Steihaug, "Sparsity in higher order methods for unconstrained optimization," *Optimization Methods and Software*, vol. 27, no. 2, pp. 275–294, 2012. [Online]. Available: http://dx.doi.org/10.1080/10556788.2011.597853

[182] T. A. Davis, *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2006.

[183] A. Buluç, J. Gilbert, and V. B. Shah, "Implementing Sparse Matrices for Graph Algorithms," in *Graph Algorithms in the Language of Linear Algebra*, J. Kepner and J. Gilbert, Eds. SIAM, 2011.

[184] E. A. Bender and S. G. Williamson, *Foundations of Applied Combinatorics*. Addison-Wesley, 1991.

[185] J. A. Parkhill and M. Head-Gordon, "A sparse framework for the derivation and implementation of fermion algebra," *Molecular Physics*, vol. 108, no. 3-4, pp. 513–522, 2010.

[186] J. Maddock and C. Kormanyos, "The Boost Multiprecision Library," Retrieved Sept. 1, 2015, from http://www.boost.org/doc/libs/1_59_0/libs/multiprecision/doc/html/index.html, 2015. [Online]. Available: http://www.boost.org/doc/libs/1_58_0/libs/multiprecision/doc/html/index.html

[187] GMP.Org, "The GNU Multiple Precision Arithmetic Library," Retrieved Sept. 1, 2015, from http://gmplib.org/, 2014. [Online]. Available: http://gmplib.org/

[188] A. LaMarca and R. E. Ladner, "The Influence of Caches on the Performance of Sorting," *Journal of Algorithms*, vol. 3, no. 1, pp. 66–104, 1999.

[189] U. Drepper, "What Every Programmer Should Know About Memory," Red Hat, Inc., Tech. Rep., 2007.

[190] C. Doras, "libdivide," Retrieved Sept. 1, 2015, from http://libdivide.com/, 2010. [Online]. Available: http://libdivide.com/

[191] D. Musser, "Introspective Sorting and Selection Algorithms," *Software Practice and Experience*, vol. 27, pp. 983–993, 1997.

[192] T. Peters, "timsort," Retrieved August 25, 2014, from http://bugs.python.org/file4451/timsort.txt, 2002.

[193] R. Sedgewick, *Algorithms in C++, Parts 1-4: Fundamentals, Data Structure, Sorting, Searching*, 3rd ed., ser. Algorithms in C++. Pearson Education, 1998, ch. Radix Sorting.

[194] V. J. Duvanenko, "In-place Hybrid N-bit-Radix Sort," *Dr. Dobb's*, 2009, retrieved Sept. 22, 2014. [Online]. Available: http://www.drdobbs.com/architecture-and-design/algorithm-improvement-through-performanc/221600153

[195] A. Buluç and J. Gilbert, "On the Representation and Multiplication of Hypersparse Matrices," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, April 2008, pp. 1–11.

[196] ——, "New Ideas in Sparse Matrix Matrix Multiplication," in *Graph Algorithms in the Language of Linear Algebra*, J. Kepner and J. Gilbert, Eds. SIAM, 2011.

[197] D. M. Dunlavy, T. G. Kolda, and W. P. Kegelmeyer, "Multilinear Algebra for Analyzing Data with Multiple Linkages," in *Graph Algorithms in the Language of Linear Algebra*, J. Kepner and J. Gilbert, Eds. SIAM, 2011.

[198] T. Kolda and J. Sun, "Scalable tensor decompositions for multi-aspect data mining," in *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on*, Dec 2008, pp. 363–372.

[199] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "Parcube: Sparse parallelizable tensor decompositions." in *ECML PKDD'12*, ser. Lecture Notes in Computer Science, P. A. Flach, T. D. Bie, and N. Cristianini, Eds., vol. 7523. Springer, 2012, pp. 521–536. [Online]. Available: http://dblp.uni-trier.de/db/conf/pkdd/pkdd2012-1.html#PapalexakisFS12

[200] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: A recursive model for graph mining," in *SIAM International Conference on Data Mining*, 2004.

[201] A. Buluç, J. Gilbert, and A. Lugowski, "CombBLAS," Retrieved April 15, 2015, from http://gauss.cs.ucsb.edu/~aydin/CombBLAS/html/.

[202] E. C. Chi and T. G. Kolda, "On Tensors, Sparsity, and Nonnegative Factorizations," *SIAM Journal on Matrix Analysis and Applications*, vol. 33, no. 4, pp. 1272–1299, December 2012.

[203] A. P. Harrison, N. Birkbeck, and M. Sofka, "IntellEditS: Intelligent Learning-Based Editor of Segmentations," in *Medical Image Computing and Computer-Assisted Intervention MICCAI 2013*, ser. Lecture Notes in Computer Science, K. Mori, I. Sakuma, Y. Sato, C. Barillot, and N. Navab, Eds. Springer Berlin Heidelberg, 2013, vol. 8151, pp. 235–242.

[204] G. Sapiro, *Geometric Partial Differential Equations and Image Analysis.* New York, NY, USA: Cambridge University Press, 2006.

[205] P. Perona and J. Malik, "Scale-Space and Edge Detection Using Anisotropic Diffusion," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 7, pp. 629–639, Jul. 1990. [Online]. Available: http://dx.doi.org/10.1109/34.56205

[206] S. Osher and L. I. Rudin, "Feature-oriented image enhancement using shock filters," *SIAM Journal on Numerical Analysis*, vol. 27, no. 4, pp. pp. 919–940, 1990.

[207] S. Bae, S. Paris, and F. Durand, "Two-scale Tone Management for Photographic Look," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 637–645, Jul. 2006.

[208] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed. Pearson Prentice Hall, 2008.

[209] S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework," *International Journal of Computer Vision*, vol. 56, no. 3, pp. 221–255, February 2004. [Online]. Available: http://dx.doi.org/10.1023/B:VISI.0000011205.11775.fd

[210] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, ser. CVPR '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 886–893.

