Università degli Studi di Padova

Sede amministrativa: Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi Civita"

CORSO DI DOTTORATO DI RICERCA IN SCIENZE MATEMATICHE

CURRICULUM MATEMATICA COMPUTAZIONALE

CICLO XXXIV

Topics in Numerical Linear Algebra for High-Performance Computing

Tesi redatta con il contributo finanziario di beanTech Srl

Coordinatore
Ch.mo Prof. Martino Bardi

Supervisore
Ch.mo Prof. Fabio Marcuzzi

Dottoranda
Monica Dessole

Sommario

Il problema di risolvere in maniera veloce e robusta sistemi lineari di grandi dimensioni, nelle sue varie declinazioni, è il cuore di svariate applicazioni numeriche in molti campi, dalla stastistica all'ingegneria, dalle geoscienze alla diagnostica medica per immagini e molti altri. Questo è un problema classico in algebra lineare numerica, che gode di rinnovato interesse se, come nel lavoro qui presente, ci interessiamo a problemi mal posti, dove la matrice non ha rango pieno e la soluzione può non essere unica, o alle emergenti architetture many-cores come le schede grafiche o GPU. La prima parte di questa tesi è dedicata alla risoluzione efficiente di problemi ai minimi quadrati dove la matrice della funzione obiettivo può non avere rango pieno, e di problemi ai minimi quadrati con vincoli di nonnegatività. Queste classi di problemi sono particolarmente impegnative poiché la loro risoluzione esatta richiede tecniche di ricerca enumerativa, data la loro natura combitatorica, con conseguenti difficoltà nel derivare algoritmi ad alte prestazioni. A tal fine deriviamo un algoritmo a blocchi per la selezione di colonne, investighiamo le proprietà dei metodi ottenuti da un punto di vista teorico in termini di accuratezza della soluzione e analizziamo le performance ottenute rispetto agli algoritmi stato dell'arte, mostrando i risultati di un'esaustiva campagna di esperimenti. In particolare, il solutore ai minimi quadrati nonnegativi qui proposto favorisce naturalmente la sparsità della soluzione e perciò può essere adoperato in applicazioni di compressed sensing. Come applicazione rilavante, presentiamo un pacchetto numerico per la compressione quasi G-ottimale di misure di probabilità finite in più dimensioni per la regressione polinomiale di grado elevato basato sulla risoluzioni di problemi ai minimi quadrati nonnegativi. La seconda parte della tesi affronta l'impegnativa questione su come adattare i solutori per sistemi lineari con matrici sparse, intese qui come quelle matrici nelle quali la maggior parte degli elementi sono nulli, alle architetture moderne e future proponendo metodi specificatamente pensati per le piattaforme hardware ibride e massivamente parallele oggigiorno disponibili. Ci dedichiamo al design e all'implementazione efficiente di solutori lineari di tipo diretto, cioè basati su fattorizzazioni matriciali, per la risoluzione di una classe particolare di sistemi lineari con matrici a blocchi derivanti, ad esempio, dalla risoluzione numerica di problemi di controllo ottimo. Nello specifico, discutiamo l'utilizzo di paradigmi di programmazione parallela innovativi applicati ai solutori esistenti per migliorarne la scalabilità, cioè la capacità di gestire una mole di lavoro crescente. Per la risoluzione di sistemi lineari sparsi, i metodi iterativi sono generalmente ritenuti più adatti per l'implementazione su architetture parallele in quanto basati su operazioni semplici, quali il prodotto matrice-vettore. Questi metodi vengono spesso accoppiati con una tecnica di precondizionamento, per migliorarne la velocità di convergenza. Uno dei precondizionatori generici più diffusi ed efficaci è il precondizionatore ILU, basato sulla risoluzione di sistemi triagolari sparsi, un calcolo intrinsecamente sequentaziale, e la sua applicazione risulta essere l'operazione più dispendiosa sulle architetture parallele. In questo lavoro affrontiamo la risoluzione dei problemi algebrici derivanti dalla simulazione di flussi incomprimibili con densità variabile e mostriamo che un adeguato approccio iterativo per la risoluzione dei sistemi triangolari risultanti dall'applicazione del precondizionatore ILU si rivela robusto ed efficiente sulle architetture parallele.

Abstract

The heart of numerical applications from a very broad range of domains, from statistics to engineering, geophysics, medical imaging and many others, consists in the fast and accurate solution of large size linear systems in many flavors. This is a classic topic in numerical linear algebra of renewed interest if, like in the present work, we address to large ill-posed problems, in which the matrix is possibly rank-deficient and the solution may not be unique, or to emerging hardware technologies like many-cores architectures such as Graphic Processing Units (GPUs). The first part of this thesis is devoted to the efficient solution of numerically rank-deficient least squares and nonnegative least squares problems. Such problems are particularly challenging and their exact solution would require exhaustive-search algorithms and thus their complexity is combinatorial, with a consequent difficulty to achieve high performances in computations. To this aim, we devise a new block column selection strategy, we investigate the derived methods with respect to accuracy of the solution found from the theoretical point of view and analyze their performance against the state-of-the-art algorithms, showing results with an extensive campaign of experiments. In particular, the nonnegative least squares solver presented naturally produces sparse solutions and it can be employed in many applications in the field of compressed sensing. As a relevant application, we present a numerical package for the computation of compressed multivariate near G-optimal finite probability measures for polynomial regression design of general degree by means of nonnegative least squares problems. In the second part of this thesis, we target the challenging question of how linear solvers for sparse matrices, intended here as matrices mostly filled up with zeros, can be adapted to modern and future computing facilities by proposing numerical methods specifically designed for the highly parallel and hybrid hardware platforms available nowadays. We first address the efficient design and implementation of parallel direct solvers, that is based upon matrix factorization, for a particular class of linear systems with block structured matrices, arising e.g. in optimal control problems. In particular, we discuss the use of novel parallel programming paradigms on existing direct solvers in order to improve their scalability, intended as the ability to solve problems of larger and larger size. When dealing with sparse linear systems, iterative methods are generally appreciated to be more suited for parallel architectures, as they are based upon simple computations such as matrix-vector multiplication. These methods are often coupled with a preconditioner in order to improve the rate of convergence. One of the most popular and powerful general purpose is the ILU preconditioner, that relies on the direct solution of sparse triangular systems, an inherently sequential task, and its application turns out to be the most time consuming operation on parallel architectures. In this work we deal with the solution of the algebraic problems arising from the simulation of incompressible flows with variable density and we show how an adequate iterative approach to the parallel solution of the triangular systems that result from the ILU preconditioner, turns out to be robust and efficient on massively parallel architectures.

Acknowledgments

My deepest gratitude goes to Prof. Fabio Marcuzzi, who guided me through my undergraduate and graduate studies. This work would not have been possible if he hadn't supported me from the very beginning, with his positive and constant encouragement and the helpful discussions. I am also very grateful to Prof. Marco Vianello for his pleasant and fruitful collaboration and for providing me with useful ideas and hints. I would like to thank the referees of the present thesis and of the papers here contained, who contributed with their helpful comments to substantially improve the quality of this work.
I am grateful to all the people belonging to the research community of Padova, especially to the colleagues who shared this venture with me. In this context, I also would like to thank the company beanTech Srl for their financial support enabling this research.
Last but not least, I would like to thank my family, my father Ottavio and my brother Mattia, and Giorgio, who always believe in me, and all my friends around the world for their less scientific, but nevertheless equally important support.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation and goals

We are interested in the efficient solution of the many variations of the linear system

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

where A is a matrix of size $m \times n$ and $\mathbf{x}, \mathbf{b}$ are the unknown and the right-hand side vectors. In the case in which the right-hand side does not belong to the space spanned by the columns of A, namely $\mathbf{b} \notin \mathcal{R}(A)$, then equation (1.1) does not admit a solution and we rather look for a solution that minimizes the 2-norm of the residual, namely we seek the solution of the *least squares problem*

$$\min_{\mathbf{x}\in\Omega} \|A\mathbf{x} - \mathbf{b}\|, \tag{1.2}$$

where the solution vector is possibly constrained to range in the subset $\Omega \subset \mathbb{R}^n$. On the other hand, if the right-hand side does belong to the space spanned by the columns of A, namely $\mathbf{b} \in \mathcal{R}(A)$ but the matrix A has not full column rank, then (1.1) has infinitely many solutions and we want to choose the best solution $\mathbf{x}$ for our purpose. These are classic topics in numerical linear algebra of renewed interest if, like in the present work, we address to large ill-posed problems, in which the matrix is possibly rank-deficient and the solution may not be unique, or to emerging hardware technologies like many-cores architectures such as Graphic Processing Units (GPUs).

The goal of this work is to cope with these challenging problems devising robust dense and sparse linear algebra solvers that can achieve better efficiency on modern processing architectures by means of lower communications and memory consumption, that show a good scalability, namely the capacity of handle a growing amount of work, and that can guarantee a high achieved occupancy on parallel architectures, that is the fraction of active computing units (cores) over the upper limit. Implementation of efficient and scalable numerical methods suitable for high performance computers is an interdisciplinary task which requires an extended knowledge of applied mathematics and computer science. Mathematical models, numerical methods and software implementations need new conceptual and programming paradigms to make effective use of unprecedented levels of concurrency provided by modern many-core processors and the ever increasing complexity of hardware architectures. Moreover, it is important to stress that the effort for high quality product code is way larger than the effort for prototype code.

1.2 Thesis outline and contributions

Methods for solving equations (1.1) and (1.2) can be roughly grouped into two families: direct and iterative methods. Direct methods proceed by computing a factorization of the system matrix A such as the LU, SVD or QR factorizations; the solution of the original problem is then obtained as the solution of one or more linear systems involving the resulting factor matrices, which are easy and cheap to solve, e.g. triangular linear systems. Instead, iterative methods start from an initial guess solution and improve it step by step until a desired solution accuracy is achieved or when a maximum number of iterations is reached without converging to a satisfactory solution. The choice of the solution method and the consequent algorithm to employ must take into account the structural and numerical properties of the matrix A. For what concerns structural properties, the first distinction can be established between *dense* and *sparse* matrices. The terms dense and sparse refer to the data structure used to store a matrix. The matrix $A \in \mathbb{R}^{m \times n}$ is dense if it is stored as a full array of m rows and n columns with mn entries. A popular definition of a sparse matrix is attributed to James Wilkinson: *"A sparse matrix is any matrix with enough zeros that it pays to take advantage of them"*. Nowadays, a matrix is considered to be sparse if it contains a number $\mathcal{O}(\max\{m, n\})$ of nonzero entries.

The first part of this thesis addresses high-performance algorithms for dense linear algebra. Efficient numerical linear algebra code is build on top of Basic Linear Algebra Subprograms (BLAS), which are grouped in three levels:

- BLAS-1: vector-vector operations;
- BLAS-2: matrix-vector operations;
- BLAS-3: matrix-matrix operations.

In the context of high-performance coding, the classical operation count alone may not be a good indicator of the efficiency of an algorithm. In order to close this gap, it is fundamental to design algorithms according to modern computer architectures while ensuring robustness. Block algorithms have become increasingly popular in matrix computations, see e.g. [118]. Since their basic unit of data is a submatrix rather than a scalar they have a higher level of granularity than point algorithms, which allows to write algorithms in terms of BLAS-3 operations instead of BLAS-1 or BLAS-2, and this makes them well-suited to high-performance computers by increasing data locality (i.e. cache reuse).

When the matrix A has not full column rank, a solution of (1.1) or (1.2) can be obtained by selecting a minimal subset of columns which spans the same subspace spanned by the matrix A. From the machine learning and statistics viewpoint, this problem is known as *subset selection* or *feature selection* and is the process of selecting a subset of relevant features avoiding redundancy in the model. From the linear algebra viewpoint, this problem amounts to find a suitable column pivoting which separates the minimal column subset from the linearly dependent columns. Column pivoting makes it more difficult to achieve high performances in matrix factorization, see [17, 18, 19, 119, 20], mainly because it involves memory communications and most implementations are based on greedy approaches that select a column at a time relying on BLAS-2 operations. To overcome this issue, in Chapter 2 we propose a column selection technique we call *deviation maximization*. Our method relies on a correlation analysis, here intended as cosine of the angle between two vectors to stress the linear algebra viewpoint, in order to select a subset of sufficiently linearly independent vectors. Despite this strategy is not sufficient by itself to identify a minimal subset of linearly independent columns for a given numerically rank deficient matrix, it can be adopted as a block pivoting strategy in more complex applications.
In Chapter 3, we deal with rank-deficient least squares problems (1.2), where $\Omega = \mathbb{R}^n$. Here, the SVD decomposition is the safest and most expensive solution method, while approaches based on a modified QR factorization, such as the so-called rank-revealing QR, can be seen as cheaper alternatives. Since the QR factorization is essentially unique once the column ordering is fixed, these techniques all consist in finding an appropriate column permutation. In this chapter, based on [60], we propose a new block algorithm based on deviation maximization pivoting for computing a rank-revealing QR decomposition. We test the proposed strategy against the state-of-the-art algorithm in terms of execution times and quality of the decomposition found on a large set of instances of medium size, with up to thousands of rows and columns.
In Chapter 4, we cope with least squares problems (1.2) with nonnegativity contraints, where $\Omega = \{\mathbf{x} \in \mathbb{R}^n : \ \mathbf{x} \geq 0\}$. Nonnegative least squares problems arise in many applications where data points can be represented as nonnegative linear combinations of some meaningful components. Such problems are frequently encountered in signal and image processing and they are core problems in more complex computations, such as nonnegative matrix and tensor decompositions. This problem can also be seen as a column selection problem: in fact, once the set $P = \{i : x_i^\star > 0\}$ is known, where $\mathbf{x}^\star = (x_i^\star)$ is a solution vector of (1.2), then $\mathbf{x}^\star$ can be obtained by solving an unconstrained least squares problem where the objective matrix is the submatrix of A obtained by selecting the columns indexed in P. Therefore, we propose a new block algorithm based on deviation maximization pivoting for solving nonnegative least squares problems based on [64]. In particular, we address the case in which the matrix A is underdetermined, i.e. it has more columns than rows ($n > m$). In this case the nullspace of A is nontrivial, thus we have infinitely many solutions. The nonnegativity contraint is known to naturally enhance sparsity of the solution, that is the

solution attained has few nonzeros, see e.g. [31, 76, 143, 144]. An important outcome of this body of work is that nonnegativity alone may attain a satisfactory sparse solution. This problem is known as *sparse recovery* and it has drawn much attention within the research community in the last two decades, leading to the new field of *compressed sensing*, with application through engineering and medicine. As a relevant application, in Appendix A we present *dCATCH* [62], a numerical package for the computation of compressed multivariate near G-optimal finite probability measures for polynomial regression design of general degree based on the celebrated Tchakaloff theorem, a cornerstone of quadrature theory. This problem can be formulated as a nonnegative least squares problem where the matrix underdetermined and large, here up to thousands of rows and millions of columns, which can be efficiently solved, as it was already pointed out in [61].

The second part of this thesis is devoted to massively-parallel algorithms for sparse linear algebra. There are three main reasons to explicitly take advantage of the fact that a matrix is mostly filled up with zeros. First of all, the memory actually needed to store a sparse matrix is less than the memory needed for the same matrix as a dense matrix of the same size because the zeros need not be stored explicitly. Second, the complexity of most operations on a sparse matrix can be greatly reduced with respect to the same operation on a dense matrix of the same size because algorithms can be rewritten in order to avoid computations involving zero coefficients of the original matrix. Finally, parallelism can be much higher than in the dense case because some operations may involve distinct subsets of the matrix nonzero entries and can thus be applied concurrently. In this work we mainly address General Purpose computing on GPUs (GPGPU), which is nowadays a cost effective solution for computational intensive simulations. The inherent massively parallel architecture of GPUs demands for completely different algorithms from that used in a mainly sequential, CPU based, computing architecture. Direct methods for sparse linear systems are widely appreciated for their numerical robustness and reliability, as they are capable of computing accurate solutions for a very wide range of problems without the need for the user to have any knowledge of linear systems solvers. Another case where direct methods are often employed is where the same matrix has to be solved with multiple right-hand sides because the matrix factorization, which is the most expensive operation, only has to be computed once and its result reused for multiple, cheap, solve operation. However, they are also characterized by their high computational complexity: in fact, even if the matrix A is sparse, its factors in classic matrix decompositions are usually dense matrices. Ideally, we would like the factors to have the same sparsity pattern as A, namely the set $\mathcal{S}(A) = \{(i, j) : \ a_{ij} \neq 0\}$, or at least close enough. However, in the course of calculating the factorization, other nonzero entries may be introduced in the factors and those are referred to as the *fill-in*. In order to limit computational cost and memory usage, direct solvers need to be carefully designed. Moreover, the solution of sparse linear systems often decomposes in subproblems of different size, leading to irregular and unbalanced computations for which efficient solvers are complex to design. This is the reason why direct methods are generally considered to be more difficult to efficiently implement on massively parallel architectures. Efficient implementations require the exploitation of all types of parallelism in the programs and the reduction of communication to almost nothing. The real challenge is left to the programmers, as the application has to be broken down into smaller parallel tasks, and also each task has to be assigned to the suitable hardware device. To this aim, parallel programming models based on Directed Acyclic Graph (DAG) parallelism recently regained popularity in the high-performance, scientific computing community. This technique consists in writing the numerical algorithm at a high level independently of the hardware architecture as a DAG of tasks where a vertex represents a task and an edge represents a dependency between tasks. From the parallel programming viewpoint, such a DAG highlights data dependencies in computations and it is therefore fundamental for performance analysis before investing in a real implementation. In the sparse linear algebra community, only few research efforts have been conducted in this sense. Iterative methods for sparse linear systems are generally considered well suited to be implemented on parallel architectures, as they are based upon easy operations such as matrix-vector product. The convergence of an iterative method essentially depends on the numerical properties of the matrix A. More precisely, the eigenvalues of a normal matrix provide all of the essential information about that matrix, as far as iterative linear system solvers are concerned. Under suitable hypotheses, other tools can be used such as field of values or pseudospectra when dealing with nonnormal matrices. Indeed, in some cases the solution of a linear system may require a very high number of iterations or convergence may not be reached at all. For this reason iterative solvers are often used in combination with preconditioning techniques. A preconditioner can be seen as a matrix M such that $M^{-1} \approx A^{-1}$, i.e. its inverse approximates that of A. This is used to transform the linear system (1.1) into the modified linear system $M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}$. At each step of the preconditioned algorithm, it is necessary to compute the product of M^{-1} with a vector or to solve a linear system with matrix M. Notice that these operations are equivalent in exact arithmetic, but the computational effort required depends on the structure of M or on how M^{-1} is computed. A good preconditioner M should be chosen so that such linear systems are much easier to solve than the original problem and so that the modified problem shows a faster convergence. Many research efforts have been dedicated to the subject of preconditioning in last decades, most of them have dealt with the design of preconditioners for specific classes of problems.

Sparse systems do not only come from numerical methods for PDEs: Chapter 5, which is based on [59], focuses on a family of sparse matrices with block structure, called Bordered Almost Block Diagonal (BABD) matrices, arising e.g. in optimal control problems. General purpose direct solvers are undesirable for many reason, not last the introduction of fill-in in the solution procedure, leading to significant inefficiencies. We propose a new direct solver tailored for parallel many-cores architectures, proving its performance analysis in terms of data dependency DAG, which ensures a better work per step balance and remarkable memory savings. Here we address large matrices with up to millions of rows and columns. Last, Chapter 6 is based on [58] and it deals with the parallel iterative solution of the Navier-Stokes equations, where often $\mathcal{O}(10^5)$ coupled nonlinear equations must be solved at each discretization time instant, to capture the relevant underlying physics of the fluid motion. Here we focus on Incomplete LU (ILU) preconditioner, one of the most popular general purpose preconditioners. The ILU precondidioner relies on the solution of two triangular systems at each application. Solving a triangular system is an inherently sequential computation activity: in fact, these systems are typically solved by forward and backward substitution. Here, we focus on obtaining from an ILU preconditioner the best performance on a GPU architecture. For this reason, we mainly investigate the parallelization of the triangular systems solver needed at each iteration within the preconditioned GMRES method. This is the computationally intensive operation that requires the most of the time spent by the simulation.

This research work has been accomplished within the PhD fellowship *"GPU computing for modelling, nonlinear optimization and machine learning"* funded by beanTech Srl. Given the extent and the complexity of the themes here discussed, this thesis is far from being self-contained. For a general background on the topic see e.g. [81]. The linear algebra algorithms presented within this thesis have been implemented in C or C/CUDA compiled languages, while high level layers of the software have been written in Python or Matlab interpreted languages. The codes here used are freely available online [57].

1.3 Notation

In what follows we denote matrices by capital letters, $A \in \mathbb{R}^{m\times n}$, where $\mathbb{R}^{m\times n}$ denotes the space of $m \times n$ real valued matrices. Vectors are indicated in bold, $\mathbf{x} \in \mathbb{R}^n$, while scalars are denoted by lower case letters, $a \in \mathbb{R}$. We access the elements of a matrix A in many ways, therefore we introduce different matrix partitions. Let us write the matrix A as a stack of n column vectors of length m, namely

$$A = (\mathbf{a}_1 \dots \mathbf{a}_n).$$

This is called a column partition of A. Similarly, the matrix A can be written as a stack of m row vectors of length n. This is called a row partition, that is

$$A = \begin{pmatrix} \mathbf{r}_1^T \\ \vdots \\ \mathbf{r}_m^T \end{pmatrix}.$$

For any matrix A of size $m\times n$, we denote by $A(I, J)$ the submatrix of A obtained considering the entries with row indices ranging in the set I and column indices ranging in the set J. We make use of the so called "colon notation", that is we denote by $A(k : l, p : q)$ the submatrix of A obtained considering the entries with row indices $k \le i \le l$ and column indices $p \le j \le q$. We denote by $A(:, p : q)$ $(A(k : l, :))$ the submatrix of A obtained considering the entries with column (row) indices $p \le j \le q$ ($k \le i \le l$) and row (column) indices $1 \le i \le m$ ($1 \le j \le n$). We use the shorthand A_J to indicate the submatrix $A(1:m, J)$, with column indices ranging in $J \subseteq \{1, \dots, n\}$ and row indices $1 \le i \le m$. We also denote the (i,j)-th entry as a_{ij} or A_{ij} ($A(i,j)$). Last, we denote I_n the identity matrix of order n, omitting the subscript when the size is clear from the context, and by 0 the null matrix, which may even be rectangular.

For any matrix $A \in \mathbb{R}^{m\times n}$, we write A^T to indicate the transpose matrix of size $n \times m$, namely if $A = (a_{ij})$ then $A^T = (a_{ji})$, for $1 \le i \le m$ and $1 \le j \le n$. If A is square and nonsingular, we write A^{-1} to indicate the inverse of A, otherwise we use the symbol $A^\dagger$ to denote the Moore–Penrose inverse of A. We denote by $\mathcal{R}(A) \subseteq \mathbb{R}^m$ the range of A, i.e. the subspace spanned by the columns of A, and by $\mathcal{N}(A) \subseteq \mathbb{R}^n$ the nullspace of A, namely the subspace of those vectors $\mathbf{x}$ for which we have $A\mathbf{x} = 0$. For any subspace $\mathcal{S}$ of $\mathbb{R}^n$, we denote by $\mathcal{P}_{\mathcal{S}}$ the orthogonal projection on $\mathcal{S}$.

The singular values of a matrix A are denoted as

$$\sigma_{\max}(A) = \sigma_1(A) \ge \sigma_2(A) \ge \cdots \ge \sigma_{\min}(A) = \sigma_{\min(m,n)}(A) \ge 0.$$

Given the vector norm $\|\mathbf{x}\|_p = (|x_1|^p + \dots |x_n|^p)^{1/p}$, $p \ge 1$, we denote the family of p-norms as

$$\|A\|_p = \sup_{\|\mathbf{x}\|_p = 1} \|A\mathbf{x}\|_p.$$

We denote the operator norm by $\|A\|_2 = \sigma_{\max}(A)$. When the context allows it, we drop the subscript on the 2-norm. With a little abuse of notation, we define the max-norm of A as $\|A\|_{\max} = \max_{i,j} |a_{ij}|$. Recall that the max-norm is not a matrix norm (it is not submultiplicative), and it should not be confused with the ∞-norm $\|A\|_\infty = \max_i \sum_j |a_{ij}|$.

Equalities and inequalities between vectors and scalars are meant elementwise, that is $A\mathbf{x} = b$ stands for $\sum_j a_{ij}x_j = b$, for every i.

Part I

High-performance algorithms for dense numerical linear algebra

Chapter 2

Deviation maximization for column selection

2.1 Preliminary results

For sake of completeness, let us list in what follows some easy but useful facts we often use in this work. In order to help the reader, some results are stated together with their proof, others are simply reported and referenced.

2.1.1 About Singular Values

Let A be an $m \times n$ matrix, and recall that the singular values of A are the roots of the largest $\min(m, n)$ eigenvalues of A^TA or AA^T. This is quite evident using the SVD decomposition $A = U\Sigma V^T$, where U and V are unitary matrices of order m and n respectively, and Σ is an $m \times n$ pseudo-diagonal matrix (its extra-diagonal elements are null). Since $A^TA = V\Sigma^T\Sigma V^T$ and $AA^T = U\Sigma\Sigma^TU^T$, where $\Sigma^T\Sigma$ and $\Sigma\Sigma^T$ are diagonal matrices of order n and m respectively, but they clearly share the same diagonal elements up to index $\min(m, n)$. For any orthogonal matrix Q of order m, we have

$$A^TA = A^TQ^TQA = (QA)^TQA, \tag{2.1}$$

therefore A and QA have the same singular values. On the other hand, if Q is an orthogonal matrix of order n, we have

$$AA^T = AQQ^TA = AQ(AQ)^T. \tag{2.2}$$

This holds in particular for any permutation matrix Π, hence column or row permutations do not change the singular values of a matrix. We also have

$$\begin{pmatrix} A^T & 0^T \end{pmatrix} \begin{pmatrix} A \\ 0 \end{pmatrix} = \begin{pmatrix} 0^T & A^T \end{pmatrix} \begin{pmatrix} 0 \\ A \end{pmatrix} = A^TA + 0^T0 = A^TA, \tag{2.3}$$

hence the singular values of a matrix do not change if we add a null block of rows or columns to a matrix A.

Let us now list and prove some inequalities involving the 2-norm of a matrix $A = (\mathbf{a}_1 \dots \mathbf{a}_n)$.

Lemma 1. *For any matrix A we have*

$$\max_i \|\mathbf{a}_i\|_2 \leq \|A\|_2 \leq \sqrt{n} \max_i \|\mathbf{a}_i\|_2. \tag{2.4}$$

Proof. Let $\mathbf{e}_i$ be the i-th element of the canonical basis of $\mathbb{R}^n$. Then $A\mathbf{e}_i = \mathbf{a}_i$, and the left-hand inequality is proved. For the right-hand inequality, consider $\mathbf{x} \in \mathbb{R}^n$, then

$$A\mathbf{x} = \sum x_i \mathbf{a}_i \Rightarrow \|A\mathbf{x}\|_2 \leq \sum |x_i| \|\mathbf{a}_i\|_2.$$

Apply Cauchy-Schwarz inequality and take $\mathbf{x}$ such that $\|\mathbf{x}\| = 1$ to conclude

$$\|A\mathbf{x}\|_2 \leq \|\mathbf{x}\|_2 \sqrt{\sum_i \|\mathbf{a}_i\|_2^2} \leq \sqrt{n} \max_i \|\mathbf{a}_i\|_2.$$

□

The following inequalities are easy consequences of the result above.

Corollary 1. *For any matrix A we have*

$$\|A\|_{\max} \leq \|A\|_2 \leq \sqrt{mn} \|A\|_{\max}. \tag{2.5}$$

Proof. Let $\mathbf{a}_i$ be the i-th column of A. Then we have

$$\|\mathbf{a}_i\| = \sqrt{a_{i1}^2 + \cdots + a_{im}^2} \leq \sqrt{m} \max_j \sqrt{a_{ij}^2} = \sqrt{m} \max_j |a_{ij}|,$$

and, for any $1 \leq j \leq m$, we have

$$\|\mathbf{a}_i\| \geq \sqrt{a_{ij}^2} = |a_{ij}|.$$

Apply these inequalities to (2.4) to conclude. □

Corollary 2. *If A is a nonsingular and its inverse is partitioned into rows as*

$$A^{-1} = \begin{pmatrix} \mathbf{b}_1^T \\ \vdots \\ \mathbf{b}_n^T \end{pmatrix},$$

then

$$\sigma_{\min}(A) \leq \min_i (\|\mathbf{b}_i\|_2^{-1}) \leq \sqrt{n} \sigma_{\min}(A). \tag{2.6}$$

Proof. The Lemma above applied to A^{-T} yields

$$\max_i \|\mathbf{b}_i\|_2 \leq \|A^{-T}\|_2 \leq \sqrt{n} \max_i \|\mathbf{b}_i\|_2, \tag{2.7}$$

from which we deduce the left-hand inequality

$$\min_i(\|\mathbf{b}_i\|_2^{-1}) \geq \frac{1}{\|A^{-T}\|_2} = \frac{1}{\|A^{-1}\|_2} = \sigma_{\min}(A). \tag{2.8}$$

For the right-hand inequality, consider $\mathbf{x} \in \mathbb{R}^n$, then

$$\|A^{-1}\mathbf{x}\|_2^2 = \left\| \begin{array}{c} \mathbf{b}_1^T\mathbf{x} \\ \vdots \\ \mathbf{b}_n^T\mathbf{x} \end{array} \right\|_2^2 = \sum_i \left(\mathbf{b}_i^T\mathbf{x}\right)^2 \leq \sum_i \|\mathbf{b}_i\|_2^2 \, \|\mathbf{x}\|_2^2 \, ,$$

where we used Cauchy-Schwarz inequality. We have

$$\|A^{-1}\|_2^2 = \max_{\|\mathbf{x}\|_2=1} \|A^{-1}\mathbf{x}\|_2^2 \leq \max_{\|\mathbf{x}\|_2=1} \sum_i \|\mathbf{b}_i\|_2^2 \, \|\mathbf{x}\|_2^2 = \sum_i \|\mathbf{b}_i\|_2^2 \leq n \max_i \|\mathbf{b}_i\|_2^2 \, ,$$

from which we deduce

$$\min_i(\|\mathbf{b}_i\|_2^{-1}) \geq \frac{\sqrt{n}}{\|A^{-1}\|_2} = \sqrt{n}\sigma_{\min}(A).$$

□

2.1.2 About Strictly Diagonally Dominant matrices

A matrix A is said to be strictly diagonally dominant by rows if

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|,$$

for all i. We say A is strictly diagonally dominant by columns if A^T is strictly diagonally dominant by rows. The gap of diagonal dominance of a strictly diagonally dominant matrix A is the positive number γ defined as

$$\gamma = \min_i \left(1 - \sum_{j \neq i} |a_{ij}| \right).$$

The following result is taken from [120], and we prove it here for sake of completeness.

Lemma 2. *Let $M = I - S$, with $\|S\|_\infty < \frac{1}{2}$. Then M^{-1} exists, it has a positive diagonal and it is strictly diagonally dominant.*

Proof. In this case Neumann series converges, and we have

$$\overline{M} = M^{-1} = \sum_{k=0}^{\infty}(I-M)^k = \sum_{k=0}^{\infty} S^k = I + \sum_{k=1}^{\infty} S^k, \tag{2.9}$$

hence

$$\max_i \sum_j |\overline{M}|_{ij} = \|\overline{M}\|_\infty = \left\| I + \sum_{k=1}^{\infty} S^k \right\|_\infty \leq 1 + \left\| \sum_{k=1}^{\infty} S^k \right\|_\infty < 2,$$

since $\left\|\sum_{k=1}^{\infty} S^k\right\|_\infty \leq \sum_{k=1}^{\infty} \left\|S^k\right\|_\infty \leq \sum_{k=1}^{\infty} \|S\|_\infty^k < \sum_{k=1}^{\infty} \frac{1}{2}^k = 1$. Moreover, we also have that

$$1 > \left\| \sum_{k=1}^{\infty} S^k \right\|_\infty = \max_i \sum_j \left| \sum_{k=1}^{\infty} S_{ij}^k \right| \geq \sum_j \left| \sum_{k=1}^{\infty} S_{ij}^k \right| \geq \left| \sum_{k=1}^{\infty} S_{ij}^k \right|,$$

for all choices of i, j. Considering eq. (2.9) entrywise, we get

$$\overline{M}_{ij} = \left(\sum_{k=0}^{\infty} S^k \right)_{ij} = I_{ij} + \sum_{k=1}^{\infty} S_{ij}^k,$$

implying that $\overline{M}_{ii} > 0$. Moreover, we have

$$\begin{aligned}
\overline{M}_{ii} - \sum_{j\neq i} \left|\overline{M}_{ij}\right| &= 1 + \sum_{k=1}^{\infty} S_{ii}^k - \sum_{j\neq i} \left| \sum_{k=1}^{\infty} S_{ij}^k \right| \\
&> \sum_j \left| \sum_{k=1}^{\infty} S_{ij}^k \right| + \sum_{k=1}^{\infty} S_{ii}^k - \sum_{j\neq i} \left| \sum_{k=1}^{\infty} S_{ij}^k \right| \\
&= \sum_{j\neq i} \left(\left| \sum_{k=1}^{\infty} S_{ij}^k \right| - \left| \sum_{k=1}^{\infty} S_{ij}^k \right| \right) + \left| \sum_{k=1}^{\infty} S_{ii}^k \right| + \sum_{k=1}^{\infty} S_{ii}^k \\
&= \left| \sum_{k=1}^{\infty} S_{ii}^k \right| + \sum_{k=1}^{\infty} S_{ii}^k \geq 0,
\end{aligned}$$

therefore $\overline{M}$ is a strictly diagonally dominant matrix with a positive diagonal. □

Corollary 3. *Let $A = \alpha(I - S)$, with $\alpha > 0$ and $\|S\|_\infty < \frac{1}{2}$. Then A^{-1} exists, it has a positive diagonal and it is strictly diagonally dominant.*

Proof. Apply Lemma 2 to $I - S$, then $(I - S)^{-1}$ is strictly diagonally dominant with a positive diagonal, and so is $\alpha^{-1}(I - S)^{-1} = A^{-1}$. □

Let us state some results, for the proof see [142]. Let A be strictly diagonally dominant by rows, and set $\alpha = \min_i |a_{ii}| - \sum_{j\neq i} |a_{ij}| > 0$. Then

$$\|A^{-1}\| < \frac{1}{\alpha} \quad \Rightarrow \quad \|A^{-1}\|^{-1} = \sigma_{\min}(A) > \alpha. \tag{2.10}$$

If A is strictly diagonally dominant both by rows and columns, and $\beta = \min_j |a_{jj}| - \sum_{i \neq j} |a_{ij}| > 0$, then

$$\|A^{-1}\|^{-1} = \sigma_{\min}(A) \geq \sqrt{\alpha\beta}.$$

Lemma 3. *Let A be an $n \times n$ strictly diagonally dominant matrix, with $\gamma = \min_i 1 - \sum_{j \neq i} |a_{ij}/a_{ii}| > 0$. Let $D = \operatorname{diag}(d_1, \ldots, d_n)$, and let $1 \geq \tau > 0$ such that $|d_i| \geq \tau \bar{d} > 0$, where $\bar{d} = \max_i |d_i| > 0$, for all i. The matrix DAD is strictly diagonally dominant if $\gamma > 1 - \tau$.*

Proof. We have $(DAD)_{ij} = d_i d_j a_{ij}$. For all $1 \leq i \leq k$, we have

$$\sum_{j \neq i} |d_i d_j a_{ij}| = |d_i| \sum_{j \neq i} |d_j a_{ij}| \leq |d_i| \bar{d} \sum_{j \neq i} |a_{ij}|$$

and hence

$$d_i^2 |a_{ii}| - \sum_{j \neq i} |d_i d_j a_{ij}| \geq |d_i| \bar{d} \left(\tau |a_{ii}| - \sum_{j \neq i} |a_{ij}| \right).$$

Since $|d_i| \geq \tau \bar{d} > 0$ for all i, the quantity at the right-hand side of the above inequality is positive if and only if

$$\tau - \max_i \sum_{j \neq i} \frac{|a_{ij}|}{|a_{ii}|} > 0,$$

or, in our notation, $\gamma > 1 - \tau$. $\square$

2.2 The deviation maximization algorithm

Consider an $m \times n$ matrix A which has not full column rank, that is $\operatorname{rank}(A) = r < n$, and consider the problem of finding a subset of well conditioned columns of A. Before presenting a strategy to solve this problem, let us first introduce the notion of cosine matrix associated to a given matrix.

Definition 1. *Let $C = (\mathbf{c}_1 \ \ldots \ \mathbf{c}_k)$ be an $m \times k$ matrix whose columns $\mathbf{c}_i$ are non-null. Let D be the diagonal matrix with entries $D_{ii} = \|\mathbf{c}_i\|$, $1 \leq i \leq k$. Then the* ***cosine matrix*** *associated to C is defined as $\Theta = \Theta(C) = \left(CD^{-1}\right)^T CD^{-1} = D^{-1}C^TCD^{-1}$, and its entries are*

$$\theta_{ij} = \frac{\mathbf{c}_i^T \mathbf{c}_j}{\|\mathbf{c}_i\| \|\mathbf{c}_j\|} = \cos(\alpha_{ij}), \quad 1 \leq i, j \leq k. \tag{2.11}$$

where $\alpha_{ij} = \alpha(\mathbf{c}_i, \mathbf{c}_j)$ is the acute angle between $\mathbf{c}_i$ and $\mathbf{c}_j$.

In statistics, the just introduced notion of cosine matrix is better known as correlation matrix. It is immediate to see that the cosine matrix Θ is symmetric

positive semidefinite, it has only ones on the diagonal, and its entries range from -1 to 1. Here, in order to stress the linear algebra viewpoint, we call it cosine matrix.
The main idea of deviation maximization is to use the cosines to select a block of vectors whose pairwise deviations are large, i.e. two selected columns do not (nearly) belong to the same one dimensional subspace. Before formally presenting the procedure, let us state the following result.

Lemma 4. *Let $C = (\mathbf{c}_1 \ \dots \ \mathbf{c}_k)$ be an $m \times k$ matrix whose columns are nonzero. Suppose that $\|\mathbf{c}_1\| = \max_j \|\mathbf{c}_j\|$ and that there exists $1 \geq \tau > 0$ such that $\|\mathbf{c}_j\| \geq \tau \|\mathbf{c}_1\|$, for all $1 \leq j \leq k$. Let Θ be the cosine matrix associated to C and suppose that Θ is a strictly diagonally dominant matrix with*

$$\gamma = \min_i \left\{ 1 - \sum_{j \neq i} |\theta_{ij}| \right\} > 1 - \tau > 0.$$

Then

$$\sigma_{\min}(C) \geq \sqrt{\tau(\gamma + \tau - 1)}\ \|\mathbf{c}_1\|.$$

Proof. Let us first show that if Θ is a strictly diagonally dominant matrix. Then the symmetric positive definite matrix $C^T C = D \Theta D$, $D = \operatorname{diag}(\|\mathbf{c}_i\|)$ is strictly diagonally dominant. This follows from Lemma 3, since $|\theta_{ii}| = 1$ for all $1 \leq i \leq k$, and we have $\gamma > 1 - \tau$ by assumption.

Applying the bound (2.10), we have

$$\sigma_{\min}(C^T C) \geq \tau(\gamma + \tau - 1)\|\mathbf{c}_1\|^2 \quad \Rightarrow \quad \sigma_{\min}(C) \geq \sqrt{\tau(\gamma + \tau - 1)}\|\mathbf{c}_1\|.$$

□

The result above shows quite clearly that the bound on the smallest singular value of C depends on the norms of the column vectors and on the angles between each pair of such columns.

Recall that we would like to select k linearly independent and well-conditioned columns of A. The result above suggests to choose columns with indices $J = \{j_1, \dots, j_k\} \subseteq \{1, \dots, n\}$, with $k \leq r$, with a large euclidean norm, i.e. larger than a length defined by τ, and with large pairwise angles, meaning that the absolute values of their cosines are small enough. The set J can be chosen in a way such that the corresponding submatrix $C = A_J$ satisfies the hypotheses of Lemma 4.

Let us now present the deviation maximization method, which addresses this column selection task. Here, the information available on each column is encoded in two vectors $\mathbf{v}_1, \mathbf{v}_2$ of length n, one of which typically contains the column norm and the other some additional information, e.g. descend direction for optimization problems. Let us fix three parameters $\tau_1, \tau_2, \tau_\theta$, whose values range from 0 to 1. The deviation maximization incrementally builds the set J by computing a sequence of index subsets $J^{(0)} \subseteq J^{(1)} \subseteq \cdots \subseteq J$. The set $J^{(0)}$ is initialized with an index corresponding to the maximum element of $\mathbf{v}_1$, namely $J^{(0)} = \{j_1\}$, where $j_1 \in \operatorname{argmax} \mathbf{v}_1$. The set J is obtained by adding some indices within the candidate set I, defined as

$$I = \{i \;:\; (\mathbf{v}_1)_i \geq \tau_1 \max \mathbf{v}_1,\; i \neq j_1\}\,.$$

namely the set of column indices i such that the corresponding value $(\mathbf{v}_1)_i$ is larger than a given parameter τ_1 times the maximum value in $\mathbf{v}_1$. If the cardinality of I exceeds a value $k_{\max}$, then I is eventually restricted in order to limit memory usage and computational cost, mainly due to the creation of the cosine matrix $\Theta_I \coloneqq \Theta(C_I) = (\theta_{ij})$, where C_I denotes the submatrix of C with column indices in I.
Let $I = \{i_1, \ldots, i_{k_{\max}}\}$. At the l-th iteration, the index $i_l \in I$ is inserted in $J^{(l-1)}$ to give $J^{(l)}$ only if the i_l-th column has a large value $(\mathbf{v}_2)_{i_l}$ with respect to the parameter τ_2, that is

$$\|\mathbf{c}_{i_l}\| \geq \tau_2 \max_{i \in I} \|\mathbf{c}_i\|\,,$$

and if the corresponding cosines of the angles between the column i_l with the columns in $J^{(l-1)}$ are small with respect to the parameter τ_θ, that is

$$|\theta_{i_l,j}| < \tau_\theta, \qquad \text{for all } j \in J^{(l-1)}.$$

The resulting procedure of column selection is summarized in Algorithm 1. At

Algorithm 1 Deviation Maximization (DM)

```
    Inputs: $C, \mathbf{v}_1, \mathbf{v}_2, \tau_1, \tau_2, \tau_\theta, k_{\max}$
    Output: $J$
 1: $J^{(0)} = \{j_1 \;:\; j_1 \in \operatorname{argmax} \mathbf{v}_1\}$
 2: $I = \{i \;:\; (\mathbf{v}_1)_i \geq \tau_1 \max \mathbf{v}_1,\; i \neq j\}$
 3: **if** $|I| > k_{\max}$ **then**
 4:     $I = \{i_l \in I \;:\; l = 1, \ldots, k_{\max}\}$
 5: **end if**
 6: compute the cosine matrix $\Theta$ associated to $C_I$
 7: **for** $l = 1, \ldots, k_{\max}$ **do**
 8:     **if** $(\mathbf{v}_2)_{i_l} > \tau_2 \ \max \mathbf{v}_2$ and $|\theta_{i_l,j}| < \tau_\theta, \forall j \in J^{(l-1)}$ **then**
 9:         $J^{(l)} = J^{(l-1)} \cup \{i_l\}$
10:     **else**
11:         $J^{(l)} = J^{(l-1)}$
12:     **end if**
13: **end for**
14: $J = J^{(k_{\max})}$
```

the end of the iterations, we have $J = J^{(k_{\max})} = \{j_1, \ldots, j_k\}$, with $1 \leq k \leq k_{max}$. Notice that the following choice of the parameter τ_θ, namely

$$\tau_\theta \leq \frac{1 - \gamma}{k_{\max} - 1}, \tag{2.12}$$

allows to identify $\Theta_J \coloneqq \Theta(C_J)$, i.e. a square submatrix of Θ_I, whose gap of diagonal dominance is larger than a value γ. Indeed for every $j \in J$ we have

$$\sum_{\substack{i \in J \\ i \neq j}} |\theta_{ij}| < (k_{\max} - 1)\frac{1-\gamma}{k_{\max}-1} = 1 - \gamma,$$

and hence

$$\min_{j \in J} \left\{ 1 - \sum_{\substack{i \in J \\ i \neq j}} |\theta_{ij}| \right\} > \gamma. \tag{2.13}$$

At the end of the deviation maximization, we clearly have

$$\begin{aligned} (\mathbf{v}_1)_j &\geq \tau_1 \max \mathbf{v}_1(J), \\ (\mathbf{v}_2)_j &\geq \tau_2 \max \mathbf{v}_2(J), \end{aligned}$$

for every $j \in J$. When $\mathbf{v}_2$ (or $\mathbf{v}_1$) is the column norms vector, in order to make C_J satisfy Lemma 4 it is sufficient to set

$$\tau_2 > 1 - \gamma \qquad (\text{or } \tau_1 > 1 - \gamma)\,, \tag{2.14}$$

for a given $1 \geq \gamma > 0$. Notice that when γ is not available, the following choice of τ_θ also ensures that C_J satisfies Lemma 4, that is

$$\tau_\theta < \frac{\tau_2}{k_{\max}-1} \qquad \left(\text{or } \tau_\theta < \frac{\tau_1}{k_{\max}-1}\right).$$

Let us briefly comment the choice of the parameters τ_1, τ_2 and τ_θ. The value of parameter τ_1 should be small in order to get a large candidate set I. Similarly, a small value of parameter τ_2 likely yields a larger $k = |J|$. For what concerns τ_θ, when its value is equal (or close) to zero only pairwise (nearly) orthogonal columns are accepted to be inserted in the set J. However, this is a rare occurrence in real world problems, and therefore a large value of τ_θ is desirable.

The procedure here presented exploits diagonal dominance in order to ensure linear independence. In practice, this often turns out to be a too strong condition to be satisfied, and as a result the number k of columns found is usually way smaller than the matrix rank r. Indeed, diagonal dominance of the cosine matrix is a sufficient but obviously not a necessary condition for linear independence of a set of column vectors, and this suggests that it is not necessary to choose τ_θ accordingly to (2.12).

Deviation maximization may be adopted to perform block column pivoting in various algorithms that deal with column selection. In practice, the values of parameters τ_1, τ_2 and τ_θ can be tied to the properties of the specific algorithm that uses the deviation maximization as pivoting. In what follows, we successfully apply the deviation maximization pivoting to derive a rank-revealing QR decomposition and to the solution of nonnegative least squares problems.

2.3 Implementation of deviation maximization algorithm

In this section we discuss implementation aspects of deviation maximization. In particular, we address the following issues

1. the practical computation of the candidate set I defined in (2.2);
2. the efficient computation of the cosine matrix Θ defined in (2.11);
3. the structure of the pivoting, which has a significant impact on the cost of the overall algorithm.

Let us first focus on some details of the implementation of the deviation maximization presented in Algorithm 1.

The candidate set I can be computed with a fast sorting algorithm, e.g. quicksort, applied to the array of partial column norms.

The most expensive operation in Algorithm 1 is the computation of the cosine matrix in step 6. If we write the matrix C by columns $C = (\mathbf{c}_1 \ \dots \ \mathbf{c}_n)$, then the cosine matrix restricted to the candidate set Θ_I has entries $\theta_{ij} = \mathbf{c}_i^T \mathbf{c}_j \|\mathbf{c}_i\|^{-1} \|\mathbf{c}_j\|^{-1}$, for $i, j \in I$. Therefore we have

$$\Theta_I = D^{-1} C_I^T C_I D^{-1}, \tag{2.15}$$

where $D = \mathrm{diag}(d_i)$ is a diagonal matrix, with $d_i = \|\mathbf{c}_i\|$, $i \in I$. The matrix Θ_I is symmetric, thus we only need its upper (lower) triangular part. This can be computed in two ways

(i) we first form the product $U_1 = C_I D^{-1}$, and then we compute $\Theta_I = U_1^T U_1$;

(ii) we first form the product $U_2 = C_I^T C_I$, and then we compute $\Theta_I = D^{-1} U_2 D^{-1}$.

The former approach requires $m \times n$ additional memory to store U_1 and it requires $m^2 k_{\max}^2$ flops to compute U_1 and $(2m-1)k_{\max}(k_{\max}-1)/2$ flops for the upper triangular part of $U_1^T U_1$, while the latter does not require additional memory since the matrix U_2 can be stored in the same memory space used for the cosine matrix Θ, and it requires $(2m-1)k_{\max}(k_{\max}-1)/2$ flops the upper triangular part of U_2 and $k_{\max}(k_{\max}-1)$ flops for the upper triangular part of $D^{-1} U_2 D^{-1}$. Therefore, we recommend the second approach, even if it requires to write an *ad hoc* low level routine which is not implemented in the BLAS library.

In order to limit the cost and the amount of additional memory of Algorithm 1, we propose a restricted version of deviation maximization pivoting. If the candidate is given by $I = \{i_l : l = 1, \dots, |I|\}$, then we limit its cardinality to be smaller or equal to a machine dependent parameter $k_{\max}$, that is

$$I = \{i_l : l = 1, \dots, \min(k_{\max}, |I|)\}. \tag{2.16}$$

We refer to the value $k_{\max}$ as block size.

Notice that the deviation maximization is aimed at identify a subset of numerically linearly independent columns. When some $\|\mathbf{c}_i\|$ is close to the working precision ϵ, we can consider $\mathbf{c}_i$ as a null vector. Hence, we require

$$\max_i \|\mathbf{c}_i\| > \mathcal{O}(\epsilon), \tag{2.17}$$

in order to carry out the deviation maximization procedure.

Last, we discuss the structure of the permutations employed in order to carry out deviation maximization pivoting, which has a significant impact on the cost of the algorithm. In fact, when the deviation maximization is used as pivoting strategy in order to exploit BLAS-3 level operations we need to physically move in memory the selected columns identified by $J = \{j_1, \dots, j_k\}$ to the left leading positions of C. Mathematically, we apply a permutation matrix Π from the right to compute $C\Pi$. The structure of the column exchanges determines the structure of Π and hence the cost of the communications in order to move selected columns in memory. We prefer permutations consisting of a sequence of cyclic shifts, that is a cyclic permutation involving only two elements and fixing all the others. In this way, the application of Π requires only m additional memory slots, that is the memory needed to swap two columns. Obviously, the fewer columns to swap the less the work involved in memory communications. A strategy that can easily be implemented consists in swapping the i-th column with the j_i-th column, for $i = 1, \dots, k$.

Chapter 3

Rank-revealing QR factorization

Rank-revealing QR (RRQR) factorization was introduced by Golub [80] and it is nowadays a classic topic in numerical linear algebra; for example, Golub and Van Loan [81] introduce RRQR factorization for least squares problems where the matrix has not full column rank: in such a case, a plain QR computation may lead to an R factor in which the number of nonzeros on the diagonal does not equal the rank and the matrix Q does not reveal the range nor the null space of the original matrix. Here, the SVD decomposition is the safest and most expensive solution method, while approaches based on a modified QR factorization can be seen as cheaper alternatives. Since the QR factorization is essentially unique once the column ordering is fixed, these techniques all amount to finding an appropriate column permutation. The first algorithm was proposed in [33] and it is referred as QR factorization with column pivoting (QRP). It should be noticed that, if the matrix of the least squares problem has not full column rank, then there is an infinite number of solutions. We must resort to rank revealing techniques which identify a particular solution as “special”. QR with column pivoting identify a particular *basic* solution (with at most r nonzero entries, where r is the rank), while biorthogonalization methods [81], identify the minimum ℓ_2 solution. Rank-revealing decompositions can be used in a number of other applications [90].

The QR factorization with column pivoting works pretty well in practice, even if there are some examples in which it fails, see e.g. the Kahan matrix [96]. However, further improvements are possible, see e.g. Chan [42] and Foster [75]: the idea here is to identify and remove small singular values one by one. Gu and Eisenstat [83] introduced the Strong RRQR factorization, a stable algorithm for computing a RRQR factorization with a good approximation of the null space, which is not guaranteed by QR factorization with column pivoting. Both can be used as optional improvements to the QR factorization with column pivoting. Rank revealing QR factorizations were also treated in [82, 43, 93].

Column pivoting makes it more difficult to achieve high performances in QR computation, see [17, 18, 19, 119, 20]. The state-of-the-art algorithm for computing RRQR, named QP3, is a block version [119] of the standard column pivoting and it is currently implemented in LAPACK [6]. Other recent high-performance approaches are tournament pivoting [55] and randomized pivoting [71, 147, 107]. In this chapter we apply the deviation maximization column selection technique to the problem of computing a RRQR factorization, deriving an alternative block algorithm to QP3 we call QRDM.

The rest of this chapter is organized as follows. In Section 3.1 we define rank-revealing factorizations, we review the QRP algorithm and then we introduce QRDM, a block algorithm for RRQR by means of deviation maximization; furthermore, we give theoretical worst case bounds for the smallest singular value of the R factor of the RRQR factorizations obtained with these two methods. In Section 3.1.5 we discuss some fundamental issues regarding the implementation of QRDM; in Section 3.2 we compare QP3 and QRDM against a relevant database of singular matrices.

