# High Performance and Reliable Algebraic Computing

Clément Pernet

**UNIVERSITÉ JOSEPH FOURIER (GRENOBLE 1)**

**MÉMOIRE D'HABILITATION À DIRIGER DES RECHERCHES**

Spécialité : Informatique

---

# HIGH PERFORMANCE AND RELIABLE ALGEBRAIC COMPUTING

---

Présentée et soutenue publiquement par

**CLÉMENT PERNET**

le 21 novembre 2014.

Après avis des rapporteurs :

| | | | |
|---|---|---|---|
| **M.** | **Daniel Augot** | **Directeur de recherches** | **Inria Saclay - Île-de-France** |
| **M.** | **Mark Giesbrecht** | **Professeur** | **Univ. of Waterloo, ON. Canada** |
| **Mme.** | **Laura Grigori** | **Directrice de recherches** | **Inria, Paris - Rocquencourt** |

Devant la commission d'examen constituée de :

| | | | |
|---|---|---|---|
| **M.** | **Daniel Augot** | **Directeur de recherches** | **Inria Saclay - Île-de-France** |
| **M.** | **Jean-Guillaume Dumas** | **Professeur** | **Univ. Jospeh Fourier, Grenoble** |
| **M.** | **Jean-Charles Faugère** | **Directeur de recherches** | **Inria, Paris - Rocquencourt** |
| **M.** | **Mark Giesbrecht** | **Professeur** | **Univ. of Waterloo, ON. Canada** |
| **Mme.** | **Laura Grigori** | **Directrice de recherches** | **Inria, Paris - Rocquencourt** |
| **M.** | **Erich L. Kaltofen** | **Professeur** | **North Carolina State Univ., USA** |
| **Mme.** | **Brigitte Plateau** | **Professeure** | **Grenoble INP** |

# Contents

# Introduction

Computer algebra is about computing exactly with mathematical objects, such as rational numbers, polynomials, or finite field elements. It has developed, motivated by a wide range of applications, ranging from symbolic manipulation, algebraic cryptanalysis, computational number theory, linear programming, formal proof verification, ...

In this process, it has benefited from algorithms and techniques of various other fields: Wiedemann algorithm [Wie86], a breakthrough in solving sparse linear systems over a finite field is a great illustration, as it combines an adaptation of the iterative Krylov methods from numerical linear algebra, and the Berlekamp/Massey algorithm developed in coding theory to efficiently decode Reed-Solomon codes.

These interactions have happened, since then, in both ways: recent progresses in polynomial lattice reduction developed in computer algebra have proven useful for the list decoding of algebraic codes [Cho+14]; symbolic-numeric approaches and iterative refinement combine exact and approximate computations to improve the accuracy of numerical computations [GSW12] or produce exact certificates of numerical global optimization problems [Kal+12].

This monograph summarizes our contribution to the design of high performance computer algebra which naturally crosses the border between those three areas: exact and numerical linear algebra overlap in the big picture, but also can diverge in interesting ways; we will also investigate coding theory to design fault tolerant parallelizations. The focus is on efficiency in practice: improvement at every level are studied, including the asymptotic time and space complexities, their leading constants, but also implementation techniques to better harness the various layers of parallelization in modern computer architectures: from SIMD instructions to multi-core and multi-processor shared memory parallelism. At this early stage, we still did not address the design for distributed computation in its specificites, even if it shares the care of minimizing communications taken into account in the shared memory context. The implementations, which we refer to, are made available in the mainstream distribution of the linear algebra libraries we contribute to: `fflas-ffpack` [FFL14] and `LinBox` [Lin12]. These routines are being used in a variety of contexts, and serve as linear algebra kernels for the general purpose open-source mathematical software `Sage` [Ste+14].

Our work has been vastly supported by the collaboration framework of the HPAC project (High Performance Algebraic Computing, ANR 11 BS02 013), funding the Ph.D. program of Ziad Sultan, co-supervised by Jean-Guillaume Dumas and myself. Ziad Sultan's research is on the design of efficient parallel dense exact linear algebra kernels. He contributed to most algorithmic and implementation aspects of the parallel gaussian elimination and the computation of rank profiles that will be presented here. Our collaboration with Erich Kaltofen on rational function and sparse evaluation codes was supported by the QOLAPS Inria associate team.

## Notations

For a field $\mathsf{K}$, $\mathsf{K}[X]$ denotes the ring of univariate polynomials over $\mathsf{K}$ and $\mathsf{K}[X]_{\leq d}$, the sub-ring of such polynomials with degree less or equal than $d$. The ring of $m \times n$ matrices over $\mathsf{K}$ is denoted by $\mathsf{K}^{m\times n}$. Finally, the notation $\mathbb{Z}_m$ represents the set of integers between 0 and $m-1$, viewed as the representatives of the congruence classes of $\mathbb{Z}/m\mathbb{Z}$. The arithmetic over these sets is that of $\mathbb{Z}/m\mathbb{Z}$.

## Model of computation, complexity and computation speed

We will express the time and space complexities of algorithms informally without specifying rigorously the computational model, but bearing in mind the RAM and PRAM models. Over a finite field, we usually consider field operations as atomic and therefore express the time complexity in terms of number of field operations. Over ring or fields of variable size, we use bit-complexity for $\mathbb{Z}$ and $\mathbb{Q}$ and for $\mathsf{K}[X]$ and $\mathsf{K}(X)$, we count operations over $\mathsf{K}$.

We denote by $\mathsf{M}(n)$ the number of field operations required to multiply two degree $n$ polynomials over a field and by $\mathsf{MM}(n) = O(n^\omega)$ the number of field operations to multiply two $n \times n$ matrices over a field. The exponent $\omega$ stands for any feasible exponent for which an algorithm exist. The currently best known value is $\omega = 2.3728639$ [LeG14]. It slightly differs from the definition in [BCS97] where the complexity exponent for a problem is defined as an infinimum over all feasible exponents. Indeed, the latter definition does not distinguish the presence of logarithmic factors in a big-O complexity, which we will care about. We recall that the soft-O notation $f = \tilde{O}(g)$ means $f = O(g \log^\alpha g)$ for some $\alpha \geq 0$, conveniently absorbing logarithmic and poly-logarithmic factors arising e.g. from FFT based fast integer or polynomial arithmetic.

Lastly, we will often illustrate the efficiency of a linear algebra implementation by presenting computation speed rather than time, as this allows to represent and compare efficiency on large ranges of dimensions. We use the effective Gfops (Giga field operations per second) metric defined as $Gfops = \frac{\#\text{ of field ops using classic matrix product}}{\text{time}}$. This is $\frac{2mnk}{\text{time}}$ for the product of an $m \times k$ by a $k \times n$ matrix, and $\frac{2n^3}{3\text{time}}$ for the Gaussian elimination of a full rank $n \times n$ matrix. We note that the effective Gfops are only true Gfops (consistant with the Gflops of numerical computations) when the classic matrix multiplication algorithm is used. Still this metric allows us to compare all algorithms on a uniform measure: the inverse of the time, normalized by an estimate of the problem size; the goal here is not to measure the bandwidth of our usage of the processor's arithmetic instructions.

Chapter 1

# Linear Algebra

## 1.1 Design principles

Our contribution to the development of exact linear algebra algorithmic and implementations, reported in this chapter, is based the following thesis:

> *Efficient implementations in exact linear algebra are obtained by making effective the theoretical algorithmic reductions used in complexity analysis.*

**Reductions.** Before the rise of asymptotically fast linear algebra algorithms, most computations over a field shared the same time complexity $O(n^3)$ using often unrelated algorithms (matrix multiplication, Gaussian elimination for solving linear systems, QR decomposition for eigenvalue problems, Danilevskiĭ's elimination for the characteristic polynomial, etc). The breach opened with Strassen's $O(n^{\log_2 7})$ algorithm for matrix multiplication [Str69] led to design reductions of all other computations to matrix multiplication, for them to enjoy the same complexity. Matrix multiplication became a building block.

The same also happened for matrices with coefficients of variable size (polynomials or integers). A first reduction based on evaluation-interpolation techniques led to complexities equal to the algebraic complexity times the size of the output. For the determinant of an integer matrix, this amounts to $n$ times the cost of multiplying two such matrices. Another breach was opened with Storjohann's algorithm [Sto03; Sto05], reducing the bit complexity of linear system solving to essentially that of matrix multiplication: $\tilde{O}(n^\omega \log \|A\|_\infty)$. It made way to a series of reductions of most linear algebra problems over variable size domains: the Smith normal form, the determinant [Sto03; Sto05], computing a small null space basis [SV05], etc, reducing their complexities by up to a factor $n$. Here the building block became the linear system solving.

In black-box linear algebra, where matrices are only considered as linear operators, the minimal polynomial computed by Wiedemann's algorithm [Wie86; KS91], also became the building block to which most other problem reduce thanks to preconditioning techniques [Che+02].

If the above thesis sounds trivial at first — as complexity analysis is first meant to quantify the efficiency of an algorithm, and efficient implementations should therefore adhere to the asymptotically best algorithms and use the related reductions — it still goes against a common belief, that asymptotic complexity estimates, are no longer connected with any practicable algorithm for several reasons:

- the constants hidden by the $O()$ notation can get so large, that the asymptotically best algorithm will only become competitive for instances of a size that would never fit in a computer's memory;
- high performance numerical linear algebra has developed using cubic time algorithms only, avoiding Strassen's sub-cubic time matrix multiplication for its lesser numerical stability.

**Building blocks.** The key point of this thesis is that, even when the asymptotically best algorithms for the building blocks are not practicable, these building block problems happen to be solved by other algorithmic variants, delivering the best efficiency in practice, thus making the whole reduction structure relevant also for efficient implementations.

As an example, matrix multiplication over a fixed size coefficient domain reaches near peak efficiency of nowadays general purpose processors for it fits best with their design: repeated multiply and accumulation operations benefit from SIMD vectorization intructions, and instruction pipelining is not interrupted by any branching. Also, the cubic amount of computations for a quadratic amount of data ensures that the multiple cache memory layers can be efficiently used to overlap expensive memory accesses by numerous computations. Based on these facts, the design of dense numerical linear algebra, formerly focused on vector operations (BLAS levels 1 [Law+79] and 2 [Don+85]), changed with the creation of the level 3 BLAS [Don+87] built around the `gemm` routine (GEneral Matrix Matrix multiplication) and reductions to it.

**Which reduction in practice?** Still reductions in numerical dense linear algebra use block iterative algorithms, as e.g. for the Gaussian elimination or the QR decomposition in `LAPACK` [And+90] or `PLASMA` [Agu+09], and not the recursive reduction used to obtain the $O(n^\omega)$ complexity. Indeed, the choice of using a cubic time matrix multiplication, for numerical stability purposes, make the recursive reduction perform equally to any block iterative algorithm: the arithmetic cost function is associative, and any choice of block decomposition yield the same amount of operations (the product $A\left[B_1 B_2\right]$ takes essentially the same time as the two products $AB_1$ and $AB_2$ done in sequence). The block iterative reduction is preferred for its better control over the block sizes, used for cache aware algorithms and a simpler thread management for its parallelization. In exact dense linear algebra, the use of not only sub-cubic algorithms such as Strassen's [Str69], but also of delayed modular reductions make the cost no longer associative: the larger the matrix product, the more efficient the computation. The recursive reduction therefore naturally stands as the reduction of choice.

We summarize in figure 1.1 some of the standard problems in exact linear algebra and show their reductions to the corresponding building blocks. References to articles or paragraphs of this monograph point to either the theoretical reductions, or some reductions used in practice.

**Making theoretical reductions effective.** The key steps in making these theoretical reductions effective will be illustrated in this chapter with our contributions:

1. producing new reduction schemes: for the computation of the characteristic polynomial of a dense matrix (section 1.5.1), or a blackbox [R-DPS09], for the computation of the Frobenius normal form over a field and its transformation matrix (section 1.5.2), or for the rank deficient Gaussian Elimination (section 1.4)
2. improving existing reductions efficiency by tuning the constant factor of both time and space complexities: for the characteristic polynomial ([R-DPW05; R-PS07a]), rank deficient Gaussian eliminations and echelon forms (section 1.4) or for the Hermite normal form [J-PS10],
3. fine tuning the building blocks: our work focused on dense matrix multiplication (section 1.2).

Along these lines we took more specifically care of the following aspects on both building blocks and reductions:

**memory footprint:** with memory efficient scheduling of matrix multiplication algorithms [R-Boy+09], and in-place reductions of LU decompositions [J-JPS13; R-DPS13];

**amount of modular reductions:** in matrix multiplications [J-DGP08] and reductions of LU decompositions [J-JPS13; R-DPS13], (section 1.3.2);

**data locality:** reducing distant memory accesses, and avoiding permutations [R-DPS13; R-Dum+14].

**Design for parallel computing.** The development of parallel versions of these implementations challenges the above design based on reductions. Some building blocks, like matrix multiplication over a finite field, are easily parallelized and scale nicely with the number of processors, but others, like the linear system solving over the integers, will less likely remain as efficient in parallel. Moreover, even when building blocks perform well in parallel, the reduction scheme of other computations to it often have a rather long critical path and plugging parallel building blocks to them, yields a poor efficiency. On the other hand, the best parallel algorithms in theory are still rarely of any relevance for practical implementations, as they often involve an asymptotically large amount of processors.

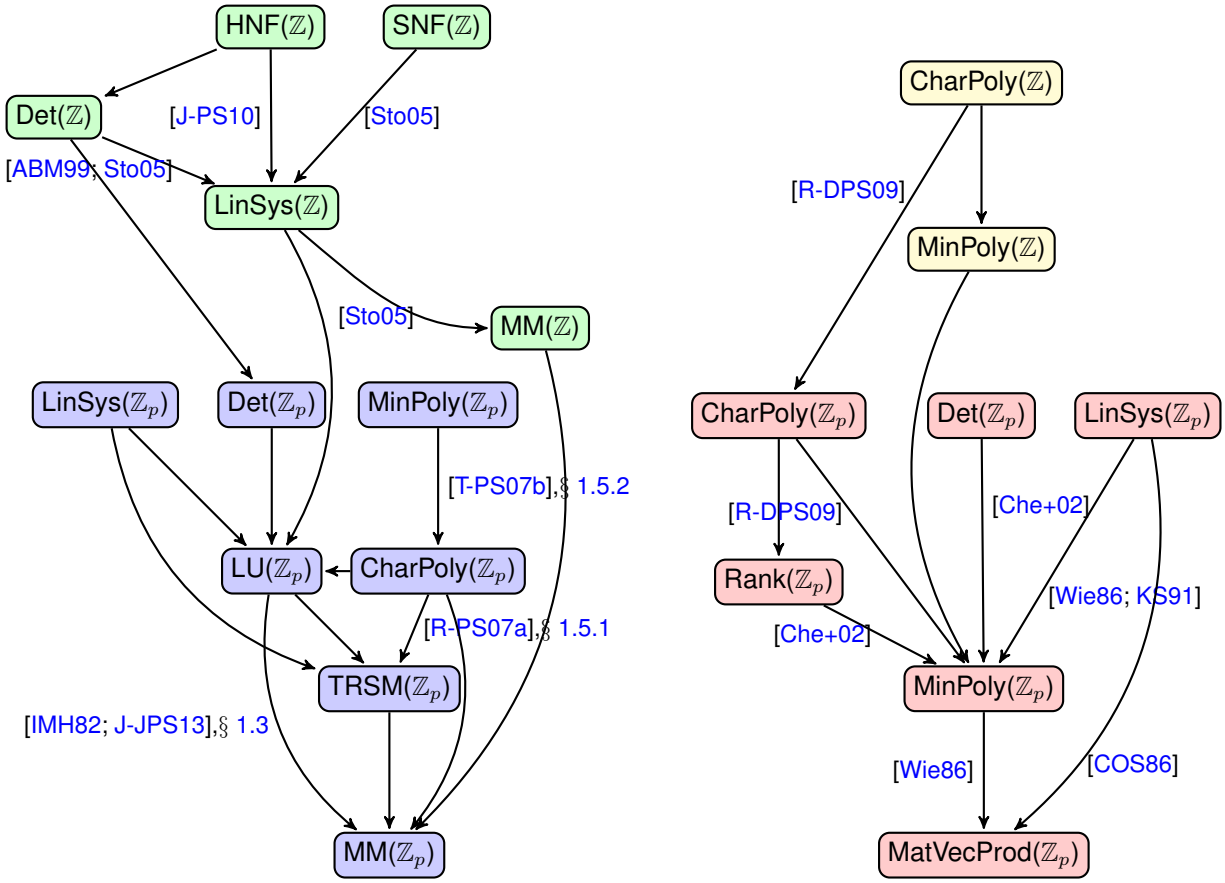



Figure 1.1: Reductions of some classical problems in dense linear algebra (left) and black-box linear algebra (right) over a field (bottom part) and over the integers (upper part).

The solution that we use is to keep using the same reduction structure as in sequential computing, but incorporate the parallelization at a higher level than the building blocks. As an illustration, in dense numerical linear algebra, the shared memory parallelization of `LAPACK`'s routine used to rely on efficient parallelization of the BLAS building block routines. However, recent developments (as in the `PLASMA` library [Agu+09]) trying to address the heterogeneity of the tasks and reduce the number of synchronizations have given up on this approach. Instead, their design introduce the parallelization at a coarser grain, e.g. the Gaussian elimination block algorithm, and use sequential BLAS building block routines.

Our approach is similar in the sense that we will keep using the similar reduction structure and incorporate the parallelization at higher levels than the building blocks. However, as we need to keep the reductions recursive, this means that the parallelization will not happen at only one algorithmic level but possibly all levels. This imposes stronger constraints on the parallel programming language, and on the management of synchronizations, as we will see in section 1.4.4. We use for our implementations the parallel task semantic offered by the OpenMP standard. From our experiments, using tasks was never slower than parallel loop structures, and most often delivered higher efficiency. We will denote by `libgomp` the GNU implementation of the OpenMP API for multi-platform shared-memory parallel programming in C/C++ and Fortran. Alternatively, we will also use `libkomp` [BGD12], a competitive implementation of the OpenMP norm, based on the `XKaapi` parallel runtime [Gau+12], that reduces the overhead of the OpenMP directives and handles more efficiently threads creation, synchronization and management.

**Arithmetic.** Another general guideline in the design of efficient exact linear algebra is the interplay of various arithmetics, and reductions between them and finally to fixed size machine word arithmetic, delivering peak efficiency. More precisely, the most frequent type of arithmetic used are the following:

**integers:** using multiprecision or arbitrary precision representations, and asymptotically fast arithmetic algorithms as proposed in the libraries `gmp` [Gt14] or `mpir` [Ht14]. In order to avoid the growth of intermediate computations, and also to benefit from algorithms with divisions, computations with integers may be reduced to multiple computations over word-size prime fields, using the Chinese Remainder Theorem.

**large finite field:** using dedicated arithmetic, especially when the bit size is only a small factor of a machine word-size, or using multi-precision integer arithmetic and modular reductions. The latter can be done via Chinese remaindering and word-size finite fields; the modular reductions can be performed over the multi-modular representation without reconstructing the integers.

**word-size finite field:** an element is embedded into an integer, the unique residue in $\{-\frac{p-1}{2} \dots \frac{p-1}{2}\}$ for elements of a prime field $\mathbb{Z}_p$, or using Kronecker substitution for elements of a Galois field $\mathbb{F}_{p^k}$ represented as polynomials over $\mathbb{Z}_p$ [R-DGP02; DFS11]. Integer arithmetic is then used as long as possible, and modular reductions are delayed until the point where the result could overflow the capacity of the machine word. The basic type can be 32 or 64 bit integers, or single or double precision floating point numbers. Although the representation capacity of integers on floating point types is limited to 24, (respectively 53) bits on a 32 (resp. 64) bit word, the arithmetic units in most of nowadays CPUs and GPUs perform more efficiently on them than on integer types [Gio14].

**very small finite field using boolean word arithmetic:** bit-packing (fitting $n/b$ elements of $b$ bits in a word of $n$ bits) and bit-slicing (storing $n$ elements of $b$ bits in $b$ words of $n$ bits, each of which storing one bit position) are used. The arithmetic is then performed using boolean word instructions. Over $\mathbb{Z}_2$, libraries like `m4ri` [ABH10] combine boolean arithmetic, pre-computed tables and the technique of the Four Russians [Arl+70]. A generalization for small Galois field in characteristic 2 is proposed in [Alb12]. Over $\mathbb{Z}_3, \mathbb{Z}_5, \mathbb{Z}_7$, bit-slicing can be combined with the binary representations proposed in [BB09], optimizing the number of boolean operations required to perform the basic arithmetic operations.

**very small finite field using integer arithmetic:** Another variant of bit-packing for slightly larger fields is to use Kronecker substitution to store a few elements in a word, leaving room for them to grow after a few integer arithmetic operations [DFS11]. Although efficient, this approach requires to compress either the row or the column dimension of a matrix depending whether it is on the left or right hand side of a multiplication, which is constraining for a general purpose building block routine.

A typical path of reductions is e.g. multi-precision integers that reduce via Chinese Remainder Theorem to computations with word size prime fields, which in turn use delayed modular reductions and integer operations performed by the floating point units of the CPU.

In the remaining of this chapter we will focus on our contributions illustrating some aspects of these general guidelines for the development of high performance exact linear algebra software. These contributions were made concrete in the development of the `fflas-ffpack` [FFL14] and `LinBox` [Lin12] libraries.

## 1.2 Matrix multiplication

### 1.2.1 Design of `fgemm`

The design of matrix multiplication routines over a word size finite field, the main building block for computations in exact dense linear algebra, illustrates the following guidelines described earlier:

1. finite field arithemtic is reduced to integer arithmetic with delayed modular reductions,
2. integer arithmetic is performed by floating point units (taking advantage of SIMD instructions),
3. computations are structured in blocks so as to optimize cache usage,
4. asymptotically fast algorithms are used.

In our early work on the topic [R-DGP02; J-DGP08], we proposed to harness the efficiency of existing numerical linear algebra routines. The BLAS interface provides a standardized way to link against any

implementation of a well-defined set of basic routines, including the `sgemm` and `dgemm` routines for matrix multiplication with single and double precision floating point numbers. As it lies at the core of high performance numerical computing, finely tuned implementations for most architectures are therefore available, and likely will be in the near future. In this process, items 2 and 3 of the list above are forwarded to a BLAS library.

As for item 1, several options are available to perform the reduction of a floating point number $x$ modulo and integer $p$. The instruction `fmod` of the `cmath.h` library requires an expensive library call. Instead, storing the floating point inverse $i$ of $p$, and computing $x - \lfloor x \times i \rfloor p$ allows to compute several reductions in parallel using SIMD instructions on the SSE or AVX registers. This approach, recently implemented by Brice Boyer, Pascal Giorgi and Bastien Vialla in `fflas-ffpack`, has produced significant improvements, reducing the overhead between finite field and numerical computations especially for small dimensions. We also refer to [HLQ14] for further details on the arithmetic of modular reductions and their vectorization on SIMD architectures.

Lastly, sub-cubic time complexity is obtained using the $O(n^{\log_2 7})$ Winograd's variant of Strassen's algorithm [GG13, Algorithm 12.1]. This recursive algorithm, is applied down to a threshold on the dimension, experimentally set (typically near $n = 1000$), where the classic algorithm, based on the BLAS is then applied. This naturally impacts the frequency at which delayed modular reductions need to be done. With a classic matrix multiplication and field elements of $\mathbb{Z}_p$ represented as integers in $\{-\frac{p-1}{2} \dots \frac{p-1}{2}\}$ on a type with a mantissa of $m$ bits, the condition is that the modular reduction in a scalar product of dimension $k$ can be delayed to the end as long as

$$k \left(\frac{p-1}{2}\right)^2 < 2^m.$$

When applying $\ell$ recursive levels of Strassen-Winograd's algorithm, we showed [Per06; J-DGP08] that some intermediate computations could grow above this bound, and the condition becomes

$$9^\ell \left\lfloor \frac{k}{2^\ell} \right\rfloor \left(\frac{p-1}{2}\right)^2 < 2^m. \tag{1.1}$$

which imposes to perform a factor of about $(9/2)^\ell$ as many modular reductions.

However, this bound is pessimistic in the sense that it applies the worst case analysis of one branch in the recursion tree, to all other branches, where intermediate computations do not grow as fast. We therefore improved the implementation, making recursive calls to `fgemm` maintain bounds on the minimum and maximum values of its input, so as to decide to perform modular reductions with tighter estimates.

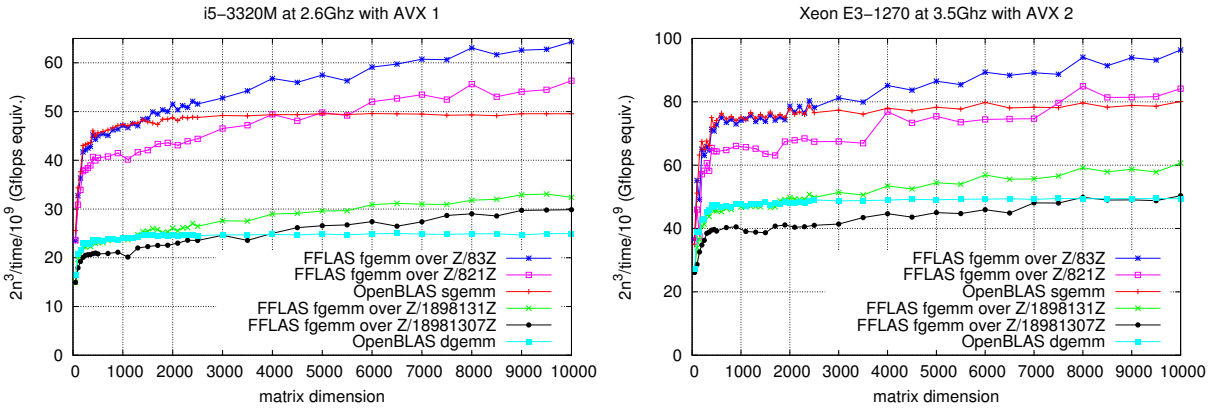



Figure 1.2: Computation speed of `fgemm` compared with the numerical BLAS routines `dgemm` and `sgemm` of OpenBLAS-0.2.9 for the operation $C \leftarrow C - AB$ with square matrices of order $n$.

Figure 1.2 compares the computation speed of our implementations in `fflas-ffpack`, called `fgemm`, on two architectures and various field sizes. Both architectures have 256 bits AVX registers for SIMD instructions, the right one offering the recent `fma` (fused multiply and accumulation) of the AVX2 set.

Small fields use single precision floating points and the BLAS routine `sgemm` and large fields use `double` and the BLAS routine `dgemm`. The numerical routines deliver a steady computing speed. The overhead of computing over a finite field appears almost unnoticeable for small matrix dimensions. Then with larger dimensions, the effective speed of `fgemm` keeps increasing with the matrix dimension, indicating the speed-up of Strassen-Winograd's algorithm over the naive $O(n^3)$ algorithm. The fields $\mathbb{Z}_{83}$ in single precision, and $\mathbb{Z}_{1898131}$ in double precision require few modular reductions (only one for a classical product of dimension up to about $n = 10000$). On the other hand the fields $\mathbb{Z}_{821}$ and $\mathbb{Z}_{18981307}$ require many more reductions (one every 100 multiplication in a classical product). The smaller speed for these latter two fields shows the overhead of the modular reductions but is contained to a small impact on the efficiency and does not prevent a sub-cubic asymptotic behavior.

More recently, Brice Boyer and Jean-Guillaume Dumas also showed how Bini & Al. $O(n^{2.7799})$ algorithm [Bin+79] could be efficiently put into practice [BD14] and also offer interesting speed-up for prime fields of size near 10 bits.

### 1.2.2 Memory efficient schedules

Strassen-Winograd's algorithm requires external temporary memory allocations which can penalize the efficiency of a computation, especially with large matrices, near the storage capacity of the main memory.

We proposed new schedules in [R-Boy+09], improving over the preceding state of the art: [Dou+94] for the simple product $C \leftarrow AB$ and [Hus+96] for the more general form $C \leftarrow \alpha AB + \beta C$, especially useful to compute Schur complements in many reductions. The techniques introduced consist in adding a few extra additions and the possibility to overwrite intermediate computations or possibly the input matrices (when given the permission to). For example we produced a schedule computing $C \leftarrow AB$ with no extra allocation, provided that $A$ and $B$ could be overwritten (the previously best extra memory requirement was of $2/3n^2$ field elements [Dou+94]), or reduced the memory requirement for $C \leftarrow \alpha AB + \beta C$ from $n^2$ to $2/3n^2$. The overhead in the time complexity only affects non dominant terms. A combination of a recursive classic algorithm and Strassen-Winograd's yields a fully in-place algorithm for $C \leftarrow AB$ without overwriting the input matrices, at the expense of a slightly larger constant leading constant (7.42 instead of 6) in the time complexity. These results are summarized in table 1.1, with references to the corresponding algorithms in [R-Boy+09]. The new schedules have been discovered by hand, but verified by a pebble game program. Exhaustive search, when succesfully terminated, could also prove their optimality. We also refer to Brice Boyer's thesis [Boy12] for the latest development on the topic.

| | Algorithm | Input matrices | # temp. | extra mem. | # extra alloc. | arithmetic complexity |
|---|---|---|---|---|---|---|
| $A \times B$ | [Dou+94] | Constant | 2 | $\frac{2}{3}n^2$ | $\frac{2}{3}(n^{2.808} - n^2)$ | $6n^{2.808} - 5n^2$ |
| | [R-Boy+09, Tab. 3] | Both Overwr. | 0 | 0 | 0 | $6n^{2.808} - 5n^2$ |
| | [R-Boy+09, Tab. 4, 5] | $A$ or $B$ Overwr. | 1 | $\frac{1}{3}n^2$ | $\frac{1}{4}n^2 \log_2(n)$ | $6n^{2.808} - 5n^2$ |
| | [R-Boy+09, Alg. 7.1] | Constant | 0 | 0 | 0 | $7.2n^{2.808} - 13n^2$ |
| $\alpha A \times B + \beta C$ | [Hus+96] | Constant | 3 | $n^2$ | $\frac{2}{3}n^{2.808} + n^{2.322} - \frac{5n^2}{3}$ | $6n^{2.808} - 4n^2$ |
| | [R-Boy+09, Tab. 6] | Both Overwr. | 2 | $\frac{2}{3}n^2$ | $\frac{1}{2}n^2 \log_2(n)$ | $6n^{2.808} - 4n^2 + \frac{n^2 \log_2(n)}{2}$ |
| | [R-Boy+09, Tab. 7] | $B$ Overwr. | 2 | $\frac{2}{3}n^2$ | $2n^{2.322} - 2n^2$ | $6n^{2.808} - 4n^2 + \frac{n^2 \log_2(n)}{2}$ |
| | [R-Boy+09, Tab. 9] | Constant | 2 | $\frac{2}{3}n^2$ | $\frac{2}{9}n^{2.808} + 2n^{2.322} - \frac{22n^2}{9}$ | $6n^{2.808} - 4n^2 + \frac{4n^2 \log_2(n)}{3}$ |
| | [R-Boy+09, Alg. 7.2] | Constant | N/A | $\frac{1}{4}n^2$ | $\frac{1}{4}n^2$ | $6.86n^{2.808} - 8n^2$ |
| | [R-Boy+09, Alg. 7.2] | Constant | N/A | $\frac{1}{9}n^2$ | $\frac{1}{9}n^2$ | $7.42n^{2.808} - 12n^2$ |

Table 1.1: Complexities of the schedules presented for square matrix multiplication

### 1.2.3 Parallelization

Our proposed parallel algorithm is recursive and splits the largest of either the row dimension of $A$ or the column dimension of $B$, to form two independent tasks. The granularity of the split can be chosen in two different ways: either as a fixed value, or by terminating the recursion whenever the number of tasks created gets larger than the number of computing ressources (e.g. the total number of cores). By choosing the second option, one maximizes the size of the blocks, and therefore the benefit of Strassen-Winorgad's

algorithm, while ensuring a large enough number of tasks for the computing resources. Our experiments confirmed that this option performs better than the first one. When used as a subroutine in a parallel matrix factorization, an estimate of the number of available threads will be passed to generate the most appropriate split. Of course this number can only be guessed, and an upper estimation will guarantee to keep processors active, while limiting the overhead of managing too many tasks.

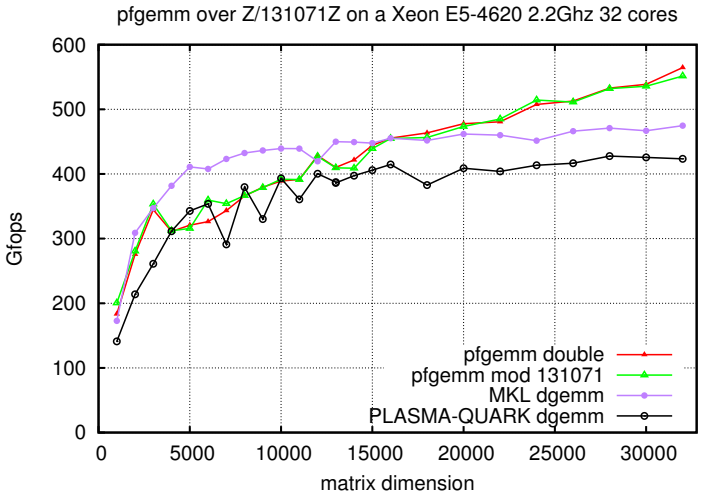



Figure 1.3: Comparison of `FFLAS pfgemm` over `double` and modulo 131071 with the numerical `dgemm` of `PLASMA` and of `Intel-MKL` on a 32 core Xeon E5-4620 @ 2.2Ghz

Figure 1.3 shows the computation speed on 32 cores of various matrix multiplications: the numerical `dgemm` implementation of `Plasma-Quark` and of `Intel-MKL`, and the implementation of `fgemm` of `fflas-ffpack` using OpenMP tasks, linked against the `libkomp` library. This implementation is run over the finite field $\mathbb{Z}/131071\mathbb{Z}$ or over the field of double precision floating point real numbers. For small matrices, `pfgemm` performs similarly as `Plasma-Quark` and is slower than the `MKL` library. With larger dimensions, Strassen-Winograd's algorithm improves the speed of `FFLAS-FFPACK pfgemm` which computes faster than both numerical libraries. The overhead of computing modulo 131071 is not noticeable.