[211] M. Nolden, S. Zelzer, A. Seitel, D. Wald, M. Mller, A. Franz, D. Maleike, M. Fangerau, M. Baumhauer, L. Maier-Hein, K. Maier-Hein, H.-P. Meinzer, and I. Wolf, "The medical imaging interaction toolkit: challenges and advances," *International Journal of Computer Assisted Radiology and Surgery*, vol. 8, no. 4, pp. 607–620, 2013. [Online]. Available: http://dx.doi.org/10.1007/s11548-013-0840-8

[212] N. R. Pal and S. K. Pal, "A review on image segmentation techniques," *Pattern Recognition*, vol. 26, no. 9, pp. 1277 – 1294, 1993. [Online]. Available: http://www.sciencedirect.com/science/article/pii/003132039390135J

[213] L. Grady, "Multilabel random walker image segmentation using prior models," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, June 2005, pp. 763–770 vol. 1.

[214] L. Grady and G. Funka-Lea, "An energy minimization approach to the data driven editing of presegmented images/volumes," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2006*. Springer, 2006, pp. 888–895.

[215] W. Yang, J. Cai, J. Zheng, and J. Luo, "User-friendly Interactive Image Segmentation Through Unified Combinatorial User Inputs," *IEEE Transactions on Image Processing*, vol. 19, no. 9, pp. 2470 –2479, 2010.

[216] J. W. Eaton, D. Bateman, and S. Hauberg, *GNU Octave Manual Version 3*. Network Theory Ltd., 2008.

[217] A. Georghiades, P. Belhumeur, and D. Kriegman, "From few to many: Illumination cone models for face recognition under variable lighting and pose," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 6, pp. 643–660, 2001.

[218] A. P. Harrison, C. Wong, and D. Joseph, "Virtual Reflected-Light Microscopy," *Journal of Microscopy*, vol. 244, pp. 293–304, 2011.

[219] C. Boncelet, "Image Noise Models," in *The Essential Guide to Image Processing*, second edition ed., A. Bovik, Ed. Boston: Academic Press, 2009, pp. 143 – 167.

[220] G. A. Seber and C. J. Wild, *Linear Regression Analysis*, 2nd ed. John Wiley & Sons, Inc., 2003.

[221] A. K. Agrawal, R. Raskar, and R. Chellappa, "What is the range of surface reconstructions from a gradient field?" in *Proceedings of the 9th European conference on Computer Vision*, ser. ECCV'06, vol. 1. Springer-Verlag, 2006, pp. 578–591.

[222] J.-D. Durou, M. Falcone, and M. Sagona, "Numerical Methods for Shape-from-shading: A New Survey with Benchmarks," *Computer Vision and Image Understanding*, vol. 109, no. 1, pp. 22 – 43, 2008.

[223] G. A. Seber and C. J. Wild, *Nonlinear Regression*. John Wiley & Sons, Inc., 1989.

[224] K. P. Burnham and D. R. Anderson, *Model Selection and Multimodel Inference*, 2nd ed. New York: Springer-Verlag, 2002.

[225] ——, "Multimodel Inference: Understanding AIC and BIC in Model Selection," *Sociological Methods & Research*, vol. 33, no. 2, pp. 261–304, 2004.

[226] E. Sober, "Instrumentalism, Parsimony, and the Akaike Framework," *Philosophy of Science*, vol. 69, pp. S112–S123, 2002.

[227] L. Noakes and R. Kozera, "Nonlinearities and noise reduction in 3-source photometric stereo," *Journal of Mathematical Imaging and Vision*, vol. 18, no. 2, pp. 119–127, 2003.

[228] ——, "Denoising images: Non-linear leap-frog for shape and light-source recovery," in *Geometry, Morphology, and Computational Imaging*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2003, vol. 2616, pp. 143–162.

[229] T. Cameron, R. Kozera, and A. Datta, "A parallel leap-frog algorithm for 3-source photometric stereo," in *Computer Vision and Graphics: International Conference, ICCVG 2004*, ser. Computational Imaging and Vision. Springer, 2006, pp. 95–102.

[230] O. Ikeda, "Synthetic Shape Reconstruction Combined with the FT-Based Method in Photometric Stereo," in *ISVC 2010*, ser. Lecture Notes in Computer Science, G. Bebis, R. D. Boyle, B. Parvin, D. Koracin, R. Chung, R. I. Hammoud, M. Hussain, K.-H. Tan, R. Crawfis, D. Thalmann, D. Kao, and L. Avila, Eds., vol. 6453. Springer, 2010, pp. 678–687.

[231] ——, "Photometric Stereo Using Four Surface Normal Approximations and Optimal Normalization of Images," in *Signal Processing and Information Technology, 2006 IEEE International Symposium on*, 2006, pp. 672–679.

[232] R. Zhang, P.-S. Tsai, J. Cryer, and M. Shah, "Shape from Shading: A Survey," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 21, no. 8, pp. 690–706, Aug 1999.

[233] A. N. Whithead, *An Introduction to Mathematics.* New York: Henry Holt and Company, 1911.

[234] M. Schiffer and L. Bowden, *The Role of Mathematics in Science*, A. Lax, Ed. The Mathematical Society of America, 1984.

[235] C. Prada, S. Manneville, D. Spoliansky, and M. Fink, "Decomposition of the time reversal operator: Detection and selective focusing on two scatterers," *Journal of the Acoustical Society of America*, vol. 99, no. 4, pp. 2067–2076, 1996.

[236] L.-H. Lim, "Tensors and Hypermatices," Retrieved March 12, 2014, from http://www.stat.uchicago.edu/~lekheng/work/tensors.pdf, University of Chicago, Tech. Rep., 2013.