3.1 Rank-Revealing QR decompositions

Let us introduce the mathematical formulation for the problem of finding a rank-revealing decomposition of a matrix A of size $m \times n$. We say that the matrix A has numerical rank $1 \le r \le \min(m,n)$ if $\sigma_{r+1}(A) \ll \sigma_r(A)$ and $\sigma_{r+1}(A) \approx \epsilon$, see [43], where ϵ is the *machine precision.* Let Π denote a permutation matrix of size n, then we can compute

$$A\Pi = QR = (Q_1 \; Q_2) \begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix}, \tag{3.1}$$

where Q is an orthogonal matrix of order m, $Q_1 \in \mathbb{R}^{m\times r}$ and $Q_2 \in \mathbb{R}^{m\times(m-r)}$, R_{11} is upper triangular of order r, $R_{12} \in \mathbb{R}^{r\times(n-r)}$ and $R_{22} \in \mathbb{R}^{(m-r)\times(n-r)}$. The blank space in the left bottom corner has to be intended as a zero block of size $(m-r)\times r$. The QR factorization above is called *rank-revealing* if

$$\sigma_{\min}(R_{11}) = \sigma_r(R_{11}) \approx \sigma_r(A),$$

or

$$\sigma_{\max}(R_{22}) = \sigma_1(R_{22}) \approx \sigma_{r+1}(A),$$

or both conditions hold. Notice that if $\sigma_{\min}(R_{11}) \gg \epsilon$ and $\|R_{22}\|$ is small, then the matrix A has numerical rank r, but the converse is not true. In other words, even if A has $(\min(m,n)-r)$ small singular values, it is does not follow that any permutation Π yields a small $\|R_{22}\|$, even if there exist strategies that ensure a small value of $\|R_{22}\|$ by identifying and removing small singular values, see e.g. [42, 75]. It is easy to show that for any factorization like (3.1) the following relations hold

$$\sigma_{\min}(R_{11}) \le \sigma_r(A), \tag{3.2}$$

$$\sigma_{\max}(R_{22}) \ge \sigma_{r+1}(A). \tag{3.3}$$

The proof directly follows by applying the interlacing inequalities for singular values [133], namely

$$\sigma_k(A) \geq \sigma_k(B) \geq \sigma_{k+r+s}(A), \quad k \geq 1,$$

which hold for any $(m-s) \times (n-r)$ submatrix B of A. In fact we have

$$\sigma_{\min}(R_{11}) = \sigma_{\min} \begin{pmatrix} R_{11} \\ 0 \end{pmatrix} = \sigma_r((Q^T A\Pi)(:, 1:r)) \leq \sigma_r(Q^T A\Pi) = \sigma_r(A),$$

$$\sigma_{\max}(R_{22}) = \sigma_{\max}(0\ R_{22}) = \sigma_1((Q^T A\Pi)(r+1:m, :)) \geq \sigma_{r+1}(Q^T A\Pi) = \sigma_{r+1}(A).$$

We also used the invariance of the singular values under orthogononal transformations and under the insertion of a zero block, see equations (2.1)–(2.3). Ideally, the best rank-revealing QR decomposition is obtained by the column permutation Π which solves

$$\max_{\Pi} \sigma_{\min}(R_{11}). \tag{3.4}$$

However, the problem above clearly has a combinatorial nature. Therefore, algorithms that compute RRQR usually provide (see, e.g. [43, 93]) at least one of the following bounds

$$\sigma_{\min}(R_{11}) \geq \frac{\sigma_r(A)}{p(n)}, \tag{3.5}$$

$$\sigma_{\max}(R_{22}) \leq \sigma_{r+1}(A)q(n), \tag{3.6}$$

where $p(n)$ and $q(n)$ are low degree polynomials in n. These are worst case bounds and are usually not sharp. We provide a bound of type (3.5) in Section 3.1.3.

3.1.1 QR factorization with column pivoting

Let us introduce the QR factorization with column pivoting proposed by Businger and Golub [33], which can be labeled as a greedy approach in order to cope with the combinatorial optimization problem (3.4). Suppose at the s-th algorithmic step we have already selected $s < r$ well-conditioned columns of A, which are moved to the leading positions by the permutation matrix $\Pi^{(s)}$ as follows

$$A\Pi^{(s)} = Q^{(s)}R^{(s)} = Q^{(s)} \begin{pmatrix} R_{11}^{(s)} & R_{12}^{(s)} \\ & R_{22}^{(s)} \end{pmatrix}, \tag{3.7}$$

where $R_{11}^{(s)}$ is an upper triangular block of order s, and the blocks $R_{12}^{(s)}$ and $R_{22}^{(s)}$ have size $s \times (n-s)$ and $(m-s) \times (n-s)$ respectively. The block $R_{22}^{(s)}$ is what is left to be processed, and it is often called "trailing matrix". Let us introduce the following column partitions for $R_{12}^{(s)}$, $R_{22}^{(s)}$ respectively

$$\begin{aligned} R_{12}^{(s)} &= \left(\mathbf{b}_1^{(s)} \dots \mathbf{b}_{n-s}^{(s)}\right), \\ R_{22}^{(s)} &= \left(\mathbf{c}_1^{(s)} \dots \mathbf{c}_{n-s}^{(s)}\right). \end{aligned} \tag{3.8}$$

We aim to select, within the $n-s$ remaining columns, the column such that the condition number of the block $R_{11}^{(s+1)}$ is the largest possible. Formally, we would like to find the index j_s that solves

$$\sigma_{\min}\begin{pmatrix} R_{11}^{(s)} & \mathbf{b}_{j_s}^{(s)} \\ & \mathbf{c}_{j_s}^{(s)} \end{pmatrix} = \max_{1\le i\le n-s} \sigma_{\min}\begin{pmatrix} R_{11}^{(s)} & \mathbf{b}_{i}^{(s)} \\ & \mathbf{c}_{i}^{(s)} \end{pmatrix}. \tag{3.9}$$

Define the vector $\mathbf{u}^{(s)}$ whose entry $u_i^{(s)}$ is the i-th partial column norm of $A\Pi^{(s)}$, that is the norm of the subcolumn with row indices ranging from $m-n_s$ to m, namely

$$\begin{cases} u_i^{(s)} = \left\| \mathbf{c}_{i-s}^{(s)} \right\|, & s < i \le n, \\ u_i^{(s)} = 0, & i \le s. \end{cases}$$

Using the following fact

$$\sigma_{\min}\begin{pmatrix} R_{11}^{(s)} & \mathbf{b}_{j_s}^{(s)} \\ & \mathbf{c}_{j_s}^{(s)} \end{pmatrix} = \sigma_{\min}\begin{pmatrix} R_{11}^{(s)} & \mathbf{b}_{j}^{(s)} \\ & u_{s+j_s}^{(s)} \end{pmatrix},$$

which is a simple consequence of the invariance of singular values under left multiplication by orthogonal matrices and the insertion of null rows (2.3), and using the bound (2.6), we can approximate (up to a factor $\sqrt{s+1}$) the smallest singular value as

$$\sigma_{\min}\begin{pmatrix} R_{11}^{(s)} & \mathbf{b}_{j_s}^{(s)} \\ & \mathbf{c}_{j_s}^{(s)} \end{pmatrix} \approx \min_h \left\| \mathbf{e}_h^T \begin{pmatrix} R_{11}^{(s)} & \mathbf{b}_{j_s}^{(s)} \\ & u_{s+j_s}^{(s)} \end{pmatrix}^{-1} \right\|^{-1},$$

where $\mathbf{e}_h$ is the h-th element of the canonical basis of $\mathbb{R}^{s+1}$.

Using this result, as argued in [43], the maximization problem (3.9) can be solved approximately by solving

$$j_s = \operatorname*{argmax}_{s+1\le i\le n} u_i^{(s)} \approx \operatorname*{argmax}_{1\le i\le n-s} \sigma_{\min}\begin{pmatrix} R_{11}^{(s)} & \mathbf{b}_{i}^{(s)} \\ & \mathbf{c}_{i}^{(s)} \end{pmatrix}.$$

Notice that this strategy requires to compute the norms of columns $\mathbf{c}_i^{(s)}$, $1 \le i \le n-n_s$, whose size changes at each iteration. For an efficient implementation, the column norms of the trailing matrix can be updated, instead of being recomputed from scratch, by exploiting the following property [81]

$$Q\mathbf{a} = \begin{pmatrix} \beta \\ \mathbf{c} \end{pmatrix} \begin{matrix} \in \mathbb{R} \\ \in \mathbb{R}^{m-1} \end{matrix} \quad \Rightarrow \|\mathbf{a}\|^2 = \|Q\mathbf{a}\|^2 = \beta^2 + \|\mathbf{c}\|^2,$$

which holds for any orthogonal matrix Q and any vector $\mathbf{a}$ of order m. Therefore, once initialized the vector $\mathbf{u}^{(0)}$, whose entries are $u_i^{(0)} = \|\mathbf{a}_i\|^2$, with $1 \le i \le n$, we can perform the following update

$$u_i^{(s+1)} = \begin{cases} \sqrt{\left(u_i^{(s)}\right)^2 - r_{si}^2}, & s+1 \le i \le n, \quad 1 \le s \le n, \\ 0, & i < s+1, \end{cases} \tag{3.10}$$

where r_{si} is the entry of indices (s, i) in $R^{(s)}$, $1 \leq s \leq m$, $1 \leq i \leq n$. The partial column norm update allows to reduce the operation count from $\mathcal{O}(mn^2)$ to $\mathcal{O}(mn)$. Actually, the formula (3.10) cannot be applied as it is because of numerical cancellation, and it needs to modified, see e.g. [70] for a robust implementation.

The resulting procedure is referred as QR factorization with column pivoting, and it is presented in Algorithm 2. In a real implementation, the QR decomposition in computed in place: the upper triangular part of A if overwritten with the triangular factor R, the lower triangular part of A together with n additional vector of size n is used to store the Householder vectors that form the orthogonal factor Q and a vector of size n is used to represent the permutation matrix Π. A block version of Algorithm 2 has been proposed [119], and it is currently implemented in LAPACK's `xgeqp3` routine, that we will use in the numerical section for comparison.

Algorithm 2 QR with column pivoting (QRP)

```
   Inputs: $A$
   Outputs: $Q, R, \Pi$
1: initialize $\mathbf{u}^{(0)} = (u_i) = (\|\mathbf{a}_i\|)$, $R^{(0)} = A$, $Q^{(0)} = I$, $\Pi^{(0)} = I$
2: for $s = 0, \ldots, n-1$ do
3:     $j = \operatorname{argmax} \mathbf{u}^{(s)}(s+1 : n)$
4:     move the element of index $s+j$ to the leading position $s+1$ of $R^{(s)}$ and $\mathbf{u}^{(s)}$
5:     compute the Householder reflector $\mathbf{v}^{(s)}$ w.r.t. $R_{22}^{(s)}(1)$
6:     update the trailing matrix $R^{(s+1)}(s+1 : m, s+1 : n) = \left(I - \mathbf{v}^{(s)}(\mathbf{v}^{(s)})^T\right) R_{22}^{(s)}$
7:     update $\mathbf{u}^{(s+1)}$, $Q^{(s+1)}$, $\Pi^{(s+1)}$
8: end for
9: $Q = Q^{(n-1)}$, $R = R^{(n-1)}$, $\Pi = \Pi^{(n-1)}$
```

Remark 1. *Such a choice of pivot can be explained with a geometric interpretation. Introduce the following block column partitioning* $R^{(s)} = \left(R_1^{(s)}\ R_2^{(s)}\right)$, $Q^{(s)} = \left(Q_1^{(s)}\ Q_2^{(s)}\right)$, *and recall that we have*

$$\mathcal{R}\left(Q_1^{(s)}\right) = \mathcal{R}\left(R_1^{(s)}\right) \quad \mathcal{R}\left(Q_2^{(s)}\right) = \mathcal{R}\left(R_1^{(s)}\right)^{\perp}.$$

where $\mathcal{R}(B)$ *denotes the subspace spanned by the columns of a matrix* B. *Every unprocessed column of* A *rewrites as*

$$\mathbf{a}_i = Q_1^{(s)}\mathbf{b}_{i-s}^{(s)} + Q_2^{(s)}\mathbf{c}_{i-s}^{(s)},$$

where $Q_1^{(s)}\mathbf{b}_{i-s}^{(s)}$ *and* $Q_2^{(s)}\mathbf{c}_{i-s}^{(s)}$ *are the orthogonal projections of* $\mathbf{a}_i$ *on* $\mathcal{R}\left(R_1^{(s)}\right)$ *and* $\mathcal{R}\left(R_1^{(s)}\right)^{\perp}$ *respectively. The most linearly independent column* $\mathbf{a}_i$ *from the* *ones already processed can be seen as the one with the largest orthogonal projection on the complemented subspace of the subspace spanned by such columns, namely*

$$\max_{i>s} \left\| \mathcal{P}_{\mathcal{R}\left(R_1^{(s)}\right)^{\perp}} (\mathbf{a}_i) \right\| = \max_{i>s} \left\| Q_2^{(s)} \mathbf{c}_{i-s}^{(s)} \right\| .$$

However, the matrix $Q^{(s)}$ is never directly available unless it is explicitly computed. We then settle for the the solution of the maximization problem

$$\max_{i>s} \left\| \mathbf{c}_{i-s}^{(s)} \right\| .$$

3.1.2 QR factorization with deviation maximization pivoting

Consider the partial factorization in eq. (3.7), and now suppose at the s-th algorithmic step we have already selected n_s, with $s \leq n_s < r$, linearly independent and well-conditioned columns of A, so that $R_{11}^{(s)}$ is upper triangular of order n_s, while blocks $R_{12}^{(s)}$ and $R_{22}^{(s)}$ have size $n_s \times (n - n_s)$ and $(m - n_s) \times (n - n_s)$, respectively.

We aim to select k_s, with $n_{s+1} = n_s + k_s \leq r$, linearly independent and well-conditioned columns from the remaining $n - n_s$ columns of A, which are also sufficiently linearly independent from the n_s columns already selected, in order to keep the smallest singular value of the R_{11} block as large as possible. Ideally, we look for those columns with indices $j_1, \dots, j_{k_s}$ that solve

$$\sigma_{\min} \begin{pmatrix} R_{11}^{(s)} & \mathbf{b}_{j_1}^{(s)} & \dots & \mathbf{b}_{j_{k_s}}^{(s)} \\ & \mathbf{c}_{j_1}^{(s)} & \dots & \mathbf{c}_{j_{k_s}}^{(s)} \end{pmatrix} = \max_{1 \leq i_1, \dots, i_{k_s} \leq n - n_s} \sigma_{\min} \begin{pmatrix} R_{11}^{(s)} & \mathbf{b}_{i_1}^{(s)} & \dots & \mathbf{b}_{i_{k_s}}^{(s)} \\ & \mathbf{c}_{i_1}^{(s)} & \dots & \mathbf{c}_{i_{k_s}}^{(s)} \end{pmatrix} . \tag{3.11}$$

Of course, this maximization problem has the same combinatorial nature as problem (3.4), so we rather solve it approximately. We propose to approximate the indices $\{j_1, \dots, j_{k_s}\}$ that solve problem (3.11) with the indices selected by the deviation maximization procedure presented in Algorithm 1 applied to the trailing matrix $R_{22}^{(s)}$. In order to define the candidate set exploiting the column norm vector, we set $\mathbf{v}_1 = \mathbf{u}^{(s)}(n_s + 1 : n)$. Since no additional information is available, we set $\mathbf{v}_2 = \mathbf{0}$ and $\tau_2 = 0$, while suppose the values of τ_θ and $\tau_u := \tau_1$ are chosen accordingly to equations (2.12) and (2.14), respectively. More efficient choices will be widely discussed in Section 3.2. To sum up, the indices $j_1, \dots, j_{k_s}$ are obtained by Algorithm 1 as follows

$$\{j_1, \dots, j_{k_s}\} = \mathrm{DM}\left(R_{22}^{(s)}, \mathbf{u}^{(s)}(n_s + 1 : n), \mathbf{0}, \tau_u, 0, \tau_\theta, k_{\max}\right) .$$

For sake of brevity, let us write

$$B^{(s)} = \left(\mathbf{b}_{j_1}^{(s)} \dots \mathbf{b}_{j_{k_s}}^{(s)}\right) \in \mathbb{R}^{n_s \times k},$$
$$C^{(s)} = \left(\mathbf{c}_{j_1}^{(s)} \dots \mathbf{c}_{j_{k_s}}^{(s)}\right) \in \mathbb{R}^{(m-n_s) \times k},$$

and by $\overline{B}^{(s)}$ and $\overline{C}^{(s)}$ the matrices made up by the remaining unselected columns. The rest of the block update, which we detail below, proceeds in an analogous way to the recursive block QR. Let $\widetilde{Q}^{(s)}$ be an orthogonal matrix of order $(m - n_s)$ such that

$$\left(\widetilde{Q}^{(s)}\right)^T C^{(s)} = \begin{pmatrix} T^{(s)} \\ 0 \end{pmatrix}, \tag{3.12}$$

where $T^{(s)}$ is an upper triangular matrix of size $(m - n_s) \times k_s$. The matrix $\widetilde{Q}^{(s)}$ is obtained as a product of k_s Householder reflectors, that we represent by means of the so-called compact WY form [124] as

$$\widetilde{Q}^{(s)} = I - Y^{(s)} W^{(s)} \left(Y^{(s)}\right)^T,$$

where $Y^{(s)}$ is lower trapezoidal with k_s columns and $W^{(s)}$ is upper triangular of order k_s. This allows us to carry out the update of the rest of trailing matrix by means of BLAS-3 kernels, for performance efficiency, that is

$$\left(\widetilde{Q}^{(s)}\right)^T \overline{C}^{(s)} = \begin{pmatrix} \overline{T}^{(s)} \\ R_{22}^{(s+1)} \end{pmatrix}, \tag{3.13}$$

where $\overline{T}^{(s)}$ has size $k_s \times (n - n_{s+1})$ and the new trailing matrix $R_{22}^{(s+1)}$ has size $(m - n_{s+1}) \times (n - n_{s+1})$, for $n_{s+1} = n_s + k_s$. Denoting by $\widetilde{\Pi}^{(s)}$ a permutation matrix that moves columns with indices $j_1, \ldots, j_{k_s}$ to the current leading positions, we set $\Pi^{(s+1)} = \Pi^{(s)} \widetilde{\Pi}^{(s)}$ and

$$Q^{(s+1)} = Q^{(s)} \begin{pmatrix} I & \\ & \widetilde{Q}^{(s)} \end{pmatrix} \in \mathbb{R}^{m \times m},$$

then the overall factorization of $A\Pi^{(s+1)}$ takes the form

$$Q^{(s)} \begin{pmatrix} R_{11}^{(s)} & B^{(s)} & \overline{B}^{(s)} \\ & C^{(s)} & \overline{C}^{(s)} \end{pmatrix} = Q^{(s+1)} \begin{pmatrix} R_{11}^{(s)} & B^{(s)} & \overline{B}^{(s)} \\ & T^{(s)} & \overline{T}^{(s)} \\ & & R_{22}^{(s+1)} \end{pmatrix}, \tag{3.14}$$

where, for the successive iteration, we set

$$\begin{aligned} R_{11}^{(s+1)} &= \begin{pmatrix} R_{11}^{(s)} & B^{(s)} \\ & T^{(s)} \end{pmatrix} \in \mathbb{R}^{n_{s+1} \times n_{s+1}}, \\ R_{12}^{(s+1)} &= \begin{pmatrix} \overline{B}^{(s)} \\ \overline{T}^{(s)} \end{pmatrix} \in \mathbb{R}^{n_{s+1} \times (n - n_{s+1})}. \end{aligned}$$

Last, we point out that the partial column norms can be updated at each iteration also in this case with some straightforward changes of equation (3.10), namely

$$u_j^{(s+1)} = \begin{cases} \sqrt{\left(u_j^{(s)}\right)^2 - \sum_{l=n_s}^{n_{s+1}} r_{lj}^2}, & n_{s+1} < j \le n, \quad n_{s+1} \le n, \\ 0, & j \le n_{s+1}. \end{cases}$$

The resulting procedure is called QR factorisation with deviation maximization pivoting and it presented in Algorithm 3. The QRP algorithm has the particular

Algorithm 3 QR with Deviation Maximization (QRDM)

```
    Inputs: $A, \tau_u, \tau_\theta, k_{\max}$
    Outputs: $Q, R, \Pi$
 1: set $n_s = 0$, $s = 0$, $\mathbf{u}^{(0)} = (u_i) = (\|\mathbf{a}_i\|)$, $R^{(0)} = A$, $Q^{(0)} = I$, $\Pi^{(0)} = I$
 2: while $n_s < n$ do
 3:     $\{j_1, \ldots, j_{k_s}\} = \mathrm{DM}\left(R_{22}^{(s)}, \mathbf{u}^{(s)}(n_s + 1 : n), \mathbf{0}, \tau_u, 0, \tau_\theta, k_{\max}\right)$
 4:     move elements of indices $n_s + j_1, \ldots, n_s + j_{k_s}$ to the leading positions
        $n_s + 1, \ldots, n_s + k_s$ of $R^{(s)}$ and $\mathbf{u}^{(s)}$
 5:     for $l = 1, \ldots, k_s$ do
 6:         compute the Householder reflector $\mathbf{v}^{(n_s+l)}$ w.r.t. $R_{22}^{(s)}(l : (m - n_s), l)$
 7:         update $R^{(s+1)}(n_s + l : m, n_s + l : n_s + k_s) =$
            $\left(I - \mathbf{v}^{(n_s+l)}(\mathbf{v}^{(n_s+l)})^T\right) R_{22}^{(s)}(l : (m - n_s), l : k_s)$
 8:     end for
 9:     compute the block representation $W^{(s)}, Y^{(s)}$ of $\mathbf{v}^{(n_s+1)}, \ldots, \mathbf{v}^{(n_s+k_s)}$
10:     block update $R^{(s+1)}(n_s + 1 : m, n_s + k_s + 1 : n) =$
        $\left(I - Y^{(s)} (W^{(s)})^T (Y^{(s)})^T\right) R_{22}^{(s)}(k_s + 1 : (n - n_s))$
11:     update $\mathbf{u}^{(s+1)}$, $Q^{(s+1)}$, $\Pi^{(s+1)}$
12:     $n_s = n_s + k_s$, $s = s + 1$
13: end while
14: $Q = Q^{(s)}$, $R = R^{(s)}$, $\Pi = \Pi^{(s)}$
```

feature that the diagonal elements of the final upper triangular factor R are monotonically non increasing in modulus. This property cannot be guaranteed by the QRDM algorithm, as by other recently proposed pivoting strategies [55]. In practice, there are small fluctuations around a non-increasing trend.

3.1.3 Worst-case bound on the smallest singular value

Let us denote by $\bar{\sigma}^{(s)}$ the smallest singular value of the computed $R_{11}^{(s)}$ block at step s, that is

$$\bar{\sigma}^{(s)} = \sigma_{\min}\left(R_{11}^{(s)}\right).$$

Notice that it corresponds exactly to the s-th singular value of $R_{11}^{(s)}$ computed with the standard column pivoting, while it corresponds to the n_s-th singular value when $R_{11}^{(s)}$ is computed with the deviation maximization. Let us first state an estimate of $\bar{\sigma}^{(s+1)}$ for QRP [43].

Theorem 1. *Let $R_{11}^{(s)}$ be the upper triangular factor of order s computed by Algorithm 2. Then we have*

$$\bar{\sigma}_{s+1} \geq \sigma_{s+1}(A) \frac{\bar{\sigma}_s}{\sigma_1(A)} \frac{1}{\sqrt{2(n-s)(s+1)}}.$$

Proof. We make use of (2.6) in order to estimate $\bar{\sigma}_{s+1}$ by the reciprocal of the largest 2-norm of the rows of the inverse

$$\left(R_{11}^{(s+1)}\right)^{-1} = \begin{pmatrix} R^{-1} & -R^{-1}\mathbf{b}u^{-1} \\ & u^{-1} \end{pmatrix},$$

where we dropped the subscript and the superscript on $\mathbf{b}_{j_s}^{(s)}$, $u_{s+j_s}^{(s)}$, and on the matrix $R_{11}^{(s)}$ and its inverse, which will be denoted as $\mathbf{b}$, u, R and R^{-1} respectively.
The norm of the row with the largest norm among the leading s rows is bounded by

$$\max_i \|\mathbf{e}_i^T R^{-1}\| + \|R^{-1}\mathbf{b}u^{-1}\| \leq \frac{1}{\bar{\sigma}_s} + \frac{\|\mathbf{b}\|}{\bar{\sigma}_{s+1}u}.$$

Since $a + b \leq \sqrt{2}\sqrt{a^2 + b^2}$, we have

$$\max_i \|\mathbf{e}_i^T R^{-1}\| + \|R^{-1}\mathbf{b}u^{-1}\| \leq \sqrt{2}\frac{\sqrt{u^2 + \|\mathbf{b}\|}}{\bar{\sigma}_s u} \leq \sqrt{2}\frac{\|A\|}{\bar{\sigma}_s u},$$

where the last inequality follows from the fact that $u^2 + \|\mathbf{b}$ is the norm of the $s+j_s$-th column of $A\Pi^{(s)}$. The norm $u^{-1}$of the $s+1$-st row of the inverse clearly cannot exceed the bound for the leading s rows. Using (2.4) and the interlacing property of singular values (3.3) we have

$$u = \max_{1 \leq i \leq n-s} u_{s+i}^{(s)} \geq \frac{\sigma_{s+1}(A)}{\sqrt{n-s}}.$$

Last, (2.6) yields

$$\bar{\sigma}_{s+1} \geq u\frac{\bar{\sigma}_s}{\sigma_1(A)}\frac{1}{\sqrt{2(s+1)}} \geq \sigma_{s+1}(A)\frac{\bar{\sigma}_s}{\sigma_1(A)}\frac{1}{\sqrt{2(n-s)(s+1)}}.$$

□

Before coming to the main result, we introduce the following auxiliary Lemma.

Lemma 5. *Consider the s-th algorithmic step of Algorithm 3. With reference to the notation used for introducing the block partition in* (3.14), *we have*

$$\sigma_{\min}\left(T^{(s)}\right) \geq \frac{\sqrt{\tau_u(\gamma + \tau_u - 1)}}{\sqrt{n - n_{s+1} + 1}}\sigma_{n_{s+1}}(A). \tag{3.15}$$

Proof. Consider following column partitions

$$T^{(s)} = \left(\mathbf{t}_1^{(s)} \dots \mathbf{t}_{k_s}^{(s)}\right),$$
$$\overline{T}^{(s)} = \left(\mathbf{t}_{k_s+1}^{(s)} \dots \mathbf{t}_{n-n_s}^{(s)}\right),$$
$$R_{22}^{(s+1)} = \left(\mathbf{r}_{k_s+1}^{(s+1)} \dots \mathbf{r}_{n-n_s}^{(s+1)}\right),$$

and set $\mathbf{r}_j^{(s+1)} = \mathbf{0}$, for $1 \le j \le k_s$. Moreover, let $T^{(s)} = \left(t_{i,j}^{(s)}\right)$, with $1 \le i \le j \le k_s$, and $\overline{T}^{(s)} = \left(t_{i,j}^{(s)}\right)$ with $1 \le i \le k, 1 \le j \le n - n_s$. First, notice that by eq. (3.3) we have

$$\left\| \begin{array}{cc} t_{k_s,k_s}^{(s)} & t_{k_s,k_s+1}^{(s)}, \dots, t_{k_s,n-n_s}^{(s)} \\ \mathbf{0} & R_{22}^{(s+1)} \end{array} \right\| \ge \sigma_{n_{s+1}}(A).$$

From eq. (2.4), we have

$$\begin{aligned} &\left\| \begin{array}{cc} t_{k_s,k_s}^{(s)} & t_{k_s,k_s+1}^{(s)}, \dots, t_{k_s,n-n_s}^{(s)} \\ \mathbf{0} & R_{22}^{(s+1)} \end{array} \right\|^2 \\ &\le (n - n_{s+1} + 1) \max \left\{ \left(t_{k_s,k_s}^{(s)}\right)^2, \max_{j \ge k+1} \left(\left\| \mathbf{r}_j^{(s)} \right\|^2 + \left(t_{k_s,j}^{(s)}\right)^2 \right) \right\}. \end{aligned}$$

Since $\left(t_{k_s,j}^{(s)}\right)^2 \le \left\| \mathbf{t}_j^{(s)} \right\|^2$, for all $1 \le j \le n - n_s$, and computing the maximum on a larger set of indices, we have

$$\begin{aligned} &\max \left\{ \left(t_{k_s,k_s}^{(s)}\right)^2, \max_{j \ge k+1} \left(\left\| \mathbf{r}_j^{(s)} \right\|^2 + \left(t_{k,j}^{(s)}\right)^2 \right) \right\} \\ &\le \max \left\{ \left\| \mathbf{t}_{k_s}^{(s)} \right\|^2, \max_{j \ge k+1} \left(\left\| \mathbf{r}_j^{(s)} \right\|^2 + \left\| \mathbf{t}_j^{(s)} \right\|^2 \right) \right\} \\ &\le \max_{j \ge 1} \left(\left\| \mathbf{r}_j^{(s)} \right\|^2 + \left\| \mathbf{t}_j^{(s)} \right\|^2 \right). \end{aligned}$$

From equations (3.12-3.13), for all $1 \le j \le n - n_s$, we have

$$\left\| \mathbf{c}_j^{(s)} \right\|^2 = \left\| \mathbf{r}_j^{(s)} \right\|^2 + \left\| \mathbf{t}_j^{(s)} \right\|^2,$$

and, finally, since $\left\| \mathbf{t}_1^{(s)} \right\|^2 = \left\| \mathbf{c}_1^{(s)} \right\|^2 = \max_j \left\| \mathbf{c}_j^{(s)} \right\|^2$ and by using Lemma 4, we get

$$\begin{aligned} &\left\| \begin{array}{cc} t_{k_s,k_s}^{(s)} & t_{k_s,k_s+1}^{(s)}, \dots, t_{k_s,n-n_s}^{(s)} \\ \mathbf{0} & R_{22}^{(s+1)} \end{array} \right\|^2 \\ &\le (n - n_{s+1} + 1) \left\| \mathbf{c}_1^{(s)} \right\|^2 \\ &\le \frac{n - n_{s+1} + 1}{\tau_u(\gamma + \tau_u - 1)} \sigma_{\min}^2 \left(C^{(s)} \right). \end{aligned}$$

We can conclude by noticing that $\sigma_{\min}\left(T^{(s)}\right) = \sigma_{\min}\left(C^{(s)}\right)$, since the two matrices differ by a left multiplication by an orthogonal matrix. □

By the interlacing property of singular values, we have

$$\overline{\sigma}^{(s+1)} \le \min \left\{ \overline{\sigma}^{(s)}, \sigma_{\min} \left(\begin{array}{c} B^{(s)} \\ T^{(s)} \end{array} \right) \right\},$$

thus the bounds on $\bar{\sigma}^{(s)}$ and $\sigma_{\min}(T^{(s)})$ are, by themselves, not a sufficient condition. Let us introduce the following result, which provides a bound of type (3.5) for QRDM.

Theorem 2. *Let $R_{11}^{(s)}$ be the upper triangular factor of order n_s computed by Algorithm 3. Suppose that $\bar{\sigma}^{(s)}$ is a good approximation of $\sigma_{n_s}(A)$. Then e have*

$$\bar{\sigma}^{(s+1)} \geq \sigma_{n_{s+1}}(A)\frac{\bar{\sigma}^{(s)}}{\sigma_1(A)}\frac{1}{\sqrt{2(n-n_{s+1})n_{s+1}}}\frac{\sqrt{\tau_u(\gamma+\tau_u-1)}}{k_s^2 n_s}.$$

Proof. Let us drop the subscript and the superscript on $B^{(s)}$, $T^{(s)}$, $R_{11}^{(s)}$ and its inverse $\left(R_{11}^{(s)}\right)^{-1}$, which will be denoted as B, T, R and R^{-1} respectively. Then, the inverse of matrix $R_{11}^{(s+1)}$ is given by

$$\left(R_{11}^{(s+1)}\right)^{-1} = \begin{pmatrix} R^{-1} & -R^{-1}BT^{-1} \\ & T^{-1} \end{pmatrix}.$$

Let us introduce the following partitions into rows

$$F = R^{-1}BT^{-1} = \begin{pmatrix} \mathbf{f}_1^T \\ \vdots \\ \mathbf{f}_{n_s}^T \end{pmatrix}, \quad R^{-1} = \begin{pmatrix} \mathbf{g}_1^T \\ \vdots \\ \mathbf{g}_{n_s}^T \end{pmatrix}, \quad T^{-1} = \begin{pmatrix} \mathbf{h}_1^T \\ \vdots \\ \mathbf{h}_{k_s}^T \end{pmatrix}.$$

The idea is to use eq. (2.6), that is

$$\bar{\sigma}^{(s+1)} \leq \min_h \left\| \mathbf{e}_h^T \begin{pmatrix} R^{-1} & F \\ & T^{-1} \end{pmatrix} \right\|^{-1} \leq \sqrt{n_{s+1}}\sigma_{\min}\bar{\sigma}^{(s+1)},$$

to estimate the minimum singular value up to a factor $\sqrt{n_{s+1}}$. For $1 \leq h \leq n_{s+1}$ we have

$$\left\| \mathbf{e}_h^T \begin{pmatrix} R^{-1} & F \\ & T^{-1} \end{pmatrix} \right\|^2 = \begin{cases} \|\mathbf{g}_h\|^2 + \|\mathbf{f}_h\|^2, & h \leq n_s, \\ \|\mathbf{h}_{h-n_s}\|^2, & h > n_s. \end{cases}$$

We can bound $\|\mathbf{g}_h\|$ using eq. (2.6) again, which gives

$$\bar{\sigma}^{(s)} \leq \min_h \left(\|\mathbf{g}_h\|^{-1} \right) \leq \sqrt{n_s}\bar{\sigma}^{(s)}.$$

In particular, for every $1 \leq h \leq n_s$, we get

$$\bar{\sigma}^{(s)} \leq \min_h \left(\|\mathbf{g}_h\|^{-1} \right) \leq \|\mathbf{g}_h\|^{-1},$$

and thus we have

$$\|\mathbf{g}_h\| \leq= \frac{1}{\bar{\sigma}^{(s)}} = \frac{1}{\sigma_{\min}(R)} = \sigma_{\max}(R^{-1}) = \|R^{-1}\|.$$

Similarly, we can bound $\|\mathbf{h}_{h-n_s}\|$ by $\|T^{-1}\|$. Let us now concentrate on bounding $\|\mathbf{f}_h\|$. We have

$$\begin{aligned}
\|\mathbf{f}_h\|_2 \leq \|\mathbf{f}_h\|_1 = \sum_{l=1}^{k_s} |f_{hl}| &= \sum_{l=1}^{k_s} \left| \sum_{i=1}^{k_s} (R^{-1}B)_{hi} {T^{-1}}_{il} \right| \\
&= \sum_{l=1}^{k_s} \left| \sum_{i=1}^{k_s} \sum_{j=1}^{n_s} {R^{-1}}_{hj} B_{ji} {T^{-1}}_{il} \right| \\
&\leq \sum_{l=1}^{k_s} \sum_{i=1}^{k_s} \sum_{j=1}^{n_s} \left|{R^{-1}}_{hj}\right| \, |B_{ji}| \, \left|{T^{-1}}_{il}\right| \\
&\leq \sum_{l=1}^{k_s} \sum_{i=1}^{k_s} \sum_{j=1}^{n_s} \left\|R^{-1}\right\|_{\max} \|B\|_{\max} \left\|T^{-1}\right\|_{\max} \\
&= k_s^2 n_s \left\|R^{-1}\right\|_{\max} \|B\|_{\max} \left\|T^{-1}\right\|_{\max} \\
&\leq k_s^2 n_s \left\|R^{-1}\right\| \|B\| \left\|T^{-1}\right\| \\
&= \frac{k_s^2 n_s}{\bar{\sigma}^{(s)}} \|B\| \left\|T^{-1}\right\|,
\end{aligned}$$

where we use the following facts $\|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_1$, and $\|A\|_{\max} \leq \|A\|$, see (2.5). Moreover, we can write

$$\begin{aligned}
&\|\mathbf{g}_h\|^2 + \|\mathbf{f}_h\|^2 \leq \frac{1}{(\bar{\sigma}^{(s)})^2} + \frac{k_s^4 n_s^2}{(\bar{\sigma}^{(s)})^2} \|B\|^2 \left\|T^{-1}\right\|^2 \\
&= \frac{\sigma_{\min}^2(T) + k_s^4 n_s^2 \|B\|^2}{(\bar{\sigma}^{(s)} \sigma_{\min}(T))^2} \leq \frac{\|T\|^2 + k_s^4 n_s^2 \|B\|^2}{(\bar{\sigma}^{(s)} \sigma_{\min}(T))^2} \\
&\leq \frac{2 k_s^4 n_s^2}{(\bar{\sigma}^{(s)} \sigma_{\min}(T))^2} \max\left\{\|T\|^2, \|B\|^2\right\} \leq \frac{2 k_s^4 n_s^2}{(\bar{\sigma}^{(s)} \sigma_{\min}(T))^2} \|A\|^2,
\end{aligned}$$

where, in the last inequality, we used the interlacing property and the invariance under matrix transposition of the singular values. In fact

$$\sigma_1(A) \geq \sigma_1 \begin{pmatrix} B \\ T \end{pmatrix} = \sigma_1 \left(B^T \; T^T\right) \geq \max\left\{\sigma_1(B), \sigma_1(T)\right\}.$$

Hence, we get

$$\frac{1}{\sqrt{\|\mathbf{g}_h\|^2 + \|\mathbf{f}_h\|^2}} \geq \frac{\bar{\sigma}^{(s)} \sigma_{\min}(T)}{\sqrt{2} k_s^2 n_s \sigma_1(A)}.$$

If $\bar{\sigma}^{(s)}$ is a good approximation of $\sigma_{n_s}(A)$, we can suppose that $\bar{\sigma}^{(s)}/\sigma_{n_s}(A) \approx 1$, and we can write

$$\begin{aligned}\sqrt{n_{s+1}}\bar{\sigma}^{(s+1)} &\geq \min\left\{\min_h \|\mathbf{h}_h\|^{-1}, \min_h \frac{1}{\sqrt{\|\mathbf{g}_h\|^2+\|\mathbf{f}_h\|^2}}\right\}\\ &\geq \min\left\{1, \frac{\bar{\sigma}^{(s)}}{\sqrt{2}k_s^2 n_s \sigma_1(A)}\right\}\sigma_{\min}(T)\\ &= \frac{\bar{\sigma}^{(s)}}{\sqrt{2}k_s^2 n_s \sigma_1(A)}\sigma_{\min}(T).\end{aligned}$$

Finally, using Lemma 5, we get

$$\bar{\sigma}^{(s+1)} \geq \sigma_{n_{s+1}}(A)\frac{\bar{\sigma}^{(s)}}{\sigma_1(A)}\frac{1}{\sqrt{2(n-n_{s+1})n_{s+1}}}\frac{\sqrt{\tau_u(\gamma+\tau_u-1)}}{k_s^2 n_s},$$

which is the desired bound. $\square$

The result above shows that even if the leading n_s columns have been carefully selected, so that $\bar{\sigma}^{(s)}$ is an accurate approximation of $\sigma_{n_s}(A)$, there could be a potentially dramatic loss of accuracy in the estimation of the successive block of singular values, namely $\sigma_{n_s+1}(A), \ldots, \sigma_{n_{s+1}}(A)$, just like for the standard column pivoting. In fact, it is well known that failure of QRP algorithm may occur (one such example is the Kahan matrix [96]), as well as for other greedy algorithms, but it is very unlikely in practice.

3.1.4 Termination criteria

In principle, both QRP and QRDM reveal the rank of a matrix. In finite arithmetic we have

$$\begin{pmatrix} \hat{R}_{11}^{(s)} & \hat{R}_{12}^{(s)} \\ & \hat{R}_{22}^{(s)} \end{pmatrix}, \tag{3.16}$$

where $\hat{R}_{ij}^{(s)}$ is the block $R_{ij}^{(s)}$ computed in finite representation, for $i = 1, 2, j = 2$. If the block $\hat{R}_{22}^{(s)}$ is small in norm, then it is reasonable to say that the matrix A has rank n_s, where n_s is the order of the upper triangular block $\hat{R}_{11}^{(s)}$. Golub and Van Loan [81] propose the following termination criterion

$$\left\|\hat{R}_{22}^{(s)}\right\| \leq \bar{\epsilon}\,\|A\|, \tag{3.17}$$

where $\bar{\epsilon}$ is a parameter depending on the machine precision ϵ. Notice that even if a block R_{22} with small norm implies numerical rank-deficiency, the converse is not true in general: an example is the Kahan matrix [96]. Since the 2-norm is not directly available, we make use of the inequalities (2.4). Let us write the column partition $\hat{R}_{22}^{(s)} = \begin{pmatrix}\hat{\mathbf{c}}_1^{(s)} & \ldots & \hat{\mathbf{c}}_{n-n_s}^{(s)}\end{pmatrix}$. We have

$$\left\|\hat{R}_{22}^{(s)}\right\| \leq \sqrt{n-n_s}\max_i\left\|\hat{\mathbf{c}}_i^{(s)}\right\|, \quad \max_i\|\mathbf{a}_i\| \leq \|A\|.$$

Therefore, the stopping criterion (3.17) holds if

$$\sqrt{n-n_s}\max_i \left\|\hat{\mathbf{c}}_i^{(s)}\right\| \leq \bar{\epsilon}\max_i \|\mathbf{a}_i\|\,. \tag{3.18}$$

Notice that the contrary does not hold. In Section 3.2 we test this practical stopping criterion (3.18) and discuss the following choice

$$\bar{\epsilon} = \epsilon\; n, \tag{3.19}$$

3.1.5 Implementation of QR with deviation maximization algorithm

In this section we discuss implementation aspects of the QRDM procedure. In particular, we address the problem of choosing less restrictive values of the parameters τ_u and τ_θ than the ones imposed by equation (2.12), without affecting the robustness of the computed QR and the structure of the pivoting, which has a significant impact on the cost of the algorithm.

We now describe the most crucial aspects of a real implementation of Algorithm 3. First, the deviation maximization block pivoting cannot be carried out when the maximum partial column norm of the trailing matrix is of the order of the working precision ϵ, that is when (2.17) holds. In such case, a practical implementation switches from the deviation maximization block pivoting to another one, e.g. the standard column pivoting.

Let us now detail how to choose τ_u and τ_θ. In practice, as we detail in Section 3.2, it is desirable to relax the requirements given by Lemma 5 on the choice of the values for τ_u and τ_θ, since their theoretical bounds turn out to be very demanding with a consequent limitation of the performance of the overall factorization. On the other side, if we settle for any choice of τ_u, τ_θ with $1 \geq \tau_u, \tau_\theta \geq 0$, then the deviation maximization may identify a set of numerically linear dependent columns. In order to overcome this issue, we incorporate an additional check in the Householder procedure in the loop at step 5 of Algorithm 3, which is modified and replaced by the following loop.

```
1: for $l = 1, \dots, k_s$ do
2:     compute the Householder reflector $\mathbf{v}^{(n_s+l)}$ w.r.t. $R_{22}^{(s)}(l:(m-n_s), l)$
3:     update $R^{(s+1)}(n_s+l:m, n_s+l:n_s+k_s) = \left(I - \mathbf{v}^{(n_s+l)}(\mathbf{v}^{(n_s+l)})^T\right)$
           $R_{22}^{(s)}(l:(m-n_s), l:k_s)$
4:     if $l+1 < k_s$ and $\left\|R_{22}^{(s)}(l+1:(m-n_s), l+1)\right\| < \varepsilon_s$ then
5:         break for
6:     end if
7: end for
```

Recall that the columns chosen by the deviation maximization at the s-th algorithmic step satisfy

$$\left\|R_{22}^{(s)}(:,j)\right\| \geq \tau_u \max_{1\leq i\leq(n-n_s)} \left\|R_{22}^{(s)}(:,i)\right\|, \tag{3.20}$$

for all $j \in \{j_1, \dots, j_{k_s}\}$. The prior check introduced at step 5 breaks the Householder procedure when a partial column norm $\|R^{(s)}(n_s + l : m, n_s + l)\|$ becomes smaller than ε_s defined above, in other words, if the l-th column is not sufficiently linearly independent from the subspace spanned by the first $l-1$ columns already processed. Different choices of ε_s are possible, e.g. a small and constant threshold. Here, we propose to fix the value of ε_s equal to the right-hand side of (3.20). Numerical tests show that this choice works well in practice.

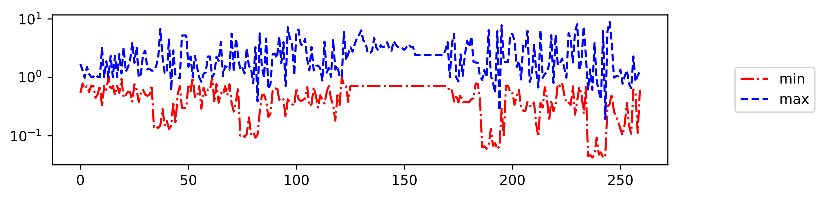
The modified loop may cause the Householder triangularization to terminate with $l < k_s$ computed reflectors, and the algorithm continues with by setting $k_s = l$. At the next iteration, the pivoting strategy moves the rejected column from the leading position, if necessary. As we show in Section 3.2, the break mechanism just presented allows to choose values for τ_u and τ_θ in a rather simple way and to obtain competitive results in execution times.

3.2 Numerical tests

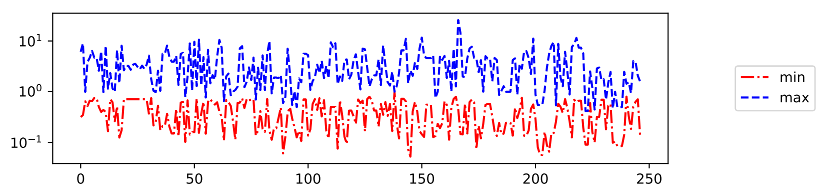
In this section we discuss the numerical accuracy of QRDM against the SVD decomposition and the the block QRP algorithm, briefly called QP3 [119]. We report experimental results comparing the double precision codes `xgeqp3` and `dgeqp3` from LAPACK, and `dgeqrdm`, a double precision C implementation of our block algorithm QRDM available online at the URL: `https://github.com/mdessole/qrdm`. All tests are carried out on a platform with an Intel(R) Core(TM) i7-2700K processor and a 8 GB system memory, employing CBLAS and LAPACKE, the C reference interfaces to BLAS and LAPACK implementations on Netlib, respectively.
Particular importance is given to the values on the diagonal of the upper triangular factor R of the RRQR factorization, which are compared with the singular values of the R_{11} block and with the singular values of the input matrix A. The tests are carried out on a subset of matrices from the San Jose State University Singular Matrix Database, which were used in other previous works on the topic, see e.g. [55, 83]. We show results coming from two subsets of this collection, that we call:

- "small matrices": it consists of the 261 matrices with $m \leq 1024$, $32 < n \leq 2048$, sorted in ascending order with respect to the number of columns n;
- "big matrices": it consists of the first 247 matrices with $m > 1024$, $n > 2048$, sorted in ascending order with respect to the number of columns n.

For each matrix A, we denote by σ_i the i-th singular value of A computed with LAPACK's `xgejsv` routine, and by n_r the numerical rank computed with the option `JOBA='A'`: in this case, small singular values are comparable with round-off noise and the matrix is treated as numerically rank deficient. As the pivoting used in QRDM does not guarantee that the diagonal values of the factor R are monotonically non-increasing in modulus, for each matrix we denote by d_i the i-th largest value among the first n_r diagonal entries considered with positive sign.



(a)

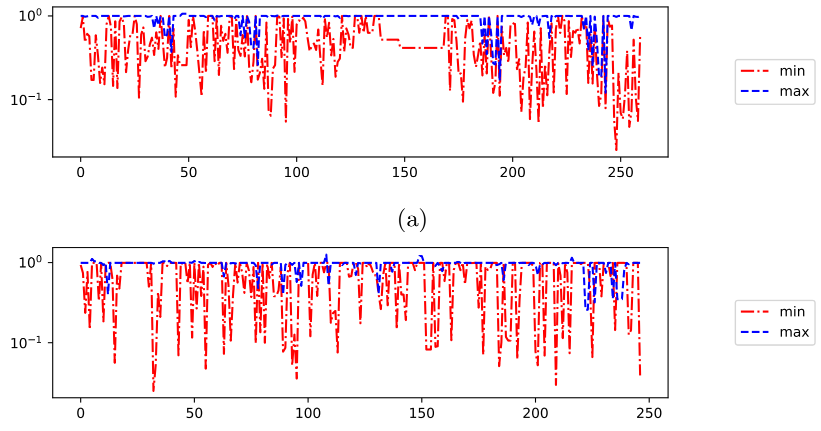


(b)

Figure 3.1: Ratio d_i/σ_i with QP3.

The results provided by QP3 for the two collections are summarised in Figure 3.1 and Figure 3.2. Figure 3.1 shows the values $\min_{i \le n_r} d_i/\sigma_i$ (red) and $\max_{i \le n_r} d_i/\sigma_i$ (blue) for each matrix in the set "small matrices" (Figure 3.1a) and "big matrices" (Figure 3.1b). The order of magnitude of the ratios d_i/σ_i ranges from 10^{-1} to 10^1, i.e. the positive diagonal value d_i approximates the corresponding singular value σ_i up to a factor 10, for $i = 1, \le, n_r$. Figure 3.2 shows the minimum (red) and maximum (blue) values of the ratio $\sigma_i(R_{11})/\sigma_i$ for each matrix in the set "small matrices" (Figure 3.2a) and "big matrices" (Figure 3.2b), where $\sigma_i(R_{11})$ is the i-th singular value of $R_{11} = R(1 : n_r, 1 : n_r)$ computed with LAPACK's `xgejsv`, and σ_i is i-th singular value of A. These results confirm that QP3 provides an approximation of the singular values up to a factor 10.

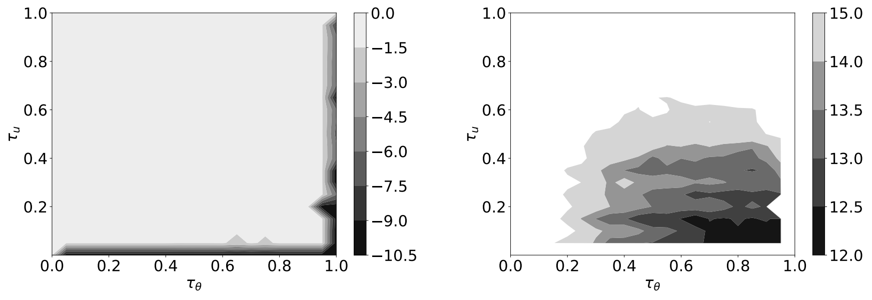
Before providing similar results for QRDM, let us discuss the sensitivity of parameters τ_u and τ_θ to the rank-revealing property (3.5). To this aim, we set a grid $\mathcal{G}$ of values $\mathcal{G}(i, j) = (\tau_\theta^i, \tau_u^j) = (0.05\, i, 0.05\, j)$, with $i, j = 0, \ldots, 20$, and we consider the R factor obtained by QRDM for each matrix in the "small matrices" collection and for each choice of pair $(\tau_\theta^i, \tau_u^j)$, $i, j = 0, \ldots, 20$. Figure 3.3a shows the order of magnitude of the minimum value over the "small matrices" dataset of the minimum ratio $\min_{i \le n_r} d_i/\sigma_i$, for each value within the point grid $\mathcal{G}$. We see that the positive diagonal elements provide an approximation up to a factor 10 of the singular values for a wide range of parameters, corresponding to the light gray region: in practice, it is sufficient to avoid the extreme cases $\tau_u = 0$ and $\tau_\theta = 1$. Indeed, the presence of an inaccurate diagonal element in R, i.e. some d_i that strongly underestimates the corresponding singular value σ_i, is avoided thanks to the additional check proposed in (3.20), which possibly breaks the Householder triangularization. This suggests that any choice of $1 \ge \tau_u > 0$ and $1 > \tau_\theta \ge 0$ may lead to a rank-revealing QR decomposition. In this way,



(a)

(b)

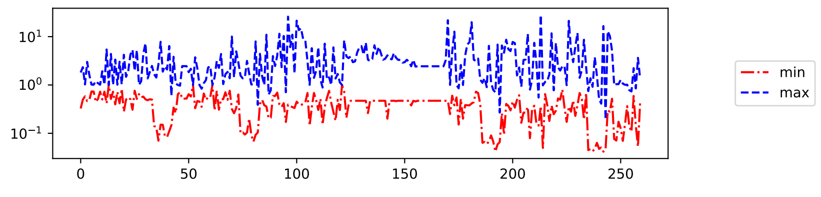
Figure 3.2: Ratio $\sigma_i(R_{11})/\sigma_i$ with QP3.



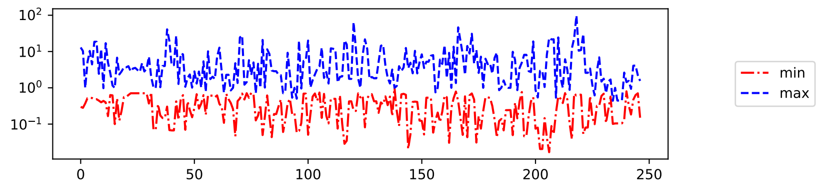
(a)

(b)

Figure 3.3: Performance of QRDM in function of parameters τ_u and τ_θ.



(a)



(b)

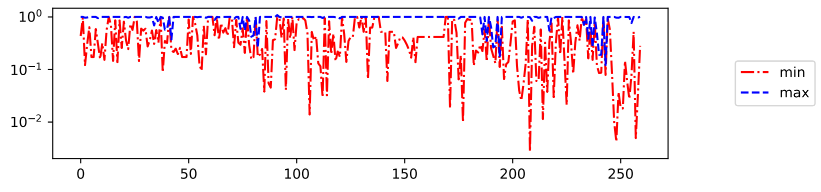
Figure 3.4: Ratio d_i/σ_i with QRDM.

even a greedy setting of the algorithm is viable: if τ_u is almost zero and τ_θ near one, the deviation maximization pivoting collects as much columns as possible.

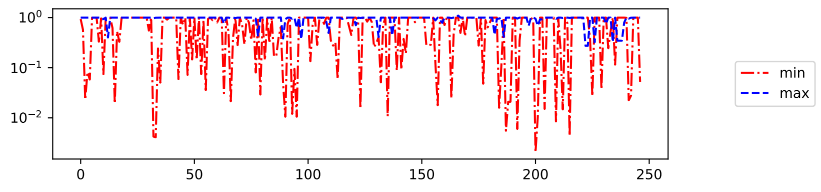
Figure 3.3b shows the cumulative execution times for all tests in the "small matrices" collection, for each grid point of $\mathcal{G}$. It is evident that the best performance is obtained toward the right-bottom corner, in correspondence of the dark gray region, confirming that a greedy approach is convenient. However, a too greedy choice of the parameters may yield less accurate ratios d_i/σ_i in a very few cases. Hence, we suggest a safer choice: from now on we set and $(\tau_\theta, \tau_u) \coloneqq (0.9, 0.15)$, which are the optimal values for the validation set here considered.

Figure 3.4 and Figure 3.5 summarize the results provided by QRDM for the two collections with this parameters' choice. Figure 3.4 shows the minimum (red) and maximum (blue) values of the ratios d_i/σ_i for each matrix in the set "small matrices" (Figure 3.4a) and "big matrices" (Figure 3.4b). The order of magnitude of the ratios d_i/σ_i ranges from 10^{-1} to 10^1, i.e. the positive diagonal value d_i approximate the corresponding singular value σ_i up to a factor 10, for $i = 1, \leq, n_r$, giving comparable results with those of QP3. Figure 3.5 shows the minimum (red) and maximum (blue) values of the ratio $\sigma_i(R_{11})/\sigma_i$ for each matrix in the set "small matrices" (Figure 3.5a) and "big matrices" (Figure 3.5b), where $\sigma_i(R_{11})$ is the i-th singular value of $R_{11} = R(1 : n_r, 1 : n_r)$ computed with LAPACK's `xgejsv`, and σ_i is i-th singular value of A. Here QRDM provides an approximation of the singular values up to a factor 10^2.