## 1.3 Gaussian elimination

We focus in this section on the algorithmic aspects of exact Gaussian elimination, ignoring the aspects related to rank deficiencies. They will be presented together with recovery of echelon forms and rank profiles in section 1.4. Consequently we will assume here that matrices are generic enough so that each block for which a LU decomposition is computed has full rank.

### 1.3.1 Block algorithm variants

Several schemes are used to design block Gaussian elimination algorithms: the splitting can occur on one dimension only, producing row or column slabs [KG95], or both dimensions, producing tiles [But+09]. Note that, here, we denote by tiles a partition of the matrix into sub-matrices in the mathematical sense regardless what the underlying data storage is. Algorithms processing blocks can be either iterative or recursive.

Figure 1.4 summarizes some of the various existing block splitting obtained by combining these two aspects. Numerous numerical dense Gaussian elimination algorithms, like in [But+09], use tiled iterative block algorithms. In [Don+14] a classic tiled iterative algorithm is combined with a slab recursive one for the panel elimination.

More precisely, tiled iterative algorithms range in three categories: the right-looking, left-looking and the Crout variant [Don+98, §5.4]. They correspond to three ways of scheduling the block operations: the panel LU decomposition and the corresponding updates using triangular system solve, (`trsm`) possibly

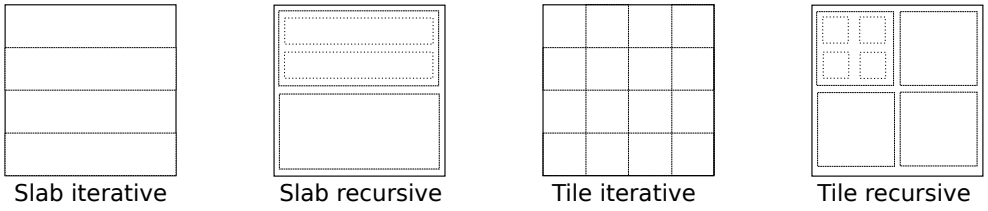



Figure 1.4: Main types of block splitting

with unit diagonal (`utrsm`) and matrix products (`gemm`). We sketch in Table 1.2 these three algorithms, indicating for each routine, the dimension of its arguments: `utrsm` (k,k,n) denotes the solving of an $k \times k$ triangular system, with an $n$ dimensional left or right hand side and `gemm` (m,k,n) denotes a multiplication of an $m \times k$ by an $k \times n$ matrix).

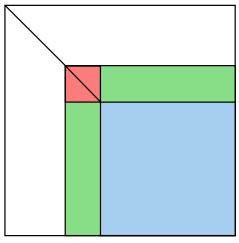
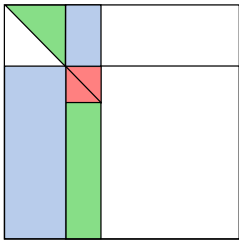
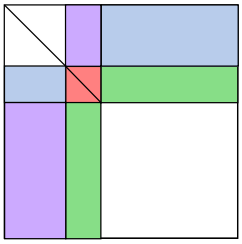
| Right looking | Left looking | Crout |
|---|---|---|
|  |  |  |
| **for** $i = 1 \dots n/k$ **do**<br>`LU` $(k,k)$<br>`utrsm` $(k,k,n-ik)$<br>`trsm` $(k,k,n-ik)$<br>`gemm` $(n-ik,k,n-ik)$<br>**end for** | **for** $i = 1 \dots n/k$ **do**<br>`utrsm` $((i-1)k,(i-1)k,k)$<br>`gemm` $(n-(i-1)k,(i-1)k,k)$<br>`LU` $(k,k)$<br>`trsm` $(k,k,n-ik)$<br>**end for** | **for** $i = 1 \dots n/k$ **do**<br>`gemm` $(n-(i-1)k,(i-1)k,k)$<br>`gemm` $(k,(i-1)k,n-ik)$<br>`LU` $(k,k)$<br>`utrsm` $(k,k,n-ik)$<br>`trsm` $(k,k,n-ik)$<br>**end for** |

Table 1.2: Right looking, Left looking and Crout variants of the tiled iterative block LU factorization, n and k are respectively matrix and block dimensions (see [Don+98, § 5.4])

Over exact domains, recursive algorithms are preferred for the reasons developed in the introduction of this chapter. The first reduction of LU decomposition to matrix multiplication [BH74] uses a slab recursive algorithm, which was then generalized in [IMH82] to handle rank deficiencies with the LSP and LQUP decompositions. We based our early work [R-DGP04; J-DGP08; J-JPS13] on this algorithm. A first tiled recursive algorithm was used in [DR02], for computing the rank using a weaker triangular decomposition. Another one is used in [Mal10] for computing an LEU decomposition which we adapted in [R-DPS13] to define a tiled recursive PLUQ decomposition algorithm (that we will present in more details in section 1.4).

### 1.3.2 Modular reductions

The choice of a block algorithm impacts the overall number of modular reductions that will happen. We compare the number of modular reductions of the above three variants of the tiled iterative algorithms, with the slab and tiled recursive algorithms (see section 1.4 and references therein for further details on the recursive algorithms).

For the sake of simplicity, we will assume that the block dimensions $k$ in the parallel algorithms always allow to fully delay the modular reduction in a matrix product of dimension $k$. For instance, with this model, the number of reductions required by a classic multiplication of matrices of size $m \times k$ by $k \times n$ is simply: $R_{\texttt{gemm}}(m,k,n) = mn$. This extends also for triangular solving with an $m \times n$ right/left hand side: with unit diagonal, $R_{\texttt{utrsm}}(m,m,n) = mn$ (actually the computation of the last row/column of the solution requires no modulo reduction as it is just a division by 1, we will therefore rather use

$R_{\texttt{utrsm}}(m,m,n) = (m-1)n)$ and $R_{\texttt{trsm}}(m,m,n) = (2m-1)n$. The number of modular reductions required for the various LU factorization variants is given in Table 1.3. Refer to [R-Dum+14] for a proof.

| | | |
|---|---|---|
| $k=1$ | Iter. Right | $\frac{1}{3}n^3 - \frac{1}{3}n$ |
| | Iter. Left | $\frac{3}{2}n^2 - \frac{5}{2}n + 1$ |
| | Iter. Crout | $\frac{3}{2}n^2 - \frac{5}{2}n + 1$ |
| $k>1$ | Tiled Iter. Right | $\frac{1}{3k}n^3 + \left(1 - \frac{1}{k}\right)n^2 + \left(\frac{1}{6}k - \frac{3}{2} + \frac{1}{k}\right)n$ |
| | Tiled Iter. Left | $\left(2 - \frac{1}{2k}\right)n^2 - \frac{5}{2}kn + 2k^2 - 2k + 1$ |
| | Tiled Iter. Crout | $\left(\frac{5}{2} - \frac{1}{k}\right)n^2 + \left(-2k - \frac{3}{2} + \frac{1}{k}\right)n + k^2$ |
| | Tiled Recursive | $2n^2 - n\log_2 n - 2n$ |
| | Slab Recursive | $(1 + \frac{1}{4}\log_2 n)n^2 - \frac{1}{2}n\log_2 n - n$ |

Table 1.3: Number of modular reductions in full rank block LU factorization of an $n \times n$ matrix modulo $p$ when $n(p-1)^2 < 2^{\text{mantissa}}$, for a block size of $k$ dividing $n$.

| | | Iterative | | | Recursive | |
|---|---|---|---|---|---|---|
| $n$ | $k$ | Right | Crout | Left | Tiled | Slab |
| 3000 | 212 | 3.02 | 2.10 | **2.05** | 2.16 | 2.26 |
| | $n/3$ | 2.97 | 2.15 | 2.10 | | |
| 5000 | 212 | 11.37 | 8.55 | 8.43 | **7.98** | 8.36 |
| | $n/3$ | 9.24 | 8.35 | 8.21 | | |
| 7000 | 212 | 29.06 | 22.19 | 21.82 | **20.81** | 21.66 |
| | $n/3$ | 22.56 | 22.02 | 21.73 | | |

Table 1.4: Timings (in seconds) of sequential LU factorization variants on one core.

The left looking variant always performs fewer modular reductions. Then the tiled recursive performs better than the Crout variant as soon as $2 \leq k \leq \frac{n}{2+\sqrt{2}}$ and finally, the right looking variant is the most expensive one.

This is confirmed by the running times of our implementation of these algorithms in practice shown on Table 1.4 (except for larger dimensions where the speed-up of fast matrix multiplication comes into play). Also, note that even when the number of modular reductions is an order of magnitude lower than that of the integer operations their cost is nonetheless not negligible. However, this hierarchy no longer holds in the parallel implementation of these algorithms, as we will now see.

### 1.3.3 Experiments in parallel

We compare the performance in practice of these variants on a 32 cores Xeon E5-4620 @ 2.2Ghz. Figures 1.5, 1.6 and 1.7 display computation speeds using `libkomp` implementation of OpenMP tasks.

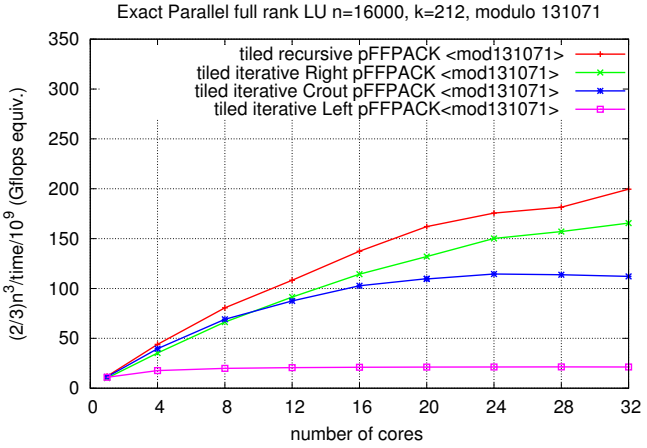



Figure 1.5: Full rank parallel LU factorization modulo 131071, comparing the computation speed of the tiled recursive and left-looking, right-looking, Crout variants of the tiled iterative algorithm.

As shown in Figure 1.5, the tiled recursive algorithm performs better than all other tiled iterative variants. This confirms that recursive algorithms deliver better speed for they group most of the arithmetic operations in large matrix multiplications, they reduce the amount of modular reductions and the tiled version also guarantees a good data locality. Then surprisingly, the left-looking variant performs poorly, seemingly for it uses intensively an expensive sequential `trsm` task of large dimension. Although the Crout

and the right-looking variants perform about the same number of matrix products, but those of the right-looking variant are independent, contrarily to those of the Crout variant, which explains a better scalability despite a larger number of modular reductions.

Figure 1.6 shows the performance without modular reductions of the tiled recursive PLUQ algorithm on full rank matrices compared to state of the art numerical libraries: `Plasma-Quark` and `Intel-MKL`. The two

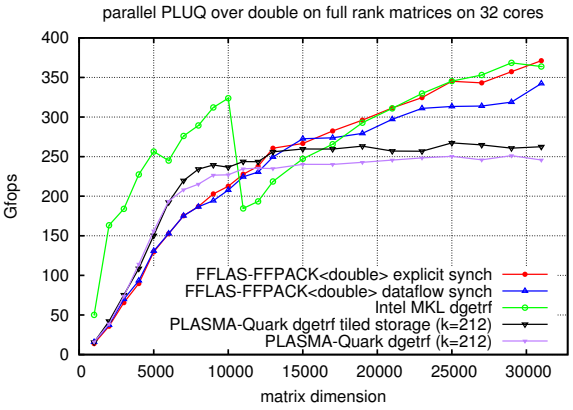



Figure 1.6: Full rank parallel PLUQ factorization over floating points. Comparing the computation speed of the tiled recursive implementation with those of the `dgetrf` routine in `Plasma-Quark` and `Intel-MKL`.

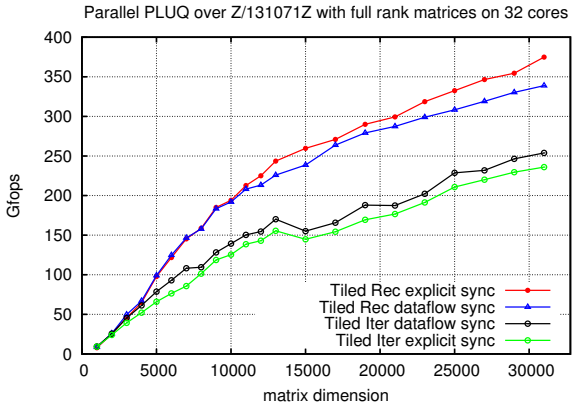



Figure 1.7: Full rank parallel PLUQ factorization modulo 131071. Comparing the computation speed of the tiled recursive and tiled iterative algorithms with explicit or dataflow task synchronization.

possible data-storage for `Plasma-Quark` are considerd: tiles and row-major. In the big picture, our tiled recursive parallel PLUQ elimination performs similarly as the best numerical routines. More precisely, it performs better than `Plasma-Quark` on large enough instances: on small instances, the architecture-aware block algorithms take better advantage of the memory hierarchy, but the higher data locality and the use of Strassen-Winograd's algorithm give the tiled recursive algorithm the advantage on large instances. Surprisingly, `Intel-MKL` performs way better on small instances but suffers from a drop in efficiency around $n = 11000$. Figure 1.7 first shows that the computation speed is maintained over a finite field. Then, the tiled iterative algorithm is compared and proves to perform slower that the recursive algorithm. Lastly, we remark that data-flow task dependencies do improve the computation speed of the iterative algorithm, as it removes unnecessary synchronizations. However, this improvement does not happen in the recursive algorithm, as data-flow dependencies of recursive tasks are badly supported by the existing languages and runtimes and many strong synchronizations remain necessary.

## 1.4 Rank profiles and echelon forms

In exact linear algebra, the rank deficiency of a matrix is not only likely to happen, but is often one of the main information that an elimination has to reveal. For example, the column echelon form of the Macaulay matrix, used in Gröbner basis computations, reveals the ideal structure of the solutions to a polynomial system of equations [Fau99].

This section, based on [J-JPS13; R-DPS13; R-Dum+14], reviews how an elimination algorithm can efficiently deal with rank deficiencies, the conditions at which it reveals rank profiles and related informations, and propose a state of the art algorithm and reductions of all previously existing variants to it.

### 1.4.1 Rank profiles

**Definition 1.4.1**

1. *The row (resp. column) rank profile, denoted by RowRP (resp. ColRP) of an $m \times n$ matrix with rank $r$ is the lexicographically minimal sequence of $r$ indices of linearly independent rows (resp. columns) of this matrix.*

2. *A matrix has generic row (resp. column) rank profile if it is the sequence* $(1, \dots r)$.
3. *A matrix has generic rank profile if its first* $r$ *leading principal minors are non zero.*

Note that a matrix with generic rank profile has also generic row and column rank profiles but the converse is false, as for example with the matrix $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.

We also recall that a matrix is in row echelon form if its zero rows are in the last row positions and if the leading coefficient of each non-zero row is to the right of the leading coefficient of the previous row. A row echelon form is called reduced if the coefficients above each leading coefficient of a row, are all zeros. These definition naturally extend to the column and reduced column echelon forms. Row and Column echelon forms reveal respectively the column and row rank profiles by the position of the pivots on their staircase shape: the column rank profile of a matrix in row echelon form is the column positions of the leading coefficients of its rows.

Laslty recall that for any $m \times n$ matrix $A$,

- there is a non-singular $m \times m$ (resp. $n \times n$) matrix $T$ and an $m \times n$ matrix $E$ in row (resp. column) echelon form such that $TA = E$ (resp. $AT = E$),
- there is a unique non-singular $m \times m$ (resp. $n \times n$) matrix $X$ and a unique $m \times n$ matrix $R$ in reduced row (resp. column) echelon form such that $XA = R$ (resp. $AX = R$),

which gives a natural way to read the row or column rank profiles of a matrix from its associated unique reduced column or row echelon form.

We define an $r$-sub-permutation matrix as an $\{0, 1\}$-matrix of rank $r$ with only $r$ non zero coefficient. Any such matrix of dimensions $m \times n$ can be written in the form $P \begin{bmatrix} I_r & \\ & 0_{(m-r)\times(n-r)} \end{bmatrix} Q$, where $P$ and $Q$ are permutation matrices. And we can now introduce the rank profile matrix.

**Definition 1.4.2**

*For any* $m \times n$ *matrix* $A$ *over a field* $\mathsf{K}$*, there exists a unique* $m \times n$ *rank*$(A)$*-sub-permutation matrix* $\mathcal{R}_A$ *of which all leading sub-matrices have the same row and column rank profiles as the corresponding leading sub-matrices of* $A$*. This sub-permutation matrix is called the* rank profile matrix of $A$.

**Example 1.4.1**

*The matrix* $A = \begin{bmatrix} 2 & 0 & 3 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 2 & 0 & 1 \end{bmatrix}$ *over* $\mathbb{Q}$ *has the rank profile matrix* $\mathcal{R}_A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$.

Such a matrix appears in the LEU decomposition proposed by Malaschonok in [Mal10, Theorem 1] in order to design a permutation free elimination algorithm. There, dimensions are restricted to $m = n = 2^k$, and no connection is made to rank profiles and echelon forms. We then generalized this matrix to arbitrary dimensions in [R-DPS13, Corollary 1], showed how to compute it more efficiently with a new tiled recursive algorithm (Algorihtm 1), and remarked that not only does the matrix $E$ reveal both row and column rank profiles of $A$ (as $\text{RowRP}(A) = \text{RowSupp}(E)$ and $\text{ColRP}(A) = \text{ColSupp}(E)$), but it also reveals the rank profiles of all leading sub-matrices of $A$: $\text{RowRP}(A_{1\dots i,1\dots j}) = \text{RowSupp}(E_{1\dots i,1\dots j})$.

### 1.4.2 Design of Rank profile revealing algorithms

Gaussian elimination of matrices with arbitrary rank profile requires permutations of both rows and columns, hence producing a PLUQ decomposition. However such PLUQ decompositions are not unique and not all of them will necessarily reveal rank profiles and echelon forms. We will characterize the conditions for a PLUQ decomposition algorithm to reveal the row or column rank profile or the rank profile matrix.

There are four key types of operations composing a Gaussian elimination algorithm in the processing of the $k$-th pivot:

**Pivot search:** finding an element that can be used as a pivot,

**Pivot permutation:** moving the pivot in diagonal position $(k, k)$ by column and/or row permutations,

**Update:** applying the elimination at position $(i, j)$: $a_{i,j} \leftarrow a_{i,j} - \frac{a_{i,k}a_{k,j}}{a_{k,k}}$,

**Normalization:** dividing the $k$-th row (resp. column) by the pivot.

Choosing how each of these operation is done, and when they are scheduled results in an algorithm. Conversely, any Gaussian elimination algorithm computing a PLUQ decomposition can be viewed as a set of specializations of each of these operations together with a scheduling.

The choice of doing the normalization on rows or columns only determines which of $U$ or $L$ will be unit triangular. The scheduling of the updates vary depending on the type algorithm used: iterative, recursive, slab or tiled, with right-looking, left-looking or Crout variants, as we saw in section 1.3. None of these two parameters impacts the capacity to reveal rank profiles and we will now focus on the pivot search and the permutations.

Choosing a search and a permutation strategies fixes the permutation matrices $P$ and $Q$ of the PLUQ decoposition obtained. Once these matrices have been fixed, the $L$ and the $U$ factors are uniquely determined.

**Pivot search.** We introduce the pivoting matrix.

**Definition 1.4.3**

*The pivoting matrix of a PLUQ decomposition $A = PLUQ$ of rank $r$ is the $r$-sub-permutation matrix*

$$\Pi_{P,Q} = P \begin{bmatrix} I_r & \\ & 0_{(m-r)\times(n-r)} \end{bmatrix} Q.$$

The $r$ non-zero elements of $\Pi_{P,Q}$ are located at the initial position of the pivots in the matrix $A$ and this matrix therefore summarizes the choices made in the Search operation. This Search operation vastly differs depending on the field of application. In numerical dense linear algebra [Gol96, §3.4], numerical stability is the main criterion for the selection of the pivot. In sparse linear algebra, the pivot is chosen so as reduce the fill-in produced by the Update operation.

In order to reveal some information on the rank profiles, a notion of precedence has to be used: a usual way to compute the row rank profile is to search a given row for a pivot and only move to the next row if it was found to be all zeros. This guarantees that each pivot will be on the first linearly independent row, and therefore the row support of $\Pi_{P,Q}$ will be the row rank profile. The precedence here is that the pivot's coordinates must minimize the order for the first coordinate (the row index). As a generalization, we consider all other preorders of the set $\{1, \dots m\} \times \{1, \dots n\}$, and express in algorithmic terms the fact that a pivot must minimize these preorders for the Search operation:

**Row order:** $(i_1, j_1) \leq (i_2, j_2)$ iff $i_1 \leq i_2$. The corresponding search strategy is to look for any invertible element in the first non zero row.

**Column order:** $(i_1, j_1) \leq (i_2, j_2)$ iff $j_1 \leq j_2$. The corresponding search strategy is to look for any invertible element in the first non zero column.

**Lexicographic order:** $(i_1, j_1) \leq (i_2, j_2)$ iff $i_1 < i_2$ or $i_1 = i_2$ and $j_1 \leq j_2$. The pivot with minimal coordinate, is the leftmost non-zero coefficient of the first non zero row.

**Reverse lexicographic order:** $(i_1, j_1) \leq (i_2, j_2)$ iff $j_1 < j_2$ or $j_1 = j_2$ and $i_1 \leq i_2$. The pivot with minimal coordinate, is the topmost non-zero coefficient of the first non zero column.

**Product order:** $(i_1, j_1) \leq (i_2, j_2)$ iff $i_1 \leq i_2$ and $j_1 \leq j_2$. This is not a total order, hence several minimal elements may exist. The search can be done by inspecting a leading sub-matrix gradually increasing in size. While this sub-matrix is zero, one increases either its row or its column dimension by one. One searches for the non-zero element of smallest row coordinate on the trailing column or smallest column coordinate on the trailing row.

**Pivot permutation.** The pivot permutation moves a pivot from its initial position to the leading diagonal. Besides this constraint all possible choices are left for the remaining values of the permutation. Most often, it is done by row or column transpositions, as it clearly involves fewer data movement. However, these

transpositions can break the precedence relation in the set of rows or columns, and therefore can make it impossible to compute a rank profile. A pivot permutation that leaves the precedence relations unchanged will be called $k$-monotonically increasing.

**Definition 1.4.4**

*A permutation of $\sigma \in \mathcal{S}_n$ is called $k$-monotonically increasing if its last $n-k$ values form a monotonically increasing sequence.*

In particular, the last $n-k$ rows of the associated row-permutation matrix (acting on rows by left multiplication) are in row echelon form. For example, the cyclic shift between indices $k$ and $i$, $(1,\ldots,k-1,i,k,k+1,\ldots,i-1,i+1,\ldots,n)$, that we will call a $(k,i)$-rotation is an elementary $k$-monotonically increasing permutation. Monotonically increasing permutations can be composed as in Lemma 1.4.1.

**Lemma 1.4.1**

*If $\sigma_1 \in \mathcal{S}_n$ is a $k_1$-monotonically increasing permutation and $\sigma_2 \in \mathcal{S}_{k_1} \oplus \mathcal{S}_{n-k_1}$ a $k_2$-monotonically increasing permutation with $k_1 < k_2$ then the permutation $\sigma_2 \circ \sigma_1$ is a $k_2$-monotonically increasing permutation.*

Therefore, an iterative algorithm using rotations as elementary pivot permutation maintains the property that the permutation matrices $P$ and $Q$ at any step $k$ is $k$-monotonically increasing. The same holds with recursive algorithms too.

We can now describe which conditions on the pivot search and permutation make the permutations $P$ and $Q$ reveal information on the rank profiles of the matrix, or more precisely, when does the pivoting matrix $\Pi_{P,Q}$ equal the rank profile matrix $\mathcal{R}_A$, or when do they simply share the same row (resp. column) support, which is the row (resp. column) rank profile. Table 1.5 summarizes these results, and points to instances known in the litterature, implementing the corresponding type of elimination. Further details and a technical proof are available in an article in preparation [T-DPS14].

| **Search** | **Row Perm.** | **Col. Perm.** | **Reveals** | **Instance** |
|---|---|---|---|---|
| Row order | Transposition | Transposition | RowRP | [IMH82; J-JPS13] |
| Col. order | Transposition | Transposition | ColRP | [Kel85; J-JPS13] |
| Lexicographic | Transposition | Transposition | RowRP | [T-DPS14] |
| Lexicographic | Transposition | Rotation | RowRP, ColRP, $\mathcal{R}$ | [T-DPS14] |
| Rev. lexico. | Transposition | Transposition | ColRP | [Sto00] |
| Rev. lexico. | Rotation | Transposition | RowRP, ColRP, $\mathcal{R}$ | [T-DPS14] |
| Product | Rotation | Transposition | RowRP | [T-DPS14] |
| Product | Transposition | Rotation | ColRP | [T-DPS14] |
| Product | Rotation | Rotation | RowRP, ColRP, $\mathcal{R}$ | [R-DPS13] |

Table 1.5: Pivoting Strategies and rank profiles

In particular, algorithms based on partial pivoting, including slab iterative and slab recursive variants [IMH82; J-JPS13] correspond to a pivot search minimizing the row or the column order. The slab recursive Algorithms of [Sto00] also impose to pick the first non-zero coefficient of the current row or column, and therefore minimize the lexicographic or reverse-lexicographic order. Now algorithm 1 presents the tiled recursive algorithm that we proposed in [R-DPS13], implementing a pivot search based on product order, and a pivot permutation based on $k$-monotonically increasing permutations: remarking that matrices $S$ and $T$ are $(r_1+r_2+r_3+r_4)$-monotonically increasing. Hence this algorithm reveals the rank profile matrix: $\Pi_{P,Q} = \mathcal{R}_A$.

**Corollary 1.4.1**

*Algorithm 1 computes the row and column rank profiles of $A$, and of any leading sub-matrix $A_{1\ldots i,1\ldots j}$ of $A$.*

This result stated in [R-DPS13] is to our knowledge the first reduction to matrix multiplication of the problem of computing the rank profiles of all leading sub-matrices of an input matrix. We will see in

**Algorithm 1** Tiled recursive PLUQ decomposition algorithm revealing the rank profile matrix $\mathcal{R}_A$

**Require:** $A = (a_{ij})$ a $m \times n$ matrix over a field
**Ensure:** $P, Q$: $m \times m$ and $n \times n$ permut. matrices
**Ensure:** $r$: the rank of $A$

**if** m=1 **then**
  **if** $A = [0 \ \dots \ 0]$ **then**
    $P \leftarrow [1], Q \leftarrow I_n, r \leftarrow 0$
  **else**
    $i \leftarrow \min\{i : A_{1,i} \neq 0\}$
    $P \leftarrow [1]; Q \leftarrow R_{1,i}, r \leftarrow 1$
    $A \leftarrow AQ^T$
  **end if**
  **return** $(P, Q, r, A)$
**end if**
**if** n=1 **then**
  **if** $A = [0 \ \dots \ 0]^T$ **then**
    $P \leftarrow I_m; Q \leftarrow [1], r \leftarrow 0$
  **else**
    $i \leftarrow \min\{i : A_{i,1} \neq 0\}$
    $Q \leftarrow [1], P \leftarrow R_{1,i}^T, r \leftarrow 1$
    $A \leftarrow P^T A$
    **for** $j = i+1 : m$ **do** $a_{j,1} \leftarrow a_{j,1} a_{1,1}^{-1}$
    **end for**
  **end if**
  **return** $(P, Q, r, A)$
**end if**
Split $A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}$ where $A_1$ is $\lfloor \frac{m}{2} \rfloor \times \lfloor \frac{n}{2} \rfloor$.
Factor $A_1 = P_1 \begin{bmatrix} L_1 \\ M_1 \end{bmatrix} [U_1 \ \ V_1] \, Q_1$ ▷ $\texttt{PLUQ}(A_1)$
$\begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \leftarrow P_1^T A_2$ ▷ $\texttt{PermR}(A_2, P_1^T)$
$[C_1 \ \ C_2] \leftarrow A_3 Q_1^T$ ▷ $\texttt{PermC}(A_3, Q_1^T)$
Here $A = \left[\begin{array}{cc|c} L_1\backslash U_1 & V_1 & B_1 \\ M_1 & 0 & B_2 \\ \hline C_1 & C_2 & A_4 \end{array}\right]$.
$D \leftarrow L_1^{-1} B_1$ ▷ $\texttt{trsm}(L_1, B_1)$
$E \leftarrow C_1 U_1^{-1}$ ▷ $\texttt{trsm}(C_1, U_1)$
$F \leftarrow B_2 - M_1 D$ ▷ $\mathsf{MM}(B_2, M_1, D)$
$G \leftarrow C_2 - E V_1$ ▷ $\mathsf{MM}(C_2, E, V_1)$
$H \leftarrow A_4 - ED$ ▷ $\mathsf{MM}(A_4, E, D)$
Here $A = \left[\begin{array}{cc|c} L_1\backslash U_1 & V_1 & D \\ M_1 & 0 & F \\ \hline E & G & H \end{array}\right]$.
Decompose $F = P_2 \begin{bmatrix} L_2 \\ M_2 \end{bmatrix} [U_2 \, V_2] \, Q_2$ ▷ $\texttt{PLUQ}(F)$
Decompose $G = P_3 \begin{bmatrix} L_3 \\ M_3 \end{bmatrix} [U_3 \, V_3] \, Q_3$ ▷ $\texttt{PLUQ}(G)$
$\begin{bmatrix} H_1 \, H_2 \\ H_3 \, H_4 \end{bmatrix} \leftarrow P_3^T H Q_2^T$ ▷ $\texttt{PermR}(H, P_3^T)$; $\texttt{PermC}(H, Q_2^T)$
$\begin{bmatrix} E_1 \\ E_2 \end{bmatrix} \leftarrow P_3^T E$ ▷ $\texttt{PermR}(E, P_3^T)$
$\begin{bmatrix} M_{11} \\ M_{12} \end{bmatrix} \leftarrow P_2^T M_1$ ▷ $\texttt{PermR}(M_1, P_2^T)$
$[D_1 \, D_2] \leftarrow D Q_2^T$ ▷ $\texttt{PermR}(D, Q_2^T)$
$[V_{11} \, V_{12}] \leftarrow V_1 Q_3^T$ ▷ $\texttt{PermR}(V_1, Q_3^T)$
Here $A = \left[\begin{array}{ccc|cc} L_1\backslash U_1 & V_{11} & V_{12} & D_1 & D_2 \\ M_{11} & 0 & 0 & L_2\backslash U_2 & V_2 \\ M_{12} & 0 & 0 & M_2 & 0 \\ \hline E_1 & L_3\backslash U_3 & V_3 & H_1 & H_2 \\ E_2 & M_3 & 0 & H_3 & H_4 \end{array}\right]$.
$I \leftarrow H_1 U_2^{-1}$ ▷ $\texttt{trsm}(H_1, U_2)$
$J \leftarrow L_3^{-1} I$ ▷ $\texttt{trsm}(L_3, I)$
$K \leftarrow H_3 U_2^{-1}$ ▷ $\texttt{trsm}(H_3, U_2)$
$N \leftarrow L_3^{-1} H_2$ ▷ $\texttt{trsm}(L_3, H_2)$
$O \leftarrow N - JV_2$ ▷ $\mathsf{MM}(N, J, V_2)$
$R \leftarrow H_4 - KV_2 - M_3 O$ ▷ $\mathsf{MM}(H_4, K, V_2)$; $\mathsf{MM}(H_4, M_3, O)$
Decompose $R = P_4 \begin{bmatrix} L_4 \\ M_4 \end{bmatrix} [U_4 \, V_4] \, Q_4$ ▷ $\texttt{PLUQ}(R)$
$\begin{bmatrix} E_{21} \, M_{31} \, 0 \, K_1 \\ E_{22} \, M_{32} \, 0 \, K_2 \end{bmatrix} \leftarrow P_4^T [E_2 \, M_3 \, 0 \, K]$ ▷ $\texttt{PermR}$
$\begin{bmatrix} D_{21} & D_{22} \\ V_{21} & V_{22} \\ 0 & 0 \\ O_1 & O_2 \end{bmatrix} \leftarrow \begin{bmatrix} D_2 \\ V_2 \\ 0 \\ O \end{bmatrix} Q_4^T$ ▷ $\texttt{PermC}$
Here $A = \left[\begin{array}{ccc|ccc} L_1\backslash U_1 & V_{11} & V_{12} & D_1 & D_{21} & D_{22} \\ M_{11} & 0 & 0 & L_2\backslash U_2 & V_{21} & V_{22} \\ M_{12} & 0 & 0 & M_2 & 0 & 0 \\ \hline E_1 & L_3\backslash U_3 & V_3 & I & O_1 & O_2 \\ E_{21} & M_{31} & 0 & K_1 & L_4\backslash U_4 & V_4 \\ E_{22} & M_{32} & 0 & K_2 & M_4 & 0 \end{array}\right]$
$S \leftarrow \begin{bmatrix} I_{r_1+r_2} & & & \\ & & I_{k-r_1-r_2} & \\ & I_{r_3+r_4} & & \\ & & & I_{m-k-r_3-r_4} \end{bmatrix}$
$T = \begin{bmatrix} I_{r_1} & & & & \\ & & I_{r_2} & & \\ & I_{r_3} & & & \\ & & & I_{r_4} & \\ & & I_{k-r_1-r_3} & & \\ & & & & I_{n-k-r_2-r_4} \end{bmatrix}$
$P \leftarrow \mathrm{Diag}(P_1 \begin{bmatrix} I_{r_1} & \\ & P_2 \end{bmatrix}, P_3 \begin{bmatrix} I_{r_3} & \\ & P_4 \end{bmatrix}) S$
$Q \leftarrow T \mathrm{Diag}(\begin{bmatrix} I_{r_1} & \\ & Q_3 \end{bmatrix} Q_1, \begin{bmatrix} I_{r_2} & \\ & Q_4 \end{bmatrix} Q_2)$
$A \leftarrow S^T A T^T$ ▷ $\texttt{PermR}(A, S^T)$; $\texttt{PermC}(A, T^T)$
Here $A = \left[\begin{array}{cccccc} L_1\backslash U_1 & D_1 & V_{11} & D_{21} & V_{12} & D_{22} \\ M_{11} & L_2\backslash U_2 & 0 & V_{21} & 0 & V_{22} \\ E_1 & I & L_3\backslash U_3 & O_1 & V_3 & O_2 \\ E_{21} & K_1 & M_{31} & L_4\backslash U_4 & 0 & V_4 \\ M_{12} & M_2 & 0 & 0 & 0 & 0 \\ E_{22} & K_2 & M_{32} & M_4 & 0 & 0 \end{array}\right]$
**return** $(P, Q, r_1 + r_2 + r_3 + r_4, A)$

section 1.4.3 that this algorithm runs with no arithmetic overhead compared to the standard Gaussian elimination.

