HAL open science

Algorithmes hybrides et adaptatifs en algèbre linéaire exacte

Anna Urbanska-Marszalek

To cite this version:

Anna Urbanska-Marszalek. Algorithmes hybrides et adaptatifs en algèbre linéaire exacte. Symbolic Computation [cs.SC]. Grenoble University, 2010. English. NNT : . tel-02143044

HAL Id: tel-02143044

https://theses.hal.science/tel-02143044v1

Submitted on 29 May 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Université de Grenoble

Thèse

pour obtenir le grade de **Docteur de l'Université de Grenoble**

Spécialité : «**Informatique et Mathématiques Appliquées**»

préparée au **Laboratoire Jean Kuntzmann (LJK)**

dans le cadre de l'**École Doctorale «Sciences et Technologies de l'Information, Informatique»**

Hybrid and adaptive algorithms in exact linear algebra

Présentée et soutenue publiquement par

Anna Marszałek

le 27 avril 2010

Directeurs de thèse

Mme. Dominique Duval,	**Université de Grenoble.**
M. Jean-Guillaume Dumas,	**Université de Grenoble.**

Jury

Mme. Dominique Duval,	**Directrice de thèse,**	**Université de Grenoble, France.**
M. Jean-Guillaume Dumas,	**co-directeur de thèse,**	**Université de Grenoble, France.**
M. Bernard Mourrain,	**Rapporteur,**	**INRIA Sophia, France.**
M. Gilles Villard,	**Rapporteur,**	**CNRS, ÉNS Lyon, France.**

Contents

Contents i

List of Figures vii

List of Tables xi

I Introduction 2

1 Motivations 3
1.1 Existing Solutions for Linear Algebra . . . 4
1.2 Goals of Middleware . . . 5
1.3 Algorithms in LinBox . . . 6
1.3.1 Ring Implementations . . . 6
1.3.2 Matrices . . . 6
1.3.3 Algorithm Types . . . 7
1.3.4 Modular Algorithms . . . 7
1.3.5 Integer Algorithms . . . 7
1.3.6 New Additions . . . 8
1.4 Hybrid and Adaptive Algorithms . . . 8
1.4.1 Origin of Hybrid Algorithms . . . 9
1.4.2 Definition of Hybrid Algorithms . . . 9
1.4.3 Classification . . . 9
1.4.4 Examples . . . 10
1.4.5 Remarks on Classification . . . 11
1.5 Outline of Thesis . . . 12

2 Expected Complexity 13
2.1 *Big Oh*,*Soft Oh* and *Semisoft Oh* Notations . . . 14
2.1.1 Introduction . . . 14
2.1.2 Classic Definitions . . . 15
2.1.3 *Soft* and *Semisoft Oh* notation . . . 16
2.1.4 *Soft* and *Semisoft Oh* Notation in Several Variables . . . 17
2.2 Computational Models . . . 20
2.2.1 Practical Remarks . . . 21

2.3 Complexity of Adaptive Algorithms . . . 22
2.3.1 Output-dependent Complexity . . . 22
2.3.2 Expected Complexity . . . 23
2.3.3 First Appearance of the Notion of Expected Complexity . . . 24
2.3.4 Expected and Output-dependent Complexity in Practice . . . 25

3 Adaptive Smith Form Algorithm 27
3.1 Presentation of Algorithm cf. [44] . . . 27
3.2 Comparison with other Smith Form Algorithms . . . 28
3.3 Adaptive Modifications . . . 29
3.4 Expected and Output Dependant Complexity . . . 30
3.5 Summary . . . 33

II Adaptive Determinant Algorithm 35

4 Motivations 37
4.1 Existing Algorithms . . . 37
4.2 Outline of this Part . . . 39

5 Probabilistic Properties of Random Matrices 41
5.1 Basic Notions for Random Matrices . . . 41
5.1.1 Introduction . . . 41
5.1.2 Basic Notion . . . 42
5.1.3 Random Matrices . . . 45
5.2 Rank of Random Matrices . . . 48
5.2.1 Main Results . . . 48
5.2.2 Comparison with Existing Results . . . 54
5.3 The Expected Number of Non-trivial Invariant Factors . . . 54
5.3.1 Case Modulo p . . . 55
5.3.2 Case of Dense Matrices . . . 59
5.3.3 Case of Sparse Matrices . . . 67
5.4 The Determinant of Random Matrices . . . 71
5.4.1 Preliminary Results . . . 71
5.4.2 The Determinant of a Random Matrix . . . 77
5.4.3 Comparison with [10] . . . 87

6 Preconditioned Chinese Remaindering Algorithm 89
6.1 Classic Chinese Remaindering Algorithm . . . 89
6.1.1 Requirements for CRA . . . 89
6.1.2 Reconstruction and Termination in CRA . . . 90
6.1.3 Modular Determinant Computation . . . 92
6.1.4 The Number of Steps in the CRA Loop . . . 93
6.2 Preconditioned Chinese Remaindering Algorithm . . . 93
6.2.1 Correctness of the Algorithm . . . 94
6.2.2 The Complexity of the Algorithm . . . 95

6.3 Preconditioning in Abbott's at al. Algorithm . . . 96
6.3.1 Correctness of Preconditioner . . . 96
6.3.2 Computation of Preconditioner . . . 97
6.3.3 Probability of LIF Algorithm . . . 98
6.3.4 Algorithm of Abbott et al. . . . 100
6.3.5 Choosing a Solver . . . 100
6.3.6 Complexity of the Determinant Algorithm of Abbott et al. . . . 102

7 Bonus and Extended Bonus Idea 105
7.1 New Preconditioning of Saunders and Wan . . . 105
7.1.1 Properties of Preconditioning . . . 106
7.2 Extended Bonus Idea . . . 108
7.3 Last k Invariant Factors . . . 108
7.4 Expected Overestimation . . . 109
7.5 Correctness of k-Bonus Algorithm . . . 111
7.5.1 Cost of Bonus Computation . . . 112

8 Adaptive Determinant Algorithm 115
8.1 General Idea and Components of the Algorithm . . . 115
8.2 Requirement of the Algorithm . . . 116
8.2.1 Evaluation of r_{max} - the Number of Iterations . . . 117
8.2.2 Summary - Parameters of the Algorithm . . . 118
8.3 Main Parts of Algorithm . . . 119
8.4 Procedure . . . 120
8.5 Correctness of Algorithm . . . 120
8.6 Expected Behavior of Algorithm . . . 122
8.7 Complexity of Algorithm . . . 123
8.7.1 Output Dependent Complexity . . . 123
8.7.2 Expected Complexity . . . 124
8.7.3 Worst Case Complexity . . . 124
8.8 Experimental Evaluation . . . 125
8.8.1 Generic Case . . . 125
8.8.2 Malicious Matrices . . . 126
8.9 Further Introspective Modifications of Algorithm . . . 128
8.10 Beyond Random Dense Matrices - Application to Structured Matrices . . . 130
8.10.1 Sparse Case . . . 130
8.10.2 Toeplitz and Hankel Matrices . . . 132

III Adaptive Rational Algorithms 133

9 Motivations 135
9.1 Matrix Storage . . . 136
9.2 Modular Computation for Rational Matrices . . . 136
9.3 Outline of this Part . . . 137

10 Chinese Remaindering Algorithm - Survey **139**
10.1 CR Algorithm . . . 139
10.2 Image of a Rational Matrix . . . 141
10.3 Iteration in CRA . . . 142
10.4 Reconstruction in CRA . . . 143
10.4.1 Incremental Reconstruction . . . 146
10.4.2 Delayed Reconstruction . . . 146
10.4.3 Reconstruction in Vector Case . . . 147
10.5 Getting Rational Result . . . 147
10.6 Termination in CR Algorithm . . . 148
10.6.1 Termination in Scalar Case . . . 148
10.6.2 Early Termination in Rational Case . . . 152
10.6.3 Early Termination in Vector Case . . . 153
10.6.4 Extremely High Number of Iterations . . . 154
10.7 In Search for Ideal Reconstruction Strategy . . . 154
10.8 Parallel CRA . . . 156

11 Rational Reconstruction **157**
11.1 Problem of Rational Reconstruction . . . 157
11.2 Existing Algorithms for Rational Reconstruction . . . 157
11.3 Rational Reconstruction in CR Algorithm and p-adic Lifting . . . 158
11.3.1 Probability of Early Termination . . . 159
11.4 Fast Rational Reconstruction - Implementation . . . 159
11.5 Maximal Quotient in Fast Rational Reconstruction . . . 167
11.5.1 Correctness and Additional Cost of Alg. 11.5.1 . . . 170
11.6 Rational Reconstruction - Experimental Results . . . 171
11.6.1 Computation of Threshold . . . 172
11.6.2 Comparison of Wang's and Monagan's Strategies . . . 173

12 Implementation in LinBox **181**
12.1 Integer CRA . . . 181
12.1.1 Rational CRA . . . 183
12.1.2 Implementations of CRABase . . . 185
12.1.3 Implementation of RatRecon . . . 191

13 Preconditioning for Rational Computation **195**
13.1 Obtaining Integer Matrices by Preconditioning . . . 196
13.2 Increase in Norm Due to Preconditioning . . . 198
13.2.1 Case of Random Matrices . . . 199
13.2.2 Examples of Rational Matrices - Experimental Evaluation . . . 203
13.3 Preconditioning of Result - Case Study . . . 206
13.3.1 Approximating Denominators . . . 207
13.3.2 Bounds on Numerators . . . 212
13.4 Quality of Preconditioners . . . 214
13.4.1 Case of Random Matrices . . . 215

13.4.2 Examples of Rational Matrices - Experimental Evaluation 220

14 Adaptive Rational Algorithms **225**
14.1 Adaptive Rational Algorithms - Case Study 225
14.1.1 System Solving 226
14.1.2 Determinant 231
14.1.3 Minimal and Characteristic Polynomial 236
14.1.4 Correctness of Rational Algorithms 237
14.2 Complexity Analysis of Alg. 14.1.5 237
14.2.1 General case 238
14.2.2 Complexity in Special Cases 241
14.3 Experiments 242
14.3.1 Homomorphic Imaging 243
14.3.2 Determinant Computation 246
14.3.3 Characteristic Polynomial Computation 254

IV Smith Form by Reduction of Chain Complexes **263**

15 Motivations **265**
15.1 Outline of this Part 266
15.2 Background on Smith Form Theory and Algorithms 267
15.2.1 Theory of the Smith Form 267
15.2.2 Historic Background 267
15.2.3 Algorithms in Spacial Cases 269
15.3 Background on Homology Theory 269
15.3.1 Computation of Homologies 270

16 Reduction of a Chain Complex **275**
16.1 Algebraic Reductions of [73] 275
16.1.1 Geometric Interpretation of Reduction 276
16.2 Reduction Algorithm for Matrices 277
16.2.1 Reductions modulo p^l 280

17 Smith Form by Reductions - Heuristic Algorithm **287**
17.1 Introduction 287
17.1.1 Problem Setting 288
17.2 Reduction Algorithm 288
17.2.1 Choice of Admissible Entry 290
17.2.2 Remarks on *Reduce* Procedure 291
17.2.3 Implementation and Complexity Study 292
17.3 Experiments and Results 294
17.3.1 Application to K-Theory 294
17.3.2 Application to Cubical Homology 302

List of Algorithms

3.4.1 Output-dependent version of the Smith form Alg. of [44]. 32
5.4.1 LRE Algorithm of [39] for $\mathrm{ord}_p(\det(A))$ 79
6.2.1 Early Terminated Preconditioned CRA 94
6.3.1 LIF cf. [2, 44, 39, 112] 97
6.3.2 The Algorithm of Abbott et al. [2] 100
7.3.1 k-Bonus Algorithm 109
8.3.1 CRA part of Adaptive Determinant Algorithm 8.4.1 119
8.3.2 LIF&Bonus Part of Adaptive Determinant Algorithm 8.4.1 120
8.4.1 Introspective Determinant Algorithm 121
10.1.1 ET CRA to compute $X(A) \in \mathbb{Q}^s$ 141
11.4.1 PrevEuclideanStep 161
11.4.2 NextEuclideanStep 161
11.4.3 Fast Extended Euclidean Algorithm FEEA cf. [131] 164
11.5.1 Fast Maximal Quotient Rational Reconstruction 168
11.5.2 MaxQ_FEEA 169
14.1.1 ChooseSolver Procedure 229
14.1.2 System solving 231
14.1.3 Correction Procedure 232
14.1.4 ChooseLIF Procedure 234
14.1.5 Determinant 235
17.2.1 Reduction/coreduction scheme for the Smith form computation 289
17.2.2 PartialElimination Algorithm 292

List of Figures

8.1 Comparison of adaptive determinant algorithm with other existing implementation. 126
8.2 Comparison of adaptive determinant algorithm and CRA 127
8.3 Comparison of sparse and dense variants of our determinant algorithm for Trefethen's matrices. 128
8.4 Comparison of sparse and dense variants of our determinant algorithm with the CRA algorithm for random sparse matrices. 128

11.1 Evaluation of optimal T in Alg. 11.4.3 173
11.2 Comparison of timings for classic and FEEA rational reconstruction for Fibbonacci numbers 174
11.3 Timings of early terminated p-adic lifting for matrices D_n 175
11.4 Time of rational reconstruction vs. lifting in the p-adic scheme for matrices D_n 175
11.5 Timings of early terminated p-adic lifting for random matrices 178

12.1 Hierarchy for CRABase 187
12.2 Hierarchy for RReconstructionBase 193

13.1 The growth of norm for preconditioned matrices 202
13.2 Growth of norm for preconditioned Hilbert and Lehmer matrices 203
13.3 Sparsity [in %] and the ratio of non-integer elements to non-zero elements for matrices of BasisLib+ Collection. 204
13.4 Growth of norm size for preconditioned matrices of BasisLib+ Collection. . 205
13.5 Growth of norm size for scaled matrices of BasisLib Collection. 206
13.6 Quality of approximations of denominators for random matrices of decimal fractions 216
13.7 Quality of approximations of denominators for random matrices, $\|A\|_r = 100$ 217
13.8 Quality of approximations of denominators for random matrices, $\|A\|_r = 10000$ 218
13.9 Quality of approximations of denominators for random matrices, $\|A\|_r = MAX\_INT$ 219
13.10 Quality of approximations of denominators for Hilber matrices 221
13.11 Quality of approximations of denominators for Lehmer matrices 221
13.12 Quality of approximations for BasisLib+ collection 222
13.13 Quality of approximations for BasisLib+ collection 222

14.1 Ratio of imaging times $RatIm/IntIm$ for random *Rat* and *RatUni*-type rational matrices. Left: $\|A\|$=100; Right: $\|A\| = 10000$. 244
14.2 Times of $RatIm$ and $IntIm$ for Hilbert (left) and Lehmer (right) matrices. Times in miliseconds are given. Scaling is logarithmic. 245
14.3 Distribution of the ratio of imaging times $RatIm/IntIm$ for BasisLib+ collection. Intger imaging is always better. 245
14.4 Times of rational determinant computation for random matrices of decimal fractions with 5 and 16 decimal places. Average times [in sec.] for 5 random matrices, for strategy PrecBonus and IntroRatDet are given. 248
14.5 Times of rational determinant computation for random matrices of rational fractions for $\|A\| = 100$. Matrices are given according to *Rat* and *RatUni* distribution. Average times [in sec.] for at least 5 random matrices have been computed. Strategy PrecBonus and IntroRatDet are compared. 248
14.6 Times of rational determinant computation for random matrices of rational fractions for $\|A\| = 10000$. Matrices are given according to *Rat* and *RatUni* distribution. Average times [in sec.] for at least 2 random matrices have been computed. Strategy PrecBonus and IntroRatDet are compared. 249
14.7 Times of rational determinant computation for Hilbert matrices. Times [in sec.] of strategy PrecBonus for $m = 50, \ldots, 500$ are given, and approximated for $m > 500$. Average time algorithm IntroRatDet is given. 250
14.8 Times of rational determinant computation for Hilbert matrices. Time [in sec.] of strategy PrecBonus for $m = 50, \ldots, 500$ is given. Optimal times for RatDet and IntroRatDet are given. 251
14.9 Distribution of timings [in sec.] for the rational determinant computation by strategy PrecBonus for BasisLib+ Collection. 251
14.10 Left: distribution of timings [in sec.] for the rational determinant computation by strategy RatDet for BasisLib+ Collection. Right: the distribution of the ratio of timings T(RatDet) / T(PrecBonus); if the ratio is > 1, strategy RatDet is better. 252
14.11 Distribution of timings [in sec.] for the rational determinant computation by strategy IntroRatDet for BasisLib+ Collection. 253
14.12 Left: distribution of the ratio of timings T(PrecBonus) / T(IntroRatDet) for BasisLib+ Collection. Right: distribution of the ratio of timings T(RatDet) / T(IntroRatDet); if the ratio is > 1, strategy IntroRatDet is better. . 253
14.13 Average time [in sec.] of the introspective rational algorithm for characteristic polynomial computation in the case of random matrices of decimal fractions with $k = 5, 16$ decimal places. Scaling is logarithmic. 255
14.14 Average time [in sec.] of the introspective rational algorithm for characteristic polynomial computation in the case of random matrices of type *Rat* and *RatUni*, with $\|A\| = 100$ and $\|A\| = 10000$ resp. Scaling is logarithmic. 256
14.15 Size of consecutive coefficients $\log(D(c_i))$ of the characteristic polynomial P_A, for 100×100 matrix A given according to *Rat* distribution. 257
14.16 Time [in sec.] of the introspective rational algorithm for characteristic polynomial computation for Hilbert matrices. Scaling is logarithmic. 258

14.17 Size of consecutive coefficients $\log(D(c_i))$ of the characteristic polynomial P_A, for H_{50}. 258
14.18 Time [in sec.] of the introspective rational algorithm for characteristic polynomial computation for Lehmer matrices. Scaling is logarithmic. 259
14.19 Size of consecutive coefficients $\log(D(c_i))$ and the difference $\log(D(c_i)) - \log(D(c_i))$ in the case of the characteristic polynomial P_A of L_{50}. 260
14.20 Maximal relative error $\max(Dc_{err}(i)/log(D(c_i)))$ of approximating c_{i-1} by c_i for Lehmer matrices. 260
14.21 Distribution of running times [in seconds] of the characteristic polynomial computation for 96 rational matrices of BasisLib+ collection. 261

16.1 Retraction of a square 276

17.1 Grid structure 293
17.2 Evolution of Ω and rank precomputation for $GL_7(\mathbb{Z})$ matrices 299
17.3 Time of one elimination in Alg. 17.2.1 for $GL_7(\mathbb{Z})$ matrices. 301

List of Tables

5.1 Bounds for $\sum_{p<\lambda} \mathcal{P}(\mathrm{rank}_p(A_{k-j+i}) \leq k-j)$. 64
5.2 Bounds for $f(h, \lambda)$. 65
5.3 Bounds for $p = 2, 3, 5, 7, 11$. 66

8.1 Comparison of the performance of Alg. 8.4.1 with r_{max} set to 1 and 2 on engineered matrices. 127

11.1 Evaluation of optimal T in Alg. 11.4.3 - selected times 172
11.2 Number of iterations in early terminated p-adic lifting for matrices D_n . . . 174
11.3 Number of iterations in early terminated p-adic lifting for random matrices 178
11.4 Time of rational reconstruction vs. lifting in the p-adic scheme - random matrices 179
11.5 Number of iterations of rational CRA loop 180

13.1 11 most difficult examples of collection BasisLib+ based on the norm ratio 206
13.2 11 most difficult examples of collection BasisLib+ based on the ratio $\frac{D_{err}}{\mathrm{bs}(D(\det(A)))}$ 223

14.1 Parameters of 13 most challenging problems of BasisLib+ collection in the case of determinant computation. Times for one system solving to approximate $s_m(\tilde{A})$ (T_{SOLVE}), times for strategy PrecBonus (T_{DIX}) , RatDet - (T_{RAT}, if applicable) and Alg. IntroRatDet (T_{INTRO})are given. All times in seconds. 254
14.2 Computation of characteristic polynomial for rational matrices 262

17.1 Results of the rank and Smith form computation for $GL_7(\mathbb{Z})$ matrices . . . 295
17.2 Approximate running times of Alg. 17.2.1 297
17.3 Times (in sec.) for the computation of homologies for the sets of over 8 million cubes. 303
17.4 Matrix sizes, rank and kernel for P0098 matrices. 303

Part I

Introduction

1

Motivations

Recent development in the field of computer algebra has left us with a handful of algorithms for every sort of problems. These solutions are often based on different principles and methodologies and may have different strong points as well as drawbacks. Three main directions in the development of algorithms may be given:

- the optimization of the worst case asymptotic complexity; the theoretic importance of such algorithms makes them reference points for other solutions; this can also be seen as a perspective for treating larger data than applicable today; worst-case asymptotic analysis is required;
- the optimization of the average running time; this often applies to Las Vegas and Monte Carlo type probabilistic algorithms; such solutions are particulary suitable for repetitive computations but probable performance is often regarded as granted in practice, see remarks in [71]; probabilistic analysis of algorithms' behavior is required;
- the optimization of performance on certain class of inputs; if the class is large enough, such solutions may provide the fastest way to obtain the result for most common inputs; experimental evaluation is required;

With respect to these directions, a user might ask the following questions in order to effectively solve his computational problem.