Let us now consider QRDM with a stopping criterion. We show the accuracy in the determination of the numerical rank, and the benefits in terms of execution times, when the matrix rank is much smaller than its number of columns. Recall that QRDM switches to the scalar pivoting when the partial column norms are not sufficiently large (2.17), affecting the algorithm's perfor-



(a)



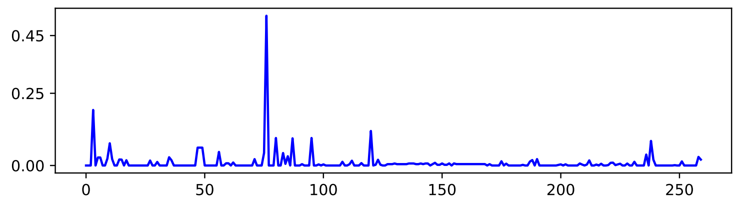
(b)

Figure 3.5: Ratio $\sigma_i(R_{11})/\sigma_i$ with QRDM.

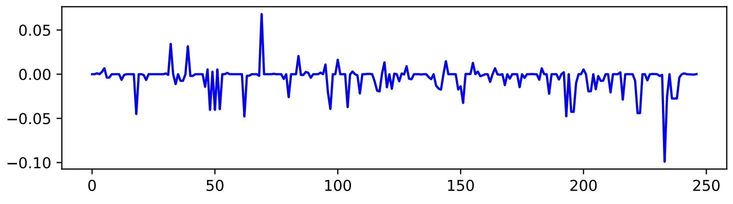
mance. We consider the stopping criterion in (3.18)–(3.19): the numerical rank is this case is given by the number of total number of columns processed by QRDM before exiting and we denote it by n_r^{QRDM}.

Figure 3.6 shows the ratio $\left(n_r^{QRDM} - n_r\right)/n_r$, where n_r is the number of singular values larger than $\epsilon n\|A\| = \epsilon n\sigma_1$, for all matrices in the "small matrices" (Figure 3.6a) and "big matrices" (Figure 3.6b) collections. The computed rank is accurate in nearly all cases. Figure 3.6a shows a case in which n_r^{QRDM} overestimates n_r with a relative error of about 45%. This is a pathological case: the matrix involved shows a gap in the singular values distribution and, immediately after, a group of singular values just below the value $\epsilon n\|A\|$. The corresponding diagonal entries of the matrix R obtained by QRDM show the same gap, but the stopping criterion (3.18)–(3.19), which approaches the quantity $\epsilon n\|A\|$ from below, does not detect so accurately the crossing of the threshold. Anyway, since the diagonal entries of R describe well the corresponding gap in the singular values, even in this case, the applications can correctly truncate the R factor in post-processing and form the corresponding Q factor.

Finally, we compare the performance of QP3 with that of QRDM in terms of their execution times for instances in the set "big matrices". Figure 3.7 shows the speedup of QRDM (in red) and QRDM with stopping criterion (in blue) over QP3, namely the ratio t_{QP3}/t_{QRDM} in function of t_{QP3}, where t_{QP3} and t_{QRDM} are the execution times (in seconds) of QP3 and QRDM, respectively. We see that QRDM achieves an average speedup of 4× for medium/large size matrices (corresponding to higher execution times). This result is a consequence of a higher BLAS-3 fraction of work provided by QRDM against QP3. Indeed, in order to compute the block Householder reflector, QP3 must update the partial column norms and identifies the next pivot by computing the maximum column norm, while QRDM selects a block of pivot columns at once. The



(a)



(b)

Figure 3.6: Relative error on the computed numerical rank for QRDM.

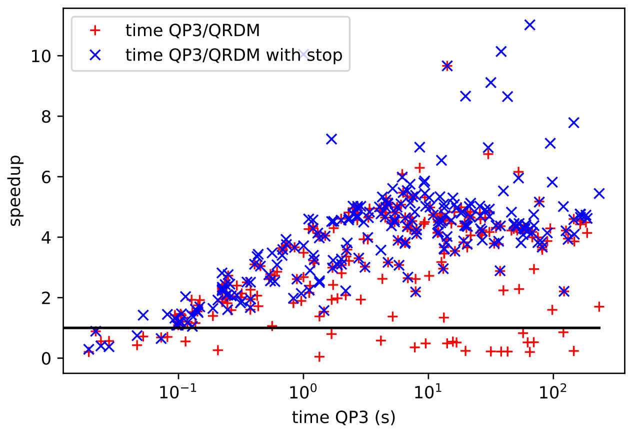


Figure 3.7: Speedup analysis of QRDM and QRDM with stopping criterion over QP3.

former strategy relies on BLAS-2 operations, while the latter mostly on BLAS-3 operations. Moreover, the stopping criterion gives an additional advantage for matrices whose numerical rank is much smaller than their number of columns.

Last, let us discuss briefly the effect of the block size $k_{\max}$ introduced to limit the cardinality of the candidate set in equation (2.16). This parameter depends on the specific architecture, mainly in terms of cache-memory size, and typical values are $k_{\max} = 32, 64, 128$. We observed that there is an optimal value of $k_{\max}$, in sense that it gives the best execution times for a fixed experimental setting, and its computation is similar to the well-known BLAS block size computation practice. For sake of clarity we say that on our personal computer we observed the optimal value $k_{\max} = 64$, but other choices gave similar performances, e.g. $k_{\max} = 32$.

3.3 Concluding remarks

In this chapter we introduced the rank-revealing QR factorization with deviation maximization pivoting, briefly called QRDM, and we compared it with the rank-revealing QR factorization with the state-of-the-art algorithm, namely standard column pivoting, briefly QRP. We have provided a theoretical worst case bound on the smallest singular value for QRDM and we have shown it is similar to available results for QRP. Extensive numerical experiments confirmed that QRDM reveals the rank similarly to QRP and provides a good approximation of the singular values obtained with LAPACK's `xgejsv` routine. Moreover, we have shown that QRDM has shorter execution times than those of the BLAS-3 version of QRP implemented in LAPACK's `xgeqp3` routine in a large number of test cases. Comparison with other RRQR algorithms is avoided, since approaches presented in Chan [42], Foster [75] can be performed as additional refinements, and distributed memory approaches like [55] can be used no matter the pivoting technique. The software implementation of QRDM here used is freely available at the URL: `https://github.com/mdessole/qrdm`.

Chapter 4

Nonnegative least squares

NonNegative Least Squares (NNLS) problems arise in many applications where data points can be represented as nonnegative linear combinations of some meaningful components. Such problems are frequently encountered in signal and image processing and they are core problems in more complex computations, such as nonnegative matrix and tensor decompositions. Moreover, when dealing with underdetermined systems of equations, the nonnegativity contraint is known to naturally enhance sparsity of the solution, that is the solution attained has few nonzeros, see e.g. [31, 76, 143, 144]. An important outcome of this body of work is that nonnegativity alone may attain a satisfactory sparse recovery. Over the last two decades, sparsity has become one of the most relevant topics in signal processing. In general, sparsity in signals describes the phenomenon where high dimensional data can be expressed with few measurements and it results in finding a sparse solution to undertermined systems of equations. The problem of finding the sparsest solution to an underdetermined linear system can be formulated as a constrained ℓ_0-minimization problem, but often its solution is not practical as it is highly nonconvex and it is NP-hard in general. However, it is well known that ℓ_1-minimization gives the same solution for a restricted class of matrices. This fact is known as $\ell_0 - \ell_1$ equivalence and it has been found empirically [46] and theoretically [68, 72]. The ℓ_1-minimization problem can be seen as a convex relaxation of the ℓ_0-minimization problem. Notice that NNLS is a convex problem too.

The first algorithm devised for NNLS is the Lawson-Hanson algorithm [100]. Since this seminal work, many modifications have been proposed in order to improve the standard Lawson-Hanson algorithm: Bro and Jong [29] have proposed a variation specifically designed for use in nonnegative tensor decompositions; their algorithm, called "fast NNLS" (FNNLS), reduces the execution time by avoiding redundant computations in nonnegative matrix factorization problems arising in tensor decompositions and performs well with multiple right-hand sides, which is not the case here discussed, thus we omit a comparison. Van Benthem and Keenan [140] presented a different NNLS solution algorithm, namely "fast combinatorial NNLS" (FCNNLS), also designed for the specific case of a large number of right-hand sides. The authors exploited a clever reorganization of computations in order to take advantage of the combinatorial nature of the problems treated (multivariate curve resolution) and introduced a nontrivial initialization of the algorithm by means of unconstrained least squares solution. The principal block pivoting method introduced by Portugal et al. [115] is an active set type algorithm which differs from the standard Lawson-Hanson algorithm, since the sequence of iterates produced does not necessarily fall into the feasible region. The convergence is ensured provided the objective function is strictly convex, while when we deal with underdetermined matrices it is simply convex. Therefore this algorithm fails in sparse recovering. Surprisingly, the Lawson-Hanson algorithm [100] does not suffer from this drawback.

The Lawson-Hanson algorithm is an active set method which incrementally builds an optimal solution by solving at each iteration an unconstrained least squares subproblem, e.g. by computing a QR decomposition, on a subset of columns of the initial matrix in such a way that the objective value decreases while the iterates are kept within the feasible region. The algorithm performs column pivoting by selecting one column at a time and therefore it relies only on BLAS-2 operations. Actually, the columns are not permuted, rather a vector of indices is used to represent the permutation. In general, column pivoting makes it more difficult to achieve high performances in QR computation, see [17, 18, 19, 119, 20].

In this chapter we apply the deviation maximization as column selection strategy to the Lawson-Hanson algorithm for the solution of NNLS, devising a new algorithm we call Lawson-Hanson with Deviation Maximization (LHDM). This algorithm allows to exploit BLAS-3 operations, leading to higher performances. In [61], a preliminary version of this technique has led to a significant performance gain in the execution of the Lawson-Hanson algorithm for underdetermined systems, there applied to the particular case of computing nearly-optimal designs on high-dimensional spaces by means of the Tchakaloff theorem. Here, we extend the theoretical discussion about the features of LHDM and we explore the sparsity recovery ability of LHDM, comparing its performance against several ℓ_1-minimization solvers.

The rest of this chapter is organized as follows. In Section 4.1, we formally introduce the NNLS problem and the Lawson-Hanson algorithm and we combine the Lawson-Hanson method with deviation maximization. We provide a theoretical result about finite convergence of LHDM by generalizing the analogous result for the standard Lawson-Hanson. In Section 4.2, we introduce the problem of sparse recovery. We recall results that ensure the so-called $\ell_0 - \ell_1$ equivalence and conditions under which the underdetermined NNLS problem has a unique solution. Last, we recall how arbitrary signed sparse recovery can be attained by NNLS solvers. In Section 4.3, we present a comparison of LHDM against several ℓ_1-minimization solvers in terms of performance and solutions found. The results are shown with an extensive set of experiments. Finally, Section 4.4 concludes the chapter.

4.1 Solving nonnegative least squares problems

Consider the nonnegative least squares problem

$$\min \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2, \qquad \text{s.t. } \mathbf{x} \geq 0, \tag{4.1}$$

where $A = (\mathbf{a}_1 \dots \mathbf{a}_n)$ is an $m \times n$ matrix, $\mathbf{b}$ is a vector of length m. It is well known that

$$\min \|A\mathbf{x} - \mathbf{b}\|_2 = \min \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2.$$

Let us define

$$\phi(\mathbf{x}) := \|A\mathbf{x} - \mathbf{b}\|^2, \tag{4.2}$$

where from now on $\|\cdot\|$ denotes the 2-norm, if not stated otherwise. We have

$$\phi(\mathbf{x}) = (A\mathbf{x} - \mathbf{b})^T(A\mathbf{x} - \mathbf{b}) = \mathbf{x}^T A^T A\mathbf{x} - 2\mathbf{b}^T A\mathbf{x} + \mathbf{b}^T\mathbf{b},$$

and

$$\min \phi(\mathbf{x}) = \min \mathbf{x}^T A^T A\mathbf{x} - 2\mathbf{b}^T A\mathbf{x}.$$

Note that the gradient for ϕ is

$$\nabla\phi(\mathbf{x}) = 2A^T A\mathbf{x} - 2A^T\mathbf{b} = -2A^T(\mathbf{b} - A\mathbf{x}) = -2A^T\mathbf{r}(\mathbf{x}),$$

where $\mathbf{r} \in \mathbb{R}^m$ is the residual function $\mathbf{r}(\mathbf{x}) = \mathbf{b} - A\mathbf{x}$. A descend direction for ϕ at $\mathbf{x}$ is any vector $\mathbf{s} \in \mathbb{R}^n$ such that

$$\mathbf{s}^T\nabla\phi(\mathbf{x}) = -2(A\mathbf{s})^T\mathbf{r}(\mathbf{x}) < 0.$$

The set of points satisfying the constraints is called feasible region, and in this case is given by the set $\Omega = \{\mathbf{x} \in \mathbb{R}^n : \ \mathbf{x} \geq 0\}$. A point $\mathbf{x}$ is said to be feasible if it belongs to the feasible region.

Definition 2. *Let $\mathbf{x}$ be a feasible point, i.e. $\mathbf{x} \geq 0$. A vector $\mathbf{s} \in \mathbb{R}^n$ is a **feasible direction** at $\mathbf{x}$ if there exists $\varepsilon > 0$ such that*

$$\mathbf{x} + \varepsilon\mathbf{s} \geq 0.$$

Notice that if $\mathbf{s}$ is a feasible direction at $\mathbf{x}$ and $s_i < 0$, then we must have $x_i > 0$. The following theorem characterizes the solution of problem (4.1). For a proof see e.g. [100].

Theorem 3. *(Karush-Kuhn-Tucker conditions for NNLS) A point $\mathbf{x}^\star \in \mathbb{R}^n$ is a solution of problem (4.1) if and only if there exists $\mathbf{w}^\star \in \mathbb{R}^n$ and a partition $Z^\star \cup P^\star = \{1, \dots, n\}$ such that*

$$\mathbf{w}^\star = A^T(\mathbf{b} - A\mathbf{x}^\star), \tag{4.3}$$

$$x_i^\star = 0, \ i \in Z^\star, \quad x_i^\star > 0, \ i \in P^\star, \tag{4.4}$$

$$w_i^\star \leq 0, \ i \in Z^\star, \quad w_i^\star = 0, \ i \in P^\star. \tag{4.5}$$

Further discussion of this theorem, including its proof, can be found in the literature of constrained optimization, see e.g. [74]. Equations (4.4)–(4.5) imply

$$x_i^\star w_i^\star = 0, \qquad i = 1, \ldots, n, \tag{4.6}$$

which is also known as complementary condition.

Let us consider the i-th constraint, $\mathbf{e}_i^T \mathbf{x} \geq 0$, where $\mathbf{e}_i$ is the i-th element of the canonical basis. The vector $\mathbf{e}_i$ is normal to the half-space defined by the constraint and it points towards the interior of the feasible region Ω. The point $\mathbf{x}^\star$ is in the interior of the half-spaces indexed in $P^\star$ and on the boundary of those indexed in $Z^\star$. The point $-\mathbf{w}^\star$ is the gradient of the objective evaluated at $\mathbf{x}^\star$. Since $w_i^\star = 0,\ i \in P^\star$, we have

$$\sum_{i \in Z^\star} (-w_i^\star)(-\mathbf{e}_i) = \mathbf{w}^\star, \tag{4.7}$$

which states that the negative gradient at $\mathbf{x}^\star$ is a nonnegative ($-w_i^\star \geq 0$) linear combination of the outward normals ($-\mathbf{e}_i$) to the constraint hyperplanes on which $\mathbf{x}^\star$ lies ($i \in Z^\star$), that is $\mathbf{w}^\star$ lies in the convex cone based at $\mathbf{x}^\star$ and generated by $-\mathbf{e}_i$, $i \in Z^\star$.

4.1.1 The Lawson-Hanson algorithm

The first algorithm to solve (4.1) is due to Lawson and Hanson [100], and it is presented in Algorithm 4 for completeness. It is an active set method and a particular case of the algorithm introduced in [129] for the least squares problem with linear inequality constraints. Recall that a constraint is said to be active if it holds with equality, here $x_i = 0$, it is said to be passive when it holds with strict inequality, here $x_i > 0$, otherwise it is violated, namely $x_i < 0$. Active set methods are in the family of descend algorithms, that is they look for the solution by decreasing the objective function value, in particular at each iteration the new solution is a feasible point in a feasible descend direction with respect to the current solution.

Let us now describe the main features of the Lawson-Hanson algorithm. Let P_s denote the passive set and Z_s denote the active set at s-th algorithmic step, with cardinality respectively n_s and $n - n_s$. Let us define the following submatrices

$$\begin{aligned} A_P^{(s)} &= \left(\mathbf{a}_{i_1} \ldots \mathbf{a}_{i_{n_s}}\right), && \{i_1, \ldots, i_{n_s}\} = P_s, \\ A_Z^{(s)} &= \left(\mathbf{a}_{j_1} \ldots \mathbf{a}_{j_{n-n_s}}\right), && \{j_1, \ldots, j_{n-n_s}\} = Z_s. \end{aligned}$$

We define the s-th iterate as $\mathbf{x}^{(s)} = \left(x_1^{(s)}, \ldots, x_n^{(s)}\right)^T \in \mathbb{R}^n$. With an analogous

Algorithm 4 Lawson-Hanson (LH)

```
    Inputs: $A, \mathbf{b}$
    Outputs: $P^\star, Z^\star, \mathbf{x}^\star$
 1: $P_0 = \emptyset$, $Z_0 = \{1, \dots, M\}$, $\mathbf{x}^{(0)} = 0$, $\mathbf{w}^{(0)} = A^T\mathbf{b}$, $s = 0$
 2: while $Z_s \neq \emptyset$ and $\max \mathbf{w}_Z^{(s)} > 0$ do
 3:     $j_s = \mathrm{argmax}_i\, w_i^{(s)}$
 4:     $P_{s+1} = P_s \cup \{j_s\}$
 5:     $Z_{s+1} = Z_s \setminus \{j_s\}$
 6:     $\mathbf{y}_P^{(s+1)} = \mathrm{argmin} \left\| A_P^{(s+1)}\mathbf{y} - \mathbf{b} \right\|$, $\mathbf{y}_Z^{(s+1)} = 0$
 7:     while $\min \mathbf{y}_P^{(s+1)} \leq 0$ do
 8:         $s = s + 1$
 9:         $Q = P_s \cap \left\{ i \;:\; y_i^{(s)} \leq 0 \right\}$
10:         $\alpha = \min_{i \in Q} x_i^{(s-1)} \left( x_i^{(s-1)} - y_i^{(s)} \right)^{-1}$
11:         $\mathbf{x}^{(s)} = \mathbf{x}^{(s-1)} + \alpha \left( \mathbf{y}^{(s)} - \mathbf{x}^{(s-1)} \right)$
12:         $P_{s+1} = P_s \setminus \left\{ i \;:\; i \in P_s,\ x_i^{(s)} \leq 0 \right\}$
13:         $Z_{s+1} = Z_s \cup \left\{ i \;:\; i \in P_s,\ x_i^{(s)} \leq 0 \right\}$
14:         $\mathbf{y}_P^{(s+1)} = \mathrm{argmin} \left\| A_P^{(s+1)}\mathbf{y} - \mathbf{b} \right\|$, $\mathbf{y}_Z^{(s+1)} = 0$
15:     end while
16:     $\mathbf{x}_P^{(s+1)} = \mathbf{y}_P^{(s+1)}$, $\mathbf{x}_Z^{(s+1)} = 0$
17:     $\mathbf{w}^{(s+1)} = A^T \left( \mathbf{b} - A\mathbf{x}^{(s+1)} \right)$
18:     $s = s + 1$
19: end while
20: $P^\star = P_s$, $Z^\star = Z_s$, $\mathbf{x}^\star = \mathbf{x}^{(s)}$
```

notation to the one introduced above for matrices, we have

$$\begin{aligned}
\mathbf{x}_P^{(s)} &= \left(x_{i_1}^{(s)}, \dots, x_{i_{n_s}}^{(s)} \right)^T \in \mathbb{R}^{n_s}, & \{i_1, \dots, i_{n_s}\} &= P_s, \\
\mathbf{x}_Z^{(s)} &= \left(x_{j_1}^{(s)}, \dots, x_{j_{n-n_s}}^{(s)} \right)^T \in \mathbb{R}^{n-n_s}, & \{j_1, \dots, j_{n-n_s}\} &= Z_s.
\end{aligned}$$

The elements of the sequence $\mathbf{x}^{(s)}$ produced by Lawson-Hanson algorithm do not leave the feasible region and we always have $\mathbf{x}_P^{(s)} > 0, \mathbf{x}_Z^{(s)} = 0$. At each outer loop iteration, the current solution $\mathbf{x}^{(s)}$ solves an unconstrained least squares subproblem

$$\mathbf{x}_P^{(s)} = \mathrm{argmin} \left\| A_P^{(s)}\mathbf{y} - \mathbf{b} \right\|, \qquad \mathbf{x}_Z^{(s)} = 0. \tag{4.8}$$

With the notation introduced in Algorithm 4, we have

$$\begin{aligned}
\mathbf{r}^{(s)} &= \mathbf{b} - A\mathbf{x}^{(s)} = \mathbf{b} - A_P^{(s)}\mathbf{x}_P^{(s)}, \\
\mathbf{w}^{(s)} &= A^T\left(\mathbf{b} - A\mathbf{x}^{(s)}\right) = A^T\left(\mathbf{b} - A_P^{(s)}\mathbf{x}_P^{(s)}\right) = A^T\mathbf{r}^{(s)}, \\
\mathbf{w}_Z^{(s)} &= (A_Z^{(s)})^T\left(\mathbf{b} - A_P^{(s)}\mathbf{x}_P^{(s)}\right) = \left(A_Z^{(s)}\right)^T\mathbf{r}^{(s)}, \\
\mathbf{w}_P^{(s)} &= (A_P^{(s)})^T\left(\mathbf{b} - A_P^{(s)}\mathbf{x}_P^{(s)}\right) = 0,
\end{aligned}$$

where the last identity is a consequence of normal equations for (4.8). Clearly we have $\left(\mathbf{w}^{(s)}\right)^T\mathbf{x}^{(s)} = 0$ for every s. As shown in [100], Algorithm 4 terminates in a finite number of steps in exact arithmetic and $\mathbf{x}^{(s)}, \mathbf{w}^{(s)}, P_s, Z_s$ satisfy KKT conditions (4.4)–(4.5) on termination.

4.1.2 Implementation of Lawson-Hanson algorithm

In [100], a Fortran implementation of the NNLS solver based on the Lawson-Hanson Algorithm 4 is provided. Steps 6 and 14 require the solution of an unconstrained least squares problem by, e.g., QR decomposition. If the columns indexed in the passive set are linearly independent, the QR decomposition of $A_P^{(s)}$ is essentially unique once the order of the columns in the passive set is fixed. Every time step 6 is reached, a new column has been inserted in the passive set, therefore QR updates by means of Householder transformation are used. When step 14 is reached, a column has been moved back to the active set: as a consequence, some columns of the new R factor have an extra element under the diagonal which can be zeroed out by means of Givens transformations. For more details, see e.g. [29, Chap. 3]. In practice, the full orthogonal factor of the QR decomposition of $A_P^{(s)}$ is never formed: at each step, an Householder transformation or a sequence of Givens transformations is applied to the whole matrix A and to the right-hand side $\mathbf{b}$. In fact, consider the QR decomposition

$$A_P^{(s)} = Q^{(s)}\begin{pmatrix} R^{(s)} \\ 0 \end{pmatrix}.$$

Then, the least square solution of problems in the form (4.8) at steps 6 and 14 is given by the following triangular linear system

$$R^{(s)}\mathbf{y}_P^{(s)} = \left(\left(Q^{(s)}\right)^T\mathbf{b}\right)_P^{(s)},$$

which can be solved by backsubstitution. The columns of A are not physically moved, thus no memory communication is needed. Instead, a pivoting vector is used in order to keep trace of the order in which the columns of $A_P^{(s)}$ have been processed to form the triangular factor $R^{(s)}$. As it is processed only one column at a time, these operations mainly require BLAS-2 level operations, typically turning out into inefficient algorithms.

4.1.3 A simple application of deviation maximization

The deviation maximization can be used to identify a subset of indices J_s to be moved from the active set Z_s to the passive set P_s at each outer iteration of the Lawson-Hanson algorithm, as it is shown in Algorithm 5.

Algorithm 5 Simple Lawson-Hanson with Deviation Maximization

```
    Inputs: $A, \mathbf{b}, \tau_w, \tau_u, \tau_\theta, k_{\max}$
    Outputs: $P^\star, Z^\star, \mathbf{x}^\star$
 1: $P_0 = \emptyset$, $Z_0 = \{1, \dots, M\}$, $\mathbf{x}^{(0)} = 0$, $\mathbf{w}^{(0)} = A^T\mathbf{b}$, $s = 0$
 2: while $Z_s \neq \emptyset$ and $\max \mathbf{w}^{(s)} > 0$ do
 3:     if $n_s > 0$ then
 4:         $A_P^{(s)} = Q^{(s)} \begin{pmatrix} R^{(s)} \\ 0 \end{pmatrix}$
 5:         $\left(Q^{(s)}\right)^T A_Z^{(s)} = \begin{pmatrix} B^{(s)} \\ C^{(s)} \end{pmatrix}$
 6:     else
 7:         $C^{(s)} = A$
 8:     end if
 9:     $J_s = \mathrm{DM}(C^{(s)}, \mathbf{w}^{(s)}(n_s + 1 :), \mathbf{u}^{(s)}(n_s + 1 :), \tau_w, \tau_u, \tau_\theta, k_{\max})$
10:     move columns $\{n_s + j_1, \dots, n_s + j_{k_s}\}$ to the leading positions
    $\{n_s + 1, \dots, n_s + k_s\}$ of $A$
11:     $P_{s+1} = P_s \cup J_s$
12:     $Z_{s+1} = Z_s \setminus J_s$
13:     $\mathbf{y}_P^{(s+1)} = \mathrm{argmin} \left\| A_P^{(s+1)}\mathbf{y} - \mathbf{b} \right\|$, $\mathbf{y}_Z^{(s+1)} = 0$
14:     while $\min \mathbf{y}_P^{(s+1)} \leq 0$ do
15:         $s = s + 1$
16:         $Q = P_s \cap \left\{ i \ : \ y_i^{(s)} \leq 0 \right\}$
17:         $\alpha = \min_{i \in Q} x_i^{(s-1)} \left( x_i^{(s-1)} - y_i^{(s)} \right)^{-1}$
18:         $\mathbf{x}^{(s)} = \mathbf{x}^{(s-1)} + \alpha(\mathbf{y}^{(s)} - \mathbf{x}^{(s-1)})$
19:         move columns $\left\{ i \ : \ i \in P_s, \ x_i^{(s)} \leq 0 \right\}$ to the rightmost positions of
    $A$
20:         $P_{s+1} = P_s \setminus \left\{ i \ : \ i \in P_s, \ x_i^{(s)} \leq 0 \right\}$
21:         $Z_{s+1} = Z_s \cup \left\{ i \ : \ i \in P_s, \ x_i^{(s)} \leq 0 \right\}$
22:         $\mathbf{y}_P^{(s+1)} = \mathrm{argmin} \left\| A_P^{(s+1)}\mathbf{y} - \mathbf{b} \right\|$, $\mathbf{y}_Z^{(s+1)} = 0$
23:     end while
24:     $\mathbf{x}_P^{(s+1)} = \mathbf{y}_P^{(s+1)}$, $\mathbf{x}_Z^{(s+1)} = 0$
25:     $\mathbf{w}^{(s+1)} = A^T \left( \mathbf{b} - A\mathbf{x}^{(s+1)} \right)$
26:     $s = s + 1$
27: end while
28: $P^\star = P_s$, $Z^\star = Z_s$, $\mathbf{x}^\star = \mathbf{x}^{(s)}$
```

Let us detail how deviation maximization is employed here. Algorithm 1 is applied to the submatrix $C^{(s)}$, where $C^{(s)}$ is the matrix of size $(m-n_s)\times(n-n_s)$ obtained as follows

$$\left(Q^{(s)}\right)^T \left(A_P^{(s)}\ A_Z^{(s)}\right) = \left(\begin{array}{cc} R^{(s)} & B^{(s)} \\ 0 & C^{(s)} \end{array}\right).$$

Here $Q^{(s)}$ is the orthogonal factor of the QR decomposition of $A_P^{(s)}$, and $B^{(s)}$ is a matrix of size $n_s \times (n-n_s)$. We make use the dual vector in order to define the candidate set of indices of the deviation maximization, thus we set $\mathbf{v}_1 = \mathbf{w}^{(s)}(n_s+1:n)$, and we exploit the partial column norm vector as additional information, that is we take $\mathbf{v}_2 = \mathbf{u}^{(s)}(n_s+1:n)$. For the moment, let us use the parameters τ_θ, $\tau_w := \tau_1$ and $\tau_u := \tau_2$ without stating their values, which will be discussed in what follows. To sum up, the set $J_s = \{j_1,\dots,j_{k_s}\}$ is obtained by Algorithm 1 as follows

$$J_s = \mathrm{DM}(C^{(s)}, \mathbf{w}^{(s)}(n_s+1:), \mathbf{u}^{(s)}(n_s+1:), \tau_w, \tau_u, \tau_\theta, k_{\max}).$$

Both steps 13 and 22 require the solution of an unconstrained least squares problem in the form eq. (4.8), e.g. by QR decomposition. This can be done efficiently by exploiting low-rank QR updates/downdates. Just like the classic Lawson-Hanson algorithm, this strategy ensures that the new iterate will stay feasible, provided that first an intermediate solution $\mathbf{y}^{(s)}$ is computed and then, eventually, an analogous inner loop will keep the iterate $\mathbf{x}^{(s)}$ into the feasible region.

Let us spend the rest of this section to derive conditions on $\tau_\theta, \tau_w, \tau_u$ under which the finite termination of Algorithm 5 is guaranteed, as it was shown for Algorithm 4 in [100]. Let us state and prove the following result, which is a generalization of Lemma 23.3 of [100] in the case of multiple columns.

Lemma 6. *Let $A \in \mathbf{R}^{m\times(n-k)}$ and $A' \in \mathbf{R}^{m\times k}$ two matrices such that their concatenation $(A\ A')$ is a full column rank matrix of size $m \times n$. Let Q be the orthogonal $m \times m$ factor of the QR decomposition of A, and let*

$$Q^T(A\ A') = \left(\begin{array}{cc} R & B \\ 0 & C \end{array}\right),$$

where $R \in \mathbb{R}^{(n-k)\times(n-k)}$ is upper triangular, $B \in \mathbb{R}^{(n-k)\times k}$ and $C \in \mathbb{R}^{m-(n-k)\times k}$. Suppose that there exists $\mathbf{r} \in \mathbb{R}^m$ such that

$$(A\ A')^T\mathbf{r} = \left(\begin{array}{c} A^T\mathbf{r} \\ (A')^T\mathbf{r} \end{array}\right) = \omega \left(\begin{array}{c} \mathbf{0} \\ \mathbf{1} \end{array}\right) \tag{4.9}$$

with $\mathbf{0} = (0,\dots,0)^T \in \mathbb{R}^{(n-k)}$, $\mathbf{1} = (1,\dots,1)^T \in \mathbb{R}^k$, $\omega > 0$, and consider the solution $\left(\begin{array}{c} \mathbf{z} \\ \mathbf{z}' \end{array}\right)$ of the least squares problem

$$\min \left\| (A\ A') \left(\begin{array}{c} \mathbf{z} \\ \mathbf{z}' \end{array}\right) - \mathbf{r} \right\|^2 .$$

If the cosine matrix Θ associated to C is a strictly diagonally dominant matrix with

$$\gamma = \min_i \left\{ 1 - \sum_{j \neq i} |\theta_{ij}| \right\} > \frac{1}{2}, \tag{4.10}$$

and we have $\|\mathbf{c}_i\| \leq \tau_u \max_j \|\mathbf{c}_j\|$ for every i, where τ_u is a parameter such that

$$1 \geq \tau_u > \sqrt{\frac{3}{2} - \gamma}, \tag{4.11}$$

then $\mathbf{z}' > 0$.

Proof. Applying the matrix Q^T by the left, we get

$$Q^T \left(A\ A' | \mathbf{r}\right) = \left(\begin{array}{cc|c} R & B & \mathbf{s} \\ 0 & C & \mathbf{t} \end{array} \right),$$

where $\mathbf{s} \in \mathbb{R}^{n-k}$ and $\mathbf{t} \in \mathbb{R}^{m-(n-k)}$. Then equation (4.9) rewrites

$$\begin{aligned} \begin{pmatrix} \mathbf{0} \\ \mathbf{w} \end{pmatrix} &= (A\ A')^T \mathbf{r} = (A\ A')^T QQ^T \mathbf{r} = \begin{pmatrix} R^T & 0 \\ B^T & C^T \end{pmatrix} \begin{pmatrix} \mathbf{s} \\ \mathbf{t} \end{pmatrix} \\ &= \begin{pmatrix} R^T \mathbf{s} \\ B^T \mathbf{s} + C^T \mathbf{t} \end{pmatrix}. \end{aligned}$$

Since R is full rank, we have $\mathbf{s} = 0$ and $C^T \mathbf{t} = \mathbf{w} > 0$. On the other hand, we have

$$\left\| (A\ A') \begin{pmatrix} \mathbf{z} \\ \mathbf{z}' \end{pmatrix} - \mathbf{r} \right\| = \left\| \begin{pmatrix} R & B \\ 0 & C \end{pmatrix} \begin{pmatrix} \mathbf{z} \\ \mathbf{z}' \end{pmatrix} - \begin{pmatrix} \mathbf{s} \\ \mathbf{t} \end{pmatrix} \right\|$$

therefore the last s entries $\mathbf{z}'$ of the solution of (6) satisfy $\mathbf{z}' = \operatorname{argmin}_{\mathbf{x}} \|C\mathbf{x} - \mathbf{t}\|^2$, i.e.

$$\mathbf{z}' = C^\dagger \mathbf{t} = (C^T C)^{-1} C^T \mathbf{t} = (C^T C)^{-1} \mathbf{w} = \omega (C^T C)^{-1} \mathbf{1}. \tag{4.12}$$

Now, we would like to apply Corollary 3 to $C^T C$, in order to prove that its inverse $(C^T C)^{-1}$ is strictly diagonally dominant with positive diagonal values. This is enough to conclude that $\mathbf{z}' > 0$, since

$$z_i' = \omega \left((C^T C)^{-1}_{ii} + \sum_{j \neq i} (C^T C)^{-1}_{ij} \right) > \omega \left(\left| \sum_{j \neq i} (C^T C)^{-1}_{ij} \right| + \sum_{j \neq i} (C^T C)^{-1}_{ij} \right) \geq 0.$$

Without loss of generality, suppose $\|\mathbf{c}_1\| = \max_i \|\mathbf{c}_i\|$. Then we have $C^T C = \|\mathbf{c}_1\|^2 (I - S)$, where the matrix $S = (s_{ij})$ is given by

$$s_{ij} = \begin{cases} 1 - \frac{\|\mathbf{c}_i\|^2}{\|\mathbf{c}_1\|^2}, & i = j, \\ -\frac{\mathbf{c}_i^T \mathbf{c}_j}{\|\mathbf{c}_1\|^2}, & i \neq j, \end{cases}$$

and

$$\|S\|_\infty = \max_i \left\{ \left| 1 - \frac{\|\mathbf{c}_i\|^2}{\|\mathbf{c}_1\|^2} \right| + \sum_{j \neq i} \frac{\left|\mathbf{c}_i^T \mathbf{c}_j\right|}{\|\mathbf{c}_1\|^2} \right\}.$$

Let us show that $\|S\|_\infty < 1/2$, which concludes the proof by Corollary 3. For every i, we have

$$\frac{\|\mathbf{c}_i\|^2}{\|\mathbf{c}_1\|^2} \geq \tau_u^2 \frac{\|\mathbf{c}_1\|^2}{\|\mathbf{c}_1\|^2} = \tau_u^2,$$
$$\frac{\left|\mathbf{c}_i^T \mathbf{c}_j\right|}{\|\mathbf{c}_1\|^2} \leq \frac{\left|\mathbf{c}_i^T \mathbf{c}_j\right|}{\|\mathbf{c}_i\|\|\mathbf{c}_j\|^2} = |\theta_{ij}|,$$

and thus, since by assumption $1 - \tau_u^2 \geq 0$ and $\tau_u > \sqrt{(3-2\gamma)/2}$, we have

$$\begin{aligned} \|S\|_\infty &\leq 1 - \tau_u^2 + \max_i \sum_{i \neq j} |\theta_{ij}| \\ &= 2 - \tau_u^2 - \min_i \left(1 - \sum_{i \neq j} |\theta_{ij}| \right) \\ &= 2 - \tau_u^2 - \gamma \\ &< 2 - \left(\frac{3}{2} - \gamma \right) - \gamma = \frac{1}{2}, \end{aligned}$$

concluding the proof. □

In order to ensure that the inverse of the matrix $C^T C$ is strictly diagonally dominant with a positive diagonal, Lemma 6 introduces quite demanding thresholds in Algorithm 5, namely τ_u and τ_θ, whose values should be picked near 1, while it has not been possible to set the parameter τ_w to a value strictly smaller than one, in fact the dual values $(A')^T \mathbf{r}$ are supposed all equal to $\omega > 0$. Indeed, it would have been possible if any bound on the gap of diagonal dominance of the matrix $(C^T C)^{-1}$ were known, as shown in equation (4.12). However, as we detail in the Section 4.1.4 and we experimentally show in Section 4.3, we can relax these requirements without loosing the convergence of the method in practice, i.e. these theoretical bounds describe a worst-case scenario.

Let us restate the following theorem, proved in [123] in the single column case and here adapted to the multiple column case.

Theorem 4. *Let* $(A\ A')$ *and* $\mathbf{r}$ *satisfy hypotheses of Lemma 6, that is* $(A\ A')$ *has full rank,* $A^T \mathbf{r} = \mathbf{0}$ *and* $(A')^T \mathbf{r} = \mathbf{w} > 0$. *Consider the solution* $\begin{pmatrix} \mathbf{z} \\ \mathbf{z}' \end{pmatrix}$ *of the least squares problem*

$$\min \left\| (A\ A') \begin{pmatrix} \mathbf{z} \\ \mathbf{z}' \end{pmatrix} - \mathbf{r} \right\|^2. \tag{4.13}$$

and the solution $\begin{pmatrix} \mathbf{y} \\ \mathbf{y}' \end{pmatrix}$ *of the least squares problem*

$$\min \left\| (A\ A') \begin{pmatrix} \mathbf{y} \\ \mathbf{y}' \end{pmatrix} - \mathbf{b} \right\|^2 . \tag{4.14}$$

If $\mathbf{r}$ *is the projection of the right-hand side* $\mathbf{b}$ *on the nullspace of* A*, namely we have* $\mathbf{r} = \mathcal{P}_{\mathcal{N}(A)}(\mathbf{b}) = (I - AA^\dagger)\mathbf{b}$*, then* $\mathbf{z}' = \mathbf{y}'$*.*

Proof. Consider the matrix $(A\ A')^T(A\ A')$ and apply block Gauss elimination so that $L\ (A\ A')^T(A\ A') = U$:

$$\begin{pmatrix} (A^TA)^{-1} & 0 \\ -(A')^TA^{T\dagger} & I \end{pmatrix} \begin{pmatrix} A^TA & A^TA' \\ (A')^TA & (A')^TA' \end{pmatrix} = \begin{pmatrix} I & A^\dagger A' \\ 0 & (A')^T(I - AA^\dagger)A' \end{pmatrix}, \tag{4.15}$$

because $(AA^\dagger)^T = (AA^\dagger)$. Premultiplying by L normal equations for (4.13) we get

$$(A')^T(I - AA^\dagger)A'\mathbf{z}' = \mathbf{w},$$

and for (4.14) we get

$$(A')^T(I - AA^\dagger)A'\mathbf{y}' = (A')^T(I - AA^\dagger)\mathbf{b} = (A')^T\mathcal{P}_{\mathcal{N}(A)}(\mathbf{b}) = (A')^T\mathbf{r} = \mathbf{w}.$$

Thus the equality $\mathbf{z}' = \mathbf{y}'$ holds provided that the matrix $(A')^T(I - AA^\dagger)A'$ is nonsingular. This follows from eq. (4.15), and from the fact that A^TA is nonsingular. □

We can now prove the following result.

Theorem 5. *Consider the NNLS problem*

$$\min \|A\mathbf{x} - \mathbf{b}\|_2^2, \qquad \textit{s.t. } \mathbf{x} \geq 0.$$

Given $1 > \gamma > \frac{1}{2}$ *and* $k_{\max} \geq 1$*, Algorithm 5 terminates in a finite number of steps, provided that*

$$\tau_w = 1, \tag{4.16}$$

$$\tau_\theta \leq \frac{1 - \gamma}{k_{\max} - 1}, \tag{4.17}$$

$$\tau_u > \sqrt{\frac{3}{2} - \gamma}. \tag{4.18}$$

On termination, $\mathbf{x}$ *will be a solution vector and* $\mathbf{w}$ *will be the dual vector.*

Proof. Consider the least squares solution $\mathbf{y}^{(s+1)}$ computed at step 13 of Algorithm 5, that is $\mathbf{y}$ solves

$$\mathbf{y}_P^{(s+1)} = \underset{\mathbf{y}}{\operatorname{argmin}} \left\| A_P^{(s+1)}\mathbf{y} - \mathbf{b} \right\| \quad \mathbf{y}_Z^{(s+1)} = 0. \tag{4.19}$$

where and $P_{s+1} = P_s \cup J_s$ and $Z_{s+1} = Z_s \setminus J_s$. The choices (4.16)–(4.18) allow to apply Lemma 6 and Theorem 4, thus implying that $y_i^{(s+1)} > 0$ for $i \in J_s$, and hence, since $\mathbf{x}^{(s)} \geq 0$, we have that $\mathbf{s}^{(s+1)} := \mathbf{y}^{(s+1)} - \mathbf{x}^{(s)}$ is a feasible direction. Said $\mathbf{r}^{(s)} = \mathbf{b} - A\mathbf{x}^{(s)}$, let us define $\hat{\mathbf{z}}^{(s+1)}$ as follows

$$\hat{\mathbf{z}}^{(s+1)} = \underset{\mathbf{z}_Z^{(s+1)}=0}{\operatorname{argmin}} \|A\mathbf{z} - \mathbf{r}^{(s)}\| = \underset{\mathbf{z}_Z^{(s+1)}=0}{\operatorname{argmin}} \|A(\mathbf{x}^{(s)} + \mathbf{z}) - \mathbf{b}\|, \tag{4.20}$$

Notice that $\hat{\mathbf{z}}^{(s+1)}$ can be obtained as

$$\hat{\mathbf{z}}_P^{(s+1)} = \underset{\mathbf{z}}{\operatorname{argmin}} \|A_P^{(s+1)}\mathbf{z} - \mathbf{r}^{(s)}\|, \quad \hat{\mathbf{z}}_Z^{(s+1)} = 0. \tag{4.21}$$

Then we have

$$\begin{aligned}\|A\hat{\mathbf{z}}^{(s+1)} - \mathbf{r}^{(s)}\| &= \|A\hat{\mathbf{z}}^{(s+1)} - (\mathbf{b} - A\mathbf{x}^{(s)})\| = \|A(\hat{\mathbf{z}}^{(s+1)} + \mathbf{x}^{(s)}) - \mathbf{b}\| \\ &\geq \|A\mathbf{y}^{(s+1)} - \mathbf{b}\|,\end{aligned}$$

since $\hat{\mathbf{z}}^{(s+1)} + \mathbf{x}^{(s)}$ vanishes on Z_{s+1} and the minimum is reached at $\mathbf{y}^{(s+1)}$. On the other hand

$$\begin{aligned}&\|A\mathbf{y}^{(s+1)} - \mathbf{b}\| = \|A\mathbf{y}^{(s+1)} - A\mathbf{x}^{(s)} - \mathbf{b} + A\mathbf{x}^{(s)}\| \\ &= \|A(\mathbf{y}^{(s+1)} - \mathbf{x}^{(s)}) - \mathbf{r}^{(s)}\| \\ &\geq \|A\hat{\mathbf{z}}^{(s+1)} - \mathbf{r}^{(s)}\|,\end{aligned}$$

since $\mathbf{y}^{(s+1)} - \mathbf{x}^{(s)}$ vanishes on Z_{s+1} and the minimum is reached at $\hat{\mathbf{z}}^{(s+1)}$. As a consequence of the uniqueness of such minimizers, we have

$$\hat{\mathbf{z}}^{(s+1)} = \mathbf{s}^{(s+1)} = \mathbf{y}^{(s+1)} - \mathbf{x}^{(s)}. \tag{4.22}$$

Consider the function

$$\varphi(\varepsilon) = \frac{1}{2}\left\|A\left(\mathbf{x}^{(s)} + \varepsilon\mathbf{s}^{(s+1)}\right) - \mathbf{b}\right\|^2.$$

We have

$$\begin{aligned}\varphi'(0) &= \left(\mathbf{s}^{(s+1)}\right)^T A^T(A\mathbf{x}^{(s)} - \mathbf{b}) = -\left(\mathbf{s}^{(s+1)}\right)^T \mathbf{w}^{(s)} = -\left(\mathbf{y}^{(s+1)}\right)^T \mathbf{w}^{(s)} \\ &= \sum_{i\in J_s} y_i^{(s+1)}\left(-w_i^{(s)}\right) < 0,\end{aligned} \tag{4.23}$$

since $w_i^{(s)} > 0$ for all $i \in J_s$ and $y_i^{(s+1)} > 0$ for all $i \in J_s$. Therefore there exists $\varepsilon > 0$ such that

$$\begin{aligned}\phi\left(\mathbf{x}^{(s)} + \varepsilon\mathbf{s}^{(s+1)}\right) &= \left\|A\left(\mathbf{x}^{(s)} + \varepsilon\mathbf{s}^{(s+1)}\right) - \mathbf{b}\right\|^2 \\ &< \|A\mathbf{x}^{(s)} - \mathbf{b}\|^2 = \phi\left(\mathbf{x}^{(s)}\right),\end{aligned}$$

where ϕ is the least squares objective function defined in (4.2). By definition (4.20) and relation (4.22), we have

$$\phi\left(\mathbf{x}^{(s)}+\mathbf{s}^{(s+1)}\right) \leq \phi\left(\mathbf{x}^{(s)}+\varepsilon\mathbf{s}^{(s+1)}\right) < \phi\left(\mathbf{x}^{(s)}\right). \tag{4.24}$$

If $\mathbf{y}^{(s)}$ is feasible, then we take $\varepsilon = 1$, i.e. $\mathbf{x}^{(s)} = \mathbf{y}^{(s)}$, otherwise the algorithm enters the inner loop and we choose a smaller step length $\varepsilon = \alpha_s$, more precisely the largest possible to keep the new iterate within the feasible region. Using Karush-Kuhn-Tucker conditions, it is easy to check that

$$\begin{aligned}\phi\left(\mathbf{x}^{(s)}+\alpha_s\mathbf{s}^{(s+1)}\right) &= \min_{\varepsilon>0}\left\{\phi\left(\mathbf{x}^{(s)}+\varepsilon\mathbf{s}^{(s+1)}\right) :\ \mathbf{x}^{(s)}+\varepsilon\mathbf{s}^{(s+1)} \geq 0\right\}\\ &\leq \phi\left(\mathbf{x}^{(s)}\right).\end{aligned} \tag{4.25}$$

The above inequality ensures that the objective function ϕ is always non increasing, in particular within the inner loop iterations. The inner loops terminates each time in at most $n_s - k_s$ steps, where $n_s = |P_s|$ and $k_s = |J_s|$. Set $s = s+1$, then

$$\begin{aligned}A^T\mathbf{r}^{(s)} &= A^T\left(\mathbf{b}-A\mathbf{x}^{(s)}\right) = A^T\left(\mathbf{b}-A\left(\mathbf{x}^{(s-1)}+\alpha\mathbf{s}^{(s)}\right)\right)\\ &= A^T\left(\mathbf{b}-(1-\alpha)A\mathbf{x}^{(s-1)}-\alpha A\mathbf{y}^{(s)}\right)\\ &= (1-\alpha)A^T\left(\mathbf{b}-A\mathbf{x}^{(s-1)}\right)+\alpha A^T\left(\mathbf{b}-A\mathbf{y}^{(s)}\right)\\ &= (1-\alpha)A^T\mathbf{r}^{(s-1)}+\alpha A^T\left(\mathbf{b}-A\mathbf{y}^{(s)}\right)\end{aligned}$$

and therefore, for $i \in J_s$ we have

$$\mathbf{a}_i^T\mathbf{r}^{(s)} = (1-\alpha)\mathbf{a}_i^T\mathbf{r}^{(s-1)}+\alpha\mathbf{a}_i^T\left(\mathbf{b}-A\mathbf{y}^{(s)}\right) = (1-\alpha)w_i^{(s-1)} > 0,$$

where the second term of the summation is zero because of normal equations ($\mathbf{y}^{(s)}$ is least squares solution). On the other hand, for all other indices $j \notin J_s$ that already were in the passive set we have

$$\mathbf{a}_j^T\mathbf{r}^{(s)} = (1-\alpha)\mathbf{a}_j^T\mathbf{r}^{(s-1)}+\alpha\mathbf{a}_j^T\left(\mathbf{b}-A\mathbf{y}^{(s)}\right) = 0, \tag{4.26}$$

where both terms vanishes because of normal equations. Hence $\mathbf{r}^{(s)}$ satisfies hypotheses of Lemma 6 and we have $s_i^{(s+1)} > 0$, for $i \in J$, where $\mathbf{s}^{(s+1)}$ is the solution of

$$\min_{\mathbf{s}_Z^{(s+1)}=0}\left\|A\mathbf{s}-\mathbf{r}^{(s)}\right\|.$$

Since $\mathbf{s}^{(s+1)} = \mathbf{y}^{(s+1)} - \mathbf{x}^{(s)}$ (eq. (4.22)), we can conclude that

$$y_i^{(s+1)} > x_i^{(s)} \geq 0, \quad i \in J_s \tag{4.27}$$

meaning that at least the indices $i \in J_s$ are kept in the passive set at the end of the inner loop.
From equation (4.24), we deduce that the objective function value $\phi\left(\mathbf{x}^{(s+1)}\right)$ is strictly smaller every time step 16 is reached, hence the solution vector $\mathbf{x}^{(s+1)}$ and its associated set $P_{s+1} = \left\{i : x_i^{(s+1)} > 0\right\}$ are distinct from all previous instances at the same step 16. Since P_{s+1} is a subset of $\{1, \dots, n\}$ and there are only a finite number of such subsets, the outer loop must terminate after a finite number of iterations, thus avoiding cycling at least in exact arithmetic. As a consequence, the upper bound on the number of iteration needed for convergence is 2^n.
This is enough to conclude that Algorithm 5 terminates in a finite number of steps and $\mathbf{x}^{(s)}, \mathbf{w}^{(s)}, P_s, Z_s$ satisfy KKT conditions (4.4)–(4.5) on termination. □

Manipulating multiple columns at a time often allows cycling. This is avoided provided that values for $\tau_\theta, \tau_u, \tau_w$ are chosen in order to satisfy the assumptions of Lemma 6 with $(A\ A') = \left(A_P^{(s)}\ A_J^{(s)}\right)$. In fact, τ_θ, τ_u are taken such that (4.10) and (4.11) are satisfied, and the choice $\tau_w = 1$ ensures that

$$\left(A_P^{(s)}\ A_J^{(s)}\right)^T \mathbf{r}^{(s)} = \omega \begin{pmatrix} \mathbf{0} \\ \mathbf{1} \end{pmatrix},$$

with $\omega = \operatorname{argmax} \mathbf{w}^{(s)}$. At a first glance, the requirement $\tau_w = 1$ may suggest that only one column index at a time can be inserted to the passive set, but it turns out that, especially in the first iterations, $\operatorname{argmax} \mathbf{w}^{(s)}$ is not a singleton. Lemma 6 and Theorem 4 allow to choose a feasible descent direction at each iteration, ensuring that the objective function ϕ has a strictly smaller value within the execution of the outer loop and, consequently, the finite convergence of the algorithm.

4.1.4 The Lawson-Hanson algorithm with deviation maximization

In the previous section, we showed that a straightforward application of the deviation maximization of column selection strategy into the Lawson-Hanson algorithm introduces quite demanding values of thresholds $\tau_w, \tau_\theta, \tau_u$ in order to ensure finite convergence. Here, we show that these requirements can be relaxed without loosing finite convergence, provided that a further inner loop is added in order to ensure that a feasible descent direction is found at each outer loop iteration. The resulting procedure, namely the Lawson-Hanson algorithm with Deviation Maximization (LHDM), is presented in Algorithm 6, and it terminates in a finite number of steps as shown in Theorem 6 below.
It is shown in Chapter 3 that when the daviation maximization is used for computing a pivoted QR decomposition, even if the columns $A_P^{(s)}$ already processed are well conditioned, there could be a potentially dramatic increase in the condition number of the next iterate $A_P^{(s+1)}$. This is a well known fact which has

Algorithm 6 Lawson-Hanson with Deviation Maximization (LHDM)

```
    Inputs: $A, \mathbf{b}, \tau_w, \tau_u, \tau_\theta, k_{\max}$
    Outputs: $P^\star, Z^\star, \mathbf{x}^\star$
 1: $P_0 = \emptyset$, $Z_0 = \{1, \dots, M\}$, $\mathbf{x}^{(0)} = 0$, $\mathbf{w}^{(0)} = A^T\mathbf{b}$, $\mathbf{y}^{(0)} = 0$, $s = 0$
 2: while $Z_s \neq \emptyset$ and $\max \mathbf{w}^{(s)} > 0$ do
 3:     if $n_s > 0$ then
 4:         $A_P^{(s)} = Q^{(s)} \begin{pmatrix} R^{(s)} \\ 0 \end{pmatrix}$
 5:         $\left(Q^{(s)}\right)^T A_Z^{(s)} = \begin{pmatrix} B^{(s)} \\ C^{(s)} \end{pmatrix}$
 6:     else
 7:         $C^{(s)} = A$
 8:     end if
 9:     $J_s = \mathrm{DM}(C^{(s)}, \mathbf{w}^{(s)}(n_s+1:), \mathbf{u}^{(s)}(n_s+1:), \tau_w, \tau_u, \tau_\theta, k_{\max})$
10:     move columns $\{n_s + j_1, \dots, n_s + j_{k_s}\}$ to the leading positions
    $\{n_s + 1, \dots, n_s + k_s\}$ of $A$
11:     $P_{s+1} = P_s \cup J_s$
12:     $Z_{s+1} = Z_s \setminus J_s$
13:     $\mathbf{y}_P^{(s+1)} = \operatorname{argmin} \left\| A_P^{(s+1)}\mathbf{y} - \mathbf{b} \right\|$, $\mathbf{y}_Z^{(s+1)} = 0$
14:     while $\min \mathbf{y}_J^{(s)} \leq 0$ do
15:         delete the last element from $J_s$
16:         $P_{s+1} = P_s \cup J_s$
17:         $Z_{s+1} = Z_s \setminus J_s$
18:         $\mathbf{y}_P^{(s+1)} = \operatorname{argmin} \left\| A_P^{(s+1)}\mathbf{y} - \mathbf{b} \right\|$, $\mathbf{y}_Z^{(s+1)} = 0$
19:     end while
20:     while $\min \mathbf{y}_P^{(s+1)} \leq 0$ do
21:         $s = s + 1$
22:         $Q = P_s \cap \left\{ i \ : \ y_i^{(s)} \leq 0 \right\}$
23:         $\alpha = \min_{i \in Q} x_i^{(s-1)} \left( x_i^{(s-1)} - y_i^{(s)} \right)^{-1}$
24:         $\mathbf{x}^{(s)} = \mathbf{x}^{(s-1)} + \alpha(\mathbf{y}^{(s)} - \mathbf{x}^{(s-1)})$
25:         $P_{s+1} = P_s \setminus \left\{ i \ : \ i \in P_s, \ x_i^{(s)} \leq 0 \right\}$
26:         $Z_{s+1} = Z_s \cup \left\{ i \ : \ i \in P_s, \ x_i^{(s)} \leq 0 \right\}$
27:         $\mathbf{y}_P^{(s+1)} = \operatorname{argmin} \left\| A_P^{(s+1)}\mathbf{y} - \mathbf{b} \right\|$, $\mathbf{y}_Z^{(s+1)} = 0$
28:     end while
29:     $\mathbf{x}_P^{(s+1)} = \mathbf{y}_P^{(s+1)}$, $\mathbf{x}_Z^{(s+1)} = 0$
30:     $\mathbf{w}^{(s+1)} = A^T \left(\mathbf{b} - A\mathbf{x}^{(s+1)}\right)$
31:     $s = s + 1$
32: end while
33: $P^\star = P_s$, $Z^\star = Z_s$, $\mathbf{x}^\star = \mathbf{x}^{(s)}$
```

been observed for other column selection strategies like the standard column pivoting. A famous example is the Kahan matrix, however it is a very unlikely occurrence in practice.
Let us now prove the main result of this chapter.