[237] D. Kats and F. R. Manby, "Sparse tensor framework for implementation of general local correlation methods," *The Journal of Chemical Physics*, vol. 138, no. 14, 2013. [Online]. Available: http://scitation.aip.org/content/aip/journal/jcp/138/14/10.1063/1.4798940

[238] J. M. Tang and Y. Saad, "Domain-decomposition-type methods for computing the diagonal of a matrix inverse." *SIAM J. Scientific Computing*, vol. 33, no. 5, pp. 2823–2847, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/siamsc/siamsc33.html#TangS11

[239] S. Li, S. Ahmed, G. Klimeck, and E. Darve, "Computing entries of the inverse of a sparse matrix using the {FIND} algorithm," *Journal of Computational Physics*, vol. 227, no. 22, pp. 9408 – 9427, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0021999108003458

[240] L. Lin, J. Lu, L. Ying, R. Car, and W. E, "Fast algorithm for extracting the diagonal of the inverse matrix with application to the electronic structure analysis of metallic systems," *Commun. Math. Sci.*, vol. 7, no. 3, pp. 755–777, 09 2009. [Online]. Available: http://projecteuclid.org/euclid.cms/1256562822

[241] L. Lin, C. Yang, J. C. Meza, J. Lu, L. Ying, and W. E, "Selinv—an algorithm for selected inversion of a sparse symmetric matrix," *ACM Trans.*

Math. Softw., vol. 37, no. 4, pp. 40:1–40:19, Feb. 2011. [Online]. Available: http://doi.acm.org/10.1145/1916461.1916464

[242] K. Batselier, "A Numerical Linear Algebra Framework for Solving Problems with Multivariate Polynomials," Ph.D. dissertation, KU Leuven, Leuven, Belgium, 2013.

[243] P. Dreesen, "Back to the Roots: Polynomial Systems Solving Using Linear Algebra," Ph.D. dissertation, KU Leuven, Leuven, Belgium, 2013.

[244] P. Dreesen, K. Batselier, and B. L. D. Moor, "Back to the Roots: Polynomial System Solving, Linear Algebra, Systems Theory," in *16th IFAC Symposium on System Identification*, 2012.

[245] J. Landsberg, *Tensors: Geometry and Applications*, ser. Graduate studies in mathematics. American Mathematical Society, 2012.

[246] H. S. Warren, *Hackers Delight*, 2nd ed. Addison-Wesley Professional, 2013.

[247] B. Chandramouli and J. Goldstein, "Patience is a Virtue: Revisiting Merge and Sort on Modern Processors," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. New York, NY, USA: ACM, 2014, pp. 731–742.

[248] S. Meyers, *Effective STL: 50 Specific Ways to Improve Your Use of the Standard Template Library*. Addison-Wesley Professional, 2001.

[249] P. Tsigas and Y. Z. 0004, "A Simple, Fast Parallel Implementation of Quicksort and its Performance Evaluation on SUN Enterprise 10000," in *Proceedings of the 11th Euromicro Conference on Parallel Distributed and Network based Processing*. IEEE Computer Society, 2003, pp. 372–384.

[250] K. Schwarz, "An implementation of the introsort algorithm, a fast hybrid of quicksort, heapsort, and insertion sort," 2010. [Online]. Available: http://www.keithschwarz.com/interesting/code/?dir=introsort

[251] R. Sedgewick, "Implementing Quicksort Programs," *Communications of the ACM*, vol. 21, no. 10, pp. 847–857, 1978.

[252] A. Martelli, *Python in a Nutshell*, 2nd ed. Cambridge: O'Reilly Media, Inc., 2006.

[253] J. Chhugani, A. D. Nguyen, V. W. Lee, W. Macy, M. Hagog, Y.-K. Chen, A. Baransi, S. Kumar, and P. Dubey, "Efficient Implementation of Sorting on Multi-core SIMD CPU Architecture," *The Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1313–1324, Aug. 2008.

[254] N. Satish, C. Kim, J. Chhugani, A. D. Nguyen, V. W. Lee, D. Kim, and P. Dubey, "Fast sort on cpus and gpus: A case for bandwidth oblivious simd sort," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 351–362.

[255] F. Goro, "A C++ implementation of timsort," Retrieved August 25, 2014, from https://github.com/gfx/cpp-TimSort.

[256] J. Wassenberg and P. Sanders, "Engineering a Multi-core Radix Sort," in *Euro-Par 2011 Parallel Processing*, ser. Lecture Notes in Computer Science, E. Jeannot, R. Namyst, and J. Roman, Eds. Springer Berlin Heidelberg, 2011, vol. 6853, pp. 160–169.

[257] M. Herf, "Radix Tricks," Retrieved August 26, 2014, from http://stereopsis.com/radix.html, 2001.

Appendices

Appendix A

Details on Sparse Software

In this appendix we provide some additional details regarding sparse numeric tensor (NT) computations that are not covered in the main body of the thesis. These consist of a discussion on fast linearised index (LI) calculations, which are highly important for fast sparse NT computations. We follow this up by providing more detailed descriptions of the sorting algorithm implementations discussed in Section 5.2.1. A discussion on computing abelian operations follows. Finally, we outline how LibNT represents the result of operations between dense and sparse NTs.

A.1 Fast Linearised Index Calculations

While it is advantageous to avoid recomputing LIs as much as possible, they remain an important and frequent operation within NT computations. For this reason, it is crucial to make these calculations as fast as possible to minimise their impact on runtime efficiency. As it turns out, LI calculations can be a significant time sink, making it worthwhile to exploit fast methods.