- is the size of my problem large enough to require running the state-of-the-art asymptotic algorithm?
- to what extend does my computation depend on 'luck' if randomized algorithms are run; is it robust to erroneous results of Monte Carlo subprocedures or to a failure of Las Vegas subprocedure; will I repeat the algorithm for several inputs to enhance the importance of the average complexity?
- does my problem belong to a special case, where a heuristic might work well?

The answers might not be obvious or known to the users, yet the choice of the algorithm will have profound implications on the computation. Therefore, the following question arises. Can all the directions of development be included in one algorithm? If so, can it still be given in an elegant form and kept as simple as possible? In this thesis we will argue that the answer is affirmative and that such solutions can be given in the form of hybrid and adaptive algorithms, see Sec. 1.4 for definition. We will focus on the design of adaptive algorithms and their implementation using a middleware, which was LinBox library in our case. We test thoroughly the performance and compare to other implementations in order to show that we are able to achieve best running times.

1.1 Existing Solutions for Linear Algebra

Nowadays, algebraic computation can be performed at different levels. On the top level, we have vast computer algebra systems such as Magma[1] [9], Maple[2], Mathematica [3] and SAGE [4]. In the case of numerical computation, Matlab [5] can be mentioned. They provide users with the environment and tools for the whole process, starting with matrix (vector, polynomial etc.) creation, by running the computation, and finishing with the presentation of results. The interface is usually easy to use, which makes it convenient for dealing with many small-size problems. But when it comes to big-scale computation, it becomes clear that the environment might not be optimized for performance: design features and the complexity of algorithms might prohibit us from completing the task.

On the lower level, the user may choose to implement the solution of his choice by himself. This with no doubt requires spending a lot of time on implementation and testing apart from actually running the algorithm. Dealing with technical details of implementations can be seen as the biggest challenge of this approach.

Then in between there is the level of *middleware*. The general concept of a middleware is explained for example on the web pages *What is a middleware*[6] and *Middleware Architecture* of Sacha Krakowiak [7]. We will consider it on the example of LinBox [8].

LinBox is a library and, thus, does not propose an environment. Instead, it provides a set of classes and procedures that can be use by the programmer. As a middleware, LinBox can be characterized by the following properties:

- it allows the source code to be transferred between different hardware and architectures and automatically deals with the required changes; this can be extended to the

[1] http://magma.maths.usyd.edu.au/magma/
[2] http://www.maplesoft.com/
[3] http://www.wolfram.com/
[4] http://www.sagemath.org/
[5] http://www.mathworks.com/
[6] http://middleware.objectweb.org/index.html
[7] http://sardes.inrialpes.fr/%7Ekrakowia/MW-Book/
[8] http://www.linalg.org

case of parallel computations , where it can be combined with MPI[9] and Kaapi [10] in the case of parallel computations;

- it provides a uniform interface for low level arithmetic procedures by providing wrappers for low level libraries such as GMP[11], Givaro[12] or BLAS[13], and, optionally, NTL[14] or Lidia [15]; allows for transition of objects between the libraries;
- it offers ready to use high level solutions e.g. for determinant or system solving;
- it can be plugged into Maple and SAGE as an optional packages;
- it maintains a great level of abstraction thanks to generic programming;

See Sec. 1.3 and the references therein for more information.

1.2 Goals of Middleware

We motivate our interest in the use of LinBox by the following arguments. Firstly, the use of its low level procedures simplifies the implementation of new algorithms, which can later be included as high level solutions. Then on the other hand, LinBox library can be plugged into a larger computational system such as Maple or SAGE, which means that it can be used by a less experienced user and for less demanding computations as well. To ensure the success of the library, we propose the following goals for the developers.

- to implement and experimentally evaluate new algorithms;
- to provide automated choices of the best algorithms i.e. hybrid and adaptive solutions;
- to include problem specific solutions for most common problems, to build and verify heuristics;
- to provide "building blocks" i.e. optimized subprocedures that allow to easily implement new algorithms and result with simple code;
- to provide parallel solutions if parallel architectures are available;

[9] e.g. http://www.open-mpi.org/
[10] http://kaapi.gforge.inria.fr/
[11] http://gmplib.org/
[12] http://www-ljk.imag.fr/CASYS/LOGICIELS/givaro/
[13] http://www.netlib.org/blas/index.html, in particular ATLAS, ([5], http://math-atlas.sourceforge.net/), GOTO (http://www.tacc.utexas.edu/resources/software/#blas) or LAPACK (http://www.netlib.org/lapack/) libraries can be used
[14] http://www.shoup.net/ntl/
[15] http://www.cdc.informatik.tu-darmstadt.de/TI/LiDIA/

We would like to present recent developments in LinBox which are meant to deal with these challenges. In particular we are going to focus on the implementation of adaptive algorithms in LinBox. Then we are going to test the library on the exact computation with rational numbers (i.e. rational algorithms).

We propose this test as we strongly believe that rational computation should be the next step for exact linear algebra, due to its importance for the ill-conditioned matrix case, for which exact computation should provide an alternative to numerical routines. As not much is known on about rational computation apart from the fact that the problem size makes it difficult, we are persuaded that adaptive solutions will prove useful here.

Finally, we will show that by including heuristics, LinBox is capable of dealing with large-scale problems, such as the computation of Smith form and the rank for matrices of extreme size.

1.3 Algorithms in LinBox

LinBox library was originated as a set of linear algebra procedures for black box matrices see e.g. [126] and [31]. The design of the library itself, which uses the concepts of archetype and diagramming modelling, is the subject of several papers see e.g. [33, 41]. In the meantime, the library developed beyond the black-box case to include other sparse matrix implementations and algorithms, as well as dense matrix case, see e.g. [109]. Engineered algorithms implemented in LinBox are presented in [127].

1.3.1 Ring Implementations

LinBox provides a wrapper for several modular, integer and rational rings implementation: this includes GMP, Givaro, NTL and LiDIA rings. The wrappers provide uniform access to basic arithmetic procedures and in some cases add new functionality to the ring. Users may also add their own implementation which fulfill the field archetype requirements.

Also, LinBox provides its own modular field implementation, for several modulus sizes e.g. int, short-int etc. and specialized field modulo 2 and 2^{32}. Symmetric finite field implementation is also available. The implementation of modular field using doubles is of particular interest as it allows for effective BLAS computation, see [35, 36, 109].

1.3.2 Matrices

At its origins, LinBox library provided black-box matrices archetype, for which only defining a matrix-vector product was obligatory. During the evolution of the library, several matrix implementation have been added, which include dense matrices (characterized by $O(1)$ access time to elements), Hankel/Toeplitz matrices, and several sparse matrix types (where the storage is proportional to the number of non-zero elements), including the one that enables parallelization.

1.3.3 Algorithm Types

LinBox library focuses on Monte Carlo and Las Vegas type algorithms. In the case of Monte Carlo algorithms, the outputted result is correct with a certain predefined probability $1 - \epsilon$, where ϵ is arbitrary small. On the contrary, Las Vegas algorithms always return correct results, but their running time may vary or the computation may be interrupted with a small probability. We require these algorithms to be probabilistically fast. Las Vegas type algorithms may often output a probably correct result first and then turn to a certification phase to confirm its correctness.

As the aim of LinBox is to provide the fastest solutions possible, a new class of algorithms is emerging, namely the algorithms that are probabilistically fast and probabilistically correct. This is the case of Monte Carlo type adaptive algorithms.

The value of ϵ might as well correspond to the probability that a cosmic ray would interfere and change the result of a deterministic computation, which provokes a philosophical question about the actual difference between deterministic and Monte-Carlo type algorithms. In [71], some motivations of using probabilistic algorithms instead of deterministic one are presented.

1.3.4 Modular Algorithms

In the case of modular computation the methods can be divided according to matrix type into dense and sparse. Sparse methods can further by divided on Wiedemann-Krylov-Lanczos type and elimination methods. The division is true for modular algorithms to compute the rank, determinant, characteristic and minimal polynomial and for system solving.

Dense methods are based on BLAS routines, see [35, 36, 109, 110]. The solution of the above-mentioned problems can be reduced to finding the LU decomposition of a matrix.

Sparse elimination and Wiedemann-Krylov-Lanczos type methods are analyzed and compared in [40]. Solutions to the above-mentioned problems can be obtained from the minimal polynomial (in the case of Wiedemann-Krylov-Lanczos type methods) or from the triangular/diagonal form (in the case of elimination methods).

1.3.5 Integer Algorithms

The common idea of integer algorithms is to repeat modular computation.

In the case of the integer **rank** algorithm, the most effective approach consist of computing the rank of the matrix modulo a random prime p. By repeating the choice of primes several times from a sufficiently large set, we obtain a Monte Carlo type algorithm. If the matrix is of full rank, one random choice of world-size prime could be sufficient to confirm it.

For the **determinant** and **characteristic/minimal polynomial** computation over integers and rationals, Chinese Remaindering Algorithm (CRA) offers a way to compute

the integer result by using modular computation, see Chapter 10 for a survey. In the case of integer **system solving**, another procedure is offered by LinBox, namely the p-adic lifting, see [27].

In both CRA and p-adic lifting procedures, the principle is to get the result modulo a sufficiently large number M. In the case of CRA, the result is computed modulo a product of distinct primes; in the case of p-adic lifting, M is a power of a prime p.

In the case of rational computation, one additionally requires the rational reconstruction procedure, which allows to reconstruct a fraction from its modular image. The procedure is described in details in [70, Sec.5.10], see also Chapter 11.

1.3.6 New Additions

The work on this thesis included the following additions to the LinBox library.

- an adaptive determinant algorithm, which combines Chinese Remaindering with p-adic lifting, see Ch. 8; the implementation of this high level algorithm uses template subprocedures, which can be substituted by code submitted by other developers; in this way new procedures can easily be incorporated in the algorithm;
- a generic implementation of Chinese Remaindering loop, which includes vector case, early termination, rational reconstruction and preconditioning, see Ch. 10;
- the implementation of fast rational reconstruction of [131] and maximal quotient fast rational reconstruction of [90], see Ch. 11;
- a new structure of a grid for sparse matrices, and a heuristic algorithm for the computation of the Smith form and rank for exact sequences of matrices, see Ch. 16;

1.4 Hybrid and Adaptive Algorithms

In every domain of computation, whenever several algorithms can solve the same problem, the choice of the right algorithm can be a way to obtain the best timing. Clearly, the problem space i.e. all problem instances, can be divided into regions, where given algorithms perform best. We will call these regions domains of feasibility of the algorithms.

It is evident that this partition is not known beforehand in every detail. At the same time, the average and worst case asymptotic complexities only give us a clue to identify the best algorithm for a particular problem. The situation is even more complicated if the running time of the algorithm depends heavily on unknown, sometimes output-dependent properties of input and when the difference between worst case and average complexities is significant.

1.4.1 Origin of Hybrid Algorithms

Historically, this was the reason for the appearance of the first adaptive algorithm of Musser [100], for sorting and search. "Common knowledge" and practice show that quicksort, with the average complexity of $O(n \log(n))$ comparisons, is often the fastest sorting algorithm. However, for every choice of the partition element in the recursive call to quicksort, a counterexamples may be shown, on which the running time of the algorithm quadratic, thus reaching its worst-case complexity. Any remedy known, such as calculating the actual median of the series makes the algorithm impractical. On the other hand, heapsort, which has the worst case complexity of $O(n \log(n))$ comparisons, is in practice 2 up to 5 times slower than quicksort on randomized data.

The idea of Musser was to combine both algorithms by limiting the number of recursive calls to quicksort and turning to heapsort on the partially sorted data, when the unpromising case was detected. The resulting algorithm had the average complexity and running time equal to that of quicksort (as it followed actually the same computational path) but at the same time it claimed the same low worst case complexity as heapsort. The author referred to his algorithm as *introspective*. Following the classification of [24] we would prefer the term *adaptive*.

1.4.2 Definition of Hybrid Algorithms

Let us now recall the essential definition after [24].

Definition 1.4.1 (Hybrid Algorithm cf. [24]) An algorithm is called *hybrid* if it combines the ideas of several existing algorithms to deliver a new solution. Every algorithm included can solve the problem on its own.

The goal of this approach is to enhance the performance by giving higher priority to algorithms experimentally proven to be fastest for randomized input. Then the average and/or worst case complexity is boasted by including best asymptotic average and worst case algorithms.

Ideally, a hybrid algorithm should perform at least as well as any of its components i.e. its domain of feasibility should be equal to the sum of the domains of feasibility of its components. In practice, we allow for small (of asymptotically lower or the same complexity) additional cost of identifying and evaluating the switches and require that the hybrid algorithm is fastest on a major part of the sum of the domains of feasibility of its components.

1.4.3 Classification

A hybrid algorithm can also be found under the name of adaptive, tuned, engineered and/or introspective algorithm. The names are used in different contexts by different authors. We adapt the following classification after [24].

Depending on the number of choices or switches between the algorithms, we distinguish *simple* (for $O(1)$ of choices) and *baroque* (for an asymptotically bounded number of choices) hybrid algorithms. The algorithm is *static* or *dynamic* depending whether the choices are defined before the execution or computed at the run time.

Depending on the type of information used we may define:

- *tuned* algorithms: if the choice depends on the architecture. In practice this means that certain parameters are computed at installation or compilation time and determine the computation path of the algorithm for every input instance. Thus, tuned algorithms are static. The parameters are either predefined for architecture types (for *engineered* algorithms) or automatically computed by the algorithm (for *self-tuned* algorithms).

- *adaptive* algorithm: if the algorithm does not depend on architecture but takes into account the availability of resources and, which is the key factor, input and output properties. The algorithm might be called *introspective* if makes the choice based on the information on timing of its components/subroutines. In what follows we would mainly like to focus on this type of hybridization.

- *oblivious* algorithms: if it does not depend on input properties but only on the availability of resources. It might be the case of parallel algorithms which are sensible to the number of idle processors, or cache-sensible algorithms.

1.4.4 Examples

Hybrid algorithms are known to exist in computer algebra for a long time. Indeed, many existing algorithms may be classified as such. Some of the examples were presented by the authors in [24]. Here, we present some cases used in computer algebra libraries that best illustrate the classification.

1. Matrix multiplication by BLAS routines is an example of a tuned algorithm. Hybridization lies in the choice between recursive division on blocks and applying classic matrix multiplication in the opposite case. The goal is to choose the best block size for which to apply the switch. In the case of GOTO and ATLAS libraries this is done at the moment of installation. The matrix multiplication algorithm is either engineered in the case of GOTO library (as optimal parameters are provided for certain architectures) or self-tuned in the case of ATLAS, where the comparison is done by actually running both algorithm at the moment of installation. See [109] and the references therein. See also [25] for further adaptive modifications.

2. In general, using different algorithms for smaller and bigger inputs is the simplest way to introduce hybridization. Integer multiplication in GMP library is another example of successful coupling of four algorithms with different worst-case complexities. See [125] for discussion.

3. Making a choice between sparse and dense versions of the algorithm is an example of static hybridization and a simple hybrid algorithm. Passing a template parameter is a convenient way of introducing a static switch. Actually, one may think of an adaptive choice between dense and sparse matrix representation as it should be possible to determined a threshold in terms of matrix density. In fact, due to fast matrix-matrix and matrix-vector multiplication, dense algorithms usually outperform their sparse variants, with exception to very sparse cases. The amount of memory available is a limiting factor for the dense approach.

   Another variant of the problem is the choice between matrix elimination and Wiedemann-type algorithms for sparse matrices. A choice of parameters and some ideas of a hybrid algorithm are given in [42] in the case of rank computation for $\{0, 1\}$ matrices; see also [40] for another timings comparison.

4. Combining numerical and exact algorithms is another way of of introducing hybridization. As an example, adaptive algorithm for system solving of [128] can be given. For well-conditioned matrices, the algorithm uses a numerical method to get the approximate result, which can then be corrected by symbolic methods. The algorithm detect the case when the matrix is ill-conditioned and can use purely symbolic methods such as p-adic solving of [27] instead the numerical one.

One of the most elaborated adaptive algorithms in up-to-date computer algebra is the algorithm for the computation of the Smith Form, first presented by Eberly et al. in [44] and then developed by Saunders and Wan in [112, 113, 127]. In Ch. 3 we will consider this algorithm in detail. We will propose a new insight into its analysis in terms of the expected and output-dependent complexity (see Ch. 2 for definitions) and provide some further modifications to the algorithm.

1.4.5 Remarks on Classification

The above examples show that hybridization is present in many low level procedures such as BLAS routines and integer multiplication. Same can be said about the use of *oblivious* schemes of parallelization and cache management. Given a more complex hybrid algorithm, which relies on calls to low level hybrid subprocedures, we restrict our interest to its high-level strategic choices and classify it accordingly.

This is justified by the fact that low level procedures are often regarded as oracles - black boxes that provide the answer in certain time. Often, more than one implementation of low level procedures is possible. By doing so, we ignore the fact that any algorithm which uses e.g. ATLAS/GOTO BLAS routines is *tuned* to some extend and as such, depends on the architecture. Yet, we are persuaded that this does not lead to confusion.

Another ambiguity appears in the use of words *engineered* and *introspective* in the classification.In some cases, see e.g. [113], an adaptive algorithm is called *engineered*, as the choices were designed by authors based on theory **and** experiments. In the same manner, the term *introspective* might be associated with every algorithm which makes its decisions

at any point, based on its performance so far. In a modified classification of [24] an algorithm with a priori hand-designed choices could be referred to as *engineered*, contrary to algorithms which make their choices based on timings, which are *introspective*.

1.5 Outline of Thesis

In the rest of Part I, we introduce the measures of performance for hybrid algorithm, giving the notions of output-dependent and expected complexities in Chapter 2. We finish the introduction by a case-study of the adaptive algorithm for Smith form computation of [44, 112] in Chapter 3, for which we propose further modifications.

The thesis itself is divided into three parts. Part II of the thesis is consecrated to the design and evaluation of the adaptive determinant algorithm. Chapter 5 introduces the probabilistic properties of random integer matrices that will be applied in Chapter 8 to produce the adaptive algorithm for the determinant (Alg. 8.4.1). Chapters 6 and 7 present other tools and theorems that will be used in order to construct the algorithm. This algorithm proved very fast in the experiments and is adaptive in every detail, see Sec. 8.8. Its design and the complexity analysis depends strongly on the statistical properties of integer matrices and probabilistic algorithms used.

Part III is consecrated to the rational computation. We start this part by a survey on Chinese Remaindering Algorithm in Chapter 10. Then in Chapter 11 we discuss the algorithms for fast rational reconstruction and present some experimental results. In Chapter 12, the details of CRA and RationalReconstruction implementation in LinBox are presented. In Chapter 13 and 14 we use the tools introduced earlier to construct adaptive rational algorithms for problems including the computation of the determinant, characteristic and minimal polynomial of a matrix and linear system solving. Experimental results are given in Sec. 14.3.

Then Part IV presents the reduction/elimination algorithm for exact sequences of matrices which is a generalized version of the concept of algebraic reductions of [73]. Algebraic reductions were originally designed and applied for the computation of homologies of cubical sets. In Sec. 16.2 we translate the concept to the language of matrices, which allows for some generalizations in Thm. 16.2.2, 16.2.8. Using Alg. 17.2.1, we were able to exactly compute the Smith form of matrices coming from the K-theory, which approach 2,000,000 in size, and compute the ranks of matrices representing cubical sets up to the value 17,906,035. This makes it the biggest rank ever computed by LinBox.