Theorem 6. *Consider the NNLS problem*

$$\min \|A\mathbf{x} - \mathbf{b}\|_2^2, \qquad \textit{s.t. } \mathbf{x} \geq 0.$$

Given $1 > \gamma > 0$ and $k_{\max} \geq 1$, Algorithm 6 terminates in a finite number of steps, provided that

$$\tau_w > 0, \tag{4.28}$$
$$\tau_\theta \leq \frac{1-\gamma}{k_{\max}-1}, \tag{4.29}$$
$$\tau_u > 1-\gamma. \tag{4.30}$$

On termination, $\mathbf{x}$ will be a solution vector and $\mathbf{w}$ will be the dual vector.

Proof. In order to avoid cycling and to ensure finite convergence of Algorithm 6, we have to ensure

1. termination of the outer loop, by choosing a feasible direction $\mathbf{s}^{(s+1)}$ such that the strict inequality (4.25) still holds;

2. termination of the inner loop with a nonempty passive set.

Let us address the first task. The values chosen in (4.29)–(4.30) ensure that the matrix $A_P^{(s+1)} = (A_P^{(s)}\ A_J^{(s)})$ has numerically linearly independent columns, as it is shown in Theorem 2 of Chapter 3, where an approximation of the smallest singular value is derived. Consider $\mathbf{s}^{(s+1)}$ that solves the triangular linear system

$$R^{(s+1)}\mathbf{s}_P^{(s+1)} = \left(\left(Q_1^{(s+1)}\right)^T \mathbf{r}^{(s)}\right)^{(s)}, \quad \mathbf{s}_Z^{(s+1)} = 0,$$

where $Q^{(s+1)}\ R^{(s+1)}$ is the QR decomposition of $\left(A_P^{(s)}\ \mathbf{a}_{j_1} \ldots \mathbf{a}_{j_{k_s}}\right)$. Since the choices of τ_w, τ_u in (4.28) and (4.30) do not allow to apply Lemma 6, it does not follow that $\mathbf{s}^{(s+1)}$ is a feasible direction. If $\mathbf{s}^{(s+1)}$ is not a feasible direction, we drop the last column of $A_J^{(s)}$ and the corresponding index j_{k_s} is moved back to the active set Z_{s+1}. We then look for the solution $\mathbf{s}^{(s+1)}$ of the downdated triangular linear system

$$R_1^{(s+1)}\mathbf{s}_P^{(s+1)} = \left(\left(Q_1^{(s+1)}\right)^T \mathbf{r}^{(s)}\right)^{(s)}, \quad \mathbf{s}_Z^{(s+1)} = 0, \tag{4.31}$$

where $Q_1^{(s+1)}\ R_1^{(s+1)}$ is the QR decomposition of $\left(A_P^{(s)}\ \mathbf{a}_{j_1} \ldots \mathbf{a}_{j_{k_s-1}}\right)$. Notice that $R_1^{(s+1)}$ can be obtained by dropping the last row and the last column of

$R^{(s+1)}$, while $Q_1^{(s+1)}$ can be obtained as the product $Q^{(s+1)}G_1$, where G_1^T is an orthogonal transformation that zeros out the last diagonal element of $R^{(s+1)}$, e.g. a Givens transformation. We continue dropping a column at a time from $A_J^{(s)}$ until $\mathbf{s}^{(s+1)}$ turns out to be a feasible solution and inequalities (4.23) and (4.25) are satisfied. It should be noticed that this procedure terminates in at most $k_s - 1$ iterations, i.e. when only the first column of $A_J^{(s)}$ is left. In this case, the computed solution is equal to that provided by an outer iteration of the standard Lawson-Hanson algorithm, thus $\mathbf{s}^{(s+1)}$ is indeed a feasible descend direction.
For what concerns the second task, once a feasible direction $\mathbf{s}^{(s+1)}$ is found, then the same arguments used in Theorem 5 can be used to show that (4.27) holds for the indices in J_s, thus ensuring the finite convergence of the inner loop with a nonempty passive set. □

In practice, instead of computing a tentative feasible direction (4.31), we compute $\mathbf{y}^{(s+1)}$ as the solution of

$$R_i^{(s+1)}\mathbf{y}_P^{(s+1)} = \left(\left(Q_1^{(s+1)}\right)^T \mathbf{b}\right)^{(s)}, \quad \mathbf{y}_Z^{(s+1)} = 0, \quad i = 0, 1, \ldots,$$

until the last $k_s - i$ entries of $\mathbf{y}_P^{(s+1)}$ are positive.
In Section 4.3, we give experimental evidence of the significant performance gain that this strategy can lead.

4.1.5 Implementation of Lawson-Hanson with deviation maximization algorithm

In the previous sections we described the the state-of-the-art implementation of the Lawson-Hanson Algorithm 4, pointing out that it does not involve memory communications. However, it does not lead to high performance since it is mainly based upon BLAS-2 level operations. In the Lawson-Hanson with deviation maximization Algorithm 6, both steps 13 and 27 require the solution of an unconstrained least squares problem of type (4.8), solved e.g. by QR decomposition, just like the standard algorithm. The deviation maximization block pivoting allows to carry out step 13 with low-rank QR updates by means of the so-called WY representation [124], which can be carried out by BLAS-3 level operations. As a drawback, in order to exploit data locality in computations, it is necessary to physically exchange the matrix columns in memory, thus introducing memory communications. However, as shown in Section 4.3, it turns out to be a better choice for performance. The QR downdates in step 27 are carried out by means of Givens transformations, as in the implementation of the Lawson-Hanson Algorithm 4.

4.2 Sparse recovery

In this section we present the problem of sparse recovery and its connections with nonnegative least squares and with the deviation maximization technique. Sparse recovery is a fundamental problem in compressed sensing, signal denoising, statistical model selection and related fields, and it consists in finding the sparsest solution to an underdetermined system of equations, see e.g. [69, 67]. There is a wide literature in the field of signal processing, in which a matrix is usually referred as a dictionary and its columns are called atoms. Here, we abandon this terminology in favour of the classical linear algebra one.

Given a vector $\mathbf{x} \in \mathbb{R}^n$, we define its support $S = S(\mathbf{x}) = \{i : \; x_i \neq 0\}$. We aim at identifying the solution $\mathbf{x}^\star$ of the linear system $A\mathbf{x} = \mathbf{b}$ with the sparsest support $S^\star$. Formally, we consider the optimization problem

$$\min \|\mathbf{x}\|_0, \qquad \text{s.t.} \quad A\mathbf{x} = \mathbf{b}. \tag{4.32}$$

Recall the "ℓ_0-norm" of a vector $\mathbf{x}$ is defined as

$$\|\mathbf{x}\|_0 = |S(\mathbf{x})| = |\{i : x_i \neq 0\}|\,,$$

and it is not an actual norm because it does not hold that for any scalar s we have $\|s\mathbf{x}\| = |s| \; \|\mathbf{x}\|$. Actually, problem (4.32) is NP-hard, and in general we seek an approximate solution. It is well known that we can solve instead the so-called *Basis Pursuit* (BP) [44] problem

$$\min \|\mathbf{x}\|_1, \qquad \text{s.t.} \quad A\mathbf{x} = \mathbf{b}, \tag{4.33}$$

which is a convex optimization problem, since $\|\mathbf{x}\|_1$ is a convex function of x. Indeed, the two problems yield the same solution provided it is sparse enough. This result is known as $\ell_0 - \ell_1$ equivalence, and it has been found empirically [46] and theoretically [68, 72].

4.2.1 Exact recovery

There are several conditions under which we have $\ell_0 - \ell_1$ equivalence, i.e. an exact recovery. In the literature, these conditions are usually found by showing the uniqueness of solution to problems (4.32) and (4.33) and their coincidence. As a consequence of these results, efficient algorithms to solve (4.33) can be used to find the sparsest solution to an underdetermined system of equations.

4.2.1.1 Uniqueness based on RIP

Let us recall the following definition.

Definition 3. *Consider a matrix $A \in \mathbb{R}^{m\times n}$, and suppose that there exists $\delta_s \in (0,1)$ such that for every $m \times s$ submatrix A_s of A and for every $y \in \mathbb{R}^s$ we have*

$$(1-\delta_s)\|y\|_2^2 \leq \|A_s y\|_2^2 \leq (1-\delta_s)\|y\|_2^2. \tag{4.34}$$

*We say that A has the s-order **Restricted Isometry Property** (RIP) with constant δ_s.*

For a given s, one is usually interested in the smallest constant δ_s for which (4.34) holds, which is referred as the s-order restricted isometry constant.

A popular result [39] states that if $\delta_{2s} < 1$, then problem (4.32) has a unique solution $\mathbf{x}$ with support size obeying $\|\mathbf{x}\|_0 \leq s$. Moreover, if $\delta_{2s} < \sqrt{2} - 1$, then the solution to problem (4.33) also solves problem (4.32). However, the problem of establishing whether a given matrix A fulfills the s-order RIP is NP-hard in general [135].

4.2.1.2 Uniqueness based on ERC

Let us state the following result from [138].

Theorem 7. *Consider a linear system $A\mathbf{x} = \mathbf{b}$. If there exists a solution $\bar{\mathbf{x}}$ with support $\bar{S} = S(\bar{\mathbf{x}})$ such that*

$$\max_{i \notin \bar{S}} \left\| A_{\bar{S}}^{\dagger} \mathbf{a}_i \right\|_1 < 1, \tag{4.35}$$

then $\bar{\mathbf{x}}$ is the unique solution to the minimum ℓ_0 problem (4.32)*, which can be recovered by solving the minimum ℓ_1 problem* (4.33)*.*

We refer to (4.35) as **Exact Recovery Condition** (ERC). This provides an easy sufficient optimality check for a given support S, but finding a support S satisfying (4.35) is a combinatorial problem.

4.2.1.3 Uniqueness based on mutual coherence

Let us briefly introduce some definitions in order to help the reader to close the gap between numerical linear algebra and compressed sensing. Let $\Theta = (\theta_{ij})$ be the cosine matrix associated to the matrix A, as introduced in (2.11). Please note that θ_{ij} is called the cross-correlation between the columns $\mathbf{a}_i, \mathbf{a}_j$ in signal processing.

Definition 4. *The **coherence** or **mutual coherence** or **two-sided coherence** of a matrix A is defined as*

$$\mu(A) = \max_{i<j} |\theta_{ij}| \left(= \max_{i>j} |\theta_{ij}| \right). \tag{4.36}$$

In [34], it is shown that if a solution $\mathbf{x}$ with $\|\mathbf{x}\|_0 \leq s$ exists and

$$\mu(A) < \frac{1}{2s-1} \tag{4.37}$$

then the solution $\mathbf{x}$ of problem (4.32) is unique. The condition above is easy to verify for a given matrix A, however it is quite demanding.

4.2.2 Sparse recovery by nonnegative least squares

The nonnegativity constraint is known to naturally produce sparse solutions, see e.g. [31, 76, 143, 144]. An important outcome of this body of work is that nonnegativity alone may attain a satisfactory sparse recovery. Notice that arbitrary signed sparse recovery is easily achievable. Given $\mathbf{x} \in \mathbb{R}^n$, decompose it as $\mathbf{x} = \mathbf{x}^+ - \mathbf{x}^-$, where $\mathbf{x}^+ \geq 0$ and $\mathbf{x}^- \geq 0$. Then the solutions of the linear system $A\mathbf{x} = \mathbf{b}$ can be attained as the solutions of the nonnegative least squares problem

$$\min_{\bar{\mathbf{x}}} \left\| \bar{A}\bar{\mathbf{x}} - \mathbf{b} \right\|_2^2, \quad \tilde{\mathbf{x}} \leq 0, \tag{4.38}$$

where $\bar{\mathbf{x}} = \begin{pmatrix} \mathbf{x}^+ \\ \mathbf{x}^- \end{pmatrix} \in \mathbb{R}^{2n}$ and $\bar{A} = (A \;\; -A)$. This has been shown e.g. in [76] and it is sometimes referred as "positivity trick".

4.2.2.1 Uniqueness for NNLS problem

This topic has been treated in [31, 127, 76].

Definition 5. *Given a matrix $A \in \mathbb{R}^{m\times n}$, we say that the columns of A are in General Linear Position (GPL) if*

$$A_J\mathbf{y} = 0 \quad \Rightarrow \quad \mathbf{y} = 0,$$

for all $\mathbf{y} \in \mathbb{R}^{|J|}$, with $J \subseteq \{1, \dots, n\}$, $|J| \leq \min\{m, n\}$.

When A is underdetermined, GPL means that A does not contain more linear dependencies than it must have.

A nonnegative solution to the linear system $A\mathbf{x} = \mathbf{b}$ exists provided that $\mathbf{b}$ is an element of the convex cone $\mathcal{C}_A$ generated by A, namely

$$\mathcal{C}_A = \{\mathbf{b} \in \mathbb{R}^m \;:\; \mathbf{b} = A\mathbf{x},\; \mathbf{x} \geq 0\}. \tag{4.39}$$

Suppose that $\mathcal{C}_A$ has a nonempty interior $\mathring{\mathcal{C}}_A$. If $\mathbf{b} \in \mathring{\mathcal{C}}_A$, then $\mathbf{b}$ does not have a unique representation in terms of nonnegative linear combination of the columns of A. therefore, in order to have a unique solution then $\mathbf{b}$ must lie on the boundary of $\mathcal{C}_A$, $\partial\mathcal{C}_A$. A necessary condition for $\partial\mathcal{C}_A \neq \emptyset$ is that $\mathcal{C}_A$ is pointed, i.e. $\mathcal{C}_A \cap \mathcal{C}_{-A} = \{\mathbf{0}\}$. A sufficient condition (also necessary under GLP) for $\mathcal{C}_A$ to be pointed is that $A \in \mathcal{M}^+$, defined as

$$\mathcal{M}^+ = \left\{A \in \mathbb{R}^{m\times n} : \; \exists\, \mathbf{h} \in \mathbb{R}^m \text{ s.t. } A^T\mathbf{h} = \mathbf{w} > 0\right\}. \tag{4.40}$$

A necessary condition in order to have a unique solution to the NNLS problem (4.1) is that A belongs to $\mathcal{M}^+$ [143, 125]. Moreover, in [125] it is shown that

$$A \in \mathcal{M}^+ \quad \Leftrightarrow \quad 0 \notin \mathcal{P}_A,$$

where $\mathcal{P}_A$ is the convex hull generated by the columns of A, namely

$$\mathcal{P}_A = \left\{\mathbf{x} \in \mathbb{R}^m \;:\; \mathbf{x} = A\boldsymbol{\lambda},\; \boldsymbol{\lambda} \geq 0,\; \mathbf{1}^T\boldsymbol{\lambda} = 1\right\}.$$

For a given A, we can verify if $A \in \mathcal{M}^+$ by solving the following least squares problem with linear constraints

$$\min \|A\boldsymbol{\lambda}\|_2^2, \quad \text{s.t. } \boldsymbol{\lambda} \geq 0, \ \mathbf{1}^T\boldsymbol{\lambda} = 1. \tag{4.41}$$

To see this, let $\boldsymbol{\lambda}^\star$ be a solution of the solution above. If the optimal value attained is zero, namely $\|A\boldsymbol{\lambda}^\star\|_2^2 = 0$, then clearly $0 \in \mathcal{P}_A$ and $A \in \mathcal{M}^+$; on the other hand, if $\|A\boldsymbol{\lambda}^\star\|_2^2 > 0$ then 0 cannot belong to $\mathcal{P}_A$ and we have $A \notin \mathcal{M}^+$. The uniqueness provided by (4.37) cannot be extended as it is to NNLS when the solution vector is nonnegative. Clearly, if a nonnegative solution $\mathbf{x}$ of $A\mathbf{x} = \mathbf{b}$ with $\|\mathbf{x}\|_0 \leq s$ exists and

$$\mu(A) < \frac{1}{2s-1}, \tag{4.42}$$

then it is the unique solution of minimum ℓ_1-norm and it is also nonnegative, however it may not be the unique nonnegative solution to this linear system of equations. Thus, any NNLS solver has no guarantees to find the minimum ℓ_1-norm solution in general, while any ℓ_1 solver does.

We can extend this result by adding the necessary condition (4.40).

Theorem 8. *Consider a linear system $A\mathbf{x} = \mathbf{b}$ such that $A \in \mathcal{M}^+$. If there exists a nonnegative solution $\bar{\mathbf{x}}$ with $\|\bar{\mathbf{x}}\|_0 \leq s$ and*

$$\mu(A) < \frac{1}{2s-1}, \tag{4.43}$$

then $\bar{\mathbf{x}}$ is the unique nonnegative solution and the minimum ℓ_1-norm solution to $A\mathbf{x} = \mathbf{b}$.

Let us now consider the matrix $\Gamma = (AD^{-2})^T A = (A^T AD^{-2})^T$ whose entries are given by

$$\gamma_{ij} = \frac{\mathbf{a}_i^T \mathbf{a}_j}{\|\mathbf{a}_i\|^2}.$$

Definition 6. *The **one-sided coherence** of a matrix A is defined as*

$$\nu(A) = \max_{i \neq j} |\gamma_{ij}|. \tag{4.44}$$

The following results are from [31].

Lemma 7. *For any matrix A, we have*

$$\nu(A) \geq \mu(A). \tag{4.45}$$

Theorem 9. *Consider a linear system $A\mathbf{x} = \mathbf{b}$ such that $A \in \mathcal{M}^+$. If there exists a nonnegative solution $\bar{\mathbf{x}}$ with $\|\bar{\mathbf{x}}\|_0 \leq s$ and*

$$\nu(A) \leq \frac{1}{2s-1}, \tag{4.46}$$

or equivalently $s \leq \frac{\nu(A)+1}{2\nu(A)}$, then $\bar{\mathbf{x}}$ is the unique nonnegative solution to $A\mathbf{x} = \mathbf{b}$.

Notice that even if we have uniqueness of the solution, we cannot ensure it when we apply the positivity trick (4.38). To see it, suppose take a matrix A that belong to $\mathcal{M}^+$, that is there exists $\mathbf{h}$ such that $A^T\mathbf{h} = \mathbf{w} > 0$. We look for $\mathbf{h}'$ such that $(A \ \ -A)^T\mathbf{h}' = \mathbf{w}'' > 0$, that is we look for $\mathbf{w}'$ that solves

$$\begin{cases} A^T\mathbf{h}' = \mathbf{w}' > 0, \\ -A^T\mathbf{h}' = -\mathbf{w}' > 0. \end{cases}$$

Clearly, the linear system of inequalities above has no solution.

4.2.3 Sparsity enhancing methods and approximate measurements

Let us now consider the more general situation in which the linear system does not have to be exactly solved, i.e. we rather look for an approximate solution satisfying $A\mathbf{x} = \mathbf{b} + \boldsymbol{\varepsilon}$, where $\boldsymbol{\varepsilon}$ is the vector containing the error in the measurements. A famous method addressing this problem is the so-called LASSO [134]. Following Foucart and Koslicki [76], we address to the solution of the following closely related problem

$$\min \|\mathbf{x}\|_1^2 + \lambda\|A\mathbf{x} - \mathbf{b}\|_2^2. \tag{4.47}$$

As we already pointed out that no theoretical result states that NNLS solvers ensure sparse recovery. Problem (4.47) can be interpreted as a sparsity enhancing or squared ℓ_1-regularization method consisting in adding a penalization term to the objective function. This problem can be thought as a weighted-sum form of a multiobjective problem where the parameter λ controls the tradeoff between the two objectives. This technique suffers from an evident drawback that is the choice of λ, which is not explicitly related norm to the sparsity, nor to the linear system.

In [76], problem (4.47) is recast as a nonnegative least squares problem. By introducing $\bar{\mathbf{x}} = \begin{pmatrix} \mathbf{x}^+ \\ \mathbf{x}^- \end{pmatrix} \in \mathbb{R}^{2n}$ and $\bar{A} = (A \ \ -A)$, problem (4.47) becomes

$$\min \left\| \begin{pmatrix} 1 \dots 1 \\ \lambda\bar{A} \end{pmatrix} \bar{\mathbf{x}} - \begin{pmatrix} 0 \\ \lambda\mathbf{b} \end{pmatrix} \right\|_2^2, \qquad \text{s.t. } \bar{\mathbf{x}} \geq 0 \tag{4.48}$$

which will be referred as ℓ_1-NNLS. A nice feature of this form is that the matrix involved clearly belongs to $\mathcal{M}^+$ and uniqueness of the solution is possible for any λ. The equivalence between problem (4.33) and problem (4.48) above can be established for $\lambda \to \infty$, meaning that the squared residual is much more important then the ℓ_1-norm, which becomes more and more vacuous as λ gets larger. This suggests it could be sufficient to solve a simple NNLS problem in order to get a sparse solution.

4.3 Comparison with existing algorithms

In this section, we compare LHDM with LH and other methods for sparse recovery on a large set of instances. We provide `nnls_dm`, a C implementation of LHDM, here used for numerical tests, freely available at `https://github.com/mdessole/lhdm`. In our tests, we compare `nnls_dm` with `nnls`, a C implementation of the Lawson-Hanson algorithm available at `https://software.sandia.gov/appspack/version3/nnls_8c-source.html`. Furthermore, we compare LHDM with the other ℓ_1 solvers tested in [102], whose implementation is open source and available online. As done in [102], we used the default setting for optional parameters of these solvers. Our purpose is not to provide an exhaustive experimental comparison between the methods, which has been done in the cited article, but instead to assess the substantial performance gain attained by the deviation maximization to the Lawson-Hanson algorithm, i.e. by the LHDM algorithm, and its competitiveness with the other methods publicly available for sparse recovery. Actually, numerical experiments reveal that LHDM is a good choice for sparse recovery, for a wide class of instances.

Numerical tests have been carried out on two different datasets. The first dataset is freely available online and it has been created by Lorenz et al. [102]. It contains many instances satisfying the ERC (4.35), thus representing a favorable situation for compressed sensing in terms of recoverability of sparse solutions via ℓ_1 minimization. We also include instances not satisfying the ERC in order to assess solvers' performance outside the so-called $\ell_0 - \ell_1$ equivalence. In our test we have used 444 instances over 548, since we restrict our investigation to dense matrices from this dataset. In fact, dealing with sparse matrices would require a dedicated implementation of LHDM which can be addressed in the future. In order to simplify the comparison with the results here presented with the ones in [102], we keep the same indexing from 1 to 548 to show the results. Results concerning sparse instances are left blank. The dimension of the instances here considered ranges from hundreds to thousands rows and columns.
The second dataset is made of a few instances of a problem in multivariate polynomial optimization, see [61, 62], in which a sparse nonnegative solution of a Vandermonde-like linear system is sought. Further discussion on this topic, together with a numerical package for its solution can be found in Appendix A. In our tests, we have used 6 instances consisting of large linear systems whose nonnegative solution is not unique and for which a solution satisfying the ERC (4.35) is not known, thus representing difficult instances of the sparse recovery problem. Note that the ℓ_1 solvers here considered do not enforce nonnegativity of the solution. The dimension of the instances here considered ranges from hundreds to thousands rows and from thousands to millions of columns.

Figure 4.1 shows a general comparison between LHDM and each other method here considered concerning execution times (figures on the right) and residual error (figures on the left), for the instances of the first dataset. We used the positivity trick presented in (4.38) in order to achieve an arbitrary signed solution by means of LHDM, which succeeded in exact recovery for all tests verifying the ERC, meaning that the solution with smallest support was found.

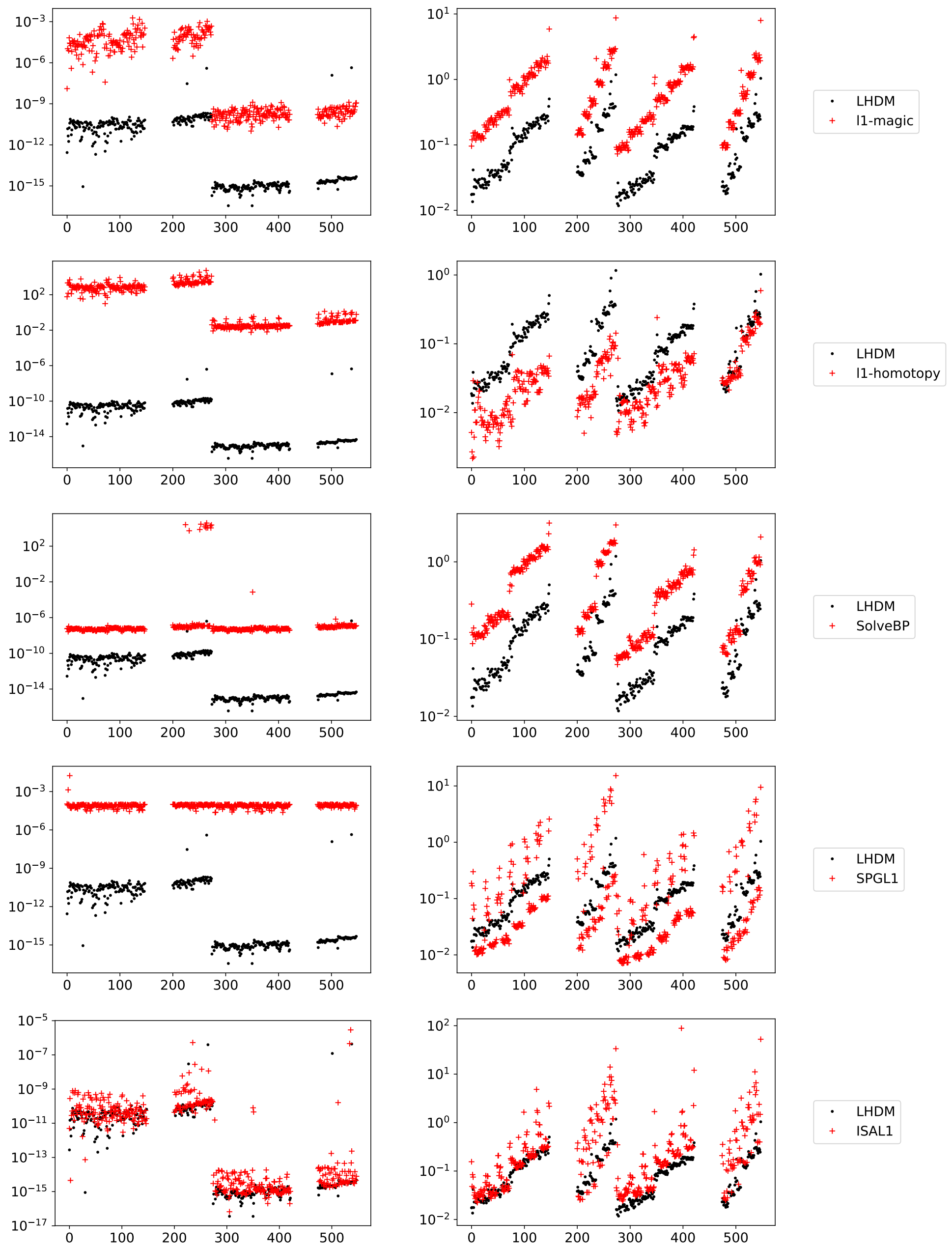


Figure 4.1: Performance of LHDM versus the five methods tested on the first dataset.

Table 4.1 shows the comparison between methods in terms of execution times and solution quality, for the instances of the second dataset. For each test and for each method, we list the residual of the solution found ($\|A\mathbf{x}-\mathbf{b}\|_2$), execution times in seconds (time (s)) and the cardinality of the support of the solution found ($\|\mathbf{x}\|_0$). Experiments in which an error occurred or whose execution times took more than 10 minutes are labeled by $-$. Notice that ℓ_1 solvers look for the minimum ℓ_1 solution of arbitrary sign, while LH and LHDM look for a nonnegative solution. As already pointed out, the nonnegative solutions to these linear systems is not unique and it is not known a solution satisfying any of the criteria presented in Section 4.2.1. Thus these instances present further difficulties.

Chebyshev $n = 6, d = 3$	$\|A\mathbf{x} - \mathbf{b}\|_2$	time (s)	$\|\mathbf{x}\|_0$	Halton $n = 5, d = 4$	$\|A\mathbf{x} - \mathbf{b}\|_2$	time (s)	$\|\mathbf{x}\|_0$
l1-magic	–	–	–	l1-magic	3.548e-13	8.050	10000
l1-homotopy	2.047e-03	19.006	405	l1-homotopy	1.831e-03	3.924	670
SolveBP	4.045e-07	3.592	110592	SolveBP	1.566e-07	0.943	10000
SPGL1	9.052e-05	5.869	100292	SPGL1	9.646e-05	0.673	9580
ISAL1	–	–	–	ISAL1	2.575e-07	39.387	10000
LH	3.062e-05	59.331	324	LH	1.405e-05	23.796	879
LHDM	2.909e-05	13.023	326	LHDM	8.745e-06	2.441	915
Chebyshev $n = 3, d = 4$	$\|A\mathbf{x} - \mathbf{b}\|_2$	time (s)	$\|\mathbf{x}\|_0$	Chebyshev $n = 2, d = 5$	$\|A\mathbf{x} - \mathbf{b}\|_2$	time (s)	$\|\mathbf{x}\|_0$
l1-magic	–	–	–	l1-magic	–	–	–
l1-homotopy	2.707e-03	10.520	106	l1-homotopy	2.466e-03	3.678	31
SolveBP	3.742e-07	4.793	331776	SolveBP	2.384e-07	9.457	1048576
SPGL1	9.993e-05	11.816	2784	SPGL1	9.315e-05	12.184	32302
ISAL1	–	–	–	ISAL1	–	–	–
LH	1.734e-05	52.704	149	LH	2.111e-05	78.857	79
LHDM	1.534e-05	18.113	159	LHDM	1.653e-05	38.214	90
Multibubble $n = 10, d = 3$	$\|A\mathbf{x} - \mathbf{b}\|_2$	time (s)	$\|\mathbf{x}\|_0$	Halton $n = 2, d = 10$	$\|A\mathbf{x} - \mathbf{b}\|_2$	time (s)	$\|\mathbf{x}\|_0$
l1-magic	1.196e-13	19.188	18915	l1-magic	8.782e-13	7.852	10000
l1-homotopy	2.057e-03	38.676	1452	l1-homotopy	1.420e-03	2.343	510
SolveBP	2.095e-07	3.023	18915	SolveBP	1.833e-07	0.793	10000
SPGL1	1.170e-05	1.192	18915	SPGL1	5.598e-05	0.543	1540
ISAL1	5.322e-17	109.900	18915	ISAL1	4.008e-07	12.989	10000
LH	8.536e-05	115.116	1393	LH	1.056e-05	21.375	817
LHDM	7.203e-05	9.526	1432	LHDM	7.491e-06	2.070	891

Table 4.1: Results of LHDM versus the five methods tested on the second dataset.

For both datasets, we used a fixed choice of the thresholds used in LHDM, that is $\tau_w = 0.5, \tau_\theta = 0.3, \tau_u = 0.1$ and $k_{\max} = 32$, hence giving evidence of the robustness of the method proposed without the need of choosing problem dependent parameters. The values we adopted gave the best overall execution times and were tuned on the first dataset only.

Figure 4.2 and Figure 4.3 summarize the role of $k_{\max}$ on the execution times. In Figure 4.2, it is shown the speedup of LHDM with respect to LH: the figure clearly highlights the performance gain produced by the deviation maximization, which turns out to be fundamental in order to make Lawson-Hanson algorithm competitive against to other methods for sparse recovery. For example, when a larger $k_{\max}$ is chosen the time to solution generally decreases, but we observe that the deviation maximization may struggle to find a large set of indices to add to the passive set, as shown by the figure on the left in which we see the evolution of the cardinality of the passive set for $k_{\max} = 8, 16, 32$. The figures below show that the larger the value of $k_{\max}$ is, the larger the average increase of the cardinality of the passive set is, as we might expect. In most cases, a larger $k_{\max}$ leads to reach the solution in a smaller number of iteration (cf. test case 512 (right) of the first dataset shown in Figure 4.3 on the right); however, it may cause the algorithm to convergence in a larger number of iterations, hence with larger execution times, to a solution whose support is larger than the optimal one (cf. test case 500 of the first dataset shown in Figure 4.3 on the left). A preliminary investigation led $k_{\max} = 32$ to be our default choice, although it could be interesting to devise an adaptive strategy for the choice of $k_{\max}$. This is far from trivial and out of scope of the current work. We rather apply the strategy implemented in Algorithm 6 that allows to set a large $k_{\max}$, which can be chosen by considering only memory usage and computational cost of deviation maximization.

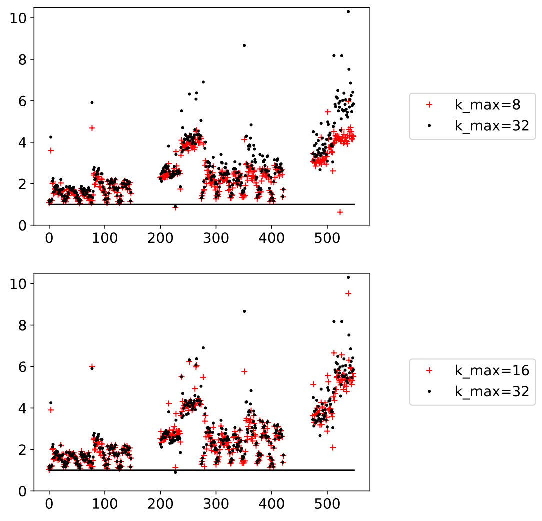


Figure 4.2: Speedup of LHDM over LH for different values of $k_{\max}$.

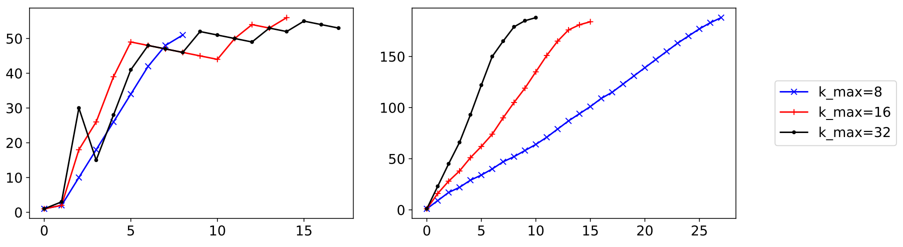


Figure 4.3: Evolution of the cardinality of the passive set during the execution of LHDM for different values of $k_{\max}$.

4.4 Conclusions and future perspectives

In this chapter we have presented a new NNLS solver, namely the Lawson-Hanson with deviation maximization algorithm. This method relies on correlation analysis by means of cosine evaluation in order to select a subset of sufficiently linearly independent descent directions and we have used it as column pivoting strategy to devise a new active set method. We have provided a theoretical analysis proving the finite convergence of LHDM, which is shown to terminate in at most 2^n steps, just like the standard Lawson-Hanson. This is a worst case bound, and in practice a polynomial rate of convergence is observed.

This chapter comes together with `nnls_dm`, an open source C implementation of LHDM, available online at `https://github.com/mdessole/lhdm`. Extensive numerical experiments have been carried out over a wide set of instances, confirming that LHDM yields a significant performance gain over LH with an average speedup of $3\times$ with peaks up to $10\times$. Moreover, we compared LHDM with several ℓ_1 solvers whose implementation is publicly available online. Numerical testing has confirmed that LHDM is competitive with ℓ_1 solvers for sparse recovery in terms of solution quality and execution times, revealing that LHDM is a good choice for sparse recovery for a wide class of instances.

We did not cope with sparse instances, since it would require a dedicated implementation based on sparse QR with column pivoting. Finally, it would be interesting to extend the deviation maximization as pivoting strategy to other problems which require column selection, e.g. more general constrained optimization problems such as least squares problems with linear inequality constraints.

Part II

Parallel computing for sparse numerical linear algebra

Chapter 5

Solution of BABD systems

In this chapter we discuss numerical methods to solve Bordered Almost Block Diagonal (BABD) systems on a massively parallel architecture like a GPU. Such systems arise in a variety of computational problems: the solution of boundary value ordinary differential equations (BVODEs), Markov chains modeling [98], parameter estimation with non-linear DAE models [23], optimal control [16] and many others. For a survey of serial and parallel algorithms see [4] and [73].

The most occurring BABD matrices have the following structure

$$\begin{pmatrix} S_0 & T_0 & & & \\ & S_1 & T_1 & & \\ & & \ddots & \ddots & \\ & & & S_{N-1} & T_{N-1} \\ B_a & & & & B_b \end{pmatrix}, \tag{5.1}$$

where B_a, B_b, S_i, T_i, $i = 0, \ldots, N-1$ are square blocks all of the same size $n \times n$. A special case of BABD matrices are Almost Block Diagonal (ABD) matrices, which typically have the following structure

$$\begin{pmatrix} D_a & & & & \\ S_0 & T_0 & & & \\ & S_1 & T_1 & & \\ & & \ddots & \ddots & \\ & & & S_{N-1} & T_{N-1} \\ & & & & D_b \end{pmatrix}, \tag{5.2}$$

where D_a, D_b are rectangular blocks of size $q \times n$ and $(n-q) \times n$ respectively, with $0 < q < n$.

As an example, such matrices mainly arise in the numerical solution of Boundary Value problems for Ordinary Differential Equations (BVODEs)

$$y' = f(x, y(x)), \qquad y, f \in \mathbb{R}^n,\ x \in [a, b] \tag{5.3}$$

$$g(y(a), y(b)) = 0, \qquad g \in \mathbb{R}^n. \tag{5.4}$$

where numerical methods like finite differences, multiple shooting or orthogonal spline collocation are implemented using a linearization, which is evaluated on a domain discretization of type $a = x_0 < x_1 < \cdots < x_N = b$, and lead to a sequence of a BABD or ABD linear systems.

When the boundary conditions (equation (5.4)) can be rewritten, possibly by a permutation, in the form

$$g(y(a), y(b)) = \begin{pmatrix} g_a(y(a)) \\ g_b(y(b)) \end{pmatrix}, \quad g_a \in \mathbb{R}^q, g_b \in \mathbb{R}^{n-q}, \tag{5.5}$$

we say that the problem shows separated boundary conditions, and it yields ABD matrices, otherwise we speak of nonseparated boundary conditions, which give BABD matrices. In the literature, Wright [145] showed that Gaussian elimination with no pivoting or row partial pivoting is potentially unstable on BABD matrices and can lead to overflow. It can be observed e.g. with matrices arising from the minimum lap-time simulator problem [16], if the pivoting is restricted to each row block (in order to reduce fill-in in the triangular factors). If, on the other hand, one allows partial pivoting to the whole matrix, then blow up is avoided at cost of a dense LU decomposition.

Between the 1980s and the 1990s, the special ABD and BABD structures have been exploited in a number of algorithms to minimize fill-in and computational cost without compromising stability, see [4]. Naive approaches consider ABD systems as banded block tridiagonal systems. Such approaches are undesirable for many reason, not last the introduction of fill-in in the solution procedure, leading to significant inefficiencies. We propose a new algorithm tailored for massively parallel architectures, which ensures a better work-steps balance and remarkable memory savings.

5.1 Existing direct solvers for ABD and BABD systems

The scope of this section is to briefly review the algorithms seen in [4] and consider them working on massively parallel architectures such as GPUs. Furthermore we introduce a new algorithm well suited for massive parallel computing hardware, that we call PARASOF.

In general, direct sequential solvers for ABD/BABD systems rely on the classical Gaussian elimination scheme (with different kinds of pivoting), which results in limited opportunity for parallelism when handling banded systems [77, Sec. 5.1]. This limitation becomes more pronounced the narrower the system's bandwidth is. For this reason we focus on specific algorithms for direct parallel ABD/BABD solvers. For what concerns in particular BABD solvers, Wright [145] showed that Gaussian elimination with row interchanges can be unstable. Amodio et al. [4] state that there exists no dedicated sequential solver to their knowledge, and they proposed some alternative approaches based on the available software at the time. As an option, an alternative guaranteed stable approach is to write the BABD system as an ABD system of twice the size by introducing dummy variables to separate boundary conditions. For what concerns parallel solvers, a robust and accurate method for BABD systems involves the reduction of the coefficient matrix into a sparse upper triangular matrix using a structured orthogonal factorization (SOF or SQR) as described in [146]. More recent approaches [3] combine cyclic reduction and the methods of the survey, and they are more suited for multi-core architectures, such as processors with up to 8 or 16 cores. Other existing approaches in the literature rely upon domain decomposition, but they apply to hybrid direct-iterative algorithms and therefore are not considered in this work. For example, the "tearing" method of Gallopoulos [77] is interesting but it cannot be reduced to a pure direct algorithm. Summing up, for both BABD and ABD systems the most efficient direct algorithms are based on *divide-and-conquer* approaches. To the best of our knowledge, none of these methods has been tailored for GPU platforms, which have thousands of small cores. In order to attain high performance on many-cores platforms such, we propose substantial modifications.

The parallel algorithms reviewed in [4] can be implemented efficiently to exploit medium granularity parallelism, that is the number of processors is much smaller than the number N of block rows (or columns) of the matrix. The same matrix decomposition is locally applied to properly chosen submatrices, leading to subproblems of smaller size and preserving the ABD/BABD block structure. In the following subsections we present the main features of one of these solvers, namely SOF, for BABD systems. Then, in the following section, we propose a novel version of SOF algorithm inspired by parallel cyclic reduction (PARACR) [92] and designed to get more parallelism and less sequential steps on many-core architectures. Such a technique can be straightforwardly extended to parallel algorithms for ABD systems.

5.1.1 Structured Orthogonal Factorization

Consider a generic BABD system

$$\begin{pmatrix} S_0 & T_0 & & & \\ & S_1 & T_1 & & \\ & & \ddots & \ddots & \\ & & & S_{N-1} & T_{N-1} \\ B_a & & & & B_b \end{pmatrix} \begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{N-1} \\ \mathbf{x}_N \end{pmatrix} = \begin{pmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_{N-1} \\ \mathbf{b}_N \end{pmatrix}, \tag{5.6}$$

where the matrix has the same structure as in (5.1), the right-hand side and the solution satisfy $\mathbf{b}_i, \mathbf{x}_i \in \mathbb{R}^n$, for $i = 0, \dots, N$. Let us consider a partition of the block rows of the system above into P *slices*, leaving out of the last block row related to boundary conditions. Such partition can be described by an integer $1 \le P \le N/2$, and a set of separator indices

$$0 = k_0 < k_1 < \cdots < k_P = N, \quad k_{p+1} - k_p \ge 2, \; p = 0, \dots, P-1.$$

In the *Structured Orthogonal Factorization* (SOF), proposed in [146] and reviewed in [4], QR decomposition is locally applied in order to independently factorize each submatrix, corresponding to a slice. We detail the process for the p-th slice with $r_p = k_{p+1} - k_p$ block rows, augmented with its right-hand side, that becomes

$$\left(\begin{array}{ccccc|c} S_{k_p} & T_{k_p} & & & & \mathbf{b}_{k_p} \\ & S_{k_p+1} & T_{k_p+1} & & & \mathbf{b}_{k_p+1} \\ & & \ddots & \ddots & & \vdots \\ & & & S_{k_{p+1}-1} & T_{k_{p+1}-1} & \mathbf{b}_{k_{p+1}-1} \end{array}\right). \tag{5.7}$$

First find an orthogonal matrix $Q_{k_p} \in \mathbb{R}^{2n\times 2n}$ such that

$$\begin{pmatrix} T_{k_p} \\ S_{k_p+1} \end{pmatrix} = Q_{k_p} \begin{pmatrix} U_{k_p} \\ 0 \end{pmatrix}, \tag{5.8}$$

where $U_{k_p} \in \mathbb{R}^{n\times n}$ is upper triangular. If we apply the matrix $Q_{k_p}^T$ to the first two block rows of (5.7), that is we compute

$$\begin{pmatrix} Q_{k_p}^T & \\ & I_{(r_p-2)n} \end{pmatrix} \left(\begin{array}{cccccc|c} S_{k_p} & T_{k_p} & & & & & \mathbf{b}_{k_p} \\ & S_{k_p+1} & T_{k_p+1} & & & & \mathbf{b}_{k_p+1} \\ & & S_{k_p+2} & T_{k_p+2} & & & \mathbf{b}_{k_p+1} \\ & & & \ddots & \ddots & & \vdots \\ & & & & S_{k_{p+1}-1} & T_{k_{p+1}-1} & \mathbf{b}_{k_{p+1}-1} \end{array}\right) \tag{5.9}$$

we obtain

$$\left(\begin{array}{cccccc|c} V_{k_p} & U_{k_p} & W_{k_p} & & & & \mathbf{f}_{k_p} \\ \overline{V}_{k_p+1} & & \overline{W}_{k_p+1} & & & & \overline{\mathbf{f}}_{k_p+1} \\ & & S_{k_p+2} & T_{k_p+2} & & & \mathbf{b}_{k_p+2} \\ & & & \ddots & \ddots & & \vdots \\ & & & & S_{k_{p+1}-1} & T_{k_{p+1}-1} & \mathbf{b}_{k_{p+1}-1} \end{array}\right). \tag{5.10}$$

Then find an orthogonal matrix $Q_{k_p+1} \in \mathbb{R}^{2n\times 2n}$ such that

$$\begin{pmatrix} \overline{W}_{k_p+1} \\ S_{k_p+2} \end{pmatrix} = Q_{k_p+1} \begin{pmatrix} U_{k_p+1} \\ 0 \end{pmatrix},$$

where $U_{k_p+1} \in \mathbb{R}^{n\times n}$ is upper triangular, and apply $Q_{k_p+1}^T$ to block rows $1, 2$ of (5.10). This process can be repeated sequentially for $r_p - 1$ steps, until we obtain an equivalent system

$$\begin{pmatrix} V_{k_p} & U_{k_p} & W_{k_p} & & & \\ V_{k_p+1} & & U_{k_p+1} & W_{k_p+1} & & \\ \vdots & & & \ddots & \ddots & \\ V_{k_{p+1}-2} & & & & U_{k_{p+1}-2} & W_{k_{p+1}-2} \\ S'_p & & & & & T'_p \end{pmatrix} \begin{pmatrix} \mathbf{x}_{k_p} \\ \mathbf{x}_{k_p+1} \\ \vdots \\ \mathbf{x}_{k_{p+1}-1} \\ \mathbf{x}_{k_{p+1}} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{k_p} \\ \mathbf{f}_{k_p+1} \\ \vdots \\ \mathbf{f}_{k_{p+1}-1} \\ \mathbf{b}'_p \end{pmatrix}, \tag{5.11}$$

Notice that the system (5.7) has been split as

$$S'_p\mathbf{x}_{k_p} + T'_p\mathbf{x}_{k_{p+1}} = \mathbf{b}'_p \tag{5.12}$$

$$U_{i-1}\mathbf{x}_i + V_{i-1}\mathbf{x}_{k_p} + W_{i-1}\mathbf{x}_{i+1} = \mathbf{f}_{i-1}, \quad i = k_{p+1}-1,\dots,k_p+1. \tag{5.13}$$

Notice that, after solving equations (5.12), the remaining variables can be recovered by back-substitution by solving the triangular systems (5.13) (U_i blocks are upper-triangular). Putting together equations (5.12), a reduced system can be formed

$$\begin{pmatrix} S'_0 & T'_0 & & & \\ & S'_1 & T'_1 & & \\ & & \ddots & \ddots & \\ & & & S'_{N'-1} & T'_{N'-1} \\ B_a & & & & B_b \end{pmatrix} \begin{pmatrix} \mathbf{x}_{k_0} \\ \mathbf{x}_{k_1} \\ \vdots \\ \mathbf{x}_{k_{N'-1}} \\ \mathbf{x}_{k'_N} \end{pmatrix} = \begin{pmatrix} \mathbf{b}'_0 \\ \mathbf{b}'_1 \\ \vdots \\ \mathbf{b}'_{N'-1} \\ \mathbf{b}_N \end{pmatrix}, \tag{5.14}$$

which is a BABD system with only $N' = P$ internal block rows. This immediately suggests that it may be possible to apply the whole process recursively for a certain number $L > 1$ of levels, until a sufficiently small system is achieved and then directly solved. The missing variables can then be recovered by L levels of back-substitution, leading to a total number of $2L+1$ sequential steps.

From now on we assume, like in [4], that each slice is assigned to a processor, in such a way that all slices can be reduced in parallel. However, the p-th slice needs $r_p - 1$ sequential QR decompositions. The following Proposition states the structure of the corresponding matrix decomposition.

Proposition 1. *Consider the BABD system* (5.6)*, and define* Π, Γ *as the permutation matrices of order* $n(N+1)$ *corresponding to the following block reorderings*

$$\Pi\left((\mathbf{x}_0^T,\dots,\mathbf{x}_N^T)^T\right) = (\mathbf{x}_{k_0}^T,\dots,\mathbf{x}_{k_1-2}^T,\mathbf{x}_{k_1}^T,\dots,\mathbf{x}_{k_2-2}^T,\mathbf{x}_{k_2+1}^T,\dots,\mathbf{x}_{k_P-2}^T,\mathbf{x}_{k_1-1}^T,\dots,\mathbf{x}_{k_P-1}^T,\mathbf{x}_{k_P}^T)^T,$$
$$\Gamma\left((\mathbf{x}_0^T,\dots,\mathbf{x}_N^T)^T\right) = (\mathbf{x}_{k_0+1}^T,\dots,\mathbf{x}_{k_1-1}^T,\mathbf{x}_{k_1+1}^T,\dots,\mathbf{x}_{k_2-1}^T,\mathbf{x}_{k_2+1}^T,\dots,\mathbf{x}_{k_P-1}^T,\mathbf{x}_{k_0}^T,\dots,\mathbf{x}_{k_P}^T)^T, \tag{5.15}$$

for any vector $\mathbf{x} = (\mathbf{x}_0^T,\dots,\mathbf{x}_N^T)^T \in \mathbb{R}^{n(N+1)}$*. The SOF algorithm yields the following decomposition*

$$A = \mathcal{Q}\Pi^T D\Gamma, \tag{5.16}$$

where D *is a block upper triangular matrix with, where the last diagonal block has a BABD structure, while* $\mathcal{Q}$ *in an* $n(N+1)\times n(N+1)$ *orthogonal matrix, which is the (commutative) product of* P *orthogonal matrices*

$$\mathcal{Q} = \mathcal{Q}_0 \cdots \mathcal{Q}_{P-1}, \tag{5.17}$$

where $\mathcal{Q}_p$*,* $p = 0,\dots,P-1$*, is given by*

$$\mathcal{Q}_p = \begin{pmatrix} I_{(k-2)n} & & \\ & Q_p & \\ & & I_{(k-2)n} \end{pmatrix},$$

and Q_p is given (non-commutative) product of $r_p - 1$ orthogonal matrices of size $r_p n \times r_p n$

$$Q_p = \begin{pmatrix} I_{(r_p-2)n} & \\ & Q^T_{k_{p+1}-2} \end{pmatrix} \cdots \begin{pmatrix} I_n & & \\ & Q^T_{k_p+1} & \\ & & I_{(r_p-3)n} \end{pmatrix} \begin{pmatrix} Q^T_{k_p} & \\ & I_{(r_p-2)n} \end{pmatrix}. \tag{5.18}$$

Proof. Looking at equation (5.9) and considering the following discussion, it is clear that we apply an orthogonal transformation in the form (5.18) on each slice in order to obtain the reduced slice (5.11). The product (5.18) is clearly non-commutative due to the sequentiality of the process. The product (5.17) of the "enlarged" Q_p matrices to the size $Nn \times Nn$ is commutative since the slices are non-overlapping.