A straightforward means to recompute LI values into a new lexicographical precedence is to first expand the linearised value into its constituent parts using a function like that in Figure A.1.

```
EXPAND_LI
Input:  LI linearised integer index
        DIMS array of integer index ranges
Output: INDS array of expanded integer indices
Begin:

    for (i=0;i<size(DIMS);i=i+1)
        INDS[i]=mod(LI,DIMS[i])
        LI=LI/DIMS[i] //integer division
    end_for

End:
```

Figure A.1: Expanding an LI into its constituent parts.

```
COMPUTE_LI
Input:  INDS array of expanded integer indices
        DIMS array of integer index ranges
Output: LI linearised integer index
Begin:
    LI-0
    MULTIPLIER-1
    for (i-0;i<size(DIMS);i-i+1)
        LI-MULTIPLIER*INDS[i]
        MULTIPLIER*-DIMS[i]
    end_for

End:
```

Figure A.2: Computing an LI from its expanded indices.

```
RECOMPUTE_LI
Input:  LI linearised integer index in OLD lexicographical order
        DIMS array of integer index ranges
        OLD old lexicographical order, e.g., {0,1,2,3} for last-to-first
                4th-degree lexicographical order
        NEW new lexicographical order, e.g., {3,2,1,0} for first-to-last
                4th-degree order precedence
Output: LI linearised integer index in new lexicographical order
Begin:

    INDS-EXPAND_LI(LI,DIMS(OLD)) //expand indices based on OLD lexicographical order
    INDS(OLD)-INDS;  //shuffle indices to default lexicographical order, i.e., {0,1,2,3}
    LI-COMPUTE_LI(INDS(NEW),DIMS(NEW)) //linearise to NEW lexicographical order

End:
```

Figure A.3: Recomputing an LI from an old to new lexicographical order.

With the LI expanded, its constituent parts can be rearranged and re-linearised using a function like that of Figure A.2. Putting it all together, recomputing an LI into a new lexicographical order can be accomplished using a routine like Figure A.3.

While straightforward, the re-computation scheme in Figure A.3 can consume a significant amount of time. To illustrate this, Table A.1 displays experiment results measuring the time taken to recompute the LIs of a large fourth-degree NT. This can be contrasted with the time taken to sort the non-zero linearised coordinate (LCO) data after LI values have been recomputed. As can be seen in the experiment, the LI re-computation, which is an $O(nnz)$ operation, consumes roughly a third as much time as the $O(nnz \log(nnz))$ sort. This is clearly an unacceptable amount of time for what should be a preprocessing step.

A major source of the bottleneck in Figure A.3 is the expansion of an LI into an array of indices and back again into a new LI value. This bottleneck can be avoided by recomputing LIs without the intermediate array. This process can be further accelerated by pre-computing the divisors and multiplicands used to expand and compute LIs respectively. Figure A.4 depicts pseudocode that accomplishes this. As Table A.1 demonstrates, the speedup of using the direct routine is significant, almost halving the time required to recompute LIs. Nonetheless, further work is required as the total time taken remains

Table A.1: Speed improvements garnered by fast LI calculations. Improvements in execution times of changing a sparse fourth-degree NTs's lexicographical order from from $\{0,1,2,3\}$ to $\{3,2,1,0\}$. The fourth-degree NT uses a range of 2^{10} for all four of its indices and holds $5 * 2^{20}$ non-zeros, providing it with equivalent sparsity to a fourth-degree $O(h^2)$ Jacobian operator. Optimisations are incrementally added to demonstrate the speed up in execution times. The time taken to sort the LCO data array using introsort [191] is also shown to provide context. Experiments were run 10 times and average values are shown.

Routine	Time (s)	Relative to Introsort
Figure A.3	0.323	0.330
Figure A.4	0.173	0.177
Figure A.4 + static `for` loop	0.170	0.174
Optimised integer division	0.0375	0.0383
Introsort	0.978	1

```
RECOMPUTE_LI_DIRECT
Input:  LI linearised integer index in
        DIVISORS array of divisors corresponding to input lexicographical order,
            e.g. {1,n0,n0*n1,n0*n1*n3} for 4th degree {0,1,2,3} lexicographical order
        MULTIPLICANDS array of new multiplicands to switch from old to new
            lexicographical order, e.g. {n4*n3*n2,n4*n3,n4,1} to switch
            4th degree {0,1,2,3} lexicographical order to {3,2,1,0}
            lexicographical order
Output: LI_NEW linearised integer index in new lexicographical order
Begin:

    LI_NEW=0
    for (i=size(DIVISORS)-1;i>=0;i=i-1)
        //use integer division to get current index
        QUOTIENT=LI/DIVISORS[i]
        //add index using new lexicographical order
        LI_NEW=LI_NEW+QUOTIENT*MULTIPLICANDS[i]
        //prime LI for next iteration
        LI=LI-QUOTIENT*DIVISORS[i]
    end_for

End:
```

Figure A.4: Recomputing an LI from an old to new lexicographical order without intermediate arrays. The `MULTIPLICANDS` input must be designed to switch from old to new lexicographical orders, *i.e.*, while the individual *values* in the `MULTIPLICANDS` array are dependant only upon the new lexicographical order, the *sequence* of the values in the array is based on how indices are shuffled from old-to-new lexicographical orders.

unacceptable, taking roughly 18% of time to sort non-zero values.

For languages that support explicit compile-time execution, *e.g.*, template metaprogramming (TMP) in C++, one option is turn the run-time `for` loop in Figure A.4 into a completely unrolled series of instructions. This is generally possible as the number of iterations in the `for` loop is based on the NT degree, which is typically known at compile time. Thus, the run-time overhead of the `for` loop can be avoided. To test this, the experiments also executed the routine in Figure A.4 but used a template-metafunction [126] that generically implements a compile-time `for` loop. The third row of Table A.1 depicts the result. As can be seen only very minor improvements in running time are realised. This suggests that that compiler optimisations on the testbed already optimised the small run-time `for` loop in Figure A.4. Thus, using statically unrolled loops provides minimal improvements in LI computation time for the experimental testbed and other strategies are needed. Nonetheless, some improvement was established, which may be more substantial on other machines. As a result, it is worthwhile retaining the static version of a `for` loop.