2

Expected Complexity as a Measure of Performance of Complex Algorithms

The performance of an algorithm can be evaluated by estimating the number of operations it performs in the worst case or on average, as a function of the input size. Some authors propose to analyze the number of operations in the best case as well.

Worst case complexity bounds the maximal number of operations that the algorithm has to perform for inputs of given size. In the case when input instances are given with a probability distribution, the average number of operations can be computed over input instances of the same size. This gives the **average complexity**.

Additionally, **best case** complexity can be given, as an approximate number of operations which are sufficient in the case of particular inputs. This information alone is not of particular interest but a more detail analyzes which determine propitious and/or malicious inputs for the algorithm may provide a useful characterization.

When dealing with more complex adaptive algorithm, we would like to introduce other measures of performance, which better reflect the subtle nature of adaptive algorithms, namely output-dependent and expected complexity. In short, **output-dependent** complexity will give the worst case or average number of bit operations, with respect to parameters other than input size. Propitious inputs might be defined with respect to these parameters. It is a way to express the worst case or average complexity of output-dependent algorithms. Examples of its use are numerous, we refer e.g. to [75, 38, 39, 4] for output-dependent early terminated CRA.

Assuming that input instances are given with a probability distribution, the **expected complexity** will evaluate the average number of bit operations under some favorable assumptions that correspond to the case of 'expected' inputs of the algorithm. The notions of expected and average complexity might seem close, but are equivalent only in a special case. In general, we will consider the expected complexity as a way to express the best (or propitious case) complexity. The expected complexity will give a valid estimation of

the number of operations for a large class of inputs. In Sec. 2.3.3 we present examples of the use of **expected complexity** so far, which inspired out approach. See [2, 44, 122].

The number of operations is usually approximated in terms of asymptotic *big Oh* notation O. In Sec. 2.1 we recall the notion of *soft Oh* notation and generalize it to the notion of *semisoft Oh* notation. We provide definitions in the case of one and several variables. For example, to describe the size of an integer matrix $A = [a_{ij}]_{i,j=1..n}$, two parameters n and $\|A\|$ can be used, where n is matrix size and $\|A\| = \max(|a_{ij}|)$ is matrix maximum norm. As a result, meaningful complexity formulas should depend on these two parameters.

In computer algebra it is generally accepted to use the bit complexity model. This means that the number of basic operations (additions, multiplications and shifts) on bits should be counted. We will use the *Random-Access Machine* RAM model of computation, see e.g. [23, 114]. See Sec. 2.2 for references on complexity theory. See Sec. 2.3 for the definitions of output-dependent and expected complexities and Sec. 2.3.4 for a practical approach to its computation.

2.1 *Big Oh*,*Soft Oh* and *Semisoft Oh* Notations

2.1.1 Introduction

Big Oh notation $O(-)$ (know also as Landau, Bachmann-Landau or asymptotic notation) has been used in the computational complexity theory from the beginning of modern computer science. The fundamentals of the notation can be found in virtually any textbook on algorithms and in most textbooks on mathematical analysis. We refer to [83, Sec. 1.2.11], [59, Ch. 9] or [23, Ch. 3] for some accessible references. The notion is usually given for functions of one variable, yet the definition can easily be generalized to the several variables case.

In order to perform a simplified analysis of the complexity of algorithms, *Soft Oh* notation has been introduced by several authors, see e.g. [70, Def. 25.8]. Def. 25.8 of [70] states in short that a function $f(n)$ is $O^{\sim}(g(n))$ if there exists a constant α, such that

$$f(n) \in O\left(g(n)\log^{\alpha}(g(n))\right).$$

Notice, that the definition of $O^{\sim}$ implies conditions on a 'meaningful' functions g. Indeed, $f(n) \in O^{\sim}(n\log(n))$ is equivalent to $f(n) \in O^{\sim}(n)$ in terms of *soft Oh* notation and both do not provide complete information on the logarithmic factors of f. In [70] the authors remark that hiding logarithmic factors can be dangerous if practical applicability of the algorithm is to be considered.

Indeed, on may think of several examples, where the use of *soft Oh* notation should be avoided. First, given two n-bits integers, the complexity of their multiplication, $M(n)$, and gcd computation, $B(n)$, by the state-of-the-art fast multiplication and fast gcd algorithms is the same in terms of *semisoft Oh* notation, see Ex. 2.1.6. Experiments show that the

running times of procedures differ significantly. The difference can be explained by the fact that $B(n) = O(M(n)\log(n))$ see e.g. [70] and [119].

Second, the article of Storjohann [122] bounds several matrix problems to matrix multiplication, thus resulting with several algorithms which all have $O^{\sim}(n^{\omega})$ bit complexity, where ω is the exponent of matrix multiplication. Yet the gradation in the difficulty of the algorithms is still present e.g. the complexity of matrix multiplication is asymptotically better than the complexity of solving a linear system of equations, which in turn is better than the complexity of the determinant computation. In [122], Storjohann shows, that the running times of these algorithm differ by several $\log(n)$ factors.

To cope with these problems we propose to introduce *semisoft Oh* notation, see Def. 2.1.5 and 2.1.8, which gives approximation of the complexity up to doubly logarithmic $\log(\log(-))$ terms. We believe that this approach allows at the same time

- to keep the swell and complexity of expressions at bay,
- to describe the essential behavior of algorithms,
- to easily compare between algorithms given their complexities in terms of *semisoft Oh* notation.

Examples 2.1.6 and 2.1.9 illustrate the use of *soft* and *semisoft* notations.

Notice, that $\log_2(\log_2(N))$ for $N = 2,000,000$ is less than 4.4, i.e a small constant. Moreover, N exceeds by far the size of inputs (matrix or integer size) on which exact computation is performed these days.

2.1.2 Classic Definitions

Let us recall the classic definitions. See e.g. [83, Sec. 1.2.11], [59, Ch. 9] or [23, Ch. 3], [70, Ch. 25].

Definition 2.1.1 (*Big Oh* Notation cf. [59]) Let $a \in \mathbb{R} \cup \{-\infty\}$ and $f, g : (a, +\infty) \to \mathbb{R}$ be two real-valued functions, $g \geq 0$. We say that $f \in O(g)$ (or $f = O(g)$) if there exist constants $N, C \in \mathbb{R}$, $N > a$,$C > 0$ such that for all $n \geq N$

$$|f(n)| \leq C \cdot g(n).$$

Definition 2.1.2 (*Big Oh* in Several Variables cf. [59]) Let $k \in \mathbb{N}$ and $a_i \in \mathbb{R} \cup \{-\infty\}$ for $i = 1..k$. Let $F, G : \prod_{i=1}^{k}(a_i, +\infty) \to \mathbb{R}$ be two real-valued functions of k arguments, $G \geq 0$. We say that $F \in O(G)$ (or $F = O(G)$) if there exist constants $C, N_1, N_2 \ldots N_k \in \mathbb{R}$, $N_i > a_i$ for $i = 1, \ldots, k$, $C > 0$ such that for all $n_i \geq N_i$, $i = 1, \ldots, k$

$$|F(n_1 \ldots n_k)| \leq C \cdot G(n_1 \ldots n_k).$$

2.1.3 *Soft* and *Semisoft Oh* Notation for Complexity Expressions

In the theory of complexity, the number of operations is often approximated as $O(g)$ where g is a complexity expression given in a form of a product

$$g(n) = n^{\alpha_1} \log^{\alpha_2}(n) \dots \log^{\alpha_s} \underbrace{(\log(\dots \log(n) \dots))}_{s-1}$$

for certain $s \in \mathbb{N}, s > 0$, $\alpha_i \geq 0$, $i = 1, \dots, s$, where $\log = \log_2$ is the binary logarithm. Hereby, we will assume that log denote the binary logarithm, although all logarithm bases $b > 1$ are equivalent in *big Oh* notation i.e $\log(n) \in O(\log_b(n))$ and $\log_b(n) \in O(\log(n))$.

In some cases, it is convenient to forget the smallest logarithmic terms. In this way, *big Oh* notation can be generalized to *soft Oh* and *semisoft Oh* notations. First, let us introduce the following notation.

Definition 2.1.3 (Logarithmic Term) Let $s \in \mathbb{N}, s \geq 0$. We recursively define a function ${}^s\log$, called the logarithmic term of order s. For $s = 0$ we put $a_0 = 0$ and

$${}^0\log : (a_0, +\infty) \ni n \to n \in \mathbb{R}_+.$$

Then for $s > 0$ we define $a_s = \min(a' :^{s-1} \log(a') \geq 1)$ and

$${}^s\log : (a_s, \infty) \ni n \to \log({}^{s-1}\log(n)) = \underbrace{\log(\log(\dots \log(n) \dots))}_{s} \in \mathbb{R}_+.$$

For $s_1, s_2 \in \mathbb{N}$ $0 \leq s_1 < s_2$ we say that the logarithmic term ${}^{s_1}\log$ is **more significant** than ${}^{s_2}\log$.

We have the following definitions.

Definition 2.1.4 (*Soft Oh* Notation, cf. Def. 25.8 [70]) Let $\alpha \in \mathbb{R}$, $\alpha > 0$ and $s \in \mathbb{N}$, $s \geq 0$. A function $f : \mathbb{R} \ni n \to f(n) \in \mathbb{R}$ is $O^{\sim}(({}^s\log(n))^{\alpha})$ if there exists a parameter $\gamma \geq 0$, such that $f(n) \in O(({}^s\log(n))^{\alpha}({}^{s+1}\log(n))^{\gamma})$.

Definition 2.1.5 (*Semisoft Oh* Notation) Let $\alpha, \beta \in \mathbb{R}$, $\alpha > 0, \beta \geq 0$ and $s \in \mathbb{N}$, $s \geq 0$. A function $f : \mathbb{R} \ni n \to f(n) \in \mathbb{R}$ is $O^{\approx}(({}^s\log(n))^{\alpha}({}^{s+1}\log(n))^{\beta})$ if there exists a parameter $\gamma \geq 0$, such that

$$f \in O(({}^s\log(n))^{\alpha}({}^{s+1}\log(n))^{\beta}({}^{s+2}\log(n))^{\gamma}).$$

In this way, *soft Oh* notation allows us keep only the most significant logarithmic term while *semisoft Oh* notation requires keeping two most significant terms.

Example 2.1.6 1. *Fast integer multiplication*

The cost of fast Schönhage & Strassen integer multiplication of two n bit integers is

$$M(n) \in O(n \log(n) \log(\log(n))),$$

see [70, Tab. 8.6]. By definition 2.1.4 and 2.1.5 this is $O^{\approx}(n \log(n))$ and $O^{\sim}(n)$.

If the integers are bounded by N, the complexity is

$$M(\log(N)) \in \begin{cases} O(\log(N) \log(\log(N)) \log(\log(\log(N)))) \\ O^{\approx}(\log(N) \log(\log(N))) \\ O^{\sim}(log(N)) \end{cases} . \tag{2.1}$$

2. *Fast integer gcd*

The cost of fast gcd algorithm for two n bit integers is

$$B(n) \in O(M(n) \log(n)), \tag{2.2}$$

see e.g. [119] and the references therein. By definition 2.1.4 and 2.1.5 this is $O^{\approx}(n \log^2(n))$ and $O^{\sim}(n)$.

2.1.4 *Soft* and *Semisoft Oh* Notation for Complexity Expressions in Several Variables

In the case of several variables, complexity expressions are usually given as a multivariate polynomial G of logarithmic terms i.e.

$$G(n_1, \dots n_k) = P({}^{s_1}\log(n_1), \dots, {}^{t_1-1}\log(n_1), \dots, {}^{s_k}\log(n_k), {}^{t_k-1}\log(n_1)),$$

where $t_i > s_i \geq 0$ for $i = 1, \dots, k$ and P is a polynomial of $\sum_{i=1}^{k}(t_i - s_i)$ variables.

As before, *soft Oh* and *semisoft Oh* allow us to keep, respectively, only one or two most significant terms.

Definition 2.1.7 (*Soft Oh* in Several Variables, cf. Def. 25.8 [70]) Let $k \in \mathbb{N}$. For $i = 1, \dots, k$ let $s_i \in \mathbb{N}$ be such that $s_i \geq 0$. Let $P(x_1, \dots, x_k)$ be a polynomial of k variables, such that $\deg_{x_i}(P) > 0$ for $i = 1, \dots, k$. Let G be a function such that

$$G(n_1, \dots n_k) = P({}^{s_1}\log(n_1), \dots, {}^{s_k}\log(n_k)).$$

A function $F : \mathbb{R}^k \ni (n_1, \dots n_k) \to F(n_1, \dots n_k) \in \mathbb{R}$ is $O^{\sim}(G)$ if there exists a polynomial P' of $2k$ variables, function G' s. t.

$$G'(n_1, \dots n_k) = P'({}^{s_1}\log(n_1), {}^{s_1+1}\log(n_1), \dots, {}^{s_k}\log(n_k), {}^{s_k+1}\log(n_k)),$$
$$P(x_1, \dots x_k) = P'(x_1, 1, \dots x_k, 1)$$

and

$$F(n_1, \ldots, n_k) \in O(G'(n_1, \ldots, n_k)).$$

Definition 2.1.8 (*Semisoft Oh* in Several Variables) Let $k \in \mathbb{N}$. For $i = 1, \ldots, k$ let $s_i \in \mathbb{N}$ be such that $s_i \geq 0$. Let $P(x_1, y_1, \ldots, x_k, y_k)$ be a polynomial of $2k$ variables, such that $\deg_{x_i}(P) > 0$ for $i = 1, \ldots, k$. Let G be a function such that

$$G(n_1, \ldots n_k) = P({}^{s_1}\log(n_1), {}^{s_1+1}\log(n_1), \ldots, {}^{s_k}\log(n_k), {}^{s_k+1}\log(n_k)).$$

A function $F : \mathbb{R}^k \ni (n_1, \ldots n_k) \to F(n_1, \ldots n_k) \in \mathbb{R}$ is $O^{\approx}(G)$ if there exists a polynomial P' of $3k$ variables, such that

$$\begin{aligned} G'(n_1, \ldots n_k) = P'({}^{s_1}\log(n_1), {}^{s_1+1}\log(n_1), {}^{s_1+2}\log(n_1), \ldots, \\ {}^{s_k}\log(n_k), {}^{s_k+1}\log(n_k), {}^{s_k+2}\log(n_k)), \end{aligned}$$

$$P(x_1, y_1, z_1 \ldots x_k, y_k, z_k) = P'(x_1, y_1, 1, \ldots x_k, y_k, 1)$$

and

$$F(n_1, \ldots, n_k) \in O(G'(n_1, \ldots, n_k)).$$

Example 2.1.9 shows how the *soft Oh* and *semisoft Oh* notations can be used in complexity considerations in order to

- to take into account the speedup resulting from fast integer multiplication and gcd computation, see Ex. 2.1.6,
- to avoid the swell of complexity formulas due to less significant logarithmic terms.

Example 2.1.9 For the rest of this example, let A be an $n \times n$ integer matrix with entries bounded by magnitude by $\|A\| = \max_{i,j=1,\ldots,n}(|a_{ij}|)$ and let b be a $n \times 1$ vector, with entries bounded in magnitude by $\|b\| = \max_{i=1,\ldots,n}(|b_i|)$. In what follows we may assume that $\|A\|$ and $\|b\|$ are related to each other e.g. both values are bounded by certain $\gamma > 0$. The complexity of multiplication of two $n \times n$ matrices mod a word-size prime is $O(n^\omega)$, where ω is 3 for the classical algorithm, and 2.38 for the Coppersmith-Winograd method, see [22].

During the course of various linear algebra algorithms, some arithmetic operations in $\mathbb{Z}$ may be performed on the numbers, whose size is $O(\log(\det(A)))$ i.e $O(n\log(n\gamma))$. This motivates the following two examples.

1. Let us consider fast integer multiplication of two $O(n\log(n\gamma)))$ bit integers, where $n, \gamma > 0$ are certain parameters. By Eq. (2.1) the complexity of multiplication is

$$M(n\log(n\gamma)) \in \begin{cases} O^{\approx}(n\log(n\gamma)(\log(n) + \log(\log(\gamma)))) \\ O^{\sim}(n\log(\gamma)) \\ n \times O^{\sim}(\log(n)\log(n\gamma)) \end{cases} . \tag{2.3}$$

PROOF It suffices to notice that $\log(n\log(n\gamma)) = \log(n) + \log(\log(n\gamma))$ and

$$\log(\log(n\log(n\gamma))) = O(\log(\log(n)) + \log(\log(\log(n\gamma)))).$$

2. Let us consider fast gcd computation of two $O(n\log(n\gamma)))$ bit integers, where $n, \gamma > 0$ are certain parameters. By Eq. (2.2) the complexity of the gcd computation is

$$B(n\log(n\gamma)) \in \begin{cases} O^{\approx}(n\log(n\gamma)(\log^2(n) + \log^2(\log(\gamma)))) \\ O^{\sim}(n\log(\gamma)) \\ n \times O^{\sim}(\log^2(n)\log(n\gamma)) \end{cases} . \tag{2.4}$$

PROOF As before, $\log(\log(n\log(n\gamma))) = O(\log(\log(n)) + \log(\log(\log(n\gamma))))$. We have $\log^2(n\log(n\gamma)) = \log^2(n) + 2\log(n)\log(\log(n\gamma)) + \log^2(\log(n\gamma))$ and

$$2\log(n)\log(\log(n\gamma)) \leq \begin{cases} 2\log^2(n) & \log(n) \geq \log(\log(n\gamma)) \\ 2\log^2(\log(n\gamma)) & \log(n) < \log(\log(n\gamma)) \end{cases}$$

which leads to a conclusion that $\log^2(n\log(n\gamma)) \in O(\log^2(n) + \log^2(\log(n\gamma)))$.

This results may be used to evaluate the complexity of X-adic linear solvers that exist in the literature.

3. *Classis p-adic solver of Dixon [27]*

   Let us consider p-adic lifting algorithm of [27] for matrix A and vector b. The cost of p-adic lifting is $O\left(n^3\log^2(n\|A\|p) + n\log^2(\|b\|)\right)$, see [97]. We assume that p is a word-size prime i.e that $\log(p) \in O(1)$ and that $\log(\|b\|) \in O(n\log(n\|A\|))$. Then the complexity of the algorithm is $O^{\sim}(n^3\log^2(\|A\|))$.

4. FastRationalSolver of [95]

   Let us consider Alg. *FastRationalSolver* of [95] for matrix A and vector b, modulo $q = p^s$, where p is a word-size prime, $p \in O(1)$, and $s > 0$. Suppose that $\|A\|$ and $\|b\|$ are both bounded by γ. In [95, Lem. 5.7] the cost of the algorithm is $O^{\sim}(n^{\frac{3+3\omega-\omega^2}{4-\omega}}\log(\gamma))$ in the *soft Oh* notation. Let us compute the missing logarithmic factors.

   Following [95, Lem. 5.7] let us assume that $l = n^{\frac{3-\omega}{4-\omega}}$ and $(n\gamma)^l \leq q < p(n\gamma)^l$, which means that $\log(q) = O(l\log(n\gamma))$. The cost of the initialization phase is dominated by the computation of $B = A^{-1} \mod q$ and integer matrix multiplication of BA. By [124] the cost of inverse computation is $O(n^\omega B(\log(q)))$, where B is the cost of gcd-like operations, see Eq. (2.4). By Eq. (2.4) this is

$$\begin{aligned} O(n^\omega B(l\log(n\gamma))) &= O^{\approx}(n^\omega l\log(n\gamma)(\log(l) + \log(\log(\gamma)))) \\ &= O^{\approx}(n^{\omega+\frac{3-\omega}{4-\omega}}\log(n\gamma)(\log(n) + \log(\log(\gamma)))). \end{aligned}$$

The cost of one lifting step is dominated by matrix multiplications, which by [95, Eq. 6] gives $O((\frac{n}{l})^2 l^\omega M(\log(n\gamma)))$ bit operations. The cost of remaining arithmetic operations is $O(B(n\log(n\gamma)))$. There are $O(\frac{n}{l})$ steps in the lifting phase, which means that the overall cost is bounded by

$$O((\frac{n}{l})^3 l^\omega B(\log(n\gamma))) = O^{\approx}\left(n^{3+\frac{(\omega-3)(3-\omega)}{4-\omega}}\log(n\gamma)\log(\log(\gamma))\right).$$

We comparing the complexities of the initialization and lifting phases we conclude that the complexity of the algorithm is

$$n^{\frac{3+3\omega-\omega^2}{4-\omega}} \times O^{\sim}(\log(n)\log(n\gamma)),$$

and is dominated by the cost of inverse computation.