By stacking together the reduced slices (5.11), for $0 \le p \le P-1$, we obtain the following equivalent system

$$\mathcal{Q}^T[A|\mathbf{b}] = \left(\begin{array}{ccccc|c} V_0 & U_0 & W_0 & & & \mathbf{f}_0 \\ \vdots & \ddots & & & & \vdots \\ S'_0 & & T'_0 & & & \mathbf{b}'_0 \\ \hline & & \ddots & & & \vdots \\ \hline & & V_{k_{P-1}} & U_{k_{P-1}} & W_{k_{P-1}} & \mathbf{f}_{k_{P-1}} \\ & & \vdots & \ddots & & \vdots \\ & & S'_{P-1} & & T'_{P-1} & \mathbf{b}'_{N'-1} \\ \hline B_a & & & & B_b & \mathbf{b}_N \end{array}\right). \tag{5.19}$$

Applying the permutation matrix Π to matrix $\mathcal{Q}^T A$ from the left, we move all equations in the form (5.12) to the bottom, obtaining

$$\left(\begin{array}{cccccccc|c} V_0 & U_0 & W_0 & & & & & & \mathbf{f}_0 \\ \vdots & \ddots & & & & & & & \vdots \\ V_{k_1-2} & & U_{k_1-2} & W_{k_1-2} & & & & & \mathbf{f}_{k_1-2} \\ \hline & & & \ddots & & & & & \vdots \\ \hline & & & & V_{k_{P-1}} & U_{k_{P-1}} & W_{k_{P-1}} & & \mathbf{f}_{k_{P-1}} \\ & & & & \vdots & \ddots & & & \vdots \\ & & & & V_{k_P-2} & & U_{k_P-2} & & \mathbf{f}_{k_P-2} \\ \hline S'_0 & & T'_0 & & & & & & \mathbf{b}'_0 \\ & & & \ddots & & & & & \vdots \\ & & & & S'_{P-1} & & T'_{P-1} & & \mathbf{b}'_{N'-1} \\ B_a & & & & & & B_b & & \mathbf{b}_N \end{array}\right),$$

and finally, by applying the permutation matrix Γ^T to matrix $\mathcal{Q}^T A$ from the right, we move to the left the columns corresponding to the reduced system, and the matrix $\left(\begin{array}{cc|c} U & V & \mathbf{f} \\ & A' & \mathbf{b}' \end{array}\right)$ can be block partitioned as follows

$$\left(\begin{array}{cccc|cccc|c}
U_0 & W_0 & & & V_0 & & & & \mathbf{f}_0 \\
& \ddots & & & \vdots & & & & \vdots \\
& U_{k_1-2} & & & V_{k_1-2} & & W_{k_1-2} & & \mathbf{f}_{k_1-2} \\
& & \ddots & & & \ddots & & & \vdots \\
& & U_{k_{P-1}} & W_{k_{P-1}} & & & V_{k_{P-1}} & & \mathbf{f}_{k_{P-1}} \\
& & & \ddots & & & \vdots & & \vdots \\
& & & U_{k_P-2} & & & V_{k_P-2} & W_{k_P-2} & \mathbf{f}_{k_P-2} \\
\hline
& & & & S'_0 & T'_0 & & & \mathbf{b}'_0 \\
& & & & & \ddots & & & \vdots \\
& & & & & & S'_{P-1} & T'_{P-1} & \mathbf{b}'_{N'-1} \\
& & & & B_a & & & B_b & \mathbf{b}_N
\end{array}\right),$$

where U is an upper triangular matrix with a block diagonal structure, and A' is the reduced BABD matrix (5.14). The matrix above is the matrix D we where looking for, and this proves relation (5.16). $\square$

5.1.2 Data dependency analysis of SOF

For the SOF algorithm, presented in the previous subsection, the number of levels L and the number of slices P can be tuned according to the resources of the physical machine, making this algorithm particularly suited to be implemented on very different hardware environments: a natural choice is to take P equal to the number of available "processors", that can be meant as physical processors or parallel threads, depending on the computing system adopted. There is an obvious trade-off between the choice of L and P: if we choose a small number P of slices (and processors), it will turn into a little fraction of parallel work during the factorization, leading however to a small reduced system which can be solved with fewer sequential recursion steps L; on the other hand, if we choose a large number P of slices, the fraction of parallel work will be higher, but the resulting reduced system will require a longer sequence of further reduction steps to be solved.

Wright [146] proposed three variations of SOF algorithm: a sequential version, in which $P = 1$ and $L = 1$, so that there's only one slice to process; a two-level version, in which $P > 1$ and $L = 1$, so that P processors are used for blocks factorization, reduction and back substitution, but the reduced system is solved sequentially; a "cyclic reduction" version: suppose for the moment that $N = P = 2^L$, then the factorization and reduction phase can be performed on slices of two block rows each for $L = \log_2(P) + 1$ levels, until a 2×2 block system is achieved, then the back-substitution is performed for L steps. The author stated this last version is optimal with regard to complexity: the execution time is $\mathcal{O}(2 \log_2 P)$ on P processors, while the serial time is $\mathcal{O}(2P \log_2 P)$. However, at each level of the reduction phase, only half the processors active at the previous level is needed, with an overall tremendous waste of resources, especially for massively parallel architectures. If $N > P$, then the BABD matrix A is partitioned into P slices of about N/P block rows each, and the factorization and reduction phase is applied to obtain a reduced system with $N' + 1 = P + 1$ block rows. The cyclic reduction version of SOF is then used to solve the reduced system, and, finally, the missing unknowns are retrieved by back substitution. From now on we will only refer to this last version of SOF, whose pseudocode is presented in Algorithm 7.

Algorithm 7 Structured Orthogonal Factorization (SOF)

```
    Inputs: $A, \mathbf{b}, P$
    Outputs: $\mathbf{x}$
 1: if $N > P$ then
 2:     Divide $A$ into $P$ slices of roughly $\frac{N}{P}$ block rows each
 3:     Reduce the $p$-th slice to one block equation on the $p$-th processor, $p = 0, \dots, P$
 4: end if
 5: $L = \log_2(P) + 1$
 6: for $l = 1, \dots, L$ do
 7:     Assign block equations $2p, 2(p+1)$ to $p$-th processor, $p = 0, \dots, N/2^l - 1$
 8:     Reduce the $p$-th pair to one block equation on the $p$-th processor, $p = 0, \dots, N/2^l - 1$
 9: end for
10: Solve a $2 \times 2$ block system
11: for $l = 1, \dots, L$ do
12:     Retrieve block unknown $p\frac{N}{2^{l-1}} + \frac{N}{2^l}$ of the reduced system with $p$-th processor, $p = 0, \dots, 2^{l-1}$,
13: end for
14: if $N > P$ then
15:     for $k = \frac{N}{P} - 1, \dots, 1$ do
16:         Retrieve block unknown $p\frac{N}{P} + k$ with the $p$-th processor, $p = 0, \dots, P$
17:     end for
18: end if
```

Figure 5.1 shows the communication pattern for SOF algorithm in the case $N = 8$. Each block equation

$$S_i \mathbf{x}_i + T_{i+1} \mathbf{x}_{i+1} = \mathbf{b}_i,$$

is labeled by e_i, $i = 0, \dots, N-1$, while the block equations related to boundary conditions $B_a \mathbf{x}_0 + B_b \mathbf{x}_N = \mathbf{b}_N$ is labeled by e_N. Letters e', e'', e''' stand for updated equation at level $L = 1, 2, 3$ respectively. The workflow highlights an inside feature of SOF: during the forward reduction phase, the parallelism available is halved at each level, forming a serial bottleneck. Although these

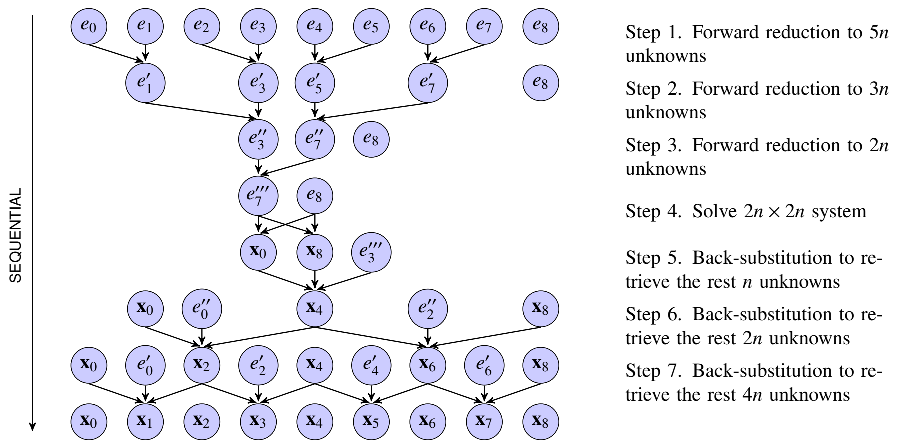

SEQUENTIAL
Step 1. Forward reduction to 5n unknowns
Step 2. Forward reduction to 3n unknowns
Step 3. Forward reduction to 2n unknowns
Step 4. Solve 2n × 2n system
Step 5. Back-substitution to retrieve the rest n unknowns
Step 6. Back-substitution to retrieve the rest 2n unknowns
Step 7. Back-substitution to retrieve the rest 4n unknowns

Figure 5.1: Data dependency DAG for SOF in the case $N = 8$ ($9n$ unknowns) with $P = 4$ slices (and processors) of $k = 2$ block rows each, showing the dataflow between each block equation.

algorithms enjoy a medium-grained parallel structure, since the same procedure - the QR decomposition and the back-substitution - must be applied to several blocks concurrently, the amount of parallel work is not constant among the levels, and it is particularly poor at the end of the forward phase and at the beginning of the back-substitution phase. These algorithms clearly suffer from irregular work distribution within each level, so that some processor is forced to stay idle in the most part of the algorithm levels. A straightforward implementation would lead to an amount of parallelism too poor to fully exploit the computing power of many-core architectures such as the GPUs. For this reason we propose and discuss in the following section a different scheme of divide-et-impera, more suited for massive parallel architectures, where P is significantly large.

5.2 Solving BABD systems on GPUs

In this section we propose a different scheme of divide-et-impera that gives a high fraction of parallelism at each algorithmic step. At each level, the method doubles the number of subproblems of half the size by separating odd from even unknowns, until at the last level of reduction the solution for all variables is found in parallel and the back-substitution phase is not required. As we will see in the experiments, the massive parallelism of this algorithm outperforms the existing algorithms, in spite of a considerable arithmetic redundancy. In the literature, a similar approach has been first exploited in PARACR [92], a parallel algorithm for solving tridiagonal systems that is not directly applicable to ABD and even less to BABD systems. Following the material at Section 5.1.1, in this section we present a direct massively parallel algorithm for BABD systems. Such a technique can be extended to ABD systems, with some minor modifications. Section 5.3 will be devoted to numerical tests and computation times assessment.

Suppose $N+1$, i.e. the total number of block rows (or columns) of system (5.6), is even. Then the system can be partitioned into $P=(N+1)/2$ slices with two block rows each, that is the maximum parallelism attainable in SOF. Let us point out that we include in the partition also the last block row, which is relative to the boundary conditions, that in SOF was excluded. This is necessary, together with the choice of slices of length two, in order to fully decouple even and odd variables, in such a way that all variables can be recovered at the end of the forward phase without the need of back-substitution. For this reason, we refer to this method as "odd/even SOF". Let us consider two different staggered partitions into slices of the system (5.6) and apply SOF to each of them. For the first partition, called *even partition* from now on, we choose $P=(N+1)/2$ and the following set of separator indices

$$k_p^e = 2p, \quad p = 0,\dots,P-1 = (N+1)/2-1.$$

From (5.9) and (5.12), since the dimension of the slices is fixed to $r_p = k_{p+1}^e - k_p^e = 2$ for all p, we get

$$\begin{aligned}
&\mathcal{Q}_e^T \left(\begin{array}{cccccc|c}
S_0 & T_0 & & & & & \mathbf{b}_0\\
 & S_1 & T_1 & & & & \mathbf{b}_1\\ \hline
 & & & \ddots & & & \vdots\\ \hline
 & & & & S_{N-1} & T_{N-1} & \mathbf{b}_{N-1}\\
B_a & & & & & B_b & \mathbf{b}_N
\end{array}\right)\\
&= \left(\begin{array}{cccccc|c}
V_0^e & U_0^e & W_0^e & & & & \mathbf{f}_0^e\\
S_0^e & & T_0^e & & & & \mathbf{b}_0^e\\ \hline
 & & & \ddots & & & \vdots\\ \hline
W_{N-1}^e & & & & V_{N-1}^e & U_{N-1}^e & \mathbf{f}_{N-1}^e\\
B_a^e & & & & B_b^e & & \mathbf{b}_{N'}^e
\end{array}\right),
\end{aligned} \tag{5.20}$$

where $\mathcal{Q}_e \in R^{(N+1)n\times(N+1)n}$ is orthogonal and the reduced system, that shows a BABD structure with $N' = P-1$ internal blocks, involves only even block unknowns $\mathbf{x}_0, \mathbf{x}_2, \dots, \mathbf{x}_{N-1}$. The corresponding reduced system can be achieved, as in the SOF case, by selecting the last block row of each updated slice, i.e. block rows with odd index, and block columns with even index; for each slice, a QR decomposition of type (5.8) is performed, and the reduced right-hand side thus obtained is $(\mathbf{b}_0^{eT}, \dots, \mathbf{b}_{N'}^{e\,T})^T$. For the second partition, namely the *odd partition*, we choose again $P=(N+1)/2$, and the following set of separator indices

$$k_p^o = 2p+1, \quad p = 0,\dots,P-1 = (N+1)/2-1.$$

We consider to join the first and the last block rows into the same slice, thus obtaining

$$
\begin{aligned}
&\mathcal{Q}_o^T \left(\begin{array}{ccccc|c}
B_a & & & & B_b & \mathbf{b}_N \\
S_0 & T_0 & & & & \mathbf{b}_0 \\
\hline
& & \ddots & & & \vdots \\
\hline
& & S_{N-2} & T_{N-2} & & \mathbf{b}_{N-2} \\
& & & S_{N-1} & T_{N-1} & \mathbf{b}_{N-1}
\end{array}\right) \\
&= \left(\begin{array}{ccccc|c}
U_N^o & W_N^o & & & V_N^o & \mathbf{f}_N^o \\
& B_a^o & & & B_b^o & \mathbf{b}_{N'}^o \\
\hline
& & \ddots & & & \vdots \\
\hline
& & W_{N-2}^o & U_{N-2}^o & V_{N-2}^o & \mathbf{f}_0^o \\
& & T_{N'-1}^o & & S_{N'-1}^o & \mathbf{b}_{N'-1}^o
\end{array}\right),
\end{aligned}
\tag{5.21}
$$

where $\mathcal{Q}_o \in R^{(N+1)n\times(N+1)n}$ is orthogonal and the reduced system, that shows a BABD structure with $N' = P$ internal blocks as in the even case, involves only odd unknowns $\mathbf{x}_1, \mathbf{x}_3, \ldots, \mathbf{x}_N$. The corresponding reduced system consists of block rows and columns with odd index, provided that the first block row has been moved as above; in this case, the reduced right-hand side is $(\mathbf{b}_0^{oT}, \ldots, \mathbf{b}_{N'}^{o\,T})^T$.

Notice that these two decompositions, even and odd, are completely independent and can therefore be performed in parallel, provided that $N+1$ processors are available. The resulting reduced systems are BABD systems of half the size

$$
\begin{aligned}
&\begin{pmatrix}
S_0^e & T_0^e & & & \\
& S_1^e & T_1^e & & \\
& & \ddots & \ddots & \\
& & & S_{N'-1}^e & T_{N'-1}^e \\
B_a^e & & & & B_b^e
\end{pmatrix}
\begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{N-3} \\ \mathbf{x}_{N-1} \end{pmatrix}
=
\begin{pmatrix} \mathbf{b}_0^e \\ \mathbf{b}_1^e \\ \vdots \\ \mathbf{b}_{N'-1}^e \\ \mathbf{b}_{N'}^e \end{pmatrix}, \\
&\begin{pmatrix}
S_0^o & T_0^o & & & \\
& S_1^o & T_1^o & & \\
& & \ddots & \ddots & \\
& & & S_{N'-1}^o & T_{N'-1}^o \\
B_a^o & & & & B_b^o
\end{pmatrix}
\begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_3 \\ \vdots \\ \mathbf{x}_{N-2} \\ \mathbf{x}_N \end{pmatrix}
=
\begin{pmatrix} \mathbf{b}_0^o \\ \mathbf{b}_1^o \\ \vdots \\ \mathbf{b}_{N'-1}^o \\ \mathbf{b}_{N'}^o \end{pmatrix}.
\end{aligned}
\tag{5.22}
$$

If $N+1 = 2^L$ for some $L \in \mathbb{N} \setminus \{0\}$, we can apply recursively this technique to the reduced systems, and, after $L-1$ steps, we end up with exactly $(N+1)/2$ block systems of size 2×2 in the unknowns $\mathbf{x}_i, \mathbf{x}_{(N+1)/2+i}$, with $i = 0, \ldots, (N-1)/2$, which can be solved directly. The odd/even SOF clearly has the same stability properties as SOF. The following Proposition clarifies the structure of the matrix decomposition we obtain.

Proposition 2. *Consider the BABD system* (5.6). *Let $N+1$ be an even number and define $\Pi \in \mathbb{R}^{n(N+1)\times n(N+1)}$ as the permutation matrix corresponding to the following block reordering*

$$
\Pi\left((\mathbf{x}_0^T, \ldots, \mathbf{x}_N^T)^T\right) = (\mathbf{x}_0^T, \mathbf{x}_2^T, \ldots, \mathbf{x}_{N-2}^T, \mathbf{x}_1^T, \mathbf{x}_3^T, \ldots, \mathbf{x}_{N-1}^T)^T, \tag{5.23}
$$

for any vector $\mathbf{x} = (\mathbf{x}_0^T, \dots, \mathbf{x}_N^T)^T \in \mathbb{R}^{n(N+1)}$*. The odd/even SOF algorithm yields the following decomposition*

$$A = M^{-1}\Pi^T D \Pi \ , \tag{5.24}$$

where M is an invertible BABD matrix, D is block diagonal with two blocks of equal size, each of which has itself a BABD structure.

Proof. Consider $\mathcal{Q}_e^T$, $\mathcal{Q}_o^T$ as in equations (5.20), (5.21) respectively. These matrices are block diagonal matrices

$$\mathcal{Q}_e^T = \operatorname{diag}\left(Q_0^{e^T}, \dots, Q_{N'}^{e^T}\right), \quad \mathcal{Q}_o^T = \operatorname{diag}\left(Q_0^{o^T}, \dots, Q_{N'}^{o^T}\right), \tag{5.25}$$

where $Q_i^{e,o}$ are $2n \times 2n$ matrices arising from local orthogonal factorizations of type (5.8). Rewrite Q_i^{e,o^T} as a 2×2 block matrix as follows

$$Q_i^{e,o^T} = \begin{pmatrix} Q_{i_{11}}^{e,o} & Q_{i_{12}}^{e,o} \\ Q_{i_{21}}^{e,o} & Q_{i_{22}}^{e,o} \end{pmatrix},$$

and define

$$M = \begin{pmatrix} Q_{0_{21}}^o & Q_{0_{22}}^o & & & & & & \\ & Q_{1_{21}}^e & Q_{1_{22}}^e & & & & & \\ & & Q_{1_{21}}^o & Q_{1_{22}}^o & & & & \\ & & & \ddots & \ddots & & & \\ & & & & Q_{N'_{21}}^e & Q_{N'_{22}}^e & \\ & & & & & Q_{N'_{21}}^o & Q_{N'_{22}}^o \\ Q_{0_{22}}^e & & & & & & Q_{0_{12}}^e \end{pmatrix}. \tag{5.26}$$

Then we have

$$MA = \begin{pmatrix} S_0^o & 0 & T_0^o & & & & \\ & S_0^e & 0 & T_0^e & & & \\ & & S_1^o & 0 & T_1^e & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & S_{N'-1}^e & 0 & T_{N'-1}^e \\ B_a^o & & & & & B_b^o & 0 \\ 0 & B_a^e & & & & & B_b^e \end{pmatrix}. \tag{5.27}$$

and

$$\Pi M A \Pi^T = \left(\begin{array}{cccccc|cccccc}
S_0^o & T_0^o & & & & & & & & & & \\
& S_1^o & T_1^o & & & & & & & & & \\
& & \ddots & \ddots & & & & & & & & \\
& & & S_{N'-1}^o & T_{N'-1}^o & & & & & & & \\
B_a^o & & & & B_b^o & & & & & & & \\
\hline
& & & & & & S_0^e & T_0^e & & & & \\
& & & & & & & S_1^e & T_1^e & & & \\
& & & & & & & & \ddots & \ddots & & \\
& & & & & & & & & S_{N'-1}^e & T_{N'-1}^e & \\
& & & & & & B_a^e & & & & B_b^e &
\end{array}\right),$$

which is the matrix D we were looking for. □

5.2.1 Data dependency analysis

Algorithm 8 Odd/Even Structured Orthogonal Factorization

```
   Inputs: $A, \mathbf{b}$
   Outputs: $\mathbf{x}$
1: $L = \log_2(P) + 1$
2: for $l = 1, \dots, L$ do
3:     Assign block equations $(p - 2^{l-1}, p) \bmod N$ to $p$-th processor, $p = 0, \dots, N$
4:     Reduce the $p$-th pair to one block equation on the $p$-th processor, $p = 0, \dots, N$
5: end for
6: Solve $\frac{N}{2}$ block systems of size $2 \times 2$
```

Figure 5.2 shows the communication pattern of the odd/even SOF algorithm in the case $N = 7$. Each block equation

$$S_i \mathbf{x}_i + T_{i+1} \mathbf{x}_{i+1} = \mathbf{b}_i,$$

is labeled by e_i, $i = 0, \dots, N-1$, while the block equations related to boundary conditions $B_a \mathbf{x}_0 + B_b \mathbf{x}_N = \mathbf{b}_N$ is labeled by e_N. Here, letters e^a stand for updated equations at level $L = 1$ and $a \in \{e, o\}$ indicates if the corresponding equation is part of a system in even or odd unknowns; letters e^{ab} stand for updated equation at level $L = 2$ with $b \in \{e, o\}$ having the same meaning as before, while a refers to the subsystem at the previous level.

Each block equation is updated at each step, so that we can assign an equation to a processor and thus avoid the problem of idle processors, at the cost of arithmetic redundancy. Moreover, this scheme also produces an halved number

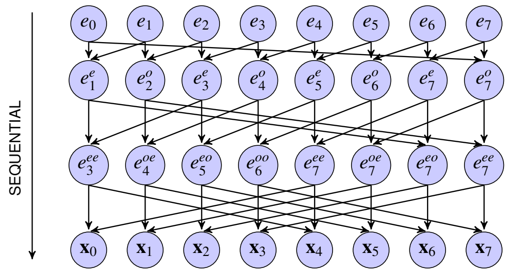

SEQUENTIAL
Step 1. Forward reduction to 2 4n unknowns systems
Step 2. Forward reduction to 4 2n unknowns systems
Step 3. Solve 4 2n unknowns systems

Figure 5.2: Data dependency DAG for odd/even SOF in the case $N = 7$ ($8n$ unknowns) with $P = 8$ slices (and processors) of $k = 2$ block rows each, showing the dataflow between each block equation.

L of total sequential steps, since back-substitution can be avoided. A straightforward implementation on a GPU should lead to high values of *achieved occupancy*, that is the fraction of active computing units over the upper limit, which is an important indicator of potential efficiency of a massively parallel algorithm.

The intrinsic difficulty of achieving massive parallelism of GPUs with data workflows similar to that of SOF (Figure 5.1) has already is also discussed in Chapter 6, where the solution of large size sparse triangular systems in considered. As it is shown, in such a case it is not possible to reorganize the DAG because of numerical issues related to preconditioning.

5.2.2 Parallel structured orthogonal factorization

In a real application we often have $N \gg P$ even in a massively parallel architecture, thus the workload is serialized in chunks by the scheduler, as we will detail in Section 5.2.4 and Section 5.3. Here we propose an algorithm, which we call PARAllel Structured Orthogonal Factorization (PARASOF), that improves SOF by switching to its even/odd version to reduce inefficient steps when there is not enough parallelism to keep a GPU busy. The resulting procedure is the following: first let us apply the forward reduction phase of SOF to obtain an $N_r \times N_r$ block system, then solve this reduced intermediate system with the odd/even SOF algorithm. Finally, the backward substitution phase of SOF is used to retrieve the missing unknowns.

Algorithm 9 PARAllel Structured Orthogonal Factorization (PARASOF)

```
    Inputs: $A, \mathbf{b}, P$
    Outputs: $\mathbf{x}$
 1: if $N > P$ then
 2:     Divide $A$ into $P$ slices of roughly $\frac{N}{P}$ block rows each
 3:     Reduce the $p$-th slice to one block equation on the $p$-th processor, $p = 0, \ldots, P$
 4: end if
 5: $L = \log_2(P) + 1$
 6: for $l = 1, \ldots, L$ do
 7:     Assign block equations $(p - 2^{l-1}, p) \mod N$ to $p$-th processor, $p = 0, \ldots, N_r$
 8:     Reduce the $p$-th pair to one block equation on the $p$-th processor, $p = 0, \ldots, N_r$
 9: end for
10: Retrieve the solution of the reduced system by solving $\frac{N}{2}$ block systems of size $2 \times 2$
11: if $N > P$ then
12:     for $k = \frac{N}{P} - 1, \ldots, 1$ do
13:         Retrieve block unknown $p\frac{N}{P} + k$ with the $p$-th processor, $p = 0, \ldots, P$
14:     end for
15: end if
```

Figure 5.3 shows PARASOF workflow schematically. In this example, perform one forward reduction on slices of $k = 3$ block rows each to reach a $4n$ unknowns system, we then switch to even/odd SOF that enable us to finish the inefficient middle steps more quickly, because they have fewer algorithmic steps than SOF.

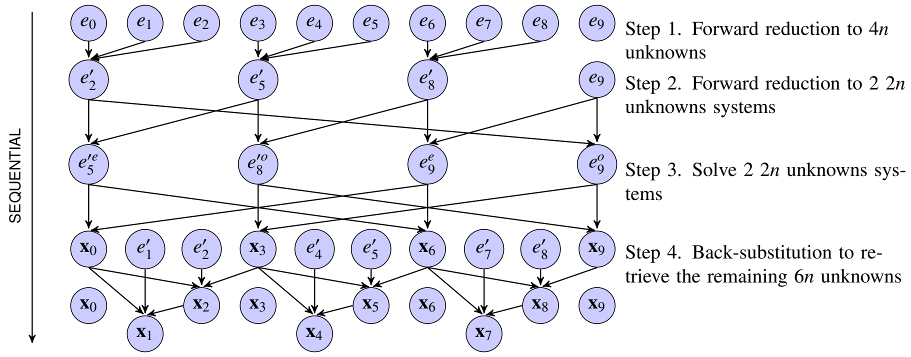

SEQUENTIAL
e_0 e_1 e_2 e_3 e_4 e_5 e_6 e_7 e_8 e_9
e'_2 e'_5 e'_8 e_9
e'^e_5 e'^o_8 e^e_9 e^o_9
$\mathbf{x}_0$ e'_1 e'_2 $\mathbf{x}_3$ e'_4 e'_5 $\mathbf{x}_6$ e'_7 e'_8 $\mathbf{x}_9$
$\mathbf{x}_0$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_5$ $\mathbf{x}_6$ $\mathbf{x}_8$ $\mathbf{x}_9$
$\mathbf{x}_1$ $\mathbf{x}_4$ $\mathbf{x}_7$
Step 1. Forward reduction to $4n$ unknowns
Step 2. Forward reduction to 2 $2n$ unknowns systems
Step 3. Solve 2 $2n$ unknowns systems
Step 4. Back-substitution to retrieve the remaining $6n$ unknowns

Figure 5.3: Data dependency DAG for PARASOF in the case $N = 8$ ($9n$ unknowns) with $P = 4$ slices (and processors).

5.2.3 Solving a sequence of BABD systems

In applications, it is often required to solve not a unique BABD system, but a sequence of them. There are in general two possible cases: in the first one, the matrix and the right-hand-side change at each step of the sequence; in the second, only the right-hand-side changes. The first one happens e.g. in optimal control problems [16]. In general, here it is necessary to factorize the matrix from scratch, and we show in the numerical section that PARASOF obtains a good speed-up in the factorization phase, justifying its adoption. In the second case, one can use the previous factorization to solve a linear system with multiple right-hand-sides (or a sequence of linear systems with the same matrix).

Let us detail how to derive a factorization from PARASOF algorithm in order to cover the latter scenario. In order to do that, one can follow Proposition 1 and Proposition 2: the first step of SOF forward reduction gives an orthogonal matrix, i.e. $\mathcal{Q}$ (5.17), to be stored, that can be subsequently applied to any new right-hand side $\mathbf{b}$ in order to obtain a reduced vector $\mathbf{b}'$, which will be then used as a right-hand side in the following algorithmic step where even/odd SOF procedure is used. Each sequential step of the even/odd SOF algorithm gives an nonsingular matrix which has a BABD structure up to a permutation, i.e. M (5.26), for a total of L_r total matrices, namely $M_{L_r-1}, \dots, M_0$, which have to be applied to the reduced right-hand-side from the left in order to update the reduced vector $\mathbf{b}'$, that can be used to partially retrieve the solution by solving the block diagonal system achieved by the even/odd SOF procedure. Then, the solution is fully recovered by the back-substitution phase.

The updated right-hand side in the first step of forward reduction is given by $\mathcal{Q}\mathbf{b}$. Notice that the matrix $\mathcal{Q}$ is typically a large dense orthogonal matrix, but expression (5.17) suggests that we can instead store the $r_p - 1$ sparse factors $Q_{k_p}, \dots, Q_{k_p-2}$ for each slice, and then apply such matrices in a sequential fashion on each slice. Again, all slices can be processed in parallel. Similarly, in the even/odd SOF procedure the updated right-hand side has the form $M_{L_r-1} \cdots M_0 \mathbf{b}'$, where the product $M_{L_r-1} \cdots M_0$ gives a large dense nonsingular matrix whose factors are again quite sparse. Therefore, from the storage point of view it is more convenient to perform these matrix vector products one at a time, with an evident drawback, that is the important loss in the attainable parallelism. Indeed, the numerical experiments of Section 5.3.2 will show that in this case PARASOF suffers from a sequence of length

$$L_r + \max_p \ r_p - 1$$

of sequential (sparse) matrix vector products, which makes it uncompetitive in this setting.

5.2.4 Implementation issues

GPUs have a massively parallel architecture consisting of thousands of small, efficient cores designed for handling multiple tasks simultaneously. A function that executes on a GPU is usually called *kernel.* All kernels described in this section are written in CUDA language [112], a general purpose parallel computing platform and programming model that leverages the parallel compute engine in NVIDIA GPUs to solve many complex computational problems, usually referred as General Purpose GPU (GPGPU) computing. Each task in CUDA is referred as *thread.* Threads are organized by the programmer into Thread Blocks (TBs), which in turn are grouped into grids. The scheduler splits the threads of the same block into *warps*, i.e. the scheduling unit which is composed by 32 threads that physically run in parallel, and distributes TBs among the Streaming Multiprocessors (SMs) according their available computing capacity. The threads of a block can execute concurrently, and multiple TBs can execute concurrently on one multiprocessor. CUDA devices have several different memory spaces, all characterized by different scopes and lifetime. Global memory is the main GPU memory, it is visible to all threads within the application (including the CPU), and lasts for the duration of the host allocation. Another invaluable device memory is called shared memory, much smaller in size than the global memory: it is visible to all threads within that block and lasts for the duration of the block. It therefore allows threads to share data between one another. The type of memory used can vary for different data structures within the same program; it determines the speed and the constraints in communicating data among processors during their parallel work.

PARASOF requires a large number of independent QR decompositions of small matrices. Solving together many small linear algebra problems is called "batching", which typically consists of a large number of independent operations (e.g., from hundreds to millions) and the size of each operand is small. The design of algorithms for batched operations is profoundly different from the design of large-scale linear algebra: for example, a key distinction with the (usual) case of a single linear algebra problem is that matrices can be very small, e.g., sub-warp in size. Therefore, instead of having multiple TBs working on a single matrix, a basic kernel for batched problems need to be parameterized to allow configurations where a TB computes e.g. several GEMMs. This design is critical for obtaining close to peak performances for very small sizes [1]. In order to achieve high performances, we started from the implementation of basic linear algebra batched operations provided by the MAGMA project [89] and modified them to build the kernels of PARASOF.

Let us briefly sketch the main features of our CUDA implementation of PARASOF. Each matrix is stored in a compressed sparse format, in particular we choose to store each block as dense in column oriented way, basically a CSC format with dense sub-matrices of fixed shape instead of scalar items. Each matrix is therefore represented as a triplet of arrays: `data` is the array of corresponding nonzero values; `indices` is the array of row indices; `indptr` points to column starts in `indices` and `data` arrays. For a BABD matrix with N internal blocks, the structure `data` is three-dimensional array of size $(2(N+1), n, n)$ and each pair of consecutive blocks is one of the sub-matrices to be orthogonally decomposed as in (5.8). In order to minimize communications and to simplify the data access pattern, each block couple of this type can be joined into a single $2n \times n$ rectangular block, thus the final batched structure actually implemented has size $(N+1, 2n, n)$ and looks like

$$\left[\begin{bmatrix} B_a \\ S_0 \end{bmatrix}, \quad \begin{bmatrix} T_0 \\ S_1 \end{bmatrix}, \quad \dots, \quad \begin{bmatrix} T_{N-2} \\ S_{N-1} \end{bmatrix}, \quad \begin{bmatrix} T_{N-1} \\ B_b \end{bmatrix} \right].$$

The arrays `indptr` and `indices` are omitted, since they can be recovered in relation to the number N of internal blocks. The rectangular blocks so defined are distributed (and possibly queued) among the SMs by the scheduler and overwritten with their QR decompositon in WY compact representation [124], which allows to apply all Householder vectors at once by mean of BLAS3 operations.

5.3 Numerical experiments

5.3.1 Operation Count

Let us first report the operation count for SOF as stated in [4]. We consider the solution of BABD (5.6) systems with N internal block rows, obtained on P processors, where n is the size of each internal block. For simplicity, we take P as a power of 2, for instance $P = 2^L$, and suppose that P divides N. The cost of a single-block forward reduction, that is, to go from (5.7) to (5.10), is

$$\frac{46}{3} n^3 + 30n^2 + O(n),$$

and the cost for a back-substitution (5.13) is

$$15n^2 + O(n).$$

Table 5.1 shows additional memory requirements and floating-point operation count for SOF and PARASOF algorithms, in terms of arithmetic operations carried out sequentially in time by the parallel processor. In the operation count for SOF algorithm, P represents the number of processors that can work in parallel. To exemplify, if each core of the parallel processor performs an addition, we count a single operation. The table also shows the amount of additional memory required during the computation. Note that we refer to the total storage, since a GPU has a single global memory space, instead of a distributed memory typical of coarse grained parallel processors.

For what concerns PARASOF, here P_f stands for the number of cores that are used to compute a single QR decomposition in parallel, the so called *fine grained* parallelism, while P_c stands for the number of QR decompositions that are actually computed in parallel, that is *coarse grained parallelism.* Moreover, $N_r + 1$ is the number of blocks of the reduced system of Section 5.2.2 that are processed in parallel, and it should be chosen to be a power of 2 closed to P_c. Consequently, we set $L_r = \lceil \log_2(N_r + 1) \rceil$. Note that this algorithm requires considerably less memory than its non-massively parallel version. This is because back-substitution equations (5.13) do not need to be stored in the L_r internal steps - actually they do not need to be computed at all - and updated equations (5.22) can be computed in place, so that each coarse-grained processor needs only $2n^2$ additional space to store the compact WY representation of the QR decomposition [124].

Algorithm	Operation count	Memory requirements
SOF	$\left(\frac{46}{3}n^3 + 30n^2 + 15n^2\right)\left(\frac{N}{P} + L - 1\right)$	$4n^2\left(N + P\ L\right) + n(N + P)$
PARASOF	$\frac{42}{3}\frac{n^3}{P_f}L_r + \left(\frac{46}{3}\frac{n^3}{P_f} + 30\frac{n^2}{P_f} + 15n^2\right)\left(\frac{N}{P_c}\right)$	$2n^2P_c + 4n^2\left(N + P_c\right) + n(N + P_c)$

Table 5.1: Operation count and additional memory requirements for SOF and PARASOF algorithms.

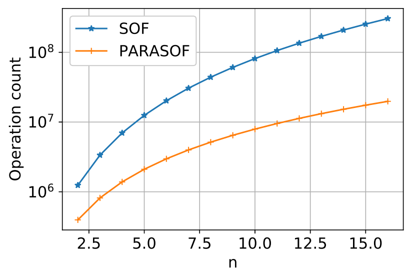
The algorithm PARASOF gains in parallelism with respect to SOF, in fact the number of sequential steps is roughly halved, as can be noted by comparing Figure 5.1 and Figure 5.3. This is confirmed by Figure 5.4, which shows the operation count of both methods SOF and PARASOF in function of the size n (Figure 5.4a on the left) and the number N of blocks (Figure 5.4b on the right). However, parallelism comes at the cost of a significant arithmetic redundancy at the internal steps, which is justified, anyway, on massively parallel computing systems such as GPUs. Here, SOF is supposed to be executed on a medium granularity parallel machine with $P = 16$ processors, while PARASOF is supposed to be executed on a GPU with 10 *Streaming Multiprocessors*, containing 64 cores each, that corresponds to an entry-level GPU. Suppose also to fix at $P_f = 32$ the fine grain cores; this gives $P_c \approx 10 \cdot 64/32 = 20$ coarse grained parallelism.

Figure 5.4 also shows the related theoretical speed-up: Figure 5.4c shows the speed-up that can be obtained in the solution of a BABD system with $N = 2^{16}$ internal blocks in function the size n of each block, while Figure 5.4d shows the speed-up that can be achieved in the solution of a BABD system with internal blocks of size 16×16 in function of the number of blocks N.

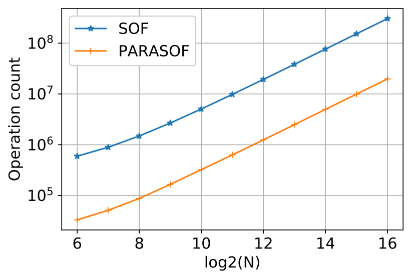
5.3.2 Execution Times

The theoretical analysis summarized in Figure 5.4 shows potential results that could be slightly modified in practice. We decided not to compare experimentally PARASOF with a straightforward implementation of SOF on GPUs because of its poor parallel efficiency, as depicted in Figure 5.1 and Figure 5.3. Indeed, for this reason we propose PARASOF as a massive parallel algorithm inspired by SOF. Moreover, we are not aware of a public domain implementation of SOF on parallel, distributed memory computers.

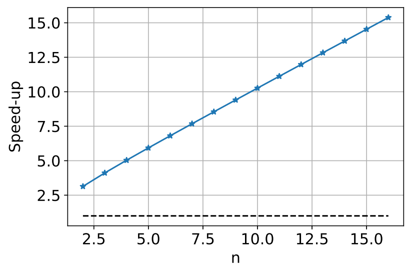
In this section we measure the execution time for PARASOF with BABD matrices (5.1) with dense square blocks whose entries have been randomly generated by mean of `numpy.random.rand()` and we compare the time to solution of PARASOF with BABDCR [2], a cyclic reduction based FORTAN90 package



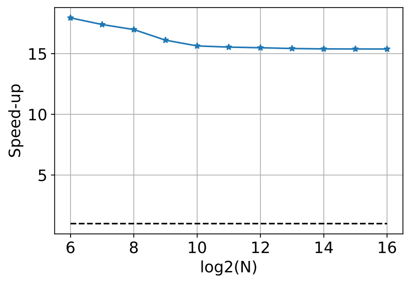
(a) Operation count, $N = 2^{16}$.



(b) Operation count, $n = 16$.



(c) Theoretical speed-up, $N = 2^{16}$.



(d) Theoretical speed-up, $n = 16$.

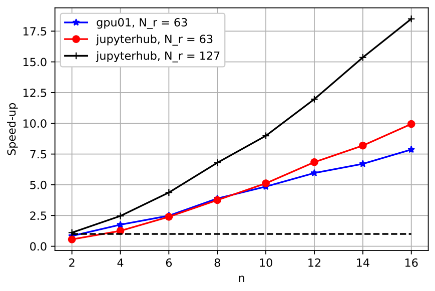
Figure 5.4: Operation count and theoretical speedup of PARASOF vs SOF.

for the solution of BABD systems with the same workflow as SOF. We test its performance by running the algorithm on two different workstations:

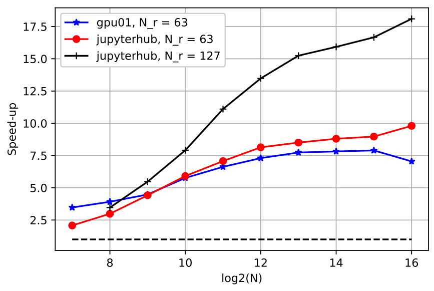
- `jupyterhub`, with a 3.70GHz Intel(R) Xeon(R) W-2145 CPU and an Nvidia TITAN V graphic card;
- `gpu01`, with a 3.50GHz Intel(R) Core(TM) i7-2700K CPU and an Nvidia GeForce GTX1060 graphic card.

The TITAN V card contists of 80 SMs with 64 cores each, while the GTX1060 card is a system of 10 SMs with 128 cores each. We measure performance using the execution times; an analysis of performance in terms of FLOPS for each algorithm can be found in Section 5.3.1. In the following, the performance comparison of PARASOF GPU solver and BABDCR CPU solver is carried without taking into account the time for the CPU-GPU data transfer over the PCI-Express bus, which can be however superposed to other computations whenever the solver is used in a more complex application. The problem sizes we choose range both in the number N of internal blocks and in the size n of each square block. We focus on subwarp n, so $n \leq 16$ since each QR is performed on rectangular blocks of $2n \leq 32$ rows and subwarp sizes are, in general, critical for GPU efficiency.

Figure 5.5 shows the speed-up of PARASOF over BABDCR in function of the number N of blocks (Figure 5.5b) and in function of the size n of each

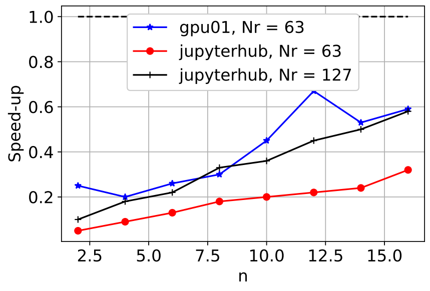


(a) Block rows $N \approx 2^{16}$.

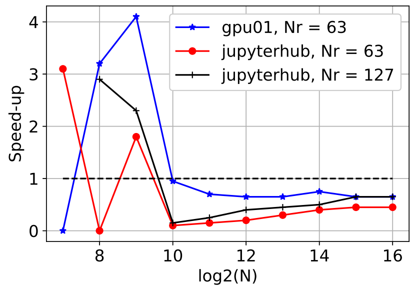


(b) Block size $n = 16$.

Figure 5.5: Speed-up (factorization and solution) of PARASOF over BABDCR.



(a) Block rows $N \approx 2^{16}$.



(b) Block size $n = 16$.

Figure 5.6: Speed-up (only solution) of PARASOF over BABDCR.

block (Figure 5.5a). Here the factorization and solution phases are performed at the same time. Figure 5.6 shows the speed-up of PARASOF over BABDCR in function of the number N of blocks (Figure 5.6b) and in function of the size n of each block (Figure 5.6a). In this case we are comparing only the solution phase on a different randomly generated right-hand side. The oscillations on the left are due to BABDCR execution times.

Notice that the maximum achieved speed-up, varies quite a lot depending on the size of the reduced system $nN_r \times nN_r$. Indeed, N_r is a parameter that should be chosen depending on the device we are using. Here, $N_r = 63$ generates a reduced system with a total number of block rows equal to 64, while $N_r = 127$ generates a reduced system with 128 block rows. The first choice is well suited for `gpu01`, and it yields similar performances on `jupyterhub`. However, since the number of streaming multiprocessors of `jupyterhub` is larger, if we allow $N_r = 127$ the performances obtained can vary from 10x to 17.5x. This highlights how the algorithm's performances can be tuned according to the parallel architecture on which it executes. For the same reason, we observe a weaker speed-up on `gpu01`.

The PARASOF GPU solver outperforms the BABDCR CPU solver on the factorization phase (Figure 5.5), except for the case $n = 2$ and for the smallest values of $\log_2(N)$. This is because for a smaller system size, serial solvers are very efficient. However, when BABD systems arise from a real BVODE model (5.4), n and N will not be so small.

5.4 Conclusions

In this chapter, we have studied existing parallel solvers for BABD systems, and we analyzed the weaknesses of their straightforward implementation on a GPU. We therefore presented PARASOF, an algorithm for the solution of linear systems with BABD matrices on massively parallel computing systems like GPUs. The proposed solver achieved up to 18x speedup over the BABDCR CPU solver, but it varies obviously with the architectures involved. Its parallel computational complexity, which is independent from the hardware architecture, and execution times show a good acceleration of this algorithm compared to the state-of-the-art existing algorithms. An analogous derivation for ABD systems can be easily obtained from the description here provided.

The GPU acceleration of these algorithms is useful, for example, in optimal control systems design [11, 12, 16], that motivated this research. Typically, this application requires the solution of a long sequence of BABD systems with different parameterizations of the controlled-system BVODE model, in the shortest possible time. These simulations are often run on embedded systems, that generally have low perfomance CPUs for computational purposes, but can have a GPU on board. For example, running the experiments of Section 5.3 on an entry-level NVIDIA Jetson TX1 we got slightly better performances than those obtained on the high-end CPU of `gpu01`.

Chapter 6

ILU preconditioning for Navier-Stokes equations

General Purpose computing on GPUs (GPGPU) is nowadays a cost effective solution for computational intensive simulations, like those arising from the discretization of the Navier-Stokes equations, where often $\mathcal{O}(10^5)$ coupled non-linear equations must be solved at each discretization time instant, to capture the relevant underlying physics of the fluid motion. The inherent massively parallel architecture of GPUs demands completely different algorithms from those used in a mainly sequential, CPU based, computing architecture. For example, it turns out that in the linear algebra core problem underlying the CFD specific discretization methods, the cost of each iteration may be much cheaper for methods that are not much efficient in the total iteration count, thus making them faster, anyway, in the time spent for the simulation, which is the main performance desired from a parallel implementation. Thus on a GPU, surprisingly, a simple diagonal preconditioner actually outperforms much more mathematically robust preconditioners, like ILU, for a wide range of Reynolds/Atwood numbers. However, this is not the case anymore when the Reynolds number decreases at low levels, and the viscosity of the fluid dominates against convection, for a wide range of Atwood numbers. This phenomenon becomes even more evident as the higher viscosity drives the fluid toward a quasi-solid behavior, like e.g. in a phase transition problem, quite important in applications.

The aim of this chapter is to obtain from an ILU preconditioner the best performance on a GPU architecture. For this reason, we mainly investigate the parallelization of the triangular systems solver needed at each time iteration, within the GMRES method preconditioned by ILU factorizations. This is the computationally intensive operation that requires the most of the time spent by the simulation. We compare an iterative method with a direct one. Several issues arise in this analysis. Mainly, direct methods suffer from a low level of parallelization, being quite inefficient on a GPU, while iterative methods suffer from the high ill-conditioning of the triangular ILU factors, requiring a lot of iteration with simple methods, like Jacobi, or a second preconditioner in more sophisticated methods, thus making the whole algorithm too expensive and/or complicated. We show, for a wide range of Reynolds/Atwood numbers, that the full-iterative ILU-based solver here considered has better performance on a GPU against intrinsic massively parallel preconditioners like the diagonal preconditioner.

In the literature, it has been shown that the quality and effectiveness of incomplete factorizations can be significantly enhanced by mean of RCM ordering [51], in particular for nonsymmetric problems [14]. Sparse triangular systems can be solved on parallel computers using e.g. direct level scheduling techniques [5, 122, 108], iterative Jacobi and block Jacobi [7] or hybrid direct/iterative methods [114, 45]. Level scheduling techniques are severely penalized by the use of RCM ordering. In blocking techniques, such as block Jacobi and SPIKE, there is still a direct component in the solver. Moreover, their efficiency highly depends on the block structure and finding the optimal blocking for a given problem is not an easy task [49, 104]. Moreover, there is a difficult trade-off between parallelism and speed of convergence, as we will show in the numerical result. Apart from ILU decompositions, another class of preconditioner which is recently growing in popularity on parallel machines is the so called Incomplete Sparse Approximate Inverse (ISAI) [9].

Recent applications of this kind of simulations concern e.g. the solution of inverse problems arising in the monitoring of systems whose relevant physical variables cannot be overall measured, e.g. where some time-dependent boundary conditions must be inferred from measurements at the down-stream of a river [56]. The solution of such an inverse problem requires hundreds of simulations of the direct problem here considered, making its parallel acceleration highly desirable or even essential.

Note that a relevant topic that we do not consider here is mesh adaptation based on a-posteriori error indicators [28], that in case of fluid motion would be quite anisotropic [105] and, as is well known, demanding for an high quality adapted mesh. This is a common way to tailor the number of degrees-of-freedom to the specific problem simulation. We do not consider it here mainly because a parallel mesh generator/adaptation on GPUs is outside the scope of this work and not an easily available tool.

The structure of this chapter is as follows: in Section 6.1 we describe the parallelization of the solution of triangular systems with direct and iterative methods. In Section 6.2 we discuss a technique to update the L and U factors of an ILU factorization along the sequence of linear systems arising from the discretization of a time-dependent problem. In Section 6.3 we describe the equations governing the motion of an incompressible viscous Newtonian fluid with variable density and a reasonable discretization scheme for them. The parallel GPU implementation of this scheme is presented in Section 6.4 and the numerical experiments are detailed in Section 6.5. It follows a section with conclusions.

6.1 Sparse triangular solves on GPUs

Solving triangular systems is an inherently sequential computation activity: in fact, these systems are typically solved by forward and backward substitution. The most common parallel sparse triangular solvers are based on *level-scheduling* techniques [5, 122], and their efficient implementation on GPUs is still an open challenge [108]. These methods are made of two phases: the analysis phase represents the data dependencies in the solution of a triangular system as a directed graph, where the nodes stand for the rows and an edge represents a dependence between them; thus one can group together independent rows in levels, which are processed one by one in the solve phase. Since the matrices considered are triangular, the data dependency are represented as a directed acyclic graph (DAG). Notice that, by construction, the rows corresponding to nodes at the same level in the graph are independent and can be processed in parallel, but the levels still have to be processed in a sequential fashion. These techniques' weakness lies in the amount of work available in each level, which is often too poor too fully exploit the massive parallelism of many-cores architectures. This situation typically arises when ordering algorithms are applied.

A different approach consists in relaxing the solution of triangular systems through iterative methods. Indeed, in the case of triangular solves for preconditioning in Krylov methods, an approximate solution can be accepted instead of the exact one, at the price of increasing the outer number of iterations for convergence. This choice is furthermore justified since these triangular systems often arise from approximate factorizations.

Classical iterative methods update the components of the solution vector and therefore require synchronization between iterations. The solution can be updated using information of the previous iteration, such as Jacobi method, or also information coming from the current one, such as Gauss-Seidel. It is known that Gauss-Seidel convergence is faster with respect to Jacobi, but faster convergence comes at cost of a reduced parallelism. However, Jacobi method is based on sparse matrix-vector multiplication (SpMV), which is highly parallel and high-performing on GPUs, making it an attractive choice.

The idea of using iterative triangular solves for ILU first appeared in [48], and block Jacobi iterative triangular solves for ILU was published in [49]. Huckle and Bräckle [94] proposed other stationary iterative methods for the iterative solution of LU systems, and more recently Anzt et al. [9] explored how incomplete sparse approximate inverse preconditioning can be applied to sparse triangular systems.

6.1.1 On level-scheduling techniques

Let us consider the symmetric structured matrix $A = (a_{ij})$ shown in (6.1) as example. We are interested in its incomplete LU factorization with no fill-in, also known as ILU(0) [122], i.e. we look for $A \approx LU$, where the triangular factors L and U have the same pattern as the triangular parts of the original matrix A. Figure 6.1 shows the DAG illustrating the dependencies within the rows of the lower triangular factor, whose pattern has been highlighted in bold in (6.1). The graph is constructed so that there is an edge from node j to node i with $i > j$ if there is a non zero matrix entry a_{ij}. As we can see from Figure 6.1, the rows of (6.1) can be grouped in two levels of four elements each, the rows therein are independent and they can be processed in parallel.

$$\begin{pmatrix}
\boldsymbol{a}_{11} & & & & a_{15} & a_{16} & & \\
& \boldsymbol{a}_{22} & & & & a_{26} & a_{27} & \\
& & \boldsymbol{a}_{33} & & & & a_{37} & a_{38} \\
& & & \boldsymbol{a}_{44} & & & & a_{34} \\
\boldsymbol{a}_{51} & & & & \boldsymbol{a}_{55} & & & \\
\boldsymbol{a}_{61} & \boldsymbol{a}_{62} & & & & \boldsymbol{a}_{66} & & \\
& \boldsymbol{a}_{72} & \boldsymbol{a}_{73} & & & & \boldsymbol{a}_{77} & \\
& & \boldsymbol{a}_{83} & \boldsymbol{a}_{84} & & & & \boldsymbol{a}_{88}
\end{pmatrix} \tag{6.1}$$

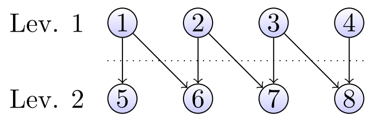


Figure 6.1: Data dependency DAG.

Now, consider a well known band-reducing matrix ordering, namely Reverse Cuthill-Mckee (RCM) algorithm [51]. Let us denote by $\tilde{A} = (\tilde{a}_{ij})$ the matrix in (6.2), obtained by applying RCM ordering to matrix (6.1), and consider again its ILU(0) factorization. Figure 6.2 shows the data dependency DAG associated to the L factor, whose pattern is highlighted in (6.2).

$$\begin{pmatrix}
\tilde{\boldsymbol{a}}_{11} & \tilde{a}_{12} & & & & & & \\
\tilde{\boldsymbol{a}}_{21} & \tilde{\boldsymbol{a}}_{22} & \tilde{a}_{23} & \tilde{a}_{24} & & & & \\
& \tilde{\boldsymbol{a}}_{32} & \tilde{\boldsymbol{a}}_{33} & \tilde{a}_{34} & & & & \\
& \tilde{\boldsymbol{a}}_{42} & \tilde{\boldsymbol{a}}_{43} & \tilde{\boldsymbol{a}}_{44} & \tilde{a}_{45} & & & \\
& & & \tilde{\boldsymbol{a}}_{54} & \tilde{\boldsymbol{a}}_{55} & \tilde{a}_{56} & & \\
& & & & \tilde{\boldsymbol{a}}_{65} & \tilde{\boldsymbol{a}}_{66} & \tilde{a}_{67} & \\
& & & & & \tilde{\boldsymbol{a}}_{76} & \tilde{\boldsymbol{a}}_{77} & \tilde{a}_{78} \\
& & & & & & \tilde{\boldsymbol{a}}_{87} & \tilde{\boldsymbol{a}}_{88}
\end{pmatrix} \tag{6.2}$$

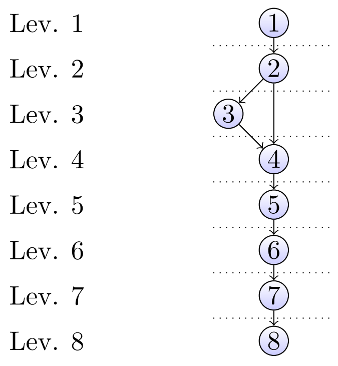


Figure 6.2: Data dependency DAG with RCM ordering.

The ordering has modified the data dependency DAG, maximizing the number of levels and making the computation purely sequential. On one hand, as the matrix entries get clustered close to the diagonal, the dependencies between the rows increase and the related DAG gets "stretched", reducing the exploitable parallelism in level-scheduling techniques. On the other hand, it has been shown that RCM ordering can significantly improve the quality and the effectiveness of incomplete factorizations [14], in particular for nonsymmetric problems.