Realising faster LI recomputations requires optimising the most expensive arithmetic operation in the calculation, *i.e.*, integer division. One simple optimisation is to ensure that all integer divisions are conducted with unsigned datatypes. However, greater improvements are possible. As Chapter 10 of Warren's book explains [246], integer division can be expressed using comparatively inexpensive multiplications and bit shifts. The C/C++ libdivide library has extended these concepts to handle run-time divisors [190]. The author of libdivide reports substantial run-time improvements. To test this, experiments used libdivide's division routines for the calculations in Figure A.4. As Table A.1 demonstrates, run-time improvements were vast, resulting in roughly 4.5 times improvement over the next best implementation. With all discussed optimisations combined, the LI re-computations consume approximately 4% of the time needed for the later sorting step. We judge this an acceptable amount of time to devote to preprocessing.

The drastic improvements from the initial LI re-computation scheme to the fully optimised approach justifies the attention paid to this topic. These optimisations bolster the viability the LCO format for sparse NTs.

A.2 Sorting

Since all the sorting algorithms we tested for sparse NTs had to be adapted to an NT setting, there were many non-trivial modifications. For this reason we supply additional details on the implementations we used for testing. We also provide some details on their amenability to parallel implementations, which was not explored in this thesis but will likely be an important feature going forward. To this end, Table A.2 outlines the four algorithms that were tested, all custom-adapted to sort an LI array and its accompanying data array. It should be noted that other algorithms were tested as well, including mergesort, natural mergesort, and a recent enhancement to patience sort [247], but while competitive, they

Table A.2: Sorting algorithms and their characteristics. Performance was tested on sorting sparse NTs.

Algorithm	Type	Hybrid	Adaptive	In-Place
Introspec-tive Sort	Comparison	Yes	Partially, if using insertion sort as final step	Yes
Timsort	Comparison	Yes	Yes	No, but uses minimal buffer size
LSD Radix Sort	Integer	Yes, with modifications	Partially, if using insertion sort as final step	No
MSD Radix Sort	Integer	Yes, with modifications	Partially, if using insertion sort as final step	Yes, depending on implementation

were outperformed by the algorithms outlined above. As such, their performance is not included in this discussion.

Of the characteristics outlined in Table A.2, the most defining is whether a sort is comparison-based or integer-based. The former category refers to general-purpose algorithms able to sort a wide array of different objects, while the latter category refers to algorithms designed specifically for sorting integers, or objects that can be mapped to integers. An algorithm is designated hybrid if it is an amalgamation of two or more sorting routines. An algorithm is adaptive if it takes advantage of any pre-existing order, *e.g.*, sorted subsequences, within the values being sorted. Finally, inplace refers to whether a sorting algorithm uses additional memory or not.

A.2.1 Introspective Sort

Introspective [191] sort forms the core of the C++ Standard Library (CSL)'s `std::sort`, which is considered a gold standard of sorting algorithms [248]. As such, it is important to test. Being a hybrid algorithm, introspective sorting combines quicksort and heap sort together, with many implementations also using insertion sort as final step. In terms of parallelism, introspective sorting's initial quicksort step can be parallelised [249]; however, heapsort is less obviously parallelizable. The implementation used for these tests is based on code by Schwarz [250]. In addition, the implementation employs the insertion sort version described by Sedgewick [251], which uses less reads and writes compared to Schwarz's version.

A.2.2 Timsort

Tim Peter's timsort [192] is an adaptive algorithm that now acts as the Python language's sorting algorithm [252], making it an important comparison-based algorithm to consider. In terms of parallelism, with some modifications, it is likely that the merging process can operate in parallel, perhaps in tandem with data-parallel bitonic merge networks seen in recent works [253,254]. To test timsort's performance, a C++ version of the algorithm [255] was modified to sort LI values and their accompanying data values.

A.2.3 LSD Radix Sort

The two previous algorithms are comparison-based sorts. Yet, as LI values are integers, sorting algorithms, such as radix sort, tailored for such datatypes should also be considered. Radix sort can be implemented as either the least-significant digit (LSD) or most-significant digit (MSD) variant. The former is considered a simpler version to implement with claims of faster execution [193], and has seen recent attention by researchers in the field [254, 256]. While not immediately obvious, LSD radix sort can be task-parallelised, but has limited potential for data-parallelism [254]. The LSD implementation used for this work departs from classic implementations in several ways. For one, in the context of sparse NTs, the maximum size of the LIs is the NT's dimensionality. Thus, the number of passes to perform, *i.e.*, the dimensionality divided by the radix size, is known *a priori* and is often less than when naively using the LI's datatype size. Additionally, based on Sedgewick's suggestion [193], the most significant LSDs is skipped, allowing insertion sort, which is memory-bandwidth friendly, to perform a final clean up. Finally, the implementation used Satish *et al.*'s software buffer enhancement [254] in order to decrease cache misses. Based on performance tests, a radix size of 11 bits, corresponding to Herf's suggestion [257], and a software buffer size of 64 bits were chosen.

A.2.4 MSD Radix Sort

In contrast to LSD radix sort, the MSD variant operates by recursively grouping items together by their MSDs. This recursive nature means that MSD radix sort is more obviously task-parallelised than the LSD variant. A commonly cited reason that practitioners opt for LSD radix sorting over its MSD cousin is the large breadth of the latter's recursion tree, *cf.* Sedgewick [193] and Wassenberg and Sanders [256]. This problem can be avoided by adapting the algorithm into a hybrid approach that switches to a comparison-sorting algorithm whenever a sub-list's size no longer justifies the recursion overhead. As well, unlike the LSD variant, in-place versions of MSD exist with good algorithmic performance [194]. The implementation used for these experiments is based off of Duvanenko's code [194]. However, the implementation used here switches to the introspective sorting algorithm described above, minus the insertion sort step, for sub-lists smaller than 3000. This significantly reduced overhead and recursion depth, resulting in notable performance gains. An insertion sort execution is then used to clean up the entire LCO data structure as a final step. A radix size of 11 bits is used.