This illustrates the danger of using $O^{\sim}(-)$ notation in application to the algorithm design. As the result, the two parts of the algorithm of [95] are not well balanced and in fact, the initialization phase dominates the cost, which could be counterintuitive to some extend.

5. *Fast-multiplication Rational Solver of [122, Sec. 9.1]*

   Let us consider rational solver from [122, Sec. 9.1] for matrix A and vector b. [122, Thm. 37] gives the complexity of the algorithm equal to $O(n^\omega \log(n)B(\log(\|A\|) + \log(\|b\|)/n + \log(n)))$.

   Let us assume that $\log(\|b\|) = O(n\log(n\|A\|))$. By Eq. (2.4), the complexity of the algorithm is $O^{\approx}(n^\omega \log(n)\log(n\|A\|)\log^2(\log(\|A\|)))$ bit operations which is also $n^\omega \times O^{\sim}(\log(n)\log(n\|A\|))$.

2.2 Computational Models

The first step in the theory of complexity is the choice of the computational model. There are several possibilities that differ mostly in the level of formalism, which includes Random-Access Machines (RAM) [23, 114] or different types of Turing Machines (deterministic, non-deterministic, multi or single tape etc.), see [108, 114, 66, 111, 6, 57]. Additionally, in the case of parallel computations, the PRAM model, see e.g. [114], should be mentioned. See also [114] for a model of a simple CPU. We refer to [23, Sec. 2.2] for a brief introduction to the RAM model and to [114] for a profound study of most widely used computational models. A computational model allows to compute the number of operations performed by the algorithm on a given problem instance. A function, which bounds the number of operations for all inputs of given size is the worst case complexity.

According to Church-Turing Thesis [6, 111], virtually all 'reasonable' computational models of algorithms are equivalent and lead to the same complexity classes. Non-equivalent models can also be consider e.g. boolean circuits [114, 6, 57, 111], but this is behind the

scope of this work. The study of computational models is of main importance in the theory of decidability, see e.g. [66] and while considering the hierarchy of complexity classes e.g. the famous NP vs. P problem. We refer to [51, 108, 115, 114, 66, 111, 6, 57] for definitions and examples of problems.

Computational models allow to define the average complexity, as long as the problem is distributional i.e. problem instances are given according to a probability distribution. See [23, Ch. 5] for a simple introduction to the problem in the case of RAM. See also [8] for a survey on average complexity of NP algorithms. Complexity classes can be defined as well in the case of average complexity, although the definitions is not as natural as in the case of worst case complexity. See [115, 6, 111, 56] for the approach of Levin [86] and [57] (cf. typical average complexity) for another approach to the hierarchy of average complexity classes.

For randomized algorithms we first refer to [67, 92] for a presentation of their design, examples and ideas for the analysis of performance. In particular [67, Ch. 2] presents models of randomized algorithms and discuss the notions of worst case and average complexities in this case. Following [67, Sec. 2.3] we propose to regard a randomized algorithm as a family of probabilistically distributed algorithm instances and propose the following classification of complexities:

- worst case complexity, which gives the upper bound on the number of operations performed by any instances the algorithm on any input of the given size;
- worst case average complexity, which gives the upper bound on the number of operations performed by the algorithm on average (over all algorithm instances) on any input of the given size;
- average worst case complexity, which gives the average number of operations (over all inputs) for the worst case instance of the algorithm for inputs of the given size;;
- average complexity, which gives the average number of operations (over all algorithm instances, over all inputs) for inputs of the given size;

See also [66, 6, 57] for the hierarchy of complexity classes in this case randomized algorithms.

2.2.1 Practical Remarks

Computability, decidability and class hierarchy is behind the scope of this thesis. In fact, through this thesis we will consider computable problems of mainly polynomial time and space algorithms. In Sec. 2.3 we focus on providing meaningful expressions characterizing the asymptotic performance of (polynomial) algorithms. Thus, we would like to present two less standard ways of approximating the number of operations performed by the algorithm.

First, we propose to relax the requirement that complexity is given solely in terms of the input size, see Def. 2.3.2 of the output-dependent complexity. Second, we would like

consider only a sufficiently large subclass of propitious inputs, see Def. 2.3.6. This is not the (typical) average complexity of [57] as ignored input is not formally negligible. Still, by Prop. 2.3.7, expected complexity can be seen as the output dependent complexity in the expected case.

In order to define the output-dependent and expected complexities, we need a computational model of algorithm $\mathcal{A}$ for which the following concepts are defined.

- *input:* at the entrance of algorithm $\mathcal{A}$, input $x \in \mathcal{I}$ is given;
- *size:* function $size(x)$ gives the size of input x;
- *operation counting:* an operation counting function $C_{\mathcal{A}}(x)$ gives the number of operations performed by $\mathcal{A}$ on input x;
- *input distribution:* additionally, a distribution on $\mathcal{I}$ can be defined; then, inputs are given according to the distribution;
- *instances of randomized algorithms:* If $\mathcal{A}$ is a randomized algorithm, it is given as a family $\mathcal{A}_\xi$ of algorithm instances given according to a probability distribution ξ;

Without loss of generality we may further assume that function $size : \mathcal{I} \ni x \to size(x) \in \mathbb{N}$ returns integer values. In a more general setting, $size$ must define a partition of $\mathcal{I}$ e.g. in the case of integer matrices, $size(M)$ can be equal to a pair $(n, \|M\|)$, where n is matrix dimension and $\|M\|$ is the maximal entry in absolute value. However, if matrices of word-size integers are considered, putting $size(M) = n$ is sufficient for complexity considerations.

2.3 Complexity of Adaptive Algorithms

2.3.1 Output-dependent Complexity

The notions of output dependent and expected complexities can be defined in any model of computation and for any complexity type (worst case or average, or for parallel and randomized algorithms). It make the complexity formulas depend on input parameter x. This approach has been used for a long time, where relevant, see e.g. early terminated CRA in [75, 38, 39, 4]. Let us give the following definitions.

Definition 2.3.1 (Output-dependent Trait) Let $\mathcal{I}$ be the set of inputs to algorithm $\mathcal{A}$. An output-dependent trait K on $\mathcal{I}$ is a function $K : \mathcal{I} \ni x \to K(x) \in \mathbb{R}_+$ which is independent on the size of x. That is, for every $n \in \mathbb{N}$ there exist two inputs $x_1, x_2 \in \mathcal{I}$ such that $size(x_1) = size(x_2) = n$ and $K(x_1) \neq K(x_2)$.

Definition 2.3.2 (Output-dependent Complexity) Let $\mathcal{A}$ be an algorithm with input domain $\mathcal{I}$. Let K be an output dependent trait on $\mathcal{I}$.

Then function $\mathcal{C}_{\mathcal{A}}^{K} : \mathbb{N} \times \mathbb{R}_{+} \to \mathbb{N}$ gives the **output-dependent** worst case [on K] complexity of $\mathcal{A}$ if

$$\mathcal{C}_{\mathcal{A}}^{K}(n,k) \geq C_{\mathcal{A}}(x) \quad \forall x \ s.t. \ size(x) = n, K(x) \leq k. \tag{2.5}$$

Here, $C_{\mathcal{A}}(x)$ denote the number of operations performed by $\mathcal{A}$ on input x.

If, additionally, entries of $\mathcal{A}$ are given according to a probability distribution, then function $\overline{\mathcal{C}}_{\mathcal{A}}^{K}$ gives the **output-dependent** average [on K] complexity of $\mathcal{A}$ if

$$\overline{\mathcal{C}}_{\mathcal{A}}^{K}(n,k) \geq \mathbf{E}_{x:size(x)=n,K(x)\leq k} C_{\mathcal{A}}(x). \tag{2.6}$$

Example 2.3.3 The determinant, rank, number of non-trivial invariant factors for a $n \times n$ integer matrix A are examples of output-dependent traits.

Remark 2.3.4 In the case of randomized algorithms, worst case average output dependent complexity can be defined by replacing $C_{\mathcal{A}}(x)$ in Eq. (2.5) with the average number of operations performed by the randomized algorithm $\mathcal{A}$ on x. Similarly, average worst case and average complexities can be defined by extending Eq. (2.6) to the case of randomized algorithms.

In the case of randomized algorithms, a trait may also be a function of algorithm instances e.g. the relative error $\frac{\tilde{s}_n}{s_n}$ in is a trait for the randomized algorithm [2, Alg. LIF]. Here, s_n is the largest invariant factor of a matrix, and $\tilde{s}_n$ is its computed approximation.

Remark 2.3.5 The use of the term "output" is well explain in the case where K is exactly the problem for which algorithm $\mathcal{A}$ has been designed. More generally, K might be a by-product of $\mathcal{A}$, and eventually it may be computed by subroutines of K. Surely, all output is "input" depended and thus, output dependent complexity is just a restatement of the worst case or average complexity.

2.3.2 Expected Complexity

To define the expected complexity, let us assume that a random distribution is defined on the set of inputs $\mathcal{I}$, and that a trait K, not necessarily output-dependent, is given.

Definition 2.3.6 (Expected Complexity) Let $\mathcal{A}$ be an algorithm with input domain $\mathcal{I}$. Let a random distribution be defined on the set of inputs $\mathcal{I}$ and let K be a trait on $\mathcal{I}$. Let $K(n) \in R_{+}$ be the bound on the expected value $\mathbf{E}_{x:size(x)=n} K(x)$. Then $\mathcal{EC}_{\mathcal{A}}^{K}(n)$ is the **expected complexity** of algorithm $\mathcal{A}$ iff

$$\forall x, size(x) = n : K(x) \leq K(n) \Rightarrow C_{\mathcal{A}}(X) \leq \mathcal{EC}_{\mathcal{A}}^{K}(n)$$

In the case of randomized algorithms, K might be a function of randomized algorithm instances as well, see Rem. 2.3.4. In short, the expected complexity describes the number of operations in the case when randomly distributed elements of the algorithm perform 'as expected'. The use of a bound $K(n)$ on the expected value is motivated by

- the fact that usually only a bound on the expected value can be found;
- the requirement that the expected complexity should give the upper bound on the number of operations for a large class of inputs; various theoretical results such as Markov inequality of Chebyshev inequality can be used here;
- the possibility to use the bounds $K(n)$ in the design of an adaptive algorithm, as we will show in Ch. 8.

In the case, when both output dependent and expected complexities can be defined, we have a simple relation.

Proposition 2.3.7 (Output-dependent vs. Expected Complexity) Let $\mathcal{A}$ be an algorithm with input domain $\mathcal{I}$. Let a random distribution be defined on the set of inputs $\mathcal{I}$ and let K be an output dependent trait on of $\mathcal{I}$. Suppose that there exist a constant A_n such that

$$K(x) \leq A_n \quad \forall x \in \mathcal{I} \ s.t. \ size(x) = n.$$

Then the expected complexity of $\mathcal{A}$ can be defined and is equal to $\mathcal{EC}^K(n) = \mathcal{C}^K(n, K_n)$, where K_n is such that $\mathbf{E}_{x,size(x)=n}K(x) \leq K_n$.

PROOF The expected value $\mathbf{E}_{x,size(x)=n}K$ is less than or equal to A_n. Therefore the expected complexity can be defined. Moreover, suppose that K_n is the bound on $\mathbf{E}_{x,size(x)=n}K$. Let us take $x \in \mathcal{I}, size(x) = n$ such that $K(x) \leq K_n$. Then $C_{\mathcal{A}}(x)$ is less than $\mathcal{C}^K(n, K_n)$ by the definition of output-dependent complexity, see Def. 2.3.2. By Def. 2.3.6 we may set $\mathcal{EC}^K(n) = \mathcal{C}^K(n, K_n)$.

2.3.3 First Appearance of the Notion of Expected Complexity

It is common in computer algebra that an algorithm contains a great number of calls to randomized subroutines, which makes the average case analysis complicated. Often, the algorithm depend on existing procedures and on previous results about its complexity and probability of correctness.

One of the methods to compute the average complexity is to compute the expected values for some random variables and to analyze the number of operations of the algorithm using these values. The use of expected values is reflected in the name of the complexity, where the word 'average' is replaced by 'expected'. The confusion is due to the fact that the word 'average' is a synonym of the expected value. Consequently, the expected running time is a generally accepted synonym of average complexity. In computer algebra, one can find other terms such as average expected complexity ([2]) or expected cost ([44, 122]) and it is worth to consider what kind of complexity are the authors actually referring to according to our classification.

Remark 2.3.8 In view of Def. 2.3.6, the use of the term 'average expected' complexity becomes redundant and should be depreciated. Averaging is required from definition for the computation of expected complexity.

Let $\mathcal{A}$ be randomized algorithm. One may compute

- worst case expected complexity - if the expected value is computed over random algorithm instances $\mathcal{A}_\xi$ according to the distribution ξ.

  Examples include the *LargestInvariantFactor* or LIF algorithm of Abbott et al. [2] or algorithms based on shifted number systems of Storjohann [122]. The determinant algorithm of [2] has the **expected** output-dependent bit complexity of $O\big(n^\omega \log\left(|\frac{\det(A)}{s_n}|\right) + n^3 \log^2(n\|A\|)\big)$. This result uses the fact that the **expected** value $\mathbf{E}\left(log\left(\frac{s_n}{\tilde{s}_n}\right)\right)$ is $O(1)$, where $\tilde{s}_n$ is the factor of s_n obtained by LIF algorithm. In [122] the expected number of calls to a randomized subroutine in order to obtain a correct (certified) result is approximated by $2 \in O(1)$. In those cases, expected complexity and average complexity are the same.

- expected worst case or expected complexity - assuming that a random distribution is defined on input $\mathcal{I}$, the expected value is computed over all input instances. This has been done for example in [44], where the expected complexity was expressed in terms of the expected number of non-trivial invariant factors.

2.3.4 Expected and Output-dependent Complexity in Practice

The expected complexity might depend on many traits. Practical approach to the expected complexity is based on common practice for the computation of average complexity. First, the expected outputs of subprocedures should be bounded (e.g. $\mathbf{E}\left(log\left(\frac{s_n}{\tilde{s}_n}\right)\right)$ is $O(1)$ in [2]). Then, the expected values of parameters, which arise naturally during the course of the algorithm (e.g. the number of non-trivial invariant factors in [44]) can be approximated.

At this point, the evaluation of expected complexity can be simplified when compared to the average complexity. Contrary to the case of average complexity evaluation, according to Def. 2.3.6, we are allowed to use the bounds as if they were the actual results (e.g. consider the situation when the expected number of invariant factors is indeed smaller to the bound of [44, Thm. 6.2]). The performance of the algorithm on inputs and instances, which violate the expected value bounds, can simply be ignored and do not contribute to the evaluation. In this way, the analysis of complex algorithms with many calls to randomized subprocedures can be simplified.

Moreover, this allows us to use the bounds in the design of switches of the adaptive algorithm, which results in non-linear dependency on the expected values. This approach try to optimize the expected complexity. The switches can be re-evaluated and slightly modified, using results such as Markov or Chebyshev inequalities to optimize the average complexity, but the analysis is more complex in this case.

The computation of expected complexity is not equivalent the average complexity, unless the dependency on the random variable is convex (recall that $E(f(X)) \leq f(E(X))$ if function f is convex), which usually makes it linear in the case of complexity functions.

However, we can imagine more complex cases of adaptive algorithms where the relation between average and expected complexity is not straightforward, in which case the expected complexity provides additional information. In Ch. 3 we give an example of an algorithm which has better expected than average complexity.

At the same time, output dependent complexity provides vital information, as it describes the actual number of operations *a posteriori* performed by the algorithm. On the other hand, we believe that the strength of expected complexity lies in the elegance of its estimation. This becomes essential and evident when it comes to adaptive algorithms.

The principle of adaptive algorithms is to take advantage of a priori slower (by means of the worst case and average complexity) algorithms by detecting propitious inputs instances for which the running time of the algorithms is faster. The motivation behind this approach is that those propitious inputs occur very often for the inputs we are dealing with. At the same time, all malicious inputs are detected and treated in the best possible (asymptotic) way.

The applicability of adaptive algorithms can be estimated by relating propitious inputs to the 'better than expected' case. Namely, an adaptive algorithm should perform well on the 'better than expected' inputs, ideally on all of them. By Def. 2.3.6, this would be reflected in the expected complexity of the algorithm. Therefore, it implies that this could be a good measure of performance of adaptive algorithms.

In our work we have observed, that the analysis of worst case and average complexity of an adaptive algorithm can often hide the idea that stands behind it. Although worst case and average complexities traditionally characterize the algorithm, the knowledge of the performance in the propitious cases seems at least as important. Therefore we would suggest that adaptive algorithms should be compared in terms of output-dependent and expected complexities.

In Ch. 3 we show how to analyze the output dependent and expected complexity on the example of the adaptive Smith form algorithm of [44]. Based on this analysis, we show some simple modifications that lead to better expected complexity estimations and actually reduce the number of operations.

3

Adaptive Smith Form Algorithm of Eberly, Giesbrecht and Villard - Case Study

The Smith form algorithm of [44] computes the Smith form and determinant of a square invertible matrix by computing all of its invariant factors. It has been subject to modifications in [112, 113, 127]. In this chapter we show some small modification of the original algorithm, which lead to better bounds on the expected complexity, see Def. 2.3.6.

3.1 Presentation of Algorithm cf. [44]

Let $i, n \in \mathbb{N}$ and let $i \leq n$. Let us now assume that a $n \times n$ non-singular matrix A is given at the input of the Smith form algorithm of [44]. Let $\|A\| = \max_{i,j=1..n}(|A_{ij}|)$ be the maximal entry in absolute value of the matrix. The algorithm requires the following two procedures.

1. *LargestInvariantFactor (or LIF)*, see [2, 44] and Alg. 6.3.1, which computes the largest factor s_n of a $n \times n$ non-singular matrix. It is a Monte Carlo type algorithm, that always returns a divisor of the actual largest invariant factor. See [2, Lem. 1,2] and [44, Thm. 2.1] for some results on the probability of success of the LIF algorithm. Notice, that in Thm. 6.3.2 we present some additional results for this problem.

2. *OneInvariantFactor(i) (or OIF(i))*, see [44], which computes the ith invariant factor of the matrix. In [44, Lem. 3.1] the authors show that by preconditioning the input matrix A a new matrix $C_i = A + B_i$ can be obtained such that $s_n(C_i)$ divides $s_i(A)$ if B_i has rank $n - i$. By taking random preconditioners one obtains a Monte Carlo type algorithm to compute OIF. The output is a multiple of the actual factor s_i as long as LIF procedure used to compute $s_n(C_i)$ returns the correct value. In general, this cannot be assumed.

Once the LIF and OIF procedures have given us way to compute any invariant factor of A, the authors propose to use the division property $s_i \mid s_{i+1}$ to limit the number of calls. This property implies that whenever $s_i(A)$ and $s_j(A)$ are equal, there is no need to compute $s_k(A)$ for $i < k < j$ as it the same value. The approach the authors propose is equivalent to a binary search for different invariant factors. Therefore they are able to state in [44, Thm. 4.2] that their algorithm outputs the Smith form with probability 0.5 using at most $O(n^{3.5}\log^{2.5}(n\|A\|)\log^2(n)$ bit operations in the worst case.

At the same time the bound on the expected number of non-zero invariant factors is given in [44, Cor. 6.3], which leads to the average complexity $O(n^3\log^2(n\|A\|)\log^2(n))$ bit operations.

In [44] the authors assume that LIF computation is done for a dense matrix A by the p-adic lifting of [27]. However, the use of other linear solvers can be envisaged. In general, the worst case complexity of the algorithm is $O(\sqrt{\log(|\det(A)|)}C(LIF)\log^2(n))$ bit operations, where $C(LIF)$ stands for the cost of one linear system solving. Taking a p-adic solver of [27] in the LIF computation results in $C(LIF) = O(n^3\log^2(n\|A\|))$ and the total cost of the algorithm is $O(n^{3.5}\log^{2.5}(n\|A\|)\log^2(n))$.

Let $B(x)$ denote the cost of gcd-like operations on x-bit integers. The usage the best worst-case complexity solver of [122] gives $C(LIF) = O(n^\omega\log(n)B(\log(n\|A\|)))$ and the cost of the algorithm becomes $O(n^\omega(\log(n\|A\|))\sqrt{|\log(\det(A)|)}\log^3(n))$.