**Case of generic row or column rank profile.** When the matrix $A$ has generic row rank profile, it has a LUP decomposition, or equivalently, it has a PLUQ decomposition with $P = I_m$. For any elimination algorithm that uses a pivot search minimizing the lexicographic order, and uses rotations for column pivot permutations, we deduce from Table 1.5 that $\mathcal{R}_A = \Pi_{P,Q} = \begin{bmatrix} I_r & \\ & 0 \end{bmatrix} Q$.

**Corollary 1.4.2**

*If an $m \times n$ matrix $A$ has generic column rank profile, then any PLU decomposition $A = PLU$ computed using reverse lexicographic order based search and rotation based row permutation is such that $\mathcal{R}_A = P \begin{bmatrix} I_r & \\ & 0 \end{bmatrix}$. In particular, $P = \mathcal{R}_A$ if $r = m$.*

*If an $m \times n$ matrix $A$ has generic row rank profile, then any LUP decomposition $A = LUP$ computed using lexicographic order based search and rotation based column permutation is such that $\mathcal{R}_A = \begin{bmatrix} I_r & \\ & 0 \end{bmatrix} P$. In particular, $P = \mathcal{R}_A$ if $r = n$.*

Lastly, remark that the only situation where the rank profile matrix $\mathcal{R}_A$ can be read directly as a sub-matrix of one the permutations $P$ or $Q$ is as in corollary 1.4.2, when the matrix $A$ has generic row or column rank profile. Consider a PLUQ decomposition $A = PLUQ$ revealing the rank profile matrix ($\mathcal{R}_A = P \begin{bmatrix} I_r & \\ & 0 \end{bmatrix} Q$) such that $\mathcal{R}_A$ is a sub-matrix of $P$. This means that $P = \mathcal{R}_A + S$ where $S$ has disjoint row and column support with $\mathcal{R}_A$. We have $\mathcal{R}_A = (\mathcal{R}_A + S) \begin{bmatrix} I_r & \\ & 0 \end{bmatrix} Q = (\mathcal{R}_A + S) \begin{bmatrix} Q_1 \\ 0_{(n-r)\times n} \end{bmatrix}$. Hence $\mathcal{R}_A(I_n - \begin{bmatrix} Q_1 \\ 0_{(n-r)\times n} \end{bmatrix}) = S \begin{bmatrix} Q_1 \\ 0_{(n-r)\times n} \end{bmatrix}$ but the row support of these matrices are disjoint and $\mathcal{R}_A \begin{bmatrix} 0 \\ I_{n-r} \end{bmatrix} = 0$ which implies that $A$ has generic column rank profile. Similarly, one shows that $\mathcal{R}_A$ can be a sub-matrix of $Q$ only if $A$ has a generic row rank profile.

### 1.4.3 Rank deficient LU decomposition algorithm bestiary and reductions

The generalization of the LU decomposition to rank deficient matrices led to numerous kinds of matrix decomposition. We briefly recall the most important ones, which can be computed with an $O(n^\omega)$ algorithm. Some complexities are $O(mnr^{\omega-2})$ which we will call is rank sensitive.

In this process, we mention the ability for a decomposition to be stored in a compact storage, that is: a lower triangular matrix $L$ and an upper triangular matrix $U$ can be stored one next to the other on the memory storage of the input matrix. Will we then denote it by $[L\backslash U]$. The computation will be called in-place, if it does not involve more than $O(1)$ extra-memory allocations, except possibly in the course of the matrix-multiplication calls.

**The LSP decomposition,** introduced in [IMH82] replaces the U factor of an LUP decomposition by a semi-upper triangular matrix $S$, that is upper triangular once its zero rows are removed. The positions of the non-zero rows of $S$ indicate the row rank profile of the input matrix. This decomposition can not use a compact storage nor can be computed in place.

**The LQUP decomposition,** also introduced in [IMH82], replaces the matrix $S$ by the product of a row permutation matrix $Q$ and an upper triangular matrix $U$. This decomposition can use a compact storage, but its computation can not be made in place (see [J-DGP08]).

**The CUP decomposition** that we introduced in [J-JPS13] replaces $L$ by $C = LQ$ which is in column echelon form. This decomposition allows compact storage and in-place rank-sensitive computation.

**The PLE decomposition** is the transposed of the CUP decomposition, with $E$, a row echelon form.

**The StepForm elimination,** introduced by Keller-Gehrig in [Kel85] and attributed to Schönage, (see [BCS97, § 16.5] and [J-JPS13, App. B] for a further description) computes a row echelon form. Its time complexity is not rank sensitive, and the computation is not in-place.

**The Gauss algorithm,** proposed in [Sto00, Alg. 2.8], computes directly the transformation to echelon form: $XA = E$. It allows compact storage and in-place rank-sensitive computation.

**The GaussJordan algorithm,** proposed in [Sto00, Alg. 2.9], computes directly the transformation to reduced echelon form. The algorithm is free of triangular matrices, which is an advantage for computation efficiency. Although a compact storage of the reduced echelon form and the transformation matrix is possible (see [J-JPS13]), the computation can not be done in-place and is rank-sensitive.

**The LEU decomposition,** proposed in [Mal10] manages to gather all permutation and rank deficiency informations in the single $r$-sub-permutation matrix $E$ at the center of the factorization. However Malaschonok's tiled recursive algorithm does not permit neither compact storage, nor in-place computation, and suffers from a large constant in the leading term of the time complexity which is not rank sensitive.

**The PLUQ decomposition** of Algorithm 1, that we proposed in [R-DPS13] makes the connection between the LEU decomposition and the standard PLUQ decompositions. In the process, algorithm 1 gains a rank sensitive time complexity, a leading constant matching the best known constant, a compact storage and an in-place computation.

Table 1.6 summarizes our comparison of these variants. Assuming that matrix multiplication can be done in $C_\omega n^\omega + o(n^\omega)$ for any admissible exponent $\omega > 2$, we state the leading constant $K_\omega$ for each algorithm and give its value for $\omega = 3$ and $\omega = \log_2 7$. We also provide the same informations for classic computations depending on which decomposition they rely as a building block: computing the determinant, the rank, the rank profiles, a transformation to echelon form $XA = E$ or reduced echelon form $XA = R$, the matrix inverse and the test for singularity.

| Algorithm | | Operation | Constant $K_\omega$ | $K_3$ | $K_{\log_2 7}$ | in-place |
|---|---|---|---|---|---|---|
| Matrix Mult. | | $C \leftarrow AB$ | $C_\omega$ | 2 | 6 | × |
| CUP [J-JPS13] | (slab) | $A \leftarrow \begin{bmatrix} C \\ U \end{bmatrix}$ | $\left(\frac{1}{2^{\omega-1}-2} - \frac{1}{2^\omega-2}\right) C_\omega$ | $\frac{2}{3}$ | $2+\frac{4}{5}$ | ∨ |
| PLUQ [R-DPS13],Alg. 1 | (tiled) | $A \leftarrow \begin{bmatrix} L \\ U \end{bmatrix}$ | $\left(\frac{1}{2^{\omega-1}-2} - \frac{1}{2^\omega-2}\right) C_\omega$ | $\frac{2}{3}$ | $2+\frac{4}{5}$ | ∨ |
| Gauss [Sto00] | (slab) | $A \leftarrow \begin{bmatrix} C \\ U^{-1} \end{bmatrix}$ | $\left(\frac{1}{2^{\omega-2}-1} - \frac{3}{2^\omega-2}\right) C_\omega$ | 1 | $4+\frac{2}{5}$ | ∨ |
| GaussJordan [Sto00] | (slab) | $A \leftarrow \begin{bmatrix} R \\ T \end{bmatrix}$ | $\frac{1}{2^{\omega-2}-1} C_\omega$ | 2 | 8 | × |
| StepForm [Kel85] | (slab) | $A \leftarrow \begin{bmatrix} C \\ U^{-1} \end{bmatrix}$ | $\left(\frac{5}{2^{\omega-1}-1} + \frac{1}{(2^{\omega-1}-1)(2^{\omega-2}-1)}\right) C_\omega$ | 4 | $15+\frac{1}{5}$ | × |
| LEU [Mal10] | (tiled) | $A \leftarrow \begin{bmatrix} L \\ U \end{bmatrix}$ | $\frac{17}{2^\omega-4} C_\omega$ | $8+\frac{1}{2}$ | 34 | × |
| Solution | | Using Algorithm | | | | |
| Rank, RankProfile IsSingular, Det, Solve | | Gauss | $\left(\frac{1}{2^{\omega-2}-1} - \frac{3}{2^\omega-2}\right) C_\omega$ | 1 | $4+\frac{2}{5}$ | ∨ |
| | | GaussJordan | $\frac{C_\omega}{2^{\omega-2}-1}$ | 2 | 8 | × |
| | | CUP | $\left(\frac{1}{2^{\omega-1}-2} - \frac{1}{2^\omega-2}\right) C_\omega$ | $\frac{2}{3}$ | $2+\frac{4}{5}$ | ∨ |
| EchTransf | | Gauss,CUP, PLUQ | $\left(\frac{1}{2^{\omega-2}-1} - \frac{3}{2^\omega-2}\right) C_\omega$ | 1 | $4+\frac{2}{5}$ | ∨ |
| Inverse, RedEchTransf | | GaussJordan | $\frac{1}{2^{\omega-2}-1} C_\omega$ | 2 | 8 | × |
| | | Gauss, CUP, PLUQ | $\frac{(2^{\omega-1}+2)}{(2^{\omega-1}-2)(2^{\omega-1}-1)} C_\omega$ | 2 | $8+\frac{4}{5}$ | ∨ |

Table 1.6: Constants of the leading term $K_\omega n^\omega$ in the algebraic complexities for $n \times n$ invertible matrices.

In short, this shows that the slab CUP and PLE algorithms and the tiled PLUQ algorithm are the best building blocks for Gaussian elimination, as

- their time complexity is rank sensitive: $O(mnr^{\omega-2})$,
- they attain the best time complexity leading constant,
- they offer compact storage and allow in-place computation,
- the reductions of the main other problems to them yield almost always the best leading constant in the time complexity, except for the reduced echelon form with an overhead of $1/10$ for $\omega = \log_2 7$.

Finally, the tiled structure of the PLUQ algorithm 1, delivers better data locality in the memory accesses, as shown by experiments in [R-DPS13]. Moreover, it even offers an additional level of parallelization (the two off-diagonal recursive calls being independent), and therefore performs even better in parallel, as we will see in the next section.

### 1.4.4 Parallelization

We recall the main variants that we have investigated in the design of a parallel Gaussian elimination algorithm dealing efficiently with rank deficiencies.

**The Slab iterative algorithm,** shown in Figure 1.8, consists in cutting one dimension in slabs. Then elimination of a slab (called panel factorization) is done by a sequential algorithm. As this task is costly, it imposes a fine granularity, which, as we saw, on the other hand implies more modular reductions and a lesser speed-up of Strassen-Winograd's algorithm. Another difficulty is that the

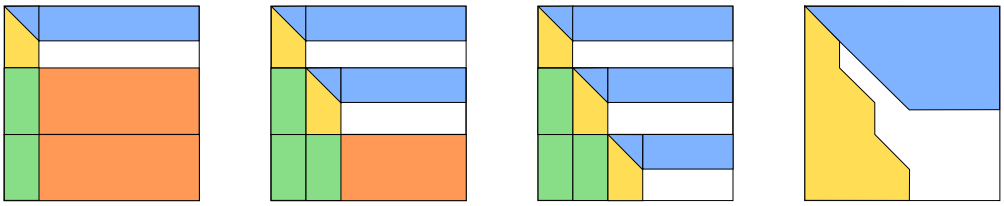


Figure 1.8: Slab iterative factorization of a matrix with rank deficiencies, with final reconstruction of the upper triangular factor

starting column position of each panel is determined by the rank of the blocks computed so far. It can only be determined dynamically upon the execution. In particular, this is why such algorithms are not suited for data-storage by static partitioning in tiles. The workload of each block operation may strongly vary, depending on the rank of the corresponding slab. Such heterogeneous tasks lead us to opt for parallel runtimes with dynamic scheduling and work-stealing instead of static thread management.

**The tiled iterative algorithm:** ideally, it consists in cutting the two dimensions of the matrix in $k$ parts, but in the presence of rank deficiencies, this implies that potentially an elimination will happen in each of the $k^2$ tiles generating a quadratic number of matrix multiplication updates, summing up to a quartic number of tasks that need to be created statically. Of course, many of them will be empty but the overhead in task creation is prohibitive. Instead we will call tiled iterative, an adaptation of the above slab iterative algorithm, where the panel factorization is performed in column tiles. Before our better understanding of the sufficient conditions to reveal rank profiles [R-DPS13; T-DPS14], summarized in Table 1.5, it seemed that, during the panel elimination, a pivot search on the whole row of the matrix (equivalent to a partial pivoting) was necessary to recover the row rank profile. By introducing rotations, we can now reduce the search space for a pivot to only a column slab of the panel. This splitting of the panel elimination in column slabs shares similarities with the recursive computation of the panel of [Don+14]. It has two main advantages: memory accesses are more local and updates can be parallelized. Figure 1.9 illustrates how the panel factorization of the row-slab iterative algorithm is done by a column-slab iterative panel factorization.

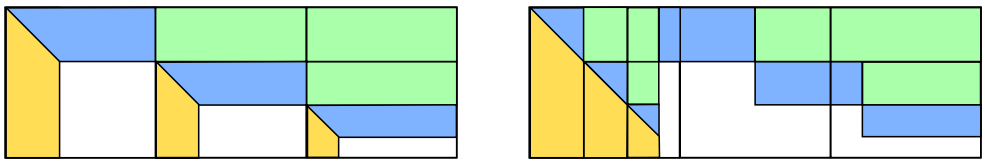


Figure 1.9: Tiled iterative algorithm with rank deficiencies: the panel decomposition step

**Tiled recursive elimination,** is that of Algorithm 1. The parallelization of this algorithm appears in two manners: the calls to `fgemm` and `trsm` and `permc` and `permr` now refer to a task parallel

implementation of these operations, and independent tasks in the algorithm are also forked into concurrent tasks. In particular, the recursive eliminations of the anti-diagonal blocks $F$ and $G$ are done in parallel.

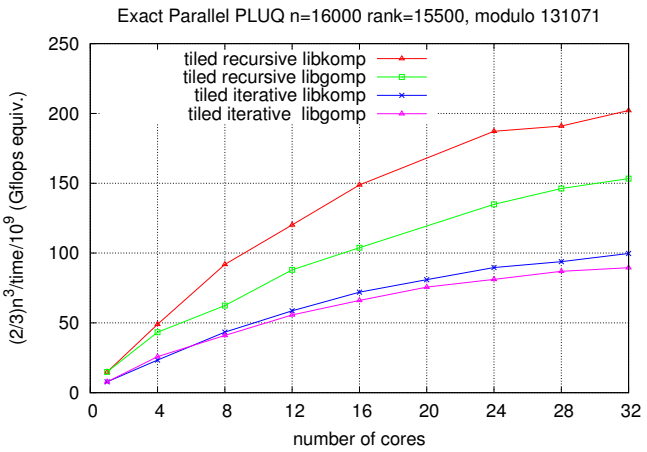



Figure 1.10: Performance of tiled recursive and tiled iterative factorizations using `libgomp` and `libkomp`. Matrix dimension $n = 16000$ with rank 15500

Figure 1.10 shows performance obtained for the tiled recursive and the tiled iterative factorizations. Both versions are tested using the standard GNU implementation of OpenMP, `libgomp`, and the faster one based on `XKaapi`, `libkomp`. The input matrix is a $16000 \times 16000$ matrix with low rank deficiency (rank is 15500). Linearly independent rows and columns of the generated matrix are uniformly distributed. It first shows the superiority of the tiled recursive algorithm over the tiled iterative one. Then, the `XKaapi` runtime proves its better efficiency in handling numerous tasks with the tiled recursive algorithm.

#### Towards dataflow task dependency based scheduling.

In parallel languages using simple task descriptions, such as those of the OpenMP-3.0 standard, the structure of the parallel program directly impacts the position of the task synchronizations which may unnecessarily prevent the concurrent execution of some tasks. For instance, in a tiled or slab iterative algorithm, all matrix multiplication updates of an iteration need to be terminated before the next iteration starts with a sequential panel elimination. If this arbitrary synchronization was removed, this panel elimination could have been concurrent to most of the update tasks, thus reducing the critical path in the execution. The same happens in the tiled or slab recursive algorithms with an additional complication that the actual dependency between tasks may traverse several recursive levels.

The tasks produced in the course of the elimination of a rank deficient matrix have heterogeneous sizes, that depend on the rank of each panel elimination, and can therefore not be predicted. This heterogeneity makes the above arbitrary synchronizations even more harmfull.

Parallel runtimes, scheduling tasks from a dataflow dependency graph, permit to avoid those unnecessary synchronizations and an appropriate related language releases the programmer from having to orchestrate taskx dependencies. This has been a core component in the development of parallel execution runtimes such as `XKaapi` [Gau+12], `Quark` [YKD11], `SMPss` [PBL08], or DAGuE [Bos+12] for distributed computing.

The multiplicity of these languages is an obstacle in the development of a portable library. Hopefully a normalization has recently been adopted in the version 4 of the OpenMP norm, introducing a language for the description of task dependencies. Consequently runtimes such as `XKaapi` now intend to only provide an optimized implementation of the OpenMP norm. Still difficulties persist, in particular as far as recursive dataflow dependencies are concerned, both in the language (postponed modes are not in the OpenMP-4.0 norm) and the algorithmic way to efficiently resolve such dependencies.

The effectiveness of task with dataflow dependencies was confirmed by our first experiments with tiled iterative algorithms. However these aspects, and many more, e.g. the design of a domain specific language

for parallel exact linear algebra, are still work in progress and will be presented in much more details in the Ph.D. dissertation of our student Ziad Sultan, in preparation.

## 1.5 Characteristic Polynomial

The characteristic polynomial of a matrix $A \in \mathsf{K}^{n\times n}$ will be denoted by $\chi_A = \det(XI_n - A)$, and the minimal polynomial of $A$ by $\mu_A$.

### 1.5.1 Characteristic polynomial of dense matrices over a field

Most algorithms computing the characteristic polynomial of a dense matrix over a field are based on a Krylov iteration technique. Given a vector $v \in \mathsf{K}^n$ and a square matrix $A \in \mathsf{K}^{n\times n}$, the Krylov matrix of order $d$ of $v$ is the matrix $K_A(v,d) = \left[\ v \mid Av \mid \cdots \mid A^{d-1}v\ \right] \in \mathsf{K}^{n\times d}$. If the first linear dependency between the iterates $A^i v$ writes $A^d = \sum_{i=0}^{d-1} \alpha_i A^i v$, we define the minimal polynomial of $A$ and $v$ as $\mu_{A,v} = X^d - \sum_{i=0}^{d-1} \alpha_i X^i$ which satisfies the following divisibility relations: $\mu_{A,v} \mid \mu_A \mid \chi_A$. We recall that the companion matrix of a monic polynomial $f = X^d - a_{d-1}X^{d-1} \cdots - a_0$ is the matrix $C_f = \begin{bmatrix} 0 & & & a_0 \\ 1 & & & a_1 \\ & \ddots & & \vdots \\ & & 1 & a_{d-1} \end{bmatrix} \in \mathsf{K}^{d\times d}$. We also use the label $B_*$ to denote a block which has all entries zero except for possibly entries in the last column. The dimension of a block labelled $B_*$ will be conformal with adjacent blocks.

The relation between Krylov matrices and characteristic polynomials is based on the the identity

$$AK = KC_{\mu_{A,v}}, \tag{1.2}$$

where $K = K_A(v, \deg \mu_{A,v})$ is the non-singular Krylov matrix of maximal order for the vector $v$. If $\deg \mu_{A,v} = n$, then $\mu_{A,v} = \chi_A$ and $K$ is square and non-singular. Hence the last column of $K^{-1}AK = C_{\chi_A}$ displays the coefficients of the characteristic polynomial of $A$. Computing the similarity transformation $K^{-1}AK$ reduces to Gaussian elimination and matrix multiplication and can therefore be done in $O(n^\omega)$, but computing the Krylov matrix, even for a single vector is the bottleneck. One can either compute $n$ matrix-vector products, for a cost of $O(n^3)$ or use the reduction to matrix multiplication by Keller-Gehrig [Kel85]: the identity

$$(1 + Y + Y^2 + \cdots + Y^{2^k-1}) = (1+Y)(1+Y^2)\ldots(1+Y^{2^{k-1}})$$

yields an algorithm to compute $n$ Krylov iterates in $\log_2 n$ repeated squaring of the matrix $A$ and matrix products, which amounts to a $O(n^\omega \log n)$ time complexity.

In the general case, when the orbite of a single vector $\mathrm{Orb}_A(v) = \mathrm{Span}(v, Av, A^2v\ldots)$ is a proper sub-space of $\mathsf{K}^n$, one needs to iterate several vectors. A first approach that we developed in [R-DPW05] is to complete the matrix $K_A(v,d)$ into a basis $\overline{K}$ of $\mathsf{K}^n$ by adding well chosen canonical vectors. The corresponding similarity transformation gives

$$\overline{K}^{-1} A \overline{K} = \begin{bmatrix} C_{\mu_{A,v}} & * \\ 0 & D \end{bmatrix},$$

which characteristic polynomial is $\mu_{A,v} \cdot \chi_D$, and it suffices to recursively apply the same algorithm on $D$ to recover the characteristic polynomial of $A$. Although efficient in practice, the time complexity of this algorithm is no better than $O(n^3)$ in the worst case.

Better algorithms are obtained by iterating several vectors simultaneously. For $V \in \mathsf{K}^{n\times j}$ we denote by $\mathrm{Orb}_A(V)$ the subspace of $\mathsf{K}^n$ spanned by all the column vectors in $\left[V \mid AV \mid A^2V \mid \ldots\right]$.

**Lemma 1.5.1**

*For a non-singular matrix $U \in \mathsf{K}^{n\times n}$,*

1 $U = \left[\ K_A(v_1,d_1) \mid \cdots \mid K_A(v_m,d_m)\ \right]$ *for some vectors $v_1,\ldots,v_m \in \mathsf{K}^n$ and positive integers*

$d_1, \ldots, d_m$ *if and only if*

$$U^{-1}AU = \begin{bmatrix} C_{f_1} & B_* & \cdots & B_* \\ B_* & C_{f_2} & \cdots & B_* \\ \vdots & \vdots & \ddots & \vdots \\ B_* & B_* & \cdots & C_{f_m} \end{bmatrix} \tag{1.3}$$

*with* $\deg f_i = d_i$, $1 \le i \le m$. *We will call such a matrix a shifted-form.*

2 *For any* $j$, $1 \le j \le m$, *the matrix (1.3) can be written as*

$$\left[\begin{array}{cccc|cccc} C_{f_1} & B_* & \cdots & B_* & B_* & B_* & \cdots & B_* \\ B^* & C_{f_2} & \cdots & B_* & B_* & B_* & \cdots & B_* \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ B_* & B_* & \cdots & C_{f_j} & B_* & B_* & \cdots & B_* \\ \hline & & & & C_{f_{j+1}} & B_* & \cdots & B_* \\ & & & & B_* & C_{f_{j+2}} & \cdots & B_* \\ & & & & \vdots & \vdots & \ddots & \vdots \\ & & & & B_* & B_* & \cdots & C_{f_m} \end{array}\right]$$

*if and only the dimension of* $\mathrm{Orb}_A([\, v_1 \mid \cdots \mid v_j \,])$ *is equal to* $d_1 + \cdots + d_j$.

3 *Consequently, the sequence* $(d_1, d_2 \ldots d_m)$ *is lexicographically maximal such that the matrix* $U = \left[\, K_A(v_1, d_1) \mid \cdots \mid K_A(v_m, d_m) \,\right]$ *is non-singular, if and only if*

$$U^{-1}AU = \begin{bmatrix} C_{f_1} & B_* & \cdots & B_* \\ & C_{f_2} & \cdots & B_* \\ & & \ddots & \vdots \\ & & & C_{f_m} \end{bmatrix}. \tag{1.4}$$

The block upper triangular matrix in equation (1.4) is called a Hessenberg polycyclic form and its characteristic polynomial is the product of the polynomials $f_i$ of each of its diagonal companion blocks.

Keller-Gehrig's branching algorithm [Kel85, § 5] applies the above-mentioned repeated squaring to the $n$ canonical vectors simultaneously, in combination with a column rank profiles computation, to remove the linearly dependent iterates. This rank profile computation is done by the $O(n^\omega)$ Step-form elimination algorithm mentionned in Table 1.6. The algorithm returns a transformation matrix $U$ such that $U^{-1}AU$ is in Hessenberg polycyclic form in $O(n^\omega \log n)$.

This is the best known time complexity for a deterministic computation of the characteristic polynomial, making this problem the last classical one in dense linear algebra that does not reduce to the cost of matrix multiplication. Note that it is necessarily at least as hard as matrix multiplication, from Baur and Strassen's reduction of the determinant to matrix multiplication [BS83].

In order to remove this $\log n$ factor, one should avoid constructing explicitly the Krylov matrix and prefer to incrementally transform the matrix $A$ in shifted forms by successive similarity transformations as in equation (1.3). Keller-Gehrig's fast algorithm [Kel85, § 6] proceeds by iterating a geometrically decreasing number of vectors of the canonical basis and applying the corresponding similarity transformation at each step. Under a very strong genericity assumption (including that the minimal polynomial of the first canonical vector is the characteristic polynomial) this algorithm runs in $O(n^\omega)$. Relaxing the elimination process in this algorithm by introducing permutations yields a generalization that works with generic matrices, as we showed in [Per06, §8.3.5]. Finally, we proposed with Arne Storjohann, the following result:

**Theorem 1.5.1 (*[R-PS07a]*)**

*There exists a randomized Las-Vegas algorithm computing the characteristic polynomial of any* $n \times n$ *matrix over a field* $\mathsf{K}$, *sufficiently large* ($|\mathsf{K}| \ge 2n^2$), *in expected time* $O(n^\omega)$.

The algorithm maintains, at each iteration $k$, a matrix $A^{(k)}$ in shifted form as in equation (1.3), with column slices of equal dimension $k$, except for possibly the last one of dimension $\le k$. We call such a

matrix a $k$-shifted form. A key feature is that the Krylov matrix of order $k+1$ for each of the $m$ canonical vectors $v_i = e_{(i-1)k+1}, 1 \le i \le m$ has the form

$$K_{A^{(k)}}(v_i, k+1) = \begin{bmatrix} 0 & \\ I_k & A^{(k)}_{*,k(i+1)-1} \\ 0 & \\ 0 & \end{bmatrix}.$$

Hence, the matrix $E = \left[ K_{A^{(k)}}(v_1, k+1) \mid \cdots \mid K_{A^{(k)}}(v_m, k+1) \right]$ is composed of all columns of the identity matrix and the non-trivial columns of $A^{(k)}$ and can thus be formed with no arithmetic operation.

Under some genericity assumptions, ensured by a randomized precondionning, the $n$ rank profile columns of $E$ form an invertible matrix that also has the shape $U = \left[ K_{A^{(k)}}(v_1, d_1) \mid \cdots \mid K_{A^{(k)}}(v_m, d_m) \right]$ where $(d_1, \ldots, d_m) \in \{0 \ldots k+1\}^m$ is monotonically non-increasing, and such that

$$U^{-1} A^{(k)} U = \begin{bmatrix} A^{(k+1)} & * \\ 0 & H \end{bmatrix}$$

where $A^{(k+1)}$ is in $k+1$-shifted form and $H$ is in Hessenberg polycyclic form. The algorithm can then iteratively process $A^{(k+1)}$ and produces a Hessenberg polycyclic form. Thanks to row and column permutations, the non-trivial columns of the matrices $A^{(k)}$, $U$ and $U^{-1}$ are compressed into dense $n \times n/k$ matrices so that the rank profile computation and the similarity transformation reduce to matrix multiplications of dimension $n/k$. The complexity is then obtained as a constant multiple of

$$\sum_{k=1}^{n} k \left(\frac{n}{k}\right)^{\omega} \le \zeta(\omega - 1) n^{\omega}.$$

which is $O(n^\omega)$ for $\omega > 2$.

**Remark 1.5.1**

*Following the analogy developed in [Vil97], we remark that a matrix in shifted form as in equation (1.3) can be viewed as an $m \times m$ diagonal dominant matrix with polynomial coefficients: its diagonal elements are the polynomials given by the companion blocks, and the $B_*$ carry the coefficients of polynomials of degree less than the diagonal element of the same block row. Any $n \times n$ matrix can be viewed as a degree 1 polynomial matrix of order $n$: its characteristic matrix; the characteristic polynomial is an order 1 polynomial matrix of degree $n$ and more generally the Hessenberg polycyclic form is an $m \times m$ upper triangular polynomial matrix which determinant is the characteristic polynomial of $A$. In this setting, the above algorithm maintains a degree/dimension compromise: at each iteration $k$, $A^{(k)}$ correspond to a degree $k$ polynomial matrix of order $n/k$. The progress of one step is an elimination done by unimodular transformations, increasing the degree of the first $n/(k+1)$ columns by one, and reducing the degrees in the trailing columns. Such an analogy could be exploited in attempting to derandomize this algorithm.*

### 1.5.2 Frobenius normal form and a transformation matrix

The Frobenius normal form of a matrix $A \in \mathsf{K}^{n \times n}$ is the unique representative of the set of all similar matrices to $A$ that is block diagonal of the form $F = U^{-1}AU = \mathrm{Diag}(C_{f_1}, \ldots, C_{f_\ell})$ where $f_{i+1} | f_i$. The first reduction of the computation of the Frobenius normal form to matrix multiplication was done by Mark Giesbrecht in [Gie93; Gie95] with a Las-Vegas randomized algorithm of expected time $O(n^\omega \log n)$. This algorithm is based on Keller-Gehrig's branching algorithm run on a set of vectors sampled uniformly from a large enough field ($|\mathsf{K}| > n^2$). It produces a Hessenberg polycyclic form $H = V^{-1}AV$ which, with a bounded probability, will share the same diagonal companion blocks as the Frobenius normal form of $A$. Such a matrix is called a quasi-Frobenius form. Lastly, Giesbrecht proposes a $O(n^\omega)$ update on $V$ to compute the transformation matrix $U$ such that $U^{-1}AU = F$ is the Frobenius normal form of $A$. Eberly proposed in [Ebe00] another randomized Las-Vegas algorithm in expected $O(n^\omega \log n)$ time, with no restriction on the size of the field. His algorithm does not use Keller-Gehrig's algorithm but the $\log n$ factor also arises from the computation of the powers of $A$: $A^2, A^4 \ldots, A^{2^{\lceil \log n \rceil}}$.

We remark that the $O(n^\omega)$ algorithm for the characteristic polynomial described in section 1.5.1 computes, when the randomization succeeds, a Hessenberg polycyclic form $H = V^{-1}AV$ in quasi-Frobenius form. This directly gives a Las-Vegas algorithm with expected time $O(n^\omega)$ for the Frobenius normal form.

The computation of the transformation matrix $U$ to the Frobenius form, is more involved. The update techniques of [Gie93] allow to recover $U$ from the transformation matrix $V$ in $O(n^\omega)$. However, the computation of $V$ by Keller-Gehrig's algorithm will again involve $O(n^\omega \log n)$ operations.

Instead, we proposed in [T-PS07b] a faster way to compose the $n$ successive elementary transformations $K_i, 2 \leq i \leq n$ happening in the $O(n^\omega)$ algorithm. Consider the product $K_{k,s} = K_k K_{k+1} \dots K_s$.