A.3 Abelian Operations

It is possible to just use one routine to execute sparse abelian operations, *i.e.*, addition and subtraction, for sparse NTs. However, efficiencies can be gained by considering the index matching, lexicographical orders, and sort status of the two sparse NT operands. We details these considerations in this section.

Like all NT arithmetic, abelian operations, such as addition and subtraction, are affected by the matching of operand indices. For instance, in the following two expressions,

$$\texttt{a(i,j,k)+b(i,j,k)}, \tag{A.1}$$

$$\texttt{a(i,j,k)+b(j,k,i)}, \tag{A.2}$$

the elements of the two operands in (A.1) match up using the same index sequence. However, in (A.2) elements are matched-up differently. For example, if `a` is indexed by `{0,1,2}`, then the corresponding element in `b` would be located in `{1,2,0}`.

In the dense case, apart from memory contiguity issues, the different access patterns pose little problem. However, in the sparse case non-zero entries must actually be rearranged using a sort or permute to reflect different access patterns. Since LibNT allows any valid lexicographical order for its internal data representation, whether a rearrangement is required is based solely on whether non-zeros of the operands are arranged properly relative to each other. This depends on both the expressed index matching and the internal lexicographical orders.

For instance, take the expression in (A.1), and assume the non-zeros of both operands are already sorted. Since the operands use the same index sequence, only their internal lexicographical order affects whether one needs to be permuted prior to addition. Should their lexicographical orders match they are synchronised and no rearrangement is required. It is immaterial what actual internal lexicographical order is used, as long as they match.

On the other hand, the situation in (A.2) is more complicated, as both how indices match-up and the internal lexicographical order determines whether a rearrangment is needed or not. Thus, in (A.2) because each operand expresses a different index sequence, matching internal lexicographical orders are actually not synchronised. A rearrangement can be avoided only if each operand's internal lexicographical order synchronises based on the index matching.

Both of the cases considered so far assumed that the operands were already sorted. If one or both of the operands are not sorted, then a sort will definitely be needed prior to performing any abelian operations regardless of the operands' choice of internal lexicographical order. Consequently, whether a sort is required is based on the following three factors:

1. the sort status of operands,
2. the index matching, and

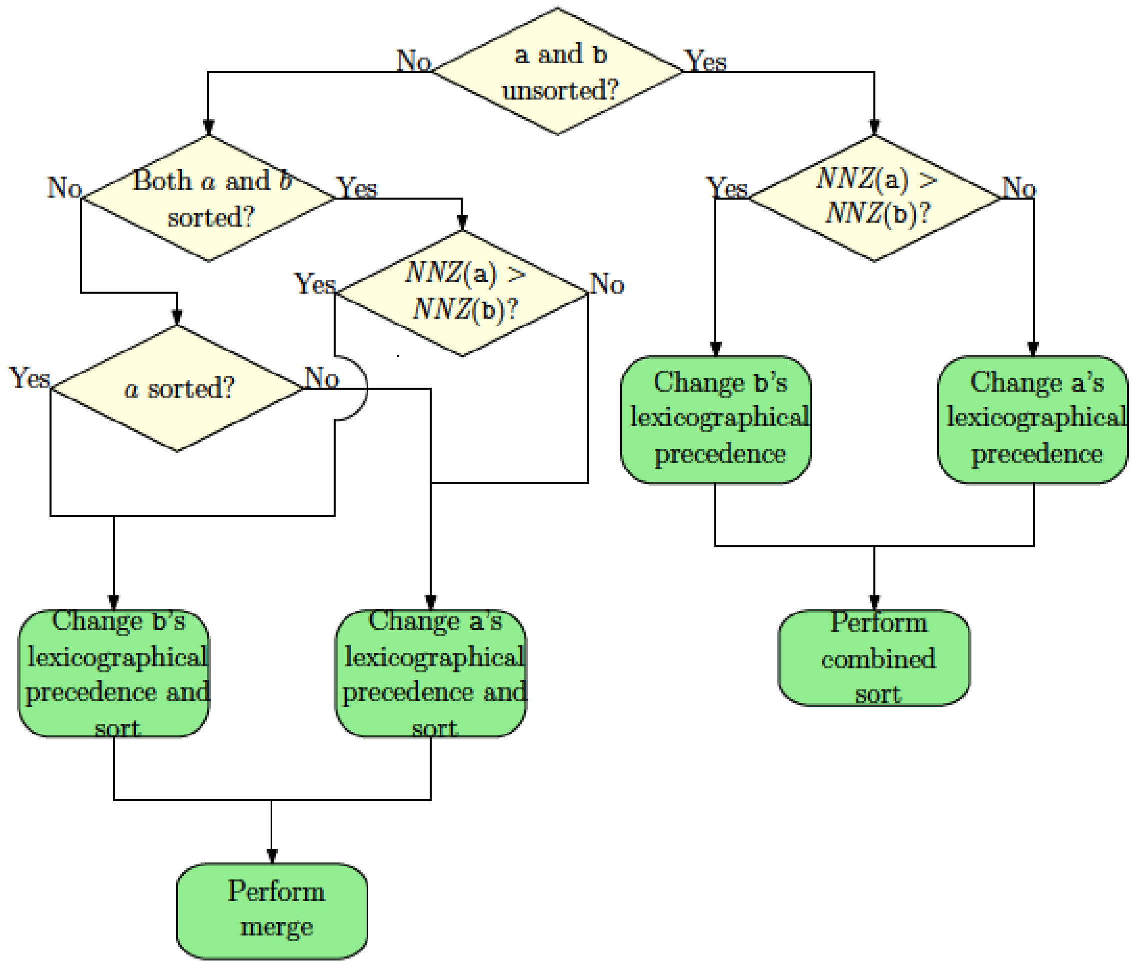

a and b unsorted?
No
Yes
Both a and b sorted?
No
Yes
NNZ(a) > NNZ(b)?
Yes
No
a sorted?
Yes
No
NNZ(a) > NNZ(b)?
Yes
No
Change b's lexicographical precedence
Change a's lexicographical precedence
Change b's lexicographical precedence and sort
Change a's lexicographical precedence and sort
Perform combined sort
Perform merge

Figure A.5: Flow chart of execution of sparse NT abelian operations.