The algorithm can also be applied to the sparse matrix case by using the sparse solver of [43], which gives $C(LIF) = O(\Omega n^{1.5}\log(n\|A\|) + n^2\log(n\|A\|)\log(\|A\|))$ if the matrix has $\Omega \geq n$ non-zero entries. The complexity of the algorithm becomes $O((\Omega n^{1.5}\log(n\|A\|) + n^2\log(n\|A\|)\log(\|A\|))\sqrt{|\log(\det(A)|)}\log^2(n))$.

For well-conditioned matrices, adaptive solver of [128] provides another opportunity of using combined symbolic-numerical methods to solve a system of equations.

3.2 Comparison with other Smith Form Algorithms

The complexities of other Smith Form algorithms are as follows. n is the matrix size and $\|A\| = \max_{i,j=1\ldots n}(|a_{ij}|)$. See Sec. 2.1 for the definition of $O^\sim$ notation.

- $O^\sim(n^5\log^2(\|A\|))$ for the Iliopoulos [69] algorithm, if fast integer multiplication is used; the computation is performed modulo the determinant or the largest invariant factors which is bounded in size by $O(n\log(n\|A\|))$;
- $O^\sim(n^{\omega+1}\log(\|A\|))$ for the Storjohann [121] algorithm; ω is the exponent for matrix multiplication; fast integer multiplication can be used;
- $O^\sim(sn^3)$ [by elimination methods] or $O^\sim(sn\Omega)$ [by black-box methods] for the valence algorithm [39]; s is the number of primes dividing the valence; Ω is the number of non-zero entries of the matrix; the algorithm is practical if s is small;

3.3 Adaptive Modifications

The first adaptive modification is mentioned by the Eberly et al. in their original paper. Other were introduced in a series of papers of Saunders and Wan, see [112, 113, 127].

1. The algorithm *FastInvariantFactors* of [44] consists of coupling the original *Invariant-Factors* algorithm with the algorithm of Storjohann [121]. Thanks to this approach the authors are able to deliver an algorithm with worst case complexity better than both components. The complexity of this adaptive solution is $O\left(n^{2+\frac{\omega}{2}} \log^{1.5}(n\|A\|) \log^{0.5}(n)\right)$.

2. In a series of papers [112, 113, 127] Saunders and Wan introduce even more adaptivity to the Smith form algorithm. The improvements include a "bonus" computation and new multiplicative preconditioning. Then the authors suggest dividing the computation to smooth and dense part, which refer to the computation of local Smith forms at smaller and bigger primes respectively. They combine the valence algorithm [39] (which provides the small primes to consider) with local form computation (for each small prime) and use the binary or backward search only to look for invariant factors which include big primes. The valence and local Smith form algorithms are particulary efficient on certain class of sparse matrices. The use of Iliopoulos [69] and Storjohann [120] algorithm can also be envisaged in this approach.

   The speed-up of [112] is mostly practical, as their approach allows first of all to reduce the leading coefficient of the complexity, due to smaller number of iterations needed to obtain the result with a sufficient probability of success compared to the original algorithm. Moreover, the authors try explore all known and implemented algorithms, which is the essence of adaptive programming.

3. In some cases, computing local Smith forms for a reasonable number of small primes $2, 3, 5, 7, \ldots$ can be more efficient than the computation of valence. This is especially true as algorithm of [39] depends heavily on the assumption that the valence is small. The following paradox occurs: the more big primes divide the valence, the less information we will use and the longest its computation. Additionally, taking into account that rough factors are usually restricted to the largest invariant factor, see [44, Thm. 6.2] and Cor. 5.3.3, this leads to a conclusion, that valence computation might be dropped from the algorithm.

   The computation of the smooth part by local form computation and the largest invariant factor computation are sufficient to produce the Smith form, that for random dense matrices is correct with certain probability. The probability can be increased by including more primes in the smooth part, leading to a Monte Carlo type algorithm. The computation of the rough part (or to be precise, confirming that it is trivial) can be seen as a certification of the Monte Carlo algorithm. We have used this modification of [112] to perform computation for extremely large sparse matrices in [34] and 16.

3.4 Expected and Output Dependant Complexity and Other Modifications

In this section we will analyze the algorithm of [44] in the context of the output dependent and expected complexities introduced in Sec. 2.3.1 and 2.3.2. This analysis yields that it should be possible to change a $\log(n)$ factor in the expected complexity of [44] to a $\log(\log(n))$. This would require a small modification in the algorithm and a careful analysis.

In [44, Thm. 4.2] the authors analyze the worst case complexity of their algorithm which is $O(n^{3.5}\log^{2.5}(n\|A\|)\log^2(n))$. The analysis yields

1. $m = O(\sqrt{\log(\det(A))}) = O(\sqrt{n\log(n\|A\|)})$ factors have to be found among n elements. This requires computing $m\log(n)$ factors[1] in what is equivalent to a binary search algorithm;

2. Each factor is computed by repeating OIF procedure $1 + \lceil\log(n)\rceil$ times;

In [44, Thm. 6.4], $m\log(n)$ is replaced by the expected number of non-trivial invariant factors, which gives the expected complexity.

Here we would like to show, that while knowing the expected number of non-trivial invariant factors we may slightly modify the algorithm in order to improve its expected and/or average performance. Then we will show how to make the algorithm sensitive to the number of non-trivial invariant factors and obtain a good output-dependent and expected complexity. The essential observation is that if the number of invariant factors to evaluate is small, less repetition of OIF is needed to obtain the same probability of success.

1. **A simple modification to enhance the expected complexity**

   Let $\lambda \in \mathbb{N}, \lambda > 1$. Let A be a $n \times n$ matrix with entries chosen uniformly and randomly from the set of λ consecutive integers. Then the expected number of non-trivial invariant factors is at most

   $$N = 3(\log_\lambda(n)) + 32, \tag{3.1}$$

   see [44, Cor. 6.3]. This bound is in fact further improved in Thm. 5.3.10 but this is not important in this considerations.

   Assume that the number of non-trivial invariant factors is the expected i.e. it is bounded by N. We can verify the hypothesis by computing the $(n-N-1)$th factor. If it is trivial, the binary search should be done among $N = O(\log_\lambda(n)) = O(\log(n))$ elements. There are at most N factors to compute, which allows to lessen the number of repetitions for

[1] Care should be taken that no more than m different values of invariant factors are computed by the algorithm. Theoretically, in the case of erroneous computation, $OIF(i)$ for $i = 1, \ldots, n$ can be all different from each other. This may violate the worst case complexity bounds. Finding more than m different values of invariant factors means that the computation of the Smith Form is erroneous and that an error has to be reported.

the OIF procedure. If the number of non-trivial invariant factors is greater than N, we run the original algorithm.

Thus, the expected complexity of the algorithm is

$$O\left(n^3 \log^2(n\|A\|)N\log(N)\right) = O^{\approx}\left(n^3 \log^2(n\|A\|)\log(n)\right),$$

as the number of evaluated factors is at most N and the number of repetitions for each OIF procedure is $\log(N)$.

If λ is now known beforehand and no additional information on A is given, we may still anticipate the expected number of non-trivial invariant factors by putting $\lambda = 2\|A\|$. The hypothesis is verified during the course of the algorithm, therefore our assumption does not influence the correctness of the result of the algorithm. However, in the propitious case when the expected number of non-trivial invariant factors is smaller than we assume, the speed-up over the classic version of the algorithm is evident.

However, the expected complexity we give here cannot be considered as the average complexity since we do not average over all possible inputs and the dependency on the expected number of non-trivial invariant factors is not linear and is $N\log(N)$. Thus, let us evaluate the average complexity in detail.

Let us assume that $\lambda \in O(1)$. Let $I(j)$ be the event that number of non-trivial invariant factors is at least j. Then the average complexity of the above version of the algorithm is less than

$$\mathcal{P}(\neg I(N+1))O^{\approx}\left(n^3 \log^2(n\|A\|)\log(n)\right) + \mathcal{P}(I(N+1))O(n^{3.5}\log^{2.5}(n\|A\|)\log^2(n))$$

The value of $I(N+1)$ can be computed thanks to [44, Cor. 6.3], which gives $\mathcal{P}(I(N+1) \leq \lambda^{-n} + 9\left(\frac{2}{3}\right)^N + n^3\left(\frac{1}{\lambda}\right)^N$. By Eq. (3.1), N is such that $n^3\left(\frac{1}{\lambda}\right)^N = O(1)$. Therefore, the average complexity is equal to the worst case complexity in this case.

Yet, we may improve the average complexity by replacing N by kN, where $k > 1$, and validating that $(n - kN - 1)$th factor is trivial. This time the probability $I(kN+1)$ is $O(n^3\left(\frac{1}{\lambda}\right))^{kN} = O(n^{-3(k-1)})$, which means that the average complexity is asymptotically the same as the expected one.

As we can see, parameter k arises in the algorithm from purely probabilistic considerations and is introduced to the algorithm in an arbitrary way. Yet its influence cannot be neglected, as it increases the actual running time of the algorithm k-folds in the propitious case.

2. **Making algorithm output dependent**

In Alg. 3.4.1 we would like to propose another modification of the algorithm, which do not require the knowledge of the expected number of non-trivial invariant factors, but allows to compute it during the course of the algorithm.

Theorem 3.4.1 (Complexity of Alg. 3.4.1) *Let $n \in \mathbb{N}$ and let A be a $n \times n$ integer matrix, such that $\frac{\log^{0.5}(\|A\|n)\log^2(n)}{n^{0.5}}$ is $O(1)$. Let N be the number of non-trivial invariant*

Algorithm 3.4.1 Output-dependent version of the Smith form Alg. of [44].

Require: a $n \times n$ matrix A,
Require: procedure $OIF(j)$, to compute the jth invariant factor of A with probability at least 0.5,
Ensure: Smith Form of A correct with probability at least 0.5;

```
Compute s_1 = gcd(a_ij) and s_n = OIF(n);
if s_1 ≠ 1 then A = A/s_1; s_n = s_n/s_1;
i = 0, S = {n};
while s_{n-2^i+1} ≠ 1 do
    i = i + 1;
    if (n - 2^i + 1 ≤ 1) then break;
    Update s_j for j ∈ S by running OIF(j) once;
    Compute s_{n-2^i+1} by running OIF(n - 2^i + 1) once;
    S = S ∪ {n - 2^i + 1};
    Update s_{n-2^i+1} by running OIF(n - 2^i + 1) i times;
end while
Find smallest j s.t. s_{n-2^j+1} = 1 or j = ⌈log(n)⌉ + 1;
Do binary search for different invariant factors between s_{n-2^j+1} and s_n (by running OIF (j + 1) times);
```

factors of A. Alg. 3.4.1 returns the Smith Form of A with probability 1/2 in the (average) output dependent complexity of $O(N\log(N)C(OIF))$ bit operations, where $C(OIF)$ is the complexity of OIF procedure required by the algorithm.

PROOF If N is the number of non-trivial invariant factors then $s_{n-N-1} = 1$. Let $k = \lceil \log(N) \rceil + 1$. Then $s_{n-2^k+1} = 1$ as well and the loop stops as soon as $s_{n-2^k+1} = 1$ is computed correctly. If this happens after $k + l$ iterations, $l = 0, 1, \ldots,$ then $OIF(n - 2^k + 1)$ procedure was already lanced $(k + l)$ times and produced (k+l-1) erroneous outputs. In total OIF procedure was lanced $(k + l)(k + l + 1)$ times at this stage. Therefore the average number of calls to OIF is at most

$$k(k+1) + \sum_{l=1}^{\infty}(k+l)(k+l+1)\left(\frac{1}{2}\right)^{k+l-1} \leq k(k+1) + 14$$

This means that on average, the loop will stop after $O(k^2)$ iterations and that the binary search has to determine at most $(2^k - k)$ remaining factors by repeating OIF $(k + 1)$ times for each of them. Then, the binary search requires $(k + 1)(2^k - k)$ calls to OIF and the total average number of calls to OIF is $O(k2^k)$.

The case when s_{n-2^k+1} is never computed correctly may occur with probability at most $\left(\frac{1}{2}\right)^{\log(n)}$, as a break is forced in $(\lceil \log(n) \rceil + 1)$th iterations. The classic algorithm of [44] is run in this case, which requires at most $O(\sqrt{\log(|\det(A)|)}\log^2(n)) = O(n^{0.5}\log^{0.5}(n\|A\|)\log^2(n))$ calls to OIF.

This forces us to

$$\left(\frac{1}{2}\right)^{\log(n)} n^{0.5}\log^{0.5}(n\|A\|)\log^2(n)C(OIF) = O(\frac{\log^{0.5}(\|A\|n)\log^2(n)}{n^{0.5}}C(OIF)).$$

to the average complexity. The average running time is thus

$$O(k2^kC(OIF) + \frac{\log^{0.5}(\|A\|n)\log^2(n)}{n^{0.5}}C(OIF)),$$

which is $O(k2^kC(OIF))$ since $\frac{\log^{0.5}(\|A\|n)\log^2(n)}{n^{0.5}} \in O(1)$ by assumption. The Smith form is computed correctly with probability at least $1 - 2^k\left(\frac{1}{2}\right)^{k+1} = \frac{1}{2}$. Since $\log(N) + 2 > k > \log(N)$, the average output dependent complexity is $O(N\log(N)C(OIF))$ bit operations.

The expected complexity can easily be obtained by replacing N with the expected number of non-trivial invariant factors. The condition on matrix A can be lessen and becomes $\frac{\log^{0.5}(\|A\|n)\log(n)}{n^{0.5}} \in O(1)$. The expected complexity is $O^{\approx}(C(OIF)\log(n))$ by Eq. (3.1).

The average complexity of the algorithm can be expressed be the formula

$$\sum_{j=1}^{n}\mathcal{P}(I(j+1)\backslash I(j))j\log(j)n^3\log^2(n\|A\|),$$

which is by far more difficult to evaluate.

3.5 Summary

We presented two modifications which have better expected case complexity than the original algorithm. The second leads to an output-dependent modification in Alg. 3.4.1. We show that in order to obtain a good average complexity an unnatural modification of parameters might be necessary, which leads to worse over-all performance. In general, we feel that from the practical point of view it might be sufficient to optimize the expected complexity instead of the average in view of the following;

- the expected values estimations are crude, see new approximations in Thm. 5.3.10 among others;
- once propitious cases are recognized, the expected complexity approximates the actual number of bit operations better than the average complexity;
- in the malicious cases, the reason of bad behavior is known and the worst case complexity approximates the number of bit operations in this case; again average complexity does not provide vital information in this case;

Giving complexity bounds cannot change the actual running time and cannot replace the experimental evaluation of the algorithms. However, we are persuaded that the analysis of the expected and output-dependent complexity, coupled with the worst case complexity considerations, yields more information on the performance of the algorithm than the average complexity.

Part II

Adaptive Determinant Algorithm

4

Motivations

The problem of determinant computation is one of the best studied in modern computer algebra. The wide availability of algorithms, which are based on many different ideas, as well as the existence of many implementations and experimental evaluations makes it an ideal target for the design of an adaptive algorithm.

We propose an *adaptive* algorithm, which combines different algorithms of various time and space complexities. The algorithm is *introspective*, as it takes into account the actual timing of its components and based on this comparison, it can emphasize a particular variant. With the use of very fast modular routines for linear algebra, our implementation is an order of magnitude faster than other existing implementations.

We present our algorithm in a general form, suggesting that various 'building-block' procedures can be used in order to obtain the final result. Modular determinant computation and integer system solving are the main components of our algorithm. We propose sparse and dense variants of the algorithms, whereas its applicability can also be envisaged in the case of structured matrices. The algorithm is output dependent on the number of non-trivial invariant factors, which leads to a good expected complexity in the case of random dense matrices.

4.1 Existing Algorithms

Before we proceed with the description of our adaptive algorithm for the determinant computation, let us recall the existing solutions and compare their complexities and requirements.

Let $n \in \mathbb{N}, n > 0$ and let A be a $n \times n$ integer matrix, $\|A\| = \max_{i,j=1..n}(|A_{ij}|)$ be the maximum norm of A. For simplicity, let us suppose that $\|A\| \in O(1)$ and analyze the asymptotic bit complexity of the algorithms in terms of matrix size n. We assume that A is a dense matrix and that dense matrix multiplication over modular field can be performed in $O(n^\omega)$ bit complexity. The value of ω is 3 for the classical algorithm, and 2.375477 for the Coppersmith-Winograd method, see [22].

Over a modular field, the computation of determinant is tied to that of matrix multiplication via block recursive matrix factorizations [68]. Therefore, using BLAS routines, such as LU factorization, we obtain an optimal algorithm of $O(n^\omega)$ complexity.

A classical method to find an integer determinant $\det(A)$ is to use the Chinese Remaindering (CR) Algorithm and reconstruct the result from its modular images using the Chinese remaindering theorem, see e.g. [70, Sec.5.4,5.5] for the description of the algorithm. An output-dependent CR algorithm has the complexity of $O^\sim(\log(|\det(A)|)n^\omega)$ bit operations, where $\log(|\det(A)|)$ is $O(n\log(n))$ in the worst case.

In [2] Abbott et al. give another variant of the algorithm in which a divisor D of $\det(A)$ is computed, and the CRA loop is run to compute $\frac{\det(A)}{D}$ instead of $\det(A)$, which in a propitious case may be significantly smaller. This requires adding a modular division in the CRA loop. In [2] the authors propose to use the largest invariant factor of A, i.e. s_n, as the preconditioner D. The algorithm does not improve the worst case complexity when compared to CRA, which remains $O(n^{\omega+1}\log(n))$. The average complexity of the algorithm is $O(n^3\log^2(n) + n^\omega\log(|\det(A)/s_n|))$, assuming that a p-adic solver of [27] is used in the computation of s_n. Yet, no convincing estimations of $\log(|\det(A)/s_n|)$ have be made. However, the value seems to be $O(1)$ in many cases, see [103]. See 6 for the full analysis of the algorithm.

In Ch. 3 we presented the Smith form algorithm of [44], see also Sec. 3.1 for the complexities. The worst case complexity of the algorithm is $O(\sqrt{\log(|\det(A)|)}C(LIF)\log^2(n)$, where $C(LIF)$ is the cost of one integer system solving.

The authors propose to use the Smith form algorithm for the determinant computation as $\det(A)$ is equal to $\prod_{i=1}^n s_i(A)$, the product of invariant factors of A. From the point of view of the determinant computation, the algorithm of [44] has considerable drawbacks. First, all factors are computed at approximately the same cost independently of its size. Secondly, the probability of correctness of the result is small, namely 0.5, and improving the probability requires rerunning the algorithm. As the authors remark, there are ways to falsify and/or certify the result. However, no possibility exists to correct the result by integer CRA as in the case of Abbott's algorithm [2], which in a way is a correction of Pan's algorithm [103]. Even if the computed result differs from the determinant by several primes and is "almost" correct, it may have additional (more often) as well as missing (in an unlucky case) primes. An erroneous result can *a priori* be a factor or a multiple of the determinant (or something in between).

The following algorithms have cubic or sub-cubic complexities, especially if fast matrix multiplication is applied. First, a modified version of [44] algorithm, see Sec. 3.3, has $O\left(n^{2+\frac{\omega}{2}}\log^2(n)\right)$ bit complexity.

Then, the division free algorithms developed by Kaltofen in [75, 78, 79] can be mentioned. For dense matrices, the algorithm has worst case complexity of $O^\sim\left(n^{2.7}\right)$, which is $O^\sim\left(n^{3.2}\right)$ without fast matrix multiplication, by [79]. The sensitivity of the algorithm to the output properties can also be taken into account, see [75]. The algorithm has never been experimentally evaluated.

On the other hand, Last Vegas type (certified) algorithm of Storjohann [122] uses an expected number of $O^{\approx}\left(n^{\omega}\log^{3}(n)\right)$ bit operations, thus binding the integer determinant computation to matrix multiplications. The algorithm requires a fast linear system solver, which is also presented in [122] and requires $O^{\approx}\left(n^{\omega}\log^{2}(n)\right)$ bit operations. Up to date no implementation of such solver has been presented and the algorithm seems to be a purely theoretical improvement. While using a p-adic solver of [27], the determinant algorithm reaches the complexity of $O^{\approx}\left(n^{3}\log^{2}(n)\right)$ bit operations.