6.1.2 An iterative approach

Let us consider the square triangular system $Tx = b$ and suppose the matrix T has all non-zeros on its diagonal, i.e. T is invertible. Jacobi iteration writes

$$\begin{cases} x^{(0)} & = \operatorname{diag}(T)^{-1}b, \\ x^{(k+1)} & = \operatorname{diag}(T)^{-1}b + (I - \operatorname{diag}(T)^{-1}T)x^{(k)}. \end{cases} \tag{6.3}$$

In what follows we will refer to Jacobi iterations as sweeps. Convergence for the Jacobi method is established as long as the iteration matrix, namely the matrix $(I - \operatorname{diag}(T)^{-1}T)$, has spectral radius strictly less than one. In this case the condition $\rho(I - \operatorname{diag}(T)^{-1}T) < 1$ is trivially verified, since the iteration matrix is nilpotent. Few sweeps of (6.3) will be effective when the matrix T is not badly ill-conditioned. However, an ILU factorization of an ill-conditioned matrix A must have ill-conditioned L, U factors to result in a faster convergence of Krylov methods [9]. A more robust alternative consists in a block approach. If a block structure is imposed on the matrix T then it can be trivially written as the sum

$$T = D + \widetilde{T}, \tag{6.4}$$

where $D = \operatorname{diag}(D_1, \cdots, D_n)$ is a block diagonal matrix. The block Jacobi iteration writes as its scalar counterpart (6.3) by simply replacing $\operatorname{diag}(T)$ with D. The new iteration matrix $(I - D^{-1}T)$ still fulfills the convergence condition, but the method is no more based on SpMV: in fact, block Jacobi method requires the computation of $D^{-1}T$. One can explicitly invert the matrix D, taking into account that matrix inversion can lead to numerical instability, or compute the solution of the triangular system

$$Dy = Tx^{(k)}, \tag{6.5}$$

at each sweep. Again, a direct level-scheduling method should be used to solve the system above. If we look at the error we have

$$\begin{aligned} e^{(k+1)} = x^{(k+1)} - x = D^{-1}(b + (D-T)x^{(k)}) - D^{-1}(b + (D-T)x) = \\ D^{-1}(D-T)(x^{(k)} - x) = (I - D^{-1}T)e^{(k)} = (I - D^{-1}T)^{k+1}e^{(0)}, \end{aligned} \tag{6.6}$$

and therefore

$$\left\| e^{(k)} \right\| \leq \left\| (I - D^{-1}T)^k \right\| \left\| e^{(0)} \right\|. \tag{6.7}$$

It's easy to see that the iteration matrix $G = (I - D^{-1}T)$ is nilpotent and its index is equal to the number of diagonal blocks of D. This means, as one could expect, that fewer bigger blocks imply a faster convergence. As a drawback, a small number of blocks also strongly limits the available parallelism in computations. Moveover, one has to consider the off-diagonal blocks of the iteration matrix, given by $G_{ij} = D_i^{-1}T_{ij}$, $i \neq j$. If the norm of D_i^{-1} is small, then the norm of the off diagonal blocks will be small [49], as desired. Block Jacobi method can be more effective for ill-conditioned problems, but clearly its performance and efficiency highly depend on the properties of the block diagonal matrix D.

6.2 Preconditioning a sequence of linear systems

Let A be a sparse nonsingular matrix of size $n \times n$. In classic numerical linear algebra literature, a preconditioner is a matrix M such that $M^{-1} \approx A^{-1}$, i.e. its inverse approximates that of A. This is used to transform the linear system (1.1) into a more tractable one, namely

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}. \tag{6.8}$$

Therefore, M should be chosen such that $M^{-1}A$ is a good approximation of the identity. At each step of the preconditioned algorithm, it is necessary to to solve a linear system with matrix M.

It is known that the incomplete LU decomposition $A \approx LU$ is a class of robust preconditioners for Krylov subspace method [122]. Incomplete factorizations are computed up to a residual matrix R which is set to zero outside a given pattern

$$R_{ij} = (A - LU)_{ij} = 0, \quad (i,j) \in \mathcal{S}, \tag{6.9}$$

where $\mathcal{S}$ is some prescribed sparsity pattern related to A. This decomposition is not unique, therefore it is customary to assume L to be a unit triangular matrix. Depending on the difficulty of the problem, one can choose the sparsity pattern (and consequently the accuracy of the incomplete factorization) on the basis of a drop tolerance or a level-of-fill scheme, e.g. by keeping the k-th order fill-ins, leading to ILU(k) preconditioners [122]. From now on we will assume that the lower triangular factor L coming from LU-type decompositions is a unit triangular matrix (i.e. all the entries of its main diagonal are ones). Given a square matrix X, we will denote by $\text{triu}(X)$ its upper triangular part, which includes the diagonal, while for the lower triangular part we will distinguish the strict lower part denoted by $\text{tril}(X)$, that has only zeros on the diagonal, from the unit lower part denoted by $\overline{\text{tril}}(X)$.

Consider now a sequence of linear systems

$$A_n \mathbf{x}^n = \mathbf{b}^n, \quad n = 0, 1, \ldots. \tag{6.10}$$

Many important applications involve a sequence A_n of slowly varying matrices, which is a common situation in computational fluid dynamics, where the equations change only slightly and possibly in some parts of the physical domain. In such situations it is wasteful to recompute entirely any preconditioner computed for the previous coefficient matrix. Calgaro et al. [37] proposed an iterative technique to incrementally update incomplete LU decompositions, called Iterative Threshold Alternating Lower Upper (ITALU) correction. Given initial guesses $L^{(0)}$, $U^{(0)}$, the ITALU algorithm is presented in Algorithm 10. In what follows, we write ITALU(m) in order to emphasize the fixed number of sweeps.

Algorithm 10 Iterative Threshold Alternating Lower Upper (ITALU)

```
    Inputs: $A$, $L^{(0)}$, $U^{(0)}$, $m$
    Outputs: $L$, $U$
 1: for $k = 0, \ldots, m-1$ do
 2:     Compute $R^{(k)} = A - L^{(k)}U^{(k)}$
 3:     Compute $X_U = \text{triu}((L^{(k)})^{-1}R^{(k)})$
 4:     Apply dropping to $X_U$
 5:     $U^{(k+1)} = U^{(k)} + X_U$
 6:     Compute $R^{(k+1/2)} = A - L^{(k)}U^{(k+1)}$
 7:     Compute $X_L = \text{tril}(R^{(k+1/2)}(U^{(k+1)})^{-1})$
 8:     Apply dropping to $X_L$
 9:     $L^{(k+1)} = L^{(k)} + X_L$
10: end for
11: $L = L^{(m)}$, $U = U^{(m)}$
```

The algorithm above is based on the corrections steps 3, 7 whose computation require the solution a set of n triangular systems, or a triangular system with multiple sparse right-hand side

$$X_U = (L^{(k)})^{-1}R^{(k)} \quad \rightarrow \quad L^{(k)}X_U = R^{(k)}.$$

This computation turns out to be a bottleneck on parallel architectures, specially when the problem size n is large. The same authors proposed [37] a less accurate but far simpler update, that can be obtained by replacing the computation in step 3 (resp. 7) with the following diagonal system

$$\operatorname{diag}(L^{(k)})X_U = R^{(k)} \qquad\qquad (\text{resp. } X_L \operatorname{diag}(U^{(k+1)}) = R^{(k+1/2)}).$$

Moreover, steps 4 and 8 can be omitted by computing only the entries of the residual matrix which lie on the sparsity pattern $\mathcal{S}$ chosen for the incomplete factorization. Formally, the sparsity pattern $\mathcal{S}$ can be associated to a $n \times n$ matrix S whose entries are zeros and ones: a zero entry corresponds to an unwanted element that will be dropped. We denote by $X \odot S$ the component-wise product of the matrices X and S. It can be proved that if $\mathcal{S}$ contains the pattern of $\widetilde{L}$ and $\widetilde{U}$, where we supposed that A admits an LU decomposition $A = \widetilde{L}\widetilde{U}$, and the initial guess $L^{(0)}$ is such that $L^{(0)} \odot S = L^{(0)}$ and $L^{(k)}$, $U^{(k)}$ are defined for $k = 0, \dots, n$, then the sequences $L^{(k)}$, $U^{(k)}$ converge to $\widetilde{L}$ and $\widetilde{U}$ respectively in at most n steps [37].

We define the matrix $M^{(k)}$ such that

$$\operatorname{triu}(M^{(k)}) = U^{(k)}, \qquad\qquad \overline{\operatorname{tril}}(M^{(k)}) = L^{(k)},$$

recalling that there is no need to store the diagonal of the lower factor since it is a unit triangular matrix. Including this notation, Algorithm 10 rewrites in its simplified version called Simplified Iterative Threshold Alternating Lower Upper (SITALU) and presented in Algorithm 11. In what follows, we write SITALU(m) in order to emphasize the fixed number of sweeps.

Algorithm 11 Simplified Iterative Threshold Alternating Lower Upper (SITALU)

```
   Inputs: $A$, $L^{(0)}$, $U^{(0)}$, $m$
   Outputs: $L$, $U$
1: for $k = 0, \dots, m-1$ do
2:     Compute $R^{(k)} = (A - \overline{\operatorname{tril}}(M^{(k)})\operatorname{triu}(M^{(k)})) \odot S$
3:     $\operatorname{triu}(M^{(k+1)}) = \operatorname{triu}(M^{(k)}) + \operatorname{triu}(R^{(k)})$
4:     $\operatorname{tril}(M^{(k+1)}) = \operatorname{tril}(M^{(k)}) + \operatorname{tril}(R^{(k)}) \operatorname{diag}(M^{(k)})^{-1}$
5: end for
6: $L = L^{(m)}$, $U = U^{(m)}$
```

Steps 3 and 4 can be performed at once with a matrix-matrix summation provided that the residual matrix has been previously scaled as follows

$$r_{ij}^{(k)} = r_{ij}^{(k)} \operatorname{diag}(M^{(k)})_{jj}, \quad i < j,$$

where $R^{(k)} = \left(r_{ij}^{(k)}\right)$. Notice that Algorithm 11 produces no fill-in in the preconditioning factors, and if we choose $\mathcal{S}$ to be the pattern of A, then all matrices involved share the same sparsity pattern, with an important memory saving. Algorithm 10 and Algorithm 11 can be used to update the factorization of A_n in (6.10) at the current step n by taking as initial guesses for the ILU factors of A_{n-1} at the previous time step.

An alternative and more general approach has been proposed in [13] in the context of Newton-Krylov methods. This procedure differs from the one just presented as it mainly relies on approximate inverse preconditioners, whose application is based on matrix-vector multiplication, and therefore it can be efficiently implemented on parallel computers. An inverse preconditioner is a matrix M such that $M \approx A^{-1}$, and thus the linear system (1.1) is transformed into

$$MA\mathbf{x} = M\mathbf{b}. \tag{6.11}$$

The main difference with (6.8) is that the application of an inverse preconditioner requires only matrix-vector products rather than the solution of one or more linear systems. Here we consider this procedure for the update of the inverse ILU preconditioner [141].
Consider the sequence (6.10) and consider first an ILU factorization of the initial matrix

$$A_0 \approx LDU, \tag{6.12}$$

where L is lower triangular with unitary diagonal, U is upper triangular with unitary diagonal, and D is diagonal and nonsingular. The idea is to avoid the triangular solving by computing sparse approximate inverses on the triangular factors

$$Z \approx L^{-T}, \qquad W \approx U^{-1}, \tag{6.13}$$

and to set the preconditioner M_0 as

$$M_0 = WD^{-1}Z^T \approx (A_0)^{-1}. \tag{6.14}$$

Since Z, W can be dense even if L and U are very sparse, again a sparsity can be enhanced on the basis on a drop rule or a level-of-fill scheme. However, the computation in (6.13) require the solution of two triangular systems with multiple right-hand-sides, or, equivalently, the solution of $2n$ triangular systems, which is indeed a computational intensive operation.
An incomplete factorization of the inverse of the current matrix A_n can be obtained as follows. Let first

$$\Delta_n = A_n - A_0, \qquad E_n = Z^T \Delta_n W, \tag{6.15}$$

consequently we have

$$A_n^{-1} = (A_0 + \Delta_n)^{-1} = WW^{-1}(A_0 + \Delta_n)^{-1}Z^{-T}Z^T \approx W(D + E_n)^{-1}Z^T. \tag{6.16}$$

This suggests to set the preconditioner M_n equal to $W(D+E_n)^{-1}Z^T$. However, there is no guarantee that the matrix $(D+E_n)^{-1}$ is sparse. Therefore, we set

$$M_n = W(D+\widetilde{E}_n)^{-1}Z^T, \tag{6.17}$$

where

$$\widetilde{E}_n = (Z^T\widetilde{\Delta}_n W) \odot S_E, \qquad \widetilde{\Delta}_n = \Delta_n \odot S_\Delta.$$

Here, S_E, S_Δ are binary matrices associated to given sparsity patterns $\mathcal{S}_E, \mathcal{S}_\Delta$. A general choice suggested in [13] consists in taking S_E, S_Δ as banded matrices with a prescribed bandwidth. In such a case, $(D+\widetilde{E}_n)$ is also a banded matrix. If its inverse $(D+\widetilde{E}_n)^{-1}$ admits an LU decomposition, then its factors are banded matrices with the same bandwidth. Alternatively, a Givens QR decomposition may be computed whose R factor has a double bandwidth with respect to the bandwidth of the original matrix. These operations are computationally expensive and their parallel implementation is not trivial. Moreover, the computation (6.13) to obtain Z and W may be cause a bottleneck, as it has been already pointed out. However, this operation does not have to be performed for each matrix in the sequence (6.10): it can be performed once for the first matrix A_0, and it can be performed again later on in order to restart the procedure, e.g. with every $\bar{n}$ iterations, or when the preconditioner loses its effectiveness. As an advantage, the proposed technique can be generalized to update any incomplete factorization of the inverse of the reference matrix, e.g. approximate inverse (AINV) preconditioner for nonsymmetric matrices [15].
In this thesis, we restrict to techniques based on direct ILU factorizations, since an accurate comparison between these two approaches would mainly require an extensive experimental activity made on benchmark problems, which is out of the scope of this treatise.

6.2.1 Comparison with existing parallel ILU algorithms

Different strategies are possible in order to update an ILU decomposition apart from those described in [37], in particular Chow and Patel [48] provide a method that was later expanded in [8]. The technique is closely related to ITALU algorithm, as we show in what fallows. With reference to Algorithm 14, write

$$\begin{aligned}
M^{(k)} &= \begin{cases} l_{ij}^{(k)}, & i > j \\ u_{ij}^{(k)}, & i \le j \end{cases}, && (i,j) \in \mathcal{S}, \\
A &= a_{ij}, && (i,j) \in \mathcal{S} \\
R^{(k)} &= r_{ij}^{(k)}, && (i,j) \in \mathcal{S}.
\end{aligned}$$

Then, the SITALU(m) algorithm rewrites component-wise as Algorithm 12.

Algorithm 12 Component-wise SITALU(m)

```
   Inputs: $A$, $L^{(0)}$, $U^{(0)}$, $m$
   Outputs: $L$, $U$
1: for $k = 0, \dots, m-1$ do
2:     for $(i,j) \in \mathcal{S}$ do
3:         Compute $r_{ij}^{(k)} = a_{ij} - \sum_{l=1}^{\min(i,j)} l_{ij}^{(k)} u_{lj}^{(k)}$,
4:         $l_{ij}^{(k+1)} = l_{ij}^{(k)} + r_{ij}^{(k)} / u_{jj}^{(k)}, \quad i > j$
5:         $u_{ij}^{(k+1)} = u_{ij}^{(k)} + r_{ij}^{(k)}, \quad i \le j$
6:     end for
7: end for
8: $L = L^{(m)}$, $U = U^{(m)}$
```

Step 4 rewrites

$$\begin{aligned} l_{ij}^{(k+1)} &= l_{ij}^{(k)} + r_{ij}^{(k)} / u_{jj}^{(k)} \\ &= l_{ij}^{(k)} + \left(a_{ij} - \sum_{l=1}^{j} l_{il}^{(k)} u_{lj}^{(k)} \right) / u_{jj}^{(k)} \\ &= l_{ij}^{(k)} + \left(a_{ij} - \sum_{l=1}^{j-1} l_{il}^{(k)} u_{lj}^{(k)} \right) / u_{jj}^{(k)} - l_{ij}^{(k)} u_{jj}^{(k)} / u_{jj}^{(k)} \\ &= \left(a_{ij} - \sum_{l=1}^{j-1} l_{il}^{(k)} u_{lj}^{(k)} \right) / u_{jj}^{(k)}. \end{aligned}$$

Similarly, step 5 rewrites

$$\begin{aligned} u_{ij}^{(k+1)} &= u_{ij}^{(k)} + r_{ij}^{(k)} \\ &= u_{ij}^{(k)} + \left(a_{ij} - \sum_{l=1}^{i} l_{il}^{(k)} u_{lj}^{(k)} \right) \\ &= u_{ij}^{(k)} + \left(a_{ij} - \sum_{l=1}^{i-1} l_{il}^{(k)} u_{lj}^{(k)} \right) - l_{ii}^{(k)} u_{ij}^{(k)} \\ &= a_{ij} - \sum_{l=1}^{i-1} l_{il}^{(k)} u_{lj}^{(k)}. \end{aligned}$$

These expressions are exactly the synchronous version of the Fine-Grained Parallel Incomplete Factorization algorithm proposed by Chow and Patel in [48] and presented in Algorithm 13.

Algorithm 13 Fine-Grained Parallel Incomplete Factorization

```
   Inputs: $A$, $L^{(0)}$, $U^{(0)}$, $m$
   Outputs: $L$, $U$
1: for $k = 0, \dots, m-1$ do
2:     for $(i, j) \in \mathcal{S}$ do
3:         $l_{ij}^{(k+1)} = \left(a_{ij} - \sum_{l=1}^{j-1} l_{il}^{(k)} u_{lj}^{(k)}\right) / u_{jj}^{(k)}, \quad i > j$
4:         $u_{ij}^{(k+1)} = a_{ij} - \sum_{l=1}^{i-1} l_{il}^{(k)} u_{lj}^{(k)}, \quad i \le j$
5:     end for
6: end for
7: $L = L^{(m)}$, $U = U^{(m)}$
```

6.3 Nonhomogeneous incompressible Navier-Stokes equations

Let Ω be a domain in $\mathbb{R}^d$, $d = 2, 3$, the equations describing the motion of an incompressible viscous Newtonian fluid with variable density are the following:

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{u}) = 0, \tag{6.18}$$

$$\partial_t (\rho \mathbf{u}) + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) - \mu \Delta \mathbf{u} + \nabla p = \rho \mathbf{f}, \tag{6.19}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{6.20}$$

where the unknowns are the velocity field $\mathbf{u} = (u_1, \dots, u_d)$ and the scalar functions ρ, p representing the density and the pressure of the fluid respectively. Here the right-hand side $\mathbf{f} = (f_1, \dots, f_d)$ of (6.19) embodies the external forces, such as gravity, and $\mu > 0$ is the viscosity of the fluid. These functions depend on the space variable $\mathbf{x} = (x_1, \dots, x_d) \in \Omega$ and on the time variable $t \in [0, T]$, with $T > 0$.

Existence and uniqueness of the solution of (6.18)-(6.20) can be proved [101], provided that the system above is coupled with initial and boundary conditions

$$\mathbf{u}_0\big|_\Gamma = \mathbf{0} \qquad \rho\big|_{t=0} = \rho_0, \qquad \mathbf{u}\big|_{t=0} = \mathbf{u}_0, \tag{6.21}$$

where Γ is the boundary $\partial\Omega$. Notice that the system (6.18)-(6.20) differs from the usual Navier-Stokes formula only because of the presence of the non-constant density ρ. Let us denote by Δt the time step and set $t^n = n\Delta t$, with $n \ge 0$. Thanks to a particular time splitting due to Strang [130], it is possible to treat separately the conservation of mass (6.18) from the momentum equation and the incompressibility constraint (6.19)-(6.20), to solve the former with a finite volume method, which is a natural choice for conservation laws and is massively parallel since the solution can be computed independently on each control volume, and the latter with a finite element approximation. The resulting hybrid finite volume/finite element scheme relies on some compatibility conditions for the velocity field. Details on the spatial and temporal discretization are provided in [35, 36].

Notice that even if the density update is separated from the other equations with the Strang splitting, the velocity and pressure are still coupled. One can furthermore simplify the numerical problem by mean of the so called projection methods, first introduced in [132, 47]. Projection methods are based on the Helmholtz–Hodge decomposition and they all are based on the solution of a problem in the form:

$$-\nabla \cdot \left(\frac{1}{\rho^{n+1}} \nabla \Phi \right) = \Psi, \qquad \partial_n \Psi \big|_{\Gamma} = 0, \tag{6.22}$$

for suitable Φ and Ψ. When the density ρ is constant this is no more than a Poisson equation, but in the variable case a finite element solution of the equation above requires the assembling (and the preconditioning) of a time dependent matrix at each time step. Note also that the higher the gap between the maximum and minimum density value, the more ill-conditioned will be the approximation of the operator above. This is why we choose to adopt a penalty coefficient based strategy [86]. We define

$$\chi \in \left(0, \min_{x \in \Omega} \rho_0 \right], \tag{6.23}$$

where we assume there is no empty region at time $t = 0$, which means that $\min_{x\in\Omega} \rho_0 > 0$. In practice we will set $\chi = \min_{x\in\Omega} \rho_0$. Let us introduce a new scalar variable ϕ which will represent the pressure correction in time and we set $\phi^0 = 0$. Suppose that the numerical solution $(\rho^n, \mathbf{u}^n, p^n, \phi^n)$ at time t^n is known, then the velocity field at time t^{n+1} is obtained with the old value of the pressure by solving

$$\rho^* \left(\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + (\mathbf{u}_\star^{n+1} \cdot \nabla)\mathbf{u}^{n+1} \right) - \mu \Delta \mathbf{u}^{n+1} + \nabla p^n = \mathbf{f}^{n+1}, \quad \mathbf{u}^{n+1}\big|_{\Gamma} = 0,$$

where $\mathbf{u}_\star^{n+1} = 2\mathbf{u}^n - \mathbf{u}^{n-1}$ is a second order extrapolation and ρ^* is equal to ρ^{n+1} or ρ^{n+2}, depending on the Strang step. Note that the equation above involves only one velocity field $\mathbf{u}$ at time t^n and t^{n+1} in this equation. The intermediate and the end-of-step velocities familiar from conventional pressure-correction methods have been consolidated to one velocity field, see [86] for details. Then we compute the pressure correction in rotational form

$$\begin{aligned} \Delta \phi^{n+1} &= \frac{\chi}{\Delta t} \nabla \cdot \mathbf{u}^{n+1}, \qquad \partial_n \phi^{n+1} = 0 \\ p^{n+1} &= p^n + \phi^{n+1} - \mu \nabla \cdot \mathbf{u}^{n+1}. \end{aligned}$$

First-order error estimates are proved, and stability of a formally second-order variant of the method is established [87]. To obtain a second-order scheme we use the three level BDF2 method. The momentum equation is then discretized as follows

$$\rho^* \left(\frac{3\mathbf{u}^{n+1} - 4\mathbf{u}^n + \mathbf{u}^{n-1}}{2\Delta t} + (\mathbf{u}_\star^{n+1} \cdot \nabla)\mathbf{u}^{n+1} \right) - \mu \Delta \mathbf{u}^{n+1} + \nabla \left(p^n + \frac{4}{3}\phi^n - \frac{1}{3}\phi^{n-1} \right) = \mathbf{f}^{n+1},$$
$$\mathbf{u}^{n+1}\big|_{\Gamma} = 0. \tag{6.24}$$

The pressure correction is evaluated by solving

$$\Delta \phi^{n+1} = \frac{3\chi}{2\Delta t} \nabla \cdot \mathbf{u}^{n+1}, \qquad \partial_n \phi^{n+1} = 0, \tag{6.25}$$

and the new pressure is finally given by

$$p^{n+1} = p^n + \phi^{n+1} - \mu \nabla \cdot \mathbf{u}^{n+1}. \tag{6.26}$$

Note that equation (6.24) holds independently for each component of the velocity field, which therefore can be computed in parallel. It is well known that the basis functions of finite element spaces for discretising the velocity and pressure fields have to be carefully chosen so that they satisfy the inf-sup, or Babuška–Brezzi condition [79]. We choose the $\mathbb{P}_2 - \mathbb{P}_1$ elements, also known as Taylor Hood elements. In view of (6.24), a finite element approximation of the density is needed. By construction, the finite volume scheme leads to three density values on each triangle of the mesh, therefore a $\mathbb{P}_1$ reconstruction is a natural choice (see [35]). Introducing the FEM nodal unknowns

$$\mathbf{u}_{h,i} = (u_{i,1}, \dots, u_{i,N_\mathbf{u}})^T \in \mathbb{R}^{N_\mathbf{u}}, \qquad i = 1, \dots, d, \tag{6.27}$$

$$p_h = (p_1, \dots, p_{N_p})^T \in \mathbb{R}^{N_p}, \tag{6.28}$$

$$\phi_h = (\phi_1, \dots, \phi_{N_p})^T \in \mathbb{R}^{N_p}, \tag{6.29}$$

what we end up with is the resolution at each time step of the algebraic problem

$$A_n \mathbf{u}_{h,i}^n = F_i^n, \qquad i = 1, \dots, d, \tag{6.30}$$

$$L \phi_h^n = F_\phi^n, \tag{6.31}$$

$$M p_h^n = F_p^n. \tag{6.32}$$

The matrices in equation (6.31) and (6.32) are respectively the stiffness and mass matrix. These matrices have size $N_p \times N_p$, they are symmetric positive definite and constant in time, thus one can compute a preconditioner once for all, while the matrix in equation (6.30) has size $N_\mathbf{u} \times N_\mathbf{u}$, it is time dependent and the non constant density implies this matrix is not symmetric, even if its symbolic structure does. The sequence $\{A_n\}$ is made of sparse and typically very large matrices that shares the same pattern and slightly differ one from the other, therefore it is desirable to take advantage of previous systems in the preconditioning strategy.

6.4 Hybrid CPU-GPU implementation

GPUs have a massively parallel architecture consisting of thousands of small, efficient cores designed for handling multiple tasks simultaneously. Each task in CUDA is referred as *thread*. Threads are organized by the programmer into blocks, which in turn are grouped into grids. Threads of the same block are split by the scheduler into *warps*, i.e. the scheduling unit which is composed by 32 threads that physically run in parallel. The threads of a block can execute concurrently, and multiple thread blocks can execute concurrently on one multiprocessor. CUDA devices have several different memory spaces, all characterized by different scopes and lifetime. Global memory is the main GPU memory, it is visible to all threads within the application (including the CPU), and lasts for the duration of the host allocation. Another invaluable device memory is called shared memory, it is visible to all threads within that block and lasts for the duration of the block. This type of memory allows for threads to communicate and share data between one another.

The kernels described in this section are written in CUDA language [109], while other algorithms such as GMRES have been accelerated by mean of NVIDIA libraries for dense and sparse parallel linear algebra [110, 111]. In order to limit memory consumption, we choose to store the matrices in compressed sparse row (CSR) format. Each matrix is composed by three arrays: *data* is the array of corresponding nonzero values; *indices* is the array of column indices; *indptr* points to row starts in *indices* and *data* arrays.

In GPGPU the device is often used as a co-processor for computations well suited to parallelization. Here, we follow this paradigm. As shown in Figure 6.3, data are initialized on the CPU and then GPU handles all the computations. The final result is sent back to the CPU for post processing. Each Strang step is composed by a Finite Volume step to obtain the new density and a Finite Element step to obtain velocity and pressure. The former can be solved with perfect parallelism, since it is possible to reconstruct the numerical fluxes independently for each grid point and then to recover the now solution by adding all flux variations. The latter requires at each time step the assembly of the time dependent matrix and the corresponding right-hand side to obtain the velocity components, which are independent on each other and therefore can be computed in parallel, and then the resolution of two more linear systems in order to recover the pressure. The control flow below shows the intrinsic sequential dependencies of the computation, while each block represents a parallel task.

For what concerns Algorithm 11, from now on we will assume that $\mathcal{S}$ is the pattern of the sequence of matrices (6.30). This way, we will need only one copy of the *indices* and *indptr* arrays in the implementation, with an important memory saving. Furthermore, the matrix sum in Algorithm 11 can be computed by simply adding the related *data* arrays. The only left issue is the parallel computation of

$$R = (A - \overline{\mathrm{tril}}(M)\mathrm{triu}(M)) \odot S. \tag{6.33}$$

Storing the triangular factors L and U in the same CSR matrix M helps to achieve a more efficient memory access pattern in matrix multiplication and then to exploit the data spatial locality. Moreover, Reverse Cuthill-Mckee algorithm also enhances this property. The computation (6.33) is handled by a single CUDA kernel that we designed on row-wise matrix product. Consider the matrix product $X = YZ$, where Y has size $n \times m$ and Z has size $n \times l$. For a matrix X, denote by $X(i,:)$ the i-th row of X and by $\mathcal{I}_X(i)$ the set of column indices

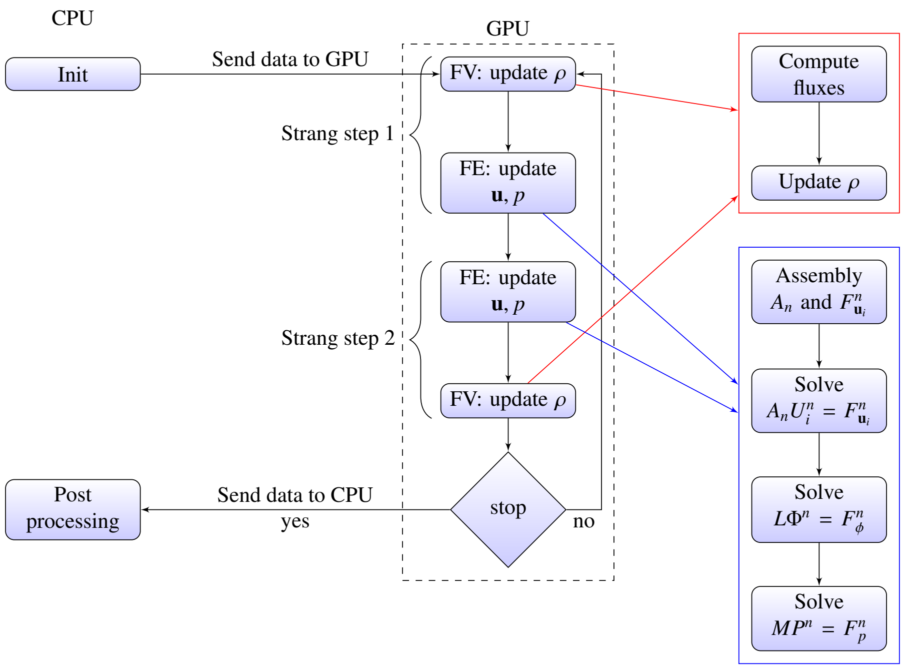


Figure 6.3: Hybrid CPU-GPU program flow chart.

corresponding to non-zero elements in the i-th row of X. Then we have

$$X(i,:) = \sum_{k\in\mathcal{I}_Y(i)} Y_{ik} Z(k,:), \qquad i = 1,\ldots,m. \tag{6.34}$$

Coming back to equation (6.33), now all rows of the residual matrix R are independent of each other and each one can be assigned to a *warp*. Thus warp i performs the following computation

$$R(i,:) = A(i,:) - \sum_{k\in\mathcal{I}_{\overline{\mathrm{tril}}(M)}(i)} \overline{\mathrm{tril}}(M)_{ik}\,\mathrm{triu}(M)(k,:). \tag{6.35}$$

Each row is incrementally computed in shared memory and then copied back to global memory. The i-th *warp* computes the sum in (6.35) as a sequential loop over the rows with index in $\mathcal{I}_{\overline{\mathrm{tril}}(M)}(i)$, and other two shared array structures are used to cyclically load the related *data* and *indices* values. Each *warp thread* processes an entry of the corresponding row of the outcoming matrix. In order achieve the correct result, each *thread* needs to read exactly once the currently loaded row of $\mathrm{triu}(M)$. This operation is done in a cyclic fashion: each *thread* first reads the value corresponding to its lane index (the local index of the thread inside the warp), and then reads the successive elements, starting back from the beginning if the index falls outside the row. This way there is no concurrency in shared memory reads. For what concerns read/write operations between global and shared memory, the CSR format ensures (partially) coalesced memory accesses. The resulting pseudo-code is shown in Algorithm 14.

In practice many optimizations can be added to this computation, for instance by exploiting warp synchronization, which is anyway an hardware concept and has to be handled with care. For this reason the algorithm presented is limited to a matrix structure that admits only 32 non zeros per row. In many cases it is sufficient and this bound is not a restriction, however with slight modifications one can make a *warp* process a row with a number of entries up to a multiple of 32.

6.5 Numerical simulations

In this section, we report numerical results confirming the expected simulation speed-up. The system of equations (6.18)-(6.20) can be useful to model Newtonian multi-fluid flows, that are used extensively from microfluidics to industrial-scale applications. To this purpose, we rewrite the system in the following dimensionless form

$$\begin{aligned}
&\partial_t \rho + \nabla(\rho \mathbf{u}) = 0,\\
&\rho\left(\partial_t \mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) - \frac{1}{Re}\Delta\mathbf{u} + \nabla p = \frac{1}{Fr^2}\rho\mathbf{g},\\
&\nabla\cdot\mathbf{u} = 0,
\end{aligned} \tag{6.36}$$

where we introduced the following parameters:

- the Reynolds number $Re = UL\rho(\mu)^{-1} = UL/\nu$;
- the Froude number $Fr = U(GL)^{-1/2}$.

Here U, L, ρ are the velocity, length and density reference values respectively, G is the magnitude of the gravitational field $\mathbf{g}$ and finally μ and ν are respectively the dynamic and kinematic viscosity of the fluid. As stated in Section 6.3, existence and uniqueness hold as long as equations (6.36) are coupled with proper boundary and initial conditions. The Atwood number is defined as

$$At = \frac{\max_\Omega \rho_0 - \min_\Omega \rho_0}{\max_\Omega \rho_0 + \min_\Omega \rho_0}, \tag{6.37}$$

according to Tryggvason's definition [139]. In what follows we aim to describe the problem sensitivity and difficulty with various Reynolds number, ranging from 10^{-2} to 10^4, and the consequent behaviour and performances of the iterative solvers used. The tests we next discuss involve the modelling of flows at millimetric and micrometric scales, characterized by low Reynolds number (typically $Re \ll 1$), but also flows at moderate-high Reynolds number, characterized by a turbulent motion. In both cases we experiment a wide range of Atwood numbers. As the fluid is more viscous $Re \to 0$, and its dynamic tends to that of a rigid body. On the other hand, the fluid becomes heavier and heavier when the Atwood number grows. These situations lead to significant differences in the matrices describing the problem.

6.5.1 Test environment

The computations are performed on a NVIDIA graphic card GeForce GTX 1060 with 2560 CUDA cores and two other more performing GPUs, namely a Tesla V100 with 5120 CUDA cores and a Titan Xp with 3840 CUDA cores. Computations have sometimes been repeated on different GPUs, giving absolute timings quite different, due to the different computing power of the boards, but confirming the conclusions given in this chapter. Comparing the different boards is outside of the scope of this work, but using different GPUs we further verified that the results are independent from the choice of GPU. The kernels are written in CUDA language [109], and NVIDIA libraries CUBLAS [110] and CUSPARSE [111] for dense and sparse parallel linear algebra are used. All computations are performed in double precision arithmetic.

In this section we will focus on the solution of systems (6.30). The iterative solver used is GMRES with restart parameter 30, the relative residual tolerance is set to 10^{-10} and the maximum number of iterations is 3000. When not stated otherwise, the triangular systems in the preconditioner application of LU-type methods are solved with CUSPARSE direct solver based on level scheduling [108]. CUSPARSE library is also used to compute ILU(0) decomposition. All SITALU instances are initialized with an ILU(0) decomposition. If not stated otherwise, the preconditioner is updated at every time step. In what follows only 3 sweeps of Jacobi and block Jacobi methods are applied in the solution of triangular systems, resulting in a satisfactory preconditioning efficiency in most cases.

6.5.2 Convergence tests

First, we test the parallelized algorithm on a problem with a known analytical solution [85]

$$\begin{cases} \rho_{\text{ex}}(t,r,\theta) & = \rho_1(r,\theta-\sin t), \\ \mathbf{u}_{\text{ex}}(t,x,y) & = \begin{pmatrix} -y\cos t \\ x\cos t \end{pmatrix}, \\ p_{\text{ex}}(t,x,y) & = \sin x \sin y \sin t, \end{cases} \tag{6.38}$$

where (r,θ) are the polar coordinates on $\mathbb{R}^2$, and $\rho_1(r,\alpha)$ in an arbitrary function. Here, we used $\rho_1(r,\alpha) = 2 + r\cos\alpha$, thus the density field in Cartesian coordinates becomes $\rho_{\text{ex}}(t,x,y) = 2 + \cos(\sin t)x + \sin(\sin t)y$. The fields $\rho_{\text{ex}}(t,x,y)$ and $\mathbf{u}_{\text{ex}}(t,x,y)$ satisfy the mass conservation equation identically and $\mathbf{u}_{\text{ex}}(t,x,y)$ is solenoidal. The momentum equation is automatically satisfied by the body force defined by

$$\mathbf{f}_{\text{ex}}(t,x,y) = \begin{pmatrix} (y\sin t - x\cos^2 t)\rho_{\text{ex}}(t,x,y) + \cos x \sin y \sin t \\ -(x\sin t + y\cos^2 t)\rho_{\text{ex}}(t,x,y) + \sin x \cos y \sin t \end{pmatrix}.$$

We solve the Navier-Stokes equations (6.36) on the square

$$\Omega = \left\{(x,y) \in \mathbb{R}^2 : |x| < 1, |y| < 1\right\}.$$

We solve the above mentioned problem up to $T = 1$. Following [35], the time step is chosen as $\Delta t = h^{3/2}$, where the meshsize h is chosen small enough so that the consistency error in space is significantly smaller than that in time. For the velocity and the pressure, the errors are respectively evaluated using the usual $L^2(\Omega)$ norms $\|\mathbf{u}_{\text{ex}} - \mathbf{u}_h\|_2$ and $\|p_{\text{ex}} - p_h\|_2$. For the density, the error is evaluated with the usual $L^1(\Omega)$ norm $\|\rho_{\text{ex}} - \tilde{\rho}_h\|$, where $\tilde{\rho}_h$ is a $\mathbb{P}_1$ reconstruction of the FV density obtained with the procedure described in [35]. We plot in Figure 6.4 the error on the velocity, pressure and density. It shows the maximum error in time evaluated in $L^2(\Omega)$ (for velocity and pressure) or $L^1(\Omega)$ (for density) norm with respect to the meshsize h. The lines corresponding to a rate of convergence of order two (Slope 2) and order three halves (Slope $\frac{3}{2}$) are also displayed.

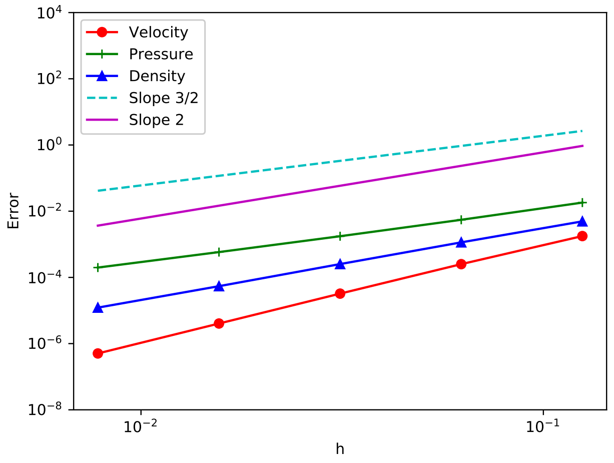


Figure 6.4: Rates of convergence.

As we see, the time discretization error is second order for the velocity field and slightly less for the density. The error on the pressure is of order $\frac{3}{2}$, rather than second order, in fact the boundary of the domain is not smooth but only piecewise smooth, according to [86].

6.5.3 Dual-fluid flow

We now illustrate the performances of the methods presented on a realistic problem. Two fluids occupy the rectangular domain $\Omega = (-d/2, d/2) \times (-2d, 2d)$ and they are initially at rest with the heavier one superposed to the lighter one. A downward gravitational field acts on the flow. We are interested in the study of a large range of Reynolds number, varying from 10^{-2} to 10^4. This test models a Rayleigh-Taylor instability with medium-high Reynolds number, while with low and very low Reynolds number it describes a bi-fluid laminar flow, characterized by small velocities and high viscosity, typical in microfluidics applications. The interface between the fluids is expressed by the initial condition

$$\rho_0(x, y) = \frac{\rho_{\max} + \rho_{\min}}{2} + \frac{\rho_{\max} - \rho_{\min}}{2} \tanh\left(\frac{y - \eta\cos(2\pi x/d)}{0.01d}\right), \tag{6.39}$$

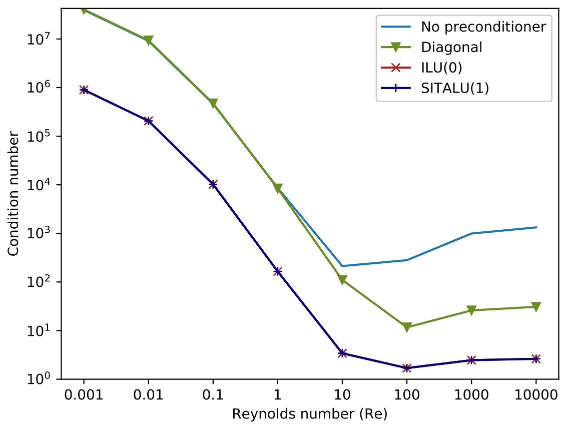
where $\rho_{\min}, \rho_{\max} > 0$ are the densities of the light and heavy fluid respectively, and $\eta > 0$ is the amplitude of the initial perturbation. The Atwood number is then

$$At = \frac{\rho_{\max} - \rho_{\min}}{\rho_{\max} + \rho_{\min}}, \tag{6.40}$$

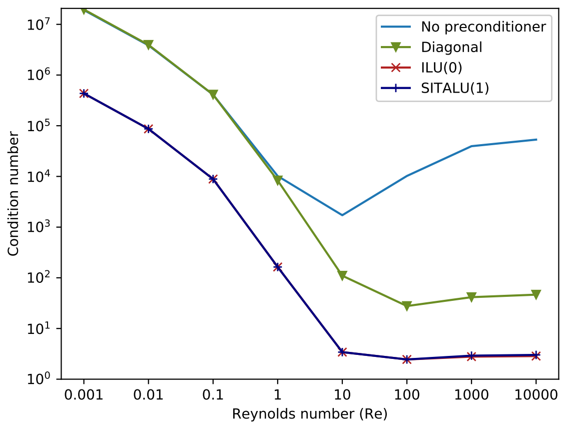
and it expresses the density ratio between the two fluids. The right hand side of the momentum equation is equal to $\rho\mathbf{g}$, where $\mathbf{g}$ is a vertical downward gravitational field of magnitude G. Navier-Stokes equations have been adimensionalized with the reference values $\rho_{\min}$ for the density, d for lengths, $\sqrt{dG}$ for time. Then Reynolds number is defined as $Re = \rho_{\min} d^{3/2} G^{1/2}/\mu$. The problem is supplemented with no-slip boundary conditions on the horizontal boundaries and symmetry is imposed on the two vertical sides

$$\begin{array}{lll} \text{horizontal boundary} & u = 0, & v = 0, \\ \text{vertical boundary} & u = 0, & \partial_x v = 0, \end{array} \tag{6.41}$$

where the velocity was written in components $\mathbf{u} = (u, v)$. Figure 6.5 shows how the condition number of the matrix of the linear system (6.30) changes in function of the Reynolds number for different values of the density ratio on a uniform mesh 10×40. The problem's ill-conditioning looks to reach a saturation level at both extremes of Reynolds number, being the condition number at low Re higher then the respective value at high Re. Notice that the condition number at low Reynolds number reaches the same values for both low Atwood number (Figure 6.5a) and high Atwood number (Figure 6.5b), while at high Reynolds number the condition number increases as the density ratio grows. As wee see, a simple diagonal preconditioner can reduce the ill-conditioning at high Reynolds number, independently from the density ratio, while it has no effect at low Reynolds number. For what concerns ILU(0) preconditioner, its effectiveness is much stronger at high Reynolds number, but it still performs well at low Reynolds number. Finally, the SITALU(1) preconditioner performs as well as ILU(0), meaning that it achieves its optimal behaviour. Beside the



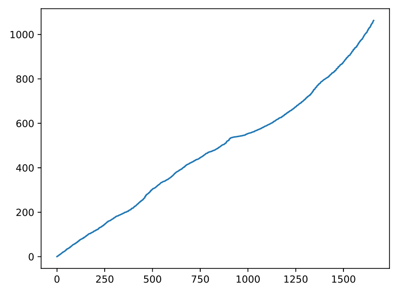
(a) Density ratio 3.



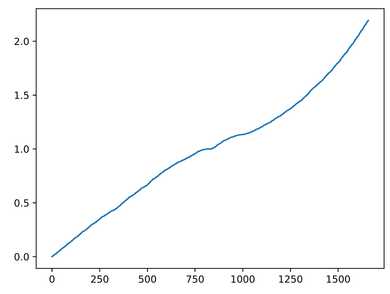
(b) Density ratio 19.

Figure 6.5: Condition number in function of the Reynolds number at different density ratios.

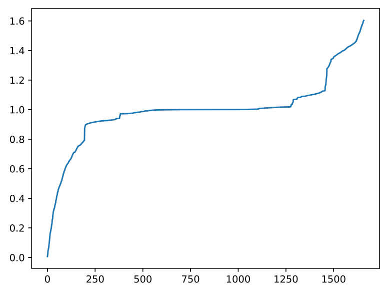
condition number, it is known that clustering eigenvalues also affects GMRES convergence [95]. Figure 6.6 shows the distribution of the eigenvalues of the matrix without preconditioning (Figure 6.6a), with the diagonal preconditioner (Figure 6.6b) and with the SITALU(1) preconditioner (Figure 6.6c). The test case here considered has density ratio 3 and Reynolds number $Re = 0.01$ on a uniform mesh 10×40. The matrix without preconditioner has no cluster of eigenvalues (Figure 6.6a). The matrix after the application of a simple diagonal preconditioner has a significantly smaller spectral radius (Figure 6.6b). However, this preconditioning technique is effective in scaling the eigenvalues, but fails in clustering. On the other hand, the matrix coupled with the SITALU(1) preconditioner has a cluster close to one (Figure 6.6c), improving the convergence of Krylov subspace methods. In the following, different tests are per-



(a) No preconditioner.



(b) Diagonal preconditioner.



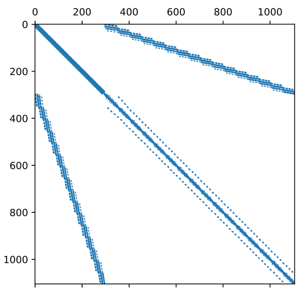
(c) SITALU(1) preconditioner.

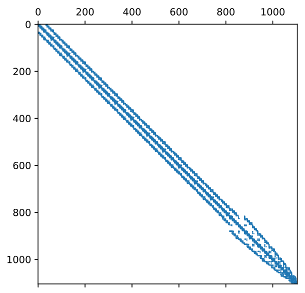
Figure 6.6: Distribution of the eigenvalues with different preconditioners.

formed in order to measure the efficiency of each method presented in terms of number of iterations to convergence and execution times. It turns out that the iterative solution of triangular systems is crucial to get an overall significant time saving and a competitive time-to-solution. In fact, the execution times for ILU(0) and ITALU(1) preconditioners are badly inflated because of the direct solution of triangular systems by level-scheduling technique, resulting in completely uncompetitive performances, thus the computation times for these methods are not shown in the figures below. However, Table 6.4 shows the mean performances of ILU(0) and ITALU(1) preconditioners in relation to the other methods investigated on the dual-fluid low test case with density ratio 3 on a uniform mesh 30×240. In particular, we list the mean number of iterations required by GMRES for convergence (Avg. iter. number) and mean execution times in seconds for solving the linear system and updating the preconditioner (Avg. sol. time (s)). For each case, the reported numbers are average values over $T = 4.0$. The minimum time is highlighted in bold.

6.5.3.1 About matrix blocking

As pointed out in Section 6.1.2, blocking may improve the robustness of Jacobi solvers for incomplete factors, but it highly depends on the chosen structure of the block diagonal matrix and on the norm of the inverses of such diagonal blocks. In the present work, the sparsity pattern of the matrices arising in the dual-fluid flow simulation is shown in Figure 6.7a, while Figure 6.7b shows the outcoming pattern after applying RCM algorithm. As we see, RCM ordered pattern clearly reveals to be somehow regular and the matrix entries are grouped in three narrow bands, therefore we choose to extract a block structure in correspondence of the diagonal band, as shown in Figure 6.7c.


(a) Natural ordering.


(b) RCM ordering.

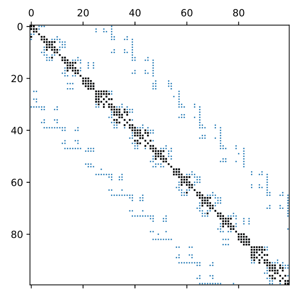

(c) Matrix blocking, detail.

Figure 6.7: Sparsity patterns.

Let us denote by B the diagonal band shown in Figure 6.7b. Then the heuristic algorithm used is the following

Algorithm 15 Matrix blocking

```
    Inputs: $B$
    Outputs: $\mathcal{S}$
 1: $s, r = 0$
 2: repeat
 3:     while the $r$-th row of tril($B$) has more than 2 non zero entries do
 4:         $r = r + 1$;
 5:     end while
 6:     for $(i, j) \in \mathcal{S}$ s.t. $s \leq i \leq r$ do
 7:         if $j < s$ or $j > r$ then
 8:             Delete $(i, j)$
 9:         end if
10:     end for
11:     $s = r + 1$
12: until all rows have been processed
```

Note that the algorithm above maximizes the size of the extracted diagonal blocks within the diagonal band B. We specify that this choice does not take into account the magnitude of the matrix entries and thus it may not be the most efficient one. When natural ordering is used, the blocking should be chosen such that pressure and velocity variables at a grid point form a block. However, such a blocking is not possible when RCM is used. For a general sparse matrix, finding the optimal blocking is not an easy task and this should be done according to the matrix structure. It is outside the scope of the present work to give an exhaustive treatment to this issue. For completeness, we limit ourselves to briefly mention some of the most common blocking techniques. The simplest way to impose such a blocking is to divide the rows into the requested number of blocks such that each block contains (approximately) the same number of rows. Other techniques are based of graph partitioning, such as nested dissection [78] or supervariable blocking [49]. Another possibility, in contrast with traditional bandwidth reduction techniques, consists in taking into account the magnitude of the matrix entries, bringing the heaviest elements closer to the diagonal, see [104].

6.5.3.2 At low Reynolds number

We investigate the dual fluid flow dominated by viscous effects, i.e. we have $Re \ll 1$, for a wide range of Atwood numbers. The computations are performed on a uniform mesh 30×240, see Table 6.3. For the initial condition we set $\eta = 0.1$. The time evolution of the density field at density ratio 100 ($\rho_{\max} = 100$, $\rho_{\min} = 1$ and $At \approx 0.98$) and Reynolds number $Re = 10^{-2}$ is shown in Figure 6.8 at times 1.0, 25.0, 50.0, 75.0, 100.0, 125.0, 150.0, 175.0 and 200.0. The motion of the fluid is very slow due to the high viscosity induced by the very small Reynolds number. In such case, the Atwood number does not affect the difficulty of the simulation, but only the rate of the motion, which is indeed faster when the superposed fluid is significantly heavier. Anyway the small Reynolds number prevents the numerical velocity to reach high values.

Figure 6.9 shows the iteration number needed by GMRES to converge (Figure 6.9a) and the corresponding execution time (Figure 6.9b) per time step. The diagonal preconditioner is ineffective in this case and do not allow GMRES to converge, therefore it is not shown in Figure 6.9. ILU(0) preconditioner is still effective, even if the iteration number is high. In this test settings, SITALU(1) (one iteration of the SITALU algorithm) is sufficient to get the same performances of an ILU(0) decomposition. Figure 6.9b shows that SITALU(1)+Jacobi(3) is the fastest time-to-solution method. The performance of SITALU(1)+block-Jacobi(3) is worst than its scalar counterpart, probably because the norm of the off-diagonal blocks is not small enough. This behaviour however depends on the Reynolds number: as it grows, block Jacobi gains accuracy against scalar Jacobi (see Table 6.4). In these settings, it is possible

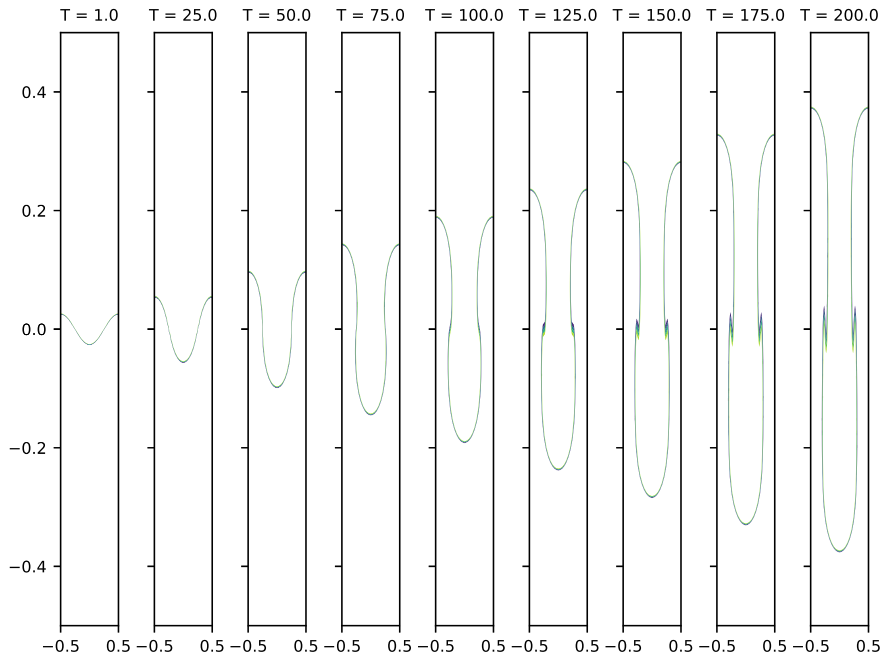
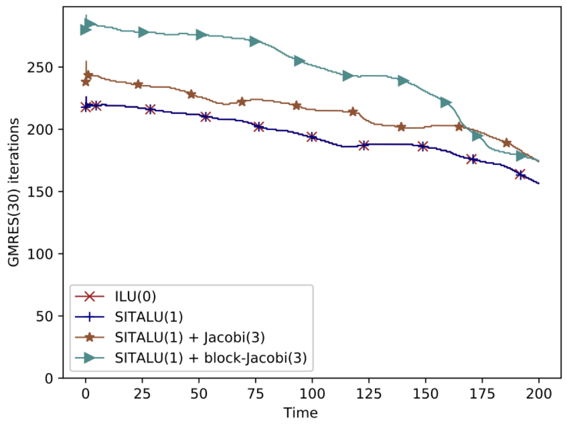
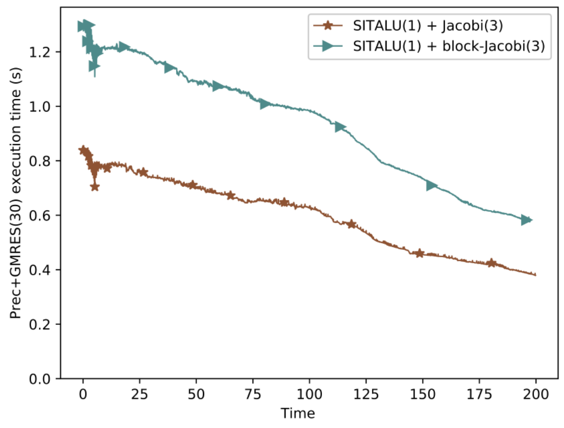


Figure 6.8: Evolution of the density contours $40.0 \leq \rho \leq 60.0$, density ratio 100, Reynolds number $Re = 0.01$.

to reuse the preconditioner instead of performing an update at each time step. Table 6.2 summarizes the performance of different updating strategies for the ILU preconditioner: for each of the methods listed, it is shown the mean number of iterations required by GMRES for convergence (Avg. iter. number) and mean execution times in seconds for solving the linear system and updating the



(a) Iteration number per time step.