**Lemma 1.5.2**

*For $2 \leq k < n$, let $s = \min(\lceil k^{\omega-1} \rceil, n)$. Then $K_{k,s}$ can be computed with $O(n^\omega)$ field operations.*

Indeed each factor in this product has less than $n/k$ plain columns. Thus $K_{k,s}$ and can be computed in $O(n \left(\frac{n}{k}\right)^{\omega-1} s)$. For $s = O(\lceil k^{\omega-1} \rceil)$, this is $O(n^\omega)$. Now the transformation matrix $U = K_{2,n}$ can be computed as follows:

$K \leftarrow I_n$; $k \leftarrow 2$
**while** $k \leq n$ **do**
  $s \leftarrow \min(\lceil k^{\omega-1} \rceil, n)$
  Compute $K_{k,s} = K_k K_{k+1} \cdots K_s$ as in Lemma 1.5.2
  $K \leftarrow K K_{k,s}$
  $k \leftarrow k + s + 1$
**end while**

The value of $k$ increases at least as rapidly as the sequence $2^{\omega-1}$, $2^{(\omega-1)^2}$, $2^{(\omega-1)^3}, \dots$ and thus will exceed $n$ after $O(\log \log n)$ iterations. Since each loop iteration has runtime $O(n^\omega)$, this shows that $K$ may be computed in time $O(n^\omega \log \log n)$. This completes the proof of Theorem 1.5.2.

**Theorem 1.5.2**

*There exists a Las Vegas algorithm that for an $n \times n$ matrix $A$ over a field $\mathsf{K}$ with $|\mathsf{K}| \geq 2n^2$, returns a $U \in \mathsf{K}^{n \times n}$ such that $U^{-1}AU$ is in Frobenius form in expected time $O(n^\omega \log \log n)$ field operations.*

**What have we learned?**

1. *Floating point arithmetic remain de facto the most efficient choice for implementations of linear algebra over a finite of bit-size between 5 and 24.*
2. *Strassen-Winograd's algorithm memory requirements can be reduced by either overwriting some inputs or by trading off the leading constant in the time complexity for lesser space requirements.*
3. *A combination of recursive tasks and coarse grain data parallelism delivers the best efficiency on shared memory parallel architectures.*
4. *The rank profile matrix summarizes the informations on the row and column rank profiles of a matrix and that of all of its leading sub-matrices. It can be computed by any Gaussian elimination algorithm provided some restrictions on the pivot search and permutations.*
5. *Tiled recursive LU decomposition is the best Gaussian elimination building block, from which all other related eliminations can be recovered. We proposed such an algorithm implementing the conditions to recover the rank profile matrix in $O(n^\omega)$ field operations.*
6. *The characteristic polynomial and the Frobenius normal form over a large enough field can be computed by a randomized Las-Vegas algorithm in expected $O(n^\omega)$ field operations. The transformation matrix to the Frobenius normal form can be computed with an extra $\log \log n$ factor.*

Chapter 2

# Coding Theory

**Fault tolerance.** In the recent developments of high performance computing, the reliability [Lap95] of the computations has been given an increasing attention. Errors during a computation may occur due to numerous reasons: software errors, hardware errors, including disk failure, bit-flip in main or cache memory, due to cosmic radiations, etc. Despite many efforts to maintain the failure rate very low (e.g. thanks to ECC RAM using basic error correction), the use of very large scale distributed systems makes the probability of failure of one computing node no longer negligible. Indeed, recent peta-scale supercomputers, such as Bluewaters [Di +14] or Tsubame 2 [Sat+12] encounter approximately two errors per day. Many approaches to handle fail-stop failures rely on check-pointing and rollback [Eln+02] where latency of recovery (approximately of about 30 minutes nowadays) is becoming a bottleneck as it is getting near the predicted mean time between failures of the next generation of exa-scale computers [Cap+09]. Consequently alternative approaches are being developed to handle errors at earlier stages and broaden the range of errors supported. Among them, Algorithm Based Fault Tolerance (ABFT) [HA84] exploits specificities of the algorithm to introduce redundancy and make the algorithm resilient to errors, thanks to error correcting codes.

The development of outsourced distributed computations, including volunteer, peer-to-peer and cloud computing, also raises the question of the trust that one should have on the result of such computations. The model of Byzantine errors [LSP82; Sch84] illustrates the case where the computation is not always corrupted, and therefore, one might still want to use the results provided by a faulty computing node. In the general setting, these errors can be managed, at least heuristically, using error detection and for instance blacklisting of corrupted nodes, or checking post-conditions on the results. When algorithms allow it, Algorithm Based Fault Tolerance can be used, offering a more efficient correction capacity.

We present in this section our contributions in the introduction of Algorithm Based Fault Tolerance in high performance algebraic computing to support in a unified way Byzantine corruptions and soft-errors.

**Error model and parallelization scheme.** In the design of a fault tolerant algorithm, the error model is directly related to the type of parallelization chosen for the algorithm. With $n$ computing nodes, each output of a node is a symbol and a valid output of the $n$ nodes form a code word. We make the assumption that errors only happen locally: one error corresponds to a single faulty node and thus produces a single incorrect symbol in word. An error can be either a hard error (or failure), for which the output is never produced. This corresponds to an erasure in the corresponding code-word. Or it can be a soft error (or silent error), where the output of the node is produced but is incorrect (due to a physical alteration of the memory, a bit-flip, or even an adversary attack).

Hence, the choice of a parallelization scheme induces a choice for the way the information is encoded and the error model of the corresponding channel. For example, the ABFT developed in numerical linear algebra [HA84; LP86; Bol+92; Bos+09; Du+12] rely on a parallelization based on the splitting in one or both dimensions of the problem. As an example, a matrix multiplication $C \leftarrow A \times B$ can be parallelized by splitting the column dimension of $B$ and $C$ and performing the product for each slice on a separate computing node. Fault tolerance is achieved by adding extra columns to $B$ and $C$ acting as parity check redundancy on the remaining columns. In this context a symbol is a matrix coefficient and a code-word a row of the matrix $C$ expanded with parity columns.

**Evaluation-interpolation based parallelization.** A particularity of exact linear algebra is that the size of the problem is not only a function of the matrix dimensions, but also of the size of their coefficients (i.e. the bit size for integers and rationals or the degree for polynomials).

Computing directly over such domains with variable sized coefficients is often inefficient due to the rapid growth in the size of the intermediate computations. Instead evaluation-interpolation transforms a computation with polynomial coefficients into several independent computations over the coefficient domain; the result is obtained by a polynomial interpolation. Similarly, computations over the integers are reduced to several computations modulo coprime numbers and the result is reconstructed by the Chinese remainder theorem.

The time complexity of these approaches is of the same order as the product of the algebraic complexity of the problem (computing over the coefficient domain or over a fixed sized finite field) multiplied by the size of the output. As an example, the determinant of a polynomial matrix of degree $d$ and dimension $n$ is computed using $O(nd)$ evaluations of the matrix, the determinant of each of which is computed in $O(n^\omega)$, for a total cost of $O(n^{\omega+1}d)$. This complexity is at a factor $n$ away from the complexity of polynomial matrix multiplication which is a lower bound for this computation. Better algorithms [Sto03; Sto05] are known, filling this gap and thus reaching the essentially optimal time complexity for this task. However, evaluation-interpolation approaches are still competitive especially for that they naturally split the computation into independent tasks, thus making the algorithm embarrassingly parallel.

**Secure Multiparty computation.** Evaluation-interpolation techniques are also at the core of the famous Shamir secrete sharing protocol [Sha79], allowing to spread a secret between $n$ shares, such that at least $k < n$ of them must agree in order to recover the secret. It has been applied to multiparty computations [BGW88; Gol97], where players taking part to the computation, should not be able to access to the final result without agreeing with enough other players. This framework naturally provides resilience to faults, thanks to the existence of evaluation-interpolation codes.

We will focus in this chapter on various techniques used to make the evaluation-interpolation based parallelization schemes fault tolerant by exploring evaluation-interpolation based error correcting codes.

All the codes that we will present are formed using an evaluation function

$$\begin{array}{rccc} \mathrm{Ev}_{(x_1,\ldots,x_n)}: & \mathsf{D} & \longrightarrow & \mathsf{K}^n \\ & f & \longmapsto & (f(x_1),\ldots,f(x_n)) \end{array}$$

for some distinct points $x_1,\ldots,x_n \in \mathsf{K}$ and where the domains $\mathsf{D}$ and $\mathsf{K}$ are specified for each code. We are interested in recovering the unknown element $f$ given a vector of its possibly erroneous evaluations. We will study this curve fitting problem with outliers in the following specializations: over the ring of polynomials ($\mathsf{D} = \mathsf{K}[X]$) in section 2.2, with sparse polynomials in section 2.6 and over the field of rational functions ($\mathsf{D} = \mathsf{K}(X)$) in section 2.3. Lastly we will extend these results to the ring of integers and the field of rationals ($\mathsf{D} = \mathbb{Z}, \mathbb{Q}$) in section 2.5 and over the set of vectors of polynomials or rational functions in 2.4.

In the process, we will apply a rather systematic approach for their study (except with sparse polynomial evaluation codes in section 2.6): the decoding of the code will be expressed as a key equation which in turn is viewed as a rational approximation problem. More precisely, the solution of the decoding problem is the minimal solution to the rational reconstruction problem expressed in the key equation. Using the fact that the Extended Euclidean algorithm (or some variants of it, like Berlekamp/Massey algorithm ) computes these minimal solutions, it immediately produces a unique decoding algorithm. The case of interleaved codes (section 2.4) yields a system of key equations, which solution is no longer unique but is described by short vectors in a lattice. Lattice reduction algorithms (generalizing the extended Euclidean algorithm to higher dimensions) are then used for their decoding.

The results on Reed-Solomon and CRT codes and their output sensitive decoding have been presented in [R-Kho+10]. The generalization to the rational function and number evaluation codes are from a collaboration with Erich Kaltofen, and will be detailed in an article in preparation. They have been applied in multi-variate and symbolic-numeric setting [KY13; BK14; KY14] where not only errors but also numerical noise is considered. The results on sparse polynomial evaluation codes have been presented in [R-CKP12] and [R-KP14].

## 2.1 Preliminaries

We briefly review the few basic concepts of coding theory that will later be used in this section.

### 2.1.1 Terminology

Error correcting codes are mainly designed to encode information by adding some redundancy allowing to detect or correct one or more errors introduced by a communication channel.

A code is defined as a subset $\mathcal{C}$ of code-words living in a larger set $\mathcal{E}$ of all possible words. When $\mathcal{E}$ is a vector space and $\mathcal{C}$ is a sub-vector space of $\mathcal{E}$, $\mathcal{C}$ is called a linear code. In this case the code dimension is its dimension as a vector space. Encoding is the transformation from an initial message $m$ into a code-word $c$, and decoding is the process of finding a code word $c$ *close* to a received word $r$ and recovering the corresponding message $m$. The definition of *close* is left voluntarily unspecified for the moment, and depend on a metric chosen over $\mathcal{E}$ and the type of decoding chosen. We will focus on bounded distance decoding: finding the code-words at distance less than a given radius of a given a received word.

A commonly used metric is the Hamming distance $d_H$ and the Hamming weight $w_H$ defined by

$$\begin{aligned} w_H((x_1, \dots, x_n)) &= |\{i \in \{1 \dots n\} : x_i \neq 0\}| \\ d_H(x, y) &= w_H(x - y). \end{aligned}$$

The minimum distance of a code is defined as the smallest distance between two code words and is the minimal weight of a non-zero code-word in the case of linear codes:

$$\delta = \min_{w_1, w_2 \in \mathcal{C}} d_H(w_1, w_2) = \min_{w \in \mathcal{C} \setminus \{0\}} w_H(w).$$

The correction capacity $\tau$ of a code is the maximum integer $t$ such that balls of radius $t$ centered around the code-words do not intersect. The code is said to be $\tau$-corrector. This indeed implies that for any received word at distance $t \leq \tau$ to a code-word, there is a unique code-word solution to the bounded distance decoding problem with bound $\tau$. This is called unique decoding. Imposing a larger decoding radius $\tau > \frac{\delta-1}{2}$ implies the loss of uniqueness, and the decoder then need to return a list of code-words at distance $\leq \tau$, this is called list decoding. Noting that balls of radius $\frac{\delta-1}{2}$ centered in the code-words do not intersect, we deduce that unique decoding is only possible up to decoding radius $\tau = \frac{\delta-1}{2}$.

Lastly, the Singleton bound states that the minimal distance of a linear code of length $n$ and dimension $k-1$ verifies

$$\delta \leq n - k. \tag{2.1}$$

Such linear codes where equality $\delta = n - k$ holds are called maximal distance separable (MDS) and maximize the unique decoding radius for a given dimension and length.

### 2.1.2 Linear recurring sequences

We recall that an infinite sequence $a = (a_0, a_1, \dots, a_n, \dots)$ of elements of a field $\mathsf{K}$ is linearly recurring if there exist an integer $t > 0$ and $t$ elements $\lambda_0, \dots, \lambda_{t-1} \in \mathsf{K}$ such that

$$a_{t+j} = \sum_{i=0}^{t-1} \lambda_i a_{i+j} \; \forall j \geq 0.$$

The monic polynomial $\Lambda(X) = X^t - \sum_{j=0}^{t-1} \lambda_j X^j$ is called a connecting or generating polynomial for the sequence. The connecting polynomial of least degree is called the minimal generating polynomial of the sequence and its degree is the linear complexity of the sequence.

These definitions can be extended to vectors, viewed as contiguous sub-sequences of an infinite sequence. The minimal generating polynomial of an $n$-dimensional vector is the monic polynomial $\Lambda(X) = X^t - \sum_{i=0}^{t-1} \lambda_i X^i$ of least degree such that $a_{j+t} = \sum_{i=0}^{t-1} \lambda_i a_{i+j} \; \forall 0 \leq j \leq n - t - 1$. Note that consequently, any vector is linearly recurring with linear complexity less than $n$.

An important connection between linear complexity and sparsity, is made in Theorem 2.1.1, referred to as Blahut's Theorem [Bla83; MS88]) though a similar form was already used in the 18th century by Prony [Pro95].

**Theorem 2.1.1 (*Blahut [Bla83; MS88]*)**

*Let* $\mathsf{K}$ *be a field containing an* $N$*-th primitive root of unity. The linear complexity of an* $N$*-periodic sequence* $A = (a_0, \ldots, a_{N-1}, a_0, \ldots)$ *over* $\mathsf{K}$ *is equal to the Hamming weight of the discrete Fourier transform of* $(a_0, \ldots, a_{N-1})$.

### 2.1.3 Approximation problems

Let $n$ be a positive integer and $\mathsf{K}$ an arbitrary field with at least $n$ elements.We recall a few classic results on rational approximation problems.

#### Rational function reconstruction.

**Problem 2.1.1 (*RatFunRec*)**

*Given two polynomials* $A, B$ *over* $\mathsf{K}$ *with* $\deg B < \deg A = n$ *and an integer* $0 < k < n$*, find a pair of polynomials* $(f, g)$*, with* $g$ *monic,* $\deg f \leq k$, $\deg g \leq n - k - 1$ *such that*

$$f = gB \mod A.$$

**Lemma 2.1.1**

*Any two pairs* $(f_1, g_1)$ *and* $(f_2, g_2)$ *of solutions of a rational function reconstruction problem satisfy* $f_1 g_2 = f_2 g_1$.

**Proof.** $f_1 g_2 = g_1 B g_2 = g_1 f_2 \mod A$. But $\max(\deg f_1 g_2, \deg f_2 g_1) < n$, and therefore $f_1 g_2 = f_2 g_1$. □

Lemma 2.1.1 implies that any solution of a rational function reconstruction problem is a multiple of a solution of minimal degree. In other terms, they are the subset of elements in a free $\mathsf{K}[X]$-module of rank one, satisfying the degree constraints. Lemma 2.1.2, a variation of [GG13, Lemma 5.15] states that the extended Euclidean algorithm solves the rational function reconstruction problem by finding the solution of least degree.

**Lemma 2.1.2**

*Consider the extended Euclidean algorithm run on* $f_0 = A, f_1 = B$*, and computing a sequence of remainders* $f_\ell$ *defined by* $f_{\ell-2} = q_\ell f_{\ell-1} + f_\ell$ *and* $0 \leq \deg f_\ell < \deg f_{\ell-1}$*, together with the multipliers* $g_\ell, h_\ell$ *defined by* $(g_0, h_0) = (0, 1); (g_1, h_1) = (1, 0)$ *and* $g_\ell = g_{\ell-2} - q_\ell g_{\ell-1}; h_\ell = h_{\ell-2} - q_\ell h_{\ell-1} \forall \ell > 1$ *such that* $f_\ell = g_\ell f_1 + h_\ell f_0$ *holds for every* $\ell$.

*Then at iteration* $j$ *such that* $\deg f_j \leq k < \deg f_{j-1}$,

1. *the pair* $(f_j, g_j)$ *is a solution to the rational reconstruction problem RatFunRec* $(A, B, k)$.
2. *this solution is minimal: any other solution* $(f, g)$ *satisfies* $f = q f_j, g = q g_j$ *for some* $q \in \mathsf{K}[X]$.

**Proof.** At iteration $j$, we have $\deg g_j = n - \deg f_{j-1} < n - k$ and $f_j = g_j B \mod A$, thus $(f_j, g_j)$ is a solution to the rational reconstruction problem.

Suppose that $(f, g)$ is another solution such that $f = gB + hA$. Then, from lemma 2.1.1, $f_j g = f g_j$. Now as

$$0 = f_j g - f g_j = (g_j B + h_j A) g - (gB + hA) g_j = (h_j g - h g_j) A.$$

we have $h_j g = h g_j$. Since $\gcd(h_j, g_j) = 1$, we deduce that $h = q h_j$ for some $q \in \mathsf{K}[X]$. Lastly, $h_j g = h_j q g_j$ leads to $g = q g_j$. □

### Padé Approximation.

When $A = X^n$, the problem is called Padé approximation. A $(d_f, d_g)$-Padé approximant of a function $h$ is a pair of polynomials $f, g \in \mathsf{K}[X]$ with $\deg f \leq d_f$ $\deg g \leq d_g$ and $g(0) = 1$ such that

$$f = gh \mod X^{m+n+1}. \tag{2.2}$$

Hence solving the rational function reconstruction problem with $A = X^{d_f+d_g+1}$, $B = h$ and $k = d_f$ computes the $(d_f, d_g)$-Padé approximant of $h$. It can therefore be computed via the extended Euclidean algorithm. An alternative is the Berlekamp/Massey algorithm , computing the minimal generating polynomial of a linearly recurring sequence. Indeed the Padé approximant equation (2.2) writes:

$$g_0 h_i + g_1 h_{i-1} + \cdots + g_{d_g} h_{i-d_g} = f_i \, \forall i \in \{0 \dots d_f + d_g\}$$

(considering $h_{<0} = 0$ for the commodity of notations) which implies

$$h_i = -g_1 h_{i-1} - \cdots - g_{d_g} h_{i-d_g} \forall i \in \{d_f + 1 \dots d_f + d_g\},$$

showing that the terms of the sequence $(h_{d_f+1}, \dots, h_{d_f+d_g})$ are linearly generated by the polynomial $X^{d_g} - \sum_{i=1}^{d_g} g_i X^{d_g - i}$.

The equivalence between the Berlekamp-Massey and the extended Euclidean algorithm has been established in [Dor87] but often independently rediscovered.

### Cauchy interpolation.

Given $n$ distinct elements $x_1, \dots, x_n$ and $n$ elements $y_1, \dots, y_n$, and setting $A = \prod_{i=1}^{n}(X - x_i)$ and $B$ the polynomial interpolant such that $B(x_i) = y_i \, \forall i \in \{1 \dots n\}$, the problem becomes the interpolation of a rational function, also called Cauchy interpolation.

**Problem 2.1.2 (*Cauchy interpolation*)**

*Given* $(x_1, \dots, x_n) \in \mathsf{K}^n$ *and* $(y_1, \dots, y_n) \in \mathsf{K}^n$, *find* $f, g \in \mathsf{K}[X]$, *co-prime, with* $\deg f \leq k$, $\deg g \leq n - k - 1$ *such that*

$$f(x_i) = y_i g(x_i) \, \forall i \in \{1 \dots n\}.$$

Cauchy interpolation problem is usually solved using polynomial interpolation followed by a rational function reconstruction (achieved by the extended Euclidean algorithm, as shown in Lemma 2.1.2).

### Vector Rational function approximation.

The rational function reconstruction problem can be generalized to vectors with a common denominator.

**Problem 2.1.3 (*Vector Rational Function Reconstruction*)**

**Given** $\ell + 1$ *polynomials* $A, B^{(1)}, \dots, B^{(\ell)}$ *over a field* $\mathsf{K}$ *with* $\deg B^{(i)} < \deg A = n$ *and two positive integers* $d_f$ *and* $d_g$.

**Find** $\ell + 1$ *polynomials* $(g, f^{(1)}, \dots, f^{(\ell)})$, *with* $g$ *monic,* $\deg f^{(i)} \leq d_f$, $\deg g \leq d_g$ *such that*

$$f^{(i)} = gB^{(i)} \mod A \quad \forall i \in \{1 \dots \ell\}$$

The set of solutions to this problem is again a subset of a $\mathsf{K}[X]$-free module, satisfying the degree conditions, but this module now may no longer be uniquely generated: it has a rank $s$ that can be bounded in theorem 2.1.2.

**Theorem 2.1.2 (*[OS07]*)**

$s \leq k$ *for* $k \in \mathbb{Z}_{>0}$ *minimal such that* $n > d_f + d_g/k$.

Aside from this result, Olesh and Storjohann show how to reduce the computation of these $s$ minimal generators to polynomial lattice reduction, which can be performed in $O(\ell k^{\omega-1} \mathsf{M} n)$ using the fast minimal basis computation of [GJV03].

Similarly as in the one dimensional case, when $A = X^n$ the problem is a simultaneous Padé approximation problem.

## 2.2 Dense polynomial evaluation codes

### 2.2.1 Reed-Solomon codes

A first and simple instance of evaluation code is that of polynomials over a finite field.

$$\mathcal{RS}(n,k) = \{\mathrm{Ev}_{(x_1,\dots,x_n)}(f) : f \in \mathsf{K}[X]_{\leq k}\} \subseteq \mathsf{K}^n.$$

Such codes correspond to the well known Reed-Solomon codes [RS60]: although first introduced in the setting of evaluation codes, they have later been cast as a special case of BCH codes, loosing this property.

These codes are MDS, as their minimum distance maximizes the Singleton bound.

$$\delta = n - k. \tag{2.3}$$

Indeed, if a code-word had less than $n-k$ non zero coefficients, it would be an evaluation of a degree $\leq k$ polynomial vanishing on at least $k+1$ points, namely the zero polynomial, and the code-word would be the zero vector.

**Unique decoding.** Consider a code-word $c = (c_i) = \mathrm{Ev}_{(x_1,\dots,x_n)}(f)$ for some $f \in \mathsf{K}[X]$. Let $r = (r_i) = c + e$ be the received word and $e$ the error vector, with $w_H(e) = t \leq \frac{\delta-1}{2}$. Let $S = \{i : e_i \neq 0\}$ be the error support and define $\Lambda = \prod_{i\in S}(X - x_i)$, the error locator polynomial. Lastly, define $\Pi = \prod_{i=1}^{n}(X - x_i)$ and $h \in \mathsf{K}[X]$, the polynomial interpolant of the pairs $(x_i, r_i)_{1\leq i\leq n}$, that is: $h(x_i) = r_i \; \forall i \in \{1 \dots n\}$.

One shows that the congruence

$$\Lambda f = \Lambda h \mod \Pi \tag{2.4}$$

holds by noting that either $\Lambda(x_i) = 0$ for $i \in S$ or $f(x_i) = h(x_i)$. This congruence is often referred to as the key equation, and is at the core of most decoding algorithm. Indeed, it is an instance of a rational function reconstruction problem: given $h$ and $\Pi$, find $F = \Lambda f$ and $G = \Lambda$ of degree respectively $k + t$ and $t$, such that $F = \Lambda h \mod \Pi$. Note that it differs from a Cauchy interpolation problem by the fact that the numerator $\Lambda f$ and the denominator $\Lambda$ are not co-prime. More precisely, the pair $(F = \Lambda f, G = \Lambda)$ is actually the one of least degree solving the rational function reconstruction problem: for any proper divisor $\Gamma$ of $\Lambda$, there is an $x_i$ such that $\Lambda(x_i) = 0 \neq \Gamma(x_i)$ and $\Gamma(x_i)f \neq \Gamma(x_i)h$. Therefore Lemma 2.1.2 ensures that the solution $(F = \Lambda f, G = \Lambda)$ will be computed by the extended Euclidean algorithm.

A wide range of algorithms have been proposed for the unique decoding of Reed-Solomon codes. The Berlekamp/Welch [BW86] decoder solves the congruence (2.4) using linear algebra whereas many algorithms [Shi88; Gao02] reduce to the extended Euclidean algorithm as in Lemma 2.1.2: applying the extended Euclid algorithm on $r_0 = \Pi, r_1 = h$ and halting when $\deg r_i \leq \frac{d_f+n-1}{2} < \deg r_{i-1}$.

Another family of algorithms, rely on a variation of the key equation, written as a Pade approximation problem. From (2.4), there exist a polynomial $Q \in \mathsf{K}[X]_{\leq t-1}$ such that $\Lambda f = \Lambda h + Q\Pi$.

For a polynomial $P$ of degree $\leq k$, we define the degree $k$ reverse polynomial of $P$ as $\mathrm{rev}_k(P)(X) = X^k P(1/X)$ [1]. Evaluating the congruence (2.4) in $X = 1/z$ and multiplying both sides by $z^{n+t-1}$ yields

$$z^{n-d_f-1}\mathrm{rev}_t(\Lambda)\mathrm{rev}_{d_f}(f) = \mathrm{rev}_t(\Lambda)\mathrm{rev}_{n-1}(h) + \mathrm{rev}_{t-1}(Q)\mathrm{rev}_n(\Pi),$$

With the additional constraint that $\mathsf{K} = \mathbb{F}_q$, $n = q - 1$ and $\{x_1, \dots, x_n\} = \mathbb{F}_q^*$ (which is the standard setup for Reed-Solomon codes viewed as BCH codes), we have $\Pi = X^n - 1$, and $\mathrm{rev}_n(\Pi)(z) = 1 - z^n$ and consequently

$$\mathrm{rev}_t(\Lambda)\mathrm{rev}_{n-1}(h) = -\mathrm{rev}_{t-1}(Q) \mod z^{n-d_f-1}. \tag{2.5}$$

This is now a Padé approximation problem for which Berlekamp/Massey algorithm computes a solution provided that $\deg \mathrm{rev}_t(\Lambda) + \deg \mathrm{rev}_{t-1}(Q) + 1 \leq n - d_f - 1$ which is $t \leq \frac{\delta-1}{2}$.

[1] This is a slight generalization of the standard definition of the reciprocal polynomial: normally $k = \deg P$ but we only have here upper bounds on the degrees of some of the polynomials in the congruence.

### 2.2.2 Parameter oblivious decoding

Evaluation-interpolation schemes are often used in a context where the size of the result to be reconstructed is only known by an a priori upper bound. The discrepency between this bound and the actual value implies an over-estimation in the number of evaluation points to be used. For example, consider the polynomial matrix

$$A = \begin{bmatrix} x^5 + x^2 + 1 & x^3 + x + 1 \\ x^4 & x^2 + 1 \end{bmatrix}.$$

While the Hadamard bound predicts a determinant of degree bounded by 8, indicating that 9 evaluations of the matrix will be sufficient, the determinant is in fact $2x + 1$, hence only two evaluations would suffice. As the interpolated result remains same for any additional evaluation, one can perform the interpolation on the fly (e.g. using Newton's formula) and declare the result correct as soon as a stabilization is detected. This technique, called *early termination* [Fre+88; KT90; Emi98; KLL00], speeds-up computations in practice and make the complexities output sensitive, at the expense of succeeding only with a guaranteed probability. This probability can be amplified exponentially by checking the stabilization on a few additional evaluations.

Now, in the case where some evaluations can be erroneous, such a stabilization will not occur and one has to compute all $n \geq d_f + 1$ evaluations, where $d_f$ bounds the result's degree. Then, up to $\tau = \frac{n-d_f-1}{2}$ errors can be corrected by running the extended Euclidean algorithm with termination when the degree of the current remainder becomes less or equal than $\frac{n+d_f-1}{2}$. Now remark that the discrepency between $\deg f$ and $d_f$ is a form a redundancy, that can be used for error correction: if $\deg f$ were known, one could run the extended Euclidean algorithm until $\deg f_i \leq \frac{n+\deg f-1}{2}$ and hence correct up to $\overline{\tau} = \frac{n-\deg f-1}{2} \geq \tau$ errors.

Therefore we proposed in [R-Kho+10] a parameter oblivious decoding algorithm that simply consists in letting the extended Euclidean algorithm run and checking at each iteration if it corresponds to a feasible decoding. The output is now a list of possible solutions. The criterion to declare that an iteration $f_i = g_i h + h_i \Pi$ in the extended Euclidean algorithm corresponds to a valid decoding is that

$$g_i \text{ divides } \Pi \text{ and } f_i. \tag{2.6}$$

**Example 2.2.1**

*Over $\mathbb{Z}/17\mathbb{Z}$, consider the evaluation points $(1, 2, 3, 4, 5, 6, 7)$ and the received word $(16, 16, 16, 12, 16, 16, 10)$ which corresponds to the polynomial interpolant $h = X^6 + 7X^5 + 5X^4 + 15X^2 + 8X + 14$. The run of the extended Euclidean algorithm applied to $f_0 = \Pi = \prod_{i=1}^{7}(X - i)$ and $f_1 = h$ is summarized in following table (with polynomials written in factorized form to reveal the possible common factors):*

| $i$ | $f_i$ | $g_i$ | $q_i$ |
|---|---|---|---|
| 0 | $(X-1)(X-2)\dots(X-6)(X-7)$ | 0 | |
| 1 | $X^6 + 7X^5 + 5X^4 - 2X^2 + 8X - 3$ | 1 | |
| 2 | $(X-1)(X^4 + X^3 + 6X^2 + 5X - 6)$ | $-(X-1)$ | $X - 1$ |
| 3 | $16(X-7)(X-4)$ | $(X-7)(X-4)$ | $X + 7$ |
| 4 | $9(X+9)$ | $(X+6)(X^4 + 11X^3 + 7X^2 + 8X + 16)$ | $-X^3 + 6X^2 + 4X - 4$ |
| 8 | 13 | $2(X^3 - 2X^2 - 5X + 4)(X^3 - X^2 + 8X + 8)$ | $15X + 6$ |
| 9 | 0 | | $2X + 1$ |

*From the divisibility criterion, there are 3 possible decodings:*

1. *the code-word $Ev(X^6 + 7X^5 + 5X^4 - 2X^2 + 8X - 3) \in \mathcal{RS}(7, 7)$ with no error,*
2. *the code-word $Ev(X^4 + X^3 + 6X^2 + 5X - 6) \in \mathcal{RS}(7, 5)$ with one error at evaluation point $x_1 = 1$,*
3. *the code-word $Ev(16) \in \mathcal{RS}(7, 3)$ with two errors at evaluation points $x_4 = 4$ and $x_7 = 7$,*

Note that contrarily to the standard list decoding techniques [Sud97; GS99], that return the list of codewords in a given code at a distance beyond the unique decoding radius, this list decoding returns the list of the possible unique decodings in all Reed-Solomon codes of length $n$ (and dimension $k$ ranging in $\{1, \dots, n\}$).

On this example, note that third possible decoding ($Ev(16)$) has some unused redundancy (5 evaluations would have been sufficient to reconstruct a degree 0 polynomial with 2 errors), and that the quotient

preceding this iteration has an abnormally large degree. This is a general fact formalized in lemma 2.2.1 that any oversampling will be detected by a surge in the degree of preceeding quotient.

**Lemma 2.2.1**

*If $n = d_f + 2t + 1 + \Delta$ for some $\Delta \geq 0$, then at the final iteration $j$ of the extended Euclidean algorithm run on $\Pi$ and $h$, such that $\deg f_j \leq \frac{n+d_f-1}{2} < \deg f_{j-1}$, we have $\deg q_j \geq \Delta$.*

**Proof.** At iteration $j$, the pair $(f_j, g_j)$ is the unique minimal solution to the rational reconstruction problem (2.4). Hence $f_j = \Lambda f$ and $g_j = \Lambda$. Now

$$\deg q_j = \deg f_{j-1} - \deg f_j = n - \deg g_j - \deg f - t \geq \Delta.$$

□

This property can be used to reduce the number of iterations where the divisibility condition (2.6) is checked, to only those with a quotient greater than an arbitrary threshold $\Delta$. This is done at the expense of loosing a decoding capacity of $\Delta/2$, but experiments show [R-Kho+10] that a very small value of $\Delta$ suffices to filter out most iterations.

Rational reconstruction with early termination based on the degree of the quotient was first proposed in [KM06]. We then applied it in a coding theory framework in [R-Kho+10] for both Reed-Solomon codes and CRT codes, introducing the approach of trying and decode all possible codes for a given length, referred to as parameter oblivious decoding.

## 2.3 Rational function codes

### 2.3.1 The code and its minimum distance

In this section we generalize the Reed-Solomon codes to the evaluation of rational functions, and therefore study the problem of Cauchy interpolation in the presence of errors in the evaluations. In the most general setting, we allow evaluations at poles of the rational function. By convention, if $x$ is a pole of a rational function $f$, we set $f(x) = \infty$. Let $\overline{\mathsf{K}} = \mathsf{K} \cup \{\infty\}$ and the function $\mathrm{Ev}_{(x_1,\ldots,x_n)}$ generalizes naturally to

$$\begin{array}{rccc} \mathrm{Ev}_{(x_1,\ldots,x_n)} : & \mathsf{K}(X) & \longrightarrow & \overline{\mathsf{K}}^n \\ & f & \longmapsto & (f(x_1), \ldots, f(x_n)) \end{array}.$$

Note that the problem of Cauchy interpolation in the presence of errors has been used as an inner tool in the list decoding of Reed-Solomon codes by Wu's algorithm [Wu08; Bee+13]. We present and study it here in the setting of a code.