4.2 Outline of this Part

In order to design an adaptive algorithm, a study of probabilistic behavior of its input should be performed, which allows to determine a large set of propitious inputs for the algorithm. On the other hand, our algorithm will use several randomized procedures, whose probabilistic behavior has to be analyzed. In Chapter 5 we present a study of probabilistic properties of random matrices, that we will use in order to design and analyze our algorithm.

We present the CRA and Abbott's [2] ideas in Chapter 6, where we introduce a more general concept of preconditioned CR loop and the necessary facts and notations. In Ch. 7 we look again on the ideas of [112, 113, 127] and generalize it to the concept of an extended k-bonus. Finally, in Ch. 8 we present our adaptive algorithm and analyze its performance based on complexity estimations and experiments.

5

Probabilistic Properties of Random Matrices

In this chapter we provide a model and tools to analyze probabilistic properties of the adaptive determinant algorithm, which we present in Ch. 8. In Sec. 5.1 basic tools and the probabilistic model are presented. In Sec. 5.2 we compute the probability that a rank mod p, where p-prime, is equal to a given r in the case of non-uniformly distributed matrices. In Sec. 5.3 the result is applied to the computation of the expected number of non-trivial invariant factors of an integer matrix. This result was first given in [44], here we provide an asymptotically better bound and refer to results in the sparse case. We finish the chapter by presenting the probability bound that p^s divides the determinant of an integer matrix in Sec. 5.4.

5.1 Basic Notions for Random Matrices

5.1.1 Introduction

In order to analyze probabilistic behavior of matrix algorithms, a distribution of matrices and/or vectors given on input has to be considered. The distribution of matrices may arise naturally given a distribution of its entries and assuming that all entries are chosen independently of each other. These kinds of distribution have been considered by several authors and in different contexts in recent years:

- in [10], the authors consider uniformly distributed random dense matrices over $\mathbb{Z}_q$, $q \in \mathbb{N}$ and give probability characterization of rank and determinant of such matrices,
- in [2], the authors consider uniformly distributed random dense integer vectors and the induced distribution over $\mathbb{Z}_q$, $q \in \mathbb{N}$, which allows them to analyze the expected performance of their *LargestInvariantFactor* algorithm,
- in [44], the authors consider uniformly distributed random dense integer matrices and the induced distribution over $\mathbb{Z}_p$, for a prime p, which allows them to analyze the probability of success of their *OneInvariantFactor* algorithm,

- in [127], Wan presents a generalization of [44] in a form of non-uniform distribution over a field; he applies the results to another variant of *OneInvariantFactor* algorithm,
- in [96, 133] random λ-sparse row and column matrices over a field are defined, these matrices are used as random preconditioners for solving a sparse system of equations,
- in [28] random s, t-sparse matrices are defined, and an analysis of the complexity of elimination method is performed.

Our presentation is based on [127], as we generalize the notion of non-uniformly distributed variables to the case of $\mathbb{Z}_q$ for composite $q \in \mathbb{N}$, see Def. 5.1.2. We propose to refer to the special case of non-uniform distribution induced on $\mathbb{Z}_q$ by the uniform distribution on $S \subset \mathbb{Z}$ as *almost-uniform distribution*, see Def. 5.1.4. Prop. 5.1.5 justifies the name. The idea of random sparse distributions in Def. 5.1.6, is inspired by [96, 133, 28, 7] and may give rise to object that are not commonly considered as sparse enough. In Sec. 5.1.3 definitions of random non-uniform, almost-uniform and sparse matrices are given.

Finally, let us refer to some other sources of random approach to matrix computation.

- Thm. 3 of [2] in Thm. 3 discuss uniform distribution of matrices with the same Hadamard's bound,
- in [11], symmetric random matrix, uniformly distributed over $\mathbb{Z}_q, q \in \mathbb{N}$ are discussed,
- [107] and [76] discuss uniformly distributed random Toeplitz matrices modulo a prime p or $q \in \mathbb{N}$, in [107] experimental statistics can be found;

However, structured matrices are not considered in this thesis.

5.1.2 Basic Notion

In this section we will consider random matrices over number rings.

Definition 5.1.1 (Number Ring) A number ring is equal to the rational field $\mathbb{Q}$ or the integer ring $\mathbb{Z}$, or the modular ring $\mathbb{Z}_q$, $q \in N, q \geq 2$.

For every probability distribution $\mathcal{P}_\xi$ we can define probability bounds α and β.

Definition 5.1.2 (Probability Bounds) Let $0 \leq \alpha \leq \beta \leq 1$. Let $\mathcal{P}_\xi$ be a discrete probability distribution on a set R. We say that (α, β) are probability bounds for $\mathcal{P}_\xi$ if α, β, and $\mathcal{P}_\xi$ fulfill the inequality:

$$\alpha \leq \mathcal{P}_\xi(x : x = x_0) \leq \beta \quad \forall x_0 \in R. \tag{5.1}$$

Proposition 5.1.3 (Existence of Probability Bounds) Probability bounds always exist.

PROOF Let $\mathcal{P}_\xi$ be a discrete probability distribution on a set R and $\xi : R \to [0,1]$ its probability mass function. We may define (α_ξ, β_ξ) by the equations:

$$\begin{aligned} \alpha_\xi &= \inf_{x_0 \in R} (\mathcal{P}_\xi(x : x = x_0)) = \inf_{x_0 \in R} (\xi(x_0)) \\ \beta_\xi &= \sup_{x_0 \in R} (\mathcal{P}_\xi(x : x = x_0)) = \sup_{x_0 \in R} (\xi(x_0)) \end{aligned} \tag{5.2}$$

Then (α_ξ, β_ξ) are probability bounds for $\mathcal{P}_\xi$.

Probability bounds may contain essential information about the probability distribution. This is the idea of Sec. 5.6.2.1 of the thesis of Z. Wan [127] in the case when R is a field. As a special case, Wan introduces almost-uniform distribution modulo prime q, which we present here in a more general case when q is composite, $q \in \mathbb{N}, q \geq 2$.

Almost Uniform Random Distribution Modulo q, cf. [127]

Definition 5.1.4 (Almost Uniform Distribution) Let $\mathcal{S}$ be a set of λ contiguous integers $\mathcal{S} = \{a, a+1, \ldots, a+\lambda-1\}$, where $a \in \mathbb{Z}$ and $\lambda > 0$. Let $\mathcal{P}_\mathcal{S}$ denote the discrete uniform distribution on $\mathcal{S}$. Let $q \in \mathbb{N}$, $q \geq 2$.

The almost uniform distribution (modulo q) is a distribution $\mathcal{P}_{\mathcal{S},q}$ on $\mathbb{Z}_q$ which is defined by the probability mass function $\xi_{\mathcal{S},q}$:

$$\xi_{\mathcal{S},q}(x_0) = \mathcal{P}_\mathcal{S}(x \in \mathbb{Z} : x = x_0 \mod q). \tag{5.3}$$

The almost uniform distribution $\mathcal{P}_{\mathcal{S},q}$ on $\mathbb{Z}_q$ describes the probability of choosing an element x from $\mathcal{S}$ such that its image modulo q is equal to a certain element $x_0 \in \mathbb{Z}_q$. This distribution is usually not uniform unless q divides λ. The name *almost uniform* is justified by the fact that the distribution originates from a uniform distribution. Moreover, Prop. 5.1.5 shows that the difference between probability bounds is small enough.

Proposition 5.1.5 (Lem. 5.9 of [127]) Let $\mathcal{S}$ be a set of λ contiguous integers $\mathcal{S} = \{a, a+1, \ldots, a+\lambda-1\}$, where $a \in \mathbb{Z}$ and $\lambda > 0$. Let $q \in \mathbb{N}$ and $q \geq 2$. Let $\mathcal{P}_{\mathcal{S},q}$ be the almost uniform distribution on $\mathbb{Z}_q$. We define

$$\alpha_q = \frac{1}{\lambda}\lfloor\frac{\lambda}{q}\rfloor, \quad \beta_q = \frac{1}{\lambda}\lceil\frac{\lambda}{q}\rceil. \tag{5.4}$$

Then (α_q, β_q) are probability bounds for $\mathcal{P}_{\mathcal{S},q}$.

Moreover, the probability distribution $\mathcal{P}_{\mathcal{S},q}$ on $\mathbb{Z}_q$ differs from the uniform distribution on $\mathbb{Z}_q$ by at most $\frac{1}{\lambda}$ i.e.

$$|\,\mathcal{P}_{\mathcal{S},q}(x = x_0) - \frac{1}{q}| \leq \frac{1}{\lambda}. \tag{5.5}$$

PROOF The proof of [127, Lem. 5.9] carry on to the case when q is not prime. Proving the second part of the lemma requires a simple estimation. Let us notice that

$$\mathcal{P}_{\mathcal{S},q}(x = x_0) - \frac{1}{q} \leq \frac{1}{\lambda}\lceil\frac{\lambda}{q}\rceil - \frac{1}{q} \leq \frac{1}{\lambda}(\frac{\lambda}{q} + 1) - \frac{1}{q} = \frac{1}{q} + \frac{1}{\lambda} - \frac{1}{q} = \frac{1}{\lambda}.$$

Analogously,

$$\mathcal{P}_{\mathcal{S},q}(x = x_0) - \frac{1}{q} \geq \frac{1}{\lambda}\lfloor\frac{\lambda}{q}\rfloor - \frac{1}{q} \geq \frac{1}{\lambda}(\frac{\lambda}{q} - 1) - \frac{1}{q} = \frac{1}{q} - \frac{1}{\lambda} - \frac{1}{q} = -\frac{1}{\lambda}.$$

Therefore

$$|\,\mathcal{P}_{\mathcal{S},q}(x = x_0) - \frac{1}{q}| \leq \frac{1}{\lambda}.$$

γ, ξ - sparse Distribution

Definition 5.1.6 (Sparse Distribution) Let R be a number ring. Let $0 < \gamma \ll 1$ and let $\mathcal{P}_\xi$ be a discrete probability distribution on R and $\xi : R \to [0, 1]$ its probability mass function.

γ, ξ - sparse probability distribution $\mathcal{P}_{\gamma,\xi}$ on R is given by its probability mass function $\eta_{\gamma,\xi}$:

$$\eta_{\gamma,\xi}(x) = \begin{cases} 1 - \gamma + \gamma\xi(0) & \text{if } x = 0 \\ \gamma\xi(x) & \text{if } x \neq 0. \end{cases} \tag{5.6}$$

In the γ, ξ - sparse probability distribution, the probability of choosing 0 is defined separately from the probability of choosing a non-zero element. This allows to construct highly non-uniform probability distributions, where the probability of choosing 0 is much larger than for non-zero elements.

Proposition 5.1.7 (Probability Bounds for Sparse Distribution) Let R be a number ring. Let $0 < \gamma \ll 1$ and let $\mathcal{P}_\xi$ be a discrete probability distribution on R and $\xi : R \to [0, 1]$ its probability mass function. Let $\mathcal{P}_{\gamma,\xi}$ be the γ, ξ - sparse probability distribution on R.

The probability bounds for $\mathcal{P}_{\gamma,\xi}$ are

$$(\alpha_{\gamma,\xi}, \beta_{\gamma,\xi}) = (\gamma\alpha_\xi, 1 - \gamma + \gamma\beta_\xi),$$

where (α_ξ, β_ξ) are probability bounds for $\mathcal{P}_\xi$.

PROOF Let (α_ξ, β_ξ) be the probability bound for $\mathcal{P}_\xi$. In particular this means that

$$\alpha_\xi \leq \xi(x) \leq \beta_\xi \quad \forall x \in R.$$

We have

$$1-\gamma+\gamma\alpha_\xi \le 1-\gamma+\gamma\xi(0) \le 1-\gamma+\gamma\beta_\xi$$
$$\gamma\alpha_\xi \le \gamma\xi(x) \le \gamma\beta_\xi \quad \forall x \in R, x \neq 0.$$

From definition of $\eta_{\xi,\gamma}$ in Eq. (5.6) we have

$$\min(\gamma\alpha_\xi, 1-\gamma+\gamma\alpha_\xi) \le \eta_{\xi,\gamma}(x) \le \max(\gamma\beta_\xi, 1-\gamma+\gamma\beta_\xi).$$

Since

$$\gamma\alpha_\xi \le 1-\gamma+\gamma\alpha_\xi$$
$$\gamma\beta_\xi \le 1-\gamma+\gamma\beta_\xi,$$

we may conclude that $(\gamma\alpha_\xi, 1-\gamma+\gamma\beta_\xi)$ are probability bounds for $\eta_{\xi,\gamma}$.

5.1.3 Random Matrices

A probability distribution $\mathcal{P}_\xi$ on R induces a probability distribution on the set $M(k,n,R)$ which consists of $k \times n$ matrices over R. We give the following classic definition.

Definition 5.1.8 (Random Matrix) Let R be a number ring. Let a discrete probability distribution $\mathcal{P}_\xi$ be given. Let $k, n \in \mathbb{N}$ and let $M(k,n,R)$ denote the set of all $k \times n$ matrices over R.

We say that $A \in M(k,n,R)$ is a $\mathcal{P}_\xi$-distributed random $k \times n$ matrix over R if the entries of A are randomly sampled from R according to probability $\mathcal{P}_\xi$, independently of each other.

By $M(k,n,R,\mathcal{P}_\xi)$ we will denote the set of all $\mathcal{P}_\xi$-distributed random $k \times n$ matrices over R. Probability distribution $\mathcal{P}_\xi$ induces a probability distribution on $M(k,n,R)$, denoted by $\overline{\mathcal{P}}_\xi$, such that

$$\overline{\mathcal{P}}_\xi(A : A = A') = \prod_{i=1}^{k}\prod_{j=1}^{n} \mathcal{P}_\xi(a_{ij} = a'_{ij}) \quad \forall A' \in M(k,n,R), \tag{5.7}$$

where a_{ij} (resp. a'_{ij}) denote the (i,j)th entry of matrix A (resp. A'). Let $A_{ij,b}$ denote an event that the (i,j)th entry of a matrix is equal to b. By Eq. (5.7), events $A_{ij,b}$ and $A_{kl,b'}$ are independent of each other for $(i,j) \neq (k,l)$. Therefore, a random matrix is an element of $M(k,n,R)$ given according to distribution $\mathcal{P}_\xi$. If it does not lead to ambiguity, we will use the same notation $\mathcal{P}_\xi$ for the distribution of elements of R and $M(k,n,R)$, as the distinction is clear from context.

Sometimes, it might be convenient to consider all probability distributions with the same probability bounds at the same time. This motivates the following definition.

Definition 5.1.9 ((α, β) Random Matrices) Let R be a number ring.

We say that $A \in M(k, n, R)$ is a (α, β)-random matrix over R, if there exists a probability distribution $\mathcal{P}_\xi$, such that

- A is a $\mathcal{P}_\xi$-distributed random matrix over R,
- (α, β) are probability bounds for $\mathcal{P}_\xi$.

By $M(k, n, R, \alpha, \beta)$ we will denote the set of all $k \times n$ (α, β)-random matrices over R.

Remark 5.1.10 An element of $M(k, 1, R)$ given according to distribution $\mathcal{P}_\xi$ is called a $\mathcal{P}_\xi$-distributed random vector. An element of $M(k, 1, \alpha, \beta)$ is called a (α, β)-random vector.

Let us now introduce two important classes of matrices, namely almost-uniformly distributed dense and sparse random matrices.

Almost Uniformly Distributed Dense Matrices, cf. [127]

Let $\mathcal{S}$ be a set of λ contiguous integers $\mathcal{S} = \{a, a+1, \ldots, a+\lambda-1\}$, where $a \in \mathbb{Z}$ and $\lambda > 0$. Let $\mathcal{P}_\mathcal{S}$ denote the discrete uniform distribution on $\mathcal{S}$. Let $q \in \mathbb{N}$ and $q \geq 2$.

Let $k, n \in \mathbb{N}$ and let $A \in M(k, n, R, \mathcal{P}_\mathcal{S})$ be an uniformly distributed random integer matrix. Let $A(q)$ be the image of A modulo q. Then $A(q) \in M(k, n, \mathbb{Z}_q, \mathcal{P}_{\mathcal{S},q})$ is an almost uniformly distributed matrix over $\mathbb{Z}_q$. Indeed, if an entry a_{ij} of A is chosen from $\mathcal{S}$ with uniform distribution, $a_{ij} \mod q$ is chosen with almost uniform distribution, see Def. 5.1.4.

Moreover, $A(q)$ is a random (α_q, β_q) matrix over $\mathbb{Z}_q$, where

$$\alpha_q = \frac{1}{\lambda}\lfloor\frac{\lambda}{q}\rfloor, \quad \beta_q = \frac{1}{\lambda}\lceil\frac{\lambda}{q}\rceil,$$

see Def. 5.1.9 for a definition of random (α_q, β_q) matrix and Prop. 5.1.5 for the probability bound for almost uniform distribution.

Sparse Matrices.

Definition 5.1.11 (Expected Number of Non-zero Elements) Let R be a number ring. Let $0 < \gamma \ll 1$ and let $\mathcal{P}_\xi$ be a discrete probability distribution on R. Let $\mathcal{P}_{\gamma,\xi}$ be the γ, ξ - sparse probability distribution on R.

Let $k, n \in \mathbb{N}, k, n > 0$. Consider the set $M(k, n, R, \mathcal{P}_{\gamma,\xi})$ of $\mathcal{P}_{\gamma,\xi}$-distributed random $k \times n$ matrices over R, together with the induced distribution $\overline{\mathcal{P}}_{\gamma,\xi}$, see Eq. (5.7). Let $A \in M(k, n, R, \mathcal{P}_{\gamma,\xi})$ and $\Omega_{\gamma,\xi}(A)$ define a random variable on $M(k, n, R, \mathcal{P}_{\gamma,\xi})$ equal to *the number of non-zero entries of matrix* A. Then

$$\Omega_{\gamma,\xi} = \mathbf{E}_{\overline{\mathcal{P}}_{\gamma,\xi}}(\Omega)$$

is the expected number of non-zero elements of a $\mathcal{P}_{\gamma,\xi}$-distributed random $k \times n$ matrix.

Proposition 5.1.12 (Value of $\Omega_{\gamma,\xi}$) Let R be a number ring. Let $0 < \gamma \ll 1$ and let $\mathcal{P}_\xi$ be a discrete probability distribution on R and $\xi : R \to [0,1]$ its probability mass function. Let $\mathcal{P}_{\gamma,\xi}$ be the γ, ξ - sparse probability distribution on R.

Let $k, n \in \mathbb{N}, k, n > 0$. Let $\Omega_{\gamma,\xi}$ be the expected number of non-zero elements of $\mathcal{P}_{\gamma,\xi}$-distributed random $k \times n$ matrices. Then $\Omega_{\gamma,\xi}$ is less than or equal $kn\gamma$.

PROOF The probability that an entry of a $\mathcal{P}_{\gamma,\xi}$-distributed random $k \times n$ matrix A is non-zero is

$$q = \sum_{x \neq 0} \gamma\xi(x) = \gamma(1 - \xi(0)) \leq \gamma$$

We can see the process of choosing the entries of a $k \times n$ matrix as a finite Bernoulli process consisting of kn independent Bernoulli trails [1], where choosing a non-zero entry is a 'success' event and happens with probability q. The number of non-zero elements is equal to the number of 'successes' in kn trails and therefore has a binomial distribution. Thus, the expected number of non-zero elements is equal to the expected number of successes in kn trails for a binomial distribution and is $knq \leq kn\gamma$.

Definition 5.1.13 (Random Sparse Matrix) Let R be a number ring.

Let $k, n \in \mathbb{N}$, and $k, n > 0$. Let $\gamma : \mathbb{N}^2 \ni (k,n) \to \gamma(k,n) \in [0,1]$ be a function of k and n. Let $\mathcal{P}_\xi$ be a discrete probability distribution on R and $\xi : R \to [0,1]$ its probability mass function. Let $\mathcal{P}_{\gamma,\xi}$ be the γ, ξ - sparse probability distribution on R.

Let $A \in M(k, n, R, \mathcal{P}_{\gamma,\xi})$ be a $\mathcal{P}_{\gamma,\xi}$-distributed random $k \times n$ matrix over R. Let $\Omega_{\gamma,\xi}$ be the expected number of non-zero elements of $\mathcal{P}_{\gamma,\xi}$-distributed random $k \times n$ matrices.