(b) Execution time per time step.

Figure 6.9: Iteration number and execution times, density ratio 100, Reynolds number $Re = 0.01$.

preconditioner (Avg. sol. time (s)) per time step when the preconditioner is updated every k time steps. The stated values are average values over a simulation up to the physical time $T = 4.0$. The number of iterations required by GMRES slightly grows with k, but the execution times do not shorten, confirming that the SITALU implementation here provided is indeed inexpensive. Moreover, increasing the number of iterations of SITALU do not prevent the growth of GMRES iterations.

Preconditioner	k	Avg. iter. number	Avg. sol. time (s)
SITALU(1)	2	216.30	0.3840
SITALU(1)	3	216.33	0.3849
SITALU(1)	4	216.30	0.3949
SITALU(1)+Jacobi(3)	2	239.64	0.2730
SITALU(1)+Jacobi(3)	3	239.65	0.2733
SITALU(1)+Jacobi(3)	4	239.66	0.2739

Table 6.1: Performance of different updates of ILU preconditioner, density ratio 100, Reynolds number $Re = 0.01$.

6.5.3.3 At moderate and high Reynolds number

We now focus on the study of the convection-dominated case, i.e. $Re \gg 1$, for a wide range of Atwood numbers. This is the case of the well known Rayleigh-Taylor instability [139]. The computations are performed on a uniform mesh 50×440, see Table 6.3. The time evolution of the density field at density ratio 3 ($\rho_{\max} = 3$, $\rho_{\min} = 1$ and $At = 0.5$) and Reynolds number $Re = 20000$ is shown in Figure 6.10 at times 1.0, 1.5, 2.0, 2.5, 3.0, 3.25, 3.5, 3.75 and 4.0. Some spurious vortices seem to start near the vertical boundary for $T \geq 2$ and they obviously grow with time, as already noticed in [35]. This would require some form of stabilization, e.g. streamline-upwind/Petrov-Galerkin (SUPG) [30] or subgrid methods [88].

Figure 6.11 shows the evolution in time of the iteration number for convergence and the corresponding execution times. Being particularly suited for parallel computation, the diagonal preconditioner performs well for what concerns the execution time despite a higher iteration number for convergence with respect to LU-type preconditioners. Figure 6.11a shows that LU-type preconditioners (ILU(0), SITALU(1), SITALU(1)+Jacobi(3) and SITALU(1)+block-Jacobi(3)) lead to the same convergence rate of GMRES on this test. The execution times in Figure 6.11b shows that SITALU(1)+Jacobi(3) outperforms the other methods. The diagonal preconditioner's performances get worst when $T \geq 1.5$, this phenomenon is due to the fact the difficulty of the simulation grows as the interface between the fluids starts to show turbulence whirlpools more and more complex. SITALU(1)+block-Jacobi(3) is clearly affected by direct triangular solve at each iteration, therefore its time performance are poorer respect to SITALU(1)+Jacobi(3). The results are similar if we choose a moderate Reynolds number, e.g. $Re = 1000$ or $Re = 5000$.

Table 6.2 summarizes the performance of different updating strategies for the ILU preconditioner: for each of the methods listed, it is shown the mean number of iterations required by GMRES for convergence (Avg. iter. number) and mean execution times in seconds for solving the linear system and updating the preconditioner (Avg. sol. time (s)) per time step when the preconditioner is updated every k time steps. The stated values are average values over a simula-

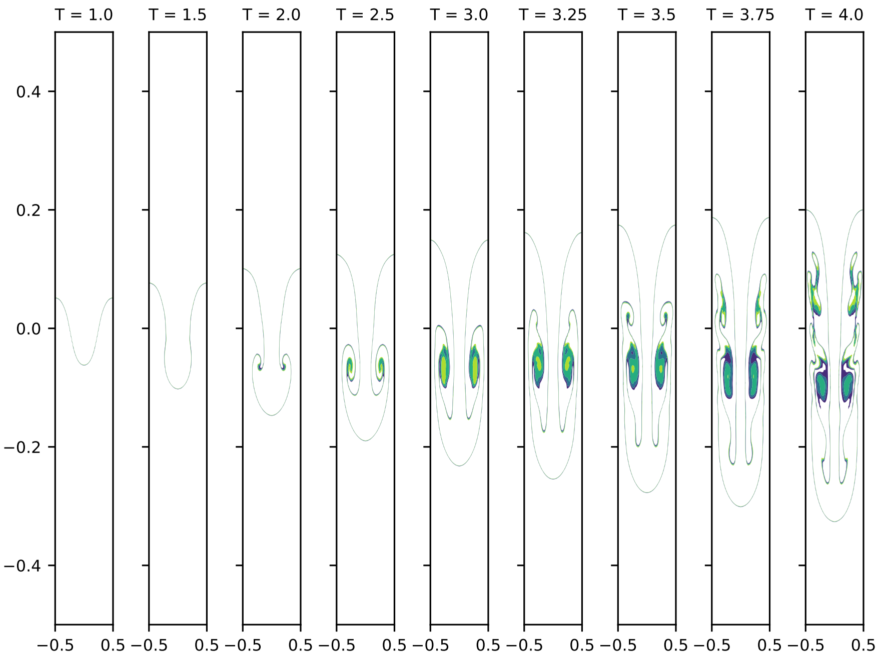
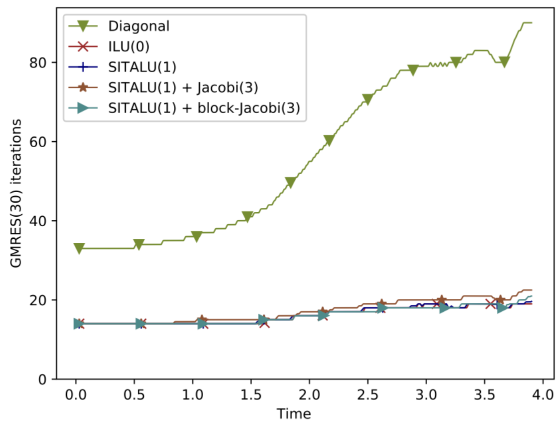


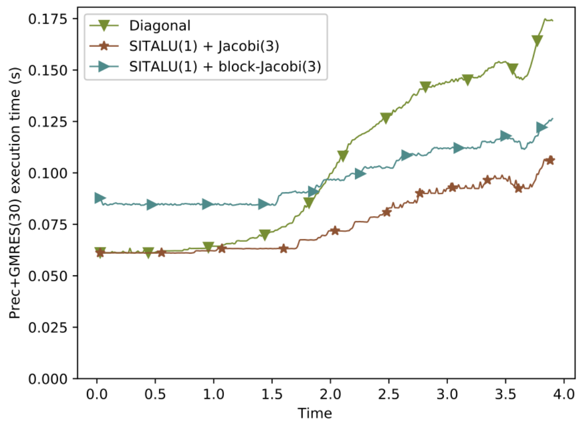
Figure 6.10: Evolution of the density contours $1.4 \leq \rho \leq 1.6$, density ratio 3, Reynolds number $Re = 20000$.

tion up to the physical time $T = 4.0$. Again, the number of iterations required by GMRES slightly grows with k, and the execution times stay approximately unchanged. As in the low Reynolds number case, increasing the number of iterations of SITALU do not prevent the growth of GMRES iterations.

Figure 6.12 shows the time evolution of the density field at density ratio 19 ($\rho_{\max} = 19$, $\rho_{\min} = 1$ and $At = 0.9$) and Reynolds number $Re = 1000$ at times 1.0, 1.5, 2.0, 2.5, 3.0, 3.25 3.5 and 3.75. The results are qualitatively similar to those in [35]. As the density ratio grows, for $T \geq 3.5$ we can observe the appearance of a peak in the iteration number highlighted by a dashed line in Figure 6.13. This behaviour exposes an instability phenomenon related to the physics of this problem: the high density ratio and the high Reynolds number together make the heavier fluid reach the bottom of the domain with particularly high velocity. As a consequence, the pressure gradient reaches high values in correspondence of the interface, see Figure 6.14b. Indeed, the instability is also revealed by the evolution in time of the divergence of the velocity field, which explodes just in correspondence of the impact. Notice that the divergence is non-zero even before the heavier fluid reaches the bottom: this is typical of projection schemes (see [84] for an overview), where the end-of-step divergence free velocity is eliminated and only an intermediate velocity field needs to be



(a) Iteration number per time step.



(b) Execution time per time step.

Figure 6.11: Iteration number and execution times, 3, Reynolds number $Re = 20000$.

computed. LU-type preconditioners with direct solution of triangular systems (here ILU(0) and SITALU(1)) don't seem to be affected, while iterative methods are significantly more sensitive: indeed, SITALU(1)+Jacobi(3) fails to make GMRES converge around the peak and the solution explodes. In such a case, restarting the preconditioner by recomputing an ILU(0) decomposition from scratch does not solve the instability. Instead, we restore the convergence of GMRES by increasing STALU iterations, e.g. using SITALU(5)+Jacobi(3). Anyway, increasing the number of iterations also implies an additional time

Preconditioner	k	Avg. iter. number	Avg. sol. time (s)
SITALU(1)	2	17.1	0.3900
SITALU(1)	3	17.2	0.3890
SITALU(1)	4	17.5	0.3952
SITALU(1)+Jacobi(3)	2	18.3	0.0770
SITALU(1)+Jacobi(3)	3	18.4	0.0769
SITALU(1)+Jacobi(3)	4	18.7	0.0776

Table 6.2: Performance of different updates of ILU preconditioner, density ratio 3, Reynolds number $Re = 20000$.

cost, making the method loose its gain against the diagonal preconditioner, as shown in Figure 6.13b. As one can expect, in this case is is not possible to reuse the preconditioner, even if additional iterations of SITALU are performed.

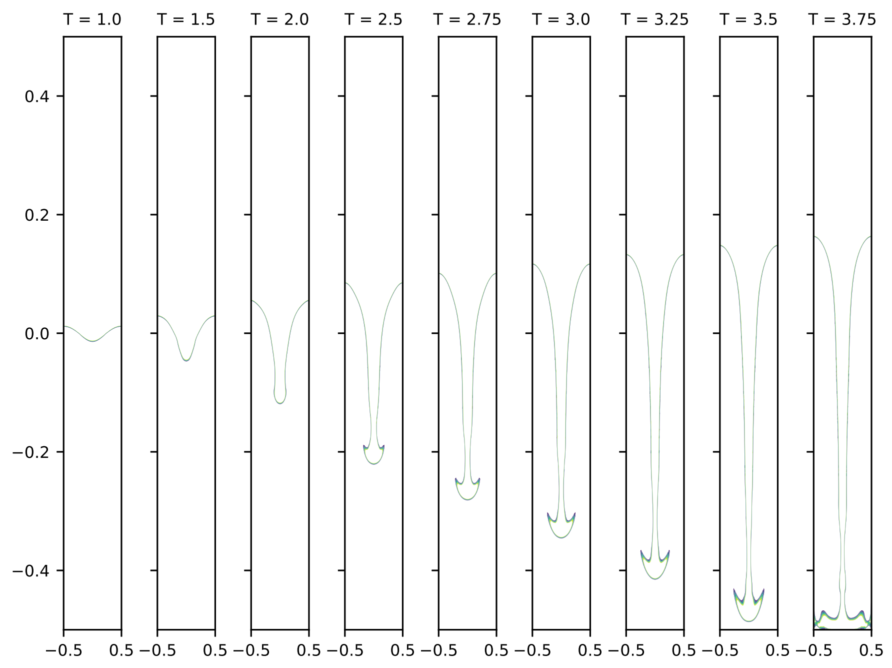
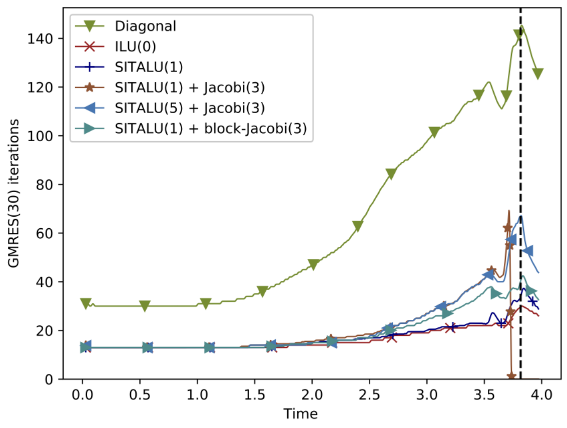
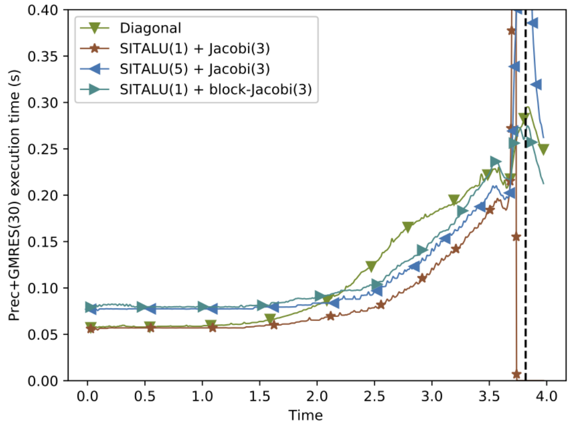


Figure 6.12: Evolution of the density contours $8.0 \leq \rho \leq 12.0$, density ratio 19, Reynolds number $Re = 1000$.

In order to better understand the problem, Figure 6.15 shows the evolution of the minimum and maximum singular values (denoted as `sigma_min` and `sigma_max` resp.) of the L and U factors updated with SITALU(1) (Figure 6.15a)



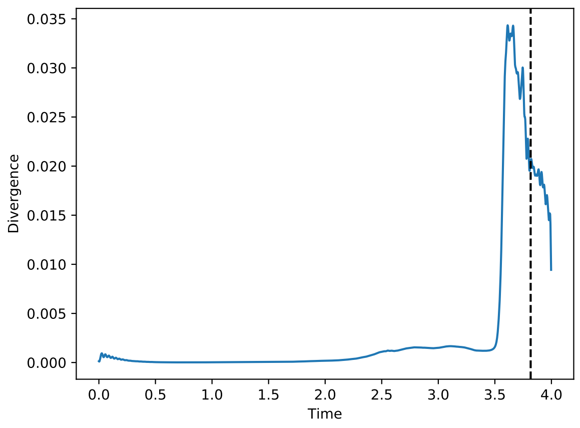
(a) Iteration number per time step.



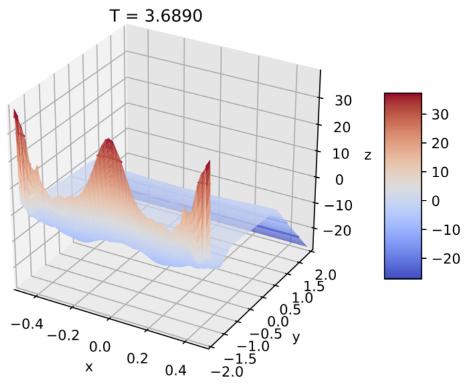
(b) Execution time per time step.

Figure 6.13: Iteration number and execution times, density ratio 19, Reynolds number $Re = 1000$.

and SITALU(5) (Figure 6.15b). As we see, the minimum singular values of the L and U factors updated with SITALU(1) oscillate in a spurious way until the solution explodes. Some additional iteration of SITALU can reduce the amplitude of the oscillations. This instability phenomenon is known in dual fluid flow simulations: Coppola-Owen and Codina [50], for instance, proposed to enrich the finite element approximation space for the pressure, in order to enable the pressure gradient to be discontinuous at the interface.



(a) Evolution of the divergence of the velocity field.

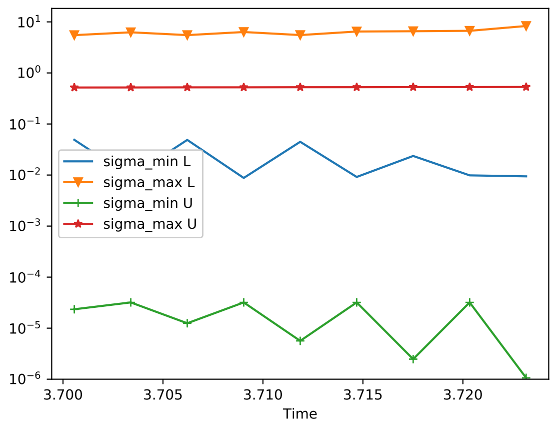


(b) Pressure field.

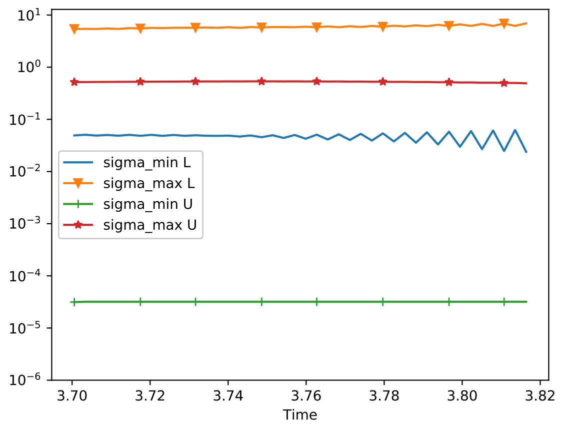
Figure 6.14: Instability phenomenon, density ratio 19, Reynolds number $Re = 1000$.

6.5.4 Tradeoff between accuracy and efficiency

In the previous section we tried to investigate the interplay between the Reynolds and Atwood numbers, two adimensional quantities that govern the main physical behaviour in the density dependent Navier-Stokes problem. In order to better estimate the performance of each method in relation to the physics of this problem, we test the SITALU algorithm on the analytical solution (6.38) at different Reynolds and Atwood numbers. In order to make the Atwood num-



(a) SITALU(1) preconditioner.



(b) SITALU(5) preconditioner.

Figure 6.15: Evolution in time of the minimum singular and maximum singular values of the factors of the updated ILU preconditioner.

ber significantly change, we choose $\rho_1(r,\alpha) = 2 + r\cos\alpha + Ce^{(-r^2)}$, where C is a constant. The density field in Cartesian coordinates becomes $\rho_{\text{ex}}(t,x,y) = 2 + \cos(\sin t)x + \sin(\sin t)y + Ce^{(-x^2-y^2)}$ and it verifies the mass conservation equation together with the velocity field defined in (6.38). The momentum equation is verified as well. In these test we measure the maximum error in time over $T = 1$, using L^2 norm for velocity and pressure and L^1 norm for the density. The spatial discretization is chosen fine enough so that the consistency error in space is significantly smaller than that in time.

Mesh	Mesh size (h)	Time step (Δt)	Triangles	$\mathbb{P}_2$ nodes ($N_{\mathbf{u}}$)	$\mathbb{P}_1$ nodes (N_p)
10×40	0.1	0.0316	800	1701	451
20×140	0.05	0.0112	5600	11521	2961
30×240	0.033	0.0061	14400	29341	7471
40×340	0.025	0.0040	27200	55161	13981
50×440	0.028	0.0029	44000	88981	22491

Table 6.3: Mesh properties.

Table 6.5 shows the mean number of iterations required by GMRES for convergence (Avg. iter. number) and mean execution times in seconds for solving the linear system and updating the preconditioner (Avg. sol. time (s)) per time step, and the maximum error in time (Error $(\mathbf{u}, p, \rho)$) at different values of Reynolds number. For each case the reported numbers are average values over $T = 1.0$. We set $C = 0.0$, corresponding to $At = 0.5$. The simulation is performed on a uniform mesh 128×128 on $\Omega = [-1, 1]^2$. The errors are clearly independent from the method used, as well as from the Reynolds number.

Table 6.6 shows the mean number of iterations required by GMRES for convergence (Avg. iter. number) and mean execution times in seconds for solving the linear system and updating the preconditioner (Avg. sol. time (s)) per time step, and the maximum error in time (Error $(\mathbf{u}, p, \rho)$) at different values of Atwood number. For each case the reported numbers are average values over $T = 1.0$. Here the Reynolds number is $Re = 1$, and the simulation is performed on a uniform mesh 128×128 on $\Omega = [-1, 1]^2$. Again, the errors are independent from the method used and from the Atwood number.

6.6 Summary

In the present work we have explored the applicability and efficiency of Jacobi iterative methods applied to the solution of sparse triangular systems arising from the incrementally updated ILU decomposition for the parallel simulation of the variable density Navier-Stokes system. It turns out that a fully iterative approximation of an ILU preconditioner is able to get the best performance on a highly parallel architecture like a GPU. We have choosen a fixed number of sweeps for Jacobi and block Jacobi, based on a quite long sperimentation. This choice is indeed advantageous, as the preconditioner is then a fixed operator and a flexible solver, e.g. FGMRES [121], is not needed. Alternatively, with a flexible solver the number of sweeps could be adjusted dynamically: for instance, the number of Jacobi sweeps for each approximate triangular solve could be tuned on the residual norm reduction of each Krylov iteration.
The SITALU algorithm turns out to be an efficient method for updating an ILU decomposition and the implementation provided is very inexpensive. Recomputing from scratch is costly and often not needed, furthermore it is not a cure for instabilities. For what concerns the update schedule, it is possible to reuse

the same preconditioner without updating it at each time step, but it leads to a slight increase in the number of iteration of GMRES with no global benefit to the execution times.

The SITALU technique coupled with Jacobi methods is a flexible fully iterative algorithm for the update and the application of ILU preconditioners within highly parallel computers.

Algorithm 14 Residual matrix computation

```
1: Let $\mathbf{data_A}$ be the data array of $A$.
2: Let $\mathbf{data_M}$ be the data array of $M$.
3: Let **indices** be the common indices array.
4: Let **indptr** be the common indptr array.
5: Let $\mathbf{s\_data_R}$ be the auxiliary shared array containing the data array of $R$.
6: Let $\mathbf{s\_data_M}$ be the auxiliary shared array containing the data array of $M$.
7: Let **s_indices** be the auxiliary shared array containing the indices array.
8: Let $\mathbf{val_L}$ be the shared array of index of currently processed entry of $L$.
9: Let $\mathbf{row_U}$ be the shared array of index of currently processed row of $U$.
10: Let $\mathbf{data_R}$ be the data array of $R$.
11: Let $w\_id$ be the global *warp* index.
12: Let $l\_id$ be the local thread index within its *warp*.
13: Let $val$ be the value of the element of tril($M$) assigned to each *thread*.
14: Let $col$ be the column index of the entry of tril($M$) assigned to each *thread*.
15: Let $nnz$ be the number of entries in the row $w\_id$ that lie in tril($M$).
16: $\mathbf{s\_data_R} \leftarrow \mathbf{data_A}[\mathbf{indptr}[w\_id] : \mathbf{indptr}[w\_id + 1]]$   ▷ Each *warp* loads in shared its row
17: $\mathbf{s\_data_M} \leftarrow \mathbf{data_M}[\mathbf{indptr}[w\_id] : \mathbf{indptr}[w\_id + 1]]$
18: $\mathbf{s\_indices} \leftarrow \mathbf{indices}[\mathbf{indptr}[w\_id] : \mathbf{indptr}[w\_id + 1]]$
19: if $\mathbf{s\_indices}[\mathbf{s\_indptr}[w\_id] + \_id] < w\_id$ then
20:     $val = \mathbf{s\_data_M}[\mathbf{s\_indptr}[w\_id] + l\_id]$
21:     $col = \mathbf{s\_indices}[\mathbf{s\_indptr}[w\_id] + l\_id]$
22: else if $\mathbf{s\_indices}[\mathbf{s\_indptr}[w\_id] + l\_id] == w\_id$ then
23:     $val = 1$, $col = w\_id$
24: else
25:     $val = 0$, $col = -1$
26: end if
27: for $i = 0, \ldots, nnz - 1$ do   ▷ Each *warp* loops over $\mathcal{I}_{\mathrm{tril}(M)}(w\_id)$
28:     if $l\_id == i$ then
29:         $\mathbf{val_L}[w\_id] \leftarrow v$, $\mathbf{row_U}[w\_id] \leftarrow \mathbf{s\_indices}[\mathbf{s\_indptr}[w\_id] + l\_id]$
30:     end if
31:     $\mathbf{s\_data_M} \leftarrow \mathbf{data_M}[\mathbf{indptr}[\mathbf{row_U}[w\_id]]$   ▷ Each *warp* loads shared index
32:     $\mathbf{s\_indices} \leftarrow \mathbf{indices}[\mathbf{indptr}[\mathbf{row_U}[w\_id]]$
33:     $k = l\_id$
34:     while there are unread values in $row_U$ do
35:         $index = \mathbf{s\_indices}[\mathbf{s\_indptr}[w\_id] + k]$
36:         if $index \geq \mathbf{row_U}[w\_id]$ and $index == col$ then
37:             $\mathbf{s\_data_A}[\mathbf{s\_indptr}[w\_id] + l\_id] - = \mathbf{val_L}[w\_id] * \mathbf{s\_data_M}[\mathbf{s\_indptr}[w\_id] + k]$
38:         end if
39:         $k = k + 1$
40:     end while
41: end for
42: $\mathbf{data_R}[\mathbf{indptr}[w\_id] : \mathbf{indptr}[w\_id + 1]] \leftarrow \mathbf{s\_data_R}$   ▷ Each *warp* copies in global the result
```

Re	Preconditioner	Avg. iter. number	Avg. sol. time (s)
0.01	Diagonal	-	-
	ILU(0)	248.5	5.800
	ITALU(1)	248.5	5.449
	ITALU(1) + Jacobi(3)	273.0	**0.915**
	ITALU(1) + block Jacobi(3)	309.0	1.289
0.025	Diagonal	-	-
	ILU(0)	218.0	4.239
	ITALU(1)	218.0	3.883
	ITALU(1) + Jacobi(3)	250.0	**0.658**
	ITALU(1) + block Jacobi(3)	283.0	1.006
0.05	Diagonal	909.7	1.790
	ILU(0)	199.0	3.508
	ITALU(1)	199.0	3.153
	ITALU(1) + Jacobi(3)	226.0	**0.523**
	ITALU(1) + block Jacobi(3)	248.0	0.788
0.1	Diagonal	791.1	1.233
	ILU(0)	173.0	2.777
	ITALU(1)	173.0	2.424
	ITALU(1) + Jacobi(3)	196.0	**0.401**
	ITALU(1) + block Jacobi(3)	201.0	0.591
1	Diagonal	301.1	0.330
	ILU(0)	72.0	1.249
	ITALU(1)	72.0	0.893
	ITALU(1) + Jacobi(3)	81.0	**0.146**
	ITALU(1) + block Jacobi(3)	84.0	0.218
10	Diagonal	96.4	0.099
	ILU(0)	25.9	0.670
	ITALU(1)	25.9	0.313
	ITALU(1) + Jacobi(3)	29.0	**0.052**
	ITALU(1) + block Jacobi(3)	29.0	0.075
100	Diagonal	38.7	0.040
	ILU(0)	10.8	0.490
	ITALU(1)	10.8	0.134
	ITALU(1) + Jacobi(3)	11.5	**0.020**
	ITALU(1) + block Jacobi(3)	11.0	0.026
1000	Diagonal	44.4	0.047
	ILU(0)	13.8	0.523
	ITALU(1)	13.8	0.166
	ITALU(1) + Jacobi(3)	14.1	**0.023**
	ITALU(1) + block Jacobi(3)	13.8	0.033

Table 6.4: Performance of the preconditioners on the dual-flow at different Reynolds number.

Re	Preconditioner	Avg. iter. number	Avg. sol. time (s)	Error $(\mathbf{u}, p, \rho)$
0.01	ITALU(1)	379.4	4.160	(2.99e-6, 1.76e-3, 8.72e-5)
	ITALU(1) + Jacobi(3)	440.0	1.356	(2.99e-6, 1.76e-3, 8.72e-5)
	ITALU(1) + block Jacobi(3)	389.9	4.967	(2.99e-6, 1.76e-3, 8.72e-5)
0.1	ITALU(1)	156.7	1.727	(6.66e-6, 5.15e-4, 9.10e-5)
	ITALU(1) + Jacobi(3)	168.7	0.541	(6.66e-6, 5.15e-4, 9.10e-5)
	ITALU(1) + block Jacobi(3)	159.4	2.025	(6.66e-6, 5.15e-4, 9.10e-5)
1	ITALU(1)	62.6	0.693	(7.20e-6, 5.26e-4, 8.95e-5)
	ITALU(1) + Jacobi(3)	68.0	0.220	(7.20e-6, 5.26e-4, 8.95e-5)
	ITALU(1) + block Jacobi(3)	64.0	0.820	(7.20e-6, 5.26e-4, 8.95e-5)
10	ITALU(1)	24.0	0.264	(7.16e-6, 6.20e-4, 8.90e-5)
	ITALU(1) + Jacobi(3)	25.3	0.082	(7.16e-6, 6.20e-4, 8.90e-5)
	ITALU(1) + block Jacobi(3)	24.2	0.310	(7.16e-6, 6.20e-4, 8.90e-5)
100	ITALU(1)	10.7	0.121	(1.25e-5, 6.45e-4, 8.82e-5)
	ITALU(1) + Jacobi(3)	10.9	0.035	(1.25e-5, 6.45e-4, 8.82e-5)
	ITALU(1) + block Jacobi(3)	10.7	0.142	(1.25e-5, 6.45e-4, 8.82e-5)
1000	ITALU(1)	13.6	0.151	(4.99e-5, 6.52e-4, 8.81e-5)
	ITALU(1) + Jacobi(3)	14.0	0.044	(4.99e-5, 6.52e-4, 8.81e-5)
	ITALU(1) + block Jacobi(3)	13.6	0.177	(4.99e-5, 6.52e-4, 8.81e-5)

Table 6.5: Performance of the preconditioners on the analytical solution at different Reynolds numbers.

At (*C*)	Preconditioner	Avg. iter. number	Avg. sol. time (s)	Error $(\mathbf{u}, p, \rho)$
0.71 (20)	ITALU(1)	27.9	0.309	(2.32e-5, 9.42e-2, 2.31e-4)
	ITALU(1) + Jacobi(3)	30.0	0.100	(2.32e-5, 9.42e-2, 2.31e-4)
	ITALU(1) + block Jacobi(3)	28.0	0.357	(2.32e-5, 9.42e-2, 2.31e-4)
0.67 (10)	ITALU(1)	35.0	0.235	(1.58e-5, 7.35e-2, 1.99e-4)
	ITALU(1) + Jacobi(3)	37.4	0.121	(1.58e-5, 7.35e-2, 1.99e-4)
	ITALU(1) + block Jacobi(3)	35.5	0.460	(1.58e-5, 7.35e-2, 1.99e-4)
0.61 (5)	ITALU(1)	41.8	0.461	(1.01e-5, 6.25e-2, 1.52e-4)
	ITALU(1) + Jacobi(3)	45.2	0.144	(1.01e-5, 6.25e-2, 1.52e-4)
	ITALU(1) + block Jacobi(3)	42.7	0.55	(1.01e-5, 6.25e-2, 1.52e-4)
0.50 (0)	ITALU(1)	62.6	0.693	(7.20e-6, 5.26e-4, 8.95e-5)
	ITALU(1) + Jacobi(3)	68.0	0.220	(7.20e-6, 5.26e-4, 8.95e-5)
	ITALU(1) + block Jacobi(3)	64.0	0.820	(7.20e-6, 5.26e-4, 8.95e-5)

Table 6.6: Performance of the preconditioners on the analytical solution at different Atwood numbers.

Appendix A

dCATCH numerical software

A.1 Introduction

In this appendix we present the numerical software package *dCATCH* [63] for the computation of a d-variate near G-optimal polynomial regression design of degree m on a finite design space $X \subset \mathbb{R}^d$. In particular, it is the first software package for general-purpose Tchakaloff-like compression of d-variate designs via NonNegative Least Squares (NNLS), freely available on the Internet. The code is an evolution of the codes in [25] (limited to $d = 2, 3$), with a number of features tailored to higher dimension and large-scale computations. The key ingredients are:

- use of d-variate Vandermonde-like matrices at X in a discrete orthogonal polynomial basis (obtained by discrete orthonormalization of the total-degree product Chebyshev basis of the minimal box containing X), with automatic adaptation to the actual dimension of $\mathbb{P}^d_m(X)$;
- few tens of iterations of the basic Titterington multiplicative algorithm until near G-optimality of the design is reached, with a checked G-efficiency of say 95% (but with a design support still far from sparsity);
- Tchakaloff-like compression of the resulting near G-optimal design via NNLS solution of the underdetermined moment system, with concentration of the discrete probability measure by sparse re-weighting to a support strictly contained in X, of cardinality at most $\mathbb{P}^d_{2m}(X)$, keeping the same G-efficiency;
- iterative solution of the large-scale NNLS problem by a new accelerated version of the classical Lawson-Hanson active set algorithm, that we recently introduced in [61] for $d = 2$ and $d = 3$ and here we validate on higher dimensions.

Before giving a more detailed description of the algorithm, it is worth recalling in brief some basic notions of optimal design theory. Such a theory has its roots and main applications within statistics, but also strong connections with approximation theory. In statistics, a design is a probability measure μ supported on a (discrete or continuous) compact set $\Omega \subset \mathbb{R}^d$. The search for designs that optimize some properties of statistical estimators (optimal designs) dates back to at least one century ago, and the relevant literature is so wide and still actively growing and monographs and survey papers are abundant in the literature. For readers interested in the evolution and state of the art of this research field, we may quote, for example, two classical treatises such as in [10, 116], the recent monograph [41] and the algorithmic survey [103], as well as [52, 65, 137] and references therein. On the approximation theory side we may quote, for example, [21, 24].

The present appendix is organized as follows—in Appendix A.2 we briefly recall some basic concepts from the theory of Optimal Designs, for the reader's convenience, with special attention to the deterministic and approximation theoretic aspects. In Appendix A.3 we present in detail our computational approach to near G-optimal d-variate designs via Caratheodory-Tchakaloff compression. All the routines of the *dCATCH* software package here presented, are described. In Appendix A.4 we show several numerical results with dimensions in the range 3–10 and a Conclusions section follows.

For the reader's convenience we also display Table A.1 and Table A.2, describing the acronyms used in this appendix and the content (subroutine names) of the *dCATCH* software package.

Table A.1: List of acronyms.

LS	Least Squares
NNLS	NonNegative Least Squares
LH	Lawson-Hawson algorithm for NNLS
LHI	Lawson-Hawson algorithm with unconstrained LS Initialization
LHDM	Lawson-Hawson algorithm with Deviation Maximization pivoting

Table A.2: *dCATCH* package content.

`dCATCH`	d-variate CAratheodory-TCHakaloff discrete measure compression
`dCHEBVAND`	d-variate Chebyshev-Vandermonde matrix
`dORTHVAND`	d-variate Vandermonde-like matrix in a weighted orthogonal polynomial basis
`dNORD`	d-variate Near G-Optimal Regression Designs
`LHDM`	Lawson-Hawson algorithm with Deviation Maximization pivoting

A.2 G-optimal designs

Let $\mathbb{P}^d_m(\Omega)$ denote the space of d-variate real polynomials of total degree not greater than n, restricted to a (discrete or continuous) compact set $\Omega \subset \mathbb{R}^d$, and let μ be a design, that is, a probability measure, with $supp(\mu) \subseteq \Omega$. In what follows we assume that *supp*(μ) is *determining* for $\mathbb{P}_m^d(\Omega)$ [38], that is, polynomials in $\mathbb{P}_m^d$ vanishing on *supp*(μ) vanish everywhere on Ω.

In the theory of optimal designs, a key role is played by the diagonal of the reproducing kernel for μ in $\mathbb{P}_m^d(\Omega)$ (also called the Christoffel polynomial of degree m for μ)

$$K_m^\mu(\mathbf{x},\mathbf{x}) = \sum_{j=1}^{N_m} p_j^2(\mathbf{x}) \ , \quad N_m = \dim(\mathbb{P}_m^d(\Omega)) \ , \tag{A.1}$$

where $\{p_j\}$ is any μ-orthonormal basis of $\mathbb{P}_m^d(\Omega)$. Recall that $K_m^\mu(x,x)$ can be proved to be independent of the choice of the orthonormal basis. Indeed, a relevant property is the following estimate of the L^∞-norm in terms of the L^2_μ-norm of polynomials

$$\|p\|_{L^\infty(\Omega)} \leq \sqrt{\max_{\mathbf{x}\in\Omega} K_m^\mu(\mathbf{x},\mathbf{x})}\, \|p\|_{L^2_\mu(\Omega)} \ , \quad \forall p \in \mathbb{P}_m^d(\Omega) \ . \tag{A.2}$$

Now, by (A.1) and μ-orthonormality of the basis we get

$$\int_\Omega K_m^\mu(\mathbf{x},\mathbf{x})\, d\mu = \sum_{j=1}^{N_m} \int_\Omega p_j^2(\mathbf{x})\, d\mu = N_m \ , \tag{A.3}$$

which entails that $\max_{x\in\Omega} K_m^\mu(\mathbf{x},\mathbf{x}) \geq N_m$.

Then, a probability measure $\mu_* = \mu_*(\Omega)$ is then called a G-optimal design for polynomial regression of degree m on Ω if

$$\min_\mu \max_{\mathbf{x}\in\Omega} K_m^\mu(\mathbf{x},\mathbf{x}) = \max_{x\in\Omega} K_m^{\mu_*}(\mathbf{x},\mathbf{x}) = N_m \ . \tag{A.4}$$

Observe that, since $\int_\Omega K_m^\mu(\mathbf{x},\mathbf{x})\, d\mu = N_m$ for every μ, an optimal design has also the following property $K_m^{\mu_*}(\mathbf{x},\mathbf{x}) = N_m$, μ_*-a.e. in Ω.

Now, the well-known Kiefer-Wolfowitz General Equivalence Theorem [97] (a cornerstone of optimal design theory), asserts that the difficult min-max problem (A.4) is equivalent to the much simpler maximization problem

$$\max_\mu \det(G_m^\mu) \ , \quad G_m^\mu = \left(\int_\Omega \phi_i(\mathbf{x})\phi_j(\mathbf{x})\, d\mu\right)_{1\leq i,j\leq N_m} \ ,$$

where G_m^μ is the Gram matrix (or information matrix in statistics) of μ in a fixed polynomial basis $\{\phi_i\}$ of $\mathbb{P}_m^d(\Omega)$. Such an optimality is called D-optimality, and ensures that an optimal measure always exists, since the set of Gram matrices of probability measures is compact and convex; see for example, [24, 116] for a general proof of these results, valid for continuous as well as for discrete compact sets.

Notice that an optimal measure is neither unique nor necessarily discrete (unless Ω is discrete itself). Nevertheless, the celebrated Tchakaloff Theorem ensures the existence of a positive quadrature formula for integration in $d\mu_*$ on

Ω, with cardinality not exceeding $N_{2m} = \dim(\mathbb{P}^d_{2m}(\Omega))$ and which is exact for all polynomials in $\mathbb{P}^d_{2m}(\Omega)$. Such a formula is then a design itself, and it generates the same orthogonal polynomials and hence the same Christoffel polynomial of μ_*, preserving G-optimality (see [117] for a proof of Tchakaloff theorem with general measures).

We recall that G-optimality has two important interpretations in terms of statistical and deterministic polynomial regression. From a statistical viewpoint, it is the probability measure on Ω that minimizes the maximum prediction variance by polynomial regression of degree m, cf. for example, [116]. On the other hand, from an approximation theory viewpoint, if we call $\mathcal{L}^{\mu_*}_m$ the corresponding weighted least squares projection operator $L^\infty(\Omega) \to \mathbb{P}^d_m(\Omega)$, namely

$$\|f - \mathcal{L}^{\mu_*}_m f\|_{L^2_{\mu_*}(\Omega)} = \min_{p \in \mathbb{P}^d_m(\Omega)} \|f - p\|_{L^2_{\mu_*}(\Omega)} \ , \tag{A.5}$$

by (A.2) we can write for every $f \in L^\infty(\Omega)$

$$\begin{aligned}\|\mathcal{L}^{\mu_*}_m f\|_{L^\infty(\Omega)} &\leq \sqrt{\max_{\mathbf{x}\in\Omega} K^{\mu_*}_m(\mathbf{x},\mathbf{x})}\, \|\mathcal{L}^{\mu_*}_m f\|_{L^2_{\mu_*}(\Omega)} = \sqrt{N_m}\, \|\mathcal{L}^{\mu_*}_m f\|_{L^2_{\mu_*}(\Omega)} \\ &\leq \sqrt{N_m}\, \|f\|_{L^2_{\mu_*}(\Omega)} \leq \sqrt{N_m}\, \|f\|_{L^\infty(\Omega)} \ ,\end{aligned}$$

(where the second inequality comes from μ_*-orthogonality of the projection), which gives

$$\|\mathcal{L}^{\mu_*}_m\| = \sup_{f\neq 0} \frac{\|\mathcal{L}^{\mu_*}_m f\|_{L^\infty(\Omega)}}{\|f\|_{L^\infty(\Omega)}} \leq \sqrt{N_m} \ , \tag{A.6}$$

that is a G-optimal measure minimizes (the estimate of) the weighted least squares uniform operator norm.

We stress that in this context we are interested in the fully discrete case of a finite design space $\Omega = X$, so that any design μ is identified by a set of positive weights (masses) summing up to 1 and integrals are weighted sums.

A.3 Computing near G-optimal compressed designs

Since in the present context we have a finite design space $\Omega = X = \{\mathbf{x}_1, \dots, \mathbf{x}_M\} \subset \mathbb{R}^d$, we may think a design μ as a vector of nonnegative weights $\mathbf{u} = (u_1, \cdots, u_M)$ attached to the points, such that $\|\mathbf{u}\|_1 = 1$ (the support of μ being identified by the positive weights). Then, a G-optimal (or D-optimal) design μ_* is represented by the corresponding nonnegative vector $\mathbf{u}_*$. We write $K^u_m(\mathbf{x},\mathbf{x}) = K^\mu_m(\mathbf{x},\mathbf{x})$ for the Christoffel polynomial and similarly for other objects (spaces, operators, matrices) corresponding to a discrete design. At the same time, $L^\infty(\Omega) = \ell^\infty(X)$, and $L^2_\mu(\Omega) = \ell^2_u(X)$ (a weighted ℓ^2 functional space on X) with $\|f\|_{\ell^2_u(X)} = \left(\sum_{i=1}^M u_i\, f^2(x_i)\right)^{1/2}$.

In order to compute an approximation of the desired $\mathbf{u}_*$, we resort to the basic multiplicative algorithm proposed by Titterington in the '70s (cf. [136]), namely

$$u_i^{(k+1)} = u_i^{(k)} \, \frac{K_m^{\mathbf{u}^{(k)}}(\mathbf{x}_i, \mathbf{x}_i)}{N_m} \ , \quad 1 \le i \le M \ , \quad k = 0, 1, 2, \dots \ , \tag{A.7}$$

with initialization $\mathbf{u}^{(0)} = (1/M, \dots, 1/M)^T$. Such an algorithm is known to be convergent sublinearly to a D-optimal (or G-optimal by the Kiefer-Wolfowitz Equivalence Theorem) design, with an increasing sequence of Gram determinants

$$\det(G_m^{\mathbf{u}^{(k)}}) = \det(V^T \operatorname{diag}(\mathbf{u}^{(k)})\, V),$$

where V is a Vandermonde-like matrix in any fixed polynomial basis of $\mathbb{P}_m^d(X)$; cf., for example, [103, 137]. Observe that $\mathbf{u}^{(k+1)}$ is indeed a vector of positive probability weights if such is $\mathbf{u}^{(k)}$. In fact, the Christoffel polynomial $K_m^{\mathbf{u}^{(k)}}$ is positive on X, and calling μ_k the probability measure on X associated with the weights $\mathbf{u}^{(k)}$ we get immediately $\sum_i u_i^{(k+1)} = \frac{1}{N_m} \sum_i u_i^{(k)} \, K_m^{\mathbf{u}^{(k)}}(\mathbf{x}_i, \mathbf{x}_i) = \frac{1}{N_m} \int_X K_m^{\mathbf{u}^{(k)}}(\mathbf{x}, \mathbf{x})\, d\mu_k = 1$ by (A.3) in the discrete case $\Omega = X$.

Our implementation of (A.7) is based on the functions

- `C=dCHEBVAND(n,X)`,
- `[U,jvec]=dORTHVAND(n,X,u,jvec)`,
- `[pts,w]=dNORD(m,X,gtol)`.

Given the degree `n` and the initial point cloud represented by the two dimensional array `X`, the function `dCHEBVAND` computes the two dimensional array `C` containing the d-variate Chebyshev-Vandermonde matrix $C = (\phi_j(x_i)) \in \mathbb{R}^{M \times N_n}$, where $\{\phi_j(\mathbf{x})\} = \{T_{\nu_1}(\alpha_1 x_1 + \beta_1) \dots T_{\nu_d}(\alpha_d x_d + \beta_d)\}$, $0 \le \nu_i \le n$, $\nu_1 + \dots + \nu_d \le n$, is a suitably ordered total-degree product Chebyshev basis of the minimal box $[a_1, b_1] \times \dots \times [a_d, b_d]$ containing X, with $\alpha_i = 2/(b_i - a_i)$, $\beta_i = -(b_i + a_i)/(b_i - a_i)$. Here we have resorted to the codes in [32] for the construction and enumeration of the required "monomial" degrees. Though the initial basis is then orthogonalized, the choice of the Chebyshev basis is dictated by the necessity of controlling the conditioning of the matrix. This would be on the contrary extremely large with the standard monomial basis, already at moderate regression degrees, preventing a successful orthogonalization.

Said `u` the array representing the probability weight vector $\mathbf{u}$, the second function `dORTHVAND` computes the two dimensional array `U` containing a Vandermonde-like matrix in a $\mathbf{u}$-orthogonal polynomial basis on X. This is accomplished essentially by numerical rank evaluation for `C=dCHEBVAND(n,X)` and consequent thin QR factorization

$$\operatorname{diag}(\sqrt{\mathbf{u}})\, C_0 = QR \ , \quad U = C_0 \, R^{-1} \ , \tag{A.8}$$

where Q rectangular with orthogonal columns, R is an upper triangular nonsingular matrix and $\sqrt{\mathbf{u}} = (\sqrt{u_1}, \dots, \sqrt{u_M})^T$. The matrix C_0 has full rank and corresponds to a selection of the columns of C (i.e., of the original basis polynomials) via QR with column pivoting, in such a way that these form a basis of $\mathbb{P}_n^d(X)$, since $\mathrm{rank}(C) = \dim(\mathbb{P}_n^d(X))$. A possible alternative, not yet implemented, is the use of a rank-revealing QR factorization. The in-out parameter `jvec` allows to pass directly the column index vector corresponding to a polynomial basis after a previous call to `dORTHVAND` with the same degree `n`, avoiding numerical rank computation and allowing a simple "economy size" QR factorization of the matrix `diag(sqrt(u))C(:,jvec)`.

Summarizing, U is a Vandermonde-like matrix for degree n on X in the required $\mathbf{u}$-orthogonal basis of $\mathbb{P}_n^d(X)$, that is

$$[p_1(\mathbf{x}), \dots, p_{N_n}(\mathbf{x})] = [\phi_{j_1}(\mathbf{x}), \dots, \phi_{j_{N_n}}(\mathbf{x})]\, R^{-1} \ , \tag{A.9}$$

where $\mathbf{j}_{vec} = (j_1, \dots, j_{N_n})^T$ is the multi-index `jvec` resulting from pivoting. Indeed by (A.8) we can write the scalar product $(p_h, p_k)_{\ell^2_u(X)}$ as

$$(p_h, p_k)_{\ell^2_u(X)} = \sum_{i=1}^{M} u_i\, p_h(x_i)\, p_k(x_i) = (U^T \,\mathrm{diag}(\mathbf{u})\, U)_{hk} = (Q^T Q)_{hk} = \delta_{hk} \ ,$$

for $1 \le h, k \le N_n$, which shows orthonormality of the polynomial basis in (A.9).

We stress that $\mathrm{rank}(C) = \dim(\mathbb{P}_n^d(X))$ could be strictly smaller than $\dim(\mathbb{P}_n^d) = \binom{n+d}{d}$, when there are polynomials in $\mathbb{P}_n^d$ vanishing on X that do not vanish everywhere. In other words, X lies on a lower-dimensional algebraic variety (technically one says that X is not $\mathbb{P}_n^d$-determining [38]). This certainly happens when $|X|$ is too small, namely $|X| < \dim(\mathbb{P}_n^d)$, but think for example also to the case when $d = 3$ and X lies on the 2-sphere S^2 (independently of its cardinality), then we have $\dim(\mathbb{P}_n^d(X)) \le \dim(\mathbb{P}_n^d(S^2)) = (n+1)^2 < \dim(\mathbb{P}_n^3) = (n+1)(n+2)(n+3)/6$.

Iteration (A.7) is implemented within the third function `dNORD` whose name stands for d-dimensional Near G-Optimal Regression Designs, which calls `dORTHVAND` with $n = m$. Near optimality is here twofold, namely it concerns both the concept of G-efficiency of the design and the sparsity of the design support.

We recall that G-efficiency is the percentage of G-optimality reached by a (discrete) design, measured by the ratio

$$G_m(u) = \frac{N_m}{\max_{\mathbf{x} \in X} K_m^{\mathbf{u}}(\mathbf{x}, \mathbf{x})} \ ,$$

knowing that $G_m(\mathbf{u}) \le 1$ by (A.3) in the discrete case $\Omega = X$. Notice that $G_m(\mathbf{u})$ can be easily computed after the construction of the $\mathbf{u}$-orthogonal Vandermonde-like matrix U by `dORTHVAND`, as $G_m(\mathbf{u}) = N_m / (\max_i \|row_i(U)\|_2^2)$.

In the multiplicative algorithm (A.7), we then stop iterating when a given threshold g_{tol} of G-efficiency (the input parameter `gtol` in the call to `dNORD`) is reached by $\mathbf{u}^{(k)}$, since $G_m(\mathbf{u}^{(k)}) \to 1$ as $k \to \infty$, say for example $G_m(\mathbf{u}^{(k)}) \ge$

95% or $G_m(\mathbf{u}^{(k)}) \geq 99\%$. Since convergence is sublinear and in practice we see that $1 - G_m(\mathbf{u}^{(k)}) = \mathcal{O}(1/k)$, for a 90% G-efficiency the number of iterations is typically in the tens, whereas it is in the hundreds for 99% one and in the thousands for $99,9\%$. When a G-efficiency very close to 1 is needed, one could resort to more sophisticated multiplicative algorithms, see e.g. [65, 137].

In many applications however a G-efficiency of 90%–95% could be sufficient (then we may speak of near G-optimality of the design), but though in principle the multiplicative algorithm converges to an optimal design μ_* on X with weights $\mathbf{u}_*$ and cardinality $N_m \leq |\operatorname{supp}(\mu_*)| \leq N_{2m}$, such a sparsity is far from being reached after the iterations that guarantee near G-optimality, in the sense that there is a still large percentage of non-negligible weights in the near optimal design weight vector, say

$$\mathbf{u}^{(\overline{k})} \text{ such that } G_m(\mathbf{u}^{(\overline{k})}) \geq g_{tol} \; . \tag{A.10}$$

Following [26, 27], we can however effectively compute a design which has the same G-efficiency of $\mathbf{u}^{(\overline{k})}$ but a support with a cardinality not exceeding $N_{2m} = \dim(\mathbb{P}^d_{2m}(X))$, where in many applications $N_{2m} \ll |X|$, obtaining a remarkable compression of the near optimal design.

The theoretical foundation is a generalized version [117] of Tchakaloff Theorem [131] on positive quadratures, which asserts that for every measure on a compact set $\Omega \subset \mathbb{R}^d$ there exists an algebraic quadrature formula exact on $\mathbb{P}^d_n(\Omega))$, with positive weights, nodes in Ω and cardinality not exceeding $N_n = \dim(\mathbb{P}^d_n(\Omega)$.

In the present discrete case, that is, where the designs are defined on $\Omega = X$, this theorem implies that for every design μ on X there exists a design ν, whose support is a subset of X, which is exact for integration in $d\mu$ on $\mathbb{P}^d_n(X)$. In other words, the design ν has the same basis moments (indeed, for any basis of $\mathbb{P}^d_n(\Omega)$)

$$\int_X p_j(\mathbf{x})\, d\mu = \sum_{i=1}^{M} u_i\, p_j(x_i) = \int_X p_j(\mathbf{x})\, d\nu = \sum_{\ell=1}^{L} w_\ell\, p_j(\xi_\ell) \; , \;\; 1 \leq j \leq N_n \; ,$$

where $L \leq N_n \leq M$, $\{u_i\}$ are the weights of μ, $\operatorname{supp}(\nu) = \{\xi_\ell\} \subseteq X$ and $\{w_\ell\}$ are the positive weights of ν. For $L < M$, which certainly holds if $N_n < M$, this represents a compression of the design μ into the design ν, which is particularly useful when $N_n \ll M$.

In matrix terms this can be seen as the fact that the underdetermined $\{p_j\}$-moment system

$$U_n^T \mathbf{v} = U_n^T \mathbf{v} \tag{A.11}$$

has a nonnegative solution $\mathbf{v} = (v_1, \ldots, v_M)^T$ whose positive components, say $w_\ell = v_{i_\ell}$, $1 \leq \ell \leq L \leq N_n$, determine the support points $\{\xi_\ell\} \subseteq X$ (for clarity we indicate here by U_n the matrix `U` computed by `dORTHVAND` at degree n). This fact is indeed a consequence of the celebrated Caratheodory theorem on conic combinations [40], asserting that a linear combination with nonnegative coefficients of M vectors in $\mathbb{R}^N$ with $M > N$ can be re-written as linear positive combination of at most N of them. So, we get the discrete version of Tchakaloff theorem by applying Caratheodory theorem to the columns of U_n^T in the system (A.11), ensuring then existence of a nonnegative solution $\mathbf{v}$ with at most N_n nonzero components.