**Definition 2.3.1**

*The rational function code over the alphabet $\overline{\mathsf{K}}$ with parameters $(n, d_f, d_g)$ with $n > d_f + d_g$ is the set*

$$\mathcal{RF}(n, d_f, d_g) = \{Ev_{(x_1,\ldots,x_n)}\left(\frac{f}{g}\right) : f \in \mathsf{K}[X]_{\leq d_f}, g \in \mathsf{K}[X]_{\leq d_g}\} \subseteq \overline{\mathsf{K}}^n.$$

**Theorem 2.3.1**

*A rational function code of parameters $(n, d_f, d_g)$ has minimal distance equal to $\delta = n - d_f - d_g$.*

**Proof.** Let $(b_1^{(1)}, \ldots, b_n^{(1)})$ and $(b_1^{(2)}, \ldots, b_n^{(2)})$ be two code-words and $\frac{f_1}{g_1}$ and $\frac{f_2}{g_2}$ their corresponding reduced rational functions. Suppose $d_H(b^{(1)}, b^{(2)}) < n - d_f - d_g$. Then $|\{i : b_i^{(1)} = b_i^{(2)}\}| > d_f + d_g$. The polynomial $f_1 g_2 - f_2 g_1$ has degree less or equal than $d_f + d_g$ and vanishes on at least $d_g + d_f + 1$ values $x_i$. Hence $f_1 g_2 = f_2 g_1$ which implies that $b^{(1)} = b^{(2)}$ and consequently $\delta \geq n - d_f - d_g$.

On the other hand, consider the polynomials $f_1 = \prod_{i=1}^{d_f}(X - x_i)$, $f_2 = \alpha f_1$ and $g = \prod_{i=1}^{d_g}(X - x_{i+d_f+1})$, for some $\alpha \in \mathsf{K} \setminus \{0, 1\}$. The polynomials $f_1$ and $f_2$ of degree $d_f$ are co-prime with $g$. Then the vectors

$$b^{(1)} = \mathrm{Ev}_{(x_1,\ldots,x_n)}\left(\frac{f_1}{g}\right) = (\underbrace{0,\ldots,0}_{d_f \text{ times}}, \frac{f_1}{g}(x_{d_f+1}), \underbrace{\infty,\ldots,\infty}_{d_g \text{ times}}, *, \ldots, *)$$

$$b^{(2)} = \mathrm{Ev}_{(x_1,\ldots,x_n)}\left(\frac{f_2}{g}\right) = (\underbrace{0,\ldots,0}_{d_f \text{ times}}, \alpha\frac{f_1}{g}(x_{d_f+1}), \underbrace{\infty,\ldots,\infty}_{d_g \text{ times}}, *, \ldots, *)$$

are distinct code-words of an $(n, d_f, d_g)$ rational function code at Hamming distance at most $n - d_f - d_g$. □

**Corollary 2.3.1**

*Let $d_f, d_g, n \in \mathbb{Z}_{>0}$ such that $\delta = n - d_g - d_f > 0$ and $\mathsf{K}$ be a field. Then for any vector $(r_1, \ldots, r_n) \in \overline{\mathsf{K}}^n$, there exists at most one pair of polynomials $f \in \mathsf{K}[X]_{\leq d_f}, g \in \mathsf{K}[X]_{\leq d_g}$ such that*

$$|\{i \text{ s.t. } (r_i \neq \infty \text{ and } f(x_i) \neq r_i g(x_i)) \text{ or } (r_i = \infty \text{ and } g(x_i) \neq 0)\}| \leq \frac{\delta - 1}{2}.$$

**Proof.** If two such pairs of polynomials $(f, g)$ would exist, this would contradict the fact that $\delta$ is the minimal distance of the code. □

### 2.3.2 A unique decoding algorithm

As for the Reed-Solomon codes, we will show that the decoding reduces to solving a rational function reconstruction problem, which in turn is solved by e.g. the extended Euclidean algorithm as presented in section 2.1.3.

Consider a rational function code $\mathcal{RF}(n, d_f, d_g)$. Let $f(X) \in \mathsf{K}[X]_{\leq d_f}, g(X) \in \mathsf{K}[X]_{\leq d_f}$ with $g$ monic, $\mathrm{GCD}(f, g) = 1$ and consider the corresponding code-word $(c_1, \ldots, c_n) = \mathrm{Ev}_{(x_1,\ldots,x_n)}\left(\frac{f}{g}\right)$ in $\mathcal{RF}$. Given $n, d_f, d_g$ and a received vector $(r_1, \ldots, r_n)$ such that $r_i \neq c_i$ on $t$ values of $i$, forming the error position set $S$. We wish to recover the rational function $f/g \in \mathsf{K}(X)$.

Let $\Pi = \prod_{i=1}^{n}(X - x_i)$ and $P_\infty(X) = \prod_{r_i=\infty}(X - x_i)$ with $\deg P_\infty = n_\infty$. Let $\bar{h}(X)$ be the polynomial interpolant for the $n - n_\infty$ argument/value pairs $(x_i, P_\infty(x_i)r_i)_{r_i \neq \infty}$. We have $\deg \bar{h} \leq n - n_\infty - 1$. Note that if $x_i$ is a pole, $f(x_i) \neq 0$ because $f/g$ is reduced; however for such poles we can erroneously have $r_i \neq \infty$ ("false non-pole"). In addition, for an error index $i \in S$ we allow $r_i = \infty$ even when $c_i \neq \infty$, that is when $g(x_i) \neq 0$ ("false pole"). Let $\Lambda(X) = \prod_{i \in S}(X - x_i)$ be the error locator, and let

$$\Lambda_\infty(X) = \prod_{i \in S : r_i = \infty}(X - x_i), \quad g_\infty(X) = \prod_{i \notin S : r_i = \infty}(X - x_i). \tag{2.7}$$

Note that, at the roots $x_i$ of $g_\infty$, we have $r_i = c_i = \infty$, hence $g_\infty$ divides $g$. Because the roots of $\Lambda_\infty$ constitute the false poles, none are roots of $g$, so $\mathrm{GCD}(\Lambda_\infty, g) = 1$. Moreover $P_\infty = \Lambda_\infty g_\infty$. Let $\bar{g} = g/g_\infty$, $\bar{\Pi} = \Pi/P_\infty$ and $\bar{\Lambda} = \Lambda/\Lambda_\infty$.

**Lemma 2.3.1**

*If $t \leq \frac{\delta-1}{2}$, the pair $(F = f\Lambda, G = \bar{g}\bar{\Lambda})$ is a minimal solution to the rational function reconstruction problem RatFunRec $(\bar{\Pi}, \bar{h}, d_f + \frac{\delta-1}{2})$.*

**Proof.** The decoding relies on a generalization of the key equation (2.4):

$$\underbrace{\Lambda f}_{F} \equiv \underbrace{\bar{\Lambda}\bar{g}}_{G}\,\bar{h} \pmod{\bar{\Pi}} \tag{2.8}$$

There are two kinds of $x_i$ in the congruence (2.8): either $x_i$ for $i \in S$ and $r_i \neq \infty$: then $\bar{\Lambda}(x_i) = 0$ on both sides; or $x_i$ with $r_i = c_i \neq \infty$: then $(\Lambda f)(x_i) = \Lambda(x_i)g(x_i)c_i = \bar{\Lambda}(x_i)\bar{g}(x_i)P_\infty(x_i)r_i = \bar{\Lambda}(x_i)\bar{g}(x_i)\bar{h}(x_i)$,

which proves (2.8). Now since $\deg F \leq d_f + t \leq d_f + \frac{\delta-1}{2}$ and $\deg G \leq d_g - \deg g_\infty + t - \deg \Lambda_\infty = d_g - n_\infty + t = n - d_f - \delta + t \leq n - d_f - \frac{\delta-1}{2} - 1$, we deduce that the pair $(F, G)$ is a solution to the problem $RatFunRec(\bar{\Pi}, \bar{h}, d_f + \frac{\delta-1}{2})$.

Let $(F', G')$ be another solution to this problem. From lemma 2.1.1, $FG' = F'G$, hence

$$f\Lambda_\infty G' = F'\bar{g} = G'\bar{h}\bar{g}.$$

Let $i \in S$ with $r_i \neq \infty$, so that $\Lambda_\infty(x_i) \neq 0$ and $r_i \neq c_i$, we have $f(x_i)\Lambda_\infty(x_i)G'(x_i) = G'(x_i)r_i\Lambda_\infty(x_i)g(x_i)$ or equivalently $G'(x_i)(f(x_i) - r_i g(x_i)) = 0$. If $g(x_i) = 0$ then necessarily $f(x_i) \neq 0$, and therefore $G'(x_i) = 0$. If $g(x_i) \neq 0$, then $G'(x_i)(c_i - r_i) = 0$, and again $G'(x_i) = 0$ as $c_i \neq r_i$. Thus $\bar{\Lambda}$ divides $G'$, but also $F'$, from equation (2.8), and therefore $\mathrm{GCD}(F, G)$ divides $\mathrm{GCD}(F', G')$ showing the minimality of $(F, G)$. □

Corollary 2.3.2 is a direct consequence of lemmas 2.1.2 and 2.3.1 and the decoding algorithm (Algorithm 2) follows.

**Corollary 2.3.2**

*The extended Euclidean algorithm run on* $(f_0 = \prod_{r_i \neq \infty}(X - x_i), f_1 = \bar{h})$ *and terminated at iteration* $j$ *when* $\deg f_j \leq d_f + \frac{\delta-1}{2} < \deg f_{j-1}$ *returns*

$$f_j = f\Lambda,\ g_j = \bar{g}\bar{\Lambda}.$$

**Algorithm 2** Decoding Rational Function Code

**Require:** $n, d_f, d_g \in \mathbb{Z}_{>0}$
**Require:** $(x_1, \ldots, x_n) \in \mathsf{K}^n$: $n$ distinct elements of $\mathsf{K}$
**Require:** $(r_1, \ldots, r_n) \in \overline{\mathsf{K}}^n$
**Ensure:** $f \in \mathsf{K}[X]_{\leq d_f}, g \in \mathsf{K}[X]_{\leq d_g}$ if they exist (or returns FAIL) such that $\mathrm{GCD}(f, g) = 1$ and $|\{i \in \{1 \ldots n\} : (r_i = \infty \text{ and } g(x_i) \neq 0) \text{ or } f(x_i) \neq r_i g(x_i)\}| \leq \frac{n - \deg f - \deg g - 1}{2}$
**Ensure:** $\bar{\Lambda} = \prod_{r_i \neq \infty \text{ and } f(x_i) \neq r_i g(x_i)}(X - x_i)$
**Ensure:** $\Lambda_\infty = \prod_{r_i = \infty \text{ and } g(x_i) \neq 0}(X - x_i)$
**Ensure:** $g_\infty = \prod_{r_i = \infty \text{ and } g(x_i) = 0}(X - x_i)$
1: $P_\infty \leftarrow \prod_{r_i = \infty}(X - x_i)$; $n_\infty = \deg P_\infty$
2: $\bar{h} \leftarrow \mathrm{Interpolant}((r_i P_\infty(x_i), x_i)_{r_i \neq \infty})$
3: $f_0 \leftarrow \prod_{r_i \neq \infty}(X - x_i)$; $f_1 = \bar{h}$
4: $f_j, g_j \leftarrow RatFunRec(f_0, f_1, d_f + \frac{\delta-1}{2})$
5: $\bar{\Lambda} \leftarrow \mathrm{GCD}(f_j, g_j)$
6: $\Gamma \leftarrow f_j / \bar{\Lambda}$
7: $\Lambda_\infty \leftarrow \prod_{i : r_i = \infty \text{ and } \Gamma(x_i) = 0}(X - x_i)$
8: $g_\infty \leftarrow \prod_{i : r_i = \infty \text{ and } \Gamma(x_i) \neq 0}(X - x_i)$
9: $f \leftarrow \Gamma / \Lambda_\infty$
10: $\bar{g} \leftarrow g_j / \bar{\Lambda}$
11: $g \leftarrow g_\infty \bar{g}$
12: **if** $\deg g \geq d_g$ or $\deg f \geq d_f$ or $\bar{\Lambda}$ does not divide $f_0$ **then return** FAIL
13: **end if**

## 2.4 Interleaving

Applications in signal processing, where error correcting codes were primarily used, revealed a need to handle bursts of errors. In this context, interleaving became a powerfull tool to enable the correction of such bursts that a classic code would have not been able to correct. A simple interleaving of depth $\ell$ of a block code consists in storing $\ell$ code-words $c^{(1)}, \ldots, c^{(\ell)}$ in a matrix (or interleaving table) $[c_{i,j}] \in \mathsf{K}^{\ell \times n}$ such that $c^{(i)} = (c_{i,1}, \ldots, c_{i,n})$ and then send the symbols of the matrix in column-major order: $c_{1,1}, c_{2,1}, \ldots, c_{\ell,1}, c_{1,2}, \ldots$.

A burst of error in the channel (corrupting consecutive symbols) of length $\leq \ell$ would then be spread over distinct code words, and can therefore be more easily corrected. More precisely if one can correct $t$ errors on each code-word, an interleaving of depth $\ell$ allows to correct a burst of length up to $\ell t$. More sophisticated interleaving techniques have been developed to further improve the resilience to bursts and isolated errors simultaneously, the most famous one being the Cross Interleaved Reed-Solomon codes (CIRC) used in the audio CD standard [Oda+83; Moo05].

### 2.4.1 Collaborative decoding

A further improvement is to remark that in a simple interleaving with error bursts, the decoding of each row should not be done independently as the errors share the same column support. Instead, collaborative decoding factors out this information so as to increase the correction capactity.

Let $h^{(1)}, \dots, h^{(\ell)}$ be the polynomial interpolants of the $\ell$ received words and $f^{(1)}, \dots, f^{(\ell)}$ be the polynomial interpolants of the $\ell$ code-words $c^{(1)}, \dots, c^{(\ell)}$. Let $S_1, \dots, S_\ell$ be the error support for each of received word and $S = S^{(1)} \cup \dots \cup S^{(\ell)}$. We have a system of key equations

$$\begin{cases} \Lambda f^{(1)} &= \Lambda h^{(1)} \mod \Pi \\ &\vdots \\ \Lambda f^{(\ell)} &= \Lambda h^{(\ell)} \mod \Pi \end{cases} \tag{2.9}$$

with $\Lambda = \prod_{i \in S}(X - x_i)$. The linearization $N^{(i)} = \Lambda f^{(i)} \, \forall i \in \{1 \dots \ell\}$ transforms it into a vector rational function reconstruction problem, as problem 2.1.3. As for the one-dimensional key equation, it can also be transformed into a simultaneous Padé approximation problem.

Making the assumption that errors happen in bursts and have therefore the same support for every interleaved words, we set $t = |S| \approx |S^{(1)}| \approx \dots \approx |S^{(\ell)}|$. The corresponding linear system is formed by $n\ell$ equations for $t + \ell(d_f + 1 + t)$ unknowns. Hence a necessary condition for a unique decoding is

$$t \leq \frac{\ell}{\ell + 1}(n - d_f - 1).$$

that generalizes the unique decoding radius of Reed-Solomon codes (obtained with $\ell = 1$).

However, the system is not necessarily non-singular, and in the worst case, fewer errors can be decoded. Collaborative decoding usually makes genericity assumptions [SSB06] or guaranties success with a bounded probability with respect to a given distribution of vector rational reconstruction systems [BKY03; KY14]. Using the vector rational function reconstruction of [OS07] mentioned in section 2.1.3, the failure case is when no element of the minimal basis generating the free module of solutions of (2.9) is a vector of the form $(\Lambda, \Lambda f^{(1)}, \dots, \Lambda f^{(\ell)})$.

This approach, based on a system of key equations, was first proposed in [BKY03; BKY07], in the context of simultaneous rational reconstruction, where the resolution of system 2.9 is done by linear algebra, as in the Berlekamp/Welch decoding. Earlier work, by Feng and Tzeng proposed the so called *fundamental iterative algorithm* [FT85; FT91] that generalizes the Berlekamp/Massey algorithm over multiple recurring sequences and therefore solves the corresponding simultaneous Padé approximation problem. More recently Schmidt, Sidorenko and Bossert improved Feng and Tzeng's algorithm to support sequences of varying length [SSB06].

This made way to an interesting construction for decoding Reed-Solomon codes beyond half their minimum distance: power decoding [SSB10]. Consider a code-word $c = \mathrm{Ev}_{(x_1,\dots,x_n)}(f)$, with $f \in \mathsf{K}[X]_{\leq k}$ and define $\ell \leq \lfloor \frac{n}{k} \rfloor$. Noting that the vector $c^{(i)} = (c_1^i, \dots, c_n^i)$ for $i \in \{1 \dots \ell\}$ is a code-word of a $(n, ki)$-Reed-Solomon code, one can build an interleaving table by simply raising the code-word symbols to consecutives powers. Given a receveived word $r$, of interpolant $h$, define $r^{(i)} = (r_1^i, \dots, r_n^i)$ and $h^{(i)} = h^i \mod \Pi$, so that $h^{(i)}(x_j) = r_j^i \, \forall i \in \{1 \dots \ell\}, j \in \{1 \dots n\}$. This construction ensures a structure of burst to the errors: each $r^{(i)}$ differs from the corresponding $c^{(i)}$ on the exact same positions. The key equation system becomes

$$\begin{cases} \Lambda f^{1} &= \Lambda h^{(1)} \mod \Pi \\ &\vdots \\ \Lambda f^{\ell} &= \Lambda h^{(\ell)} \mod \Pi \end{cases}. \tag{2.10}$$

The necessary condition for uniqueness becomes $(\ell+1)t+\sum_{i=1}^{\ell} ik \leq \ell n$. With the additional constraint that a syndrome of size at least $t$ should exist on the last row, we have $t \leq n - lk$. The two equations combined together yield $\frac{k}{n} = \frac{2}{\ell(\ell+1)}$, and $t \leq n - \sqrt{2kn}$, matching the correction radius of Sudan's first list decoding algorithm [Sud97].

Algorithmic aspects of efficient list decoding for Reed-Solomon codes have been intensively studied [Ale02; CS03; PV05; Wu08; CH10; Ber11; Qui12; Zeh12; Nie13; Cho+14] and the best complexities are now obtained through either polynomial lattice reduction, or structured linear algebra. The reader will find in [Cho+14] an survey on the topic with more complete references.

### 2.4.2 Application to fault tolerant exact linear system solver

Interleaving of the rational function codes of section 2.3 finds a natural application in fault tolerant linear system solving over the field of rational functions $\mathsf{K}(X)$.

Consider a polynomial linear system $Ay = b$ for a non-singular matrix $A \in \mathsf{K}[X]_{\leq d}^{m\times m}$ and a vector $b \in \mathsf{K}[X]_{\leq d}^{m}$ with a solution $y$ living in $\mathsf{K}(X)^m$.

A distributed computation of the solution can be obtained by solving $n$ evaluations of the system in distinct points $x_1, \ldots, x_n \in \mathsf{K}$: $y^{(i)} = A(x_i)^{-1}b(x_i)$, using for example Gaussian elimination over $\mathsf{K}$ as in section 1.3. A polynomial vector interpolant $h = (h_1, \ldots, h_m) \in \mathsf{K}[X]^m$ of these solutions is computed, and the result is obtained by rational function reconstructions.

From Cramer's rule and Hadamard bound, the numerators and the denominator of the solution $y$ have degree $\leq md$. Hence $n = 2md+1$ evaluations suffice for the rational function reconstruction (problem 2.1.1) of each component of $y$ to succeed. But remarking that the $m$ instances of these reconstructions share the same denominator, the problem is more precisely a vector rational function reconstruction (problem 2.1.3), and therefore only requires $n = md + d + 1$ evaluation points to find its unique solution. Indeed, from theorem 2.1.2, we know that the minimal basis of solutions to the vector rational function reconstruction contains $s \leq m$ vectors $(g^{(i)}, f_1^{(i)}, \ldots, f_m^{(i)})$, $i \in \{1 \ldots s\}$ such that $A(f_1^{(i)}, \ldots, f_m^{(i)})^T = g^{(i)}b \mod \Pi$. As $\max(\deg A + \deg f^{(i)}, \deg b + \deg d^{(i)}) \leq md + d < \deg \Pi$ we have $A(f_1^{(i)}, \ldots, f_m^{(i)})^T = d^{(i)}b$, and since the solution is unique, $s = 1$. This is the approach proposed in [OS07], originating from [Cab71].

Now consider the case where at some evaluation points $x_i$, the matrix $A_i$ may be singular, or that the result $y_i$ returned is erroneous. An interleaved version of the rational function codes of section 2.3 can be applied to reconstruct the result.

The key equation system now becomes

$$\begin{cases} \Lambda f_1 &= \bar{\Lambda}\bar{g}\bar{h}_1 \mod \bar{\Pi} \\ &\vdots \\ \Lambda f_m &= \bar{\Lambda}\bar{g}\bar{h}_m \mod \bar{\Pi} \end{cases}, \tag{2.11}$$

where $y = (f_1, \ldots, f_\ell)/g$ and using the notations of section 2.3.2.

Comparing the number of unknown and of equations, we deduce a necessary condition for unique decoding:

$$n \geq \frac{m+1}{m}t + (m+1)d + n_\infty(1 - \frac{1}{m}) + 1. \tag{2.12}$$

Note that, as expected, the amount of oversampling required to support $t$ errors is reduced to $\frac{m+1}{m}t$. However the presence of poles is no longer transparent for the decoding, as it also impacts $n$ as soon as $m > 1$. An interpretation is that with $m = 1$, when the evaluation point $x_i$ hits a pole, the $\infty$ symbol does not contribute to the interpolation (it is considered as an erasure), but on the other hand, it contributes to the knowledge of a degree one factor of the denominator which compensates the oversampling required for the erasure. Now with $m > 1$, the vector $y = (\infty, \ldots, \infty)$ still only contributes to a degree one factor of denominator, but all other $m - 1$ repetitions of this symbol are useless, and treated as erasures, hence the $\frac{m-1}{m}$ additional term.

## 2.5 Over the integers and rationals

The evaluation-interpolation technique over $\mathsf{K}[X]$ transfers over the ring of integers under the form of residue number systems: evaluations in the distinct points $x_i$ (or equivalently reductions modulo co-prime

$(X - x_i)$'s) are replaced by reductions modulo co-prime integers $m_1, \ldots, m_n$, and interpolation correspond the Chinese Remainder algorithm. Hence most constructions that we used over $\mathsf{K}[X]$ can be transferred to $\mathbb{Z}$, but some complications occur, mostly due to the effect of carries: $\mathsf{K}[X]_{\leq d}$ is stable by addition whereas $\mathbb{Z}_{\leq n}$ is not. For this reason, the rational number reconstruction problem, Problem 2.5.1, also called the Thue problem does not necessarily admit a minimal solution generating the set of solutions as in Lemma 2.1.1.

**Problem 2.5.1 (*RatNumRec* $(A, B, k)$)**

*Given two integers $A > B \geq 0$ and a real number $0 \leq k < A$, find a pair of integers $f, g$ such that*

$$|f| \leq k,\ 0 < g < \frac{A}{k} \text{ and } f = gB \mod A$$

To ensure this, we will use a slightly weaker version of this problem:

**Problem 2.5.2 (*WeakRatNumRec* $(A, B, k)$)**

*Given two integers $A > B \geq 0$ and a real number $0 \leq k < A$, find a pair of integers $f, g$ such that*

$$|f| \leq \frac{k}{2},\ 0 < g < \frac{A}{k} \text{ and } f = gB \mod A$$

so as to state lemma 2.5.1.

**Lemma 2.5.1**

*Any two pairs $(f_1, g_1)$ and $(f_2, g_2)$ of solutions of WeakRatNumRec $(A, B, k)$ problem satisfy $f_1 g_2 = f_2 g_1$.*

**Proof.** $f_1 g_2 = g_1 B g_2 = g_1 f_2 \mod A$. But $|f_1 g_2 - f_2 g_1| \leq |f_1 g_2| + |f_2 g_1| < A$, and therefore $f_1 g_2 = f_2 g_1$. □

The extended Euclidean algorithm solves the rational number reconstruction problem by finding the minimal solution. This is stated in the following lemma, a variation of [KR89, Theorem 5.1] and [GG13, Lemma 5.25].

**Lemma 2.5.2**

*Consider the extended Euclidean algorithm run on $f_0 = A, f_1 = B$, and computing a sequence of remainders $f_\ell$ defined by $f_{\ell-2} = q_\ell f_{\ell-1} + f_\ell$ and $0 \leq |f_\ell| < |f_{\ell-1}|$, together with the multipliers $g_\ell, h_\ell$ defined by $(g_0, h_0) = (0, 1); (g_1, h_1) = (1, 0)$ and $g_\ell = g_{\ell-2} - q_\ell g_{\ell-1}; h_\ell = h_{\ell-2} - q_\ell h_{\ell-1} \forall \ell > 1$ such that $f_\ell = g_\ell f_1 + h_\ell f_0$ holds for every $\ell$.*

*Then at iteration $j$ such that $|f_j| \leq k/2 < |f_{j-1}|$, the pair $(f_j, g_j)$ is a solution to the problem RatNumRec $(A, B, k)$. Furthermore if $|f_j| \leq k/2 < k < |f_{j-1}|$, then*

1. *the pair $(f_j, g_j)$ is also a solution to the problem WeakRatNumRec $(A, B, k)$.*
2. *this solution is minimal: any other solution $(f, g)$ satisfies $f = Qf_j, g = Qg_j$ for some positive integer $Q$.*

### 2.5.1 CRT codes

The evaluation function

$$\begin{array}{rccc} \mathrm{Ev}_{(m_1,\ldots,m_n)}: & \mathbb{Z} & \longrightarrow & \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_n} \\ & f & \longmapsto & (f \mod m_1, \ldots, f \mod m_n) \end{array}$$

defines the Chinese remainder codes (CRT codes) similarly as the Reed-Solomon codes over $\mathsf{K}[X]$:

$$\mathcal{CRT}(\mathcal{N}, \mathcal{K}) = \{\mathrm{Ev}_{(m_1,\ldots,m_n)}(f) : f \in \mathbb{Z} \text{ with } \log_2 |f| \leq \mathcal{K} - 1\}. \tag{2.13}$$

where $\mathcal{N} = \sum_{i=1}^{n} \log_2 m_i$. Note that the parameters $\mathcal{N}$ and $\mathcal{K}$ are positive real numbers, corresponding respectively to the amount of information, in bit-size that the residue system can carry, and the amount of information that the code carries.

Over $\mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_n}$, we define the distance

$$\begin{array}{rccc} \Delta : & \mathcal{S} \times \mathcal{S} & \to & \mathbb{R}_+ \\ & (x, y) & \mapsto & \displaystyle\sum_{1 \le i \le n, x_i \neq y_i} \log_2 m_i \end{array}$$

counting the bit-size of the moduli where two words differ and define the minimum distance of a code $\mathcal{C}$ as $\delta = \min_{(x,y)\in\mathcal{C}^2} \Delta(x, y)$.

**Remark 2.5.1**

*The main difficulty with CRT codes, compared to Reed-Solomon codes, is that each symbol of a word carries a varying amount of information depending on the bit-size of the corresponding moduli $m_i$. The use of the Hamming distance, as it is usually done [WH66; Man76; GRS99], does not take it into account, which imposes a limitation: the code is defined as the evaluation of the integers less than the product of the first smallest $k$ moduli ($\mathcal{K} = \sum_{i=1}^{k} \log_2 m_i$ for $m_1 < \cdots < m_n$). This condition is required in order to state a minimal distance of $\delta = n - k + 1$ and guarantee that $\frac{n-k}{2}$ errors can always be uniquely corrected. Algorithms for unique decoding naturally rely on the extended Euclidean algorithm. However, decoding up to half the minimum distance is difficult: Mandelbaum [Man76] uses a pertubation technique, making the complexity exponential in the worst case. Goldreich, Ron and Sudan [GRS99] showed how to uniquely decode up to $t \le \frac{(n-k)\log m_1}{\log m_1 + \log m_n} < \frac{n-k}{2}$ errors using a rational number reconstruction problem. Finally, Guruswami, Sahai and Sudan [GSS00] proposed the first polynomial time algorithm, using weights, decoding up to half the minimum distance. The last two papers, as well as [Bon00] also addressed the list decoding of CRT codes, thus completing the analogy with the Reed-Solomon codes.*

*The distance $\Delta$ and the parameter setting of Equation (2.13) correspond to an information theoretic point of view on the codes, that we used in [R-Kho+10]. It generalizes the notion of amplitude for a word of [GRS99].This way we can remove the requirement that the moduli must be sorted: in the context of CRT based distributed computations, one may need to chose them randomly (in order to use a probabilistic algorithm). This is also required if one needs to reconstruct and decode the result on the fly, using early termination, and parameter oblivious decoding as in section 2.2.2, as the residues will very likely not appear by increasing order of the corresponding modulo.*

Lemma 2.5.3 states a form of MDS property for the CRT codes defined in (2.13).

**Lemma 2.5.3**

*A $\mathcal{CRT}(\mathcal{N}, \mathcal{K})$ code has minimum distance $\delta \ge \mathcal{N} - \mathcal{K}$.*

**Proof.** Let $f$ and $g$ be two integers, with $|f|, |g| \le 2^{\mathcal{K}-1}$, $c^{(1)}, c^{(2)}$ the corresponding code words and define $\bar{S} = \{i : f = g \mod m_i\}$. Then $\Delta(b^{(1)}, b^{(2)}) = \log_2 \prod_{i:f\neq g \mod m_i} m_i = \mathcal{N} - \log_2 \prod_{i\in\bar{S}} m_i$.

On one hand $e = f - g$ is a non-zero multiple of $\prod_{i\in\bar{S}} m_i$ and therefore $\log |e| \ge \mathcal{N} - \Delta(b^{(1)}, b^{(2)})$. On the other hand $|e| \le |f| + |g| \le 2^{\mathcal{K}}$. Therefore $\Delta(b^{(1)}, b^{(2)}) \ge \mathcal{N} - \mathcal{K}$. □

Similarly as for Reed-Solomon codes, the unique decoding rely on a key equation of the form

$$\Lambda f = \Lambda h \mod \Pi \tag{2.14}$$

where $\Pi = \prod_{i=1}^{n} m_i$, $h$ is the CRT interpolant of the received word, $f \in \{-2^{\mathcal{K}-1} \ldots 2^{\mathcal{K}-1}$ is the CRT interpolant of the code-word, and $\Lambda = \prod_{i:r\neq f \mod m_i} m_i$.

This is an instance of a weak rational number reconstruction. More precisely, if the error locator $\Lambda$ has bit-size $< (\mathcal{N} - \mathcal{K})/2$, then $|\Lambda f| < 2^{\frac{\mathcal{N}+\mathcal{K}}{2}-1}$ and the pair $(F = \Lambda f, G = \Lambda)$ is the minimal solution to the problem: $WeakRatNumRec(\Pi, h, (\mathcal{K} + \mathcal{N})/2)$. Finally, Lemma 2.5.2 ensures that it will be computed by the extended Euclidean algorithm.

### 2.5.2 Rational number codes

The rational function codes can also be transferred to rational number codes. Let $\overline{\mathbb{Z}_m} = \mathbb{Z}_m \cup \{\infty\}$. By convention, for $p$ and $q$ co-prime, we set $\frac{p}{q} = \infty \mod m$ if $\text{GCD}(m, q) > 1$. The evaluation function over $\mathbb{Q}$ becomes:

$$\begin{array}{rccc} \text{Ev}_{(m_1,\ldots,m_n)}: & \mathbb{Q} & \longrightarrow & \overline{\mathbb{Z}_{m_1}} \times \cdots \times \overline{\mathbb{Z}_{m_n}} \\ & r & \longmapsto & (r \mod m_1, \ldots, r \mod m_n) \end{array}$$

**Definition 2.5.1**

*Let $m_1, \ldots, m_n$ be $n$ prime numbers and let $\mathcal{N} = \log_2 \prod_{i=1}^n m_i$ and $0 < \mathcal{K}_f, \mathcal{K}_g < \mathcal{N}$ be two real numbers. The rational number code with parameters $(\mathcal{N}, \mathcal{K}_f, \mathcal{K}_g)$ is defined as*

$$\mathcal{RN}(\mathcal{N}, \mathcal{K}_f, \mathcal{K}_g) = \{Ev_{(m_1,\ldots,m_n)}\left(\frac{f}{g}\right) : f, g \in \mathbb{Z} \times \mathbb{Z}_{>0} \text{ with } \log_2 |f| \leq \mathcal{K}_f - 1, \log_2 g \leq \mathcal{K}_g\}$$

We extend the distance $\Delta$ over the set $\overline{\mathbb{Z}_{m_1}} \times \cdots \times \overline{\mathbb{Z}_{m_n}}$ from which we can define the minimal distance of a code $\mathcal{C}$ as $\delta = \min_{(x,y)\in\mathcal{C}^2} \Delta(x, y)$.