Then matrix A is a γ, ξ-sparse random matrix if $\Omega_{\gamma,\xi} \in o(kn)$ i.e.

$$\lim_{k,n \to \infty} \frac{\Omega_{\gamma,\xi}}{kn} = 0.$$

For other definitions of sparse matrices, see e.g. [96, 133, 28, 7].

Proposition 5.1.14 (Family of Sparse Matrices) Let R be a number ring.

Let $k, n \in \mathbb{N}, k, n > 0$. Let $\gamma : \mathbb{N}^2 \ni (k,n) \to \gamma(k,n) \in [0,1]$ be a function of k and n. Suppose that there exists $\epsilon > 0$ such that $\gamma(n,k) \in O(n^{-\epsilon})$ (resp. $O(k^{-\epsilon})$).

Let $\mathcal{P}_\xi$ be a discrete probability distribution on R and $\xi : R \to [0,1]$ its probability mass function. Let $\mathcal{P}_{\gamma,\xi}$ be the γ, ξ - sparse probability distribution on R.

Let $A \in M(k, n, R, \mathcal{P}_{\gamma,\xi})$ be a $\mathcal{P}_{\gamma,\xi}$-distributed random $k \times n$ matrix over R. Then A is a random sparse matrix.

[1] A Bernoulli trail is an experiment that have two possible outputs: a "success" with probability q and a "failure" with probability $1 - q$

PROOF Let us consider the case where $\gamma(n,k) = O(n^{-\epsilon})$. There exist $N, C > 0$ such that $\gamma(n,k) \leq Cn^{-\epsilon}$ for every $k, n > N$. By Prop. 5.1.12

$$\Omega_{\gamma,\xi} \leq kn\gamma(k,n)$$

Therefore

$$\lim_{k,n\to\infty} \frac{\Omega_{\gamma,\xi}}{kn} \leq \lim_{k,n\to\infty} \gamma(k,n) \leq \lim_{k,n\to\infty} Cn^{-\epsilon} = 0,$$

and the matrix is sparse, according to definition 5.1.13.

5.2 Rank of Random Matrices

Most of the known results for the rank of random matrices consider the case of uniform distribution over a finite field $\mathbb{Z}_p$, where p is a prime. This is the case of [10], where the rank modulo p^s is considered[2]. For $s = 1$ this is a classic result, see e.g. [10, Thm. 1.1] and the references therein. Also, [7] gives the estimation on the expected rank mod p for sparse matrices, in which non-zero entries are uniformly distributed.

In our work, we will mostly consider integer matrices modulo a prime p, which results in an *almost uniform distribution* of entries mod p, see Sec. 5.1.2, 5.1.3. This is an example of a non-uniform distribution. This kind of distribution has already been considered in the context of full rank, see [127, Sec. 5.6.2.1] and [95, Sec. 3].

The outline of the section is as follows. In Sec. 5.2.1 we show how to carry on probability estimations in the case on non-uniform distribution. In Lem. 5.2.2 and 5.2.3 we give the probability that the rank of a $k \times n$ matrix modulo p (or also over $\mathbb{Q}, \mathbb{Z}$) is r or is less than r respectively, in the case when the matrix is given according to non-uniform distribution, in the sense of Def. 5.1.8. We finish the section by comparison with the existing results in Sec. 5.2.2.

5.2.1 Main Results

For a uniform distribution modulo p, where p is a prime, the problem has widely be considered, see e.g. [10] and the references therein. Let $\Pi_0 = 1, \Pi_i(q) = \prod_{l=1}^{k}(1-q^l)$[3], for $0 \leq q < 1$. We have the following result.

Theorem 5.2.1 (Thm. 1.1 of [10]) *Let p be a prime. Let us set $q = \frac{1}{p}$ and $\Pi_k(q) = \prod_{i=1}^{k}(1-q^i)$, for $k \in \mathbb{N}$, $\Pi_0 = 1$. Let $k, n \in \mathbb{N}$ and $k, n > 0$. Let A be a $k \times n$ matrix over $\mathbb{Z}_p$ with entries chosen randomly and uniformly from $\mathbb{Z}_p$, independently of each other. Then the probability $\mathcal{P}$ that* $\operatorname{rank}(A)$ *is r, $0 < r \leq \min(k,n)$ is equal to*

$$\mathcal{P}\left(A : \operatorname{rank}_p(A) = r\right) = q^{(n-r)(k-r)} \frac{\Pi_n(q)\Pi_k(q)}{\Pi_{n-r}(q)\Pi_r(q)\Pi_{k-r}(q)}. \tag{5.8}$$

[2]Precisely, McKay rank mod p^s is the size of the greatest non-zero minor.

[3]This is the q-Binomial Coefficient, or Gaussian Coefficient polynomial.

PROOF See [10] and the references therein.

In the case of non-uniform distribution, we are able to obtain an analogous result.

Lemma 5.2.2 (Probability of rank(A) = r) *Let R be a number field and let $\alpha, \beta \in \mathbb{R}, 0 \leq \alpha \leq \beta < 1$ be given. Let $k, n \in \mathbb{N}$, $k, n > 0$ and let A be a $k \times n$ (α, β)-random matrix, see Def. 5.1.9.*

Let $k' = \min(k, n)$ and $n' = \max(k, n)$. The probability $\mathcal{P}$ that $\operatorname{rank}(A)$ is r, $0 < r \leq k'$ is less than or equal to

$$\mathcal{P}\left(A : \operatorname{rank}_p(A) = r\right) \leq \beta^{(n'-r)(k'-r)} \frac{\Pi_{n'}(\alpha)}{\Pi_{n'-r}(\alpha)} \frac{\Pi_{k'}(\beta)}{\Pi_r(\beta)\Pi_{k'-r}(\beta)}. \tag{5.9}$$

PROOF Without loss of generality we may assume that $k \leq n$ i.e. $k' = k$ and $n' = n$. In the opposite case, the reasoning can be performed for matrix A^T. For $i = 1 \ldots k$ let A_i denote the submatrix of A consisting of i first rows of A.

The proof is inductive on $k - r$ and r, $n \geq k$ is set.

1. Base cases.

   (a) $r = 0$.
   The fact that $\operatorname{rank}(A) = 0$ means that all the entries of A are zero. Thus

   $$\mathcal{P}(\operatorname{rank}(A) = 0) = \mathcal{P}_\xi(a_{ij} = 0, \forall i = 1..k; j = 1..n) \leq \beta^{nk}.$$

   Since $\frac{\Pi_n(\alpha)}{\Pi_n(\alpha)} = 1$ and $\frac{\Pi_k(\beta)}{\Pi_0(\beta)\Pi_k(\beta)} = 1$ we have the required inequality.

   (b) $k - r = 0$.
   For every $i = 2 \ldots k$, the fact that $\operatorname{rank}(A_i) = i$ implies that the rank of $\operatorname{rank}(A_{i-1}) = i - 1$. Therefore by the definition of conditional probability

   $$\begin{aligned}\mathcal{P}(\operatorname{rank}(A_i) = i \mid \operatorname{rank}(A_{i-1}) = i - 1) &= \frac{\mathcal{P}(\operatorname{rank}(A_i) = i \wedge \operatorname{rank}(A_{i-1}) = i - 1)}{\mathcal{P}(\operatorname{rank}(A_{i-1}) = i - 1)} \\ &= \frac{\mathcal{P}(\operatorname{rank}(A_i) = i)}{\mathcal{P}(\operatorname{rank}(A_{i-1}) = i - 1)},\end{aligned}$$

   which for $i = k$ gives

   $$\begin{aligned}\mathcal{P}(\operatorname{rank}(A_k) = k) &= \mathcal{P}(\operatorname{rank}(A_k) = k \mid \operatorname{rank}(A_{k-1}) = k - 1) \cdot \mathcal{P}(\operatorname{rank}(A_{k-1}) = k - 1), \\ &= \mathcal{P}(\operatorname{rank}(A_1) = 1) \prod_{i=2}^{k} \mathcal{P}(\operatorname{rank}(A_i) = i \mid \operatorname{rank}(A_{i-1}) = i - 1)\end{aligned}$$

   To compute $\mathcal{P}(\operatorname{rank}(A_i) = i \mid \operatorname{rank}(A_{i-1}) = i - 1)$ we notice that the fact that $\operatorname{rank}(A_{i-1}) = i - 1$ means that there exist an $(i - 1) \times (i - 1)$ minor S of A_{i-1},

such that $\det(S) \neq 0$. Let us now consider the choice of entries in the ith row. Notice, that S consists of $(i-1)$ columns of A_{i-1} and let T be a $(i-1)\times(n-i+1)$ submatrix consisting of the remaining columns. Let v be a $1\times(i-1)$ row vector of entries in the ith row corresponding to S and let w be a $1\times(n-i+1)$ vector corresponding to T.

To estimate the probability that $\operatorname{rank}(A_i) = i$, we can notice, that the entries of v can be chosen randomly and w needs to fulfill $TS^{-1}v \neq w$. This gives the probability

$$\mathcal{P}(\operatorname{rank}(A_i) = i \mid \operatorname{rank}(A_{i-1}) = i-1) = \mathcal{P}_\xi(v, w : TS^{-1}v \neq w) \leq (1-\alpha^{n-i+1}),$$

since w is a $1\times(n-i+1)$ vector. Also, $\mathcal{P}(\operatorname{rank}(A_1) = 1)$ means that $A_1 \neq 0$, which occurs with probability at most $(1-\alpha^n)$.

In consequence

$$\mathcal{P}(\operatorname{rank}(A_k) = k) \leq \prod_{i=0}^{k-1}(1-\alpha^{n-i}) = \frac{\Pi_n(\alpha)}{\Pi_{n-k}(\alpha)}.$$

As $\frac{\Pi_k(\beta)}{\Pi_{k-k}(\beta)\Pi_k(\beta)} = 1$ for $r = k$, we get the result.

2. Inductive step.

Let us consider the case where $k - r = M > 0$. In the inductive hypothesis let us assume that for all $(\tilde{r}, \tilde{k})$ such that $\tilde{k} - \tilde{r} < M$ or $\tilde{k} - \tilde{r} = M$ and $\tilde{r} < r$ we have

$$\mathcal{P}\left(A : \operatorname{rank}(A) = \tilde{r}\right) \leq \beta^{(\tilde{n}-\tilde{r})(\tilde{k}-\tilde{r})} \frac{\Pi_{\tilde{n}}(\alpha)}{\Pi_{\tilde{n}-\tilde{r}}(\alpha)} \frac{\Pi_{\tilde{k}}(\beta)}{\Pi_{\tilde{r}}(\beta)\Pi_{\tilde{k}-\tilde{r}}(\beta)}.$$

We will show that

$$\mathcal{P}\left(A : \operatorname{rank}(A) = r\right) \leq \beta^{(n-r)(k-r)} \frac{\Pi_n(\alpha)}{\Pi_{n-r}(\alpha)} \frac{\Pi_k(\beta)}{\Pi_r(\beta)\Pi_{k-r}(\beta)}.$$

Let us consider the case $\mathcal{P}(\operatorname{rank}(A_k) = r)$. By the law of alternatives we can write:

$$\begin{aligned}\mathcal{P}(\operatorname{rank}(A_k) = r) &= \mathcal{P}(\operatorname{rank}(A_k) = r \mid \operatorname{rank}(A_{k-1}) = r) \cdot \mathcal{P}(\operatorname{rank}(A_{k-1}) = r) \\ &\quad + \mathcal{P}(\operatorname{rank}(A_k) = r \mid \operatorname{rank}(A_{k-1}) = r-1) \cdot \mathcal{P}(\operatorname{rank}(A_{k-1}) = r-1),\end{aligned}$$

as $\mathcal{P}(\operatorname{rank}(A_k) = r \mid \operatorname{rank}(A_{k-1}) = r') = 0$ for $r' < r-1$.

(a) *Case* $\mathcal{P}(\operatorname{rank}(A_k) = r \mid \operatorname{rank}(A_{k-1}) = r-1)$

To estimate $\mathcal{P}(\operatorname{rank}(A_k) = r \mid \operatorname{rank}(A_{k-1}) = r-1)$, we repeat the reasoning from 1b. Let S be a $(r-1)\times(r-1)$ non-zero minor of A_{k-1} and let T be a $(r-1)\times(n-r+1)$ submatrix which contains the same rows as S and all remaining

columns. Let us consider the choice of entries in the kth row. Let v be a $1 \times (r-1)$ row vector corresponding to the columns of S and let w be the $1 \times (n-r+1)$ vector of remaining entries. Again, the entries of v can be chosen randomly and w needs to fulfill $TS^{-1}v \neq w$. This gives the probability

$$\mathcal{P}(\operatorname{rank}(A_k) = r \mid \operatorname{rank}(A_{k-1}) = r-1) = \mathcal{P}_\xi(v, w : TS^{-1}v \neq w) \leq (1 - \alpha^{n-r+1}) \tag{5.10}$$

(b) *Case* $\mathcal{P}(\operatorname{rank}(A_k) = r \mid \operatorname{rank}(A_{k-1}) = r)$
Let us now estimate $\mathcal{P}(\operatorname{rank}(A_k) = r \mid \operatorname{rank}(A_{k-1}) = r)$. This time, let S be a $r \times r$ non-zero minor of A_{k-1} and let T be a $r \times n-r$ submatrix which has the same rows as S and the remaining columns. Let us consider the choice of entries in the kth row. Let v be a $1 \times r$ row vector corresponding to the columns of S and let w be the $1 \times (n-r)$ vector of remaining entries. Again, the entries of v can be chosen randomly and w needs to fulfill $TS^{-1}v = w$. This gives the probability.

$$\mathcal{P}(\operatorname{rank}(A_k) = r \mid \operatorname{rank}(A_{k-1}) = r) = \mathcal{P}_\xi(v, w : TS^{-1} = w) \leq \beta^{n-r} \tag{5.11}$$

By the inductive hypothesis

$$\begin{aligned}
\mathcal{P}(\operatorname{rank}(A_{k-1}) = r-1) &\leq \beta^{(n-r+1)(k-r)} \frac{\Pi_n(\alpha)}{\Pi_{n-r+1}(\alpha)} \frac{\Pi_{k-1}(\beta)}{\Pi_{r-1}(\beta)\Pi_{k-r}(\beta)}, \\
\mathcal{P}(\operatorname{rank}(A_{k-1}) = r) &\leq \beta^{(n-r)(k-1-r)} \frac{\Pi_n(\alpha)}{\Pi_{n-r}(\alpha)} \frac{\Pi_{k-1}(\beta)}{\Pi_r(\beta)\Pi_{k-1-r}(\beta)}.
\end{aligned}$$

Therefore

$$\begin{aligned}
\mathcal{P}(\operatorname{rank}(A_k) = r) &\leq (1 - \alpha^{n-r+1}) \cdot \beta^{(n-r+1)(k-r)} \frac{\Pi_n(\alpha)}{\Pi_{n-r+1}(\alpha)} \frac{\Pi_{k-1}(\beta)}{\Pi_{r-1}(\beta)\Pi_{k-r}(\beta)} \\
&\quad + \beta^{n-r} \cdot \beta^{(n-r)(k-1-r)} \frac{\Pi_n(\alpha)}{\Pi_{n-r}(\alpha)} \frac{\Pi_{k-1}(\beta)}{\Pi_r(\beta)\Pi_{k-1-r}(\beta)} \\
&= \beta^{(n-r)(k-r)} \frac{\Pi_n(\alpha)}{\Pi_{n-r}(\alpha)} \frac{\Pi_{k-1}(\beta)}{\Pi_{r-1}(\beta)\Pi_{k-1-r}(\beta)} \left(\frac{\beta^{k-r}}{1-\beta^{k-r}} + \frac{1}{1-\beta^r} \right) \\
&= \beta^{(n-r)(k-r)} \frac{\Pi_n(\alpha)}{\Pi_{n-r}(\alpha)} \frac{\Pi_{k-1}(\beta)}{\Pi_{r-1}(\beta)\Pi_{k-1-r}(\beta)} \frac{\beta^{k-r} - \beta^k + 1 - \beta^{k-r}}{(1-\beta^{k-r})(1-\beta^r)} \\
&= \beta^{(n-r)(k-r)} \frac{\Pi_n(\alpha)}{\Pi_{n-r}(\alpha)} \frac{\Pi_k(\beta)}{\Pi_r(\beta)\Pi_{k-r}(\beta)},
\end{aligned}$$

which finishes the proof.

Let us consider the situation of Lem. 5.2.2. In order to compute the probability that the rank is smaller than or equal to a given r, we can sum the probability $\mathcal{P}(\operatorname{rank}(A) = j)$ for $j = 1..r$. By Lem. 5.2.2, we obtain

$$\mathcal{P}(\operatorname{rank}(A) \leq r) \leq \sum_{j=0}^{r} \beta^{(n'-j)(k'-j)} \frac{\Pi_{n'}(\alpha)}{\Pi_{n'-j}(\alpha)} \frac{\Pi_{k'}(\beta)}{\Pi_j(\beta)\Pi_{k'-j}(\beta)}. \tag{5.12}$$

A closed algebraic formula remain to be found for Eq. (5.12). In two particular cases we obtain

- for $\alpha = \beta = q$: $\mathcal{P}(\mathrm{rank}(A) \leq r) \leq \sum_{j=0}^{r} q^{(n'-j)(k'-j)} \frac{\Pi_{n'}(q)\Pi_{k'}(q)}{\Pi_{n'-j}(q)\Pi_j(q)\Pi_{k'-j}(q)}$;
- for $\alpha = 0$: $\mathcal{P}(\mathrm{rank}(A) \leq r) \leq \sum_{j=0}^{r} \beta^{(n'-j)(k'-j)} \frac{\Pi_{k'}(\beta)}{\Pi_j(\beta)\Pi_{k'-j}(\beta)}$.

To bound the probability, without using Eq. (5.12), we propose the following reasoning.

Lemma 5.2.3 (Probability of rank(A) $\leq r$) *Let R be a number field and let $\alpha, \beta \in \mathbb{R}, 0 \leq \alpha \leq \beta < 1$ be given. Let $k, n \in \mathbb{N}$, $k, n > 0$ and let A be a $k \times n$ (α, β)-random matrix, see Def. 5.1.9.*

Let $k' = \min(k, n)$ and $n' = \max(k, n)$. The probability $\mathcal{P}$ that $\mathrm{rank}(A)$ *is smaller than or equal r, $0 < r < k'$ is less than*

$$\left(\beta^{(n'-r)(k'-r)} \frac{\Pi_{k'-1}(\beta)}{\Pi_r(\beta)\Pi_{k'-r}(\beta)}\right)(1 - \beta^{(r+1)(k'-r)}) \tag{5.13}$$

PROOF Without loss of generality we may assume that $k \leq n$ i.e. $k' = k$ and $n' = n$. In the opposite case, the reasoning can be performed for matrix A^T. As in the previous proof, let A_i denote the submatrix consisting of i first rows of A, for $i = 1 \ldots k$.

For $i = 1 \ldots r$ we will denote by $A \in \mathcal{A}_{k,r}(i)$ the event that $\mathrm{rank}(A_{k-i}) \leq r - i$.By adding a row we may increase the rank by 1 in the worst case, which implies the inclusions

$$\mathcal{A}_{k,r}(i) \subset \mathcal{A}_{k,r}(i-1),$$

for $i = 2 \ldots r$. Moreover $\mathcal{P}(\mathcal{A}_{k,r}(0))$ is the probability that $\mathrm{rank}(A) \leq r$ that we are going to estimate.