In order to compute such a solution to (A.11) we choose the strategy based on Quadratic Programming introduced in [128], namely on sparse solution of the NonNegative Least Squares (NNLS) problem

$$\mathbf{v} = \underset{\mathbf{z}\in\mathbb{R}^M,\, \mathbf{z}\geq 0}{\operatorname{argmin}} \|U_n^T \mathbf{z} - U_n^T \mathbf{u}\|_2^2$$

by a new accelerated version of the classical Lawson-Hanson active-set method, namely the Lawson-Hanson algorithm with deviation maximization pivoting discussed in Chapter 4 and implemented by the function

- `[x,resnorm,exitflag]=LHDM (A,b,options)`,

which solves the generic NNLS problem

$$\min_{\mathbf{x}\in\mathbb{R}^M,\, \mathbf{x}\geq 0} \|A\mathbf{x} - \mathbf{b}\|_2^2$$

where $A \in \mathbb{R}^{N\times M}$ and $\mathbf{b} \in \mathbb{R}^N$, with a Matlab implementation of the LHDM Algorithm 6. The input variable `options` is a structure type object containing the user parameters, for example, the aforementioned k_{max} and $\tau_1, \tau_2, \tau_\theta$. The output parameter `x` is the least squares solution, `resnorm` is the squared 2-norm of the residual and `exitflag` is set to 0 if the `LHDM` algorithm has reached the maximum number of iterations without converging and 1 otherwise. In our implementation, we call `LHDM` with `A = U'` and `b = U'*u`. In the literature, an accelerating technique was introduced by Van Benthem and Keenan [140], who introduced a nontrivial initialization of the algorithm by means of unconstrained least squares solution. In the following section we are going to compare such an approach, whose Matlab implementation is briefly named `LHI`, and the Matlab implementation `lsqnonneg` of the Lawson-Hanson algorithm, briesfly named `LH` hereafter, with `LHDM`.

We observe that working with an orthogonal polynomial basis of $\mathbb{P}_n^d(X)$ allows to deal with the well-conditioned matrix U_n in the Lawson-Hanson algorithm. The overall computational procedure is implemented by the function

- `[pts,w,momerr]=dCATCH(n,X,u)`,

where `dCATCH` stands for d-variate CAratheodory-TCHakaloff discrete measure compression. It works for any discrete measure on a discrete set X. Indeed, it could be used, other than for design compression, also in the compression of d-variate quadrature formulas, to give an example. The output array `pts` contains a subset of points $\{\xi_\ell\} \subset X$ which form the support of the compressed measure, the array `w` is the vector $\mathbf{w} = \{w_\ell\} = \{v_{i_\ell} > 0\}$ of positive weights (that we may call a d-variate near G-optimal Tchakaloff design) and last `momerr` is the moment residual $\|U_n^T v - U_n^T u\|_2$.

In the present framework we call `dCATCH` with $n = 2m$ and $\mathbf{u} = \mathbf{u}^{(\overline{k})}$, cf. (A.10), that is, we solve

$$\mathbf{v} = \underset{\mathbf{z}\in\mathbb{R}^M,\, \mathbf{z}\geq 0}{\operatorname{argmin}} \|U_{2m}^T \mathbf{z} - U_{2m}^T \mathbf{u}^{(\overline{k})}\|_2^2 \ . \tag{A.12}$$

In such a way the compressed design generates the same scalar product of $\mathbf{u}^{(\overline{k})}$ in $\mathbb{P}_m^d(X)$, and hence the same orthogonal polynomials and the same Christoffel function on X keeping thus invariant the G-efficiency

$$\mathbb{P}_{2m}^d(X) \ni K_m^v(\mathbf{x},\mathbf{x}) = K_m^{\mathbf{u}^{(\overline{k})}}(\mathbf{x},\mathbf{x}) \ \forall \mathbf{x} \in X \implies G_m(\mathbf{v}) = G_m(\mathbf{u}^{(\overline{k})}) \geq g_{tol} \tag{A.13}$$

with a (much) smaller support. From a deterministic regression viewpoint (approximation theory), let us denote by p_m^{opt} the polynomial in $\mathbb{P}_m^d(X)$ of best uniform approximation for f on X, where we assume $f \in C(D)$ with $X \subset D \subset R^d$, D being a compact domain (or even lower-dimensional manifold), and by $E_m(f;X) = \inf_{p\in\mathbb{P}_m^d(X)} \|f-p\|_{\ell^\infty(X)} = \|f - p_m^{opt}|_{\ell^\infty(X)}$ and $E_m(f;D) = \inf_{p\in\mathbb{P}_m^d(D)} \|f-p\|_{L^\infty(D)}$ the best uniform polynomial approximation errors on X and D. Then, denoting by $\mathcal{L}_m^{\mathbf{u}^{(\overline{k})}}$ and $\mathcal{L}_m^w f = \mathcal{L}_m^v f$ the weighted least squares polynomial approximation of f (cf. (A.5)) by the near G-optimal weights $\mathbf{u}^{(\overline{k})}$ and w, respectively, with the same reasoning used to obtain (A.6) and by (A.13) we can write the operator norm estimates

$$\left\|\mathcal{L}_m^{\mathbf{u}^{(\overline{k})}}\right\|, \|\mathcal{L}_m^w\| \leq \sqrt{\widetilde{N}_m} \leq \sqrt{\frac{N_m}{g_{tol}}}\ , \quad \widetilde{N}_m = \frac{N_m}{G_m(\mathbf{u}^{(\overline{k})})} = \frac{N_m}{G_m(\mathbf{v})}\ .$$

Moreover, since $\mathcal{L}_m^w p = p$ for any $p \in \mathbb{P}_m^d(X)$, we can write the near optimal estimate

$$\begin{aligned}
&\|f - \mathcal{L}_m^w f\|_{\ell^\infty(X)} \\
&\leq \|f - p_m^{opt}\|_{\ell^\infty(X)} + \|p_m^{opt} - \mathcal{L}_m^w p_m^{opt}\|_{\ell^\infty(X)} + \|\mathcal{L}_m^w p_m^{opt} - \mathcal{L}_m^w f\|_{\ell^\infty(X)} \\
&= \|f - p_m^{opt}\|_{\ell^\infty(X)} + \|\mathcal{L}_m^w p_m^{opt} - \mathcal{L}_m^w f\|_{\ell^\infty(X)} \leq (1 + \|\mathcal{L}_m^w\|)\, E_m(f;X) \\
&\leq \left(1 + \sqrt{\frac{N_m}{g_{tol}}}\right) E_m(f;X) \\
&\leq \left(1 + \sqrt{\frac{N_m}{g_{tol}}}\right) E_m(f;D) \approx \left(1 + \sqrt{N_m}\right) E_m(f;D)\ .
\end{aligned}$$

Notice that $\mathcal{L}_m^w f$ is constructed by sampling f only at the compressed support $\{\xi_\ell\} \subset X$. The error depends on the regularity of f on $D \supset X$, with a rate that can be estimated whenever D admits a multivariate Jackson-like inequality, cf. [113].

A.4 Numerical examples

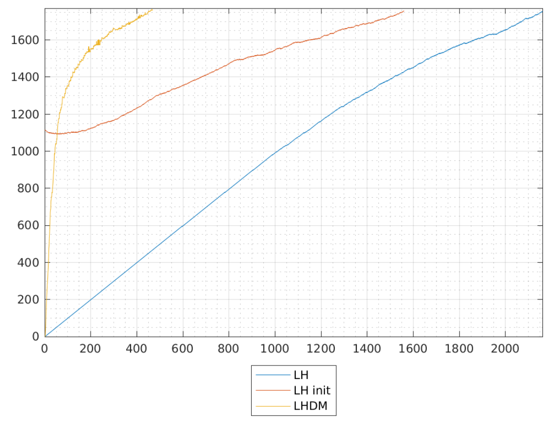
In this section, we perform several tests on the computation of d-variate near G-optimal Tchakaloff designs, from low to moderate dimension d. In practice, we are able to treat, on a personal computer, large-scale problems where $|X| \times \dim(P_{2m}^d)$ is up to 10^8–10^9, with $\dim(P_{2m}^d) = \binom{2m+d}{d} = \binom{2m+d}{2m}$. Recall that the main memory requirement is given by the $N_{2m} \times M$ matrix U^T in the compression process solved by the LHDM algorithm, where $M = |X|$ and $N_{2m} = \dim(P_{2m}^d(X)) \leq \dim(P_{2m}^d)$.

Given the dimension $d > 1$ and the polynomial degree m, the routine `LHDM` empirically sets the parameter $k_{\max}$ as follows $k_{\max} = \lceil \binom{2m+d}{d}/(m(d-1)) \rceil$, while the threshold is $\tau_\theta = \cos(\frac{\pi}{2} - \theta), \theta \approx 0.22$. Here, we set $\tau_u = \tau_w = 0$. All the tests are performed on a workstation with a 32 GB RAM and an Intel Core i7-8700 CPU @ 3.20 GHz.

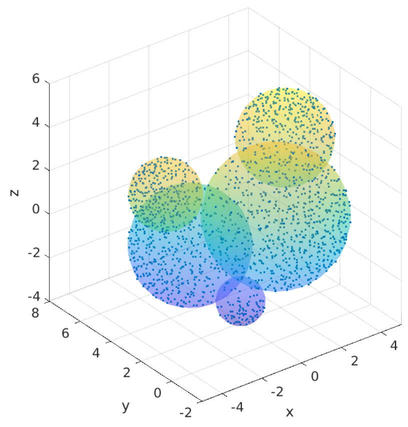
A.4.1 Complex shapes $d = 3$

To show the flexibility of the package *dCATCH*, we compute near G-optimal designs on a "multibubble" $D \subset \mathbb{R}^3$ (i.e., the union of a finite number of non-disjoint balls), which can have a very complex shape with a boundary surface very difficult to describe analytically. Indeed, we are able to implement near optimal regression on quite complex solids, arising from finite union, intersection and set difference of simpler pieces, possibly multiply-connected, where for each piece we have available the indicator function via inequalities. Grid-points or low-discrepancy points, for example, Halton points, of a surrounding box, could be conveniently used to discretize the solid. Similarly, thanks to the adaptation of the method to the actual dimension of the polynomial spaces, we can treat near optimal regression on the surfaces of such complex solids, as soon as we are able to discretize the surface of each piece by point sets with good covering properties (for example, we could work on the surface of a multibubble by discretizing each sphere via one of the popular spherical point configurations, cf. [91]).

We perform a test at regression degree $m = 10$ on the 5-bubble shown in Figure A.1b. The initial support X consists in the $M = 18\,915$ points within 64,000 low discrepancy Halton points, falling in the closure of the multibubble shown in Figure A.1b together with Multibubble with the 1763 compressed Tchakaloff points. Results are summarized in Figure A.1a, showing the evolution of the cardinality of the passive set P along the iterations of the three LH algorithms, and Table A.3: here, `compr = M / mean(cpts)` is the mean compression ratio obtained by the three methods listed; t_{LH}/t_{Titt} is the ratio between the execution time of LH and that of the Titterington algorithm; t_{LH}/t_{LHDM} (t_{LHI}/t_{LHDM}) is the ratio between the execution time of LH (LHI) and that of LHDM ; `cpts` is the number of compressed Tchakaloff points and `momerr` is the final moment residual.



(a) Cardinality of the passive set per iteration.



(b) Compressed support.

Figure A.1: Multibubble test case, regression degree $m = 10$.

Table A.3: Results for the multibubble numerical test.

Test			LH				LHI			LHDM	
m	M	compr	t_{LH}/t_{Titt}	t_{LH}/t_{LHDM}	cpts	momerr	t_{LHI}/t_{LHDM}	cpts	momerr	cpts	momerr
10	18 915	11	40.0	2.7	1755	3.4×10^{-8}	3.2	1758	3.2×10^{-8}	1755	1.5×10^{-8}

A.4.2 Hypercubes: Chebyshev grids

In a recent paper [27], a connection has been studied between the statistical notion of G-optimal design and the approximation theoretic notion of admissible mesh for multivariate polynomial approximation, deeply studied in the last decade after [38] (see, e.g., [22, 54] with the references therein). In particular, it has been shown that near G-optimal designs on admissible meshes of suitable cardinality have a G-efficiency on the whole d-cube that can be made convergent to 1. For example, it has been proved by the notion of Dubiner distance and suitable multivariate polynomial inequalities, that a design with G-efficiency γ on a grid X of $(2km)^d$ Chebyshev points (the zeros of $T_{2km}(t) = cos(2km \arccos(t))$, $t \in [-1, 1]$), is a design for $[-1, 1]^d$ with G-efficiency $\gamma(1 - \pi^2/(8k^2))$. For example, taking $k = 3$ a near G-optimal Tchakaloff design with $\gamma = 0.99$ on a Chebyshev grid of $(6m)^d$ points is near G-optimal on $[-1, 1]^d$ with G-efficiency approximately $0.99 \cdot 0.86 \approx 0.85$, and taking $k = 4$ (i.e., a Chebyshev grid of $(8m)^d$ points) the corresponding G-optimal Tchakaloff design has G-efficiency approximately $0.99 \cdot 0.92 \approx 0.91$ on $[-1, 1]^d$ (in any dimension d).

We perform three tests in different dimension spaces and at different regression degrees. Results are shown in Figure A.2 and Table A.4. Here, `compr = M / mean(cpts)` is the mean compression ratio obtained by the three methods listed; t_{LH}/t_{Titt} is the ratio between the execution time of LH and that of Titterington algorithm; t_{LH}/t_{LHDM} (t_{LHI}/t_{LHDM}) is the ratio between the execution time of LH (LHI) and that of LHDM; `cpts` is the number of compressed Tchakaloff points and `momerr` is the final moment residual.

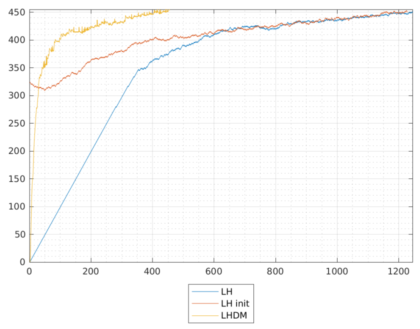
Table A.4: Results of numerical tests on $M = (2km)^d$ Chebyshev's nodes, with $k = 4$, with different dimensions and degrees.

Test				LH				LHI			LHDM	
d	m	M	`compr`	t_{LH}/t_{Titt}	t_{LH}/t_{LHDM}	`cpts`	`momerr`	t_{LHI}/t_{LHDM}	`cpts`	`momerr`	`cpts`	`momerr`
3	6	110 592	250	0.4	3.1	450	5.0×10^{-7}	3.5	450	3.4×10^{-7}	450	1.4×10^{-7}
4	3	331 776	1607	0.2	2.0	207	8.9×10^{-7}	3.4	205	9.8×10^{-7}	207	7.9×10^{-7}
5	2	1 048 576	8571	0.1	1.4	122	6.3×10^{-7}	1.5	123	3.6×10^{-7}	122	3.3×10^{-7}

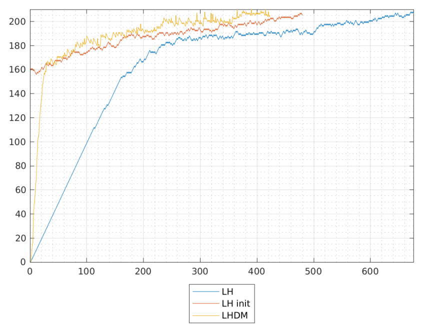
A.4.3 Hypercubes: low-discrepancy points

The direct connection of Chebyshev grids with near G-optimal designs discussed in the previous subsection suffers rapidly of the curse of dimensionality, so only regression at low degree in relatively low dimension can be treated. On the other hand, in sampling theory a number of discretization nets with good space-filling properties on hypercubes has been proposed and they allow to increase the dimension d. We refer in particular to Latin hypercube sampling or low-discrepancy points (Sobol, Halton and other popular sequences); see e.g. [66]. These families of points give a discrete model of hypercubes that can be used in many different deterministic and statistical applications.

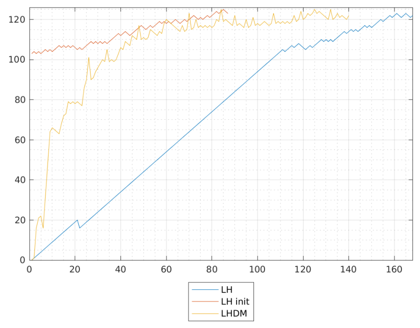
Here we consider a discretization made via Halton points. We present in particular two examples, where we take as finite design space X a set of $M = 10^5$ Halton points, in $d = 4$ with regression degree $m = 5$, and in $d = 10$ with $m = 2$. In both examples, $\dim(P^d_{2m}) = \binom{2m+d}{d} = \binom{2m+d}{2m} = \binom{14}{4} = 1001$, so that the largest matrix involved in the construction is the 1001 × 100 000 Chebyshev-Vandermonde matrix C for degree $2m$ on X constructed at the beginning of the compression process (by `dORTHVAND` within `dCATCH` to compute U_{2m} in (A.12)).



(a) $d = 3$, $n = 6$, $M = 110\ 592$.



(b) $d = 4$, $n = 3$, $M = 331\ 776$.



(c) $d = 5$, $n = 2$, $M = 1\ 048\ 576$.

Figure A.2: Cardinality of the passive set per iteration of the three LH algorithms for Chebyshev nodes' tests.

Results are shown in Figure A.3 and Table A.5. Here, `compr = M / mean(cpts)` is the mean compression ratio obtained by the three methods listed; t_{LH}/t_{Titt} is the ratio between the execution time of LH and that of Titterington algorithm; t_{LH}/t_{LHDM} (t_{LHI}/t_{LHDM}) is the ratio between the execution time of LH (LHI) and that of LHDM; `cpts` is the number of compressed Tchakaloff points and `momerr` is the final moment residual.

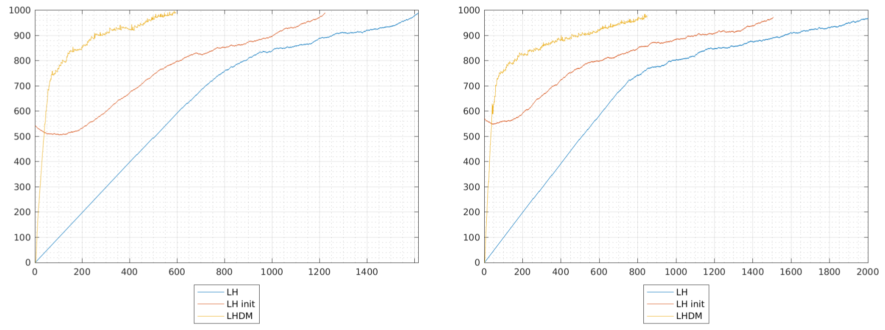
Remark 2. *The computational complexity of `dCATCH` mainly depends on the QR decompositions, which clearly limit the maximum size of the problem and mainly determine the execution time. Indeed, the computational complexity of a QR factorization of a matrix of size $n_r \times n_c$, with $n_c \leq n_r$, is high, namely $2(n_c^2\ n_r - n_c^3/3) \approx 2n_c^2\ n_r$ (see e.g. [81]).*

Titterington algorithm performs a QR factorization of a $M \times N_m$ matrix at each iteration, with the following overall computational complexity

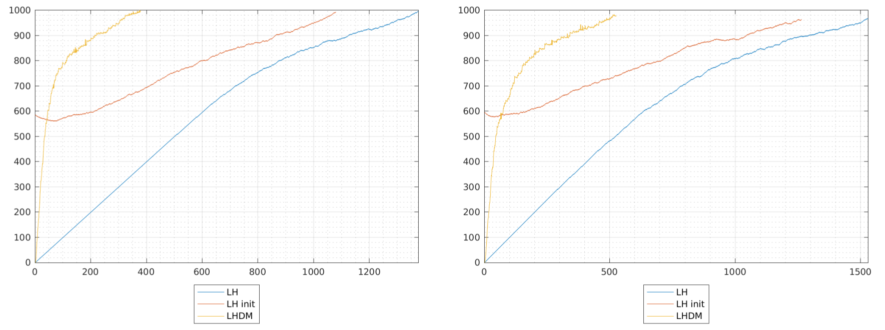
$$C_{Titt} \approx 2\bar{k}\ M\ N_m^2\ ,$$

where $\bar{k}$ is the number of iterations necessary for convergence, that depends on the desired G-efficiency.

On the other hand, the computational cost of one iteration of the Lawson-Hanson algorithm, fixed the passive set P, is given by the solution of an unconstrained least squares problem problem with a $N_{2m} \times |P|$ matrix, which approximately takes $2N_{2m}|P|^2$ flops, that is the cost of the corresponding QR decomposition.However, as experimental results confirm, the evolution of the set P along the execution of the algorithm may vary significantly depending on the experiment settings, so that the exact overall complexity is hard to estimate. Lower and upper bounds are available, but may lead to heavy under- and over-estimations, respectively; cf. [126] for a discussion on complexity issues.



(a) $d = 10, m = 2, M = 10\ 000$. (b) $d = 10, m = 2, M = 100\ 000$.



(c) $d = 4, m = 5, M = 10\ 000$. (d) $d = 4, m = 5, M = 100\ 000$.

Figure A.3: Cardinality of the passive set per iteration of the three LH algorithms for Halton points' tests.

Table A.5: Results of numerical tests on Halton points.

Test				LH				LHI			LHDM	
d	m	M	compr	t_{LH}/t_{Titt}	t_{LH}/t_{LHDM}	cpts	momerr	t_{LHI}/t_{LHDM}	cpts	momerr	cpts	momerr
10	2	10 000	10	41.0	1.9	990	1.1×10^{-8}	1.9	988	9.8×10^{-9}	990	9.4×10^{-9}
10	2	100 000	103	6.0	3.1	968	3.6×10^{-7}	2.8	973	2.7×10^{-7}	968	4.2×10^{-7}
4	5	10 000	10	20.2	2.3	997	9.7×10^{-9}	2.4	993	1.3×10^{-8}	997	2.1×10^{-9}
4	5	100 000	103	2.0	3.8	969	6.6×10^{-7}	3.8	964	6.3×10^{-7}	969	5.3×10^{-7}

A.5 Conclusions and outlook

In this appendix, we have presented *dCATCH* [63], a numerical software package for the computation of a d-variate near G-optimal polynomial regression design of degree m on a finite design space $X \subset \mathbb{R}^d$. The mathematical foundation is discussed connecting statistical design theoretic and approximation theoretic aspects, with a special emphasis on deterministic regression (Weighted Least Squares). The package takes advantage of an accelerated version of the classical NNLS Lawson-Hanson solver discussed in Chapter 4 and applied to design compression. As a few examples of use cases of this package we have shown the results on a complex shape (multibubble) in three dimensions, and on hypercubes discretized with Chebyshev grids and with Halton points, testing different combinations of dimensions and degrees which generate large-scale problems for a personal computer.

The present package, *dCATCH* works for any discrete measure on a discrete set X. Indeed, it could be used, other than for design compression, also in the compression of d-variate quadrature formulas, even on lower-dimensional manifolds, to give an example. We may observe that with this approach we can compute a d-variate compressed design starting from a high-cardinality sampling set X, that discretizes a continuous compact set (see Appendix A.4.2 and Appendix A.4.3). This design allows an m-th degree near optimal polynomial regression of a function on the whole X, by sampling on a small design support. We stress that the compressed design is function-independent and thus can be constructed "once and for all" in a pre-processing stage. This approach is potentially useful, for example, for the solution of d-variate parameter estimation problems, where we may think to model a nonlinear cost function by near optimal polynomial regression on a discrete d-variate parameter space X; cf., for example, [12, 11] for instances of parameter estimation problems from mechatronics applications (*Digital Twins* of controlled systems) and references on the subject. Minimization of the polynomial model could then be accomplished by popular methods developed in the growing research field of Polynomial Optimization, such as Lasserre's SOS (Sum of Squares) and measure-based hierarchies, and other recent methods; cf., for example, [99, 53, 106] with the references therein.

From a computational viewpoint, the results presented show relevant speed-ups in the compression stage, with respect to the standard Lawson-Hanson algorithm, in terms of the number of iterations required and of computing time within the Matlab scripting language. In order to further decrease the execution times and to allow us to tackle larger design problems, a proper strategy should be devised in order to break the curse of dimensionality which affects the size of the Vandermonde-like matrix on the regression degree and/or the number of dimensions increase.

Bibliography

[1] A. Abdelfattah, A. Haidar, S. Tomov, and J. Dongarra. Performance, Design, and Autotuning of Batched GEMM for GPUs. In J. M. Kunkel, P. Balaji, and J. Dongarra, editors, *High Performance Computing*, pages 21–38, Cham, 2016. Springer International Publishing. ISBN 978-3-319-41321-1.

[2] P. Amodio and G. Romanazzi. Algorithm 859: BABDCR - a Fortran 90 package for the solution of bordered ABD linear systems. *ACM Trans. Math. Softw.*, 32:597–608, 01 2006.

[3] P. Amodio and G. Romanazzi. Parallel Numerical Solution of ABD and BABD linear systems arising from BVPs. *Scalable Computing: Practice and Experience*, 10, 01 2009.

[4] P. Amodio, J. R. Cash, G. Roussos, R. W. Wright, G. Fairweather, I. Gladwell, G. L. Kraut, and M. Paprzycki. Almost block diagonal linear systems: sequential and parallel solution techniques, and applications. *Numerical Linear Algebra with Applications*, 7(5):275–317, 2000.

[5] E. Anderson and Y. Saad. Solving sparse triangular systems on parallel computers. *Int. J. High Speed Comput.*, 1, 06 1989.

[6] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8 (paperback).

[7] H. Anzt, E. Chow, and J. Dongarra. Iterative sparse triangular solves for preconditioning. In J. L. Träff, S. Hunold, and F. Versaci, editors, *Euro-Par 2015: Parallel Processing*, pages 650–661, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[8] H. Anzt, E. Chow, J. Saak, and J. Dongarra. Updating incomplete factorization preconditioners for model order reduction. *Numerical Algorithms*, 73(3):611–630, Nov 2016.

[9] H. Anzt, T. K. Huckle, J. Bräckle, and J. Dongarra. Incomplete sparse approximate inverses for parallel preconditioning. *Parallel Computing*, 71, Oct 2017. doi: 10.1016/j.parco.2017.10.003.

[10] A. Atkinson, A. Donev, and R. Tobias. *Optimum Experimental Designs, with SAS*. Oxford University Press, 01 2007. ISBN 978-0-19-929659-0.

[11] A. Beghi, F. Marcuzzi, and M. Rampazzo. A virtual laboratory for the prototyping of cyber-physical systems. *IFAC-PapersOnLine*, 49(6):63 – 68, 2016.

[12] A. Beghi, F. Marcuzzi, P. Martin, F. Tinazzi, and M. Zigliotto. Virtual prototyping of embedded control software in mechatronic systems: A case study. *Mechatronics*, 43:99 – 111, 2017.

[13] S. Bellavia, D. Bertaccini, and B. Morini. Nonsymmetric Preconditioner Updates in Newton–Krylov Methods for Nonlinear Systems. *SIAM Journal on Scientific Computing*, 33(5):2595–2619, 2011. doi: 10.1137/100789786.

[14] M. Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182, 2002. doi: 10.1006/jcph.2002.7176.

[15] M. Benzi and M. Tuma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, 19 (3):968–994, 1998. doi: 10.1137/S1064827595294691.

[16] E. Bertolazzi, F. Biral, and M. Da Lio. Symbolic-numeric efficient solution of optimal control problems for multibody systems. *Journal of Computational and Applied Mathematics*, 185(2):404 – 421, 2006. ISSN 0377-0427. doi: https://doi.org/10.1016/j.cam.2005.03.019. Special Issue: International Workshop on the Technological Aspects of Mathematics.

[17] C. Bischof and P. Hansen. A block algorithm for computing rank-revealing QR factorizations. *Numerical Algorithms*, 2:371–391, 10 1992. doi: 10.1007/BF02139475.

[18] C. Bischof and G. Quintana-Ortí. Computing Rank-Revealing QR Factorizations of Dense Matrices. *ACM Trans. Math. Softw.*, 24:226–253, 06 1998. doi: 10.1145/290200.287637.

[19] C. Bischof and G. Quintana-Ortí. Algorithm 782: Codes for Rank-Revealing QR Factorizations of Dense Matrices. *ACM Transactions on Mathematical Software*, 24:254–257, 07 1998. doi: 10.1145/290200.287638.

[20] J. R. Bischof. A block QR factorization algorithm using restricted pivoting. In *Supercomputing '89:Proceedings of the 1989 ACM/IEEE Conference on Supercomputing*, pages 248–256, 1989. doi: 10.1145/76263.76290.

[21] T. Bloom, L. Bos, N. Levenberg, and S. Waldron. On the convergence of optimal measures. *Constructive Approximation*, 32(1):159–179, 2010.

[22] T. Bloom, L. Bos, J.-P. Calvi, and N. Levenberg. Polynomial interpolation and approximation in $\mathbb{C}^d$. In *Annales Polonici Mathematici*, pages 53–81, 2012.

[23] H. G. Bock. Recent advances in parameter identification techniques for o.d.e. *Numerical treatment of inverse problems in differential and integral equations*, pages 95–121, 1983.

[24] L. Bos. Some remarks on the fejér problem for lagrange interpolation in several variables. *Journal of approximation theory*, 60(2):133–140, 1990.

[25] L. Bos and M. Vianello. CaTchDes: MATLAB codes for Caratheodory–Tchakaloff Near-Optimal Regression Designs. *SoftwareX*, 10:100349, 2019. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2019.100349.

[26] L. Bos, F. Piazzon, and M. Vianello. Near optimal polynomial regression on norming meshes. In *2019 13th International conference on Sampling Theory and Applications (SampTA)*, pages 1–4. IEEE, 2019.

[27] L. Bos, F. Piazzon, and M. Vianello. Near g-optimal tchakaloff designs. *Computational Statistics*, 35(2):803–819, 2020.

[28] F. Boyer, C. Lapuerta, S. Minjeaud, and B. Piar. A local adaptive refinement method with multigrid preconditionning illustrated by multiphase flows simulations. *ESAIM: Proceedings*, 27:pp 15–53, 2009. doi: 10.1051/proc/2009018.

[29] R. Bro and S. Jong. A Fast Non-negativity-constrained Least Squares Algorithm. *Journal of Chemometrics*, 11:393–401, 09 1997. doi: 10.1002/(SICI)1099-128X(199709/10)11:53.0.CO;2-L.

[30] A. N. Brooks and T. J. Hughes. Streamline upwind/petrov-galerkin formulations for convection dominated flows with particular emphasis on the incompressible navier-stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 32(1):199 – 259, 1982. ISSN 0045-7825. doi: https://doi.org/10.1016/0045-7825(82)90071-8.

[31] A. M. Bruckstein, M. Elad, and M. Zibulevsky. On the uniqueness of nonnegative sparse solutions to underdetermined systems of equations. *IEEE Transactions on Information Theory*, 54(11):4813–4820, 2008. doi: 10.1109/TIT.2008.929920.

[32] J. Burkardt. MONOMIAL: A Matlab Library for Multivariate Monomials. `https://people.sc.fsu.edu/~jburkardt/m_src/monomial/monomial.html`, 2020. Accessed on 1 June 2020.

[33] P. Businger and G. H. Golub. Linear Least Squares Solutions by Householder Transformations. *Numer. Math.*, 7(3):269–276, June 1965. ISSN 0029-599X. doi: 10.1007/BF01436084.

[34] T. T. Cai, G. Xu, and J. Zhang. On recovery of sparse signals via ℓ_1 minimization. *IEEE Transactions on Information Theory*, 55(7):3388–3397, 2009. doi: 10.1109/TIT.2009.2021377.

[35] C. Calgaro, E. Creusè, and T. Goudon. An hybrid finite volume–finite element method for variable density incompressible flows. *Journal of Computational Physics*, 227, 2008. doi: 10.1016/j.jcp.2008.01.017.

[36] C. Calgaro, E. Chane-Kane, E. Creusè, and T. Goudon. l^∞-stability of vertex-based muscl finite volume schemes on unstructured grids: Simulation of incompressible flows with high density ratios. *Journal of Computational Physics*, 229, 2010. doi: 10.1016/j.jcp.2010.04.034.

[37] C. Calgaro, J.-P. Chehab, and Y. Saad. Incremental incomplete LU factorizations with applications. *Numerical Linear Algebra with Applications*, 17, 2010. doi: 10.1002/nla.756.

[38] J.-P. Calvi and N. Levenberg. Uniform approximation by discrete least squares polynomials. *Journal of Approximation Theory*, 152(1):82–100, 2008.

[39] E. J. Candès. The restricted isometry property and its implications for compressed sensing. *Comptes Rendus Mathematique*, 346(9):589–592, 2008. ISSN 1631-073X. doi: https://doi.org/10.1016/j.crma.2008.03.014.

[40] C. Carathéodory. Über den variabilitätsbereich der fourier'schen konstanten von positiven harmonischen funktionen. *Rendiconti Del Circolo Matematico di Palermo (1884-1940)*, 32(1):193–217, 1911.

[41] G. Celant and M. Broniatowski. *Interpolation and Extrapolation Optimal Designs 2: Finite Dimensional General Models*. John Wiley & Sons, Ltd, 2017. ISBN 9781119422327. doi: https://doi.org/10.1002/9781119422327.fmatter.

[42] T. F. Chan. Rank revealing QR factorizations. *Linear Algebra and its Applications*, 88-89:67 – 82, 1987. ISSN 0024-3795. doi: https://doi.org/10.1016/0024-3795(87)90103-0.

[43] S. Chandrasekaran and I. C. F. Ipsen. On Rank-Revealing Factorisations. *SIAM Journal on Matrix Analysis and Applications*, 15(2):592–622, 1994. doi: 10.1137/S0895479891223781.

[44] S. Chen and D. Donoho. Basis pursuit. In *Proceedings of 1994 28th Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 41–44 vol.1, 1994. doi: 10.1109/ACSSC.1994.471413.

[45] S. C. Chen, D. J. Kuck, and A. H. Sameh. Practical parallel band triangular system solvers. *ACM Trans. Math. Softw.*, 4(3):270–277, Sept. 1978. ISSN 0098-3500. doi: 10.1145/355791.355797.

[46] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Rev.*, 43(1):129–159, Jan. 2001. ISSN 0036-1445. doi: 10.1137/S003614450037906X.

[47] A. J. Chorin. Numerical solution of the navier-stokes equations. *Mathematics of Computation*, 22, 1968. doi: 10.1090/s0025-5718-1968-0242392-2.

[48] E. Chow and A. Patel. Fine-Grained Parallel Incomplete LU Factorization. *SIAM Journal on Scientific Computing*, 37(2):C169–C193, 2015. doi: 10.1137/140968896.

[49] E. Chow, H. Anzt, J. Scott, and J. Dongarra. Using jacobi iterations and blocking for solving sparse triangular systems in incomplete factorization preconditioning. *Journal of Parallel and Distributed Computing*, pages –, 2018. ISSN 0743-7315. doi: https://doi.org/10.1016/j.jpdc.2018.04.017.

[50] A. H. Coppola-Owen and R. Codina. Improving Eulerian two-phase flow finite element approximation with discontinuous gradient pressure shape functions. *International Journal for Numerical Methods in Fluids*, 49:1287–1304, Dec. 2005. doi: 10.1002/fld.963.

[51] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference*, ACM '69, pages 157–172, New York, NY, USA, 1969. ACM. doi: 10.1145/800195.805928.

[52] Y. De Castro, F. Gamboa, D. Henrion, R. Hess, and J.-B. Lasserre. Approximate optimal designs for multivariate polynomial regression. *The Annals of Statistics*, 47(1):127–155, 2019.

[53] E. de Klerk and M. Laurent. A survey of semidefinite programming approaches to the generalized problem of moments and their error analysis. In *World Women in Mathematics 2018*, pages 17–56. Springer, 2019.

[54] S. De Marchi, F. Piazzon, A. Sommariva, and M. Vianello. Polynomial meshes: Computation and approximation. In *Proceedings of the 15th International Conference on Computational and Mathematical Methods in Science and Engineering*, pages 414–425, 2015.

[55] J. Demmel, L. Grigori, M. Gu, and H. Xiang. Communication Avoiding Rank Revealing QR Factorization with Column Pivoting. *SIAM Journal on Matrix Analysis and Applications*, 36:55–89, 01 2015. doi: 10.1137/13092157X.

[56] G. Deolmi and F. Marcuzzi. A parabolic inverse convection-diffusion-reaction problem solved using space-time localization and adaptivity. *Applied Mathematics and Computation*, 219(16):8435–8454, 2013.

[57] M. Dessole. Personal webpage. `https://mdessole.github.io/`, 2021. Accessed on 23 September 2021.

[58] M. Dessole and F. Marcuzzi. Fully iterative ILU preconditioning of the unsteady Navier–Stokes equations for GPGPU. *Computers & Mathematics with Applications*, 77(4):907 – 927, 2019. ISSN 0898-1221. doi: https://doi.org/10.1016/j.camwa.2018.10.037.

[59] M. Dessole and F. Marcuzzi. A massively parallel algorithm for Bordered Almost Block Diagonal Systems on GPUs. *Numerical Algorithms*, 2020. ISSN 1572-9265. doi: https://doi.org/10.1007/s11075-020-00931-8.

[60] M. Dessole and F. Marcuzzi. Deviation Maximization for Rank-Revealing QR Factorizations. Preprint, June 2021.

[61] M. Dessole, F. Marcuzzi, and M. Vianello. Accelerating the Lawson-Hanson NNLS solver for large-scale Tchakaloff regression designs. *Dolomites Research Notes on Approximation*, 13:20 – 29, 2020. ISSN 2035-6803. doi: http://dx.doi.org/10.14658/PUPJ-DRNA-2020-1-3.

[62] M. Dessole, F. Marcuzzi, and M. Vianello. dCATCH—A Numerical Package for d-Variate Near G-Optimal Tchakaloff Regression via Fast NNLS. *Mathematics*, 8, 7 2020. doi: https://doi.org/10.3390/math8071122.

[63] M. Dessole, F. Marcuzzi, and M. Vianello. dCATCH: A Numerical Package for Compressed d-Variate Near G-Optimal Regression. `https://www.math.unipd.it/~marcov/MVsoft.html`, 2020. Accessed on 1 June 2020.

[64] M. Dessole, M. Dell'Orto, and F. Marcuzzi. The Lawson-Hanson Algorithm with Deviation Maximization: Finite Convergence and Sparse Recovery. Preprint, August 2021.

[65] H. Dette, A. Pepelyshev, and A. Zhigljavsky. Improving updating rules in multiplicative algorithms for computing d-optimal designs. *Computational Statistics & Data Analysis*, 53(2):312–320, 2008.

[66] J. Dick and F. Pillichshammer. *Digital nets and sequences: discrepancy theory and quasi–Monte Carlo integration.* Cambridge University Press, 2010.

[67] D. Donoho. Compressed sensing. *Information Theory, IEEE Transactions on*, 52:1289 – 1306, 05 2006. doi: 10.1109/TIT.2006.871582.

[68] D. Donoho and X. Huo. Uncertainty principles and ideal atomic decomposition. *IEEE Transactions on Information Theory*, 47(7):2845–2862, 2001. doi: 10.1109/18.959265.

[69] D. L. Donoho and M. Elad. Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ_1 minimization. *Proceedings of the National Academy of Sciences*, 100(5):2197–2202, 2003.

[70] Z. Drmač and Z. Bujanović. On the Failure of Rank-Revealing QR Factorization Software – A Case Study. *ACM Trans. Math. Softw.*, 35(2), July 2008. ISSN 0098-3500. doi: 10.1145/1377612.1377616.

[71] J. A. Duersch and M. Gu. Randomized QR with Column Pivoting. *SIAM Journal on Scientific Computing*, 39(4):C263–C291, 2017. doi: 10.1137/15M1044680.

[72] M. Elad and A. M. Bruckstein. A generalized uncertainty principle and sparse representation in pairs of bases. *IEEE Trans. Inform. Theory*, 48: 2558–2567, 2002.

[73] G. Fairweather and I. Gladwell. Algorithms for Almost Block Diagonal Linear Systems. *SIAM Review*, 46(1):49–58, 2004.

[74] A. V. Fiacco and G. P. McCormick. *Nonlinear programming: sequential unconstrained minimization techniques*. SIAM, 1990.

[75] L. V. Foster. Rank and null space calculations using matrix decomposition without column interchanges. *Linear Algebra and its Applications*, 74:47–71, 1986. ISSN 0024-3795. doi: https://doi.org/10.1016/0024-3795(86)90115-1.

[76] S. Foucart and D. Koslicki. Sparse Recovery by Means of Nonnegative Least Squares. *Signal Processing Letters, IEEE*, 21:498–502, 04 2014. doi: 10.1109/LSP.2014.2307064.

[77] E. Gallopoulos, B. Philippe, and A. Sameh. *Parallelism in Matrix Computations*. Springer, 01 2016. ISBN 978-94-017-7188-7. doi: 10.1007/978-94-017-7188-7.

[78] A. George. Nested dissection of a regular finite element mesh. *Siam Journal on Numerical Analysis*, 10:345–363, 04 1973.

[79] V. Girault and P. Raviart. *Finite element methods for Navier-Stokes equations: theory and algorithms*. Springer series in computational mathematics. Springer-Verlag, 1986. ISBN 9783540157960.

[80] G. Golub. Numerical Methods for Solving Linear Least Squares Problems. *Numer. Math.*, 7(3):206–216, June 1965. ISSN 0029-599X. doi: 10.1007/BF01436075.

[81] G. Golub and C. Van Loan. *Matrix Computations (4th ed.)*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2013. ISBN 9781421407944.

[82] G. Golub, V. Klema, and G. W. Stewart. Rank degeneracy and least squares problems. Technical Report STAN-CS-76-559, Department of Computer Science, Stanford University, 1976.

[83] M. Gu and S. C. Eisenstat. Efficient Algorithms for Computing a Strong Rank-Revealing QR Factorization. *SIAM Journal on Scientific Computing*, 17(4):848–869, 1996. doi: 10.1137/0917055.

[84] J. Guermond, P. Minev, and J. Shen. An overview of projection methods for incompressible flows. *Computer Methods in Applied Mechanics and Engineering*, 195:6011–6045, 09 2006.

[85] J.-L. Guermond and L. Quartapelle. A projection fem for variable density incompressible flows. *Journal of Computational Physics*, 165(1):167 – 188, 2000. ISSN 0021-9991.

[86] J.-L. Guermond and A. Salgado. A splitting method for incompressible flows with variable density based on a pressure poisson equation. *Journal of Computational Physics*, 228, 2009. doi: 10.1016/j.jcp.2008.12.036.

[87] J.-L. Guermond and A. Salgado. Error analysis of a fractional time-stepping technique for incompressible flows with variable density. *SIAM Journal on Numerical Analysis*, 49, 01 2011. doi: 10.1137/090768758.

[88] J.-L. Guermond, A. Marra, and L. Quartapelle. Subgrid stabilized projection method for 2d unsteady flows at high reynolds numbers. *Computer Methods in Applied Mechanics and Engineering*, 195(44):5857 – 5876, 2006. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2005.08.016.

[89] A. Haidar, T. Dong, S. Tomov, P. Luszczek, and J. Dongarra. Framework for Batched and GPU-resident Factorization Algorithms to Block Householder Transformations. In *ISC High Performance*, Frankfurt, Germany, 07-2015 2015. Springer, Springer.

[90] P. C. Hansen. *Rank-Deficient and Discrete Ill-Posed Problems: Numerical Aspects of Linear Inversion.* Society for Industrial and Applied Mathematics, USA, 1999. ISBN 0898714036.

[91] D. P. Hardin, T. Michaels, and E. B. Saff. A comparison of popular point configurations on $\mathbb{S}^2$. *Dolomit. Res. Notes Approx. DRNA*, 9:16–49, 2016.

[92] R. W. Hockney and C. R. Jesshope. *Parallel Computers 2: architecture, programming and algorithms.* CRC Press, 1988.

[93] Y. P. Hong and C.-T. Pan. Rank-Revealing QR Factorizations and the Singular Value Decomposition. *Mathematics of Computation*, 58(197): 213–232, 1992. ISSN 00255718, 10886842.

[94] T. K. Huckle and J. Bräckle. Incomplete sparse approximations of matrices, inverses of matrices, and their factorizations. *Submitted manuscript*, 2015.

[95] L. Jörg and T. Petr. Convergence analysis of krylov subspace methods. *GAMM-Mitteilungen*, 27(2):153–173, 2005. doi: 10.1002/gamm.201490008.

[96] W. Kahan. Numerical linear algebra. *Canadian Mathematical Bulletin*, 9: 757–801, 1966.

[97] J. Kiefer and J. Wolfowitz. The equivalence of two extremum problems. *Canadian Journal of Mathematics*, 12:363–366, 1960.

[98] K. Kontovasilis, R. J. Plemmons, and W. J. Stewart. Block cyclic SOR for Markov chains with p-cyclic infinitesimal generator. *Linear Algebra and its Applications*, 154-156:145 – 223, 1991. ISSN 0024-3795. doi: https://doi.org/10.1016/0024-3795(91)90377-9.

[99] J. B. Lasserre. The moment-sos hierarchy. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pages 3773–3794. World Scientific, 2018.

[100] C. L. Lawson and R. J. Hanson. *Solving least squares problems*, volume 15. SIAM, 1995.

[101] P.-L. Lions. *Mathematical topics in fluid mechanics. - Incompressible models*, volume 1 of *Oxford Lecture Series in Mathematics and Its Applications, 3*. OUP, 1996. ISBN 9780198514879,0198514875.

[102] D. A. Lorenz, M. E. Pfetsch, and A. M. Tillmann. Solving basis pursuit: Heuristic optimality check and solver comparison. *ACM Trans. Math. Softw.*, 41(2), Feb. 2015. ISSN 0098-3500. doi: 10.1145/2689662.

[103] A. Mandal, W. K. Wong, and Y. Yu. Algorithmic searches for optimal designs. *Handbook of design and analysis of experiments*, pages 755–783, 2015.

[104] M. Manguoglu, M. Koyutürk, A. Sameh, and A. Grama. Weighted matrix ordering and parallel banded preconditioners for iterative linear system solvers. *SIAM J. Scientific Computing*, 32:1201–1216, 01 2010.

[105] F. Marcuzzi, M. M. Cecchi, and M. Venturin. An anisotropic unstructured triangular adaptive mesh algorithm based on error and error gradient information. *Mathematics and Computers in Simulation*, 78(5-6):645–652, 2008.

[106] A. Martinez, F. Piazzon, A. Sommariva, and M. Vianello. Quadrature-based polynomial optimization. *Optimization Letters*, 14(5):1027–1036, 2020.

[107] P. Martinsson. Blocked rank-revealing QR factorizations: How randomized sampling can be used to avoid single-vector pivoting. arXiv preprint arXiv:1505.08115, 05 2015.

[108] M. Naumov. Parallel solution of sparse triangular linear systems in the preconditioned iterative methods on the gpu. In *NVIDIA Corp., Westford, MA, USA, Tech. Rep. NVR-2011*, volume 1, 2011.

[109] NVIDIA Corporation. NVIDIA CUDA C programming guide, 2017. URL `https://docs.nvidia.com/cuda/`. Version 9.1.

[110] NVIDIA Corporation. *CUBLAS Library User Guide*, 2018. URL `https://docs.nvidia.com/cuda/cublas/index.html`. Version 9.1.

[111] NVIDIA Corporation. *CUSPARSE Library User Guide*, 2018. URL `https://docs.nvidia.com/cuda/cusparse/index.html`. Version 9.1.

[112] NVIDIA Corporation. *NVIDIA CUDA C Programming Guide*, 2019. URL `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`. Version 10.1.

[113] W. Pleśniak. Multivariate Jackson Inequality. *Journal of computational and applied mathematics*, 233(3):815–820, 2009.

[114] E. Polizzi and A. H. Sameh. A parallel hybrid banded system solver: the spike algorithm. *Parallel Computing*, 32(2):177 – 194, 2006. ISSN 0167-8191. doi: https://doi.org/10.1016/j.parco.2005.07.005. Parallel Matrix Algorithms and Applications (PMAA'04).

[115] L. F. Portugal, J. J. Júdice, and L. N. Vicente. A Comparison of Block Pivoting and Interior-Point Algorithms for Linear Least Squares Problems with Nonnegative Variables. *Mathematics of Computation*, 63(208):625–643, 1994. ISSN 00255718, 10886842.

[116] F. Pukelsheim. *Optimal Design of Experiments*. Society for Industrial and Applied Mathematics, 2006. doi: 10.1137/1.9780898719109.

[117] M. Putinar. A note on tchakaloff's theorem. *Proceedings of the American Mathematical Society*, 125(8):2409–2414, 1997.

[118] G. Quintana-Ortí, E. S. Quintana-Ortí, R. A. V. D. Geijn, F. G. V. Zee, and E. Chan. Programming matrix algorithms-by-blocks for thread-level parallelism. *ACM Trans. Math. Softw.*, 36(3), July 2009. ISSN 0098-3500. doi: 10.1145/1527286.1527288. URL `https://doi.org/10.1145/1527286.1527288`.

[119] G. Quintana-Ortí, X. Sun, and C. H. Bischof. A BLAS-3 Version of the QR Factorization with Column Pivoting. *SIAM Journal on Scientific Computing*, 19(5):1486–1494, 1998. doi: 10.1137/S1064827595296732.

[120] M. Radons. Direct solution of piecewise linear systems. *Theoretical Computer Science*, 626:97–109, 2016. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2016.02.009.

[121] Y. Saad. A Flexible Inner-outer Preconditioned GMRES Algorithm. *SIAM J. Sci. Comput.*, 14(2):461–469, Mar. 1993. ISSN 1064-8275. doi: 10.1137/0914028.

[122] Y. Saad. *Iterative methods for sparse linear systems.* Society for Industrial and Applied Mathematics, 2 edition, 2003. ISBN 9780898715347,0898715342.

[123] Á. Santos-Palomo and P. Guerrero-García. Solving a sequence of sparse linear least squares subproblems. Technical Report MA-03-03, Dept. Appl. Math., Univ. Málaga, 2003.

[124] R. Schreiber and C. Van Loan. A Storage-Efficient *WY* Representation for Products of Householder Transformations. *SIAM Journal on Scientific and Statistical Computing*, 10(1):53–57, 1989. doi: 10.1137/0910005.

[125] M. Slawski. *Topics in learning sparse and low-rank models of non-negative data.* PhD thesis, Saarland University, Saarbrücken, Germany, 2014.

[126] M. Slawski. Nonnegative least squares: Comparison of algorithms. `https://sites.google.com/site/slawskimartin/code`, 2019. Accessed on 1 June 2020.

[127] M. Slawski and M. Hein. Non-negative least squares for high-dimensional linear models: Consistency and sparse recovery without regularization. *Electronic Journal of Statistics*, 7(1):3004–3056, 2013. Cited By :81.

[128] A. Sommariva and M. Vianello. Compression of multivariate discrete measures and applications. *Numerical Functional Analysis and Optimization*, 36(9):1198–1223, 2015.

[129] J. Stoer. On the Numerical Solution of Constrained Least-Squares Problems. *SIAM Journal on Numerical Analysis*, 8(2):382–411, 1971. ISSN 00361429.

[130] G. Strang. On the construction and comparison of difference schemes. *SIAM Journal on Numerical Analysis*, 5, 09 1968. doi: 10.1137/0705041.

[131] V. Tchakaloff. Formules de cubatures mécaniques à coefficients non négatifs. *Bull. Sci. Math*, 81(2):123–134, 1957.

[132] R. Temam. Sur l'approximation de la solution des équations de navier-stokes par la méthode des pas fractionnaires (i). *Archive for Rational Mechanics and Analysis*, 32, 2 1969. doi: 10.1007/bf00247678.

[133] R. Thompson. Principal submatrices IX: Interlacing inequalities for singular values of submatrices. *Linear Algebra and its Applications*, 5(1): 1–12, 1972. ISSN 0024-3795. doi: https://doi.org/10.1016/0024-3795(72) 90013-4.

[134] R. Tibshirani. Regression Shrinkage and Selection via the LASSO. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996. ISSN 00359246.

[135] A. M. Tillmann and M. E. Pfetsch. The computational complexity of the restricted isometry property, the nullspace property, and related concepts in compressed sensing. *IEEE Transactions on Information Theory*, 60(2): 1248–1259, 2014. doi: 10.1109/TIT.2013.2290112.

[136] D. Titterington. Algorithms for computing D-optimal designs on a finite design space. In *In Proceedings of the 1976 Conference on Information Science and Systems*, volume 3, pages 213–216. Hopkins University: Baltimore, MD, USA, 1976.

[137] B. Torsney and R. Martín-Martín. Multiplicative algorithms for computing optimum designs. *Journal of Statistical Planning and Inference*, 139 (12):3947–3961, 2009.

[138] J. Tropp. Greed is good: algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50(10):2231–2242, 2004. doi: 10.1109/TIT.2004.834793.

[139] G. Tryggvason. Numerical simulations of the rayleigh-taylor instability. *Journal of Computational Physics*, 75(2):253 – 282, 1988. ISSN 0021-9991. doi: https://doi.org/10.1016/0021-9991(88)90112-X.

[140] M. H. Van Benthem and M. R. Keenan. Fast algorithm for the solution of large-scale non-negativity-constrained least squares problems. *Journal of Chemometrics*, 18(10):441–450, 2004. doi: https://doi.org/10.1002/cem.889.

[141] A. C. N. van Duin. Scalable Parallel Preconditioning with the Sparse Approximate Inverse of Triangular Matrices. *SIAM Journal on Matrix Analysis and Applications*, 20(4):987–1006, 1999. doi: 10.1137/S0895479897317788.

[142] J. Varah. A lower bound for the smallest singular value of a matrix. *Linear Algebra and its Applications*, 11(1):3 – 5, 1975. ISSN 0024-3795. doi: https://doi.org/10.1016/0024-3795(75)90112-3.

[143] M. Wang and A. Tang. Conditions for a unique non-negative solution to an underdetermined system. In *Proceedings of the 47th Annual Allerton Conference on Communication, Control, and Computing*, Allerton'09, page 301–307. IEEE Press, 2009. ISBN 9781424458707.

[144] M. Wang, W. Xu, and A. Tang. A Unique "Nonnegative" Solution to an Underdetermined System: From Vectors to Matrices. *IEEE Transactions on Signal Processing*, 59(3):1007–1016, Mar. 2011. doi: 10.1109/TSP.2010.2089624.

[145] S. Wright. A Collection of Problems for Which Gaussian Elimination with Partial Pivoting is Unstable. *SIAM Journal on Scientific Computing*, 14 (1):231–238, 1993. doi: 10.1137/0914013.

[146] S. J. Wright. Stable Parallel Algorithms For Two-Point Boundary Value Problems. *SIAM J. Sci. Statist. Comput*, 13:742–764, 1992.

[147] J. Xiao, M. Gu, and J. Langou. Fast parallel randomized qr with column pivoting algorithms for reliable low-rank matrix approximations. In *2017 IEEE 24th international conference on high performance computing (HiPC)*, pages 233–242. IEEE, 2017.