**Theorem 2.5.1**

*A rational number code with parameters $(\mathcal{N}, \mathcal{K}_f, \mathcal{K}_g)$ has minimum distance $\delta \geq \mathcal{N} - \mathcal{K}_f - \mathcal{K}_g$.*

**Proof.** Let $(b_1^{(1)}, \ldots, b_n^{(1)})$ and $(b_1^{(2)}, \ldots, b_n^{(2)})$ be two distinct code-words and $\frac{f_1}{g_1}$ and $\frac{f_2}{g_2}$ their corresponding reduced rational fractions ($\text{GCD}(f, g) = 1$). Consider the integer $e = f_1 g_2 - f_2 g_1$ and the set $S = \{i : e = 0 \mod m_i\}$.

$$\Delta(b^{(1)}, b^{(2)}) = \mathcal{N} - \log_2 \prod_{i\in S} m_i.$$

On one hand, $e$ is a non-zero multiple of $\prod_{i\in S} m_i$ as $e \mod m_i = 0 \ \forall i \in S$. Hence $\log_2 |e| \geq \mathcal{N} - \Delta(b^{(1)}, b^{(2)})$. On the other hand $|e| \leq |f_1 g_2| + |f_2 g_1| \leq 2^{\mathcal{K}_f + \mathcal{K}_g}$. This implies that $\Delta(b^{(1)}, b^{(2)}) \geq \mathcal{N} - \mathcal{K}_f - \mathcal{K}_g$. □

Unlike we did in the previous section, we can not state that this lower bound is the actual value of the minimal distance. However we can still construct a pair of code words in some rational number codes that lie within a distance of $\mathcal{N} - \mathcal{K}_f - \mathcal{K}_g + 1$: let $f_1 = \prod_{i=1}^{k_f} m_i, f_2 = -f_1, g = \prod_{i=k_f+1}^{k_f+k_g} m_i$. The code-words $b^{(1)} = \text{Ev}_{(m_1,\ldots,m_n)}\left(\frac{f_1}{g}\right)$ and $b^{(2)} = \text{Ev}_{(m_1,\ldots,m_n)}\left(\frac{f_2}{g}\right)$ are such that $\Delta(b^{(1)}, b^{(2)}) = \log_2 \prod_{i=k_f+k_g+1}^{n} m_i = \mathcal{N} - \log_2 f_1 - \log_2 g$. Hence if the code parameters are $\mathcal{K}_f = \log f_1 + 1, \mathcal{K}_g = \log_2 g$, we have $\Delta(b^{(1)}, b^{(2)}) = \mathcal{N} - \mathcal{K}_f - \mathcal{K}_g + 1$ and the minimum distance $\delta$ is comprised between $\mathcal{N} - \mathcal{K}_f - \mathcal{K}_g$ and $\mathcal{N} - \mathcal{K}_f - \mathcal{K}_g + 1$. Note that this only holds for a specific choice of $\mathcal{K}_f, \mathcal{K}_g$.

**Corollary 2.5.1**

*Let $\mathcal{K}_f, \mathcal{K}_g, \mathcal{N} \in \mathbb{R}_{>0}$ such that $d = \mathcal{N} - \mathcal{K}_f - \mathcal{K}_g > 0$. Then for any vector $(b_1, \ldots, b_n) \in \overline{\mathbb{Z}_{m_1}} \times \cdots \times \overline{\mathbb{Z}_{m_n}}$, there exists at most one pair of integers $f, g \in \mathbb{Z} \times \mathbb{Z}_{>0}$, with $\log_2 |f| \leq \mathcal{K}_f - 1$, $\log_2 g \leq \mathcal{K}_g$ such that $\sum_{i\in\bar{S}} \log_2 m_i < \frac{d}{2}$ where $\bar{S} = \{i : (b_i \neq \infty \text{ and } f \neq b_i g \mod m_i) \text{ or } (b_i = \infty \text{ and } g \neq 0 \mod m_i)\}$.*

#### Unique decoding of rational number codes

**Lemma 2.5.4**

*With $d = \mathcal{N} - \mathcal{K}_f - \mathcal{K}_g > 0$, if $\log_2 \Lambda = \sum_{i:r_i\neq c_i} \log_2 m_i < \frac{d}{2}$, the pair $(F = f\Lambda, G = \bar{g}\bar{\Lambda})$ is a minimal solution to the problem WeakRatNumRec $(\bar{\Pi}, \bar{h}, 2^{\mathcal{K}_f - 1 + \frac{d}{2}})$.*

The proof is the same as for lemma 2.3.1, and relies the key equation:

$$\underbrace{f\Lambda}_{F} \equiv \underbrace{\bar{g}\bar{\Lambda}}_{G}\ \bar{h} \pmod{\Pi/P_\infty}. \tag{2.15}$$

As a direct consequence of lemmas 2.5.2 and 2.5.4, we obtain:

**Corollary 2.5.2**

*Let* $\bar{h} = CRT((r_i P_\infty \mod m_i, m_i)_{i:r_i \neq \infty})$. *The extended Euclidean algorithm run on* $f_0 = \bar{\Pi} = \prod_{i:b_i \neq \infty} m_i$ *and* $f_1 = \bar{h}$ *and terminated at iteration* $j$ *when* $\log_2 |f_j| \leq \mathcal{K}_f - 1 + \frac{d}{2} < \log_2 |f_{j-1}|$ *returns* $f_j = f\Lambda, g_j = \bar{g}\bar{\Lambda}$.

Consequently Algorithm 2 can be directly adapted to decode rational number codes.

### Case of non-prime moduli

Note that for definition 2.5.1, we assumed that the moduli $m_i$ were prime numbers and not just pairwise co-prime, a sufficient condition for the Chinese remainder theorem. This is due to the fact that a pole in $m_i$ does no longer necessarily implies that $m_i$ divides $g$, but only that some factor of $m_i$ does. Hence the congruence (2.15) no longer holds in general. Still, a slight change in the key equation makes the decoding possible.

As previously, we define $S = \{i : r_i \neq c_i\}$ and $\Lambda_\infty = \prod_{i \in S : r_i = \infty} m_i, \quad G_\infty = \prod_{i \notin S : r_i = \infty} m_i$. First, as $G_\infty$ does not necessarily divides $g$ we set $\Gamma_\infty = \mathrm{GCD}(g, G_\infty)$, $Q_\infty = G_\infty / \Gamma_\infty$ and $\bar{g} = g/\Gamma_\infty$. Second, for a non-pole error position $i$ where $r_i - c_i \neq 0 \mod m_i$, the congruence might still be correct modulo some factors of $m_i$. Hence the minimal error locator can be a factor of $m_i$: $\bar{\Lambda} = m_i / \mathrm{GCD}(b_i - ai, m_i)$. Hence we define the agreement $A$ as

$$A = \gcd(fP_\infty - \bar{h}g, \Pi / P_\infty)$$

and $\bar{\Lambda} = \Pi/(P_\infty A)$. Lastly let $\Lambda = \bar{\Lambda}\Lambda_\infty$. Then the congruence (2.15) becomes:

$$\underbrace{Q_\infty f \Lambda}_{F} \equiv \underbrace{\bar{g}\bar{\Lambda}}_{G}\ \bar{h} \pmod{\Pi / P_\infty}, \tag{2.16}$$

since $Q_\infty f\Lambda - \bar{g}\bar{\Lambda}\bar{h} = \bar{\Lambda}\frac{P_\infty f - g\bar{h}}{\Gamma_\infty} = \frac{\Pi}{P_\infty}\frac{P_\infty f - g\bar{h}}{A\Gamma_\infty}$ and both $A$ and $\Gamma_\infty$ divide $P_\infty f - g\bar{h}$ and are co-prime.

One shows that the pair $(F, G)$ is the minimal solution to the weak rational reconstruction problem WeakRatNumRec $(\Pi/P_\infty, \bar{h}, 2^{\mathcal{K}_f - 1 + \frac{d}{2}})$ as long as the weight of the error impact is less than $d'/2$ where $d' = d - 2\mathcal{N}_\infty = \mathcal{N} - \mathcal{K}_f - \mathcal{K}_g - 2\mathcal{N}_\infty$.

## 2.6 Sparse polynomial evaluation codes

Model fitting of natural phenomena often uses an additional constraint, that the model has to be sparse. This has motivated a wide range of research including compressive sensing [CT06], and sparse interpolation of polynomials [Pro95; BT88; KLW90; GK93; GS09; GR10].

Most algorithms for the latter problem rely on the connection made in Blahut's Theorem (Theorem 2.1.1 [Bla83; MS88]) between linear complexity and sparsity. This connection is made effective by Ben-Or/Tiwari's algorithm [BT88], based on Berlekamp/Massey algorithm [Mas69].

### The Ben-Or/Tiwari Algorithm

We recall the Ben-Or/Tiwari [BT88] algorithm in the setting of univariate sparse polynomial interpolation. Let $f$ be a univariate polynomial with $t$ terms, $m_j$ and let $c_j$ the corresponding non-zero coefficients:

$$f(x) = \sum_{j=1}^{t} c_j x^{e_j} = \sum_{j=1}^{t} c_j m_j \neq 0, e_j \in \mathbb{Z}.$$

**Theorem 2.6.1 (*Ben-Or/Tiwari [BT88]*)**

*Let* $b_j = \alpha^{e_j}$, *where* $\alpha \in \mathsf{K}$ *has multiplicative order greater than* $\deg f = \max_j e_j$, *let* $a_i = f(\alpha^i) = \sum_{j=1}^{t} c_j b_j^i$, *and let* $\Lambda(X) = \prod_{j=1}^{t}(X - b_j) = X^t + \lambda_{t-1}X^{t-1} + \cdots + \lambda_0$. *The sequence* $(a_0, a_1, \ldots)$ *is linearly generated by the minimal polynomial* $\Lambda(z)$.

The Ben-Or/Tiwari algorithm then proceeds in the four following steps:

By Blahut's Theorem 2.1.1, the sequence $(a_i)_{i \geq 0}$ has linear complexity $t$, hence only $2t$ coefficients suffice for the Berlekamp/Massey algorithm to recover the minimal polynomial $\Lambda$. In the presence of errors in some of the evaluations, this fails.

**Algorithm 3** Ben-Or & Tiwari's algorithm

**Require:** $(a_0, a_1, \dots)$ a sequence of elements of $\mathsf{K}$ and $\alpha \in \mathsf{K}$
**Ensure:** $f(X) = \sum_{j=1}^{t} c_j X^{e_j}$ such that $a_i = f(\alpha^i)$
1: Find the minimal generating polynomial $\Lambda$ for $(a_0, a_1, \dots)$, using the Berlekamp/Massey algorithm.
2: Compute the roots $b_j$ of $\Lambda$, using univariate polynomial factorization.
3: Recover the exponents $e_j$ of $f$, by repeatedly dividing $b_j$ by $\alpha$.
4: Recover the coefficients $c_j$ of $f$, by solving the transposed $t \times t$ Vandermonde system

$$\begin{bmatrix} 1 & 1 & \dots & 1 \\ b_1 & b_2 & \dots & b_t \\ \vdots & \vdots & \ddots & \vdots \\ b_1^{t-1} & b_2^{t-1} & \dots & b_t^{t-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_t \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{t-1} \end{bmatrix}.$$

### 2.6.1 Sparse polynomial evaluation codes

We introduced in [R-CKP12] the problem of sparse polynomial interpolation with errors in the evaluations, and further develop it from a coding theoretic point of view in [R-KP14]. This can be viewed as a sparse version of the Reed-Solomon decoding problem or a fault tolerant instance of the sparse interpolation problem.

**Definition 2.6.1**

*Let $\mathsf{K}$ be a field, $0 < n \leq m$, two integers and let $x_0, \dots, x_{n-1}$ be $n$ distinct elements of $\mathsf{K}$. A sparse polynomial evaluation code of parameters $(n, T)$ over $\mathsf{K}$ is defined as the set*

$$\mathcal{C}(n, T) = \left\{ Ev_{(x_1,\dots,x_n)}(f) : f \in \mathsf{K}[X] \text{ is } t\text{-sparse with } t \leq T \text{ and } \deg f < m \right\}$$

In order to benefit from Ben-Or/Tiwari algorithm for error free interpolation, we will consider the special case where the evaluation points are consecutive powers of a primitive $m$-th root of unity $\alpha \in \mathsf{K}$: $x_i = \alpha^i$. Unfortunately, in this context, the minimum distance of such codes is very low. Provided that $2T$ divides $m$, we can state the following theorem (see [R-KP14, Theorem 3] for a proof).

**Theorem 2.6.2**

*If $\alpha \in \mathsf{K}$ is a primitive $m$-th root of unity, $x_i = \alpha^i, i \in \{0 \dots n-1\}$ and $2T$ divides $m$, then the corresponding $(n, T)$-sparse polynomial evaluation code has minimum distance $\delta = \lfloor \frac{n}{2T} \rfloor$.*

### 2.6.2 Unique decoding

There exists an algorithm that does unique decoding of such codes up to half the minimum distance: the Majority Rule Berlekamp/Massey algorithm [R-CKP12]. It simply consists in running a Berlekamp/Massey algorithm on each of the $\lfloor \frac{n}{2T} \rfloor$ contiguous sub-sequences

$$x^{(i)} = (x_{2Ti}, \dots, x_{2T(i+1)-1})$$

of the received word $x$. With $E \leq \lfloor \frac{\delta-1}{2} \rfloor \leq \lfloor \frac{n}{2T} \rfloor / 2$ errors in the received word, the generator occurring with majority will be the correct one for step 1 of Algorithm 3. Still, this knowledge of $\Lambda$ does not solve the decoding problem completely: one also needs to isolate a contiguous error-free sub-sequence of length $2T$ in order to

1. recover the sparse polynomial $f$ by running Ben-Or/Tiwari's algorithm on it,
2. recover the clean sequence of evaluations (the unique code-word at distance $\leq \lfloor \frac{\delta-1}{2} \rfloor$).

The difficulty is that not every segment of length $2T$ that contributed to the majority (with generator $\Lambda$) is error-free. Errors may occur within a pattern that fits with the connecting polynomial, we will call them *deceptive segments*.

Given a sequence $s$ of length $\geq t$, assumed to be error-free, the second task can be done by applying the generator iteratively, as a linear feedback shift register, to produce the next values in the sequence. The values preceding $s$ can also be produced by running the generator in reverse mode: $a_i = \frac{1}{\lambda_0}(-\lambda_1 a_{i+1} - \cdots - \lambda_{t-1}a_{i+t-1} + a_{i+t})$ since $\lambda_0 \neq 0$ as it is a power of $\alpha$. We call this algorithm the *sequence clean-up*. Note that upon fixing the sequence with this algorithm, one can keep track of the number of corrections made to the received sequence. Whenever this number of corrections gets above the maximal number of errors that can be corrected (e.g. half the minimum distance for the unique decoding), one then concludes that the sequence used as a seed is necessarily deceptive. More precisely, we have the following Lemma.

**Lemma 2.6.1**

*If the received word of length $n$ contains up to $E$ errors, then deceptive segments will be exposed in the sequence-cleanup as soon as $n \geq T(2E+1)$.*

See [R-CKP12, Theorem 4] for a proof and more details on the sequence-clean-up algorithm. Note that the unique decoding condition $E \leq \lfloor \frac{\delta-1}{2} \rfloor$ implies $n \geq 2T(2E+1)$, which proves that the above decoding algorithm works, as every deceptive segment will be exposed by the sequence clean-up.

This decoding requires $\lfloor n/(2T) \rfloor$ executions of Berlekamp/Massey algorithm.

**Remark 2.6.1**

*In summary, we proposed to make sparse interpolation resilient to errors by simply replacing the first step in Ben-Or/Tiwari's sparse interpolation algorithm, the Berlekamp/Massey algorithm, by a fault tolerant Berlekamp/Massey algorithm. The latter requires a surprisingly large amount of values to recover a unique generator, however the fact that the worst case ambiguous sequence for this problem actually corresponds to the evaluations of two sparse polynomials, implies that this approach is optimal: in the worst case, some codes can not be uniquely decoded above the limit of the fault tolerant Berlekamp/Massey algorithm. However, this worst case situation happens in the special case where the evaluation points are in geometric progression, and where the order of $x_1 = \alpha$ is divisible by $2T$. If the use of the Ben-Or/Tiwari's algorithm imposes evaluation points in geometric progression, the divisibility condition is in general not required, which leaves hope for a better minimum distance, and possibly a better unique decoding algorithm in these cases.*

### 2.6.3 List decoding

Following the same idea as for the majority rule Berlekamp/Massey algorithm, one remarks that if $n \geq 2T(E+1)$, then necessarily one sub-sequence $x^{(i)}$ has to be clean of errors and the list of all $\lfloor \frac{n}{2T} \rfloor$ generators contains the correct one. This makes a trivial list decoding algorithm, decoding up to the minimum distance. Indeed, Lemma 2.6.1 still applies, and guaranties that clean segments can still be identified in order to seed the sequence clean-up and run Ben-Or/Tiwari's algorithm to recover the sparse polynomial $f$.

In order to further reduce the bound $n \geq 2T(E+1)$ (or equivalently increase the decoding radius above $\frac{n}{2T}$), we proposed in [R-KP14] to use more sub-sequences from the received word.

**Remark 2.6.2**

*Instead of partitioning the received word into $n/(2T)$ disjoint sub-vectors, one would hope to find more error-free sequences by considering all $n-2T+1$ sub-vectors of the form $(x_i, \ldots, x_{i+2T-1})$ for $i \in \{0 \ldots n-2T\}$. This will very likely allow to decode more errors in many cases (as will be illustrated in Figure 2.1), but the worst case configuration (see proof of [R-KP14, Theorem 3]) remains unchanged. Note that the majority rule based unique decoding still works under the same conditions: at most $2TE$ sub-sequences will contain an error, hence a majority of subsequences will be correct as soon as $4TE < n-2T+1$, which is $n \geq 2T(2E+1)$. In terms of complexity, the number of arithmetic operations required for both unique and list decoding algorithms in [R-CKP12] is $O(n^2)$ field operations ($n/(2T)$ runs of Berlekamp/Massey algorithm on sequences of length $2T$, and $O(n/(2T))$ calls to the sequence clean-up, each of which costs $O(nT)$). Now the above variant requires to inspect $n-2T$ sub-sequences instead of $n/(2T)$ and the complexity becomes $O(n^2T)$ (as $T = o(n)$).*

### Affine sub-sequences

Consider a received word $(a_0, \ldots, a_{n-1})$ of evaluations of a $t$-sparse polynomial $f(X) = \sum_{j=1}^{t} c_j X^{e_j}$, with $E$ errors. We proposed in [R-KP14] to consider all length $k$ sub-sequences in arithmetic progression:

$$(a_r, a_{r+s}, a_{r+2s}, \ldots, a_{r+(k-1)s}) \text{ where } r + (k-1)s < n,$$

which will be called affine index sub-sequences or more conveniently affine sub-sequences. In the remaining of the text, $k$ will denote the length of the sub-sequence. We will consider the general case where $k$ can be any positive integer, not necessarily even of the form $2T$ as required in Ben-Or/Tiwari's algorithm.

**Lemma 2.6.2**

*If* $\gcd(s, m) = 1$ *and* $k \geq 2t$*, then such a sub-sequence with no error is sufficient to recover* $f$*.*

**Proof.** Let $\beta = \alpha^s$ and $g(X) = f(X\alpha^r)$. Note that $\deg g = \deg f$ and $g$ is also $t$-sparse with the same monomial support as $f$. If $\gcd(s, m) = 1$ then $\text{order}(\beta) \geq m$. Then the sub-sequence $(a_r, a_{r+s}, a_{r+2s}, \ldots, a_{r+(k-1)s}) = (f(\alpha^r), f(\alpha^{r+s}), f(\alpha^{r+2s}), \ldots, f(\alpha^{r+(k-1)s})) = (g(\beta^0), g(\beta^1), g(\beta^2), \ldots, g(\beta^{k-1}))$ is formed by evaluations of $g$ in $k$ consecutive powers of an element $\beta$ of order greater than $m \geq \deg g$. One can therefore compute $g = \sum_{j=1}^{t} d_j X^{e_j}$ using Ben-Or/Tiwari's algorithm on this sub-sequence. The coefficients of $f$ are directly deduced from that of $g$: $c_j = d_j \alpha^{-re_j}$. □

**Example 2.6.1**

*Let* $t = 2$*, and consider a sequence of* $n = 9$ *evaluations* $(a_0, a_1, \ldots, a_8)$*. Then* $E = 1$ *is the maximal number of errors that the trivial list decoding presented above can decode as it requires that* $n \geq 2t(E+1)$*. Indeed if two errors occurred e.g. on elements* $a_3$ *and* $a_7$*, there is no contiguous sub-sequence of length* $2t = 4$ *free of error, thus making the latter decoding fail. Now consider the sub-sequence* $(a_0, a_2, a_4, a_6)$*. It is free of error and is formed by evaluations of* $f(z)$ *in the four consecutive powers of* $\beta = \alpha^2$*. Blahut/Ben-Or/Tiwari algorithm applied on this sequence will return* $g(X) = f(X^2)$*.*

This results in a new list decoding algorithm:

**Algorithm 4** List decoding of sparse polynomial evaluation codes

1: **for all** $s \in \{1 \ldots \lfloor \frac{n}{k} \rfloor\}$, $r \in \{0 \ldots n - (k-1)s - 1\}$ **do**
2: compute $\Lambda_{r,s}$ generating $(a_r, a_{r+s}, \ldots, a_{r+s(k-1)})$ with the Berlekamp/Massey algorithm
3: (Optional heuristic reducing the list size) run the sequence clean-up and discard $\Lambda_{r,s}$ if it can not generate the sequence with less than $E$ errors, for some bound $E$ on the number of errors.
4: apply Ben-Or/Tiwari's algorithm to recover the associated sparse polynomial $f_{r,s}$
5: **end for**
6: Return the list of the $f_{r,s}$.

Regarding the type of iteration chosen for the pair $(r, s)$, a first approach is to explore all sub-sequences for any value of $s \in \{1 \ldots \lfloor n/k \rfloor\}$ and $r \in \{0 \ldots n-(k-1)s-1\}$. This amounts to $O(n^2/k)$ sub-sequences. A second approach, applying Remark 2.6.2 considers all values for $s \in \{1 \ldots \lfloor \frac{n}{k} \rfloor\}$ but then for each $s$ only considers the disjoint sub-sequences with $s\frac{n}{ks} = n/k$ choices for $r$. This amounts to $O(n^2/k^2)$ sub-sequences. For each sub-sequence, corresponding to a pair $(s, r)$, Ben-Or/Tiwari algorithm is run in $O(k^2)$ (Berlekamp/Massey algorithm and solving the transpose Vandermonde system [Zip90]). The optional sequence clean-up heuristic adds an $O(nk)$ term. Overall, the complexity of the second approach amounts to the same $O(n^2)$ estimate, as the trivial list decoding. The additional overhead of $O(n^3/k)$ when the sequence clean-up heuristic is used also remains identical. In the first approach, ignoring Remark 2.6.2, these complexity estimates are multiplied by a factor $k$.

We report in Figure 2.1 the average rate of decoding success for our implementation of Algorithm 4. For each value of the pair $(E, k)$, the success rate is averaged over 10 000 samples, where the error locations are uniformly distributed. The two variants (disjoint subsequences or all possible sub-sequences) are compared.

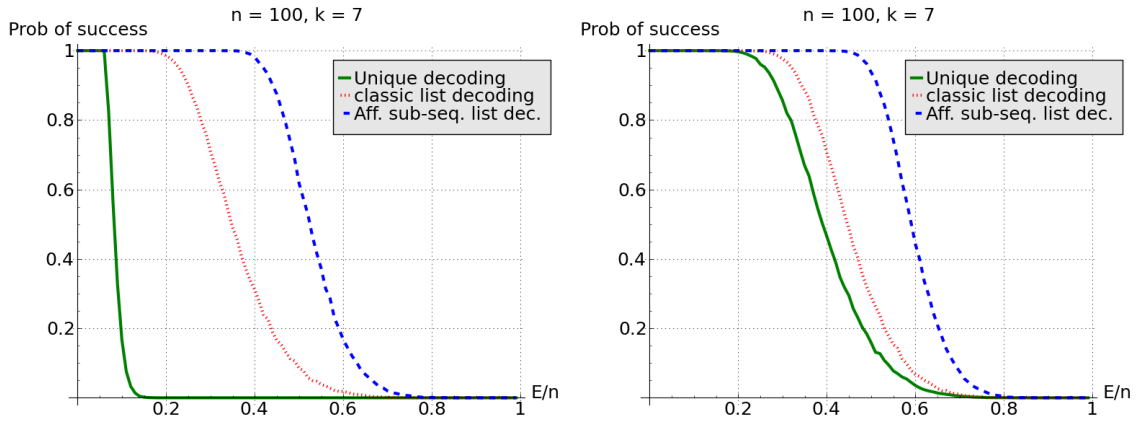



Figure 2.1: Success rate for unique, standard list decoding and affine sub-sequence list decoding. Disjoint sub-sequences only (left) or all sub-sequences (right) being considered.

### Worst case decoding radius

The worst case decoding radius for the affine sub-sequence based list-decoding is the maximal number $E(n,k)$ of errors that would always leave an affine sub-sequence of length $k = 2t$ free of error, for any error pattern. Equivalently, one can search for the minimal length $n(k,E)$ such that any pattern with up to $E$ errors will always leave a length $k = 2t$ affine sub-sequence free of errors.

**Problem 2.6.1**

*Given $k, E \in \mathbb{Z}_{>0}$, find the minimal $n \in \mathbb{Z}_{>0}$, denoted by $n(k,E)$ such that no sub-set of $E$ elements of $\{1 \dots n\}$ intersects all of its length $k$ affine sequences.*

In some cases, as shown in Example 2.6.2, the affine sub-sequence technique does not help improving the former bound $n \geq k(E+1)$, not even by saving a single evaluation point.

**Example 2.6.2**

*For $k = 5$ and $E = 3$, the worst case configuration (errors on $a_4, a_9$ and $a_{14}$) requires $n(5,3) = 20 = k(E+1)$ values to find $k$ consecutive clean values.*

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $a_0$ | $a_1$ | $a_2$ | $a_3$ | $\mathbf{a_4}$ | $a_5$ | $a_6$ | $a_7$ | $a_8$ | $\mathbf{a_9}$ | … | $\mathbf{a_{14}}$ | $a_{15}$ | $a_{16}$ | $a_{17}$ | $a_{18}$ | $a_{19}$ | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| $a_0$ | $a_2$ | $\mathbf{a_4}$ | $a_6$ | $a_8$ | $a_{10}$ | $a_{12}$ | $\mathbf{a_{14}}$ | $a_{16}$ | $a_{18}$ |
| $a_1$ | $a_3$ | $a_5$ | $a_7$ | $\mathbf{a_9}$ | $a_{11}$ | $a_{13}$ | $a_{15}$ | $a_{17}$ | $a_{19}$ |

| | | | | | | |
|---|---|---|---|---|---|---|
| $a_0$ | $a_3$ | $a_6$ | $\mathbf{a_9}$ | $a_{12}$ | $a_{15}$ | $a_{18}$ |
| $a_1$ | $\mathbf{a_4}$ | $a_7$ | $a_{10}$ | $a_{13}$ | $a_{16}$ | $a_{19}$ |
| $a_2$ | $a_5$ | $a_8$ | $a_{11}$ | $\mathbf{a_{14}}$ | $a_{17}$ | |

| | | | | |
|---|---|---|---|---|
| $a_0$ | $\mathbf{a_4}$ | $a_8$ | $a_{12}$ | $a_{16}$ |
| $a_1$ | $a_5$ | $\mathbf{a_9}$ | $a_{13}$ | $a_{17}$ |
| $a_2$ | $a_6$ | $a_{10}$ | $\mathbf{a_{14}}$ | $a_{18}$ |
| $a_3$ | $a_7$ | $a_{11}$ | $a_{15}$ | $a_{19}$ |

| | | | | |
|---|---|---|---|---|
| $a_0$ | $a_5$ | $a_{10}$ | $a_{15}$ | $a_{20}$ |
| $a_1$ | $a_6$ | $a_{11}$ | $a_{16}$ | |
| $a_2$ | $a_7$ | $a_{12}$ | $a_{17}$ | |
| $a_3$ | $a_8$ | $a_{13}$ | $a_{18}$ | |
| $\mathbf{a_4}$ | $\mathbf{a_9}$ | $\mathbf{a_{14}}$ | $\mathbf{a_{19}}$ | |

*But for $E = 4$, one verifies that $n = 21$ suffices to ensure that a length 5 sub-sequence will always be found. In particular, in the previous configuration, placing the fourth error on $e_{19}$ leaves the sub-sequence $(a_0, a_5, a_{10}, a_{15}, a_{20})$ untouched.*

We report in Figure 2.2 the value of $n(k,E)$ for all typically small values of $E$ and $k$ computed by exhaustive search.

Lemma 2.6.3 states more precisely when affine sub-sequence improves over the trivial list decoder.

**Lemma 2.6.3**

*$n(k,E) \leq k(E+1)$, with equality $n(k,E)$=$k(E$+$1)$ if and only if $E + 2 \leq g$, where $g$ denotes the smallest prime factor of $k$.*

| $k\backslash E$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 3 | 6 | 7 | 8 | 10 | 12 | 15 | 16 | 17 | 18 | 19 | 21 | 22 | 23 | 25 | 27 |
| 4 | 4 | 7 | 11 | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 29 | 31 | 32 | 35 | 36 |
| 5 | 5 | 10 | 15 | 20 | 21 | 22 | 23 | 26 | 30 | 32 | 35 | 40 | 45 | 46 | 47 | 48 |
| 6 | 6 | 11 | 16 | 21 | 27 | 28 | 30 | 31 | 34 | 38 | 42 | 43 | 47 | 52 | | |
| 7 | 7 | 14 | 21 | 28 | 35 | 42 | 43 | 44 | 45 | 47 | 49 | 54 | 58 | | | |
| 8 | 8 | 15 | 22 | 29 | 36 | 43 | 51 | 52 | 53 | 55 | 57 | 60 | | | | |
| 9 | 9 | 18 | 25 | 32 | 39 | 46 | 53 | 58 | 59 | 62 | 66 | 72 | | | | |
| 10 | 10 | 19 | 29 | 34 | 41 | 48 | 55 | 62 | 65 | 69 | 74 | | | | | |
| 11 | 11 | 22 | 33 | 44 | 55 | 66 | 77 | 88 | 99 | 110 | 111 | 112 | | | | |
| 12 | 12 | 23 | 34 | 45 | 56 | 67 | 78 | 89 | 100 | 111 | 123 | 124 | | | | |
| 13 | 13 | 26 | 39 | 52 | 65 | 78 | 91 | 104 | 117 | 130 | 143 | 156 | | | | |

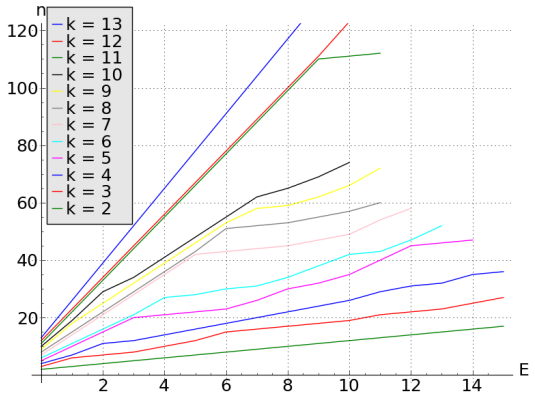



Figure 2.2: Minimal length $n(k, E)$ such that any pattern of up to $E$ errors leaves a length $k$ affine sub-sequence free of error.

In particular, this implies that for even $k = 2t$, the affine sub-sequence list decoder performs always better than the trivial list decoder. Refer to [R-KP14] for a proof.

Noting that $n(k, E)$ is monotonically increasing, and applying Bertrand's Postulate [Che52; Ram19] yields a lower bound on $n(k, E)$ for small number of errors.

**Corollary 2.6.1**

*For $k > 0$, and $E < \frac{k+1}{2}$, then $n(k, E) > \frac{k+1}{2}(E + 1) + 1$.*

In other terms, the improvement is not greater than roughly a factor of two whenever the number of errors is less than half the length $k$ of the sequence.

However, for larger values of $E$, Figure 2.2 suggests that $n(k, E)$ increases at a much lower rate. We will now focus on its asymptotic behavior by guessing a worst case error pattern that would give a lower bound estimate on $n(k, E)$.

A first approach is inspired by the worst case error pattern of the trivial list decoding ($S = \{k-1, 2k-1, 3k-1, \dots\}$ as shown in Example 2.6.2). Looking at the fourth table in this example, one sees that each of the 4 values $a_{20}, a_{21}, a_{22}, a_{23}$ will need to be erroneous in order to avoid having a length 5 affine sub-sequence free of error with $s = 5$.