3. the internal lexicographical order of operands.

The need or lack thereof of a rearrangement in turn affects how a merge is best executed. Ideally, rearrangements should be avoided or minimised. The flowchart in Fig A.5 illustrates the decision path taken by LibNT in performing abelian operations.

The sort status is the first factor considered by LibNT. If *both* operands are unsorted, then rather than sort each operand individually followed by a merger of LCO arrays, it is more efficient to simply concatenate both sets of LCO arrays together, sort them, and then collect any duplicates using the abelian operation in question. If the internal lexicographical order of one of the operands needs to changed, the operand with the least number of non-zeros (NNZ) is altered. Figure A.6 illustrates the combined sort process corresponding to addition with two simple sparse NTs. The figure illustrates non-destructive addition. For the destructive case, *i.e.*, +=, b's LCO arrays are concatenated to a's instead of to a new set of LCO arrays.

As Figure A.5 illustrates, when one or both operands are already sorted, the LCO arrays of the operands are merged together. If one of the operands is unsorted, it is permuted

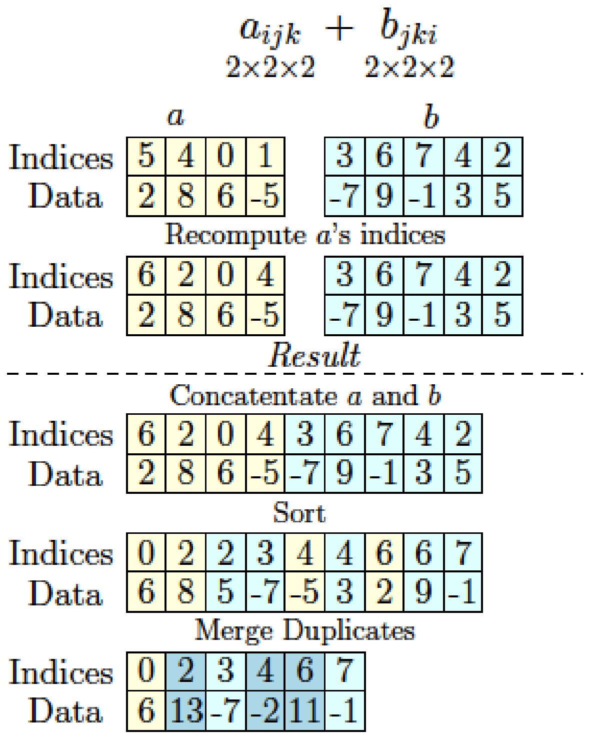

$\underset{2\times2\times2}{a_{ijk}} + \underset{2\times2\times2}{b_{jki}}$
a
b
Indices 5 4 0 1 | 3 6 7 4 2
Data 2 8 6 -5 | -7 9 -1 3 5
Recompute a's indices
Indices 6 2 0 4 | 3 6 7 4 2
Data 2 8 6 -5 | -7 9 -1 3 5
Result
Concatentate a and b
Indices 6 2 0 4 3 6 7 4 2
Data 2 8 6 -5 -7 9 -1 3 5
Sort
Indices 0 2 2 3 4 4 6 6 7
Data 6 8 5 -7 -5 3 2 9 -1
Merge Duplicates
Indices 0 2 3 4 6 7
Data 6 13 -7 -2 11 -1

Figure A.6: Performing sparse non-destructive NT addition using a combined sort. Yellow and cyan designate entries coming from a and b, respectively, while blue indicates entries that have been added together. Internal lexicographical order of both operands is $\{0, 1, 2\}$ at the start. If neither operand is sorted, then the operand with the least NNZ has its internal lexicographical order changed, if necessary. Here this corresponds to a. The LCO arrays are then concatenated together and sorted. Duplicates are merged as a final cleanup.

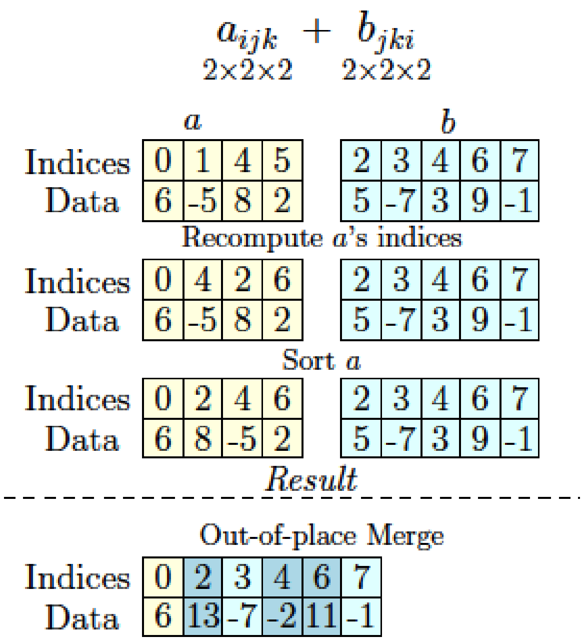

$a_{ijk} + b_{jki}$
2×2×2 2×2×2
a
b
Indices 0 1 4 5 2 3 4 6 7
Data 6 -5 8 2 5 -7 3 9 -1
Recompute a's indices
Indices 0 4 2 6 2 3 4 6 7
Data 6 -5 8 2 5 -7 3 9 -1
Sort a
Indices 0 2 4 6 2 3 4 6 7
Data 6 8 -5 2 5 -7 3 9 -1
Result
Out-of-place Merge
Indices 0 2 3 4 6 7
Data 6 13 -7 -2 11 -1