By recursively applying this approach one defines the error support

$$S_n = \{i \in \{0 \dots n-1\} \text{ which base } k \text{ expansion contains no } k-1 \text{ digit}\},$$

and shows (see [R-KP14, Lemma 3]) that for $k$ prime and $n_i = k^i - k^{i-1} - k^{i-2} - \cdots - 1 = \frac{(k-2)k^i+1}{k-1}$ the set $S_{n_i}$ intersects all length $k$ affine sub-sequences of $\{0 \dots n_i - 1\}$. Consequently, as $|S_{n_i}| = n_i - (k-1)^i$ we have

$$n(k, n_i - (k-1)^i) \geq n_i. \tag{2.17}$$

**Remark 2.6.3**

*Problem 2.6.1 is in fact closely related to th eproblem, proposed by Erdős and Turán [ET36], of finding the largest sub-sequence of $\{1, \dots, n\}$ not containing $k$ terms in arithmetic progression. Let $r(k, n)$ denote the size of such a largest sub-sequence. If $n \geq r(k, n) + E + 1$, a subset of $E$ errors can not suffice to intersect all arithmetic progressions of $k$ terms. Hence $n(k, E) = \min\{n : n - r(k, n) \geq E+1\}$. Noting that $r(k, n) \leq r(k, n + 1) \leq r(k, n) + 1$, we deduce that for given $k$ and $E$, there always exists a $n^*$ such that $n^* - r(k, n^*) = E + 1$ and consequently $n(k, E) = n^* = r(k, n^*) + E + 1$. Erdős and Turán [ET36] first conjectured that for all $k \geq 3$, $\lim_{n\to\infty} r(k, n)/n = 0$ which was later proven by Szemeredi [Sze75]. In particular the above construction of a bad error support $S_{n_i}$ for $k$ prime is similar to a construction by Szerekes (see [ET36; Wag72]). When $k$ is prime, this construction yields to the estimate*

$$r\left(k, \frac{(k-2)k^i + 1}{k-1}\right) \geq (k-1)^i, \tag{2.18}$$

*which is equivalent to equation (2.17). Szerekes conjectured that equality held in (2.18) which was disproved by Salem and Spencer [SS50], and consequently, the proposed support $S_n$ is not a worst case one.*

The error correction rate of the affine sub-sequence list decoding is therefore directly related to the growth of the ratio $r(k,n)/n$ which is a core problem in additive combinatorics.

$$\frac{E}{n} = 1 - \frac{r(k,n)}{n} - \frac{1}{n}. \tag{2.19}$$

Szemeredi's theorem states that arithmetic progressions are dense, i.e. an asymptotically large number of errors is necessary to intersect all of them and rule out any list decoding possibility. Now there is unfortunately no known expression of $r(k,n)/n$ as a function of the information rate $k/n$, to the best of our knowledge. We therefore propose in Equation (2.20) the bounds on the maximal correction rate.

$$1 - \frac{1}{(\log\log n)^{1/2^{2^{k+9}}}} - \frac{1}{n} \leq \frac{E}{n} \leq \frac{n}{k-2}\log_k \frac{n}{k-2} \tag{2.20}$$

The upper bound is a direct consequence of Equations (2.17). (see [R-KP14]). This shows that, in the worst case, the improvement of the affine sub-sequence technique to the correction capacity, compared to the previous list decoding ($\frac{n}{k} - 1$) is essentially no larger than a logarithmic factor.

Obtaining a sharp lower bound on $E/n$ is much harder. In [Rot53], Roth proved $r(3,n) \leq \frac{c}{\log\log n}$, leading to $\frac{E}{n} \geq 1 - \frac{c}{\log\log n} - \frac{1}{n}$. For an arbitrary $k$ the best known bound, given in Equation (2.20), is that of Gowers [Gow01].

### 2.6.4 Towards better minimum distances

Despite the pessimistic minimal distance obtained in section 2.6.1 and the difficult analysis of the list decoding worst case correction capacity in section 2.6.3, the sparse evaluation codes seem to perform much better in practice, as illustrated in Figure 2.1. We now provide further evidence that similar sparse evaluation codes can be built with much better minimum distances, by considering the case where the base field has characteristic zero.

**Theorem 2.6.3**

*Consider $n$ distinct positive real numbers $x_1, \ldots, x_n > 0$. The sparse polynomial evaluation code*

$$\mathcal{C}(n,T) = \left\{ Ev_{(x_1,\ldots,x_n)}(f) : f \in \mathbb{R}[X] \text{ is } t\text{-sparse with } t \leq T \right\}$$

*has minimum distance $\delta = n - 2T + 1$.*

**Proof.** Consider the code words $(f(x_1), \ldots, f(x_n))$ and $(g(x_1), \ldots, g(x_n))$ for a $t_f$-sparse polynomial $f$ and a $t_g$-sparse polynomial $g$, with $t_f, t_g \leq T$, at Hamming distance $\leq n - 2T$. Then the polynomial $f - g$ has sparsity $\leq 2T$, and vanishes in least $2T$ distinct positive reals $\xi_i$. By Descartes's rule of sign $f - g = 0$. □

**Corollary 2.6.2**

*Suppose we have, for a $t_f \leq T$ sparse real polynomial $f$, values $f(x_i) + \epsilon_i$ for $2T + 2E$ distinct positive real numbers $\xi_i > 0$, where $e \leq E$ of those values can be erroneous: $\epsilon_i \neq 0$. If a $t_g \leq T$ sparse real polynomial $g$ interpolates any $2T + E$ of the $f(x_i) + \epsilon_i$, then $g = f$.*

So $f$ can be uniquely recovered from $2T + 2E$ values with $e \leq E$ errors. This result correspond to the situation with Reed-Solomon codes: the number of evaluations to ensure unique decoding is that for the error-free case with an additive term equal to twice the number of errors.

**Remark 2.6.4**

*We do not have an efficient (polynomial time) decoder up to half this minimum distance. However, notice that when choosing the evaluation points $\xi_i = \alpha^i$ for some $\alpha \in \mathbb{R}_{>0}\backslash\{1\}$, the list decoder presented in Section 2.6.3 can be used. Interestingly, it turns out to be a unique decoder as long as*

*an affine sub-sequence free of error exists. Indeed, the list of candidates can be sieved by removing the polynomials which evaluations differ by more than $\delta/2$ positions with the received word. Finally the minimum distance of Theorem 2.6.3 ensures that only one code-word lies within less than $\delta/2$ modifications of the received word, hence the decoding is unique.*

Another result improving the minimum distance can also be obtained when sampling in primitive elements of co-prime orders in the complex unit circle.

**Theorem 2.6.4**

*Let $T, D$, and $n \geq k = 2T\frac{\log(D)}{\log(2T)}$ be given. Consider $n$ $p_i$-th roots of unity $\xi_i \neq 1$, where $2T < p_1 < p_2 < \cdots < p_n$, $p_i$ prime. The sparse polynomial evaluation code*

$$\mathcal{C}(n,T) = \left\{Ev_{(\xi_1,\ldots,\xi_n)}(f) : f \in \mathbb{Q}[X]_{\leq D} \text{ is } t\text{-sparse with } t \leq T\right\}$$

*has minimum distance*

$$\delta = n - k + 1 = n - 2T\frac{\log(D)}{\log(2T)} + 1.$$

**Proof.** Consider the code-words $(f(\xi_1), \ldots, f(\xi_n))$ and $(g(\xi_1), \ldots, g(\xi_n))$ for a $t_f$-sparse polynomial $f$ and a $t_g$-sparse polynomial $g$, with $t_f, t_g \leq T$, at Hamming distance $\leq n - k$. Then $(f-g)(\xi_j)$ vanishes for at least $k$ of the $\xi_j$, say for those sub-scripted $j \in J$.

Let $0 \leq e_1 < e_2 < \cdots < e_s$ be the term exponents in $f - g$, with $s \leq 2T$. Suppose $f - g \neq 0$. Consider $M = (e_s - e_1)(e_s - e_2)\cdots(e_s - e_{s-1})$. Since $M \leq D^{2T}$ and $\prod_{j\in J} p_j > (2T)^k \geq D^{2T}$, not all $p_j$ for $j \in J$ can divide $M$. Let $\ell \in J$ with $M \not\equiv 0 \pmod{p_\ell}$. Then the term $x^{e_t \bmod p_\ell}$ can not cancel out in a sum with any other $x^{e_i \bmod p_\ell}$ in $h(X) = (f(X) - g(X) \bmod (X^{p_\ell} - 1))$, and therefore the polynomial $h(X)$ is not zero; $h$ has at most $2T$ terms, and $h(\xi_\ell) = 0$. This means that $h(X)$ and $\Psi_\ell(X) = 1 + x + \cdots + x^{p_\ell - 1}$ have a common GCD. Because $\Psi_\ell$ is irreducible over $\mathbb{Q}$, and since $\deg(h) \leq p_\ell - 1$, that GCD is $\Psi_\ell$. So $h$ is a scalar multiple of $\Psi_\ell$ and has $p_\ell > 2T$ non-zero terms, a contradiction. □

**Corollary 2.6.3**

*Let $T, D, E$ be given and let the integer $k \geq 2T\log(D)/\log(2T)$. Suppose we have, for a $t_f$-sparse polynomial $f \in \mathbb{Q}[X]$, where $t_f \leq T$ and $\deg(f) \leq D$, the values $f(\xi_i)$ for $k + 2E$ $p_i$-th roots of unity $\xi_i \neq 1$, where $2T < p_1 < p_2 < \cdots < p_{N+2E}$, $p_i$ prime. Again $e \leq E$ of those values can be erroneous $f(\xi_i) + \epsilon_i$. If a $t_g$-sparse polynomial $g \in \mathbb{Q}[X]$ with $t_g \leq T$ and $\deg(g) \leq D$ interpolates any $k + E$ of the $f(\xi_i) + \epsilon_i$, then $g = f$.*

In order to transfer this result over positive characteristic, we recall in Theorem 2.6.5 a result on the facorization of cyclotomic polynomials over a finite field.

**Theorem 2.6.5 (*[LN97, Theorem 2.47]*)**

*Let $\mathsf{K} = \mathbb{F}_q$ and $n$ a positive integer co-prime with $q$. The $n$-th cyclotomic polynomial $Q_n \in \mathsf{K}[X]$ factors into $\Phi(n)/d$ distinct monic irreducible polynomials in $\mathsf{K}[X]$ of same degree $d$, where $d$ is the multiplicative order of $q$ modulo $n$.*

For a field $\mathsf{K}$ and a positive integer $n$, we will denote by $\mathsf{K}^{(n)}$ the splitting field of $X^n - 1$ over $\mathsf{K}$, called the $n$-th cyclotomic field over $\mathsf{K}$.

**Corollary 2.6.4**

*Let $\mathsf{K} = \mathbb{F}_q, T, D$, and $n \geq k = 2T\frac{\log(D)}{\log(2T)}$ be given. Let $2T < p_1 < p_2 < \cdots < p_n$, be $n$ prime numbers such that $q$ is a primitive root for $p_i$ for all $i \in \{1 \ldots n\}$. For all $i \in \{1 \ldots n\}$ let $\xi_i \in \mathsf{K}^{(p_i)} \setminus \{1\}$ be a $p_i$-th roots of unity.*

*The sparse polynomial evaluation code*

$$\mathcal{C}(n,T) = \left\{Ev_{(\xi_1,\ldots,\xi_n)}(f) : f \in \mathbb{F}_q[X]_{\leq D} \text{ is } t\text{-sparse with } t \leq T\right\} \subseteq \mathsf{K}^{(p_1)} \times \cdots \times \mathsf{K}^{(p_n)}$$

*has minimum distance*

$$\delta = n - k + 1 = n - 2T\frac{\log(D)}{\log(2T)} + 1.$$

**Proof.** As a primitive root for $p_i$, $q$ has multiplicative order $p_i - 1 = \Phi(p_i)$ modulo $p_i$ and by Theorem 2.6.5, the $p_i$-th cyclotomic polynomial $\Psi_i = Q_{p_i}$ is irreducible. Consequently the proof of Theorem 2.6.4 still holds over $\mathbb{F}_q$. □

The ability to find $n$ primes $p_1, \ldots, p_n$ such that $q$ generates $(\mathbb{Z}/p_i\mathbb{Z})^* \ \forall i \in \{1 \ldots n\}$ is strongly related to Artin's primitive root conjecture [Mur88]. This conjecture states that for any integer $a$ other than $1, -1$ or a perfect square, there exist infinitely many primes $p$ for which $a$ is a primitive root. More precisely, the conjecture claims that, if $a$ is not a perfect power and the square-free part of $a$ is not congruent to 1 modulo 4, the density of such primes is independant of $a$, and equals Artin's constant $C_{\text{Artin}} = 0.3739558136\ldots$. This conjecture has been proven under the generalized Riemann hypothesis [Hoo67], or unconditionally for infinitely many cases [GM84]. Laslty, Heath-Brown proved that the conjecture was true except for at most two prime numbers [Hea86] (not constructively).

Again this result is mostly of theoretical interest as first, these codes live in a sophisticated algebraic structure, which implies a strong overhead for the arithmetic cost, and second as we do not know any efficient decoding algorithm reaching this correction capacity.

### What have we learned?

1. *Parallelizations based on evaluation-interpolation schemes can be made fault tolerant through the use of evaluation based error correcting codes. The Reed-Solomon codes, solving the problem of polynomial interpolation with errors can be generalized to rational functions.*
2. *Parameter oblivious decoding allow to adaptively increase the decoding capacity taking advantage of the effective amount of redundancy.*
3. *Rational function codes have a minimal distance equal to the number of evaluations minus the number of information symbols which smoothly generalizes that of Reed-Solomon codes, even when evaluations at poles are considered.*
4. *Interleaved rational function codes naturally arise in the context of parallel linear system solving with a polynomial matrix. Considering the collaborative decoding of such codes increases the decoding capacity. However this capacity is no longer oblivious to the presence of poles (evaluations at which the system is singular).*
5. *These results naturally generalize to rational numbers via CRT codes.*
6. *An $(n, T)$-sparse polynomial evaluation code has, in the worst case, a minimum distance of $\lfloor \frac{n}{2T} \rfloor$, when the evaluation points are in geometric progression, which is required to apply Ben-Or/Tiwari's interpolation algorithm.*
7. *List decoding of such codes improves the decoding capacity by a factor of at least 2, but no greater than $\log(n/T)$ in theory, but prove to be efficient in practice.*
8. *Better minimum distances of about $n - 2T + 1$ can be reached with other sets of evaluation points, but no practical decoding algorithm is known for these cases.*

# Perspectives

The landscape of computational exact linear algebra, though very diverse, is becoming clearer.

Matrix arithmetic over a field has reached a good maturity : theoretical complexities are fairly well understood, and the design of practical implementations now relies on strong general guidelines that have proven successful over the years.

There, the question of computing the characteristic polynomial by an $O(n^\omega)$ algorithm has been partially answered, but a deterministic algorithm with no restriction on the field size is still to be found. We remark that this problem in fact better fits to the framework of polynomial matrix algorithmic, where degree-dimension trade-offs prevail. This latter area, has recently seen much improvements [GJV03; Sto03; Zho12], as far as exponents of complexities are concerned. The next step is now to refine these complexities, improving the logarithmic factors: some arise from the use of fast polynomial arithmetic but others arising from the inner structure of the algorithm may be improved. Further work on leading constants and memory complexity should also follow, to produce efficient software implementations. The recent work of Giorgi and Lebreton [GL14] based on reductions to scalar matrix arithmetic, and relaxed degree-dimension tradeoffs is a successful building block on which to elaborate a whole set of library routines.

Applications of efficient implementations of polynomial matrix arithmetic to algebraic coding theory and more specifically list decoding through polynomial lattice reduction are currently under progress. Alternatively, the recent advances on the structured linear algebra complexities [BJS08] have, to our knowledge, not been put into practice so far. Their application to the most efficient list decoding techniques, shown in [Cho+14], is one out of many motivations to work on their efficient implementations.

Our experience with the parallelization of dense linear algebra over a finite field showed the relevance of task based parallelism languages able to handle efficiently both recursion and a large number of tasks. This is motivated by the fact that, costs being non associative, tasks need to be forked at the coarser grain, and then possibly recurse to finer grains in order to keep the task pool non-empty. By nature, exact linear algebra computations often produce tasks of heterogeneous and unpredictable size. Therefore parallel runtimes offering dynamic scheduling capabilities, based on work-stealing are of utmost importance. Dataflow task dependencies should be preferred to explicit synchronizations but further improvements of existing runtimes are necessary for offering recursive dataflow dependency resolution.

Whether the parallelization strategy that we proposed will scale to large instances, and be applicable to distributed computing need to be investigated. In particular, parallel algorithms with best depth but non-optimal work, could very well become paramount. The use of evaluation-interpolation schemes is a typical example, that produces embarrassingly parallel algorithms well suited for large scale distributed computations. Their ability to offer a natural setting for fault tolerance as we demonstrated in the dense and sparse setting is a major advantage.

Applying the fault tolerant interpolations techniques, over floating point real or complex numbers is another great challenge. Interestingly, the convergence with the symbolic-numeric sparse interpolation of [KLL00; KYZ07], revealed a nice stability for the computation of the error support, in the presence of noise [R-CKP12]. Further work has been recently done for sparse rational fractions [KY13; KY14] or interleaved dense rational fractions [BK14]. Still, many improvements should be done, as for example concerning the complexity of decoding, or the correction capacities, by adapting numerical Padé approximation techniques. The approach has also to be generalized to other problems than interpolation.

The sparse evaluation codes that we introduced are difficult to apprehend: the configuration enabling efficient decoding algorithms imposes bad worst case error correction capacities whereas much better

minimum distances can be attained in settings where no practicable sparse interpolation algorithm is known at the moment. The connections of this problem to learning problems (in particular ring learning parity with noise [Hey+12]), and the potential applications to lattice based cryptography is also to be investigated further.

# Index

# Publication list

## Book chapter


[B-Cas+13] A. Casamayou, G. Connan, T. Dumont, L. Fousse, F. Maltey, M. Meulien, M. Mezzarobba, C. Pernet, N. M. Thiéry, and P. Zimmermann. "Calcul mathématique avec Sage". In: Chap. 8: Algèbre linéaire. Amazon, 2013, p. 468.

[B-DP13] J.-G. Dumas and C. Pernet. "Computational linear algebra over finite fields". In: *Handbook of Finite Fields*. Ed. by G. L. Mullen and D. Panario. Discrete Mathematics and Its Applications. Chap. 13.4: Linear algebra over finite fields. Chapman & Hall / CRC, June 2013, pp. 514–528.


## Refereed journals


[J-JPS13] C.-P. Jeannerod, C. Pernet, and A. Storjohann. "Rank-profile revealing Gaussian elimination and the CUP matrix decomposition". In: *Journal of Symbolic Computation* 56 (2013), pp. 46–68. DOI: 10.1016/j.jsc.2013.04.004 (see pp. 8, 9, 14, 16, 19, 21, 22).

[J-PS10] C. Pernet and W. Stein. "Fast computation of Hermite normal forms of random integer matrices". In: *Journal of Number Theory* 130.7 (July 2010), pp. 1675–1683. DOI: 10.1016/j.jnt.2010.01.017 (see pp. 8, 9).

[J-DGP08] J.-G. Dumas, P. Giorgi, and C. Pernet. "Dense Linear Algebra over Word-Size Prime Fields: the FFLAS and FFPACK Packages". In: *ACM Trans. on Mathematical Software (TOMS)* 35.3 (2008), pp. 1–42. DOI: 10.1145/1391989.1391992 (see pp. 8, 10, 11, 14, 21).


## Refereed conference proceedings


[R-Boy+14] B. Boyer, J.-G. Dumas, P. Giorgi, C. Pernet, and B. Saunders. "Elements of Design for Containers and Solutions in the LinBox Library". English. In: *Mathematical Software - ICMS 2014*. Ed. by H. Hong and C. Yap. Vol. 8592. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 654–662. DOI: 10.1007/978-3-662-44199-2_98.

[R-Dum+14] J.-G. Dumas, T. Gautier, C. Pernet, and Z. Sultan. "Parallel Computation of Echelon Forms". English. In: *Euro-Par 2014 Parallel Processing*. Ed. by F. Silva, I. Dutra, and V. Santos Costa. Vol. 8632. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 499–510. DOI: 10.1007/978-3-319-09873-9_42 (see pp. 8, 15, 16).

[R-KP14] E. L. Kaltofen and C. Pernet. "Sparse Polynomial Interpolation Codes and Their Decoding Beyond Half the Minimum Distance". In: *Proc. ISSAC'14*. Kobe, Japan: ACM, 2014, pp. 272–279. DOI: 10.1145/2608628.2608660 (see pp. 30, 45–47, 49, 50).

[R-BPQ13] M. Barbier, C. Pernet, and G. Quintin. "On decoding of quasi-BCH codes". In: *Proc. WCC'13*. Bergen, Norway, 2013.

[R-DPS13] J.-G. Dumas, C. Pernet, and Z. Sultan. "Simultaneous computation of the row and column rank profiles". In: *Proc. ISSAC'13*. ACM Press, 2013. DOI: 10.1145/2465506.2465517 (see pp. 8, 14, 16, 17, 19, 22, 23).

[R-CKP12] M. Comer, E. Kaltofen, and C. Pernet. "Sparse Polynomial Interpolation and Berlekamp/-Massey Algorithm That Correct Outlier Errors in Input Values". In: *Proc. ISSAC'12*. July 2012. DOI: 10.1145/2442829.2442852 (see pp. 30, 45, 46, 53).

[R-Kho+10] M. Khonji, C. Pernet, J.-L. Roch, T. Roche, and T. Stalinski. "Output-sensitive decoding for redundant residue systems". In: *ISSAC'10*. Munich, Germany: ACM Press, 2010, pp. 265–272. DOI: 10.1145/1837934.1837985 (see pp. 30, 35, 36, 42).

[R-Boy+09] B. Boyer, J.-G. Dumas, C. Pernet, and W. Zhou. "Memory efficient scheduling of Strassen-Winograd's matrix multiplication algorithm". In: *Proc. ISSAC'09*. Seoul, Corea: ACM Press, 2009. DOI: 10.1145/1576702.1576713 (see pp. 8, 12).

[R-DPS09] J.-G. Dumas, C. Pernet, and D. Saunders. "On finding multiplicities of characteristic polynomial factors of black-box matrices". In: *Proc. ISSAC'09*. Seoul, Corea: ACM Press, 2009. DOI: 10.1145/1576702.1576723 (see pp. 8, 9).

[R-PS07a] C. Pernet and A. Storjohann. "Faster algorithms for the characteristic polynomial". In: *Proc. ISSAC'07*. Waterloo, Ontario, Canada: ACM Press, 2007, pp. 307–314. DOI: 10.1145/1277548.1277590 (see pp. 8, 9, 26).

[R-DPW05] J.-G. Dumas, C. Pernet, and Z. Wan. "Efficient Computation of the Characteristic Polynomial". In: *Proc. ISSAC'05*. Beijing, China: ACM Press, July 24–27, 2005. DOI: 10.1145/1073884.1073905 (see pp. 8, 25).

[R-DGP04] J.-G. Dumas, P. Giorgi, and C. Pernet. "FFPACK: Finite Field Linear Algebra Package". In: *Proc. ISSAC'04*. Santander, Spain: ACM Press, July 4–7, 2004. DOI: 10.1145/1005285.1005304 (see p. 14).

[R-DGP02] J.-G. Dumas, T. Gautier, and C. Pernet. "Finite Field Linear Algebra Subroutines". In: *Proc. ISSAC'02*. Lille, France: ACM Press, July 7–10, 2002. DOI: 10.1145/780506.780515 (see p. 10).

## Conference proceedings

[P-KRP13] A. Kumar, J.-L. Roch, and C. Pernet. "Secured Outsourced Linear Algebra". In: *SAFECOMP 2013 FastAbstract - The 32nd Int. Conf. on Computer Safety, Reliability and Security*. Ed. by M.-O. Killijian. hal-00926445. Toulouse, France, Sept. 2013, NC.

[P-AP10] M. Albrecht and C. Pernet. "Efficient Decomposition of Dense Matrices over GF(2)". In: *Proc. of the Wokshop on Tools for Cryptanalysis*. arXiv:1006.1744 [cs.MS]. June 2010.

[P-DPR06] J.-G. Dumas, C. Pernet, and J.-L. Roch. "Adaptive Triangular System Solving". In: *Challenges in Symbolic Computation Software*. Dagstuhl Seminar Proceedings 06271. 2006.

## Technical reports

[T-DPS14] J.-G. Dumas, C. Pernet, and Z. Sultan. *Computing the rank profile matrix*. Tech. rep. http://lig-membres.imag.fr/pernet/Publications/DuPeSu14_RPM.pdf. LIP, LJK, Sept. 2014 (see pp. 19, 23).

[T-PS07b] C. Pernet and A. Storjohann. *Frobenius form in expected matrix multiplication time over sufficiently large fields*. Tech. rep. https://cs.uwaterloo.ca/~astorjoh/cpoly.pdf. Symbolic Computation Group, University of Waterloo, Nov. 2007 (see pp. 9, 28).

[T-PRV05] C. Pernet, A. Rondepierre, and G. Villard. *Computing the Kalman form*. Tech. rep. ArXiv cs.SC/0510014. Labortoire Jean Kuntzmann, Grenoble, Oct. 2005.

[T-Per01] C. Pernet. *Implementation of Winograd's fast matrix multiplication over finite fields using ATLAS level 3 BLAS*. Tech. rep. http://ljk.imag.fr/membres/Jean-Guillaume.Dumas/FFLAS/FFLAS_Download/FFLAS_technical_report.ps.gz. Laboratoire Informatique et Distribution, July 2001.

# References


[ABM99] J. Abbott, M. Bronstein, and T. Mulders. "Fast Deterministic Computation of Determinants of Dense Matrices". In: *Proc. of the 1999 International Symposium on Symbolic and Algebraic Computation*. ISSAC '99. Vancouver, British Columbia, Canada: ACM, 1999, pp. 197–204. DOI: 10.1145/309831.309934 (see p. 9).

[Agu+09] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov. "Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects". In: *Journal of Physics: Conference Series* 180.1 (2009), p. 012037. DOI: 10.1088/1742-6596/180/1/012037 (see pp. 8, 9).

[Alb12] M. R. Albrecht. "The M4RIE Library for Dense Linear Algebra over Small Fields with Even Characteristic". In: *Proc. of the 37th International Symposium on Symbolic and Algebraic Computation*. ISSAC '12. Grenoble, France: ACM, 2012, pp. 28–34. DOI: 10.1145/2442829.2442838 (see p. 10).

[ABH10] M. Albrecht, G. Bard, and W. Hart. "Algorithm 898: Efficient Multiplication of Dense Matrices over GF(2)". In: *ACM Trans. on Mathematical Software (TOMS)* 37.1 (Jan. 2010), 9:1–9:14. DOI: 10.1145/1644001.1644010 (see p. 10).

[Ale02] M. Alekhnovich. "Linear Diophantine equations over polynomials and soft decoding of Reed-Solomon codes". In: *Foundations of Computer Science, 2002. Proc. of the 43rd Annual IEEE Symposium on*. 2002, pp. 439–448. DOI: 10.1109/SFCS.2002.1181968 (see p. 40).

[And+90] E. Anderson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammerling, J. Demmel, C. Bischof, and D. Sorensen. "LAPACK: A Portable Linear Algebra Library for High-performance Computers". In: *Supercomputing, 1990, Proc. of the 1990 ACM/IEEE Conference on*. New York, New York, USA: IEEE Computer Society Press, 1990, pp. 2–11. DOI: 10.1109/SUPERC.1990.129995 (see p. 8).

[Arl+70] V. Arlazarov, E. Dinic, M. Kronrod, and I. Faradzev. "On economical construction of the transitive closure of a directed graph". In: *Dokl. Akad. Nauk.* 194.11 (1970). (in Russian), English Translation in Soviet Math Dokl. (see p. 10).

[BS83] W. Baur and V. Strassen. "The complexity of partial derivatives". In: *Theoretical Computer Science* 22.3 (1983), pp. 317–330. DOI: 10.1016/0304-3975(83)90110-X (see p. 26).

[Bee+13] P. Beelen, T. Hoholdt, J. Nielsen, and Y. Wu. "On Rational Interpolation-Based List-Decoding and List-Decoding Binary Goppa Codes". In: *IEEE Trans. on Information Theory* 59.6 (June 2013), pp. 3269–3281. DOI: 10.1109/TIT.2013.2243800 (see p. 36).

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. "Completeness Theorems for Non-cryptographic Fault-tolerant Distributed Computation". In: *Proc. of the 20th Annual ACM Symposium on Theory of Computing*. STOC '88. Chicago, Illinois, USA: ACM, 1988, pp. 1–10. DOI: 10.1145/62212.62213 (see p. 30).

[BT88] M. Ben-Or and P. Tiwari. "A Deterministic Algorithm for Sparse Multivariate Polynomial Interpolation". In: *Proc. of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC '88. Chicago, Illinois, USA: ACM, 1988, pp. 301–309. DOI: 10.1145/62212.62241 (see p. 44).

[BW86] E. R. Berlekamp and L. R. Welch. *Error correction for algebraic block codes*. US Patent 4,633,470. Dec. 1986 (see p. 34).

[Ber11] D. J. Bernstein. "Simplified High-Speed High-Distance List Decoding for Alternant Codes". In: *Post-Quantum Cryptography*. Ed. by B.-Y. Yang. Vol. 7071. LNCS. Springer, 2011, pp. 200–216. DOI: 10.1007/978-3-642-25405-5_13 (see p. 40).

[Bin+79] D. Bini, M. Capovani, F. Romani, and G. Lotti. "$O(n^{2.7799})$ complexity for $n \times n$ approximate matrix multiplication". In: *Information Processing Letters* 8.5 (1979), pp. 234–235. DOI: 10.1016/0020-0190(79)90113-3 (see p. 12).

[Bla83] R. E. Blahut. *Theory and Practice of Error Control Codes*. Addison Wesley, Reading, 1983 (see pp. 31, 32, 44).

[BKY07] D. Bleichenbacher, A. Kiayias, and M. Yung. "Decoding interleaved Reed–Solomon codes over noisy channels". In: *Theoretical Computer Science* 379.3 (2007). Automata, Languages and Programming, pp. 348–360. DOI: 10.1016/j.tcs.2007.02.043 (see p. 39).

[BKY03] D. Bleichenbacher, A. Kiayias, and M. Yung. "Decoding of Interleaved Reed Solomon Codes over Noisy Data". English. In: *Automata, Languages and Programming*. Ed. by J. Baeten, J. Lenstra, J. Parrow, and G. Woeginger. Vol. 2719. Lecture Notes in Computer Science. Springer, 2003, pp. 97–108. DOI: 10.1007/3-540-45061-0_9 (see p. 39).

[Bol+92] D. Boley, R. Brent, G. Golub, and F. Luk. "Algorithmic Fault Tolerance Using the Lanczos Method". In: *SIAM Journal on Matrix Analysis and Applications* 13.1 (1992), pp. 312–332. DOI: 10.1137/0613023 (see p. 29).

[Bon00] D. Boneh. "Finding Smooth Integers in Short Intervals Using CRT Decoding". In: *Proc. of the Thirty-second Annual ACM Symposium on Theory of Computing*. STOC '00. Portland, Oregon, USA: ACM, 2000, pp. 265–272. DOI: 10.1145/335305.335337 (see p. 42).

[BB09] T. J. Boothby and R. W. Bradshaw. "Bitslicing and the Method of Four Russians Over Larger Finite Fields". In: *CoRR* arXiv:0901.1413 (2009) (see p. 10).

[Bos+12] G. Bosilca, A. Bouteiller, A. Danalis, T. Herault, P. Lemarinier, and J. Dongarra. "DAGuE: A generic distributed DAG engine for High Performance Computing". In: *Parallel Computing* 38.1–2 (2012). Extensions for Next-Generation Parallel Programming Models, pp. 37–51. DOI: 10.1016/j.parco.2011.10.003 (see p. 24).

[Bos+09] G. Bosilca, R. Delmas, J. Dongarra, and J. Langou. "Algorithm-based fault tolerance applied to high performance computing". In: *Journal of Parallel and Distributed Computing* 69.4 (2009), pp. 410–416. DOI: 10.1016/j.jpdc.2008.12.002 (see p. 29).

[BJS08] A. Bostan, C.-P. Jeannerod, and É. Schost. "Solving structured linear systems with large displacement rank". In: *Theoretical Computer Science* 407.1–3 (2008), pp. 155–181. DOI: 10.1016/j.tcs.2008.05.014 (see p. 53).

[Boy12] B. Boyer. "Multiplication matricielle efficace et conception logicielle pour la bibliothèque de calcul exact `LinBox` ". PhD thesis. Université de Grenoble, June 2012 (see p. 12).

[BD14] B. Boyer and J.-G. Dumas. "Matrix multiplication over word-size prime fields using Bini's approximate formula". hal-00987812. May 2014 (see p. 12).

[BK14] B. Boyer and E. L. Kaltofen. "Numerical Linear System Solving with Parametric Entries by Error Correction". In: *Proceedings of the 2014 Symposium on Symbolic-Numeric Computation*. SNC '14. Shanghai, China: ACM, 2014, pp. 33–38. DOI: 10.1145/2631948.2631956 (see pp. 30, 53).

[BGD12] F. Broquedis, T. Gautier, and V. Danjean. "libKOMP, an Efficient OpenMP Runtime System for Both Fork-Join and Data Flow Paradigms". In: *OpenMP in a Heterogeneous World*. Vol. 7312. LNCS. Springer, 2012, pp. 102–115. DOI: 10.1007/978-3-642-30961-8_8 (see p. 9).

[BH74] J. R. Bunch and J. E. Hopcroft. "Triangular Factorization and Inversion by Fast Matrix Multiplication." In: *Mathematics of Computation* 28 (1974), pp. 231–236. DOI: 10.1090/S0025-5718-1974-0331751-8 (see p. 14).

[BCS97] B. Bürgisser, C. Clausen, and M. Shokrollahi. *Algebraic Complexity Theory*. Vol. 315. Grundlehren der mathematischen Wissenschaften. Springer-Verlag, 1997 (see pp. 6, 21).

[But+09] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra. "A class of parallel tiled linear algebra algorithms for multicore architectures". In: *Parallel Computing* 35.1 (2009), pp. 38–53. DOI: 10.1016/j.parco.2008.10.002 (see p. 13).

[Cab71] S. Cabay. "Exact Solution of Linear Equations". In: *Proceedings of the Second ACM Symposium on Symbolic and Algebraic Manipulation*. SYMSAC '71. New York, NY, USA: ACM, 1971, pp. 392–398. DOI: 10.1145/800204.806310 (see p. 40).

[CT06] E. Candes and T. Tao. "Near-Optimal Signal Recovery From Random Projections: Universal Encoding Strategies?" In: *IEEE Trans. on Information Theory* 52.12 (2006), pp. 5406–5425. DOI: 10.1109/TIT.2006.885507 (see p. 44).

[Cap+09] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir. "Toward Exascale Resilience". In: *International Journal of High Performance Computing Applications* (2009). DOI: 10.1177/1094342009347767 (see p. 29).

[Che52] P. L. Chebyshev. "Mémoire sur les nombres premiers". In: *J. de Mathématiques Pures et Appliquées* 17 (1852), pp. 366–390 (see p. 49).