Figure A.7: Performing sparse non-destructive NT addition using a merger. Yellow and cyan designate entries coming from a and b, respectively, while blue indicates entries that have been added together. Internal lexicographical order of both operands is $\{0, 1, 2\}$ at the start. Despite being sorted already, the different index sequences between the two operands occasions a rearrangement of non-zeros. Since both operands are sorted, then the operand with the least NNZ is permuted to synchronise lexicographical orders. Here this corresponds to a. The two LCO arrays are then merged out-of-place.

to a synchronised lexicographical order. If both operands are sorted, but their internal lexicographical orders do not synchronise, then the operand with the least NNZ is permuted. After permutation, the LCO arrays are merged out-of-place and in-place for non-destructive operations and destructive abelian operations, respectively. Figure A.7 depicts the merging process for two simple sparse NTs.

A.4 Mixed Dense/Sparse Operations

Since dense and sparse data representations are so distinct, LibNT and NTToolbox must must also decide on the data representation of the result of any arithmetic operation between two NTs. For NTToolbox, which uses a runtime resolution of data representation, this process can be simply done by computing the fill factor and altering data representation based on the result. However, data representation rules for LibNT must be decided statically, as whether NTs are dense or sparse is determined at compile-time.

Some choices are trivial. For instance, the product, sum, or difference between any two dense tensors is also dense. If both operands are sparse, then the result will also be sparse for these operations. On the other hand, the solution of a series of linear NT equations is dense regardless of the makeup of the operands.

Table A.3: LibNT's data representation rules for mixed dense/sparse NT products.

Multiplication Type	Example	Result Data Representation
Lattice	$c_{ij} = a_{ik}b_{kj}$ $c_{jki} = a_{i\underline{k}\ell}b_{\underline{k}\ell j}$	**Dense**
No Lattice	$c_{ij} = a_{i}b_{j}$ $c_{jki} = a_{i\underline{k\ell}}b_{\underline{k\ell} j}$	**Sparse**

```
1  DenseNT<double,3> a(4,6,5);
2  SparseNT<double,3> b(6,5,7);
3  DenseNT<double,3> dense_c;
4  SparseNT<double,3> sparse_c;
5  NTINDEX i; NTINDEX j;
6  NTINDEX k; NTINDEX l;
7
8  //valid assignment of dense product to dense NT
9  dense_c(j,k,i)=a(i,!k,l)*b(!k,l,j);
10 //also valid, but will convert dense product to sparse NT
11 sparse_c(j,k,i)=a(i,!k,l)*b(!k,l,j);
```

Figure A.8: Code example demonstrating how to override LibNT's data-representation choices during an NT product.

When one operand is dense and the other sparse, the situation can become more complex. For abelian operations, *i.e.*, addition or subtraction, the result between a dense and sparse NT is always dense. However, for NT multiplication, the choice depends on whether or not inner products come into play. As Section 4.2.4 explained, an NT product is only mapped to a lattice product when an inner product needs to be executed. This also affects the resulting data representation. For instance, a pure outer product between a dense and sparse NT will in general posses a sparse amount of non-zeros. The same holds if entrywise products are also executed. However, an inner product between a dense and sparse NT will in general have a dense amount of non-zeros. As Table A.3 demonstrates, these rules of thumb are encapsulated by LibNT.

Since datatypes must be specified statically in C++, LibNT follows the rules of Table A.3 by examining how indices between operands match up during compilation. If an inner product is detected, a dense datatype is chosen for the product. Otherwise, a sparse datatype is specified. These choices are made during compile time. Regardless of the choice LibNT makes, the user is still able to override the choice. For example, both expressions on Line 9 and 11 of Figure A.8 are legal. However, on Line 11 LibNT must convert the dense product to a sparse NT, meaning there are additional hidden computations that can only be justified by examining runtime sparsity patterns. For this reason, if efficiency is paramount it is on the onus of the user to decide whether or not to override the compile-time choices of LibNT. The merits of such a choice can only be assessed based on the user's knowledge of the problem context.

Appendix B

Software and Testing

This appendix provide details on the compiler and test environments used for this thesis.

B.1 C++11 Features

LibNT relies heavily on the C++11 standard. For this reason, only a compiler with near complete C++11 support is able to build programs using LibNT. The following C++11 features are used within the library:

- `constexpr`
- variadic templates
- move semantics
- rvalue references
- `auto`
- `decltype`
- `std::array`
- `std::function`
- lambda functions
- smart pointers

B.2 Benchmark Details

All benchmarks were performed on a Windows workstation using an Intel dual-core E8400 CPU with 4Gb of memory. Benchmarks are compiled using Intel's C++ compiler with full optimisation turned on. OpenMP flags were also turned on. The list of flags are provided below:

- `-Qrestrict`
- `-Quse-intel-optimised-headers`
- `-O3`
- `-Qipo`
- `-Qopt-matmul`
- `-QxHost`
- `-Ot`
- `-Oi`
- `-Qansi_alias`
- `-Qstd=c++11`
- `-Qopenmp`

As well, any benchmarks testing dense multiplication used Eigen's optional support of the Intel MKL. This was done to put the numeric tensor (NT) software on an even footing with the MATLAB Tensor Toolbox (MTT), which relies on MATLAB's optimised LAPACK routines.

It should also be noted that many of the benchmarks were tested with all three of MinGW, Microsoft, and Intel's C++ compilers. The choice of compiler did not affect any of the conclusions drawn from performance tests.