[Che+02] L. Chen, W. Eberly, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard. "Efficient matrix preconditioners for black box linear algebra". In: *Linear Algebra and its Applications* 343–344 (2002). Special Issue on Structured and Infinite Systems of Linear equations, pp. 119–146. DOI: 10.1016/S0024-3795(01)00472-4 (see pp. 7, 9).

[Cho+14] M. F. I. Chowdhury, C.-P. Jeannerod, V. Neiger, É. Schost, and G. Villard. "Faster Algorithms for Multivariate Interpolation with Multiplicities and Simultaneous Polynomial Approximations". In: *arXiv CoRR* abs/1402.0643 (2014) (see pp. 5, 40, 53).

[CH10] H. Cohn and N. Heninger. "Ideal forms of Coppersmith's theorem and Guruswami-Sudan list decoding". In: *ArXiv 1008.1284* (Aug. 2010) (see p. 40).

[COS86] D. Coppersmith, A. M. Odlzyko, and R. Schroeppel. "Discrete logarithms inGF(p)". In: *Algorithmica* 1.1 (Nov. 1, 1986), pp. 1–15. DOI: 10.1007/BF01840433 (see p. 9).

[CS03] D. Coppersmith and M. Sudan. "Reconstructing Curves in Three (and Higher) Dimensional Space from Noisy Data". In: *Proc. of the Thirty-fifth Annual ACM Symposium on Theory of Computing*. STOC '03. San Diego, CA, USA: ACM, 2003, pp. 136–142. DOI: 10.1145/780542.780563 (see p. 40).

[Di +14] C. Di Martino, F. Baccanico, J. Fullop, W. Kramer, K. Zbigniew, and R. Iyer. "Lessons Learned From the Analysis of System Failures at Petascale: The Case of Blue Waters". In: *Proc. of the 44th international conference on Dependable Systems and Networks (DSN 2014)*. Atlanta, USA, 2014 (see p. 29).

[Don+85] J. J. Dongarra, J. D. Croz, S. Hammarling, and R. J. Hanson. "A Proposal for an Extended Set of Fortran Basic Linear Algebra Subprograms". In: *SIGNUM Newsl.* 20.1 (Jan. 1985), pp. 2–18. DOI: 10.1145/1057935.1057936 (see p. 8).

[Don+98] J. J. Dongarra, L. S. Duff, D. C. Sorensen, and H. A. V. Vorst. *Numerical Linear Algebra for High Performance Computers*. SIAM, 1998 (see pp. 13, 14).

[Don+14] J. J. Dongarra, M. Faverge, H. Ltaief, and P. Luszczek. "Achieving Numerical Accuracy and High Performance using Recursive Tile LU Factorization". In: *Concurrency and Computation: Practice and Experience* 26.7 (2014). hal-00809765, pp. 1408–1431 (see pp. 13, 23).

[Don+87] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling. "A Proposal for a Set of Level 3 Basic Linear Algebra Subprograms". In: *SIGNUM Newsl.* 22.3 (July 1987), pp. 2–14. DOI: 10.1145/36318.36319 (see p. 8).

[Dor87] J. Dornstetter. "On the equivalence between Berlekamp's and Euclid's algorithms (Corresp.)" In: *IEEE Trans. on Information Theory* 33.3 (1987), pp. 428–431 (see p. 33).

[Dou+94] C. C. Douglas, M. Heroux, G. Slishman, and R. M. Smith. "GEMMW: A Portable Level 3 {BLAS} Winograd Variant of Strassen's Matrix-Matrix Multiply Algorithm". In: *Journal of Computational Physics* 110.1 (1994), pp. 1–10. DOI: 10.1006/jcph.1994.1001 (see p. 12).

[Du+12] P. Du, A. Bouteiller, G. Bosilca, T. Herault, and J. Dongarra. "Algorithm-based Fault Tolerance for Dense Matrix Factorizations". In: *PPoPP'12*. New Orleans, Louisiana, USA: ACM, 2012, pp. 225–234. DOI: 10.1145/2145816.2145845 (see p. 29).

[DFS11] J.-G. Dumas, L. Fousse, and B. Salvy. "Simultaneous modular reduction and Kronecker substitution for small finite fields". In: *Journal of Symbolic Computation* 46.7 (2011). Special Issue in Honour of Keith Geddes on his 60th Birthday, pp. 823–840. DOI: 10.1016/j.jsc.2010.08.015 (see p. 10).

[DR02] J.-G. Dumas and J.-L. Roch. "On parallel block algorithms for exact triangularizations". In: *Parallel Computing* 28.11 (Nov. 2002), pp. 1531–1548. DOI: 10.1016/S0167-8191(02)00161-8 (see p. 14).

[Ebe00] W. Eberly. *Asymptotically efficient algorithms for the Frobenius form*. Tech. rep. Dpt. of Computer Science, University of Calgary, 2000 (see p. 27).

[Eln+02] E. N. ( Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. "A Survey of Rollback-recovery Protocols in Message-passing Systems". In: *ACM Comput. Surv.* 34.3 (Sept. 2002), pp. 375–408. DOI: 10.1145/568522.568525 (see p. 29).

[Emi98] I. Z. Emiris. "A complete implementation for computing general dimensional convex hulls". In: *International Journal of Computational Geometry & Applications* 8.02 (1998), pp. 223–253. DOI: 10.1142/S0218195998000126 (see p. 35).

[ET36] P. Erdös and P. Turán. "On Some Sequences of Integers". In: *J. London Math. Soc.* S1-11.4 (1936), pp. 261–264. DOI: 10.1112/jlms/s1-11.4.261 (see p. 49).

[Fau99] J.-C. Faugère. "A new efficient algorithm for computing Gröbner bases (F4)". In: *Journal of Pure and Applied Algebra* 139.1–3 (1999), pp. 61–88. DOI: 10.1016/S0022-4049(99)00005-5 (see p. 16).

[FT85] G.-L. Feng and K. K. Tzeng. "An iterative algorithm of shift-register synthesis for multiple sequences". In: *Scientia Sinica (Science in China)* XXVIII (1985), pp. 1222–1232 (see p. 39).

[FT91] G.-L. Feng and K. Tzeng. "A generalization of the Berlekamp-Massey algorithm for multisequence shift-register synthesis with applications to decoding cyclic codes". In: *IEEE Trans. on Information Theory* 37.5 (Sept. 1991), pp. 1274–1287. DOI: 10.1109/18.133246 (see p. 39).

[Fre+88] T. S. Freeman, G. M. Imirzian, E. Kaltofen, and L. Yagati. "DAGWOOD: A system for manipulating polynomials given by straight-line programs". In: *ACM Trans. on Mathematical Software (TOMS)* 14.3 (1988), pp. 218–240 (see p. 35).

[Gao02] S. Gao. "A New Algorithm for Decoding Reed-Solomon Codes". In: *in Communications, Information and Network Security, V.Bhargava, H.V.Poor, V.Tarokh, and S.Yoon*. Kluwer, 2002, pp. 55–68 (see p. 34).

[GS09] S. Garg and É. Schost. "Interpolation of polynomials given by straight-line programs". In: *Theoretical Comput. Sci.* 410.27-29 (2009), pp. 2659–2662. DOI: 10.1016/j.tcs.2009.03.030 (see p. 44).

[GG13] J. v. Gathen and J. Gerhard. *Modern Computer Algebra*. 3rd ed. New York, NY, USA: Cambridge University Press, 2013 (see pp. 11, 32, 41).

[Gau+12] T. Gautier, F. Lementec, V. Faucher, and B. Raffin. *X-Kaapi: a Multi Paradigm Runtime for Multicore Architectures*. Tech. report RR-8058. hal-00727827. Inria, Feb. 2012 (see pp. 9, 24).

[Gie93] M. Giesbrecht. "Nearly Optimal Algorithms for Canonical Matrix Forms". PhD thesis. University of Toronto, 1993 (see pp. 27, 28).

[Gie95] M. Giesbrecht. "Nearly Optimal Algorithms for Canonical Matrix Forms". In: *SIAM Journal on Computing* 24.5 (1995), pp. 948–969. DOI: 10.1137/S0097539793252687 (see p. 27).

[GR10] M. Giesbrecht and D. S. Roche. “Interpolation of Shifted-Lacunary Polynomials”. In: *Computational Complexity* 19.3 (Sept. 2010), pp. 333–354. DOI: 10.1007/s00037-010-0294-0 (see p. 44).

[Gio14] P. Giorgi. *Toward High Performance Matrix Multiplication for Exact Computation*. talk given at SIAM Conference on Parallel Processing for Scientific Computing. www.lirmm.fr/~giorgi/seminaire-ljk-14.pdf. 2014 (see p. 10).

[GJV03] P. Giorgi, C.-P. Jeannerod, and G. Villard. “On the Complexity of Polynomial Matrix Computations”. In: *Proc. of the 2003 International Symposium on Symbolic and Algebraic Computation*. ISSAC ’03. Philadelphia, PA, USA: ACM, 2003, pp. 135–142. DOI: 10.1145/860854.860889 (see pp. 33, 53).

[GL14] P. Giorgi and R. Lebreton. “Online order basis algorithm and its impact on the block Wiedemann algorithm”. In: *Proc. ISSAC’14*. Kobe, Japan: ACM Press, July 2014 (see p. 53).

[GSW12] A. M. Gleixner, D. E. Steffy, and K. Wolter. “Improving the Accuracy of Linear Programming Solvers with Iterative Refinement”. In: *Proc. of the 37th International Symposium on Symbolic and Algebraic Computation*. ISSAC ’12. Grenoble, France: ACM, 2012, pp. 187–194. DOI: 10.1145/2442829.2442858 (see p. 5).

[GRS99] O. Goldreich, D. Ron, and M. Sudan. “Chinese remaindering with errors”. In: *STOC ’99: Proc. of the 31st Annual Symposium on Theory of Computing*. Atlanta, Georgia, United States: ACM, 1999, pp. 225–234. DOI: 10.1145/301250.301309 (see p. 42).

[Gol97] S. Goldwasser. “Multi Party Computations: Past and Present”. In: *Proc. of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*. PODC ’97. Santa Barbara, California, USA: ACM, 1997, pp. 1–6. DOI: 10.1145/259380.259405 (see p. 30).

[Gol96] G. H. Golub. *Matrix computations*. 3rd. Johns Hopkins University Press, Oct. 15, 1996 (see p. 18).

[Gow01] W. T. Gowers. “A new proof of Szemerédi’s theorem”. In: *Geom. Funct. Anal.* 11.3 (2001), pp. 465–588. DOI: 10.1007/s00039-001-0332-9 (see p. 50).

[Gt14] T. Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*. 6.0.0. gmplib.org. 2014 (see p. 10).

[GK93] D. Y. Grigoriev and M. Karpinski. “A zero-test and an interpolation algorithm for the shifted sparse polynomials”. In: *Proc. AAECC-10*. Vol. 673. Lect. Notes Comput. Sci. Springer Verlag, 1993, pp. 162–169 (see p. 44).

[GM84] R. Gupta and M. Murty. “A remark on Artin’s conjecture”. English. In: *Inventiones mathematicae* 78.1 (1984), pp. 127–130. DOI: 10.1007/BF01388719 (see p. 52).

[GSS00] V. Guruswami, A. Sahai, and M. Sudan. “”Soft-decision” decoding of Chinese remainder codes”. In: *FOCS’2000: Proc. of the 41st Annual Symposium on Foundations of Computer Science, 2000*. 2000, pp. 159–168. DOI: 10.1109/SFCS.2000.892076 (see p. 42).

[GS99] V. Guruswami and M. Sudan. “Improved decoding of Reed-Solomon and algebraic-geometry codes”. In: *IEEE Trans. on Information Theory* 45.6 (1999), pp. 1757–1767. DOI: 10.1109/18.782097 (see p. 35).

[Ht14] W. Hart and the MPIR development team. *MPIR: MMulti Precision Integers and Rationals*. 2.7.0. mpir.org. 2014 (see p. 10).

[Hea86] D. R. Heath-Brown. “Artin’s conjecture for primitive roots”. In: *The Quarterly Journal of Mathematics* 37.1 (1986), pp. 27–38. DOI: 10.1093/qmath/37.1.27 (see p. 52).

[Hey+12] S. Heyse, E. Kiltz, V. Lyubashevsky, C. Paar, and K. Pietrzak. “Lapin: An Efficient Authentication Protocol Based on Ring-LPN”. English. In: *Fast Software Encryption*. Ed. by A. Canteaut. Vol. 7549. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 346–365. DOI: 10.1007/978-3-642-34047-5_20 (see p. 54).

[HLQ14] J. van der Hoeven, G. Lecerf, and G. Quintin. “Modular SIMD arithmetic in Mathemagix”. In: *CoRR* arXiv:1407.3383 (2014) (see p. 11).

[Hoo67] C. Hooley. "On Artin's conjecture." eng. In: *Journal für die reine und angewandte Mathematik* 225 (1967), pp. 209–220 (see p. 52).

[HA84] K.-H. Huang and J. A. Abraham. "Algorithm-Based Fault Tolerance for Matrix Operations". In: *IEEE Trans. Comput.* 33.6 (June 1984), pp. 518–528. DOI: 10.1109/TC.1984.1676475 (see p. 29).

[Hus+96] S. Huss-Lederman, E. M. Jacobson, J. Johnson, A. Tsao, and T. Turnbull. "Implementation of Strassen's Algorithm for Matrix Multiplication". In: *Supercomputing, 1996. Proc. of the 1996 ACM/IEEE Conference on*. 1996, pp. 32–32. DOI: 10.1109/SUPERC.1996.183534 (see p. 12).

[IMH82] O. H. Ibarra, S. Moran, and R. Hui. "A generalization of the fast {LUP} matrix decomposition algorithm and applications". In: *Journal of Algorithms* 3.1 (1982), pp. 45–56. DOI: 10.1016/0196-6774(82)90007-4 (see pp. 9, 14, 19, 21).

[KLW90] E. Kaltofen, Lakshman, and J. M. Wiley. "Modular rational sparse multivariate polynomial interpolation". In: *Proc. ISSAC'90*. Ed. by S. Watanabe and M. Nagata. ACM Press, 1990, pp. 135–139. DOI: 10.1145/96877.96912 (see p. 44).

[Kal+12] E. L. Kaltofen, B. Li, Z. Yang, and L. Zhi. "Exact certification in global polynomial optimization via sums-of-squares of rational functions with rational coefficients". In: *Journal of Symbolic Computation* 47.1 (2012), pp. 1–15. DOI: 10.1016/j.jsc.2011.08.002 (see p. 5).

[KY13] E. L. Kaltofen and Z. Yang. "Sparse Multivariate Function Recovery from Values with Noise and Outlier Errors". In: *Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation*. ISSAC '13. Boston, Maine, USA: ACM, 2013, pp. 219–226. DOI: 10.1145/2465506.2465524 (see pp. 30, 53).

[KY14] E. L. Kaltofen and Z. Yang. "Sparse Multivariate Function Recovery with a High Error Rate in the Evaluations". In: *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. ISSAC '14. Kobe, Japan: ACM, 2014, pp. 280–287. DOI: 10.1145/2608628.2608637 (see pp. 30, 39, 53).

[KLL00] E. Kaltofen, W.-s. Lee, and A. A. Lobo. "Early Termination in Ben-Or/Tiwari Sparse Interpolation and a Hybrid of Zippel's Algorithm". In: *Proc. of the 2000 International Symposium on Symbolic and Algebraic Computation*. ISSAC '00. St. Andrews, Scotland: ACM, 2000, pp. 192–201. DOI: 10.1145/345542.345629 (see pp. 35, 53).

[KR89] E. Kaltofen and H. Rolletschek. "Computing Greatest Common Divisors and Factorizations in Quadratic Number Fields". In: *Mathematics of Computation* 53.188 (1989), pp. 697–720. DOI: 10.2307/2008732 (see p. 41).

[KS91] E. Kaltofen and B. D. Saunders. "On Wiedemann's method of solving sparse linear systems". English. In: *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*. Ed. by H. Mattson, T. Mora, and T. Rao. Vol. 539. Lecture Notes in Computer Science. Springer, 1991, pp. 29–38. DOI: 10.1007/3-540-54522-0_93 (see pp. 7, 9).

[KT90] E. Kaltofen and B. M. Trager. "Computing with Polynomials Given by Black Boxes for Their Evaluations: Greatest Common Divisors, Factorization, Separation of Numerators and Denominators". In: *J. Symb. Comput.* 9.3 (Mar. 1990), pp. 301–320. DOI: 10.1016/S0747-7171(08)80015-6 (see p. 35).

[KYZ07] E. Kaltofen, Z. Yang, and L. Zhi. "On probabilistic analysis of randomization in hybrid symbolic-numeric algorithms". In: *Proc. SNC'07*. London, Ontario, Canada: ACM Press, 2007, pp. 11–17. DOI: 10.1145/1277500.1277503 (see p. 53).

[Kel85] W. Keller-Gehrig. "Fast algorithms for the characteristic polynomial". In: *Theoretical computer science* 36 (1985), pp. 309–317 (see pp. 19, 21, 22, 25, 26).

[KM06] S. Khodadad and M. Monagan. "Fast rational function reconstruction". In: *ISSAC 2006*. New York: ACM, 2006, pp. 184–190. DOI: 10.1145/1145768.1145801 (see p. 36).

[KG95] K. Klimkowski and R. A. van de Geijn. "Anatomy of a Parallel Out-of-Core Dense Linear Solver". In: *ICPP*. Vol. 3. Urbana Champaign, Illinois, USA: CRC Press, Aug. 1995, pp. 29–33 (see p. 13).

[LSP82] L. Lamport, R. Shostak, and M. Pease. "The Byzantine Generals Problem". In: *ACM Trans. Program. Lang. Syst.* 4.3 (1982), pp. 382–401. DOI: 10.1145/357172.357176 (see p. 29).

[Lap95] J.-C. Laprie. "DEPENDABLE COMPUTING AND FAULT TOLERANCE : CONCEPTS AND TERMINOLOGY". In: *Fault-Tolerant Computing, 1995, Highlights from Twenty-Five Years., Twenty-Fifth International Symposium on*. June 1995, pp. 2–. DOI: 10.1109/FTCSH.1995.532603 (see p. 29).

[Law+79] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. "Basic Linear Algebra Subprograms for Fortran Usage". In: *ACM Trans. on Mathematical Software (TOMS)* 5.3 (Sept. 1979), pp. 308–323. DOI: 10.1145/355841.355847 (see p. 8).

[LeG14] F. Le Gall. "Powers of Tensors and Fast Matrix Multiplication". In: *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. ISSAC '14. Kobe, Japan: ACM, 2014, pp. 296–303. DOI: 10.1145/2608628.2608664 (see p. 6).

[LN97] R. Lidl and H. Niederreiter. *Finite fields*. 2nd ed. Vol. 20. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1997 (see p. 51).

[LP86] F. T. Luk and H. Park. "An Analysis Of Algorithm-Based Fault Tolerance Techniques". In: *Proc. SPIE*. Vol. 0696. 1986, pp. 222–227. DOI: 10.1117/12.936896 (see p. 29).

[Mal10] G. I. Malaschonok. "Fast generalized Bruhat decomposition". In: *CASC'10*. Vol. 6244. LNCS. Tsakhkadzor, Armenia: Springer-Verlag, 2010, pp. 194–202. DOI: 10.1007/978-3-642-15274-0_16 (see pp. 14, 17, 22).

[Man76] D. Mandelbaum. "On a class of arithmetic codes and a decoding algorithm (Corresp.)" In: *IEEE Trans. on Information Theory* 22.1 (Jan. 1976), pp. 85–88. DOI: 10.1109/TIT.1976.1055504 (see p. 42).

[Mas69] J. L. Massey. "Shift-register synthesis and BCH decoding". In: *IEEE Trans. on Information Theory* IT-15 (1969), pp. 122–127 (see p. 44).

[MS88] J. Massey and T. Schaub. "Linear complexity in coding theory". In: *Coding Theory and App.* Ed. by G. Cohen and P. Godlewski. Vol. 311. LNCS. Springer Verlag, 1988, pp. 19–32 (see pp. 31, 32, 44).

[Moo05] T. K. Moon. *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, 2005 (see p. 39).

[Mur88] M. Murty. "Artin's conjecture for primitive roots". English. In: *The Mathematical Intelligencer* 10.4 (1988), pp. 59–67. DOI: 10.1007/BF03023749 (see p. 52).

[Nie13] J. S. R. Nielsen. "List decoding of algebraic codes". http://jsrn.dk/downloads/jsrn_msc.pdf. PhD thesis. Technical University of Denmark, Sept. 2013 (see p. 40).

[Oda+83] K. Odaka, Y. Sako, I. Iwamoto, T. Doi, and L. Vries. *Error correctable data transmission method*. US Patent 4,413,340. Nov. 1983 (see p. 39).

[OS07] Z. Olesh and A. Storjohann. "The vector rational function reconstruction problems". In: *Proc. of the Waterloo Workshop on Computer Algebra: devoted to the 60th birthday of Sergei Abramov (WWCA)*. World Scientific, 2007, pp. 137–149 (see pp. 33, 39, 40).

[PV05] F. Parvaresh and A. Vardy. "Correcting errors beyond the Guruswami-Sudan radius in polynomial time". In: *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*. Oct. 2005, pp. 285–294. DOI: 10.1109/SFCS.2005.29 (see p. 40).

[PBL08] J. Perez, R. Badia, and J. Labarta. "A dependency-aware task-based programming environment for multi-core architectures". In: *Cluster Computing, 2008 IEEE International Conference on*. Sept. 2008, pp. 142–151. DOI: 10.1109/CLUSTR.2008.4663765 (see p. 24).

[Per06] C. Pernet. "Algèbre linéaire exacte efficace : le calcul du polynôme caractéristique". tel-00111346. PhD thesis. Université J. Fourier, Grenoble, France, Sept. 2006 (see pp. 11, 26).

[Pro95] R. Prony. "Essai expérimental et analytique sur les lois de la Dilatabilité de fluides élastique et sur celles de la Force expansive de la vapeur de l'eau et de la vapeur de l'alkool, à différentes températures". In: *J. de l'École Polytechnique* 1 (Floréal et Prairial III (1795)), pp. 24–76 (see pp. 31, 44).

[Qui12] G. Quintin. “On the Algorithms of Guruswami-Sudan List Decoding over Finite Rings”. PhD thesis. École polytechnique, Nov. 2012 (see p. 40).

[Ram19] S. Ramanujan. “A proof of Bertrand’s postulate”. In: *Journal of the Indian Mathematical Society* 11 (1919), pp. 181–182 (see p. 49).

[RS60] I. S. Reed and G. Solomon. “Polynomial Codes Over Certain Finite Fields”. English. In: *Journal of the Society for Industrial and Applied Mathematics* 8.2 (1960), pages (see p. 34).

[Rot53] K. F. Roth. “On certain sets of integers”. In: *J. London Math. Soc.* 28 (1953), pp. 104–109 (see p. 50).

[Ste+14] W. Stein et al. *Sage Mathematics Software (Version 6.3)*. http://www.sagemath.org. The Sage Development Team. 2014 (see p. 5).

[SS50] R. Salem and D. C. Spencer. “On sets which do not contain a given number of terms in arithmetical progression”. In: *Nieuw Arch. Wiskunde (2)* 23 (1950), pp. 133–143 (see p. 50).

[Sat+12] K. Sato, K. Mohror, A. Moody, T. Gamblin, B. de Supinski, N. Maruyama, and S. Matsuoka. “Design and modeling of a non-blocking checkpointing system”. In: *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*. Nov. 2012, pp. 1–10. DOI: 10.1109/SC.2012.46 (see p. 29).

[SSB06] G. Schmidt, V. Sidorenko, and M. Bossert. “Decoding Reed-Solomon Codes Beyond Half the Minimum Distance using Shift-Register Synthesis”. In: *Proc. of the 2006 IEEE International Symposium on Information Theory*. July 2006, pp. 459–463. DOI: 10.1109/ISIT.2006.261711 (see p. 39).

[SSB10] G. Schmidt, V. Sidorenko, and M. Bossert. “Syndrome Decoding of Reed-Solomon Codes Beyond Half the Minimum Distance Based on Shift-Register Synthesis”. In: *IEEE Trans. on Information Theory* 56.10 (Oct. 2010), pp. 5245–5252. DOI: 10.1109/TIT.2010.2060130 (see p. 39).

[Sch84] F. B. Schneider. “Byzantine generals in action: implementing fail-stop processors”. In: *ACM Trans. Comput. Syst.* 2.2 (1984), pp. 145–154. DOI: 10.1145/190.357399 (see p. 29).

[Sha79] A. Shamir. “How to Share a Secret”. In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. DOI: 10.1145/359168.359176 (see p. 30).

[Shi88] A. Shiozaki. “Decoding of redundant residue polynomial codes using Euclid’s algorithm”. In: *IEEE Trans. on Information Theory* 34.5 (Sept. 1988), pp. 1351–1354. DOI: 10.1109/18.21269 (see p. 34).

[Sto00] A. Storjohann. “Algorithms for Matrix Canonical Forms”. PhD thesis. Institut für Wissenschaftliches Rechnen, ETH-Zentrum, Zürich, Switzerland, Nov. 2000 (see pp. 19, 22).

[Sto03] A. Storjohann. “High-order lifting and integrality certification”. In: *Journal of Symbolic Computation* 36.3–4 (2003), pp. 613–648. DOI: 10.1016/S0747-7171(03)00097-X (see pp. 7, 30, 53).

[Sto05] A. Storjohann. “The shifted number system for fast linear algebra on integer matrices”. In: *Journal of Complexity* 21.4 (2005). Festschrift for the 70th Birthday of Arnold Schonhage Festschrift for the 70th Birthday of Arnold Schonhage, pp. 609–650. DOI: 10.1016/j.jco.2005.04.002 (see pp. 7, 9, 30).

[SV05] A. Storjohann and G. Villard. “Computing the Rank and a Small Nullspace Basis of a Polynomial Matrix”. In: *Proc. of the 2005 International Symposium on Symbolic and Algebraic Computation*. ISSAC ’05. Beijing, China: ACM, 2005, pp. 309–316. DOI: 10.1145/1073884.1073927 (see p. 7).

[Str69] V. Strassen. “Gaussian elimination is not optimal”. In: *Numerische Mathematik* 13.4 (1969), pp. 354–356. DOI: 10.1007/BF02165411 (see pp. 7, 8).

[Sud97] M. Sudan. “Decoding of Reed-Solomon Codes beyond the Error-Correction Bound”. In: *Journal of Complexity* 13.1 (1997), pp. 180–193. DOI: 10.1006/jcom.1997.0439 (see pp. 35, 40).

[Sze75] E. Szemerédi. "On sets of integers containing no $k$ elements in arithmetic progression". In: *Proc. Int. Congress of Mathematicians (Vancouver, BC, 1974), Vol. 2*. Canad. Math. Congress, Montreal, QC, 1975, pp. 503–505 (see p. 49).

[FFL14] The FFLAS-FFPACK group. *FFLAS-FFPACK: Finite Field Linear Algebra Subroutines / Package*. v2.0.0. http://linalg.org/projects/fflas-ffpack. 2014 (see pp. 5, 10).

[Lin12] The LinBox Group. *LinBox: Linear algebra over black-box matrices*. v1.3.2. http://linalg.org/. 2012 (see pp. 5, 10).

[Vil97] G. Villard. "Fast Parallel Algorithms for Matrix Reduction to Normal Forms". In: *Applicable Algebra in Engineering, Communication and Computing* 8.6 (1997), pp. 511–537. DOI: 10.1007/s002000050089 (see p. 27).

[Wag72] S. S. Wagstaff Jr. "On $k$-free sequences of integers". In: *Mathematics of Computation* 26 (1972), pp. 767–771. DOI: 10.1090/S0025-5718-1972-0325500-5 (see p. 49).

[WH66] R. W. Watson and C. W. Hastings. "Self-checked computation using residue arithmetic". In: *Proceedings of the IEEE* 54.12 (Dec. 1966), pp. 1920–1931. DOI: 10.1109/PROC.1966.5275 (see p. 42).

[Wie86] D. Wiedemann. "Solving sparse linear equations over finite fields". In: *IEEE Trans. on Information Theory* 32.1 (Jan. 1986), pp. 54–62. DOI: 10.1109/TIT.1986.1057137 (see pp. 5, 7, 9).

[Wu08] Y. Wu. "New List Decoding Algorithms for Reed-Solomon and BCH Codes". In: *IEEE Trans. on Information Theory* 54.8 (Aug. 2008), pp. 3611–3630. DOI: 10.1109/TIT.2008.926355 (see pp. 36, 40).

[YKD11] A. YarKhan, J. Kurzak, and J. Dongarra. *QUARK Users'Guide: QUeueing And Runtime for Kernels*. http://ash2.icl.utk.edu/sites/ash2.icl.utk.edu/files/publications/2011/icl-utk-454-2011.pdf. 2011 (see p. 24).

[Zeh12] A. Zeh. "Algebraic Soft- and Hard-decision Decoding of Generalized Reed-Solomon and Cyclic Codes". alex.codingtheory.eu/. PhD thesis. Ulm Universität, Sept. 2012 (see p. 40).

[Zho12] W. Zhou. "Fast order basis and kernel basis computation and related problems". PhD thesis. Unversity of Waterloo, 2012 (see p. 53).

[Zip90] R. Zippel. "Interpolating polynomials from their values". In: *Journal of Symbolic Computation* 9.3 (1990). Computational algebraic complexity editorial, pp. 375–403. DOI: 10.1016/S0747-7171(08)80018-1 (see p. 47).

**Résumé**

Les contributions présentées dans ce mémoire concernent le calcul exact haute performance, à l'interface entre calcul formel, théorie des codes et calcul parallèle.

L'algèbre linéaire exacte, sur des corps finis ou le corps des nombres rationnels est au coeur de nombreuses applications recourant au calcul algébrique intensif: de la cryptanalyse basée sur les cribles algébriques ou sur la résolution de systèmes polynomiaux au test de conjectures en théorie algorithmique des nombres, ou encore au décodage en liste... Le développement aussi bien de l'algorithmique que des implantations efficaces en algèbre linéaire exacte a atteint ces dernières années un grand niveau de maturité. Nous évoquons les grandes lignes de ces avancées, illustrées par nos contributions, en dégageant quelques principes généraux qui les sous-tendent, comme celui de rendre les réductions algorithmiques effectives. Le produit de matrices, combinant conversions entres arithmétiques modulaires, entières et flottantes, usage des BLAS numériques et d'algorithmes sous-cubiques à faible empreinte mémoire, sert de brique de base. L'élimination de Gauss en tire parti par ses multiples réductions récursives. La déficience de rang et le calcul des profils de rang est une spécificité du calcul algébrique que nous explorons en détails. Enfin, le calcul du polynôme caractéristique et de la forme de Frobenius illustre le cas d'une amélioriation de complexité asymptotique se révélant aussi avantageuse en pratique. Nous proposons une parallélisation des ces algorithmes par tâches récursives permettant de préserver les performances asymptotiques. Des implantations efficaces du vol travail de tâches récursives et basées sur des dépendances par flot de données, permettent d'atteindre des performances similaires aux meilleures bibliothèques numériques, et passant à l'échelle.

Dans le contexte du calcul distribué à grande échelle, la fiabilité des ressources distantes est mise en question. Pour remédier aux erreurs, d'origine soit physique soit malicieuse, nous proposons des schémas de tolérance aux fautes algorithmiques (ABFT) reposant sur les techniques d'évaluation-interpolation communes en calcul formel. Les codes correcteurs classiques (Reed-Solomon et codes CRT) basés sur l'interpolation sont étudiés et généralisés au cas des fractions rationnelles et leur entrelacement. Nous introduisons par ailleurs les codes d'interpolation creuse dont l'étude des capacité des correction peine à révéler leur efficacité en pratique. Ces travaux ouvrent en outre des perspectives sur des techniques de décodage symbolique-numérique, supportant conjointement les erreurs et le bruit numérique.

**Abstract**

This manuscript presents contributions on high performance algebraic computations, lying at the interface between computer algebra, coding theory and parallel computing.

Exact linear algebra, over a finite field or the field of rationals is a core component of many applications using intensive algebraic computations: from cryptanalysis based on algebraic sieves or polynomial system solving, to testing conjectures in computational number theory or list decoding... Development of both algorithmic and efficient implementations in exact linear algebra has now reached a great level of maturity. We survey the milestones in these recent progresses, illustrated by our contributions, trying to exhibit a few general guidelines, as for example the importance of making theoretical algorithmic reductions effective. Matrix multiplication, combining conversions between modular, integral and floating point arithmetic, the use of numerical BLAS, and of sub-cubic algorithms, with low memory footprint, is used as a building block. Gaussian elimination, harnesses its efficiency through many recursive reductions. Rank deficiency and rank profile computations is a specificity of exact computations that we explore in details. Lastly the computation of the characteristic polynomial and of the Frobenius normal form illustrates how an asymptotic improvement of the complexity can be made practical to achieve the best performance. We propose a paralellization of these algorithms based on recursive tasks maintaining the best asymptotical performances. Efficient implementations of work-stealing schedulers handling recursive tasks with dataflow dependencies allow to reach performances similar to state of the art numerical libraries with good scaling ability.

In the context of large scale distributed computing, the reliability of remote resources is in question. In order to support errors, either due to physical alterations or malicious corruption, we propose an algorithm based fault tolerance (ABFT) based on evaluation-interpolation schemes, naturally arising in computer algebra. Some classic error correcting codes (Reed-Solomon and CRT codes) based on interpolation are studied and generalized to the case of rational fractions and their interleaving. We also propose sparse interpolation codes, for which the difficult correction capacity analysis hardly reflects their good behavior in practice. These codes open many perspectives, in particular on symbolic-numeric decoding techniques jointly supporting errors and numerical noise.