Notice, that $\mathcal{A}_{k,r}(0)$ can be represented as a sum of increasing sets

$$\mathcal{A}_{k,r}(r) \subset \mathcal{A}_{k,r}(r-1) \subset \cdots \subset \mathcal{A}_{k,r}(1) \subset \mathcal{A}_{k,r}(0).$$

Therefore $\mathcal{A}_{k,r}(0)$ is a sum of disjoint sets

$$\mathcal{A}_{k,r}(0) = \mathcal{A}_{k,r}(r) \cup \bigcup_{i=0}^{r-1} (\mathcal{A}_{k,r}(i) \setminus \mathcal{A}_{k,r}(i+1)).$$

Thus, we may compute $\mathcal{P}(\mathcal{A}_{k,r}(0))$:

$$\mathcal{P}(\mathcal{A}_{k,r}(0)) = \mathcal{P}(\mathcal{A}_{k,r}(r)) + \sum_{i=0}^{r-1} \mathcal{P}(\mathcal{A}_{k,r}(i) \wedge \neg \mathcal{A}_{k,r}(i+1)). \tag{5.14}$$

1. $\mathcal{P}(\mathcal{A}_{k,r}(r))$ *Computation*

   The fact that $A \in \mathcal{A}_{k,r}(r)$ means that the submatrix $A[1..k-r][1..n]$ consisting of $k-r$ first rows is 0. Therefore

$$\mathcal{P}(\mathcal{A}_{k,r}(r)) \leq \mathcal{P}_\xi(a_{ij} = 0, i = 1..k-r; j = 1..n) \leq \beta^{n(k-r)}. \tag{5.15}$$

2. $\mathcal{P}(\mathcal{A}_{k,r}(i) \wedge \neg \mathcal{A}_{k,r}(i+1))$ *Computation*

   Now let us compute $\mathcal{P}(\mathcal{A}_{k,r}(i) \wedge \neg \mathcal{A}_{k,r}(i+1))$. In terms of the rank, it is equivalent to say that $\operatorname{rank}(A_{k-i}) \leq r-i$ and $\operatorname{rank}(A_{k-i-1}) > r-i-1$ at the same time. This implies that

$$r-i \leq \operatorname{rank}(A_{k-i-1}) \leq \operatorname{rank}(A_{k-i}) \leq r-i$$

   and therefore both the rank of A_{k-i} and A_{k-i-1} are equal to $r-i$.

   By the definition of the conditional probability we have that $\mathcal{P}(B \cap C) = \mathcal{P}(B)\mathcal{P}(B \mid C)$, for any events B and C. Thus

$$\begin{aligned}\mathcal{P}(\mathcal{A}_{k,r}(i) \wedge \neg \mathcal{A}_{k,r}(i+1)) &= \mathcal{P}(\operatorname{rank}(A_{k-i}) = \operatorname{rank}(A_{k-i-1}) = r-i) \\ &= \mathcal{P}(\operatorname{rank}(A_{k-i}) = r-i \mid \operatorname{rank}(A_{k-i-1}) = r-i)\mathcal{P}(\operatorname{rank}(A_{k-i-1}) = r-i)\end{aligned}$$

   By Eq. (5.11) for $k-i$ and $r-i$,

$$\mathcal{P}(\operatorname{rank}(A_{k-i}) = r-i \mid \operatorname{rank}(A_{k-i-1}) = r-i) \leq \beta^{n-r+i}.$$

   Then by Lem. 5.2.2 for $\alpha = 0$

$$\begin{aligned}\mathcal{P}(\operatorname{rank}(A_{k-i-1}) = r-i) &\leq \beta^{(n-r+i)(k-r-1)} \frac{\Pi_{k-i-1}(\beta)}{\Pi_{r-i}(\beta)\Pi_{k-r-1}(\beta)} \\ &\leq \beta^{(n-r+i)(k-r-1)} \frac{\Pi_{k-1}(\beta)}{\Pi_r(\beta)\Pi_{k-r-1}(\beta)}.\end{aligned}$$

   Therefore

$$\begin{aligned}\mathcal{P}(\mathcal{A}_{k,r}(i) \wedge \neg \mathcal{A}_{k,r}(i+1)) &\leq \beta^{n-r+i}\beta^{(n-r+i)(k-r-1)} \frac{\Pi_{k-1}(\beta)}{\Pi_r(\beta)\Pi_{k-r-1}(\beta)} \\ &= \beta^{(n-r+i)(k-r)} \frac{\Pi_{k-1}(\beta)}{\Pi_r(\beta)\Pi_{k-r-1}(\beta)}.\end{aligned} \tag{5.16}$$

3. *Computation of* $\mathcal{P}(\mathcal{A}_{k,r}(0))$

From Eqs. (5.14), (5.15) and (5.16) we may compute

$$\mathcal{P}(\mathcal{A}_{k,r}(0)) \leq \beta^{n(k-r)} + \sum_{i=0}^{r-1} \left(\beta^{(n-r+i)(k-r)} \frac{\Pi_{k-1}(\beta)}{\Pi_r(\beta)\Pi_{k-r-1}(\beta)} \right)$$
$$= \sum_{i=0}^{r} \left(\beta^{(n-r+i)(k-r)} \frac{\Pi_{k-1}(\beta)}{\Pi_r(\beta)\Pi_{k-r-1}(\beta)} \right) = \left(\beta^{(n-r)(k-r)} \frac{\Pi_{k-1}(\beta)}{\Pi_r(\beta)\Pi_{k-r-1}(\beta)} \right) \sum_{i=0}^{r} \beta^{i(k-r)}$$
$$\leq \left(\beta^{(n-r)(k-r)} \frac{\Pi_{k-1}(\beta)}{\Pi_r(\beta)\Pi_{k-r-1}(\beta)} \right) \frac{1-\beta^{(r+1)(k-r)}}{1-\beta^{k-r}}$$

which finishes the proof.

5.2.2 Comparison with Existing Results

In the situation of Lem. 5.2.2,5.2.3, let us consider $n = k$. In this case, we may apply Theorem 5.14 of [127] and obtain two results

$$\mathcal{P}(\text{rank}(A) = n) = \mathcal{P}(\det(A) \neq 0) \geq \prod_{i=1}^{\infty}(1-\beta^i).$$
$$\mathcal{P}(\text{rank}(A) < n) \leq 1 - \prod_{i=1}^{\infty}(1-\beta^i) \leq \sum_{i=1}^{\infty} \beta^i \leq \frac{\beta}{1-\beta}.$$

By Lem. 5.2.3 for $r = k-1, n \geq k$ we get

$$\beta \frac{1-\beta^{(r+1)(n-r)}}{1-\beta^{n-r}} \leq \frac{\beta}{1-\beta^{n-r}}.$$

Lemma 3.1 of [95] gives the same bounds as Lem 5.2.2 ours for a less general family of distributions. We have

$$\mathcal{P}(\text{rank}(A) = n) \leq \prod_{i=1}^{n}(1-\alpha^i).$$

Finally, let us consider [10, Thm 1.1] and let us assume that $\alpha = \beta = q$. In this case results of [10, Thm 1.1] and Lem. 5.2.2 coincide.

5.3 The Expected Number of Non-trivial Invariant Factors

The number of non-trivial invariant factors is one of the parameter that determine the performance of the Smith form and determinant algorithms of [44]. In this section we will give a new evaluation of the expected number of non-trivial invariant factors for random matrices, which leads to an asymptotically better bound in the case of random dense

matrices, see Thm. 5.3.10, Cor. 5.3.11. The ideas presented in this section are based on and inspired by the article of [44].

The outline of the section is as follows. In Sec. 5.3.1 we compute the expected number of invariant factors divisible by p for a random non-uniformly distributed (α, β) matrices, see Def. 5.1.9. Lem. 5.3.2 gives the result. Then we apply this result to the case of almost uniformly distributed random matrices, see Cor. 5.3.3. In Sec. 5.3 we give the main result for integer matrices. In Thm. 5.3.10 we compute the expected number of non-trivial invariant factors of an integer, almost uniformly distributed matrix. In Cor. 5.3.11 we prove that the probability that the number of non-trivial invariant factors in considerably higher, is small. In Sec. 5.3.3 we analyze the case of sparse matrices.

We start by a definition of a non-trivial invariant factor.

Definition 5.3.1 (Non-trivial factor) Let R be a number field. Let $k, n \in \mathbb{N}, k, n > 0$ and let A be a $k \times n$ matrix over R. Let $SF(A) = \operatorname{diag}(s_1(A), \ldots, s_{\min(k,n)}(A))$ be the Smith normal form of A over R. If $s_i(A) \neq 1$ then $s_i(A)$ is a non-trivial invariant factor of A.

5.3.1 Case Modulo p

In the first lemma we will consider the number of invariant factors divisible by p, where p is a prime.

Lemma 5.3.2 (The Expected Number of Non-trivial Invariant Factors at p) *Let p be a prime. Let a discrete probability distribution $\mathcal{P}$ on $\mathbb{Z}$ be given and let $0 < \beta < 1$ be such that*

$$\mathcal{P}(x : x = x_0 \mod p) \leq \beta \quad \forall x_0 \in \mathbb{Z}_p\,.$$

Let $n \in \mathbb{N}$ and $n > 0$. Let A be a $n \times n$ matrix over $\mathbb{Z}$ with entries chosen randomly with the probability distribution $\mathcal{P}$. Let S be the Smith normal form of A over $\mathbb{Z}$.

The expected number of non-trivial invariant factors of S divisible by p, denoted E_p, is less than or equal

$$E_p \leq \frac{\ln(1-\beta)}{\ln(\beta)} + \frac{4}{3}. \tag{5.17}$$

PROOF Let $I_j(p)$ denote the event, that at least j invariant factors of A are divisible by p. This is equivalent to say that the rank of $A \mod p$ is at most $n - j$. This proves that $\mathcal{P}\left(I_j\left(p\right)\right) = \mathcal{P}\left(\operatorname{rank}(A \mod p) \leq n - j\right)$.

By Eq. (5.3.2), $A \mod p$ is a $(0, \beta)$-random matrix over $\mathbb{Z}_p$. By Lem. 5.2.3 the latter is less than or equal

$$\mathcal{P}\left(\operatorname{rank}(A \mod p) \leq n - j\right) \leq \min(1, \beta^{j^2} \prod_{l=1}^{j} \frac{1}{1-\beta^l}).$$

The expected number of invariant factors divisible by p verifies:

$$E_p = \sum_{j=0}^{n} j\left(\mathcal{P}\left(I_j\left(p\right)\right) - \mathcal{P}\left(I_{j+1}\left(p\right)\right)\right) = \sum_{j=1}^{n} \mathcal{P}\left(I_j\left(p\right)\right) = \tag{5.18}$$

$$\sum_{j=1}^{n} \mathcal{P}(\operatorname{rank}(A \mod p) \leq n-j) \leq \sum_{j=1}^{n} \min(1, \beta^{j^2} \prod_{l=1}^{j} \frac{1}{1-\beta^l}).$$

Let us determine a threshold J such that for $j \geq J$ $\beta^{j^2} \prod_{l=1}^{j} \frac{1}{1-\beta^l} \leq 1$.

A rough estimation gives $\beta^{j^2} \prod_{l=1}^{j} \frac{1}{1-\beta^l} \leq \beta^{j^2} \left(\frac{1}{1-\beta}\right)^j$. Let us solve the following inequality for $j \in \mathbb{N}$:

$$\beta^{j^2} \left(\frac{1}{1-\beta}\right)^j \leq 1 \tag{5.19}$$

Notice that $0 < \beta < 1$. We obtain that $j \geq \frac{\ln(1-\beta)}{\ln(\beta)}$. Hence we can take

$$J = \max(1, \lceil \frac{\ln(1-\beta)}{\ln(\beta)} \rceil). \tag{5.20}$$

Now the expected number of invariant factors divisible by p is bounded by

$$E_p \leq \sum_{j=1}^{J-1} 1 + \sum_{j=J}^{n} \beta^{j^2} \left(\frac{1}{1-\beta}\right)^j \leq J - 1 + \sum_{j=J}^{n} \beta^{j^2} \left(\frac{1}{1-\beta}\right)^j. \tag{5.21}$$

Let us now estimate the sum

$$\sum_{j=J}^{n} \beta^{j^2} \left(\frac{1}{1-\beta}\right)^j. \tag{5.22}$$

Let us put $a_j = \beta^{j^2} \left(\frac{1}{1-\beta}\right)^j$ and $q_j = \frac{a_{j+1}}{a_j} = \frac{\beta^{2j+1}}{1-\beta}$. Then for $j \geq J$

$$a_j \leq 1 \tag{5.23}$$

by Eq. (5.19). Also, Eq. (5.19) implies that $\beta^{j^2} \leq (1-\beta)^j$ for $j \geq J$, which gives

$$(q_j)^j = \beta^{j^2+j} \frac{\beta^{j^2}}{(1-\beta)^j} \leq^{(5.19)} \beta^{j^2+j} \leq^{(5.19)} \beta^j (1-\beta)^j \leq \left(\frac{1}{4}\right)^j.$$

Therefore we may conclude that

$$q_j \leq \frac{1}{4} \quad \forall j \geq J. \tag{5.24}$$

The sum (5.22) is equal to $a_J + \sum_{j=J+1}^{n} a_j q_{j-1} = a_J + a_J q_J + a_J q_J q_{J+1} + \dots$ This allows us to evaluate the sum (5.22)

$$\sum_{j=J}^{n} \beta^{j^2} \left(\frac{1}{1-\beta}\right)^j = a_J + \sum_{j=J+1}^{n} a_j q_{j-1} \leq\leq^{(5.23)(5.24)} \sum_{j=J}^{n} \left(\frac{1}{4}\right)^{j-J} \frac{1}{1-\frac{1}{4}} \leq \frac{4}{3}. \tag{5.25}$$

Finally let us evaluate E_p. By Eq. (5.21), Eq. (5.20) and Eq. (5.25) the expected number of invariant factors divisible by p is less than or equal to

$$\lceil \frac{\ln(1-\beta)}{\ln(\beta)} \rceil - 1 + \frac{4}{3} \leq \frac{\ln(1-\beta)}{\ln(\beta)} + \frac{4}{3},$$

which finishes the proof.

Let us now consider the case of almost uniform distribution, where the entries of the matrix are uniformly sampled from a set of contiguous integers. For random dense and sparse matrices we have the following corollaries.

Corollary 5.3.3 Let $\mathcal{S}$ be a set of $\lambda > 1$ contiguous integers, $\mathcal{S} = \{a, a+1, \dots, a+\lambda-1\}$, for a certain $a \in \mathbb{Z}$. Let p be a prime, $s \in \mathbb{N}, s > 0$. Let $n \in \mathbb{N}$ and let A be a $n \times n$ integer matrix with entries chosen randomly and uniformly from set $\mathcal{S}$. The expected number of non-trivial invariant factors of A divisible by p is at most

$$\frac{\log_2(3)}{\log_2(3)-1} + \frac{4}{3} \quad \text{for } p = 2 \tag{5.26}$$

$$\frac{1+\log(p)}{\log(p)-1} + \frac{4}{3} \quad \text{for } 2 < p < \lambda \tag{5.27}$$

$$1 - log_\lambda(\lambda-1) + \frac{4}{3} \quad \text{for } p \geq \lambda. \tag{5.28}$$

PROOF The entries of $A \mod p$ are distributed according to almost uniform distribution modulo p i.e. for each $x_0 \in \mathbb{Z}_p$

$$\mathcal{P}(x : x \mod p = x_0) \leq \beta = \frac{1}{\lambda} \lceil \frac{\lambda}{p} \rceil.$$

From Lem. 5.3.2 we know that the expected number of invariant factors divisible by p is less that or equal

$$\frac{\ln(1-\beta)}{\ln(\beta)} + \frac{4}{3}. \tag{5.29}$$

Let us now estimate $\frac{\ln(1-\beta)}{\ln(\beta)}$ in the case when $\beta = \frac{1}{\lambda}\lceil\frac{\lambda}{p}\rceil$. First, let us notice that for $p < \lambda$ $\beta \leq \frac{2}{p+1}$. Indeed, if $p < \lambda$ then $\lambda \geq p+1$ and $\frac{1}{\lambda} \leq \frac{1}{p+1}$. Thus

$$\beta = \frac{1}{\lambda}\lceil\frac{\lambda}{p}\rceil \leq \frac{1}{\lambda}\frac{\lambda+p-1}{p} \leq \frac{1}{p} + \frac{p-1}{\lambda p} \leq \frac{p+1+p-1}{(p+1)p} \leq \frac{2}{p+1}.$$

Therefore for $2 < p < \lambda$:

$$\frac{1}{1-\beta} \leq \frac{p+1}{p-1} \leq \frac{2p}{1} = 2p$$
$$\frac{1}{\beta} \geq \frac{p+1}{2} \geq \frac{p}{2}.$$

We may compute

$$\frac{\ln(1-\beta)}{\ln(\beta)} \leq \frac{\log_p(2p)}{\log_p(\frac{p}{2})} = \frac{\log_p(2)+1}{1-\log_p(2)}.$$

As $\log_p(2) = \frac{1}{\log(p)}$, where $\log = \log_2$ is the binary logarithm, the latter transforms to

$$\frac{\log_p(2)+1}{1-\log_p(2)} = \frac{\frac{1}{\log(p)}+1}{1-\frac{1}{\log(p)}} = \frac{1+\log(p)}{\log(p)-1}.$$

In the case $p = 2$ we have $\frac{\ln(1-\beta)}{\ln(\beta)} = \frac{\log(1/3)}{\log(2/3)} = \frac{\log(3)}{\log(3)-1}$.

For $p \geq \lambda$, β is equal to $\frac{1}{\lambda}$ and $\frac{\ln(1-\beta)}{\ln(\beta)} = \frac{\log_\lambda(\frac{\lambda-1}{\lambda})}{\log_\lambda(\frac{1}{\lambda})} = 1 - \log_\lambda(\lambda-1)$.

The evaluation of $\frac{\ln(1-\beta)}{\ln(\beta)}$ together with Eq. (5.29) gives the result.

Remark 5.3.4 Cor. 5.3.3 allows to estimate the expected number of invariant factors as a function of p and λ. The sum (5.18) with $\beta = \frac{2}{p+1}$ or $\beta = \frac{1}{\lambda}$ can also be evaluated numerically. For $p = 2$ this gives a bound 2.22 for the expected number of invariant factors divisible by 2. As the sum is decreasing with growing p, this is also an upper bound for any prime p, $p < \lambda$. For $\lambda \geq 2$ and $p \geq \lambda$ we obtain an analogous bound of 1.18.

Let us now consider sparse non-uniform distribution. By Lem. 5.3.2 we have the following corollary.

Corollary 5.3.5 Let $\mathcal{S}$ be a set of $\lambda > 1$ contiguous integers $\mathcal{S} = \{a, a+1, \ldots, a+\lambda-1\}$, for a certain $a \in \mathbb{Z}$. Let $\mathcal{P}_\mathcal{S}$ denote the discrete uniform distribution on $\mathcal{S}$. Let p be a prime and let $n \in \mathbb{N}$ and let A be a $n \times n$ matrix. Let $0 < \epsilon \leq 1$ and $C > 0$ be given and $\gamma : \mathbb{N} \ni n \to \gamma(n) \in (0,1)$ be such that $Cn^{-\epsilon} \leq \gamma(n)$.

Suppose that the entries of A are distributed according to $(\gamma(n), \mathcal{P}_\mathcal{S})$-sparse distribution, see Def. 5.1.6. The expected number of non-trivial invariant factors of A divisible by p is at most

$$C^{-1}n^\epsilon \frac{p+1}{p-1}(\epsilon \ln(n) - \ln(C) - \ln(\frac{p-1}{p+1})) + \frac{4}{3} \tag{5.30}$$

PROOF Let us first consider the distribution mod p induced by $(\gamma(n), \mathcal{P}_\mathcal{S})$-sparse distribution. First, on $\mathcal{S}$, the almost-uniform distribution $\mathcal{P}_{\mathcal{S},p}$ is induced, which is a $(\frac{1}{\lambda}\lfloor\frac{\lambda}{p}\rfloor, \frac{1}{\lambda}\lceil\frac{\lambda}{p}\rceil)$

distribution. Therefore, the matrix $A \mod p$ is given according to $(\gamma(n), \mathcal{P}_{\mathcal{S},p})$-sparse distribution. By Prop. 5.1.7 this means that

$$\mathcal{P}(x : x = x_0) \leq 1 - \gamma(n) + \gamma(n)\frac{1}{\lambda}\lceil\frac{\lambda}{p}\rceil \quad \forall x_0 \in \mathbb{Z}_p\,.$$

From Lem. 5.3.2 we know that the expected number of invariant factors $\mod p$ is less that or equal

$$\frac{\ln(1-\beta)}{\ln(\beta)} + \frac{4}{3},$$

where $\beta = 1 - \gamma(n) + \gamma(n)\frac{1}{\lambda}\lceil\frac{\lambda}{p}\rceil$ is the upper probability bound. As in the previous proof we have to evaluate $\frac{\ln(1-\beta)}{\ln(\beta)}$. The function is increasing with growing β. Therefore its value at $1 - \gamma(n) + \gamma(n)\frac{1}{\lambda}\lceil\frac{\lambda}{p}\rceil$ is less than the value at $1 - \gamma(n)(1 - \frac{2}{p+1})$ and less than the value at $(1 - Cn^{-\epsilon}(1 - \frac{2}{p+1}))$. We have

$$\frac{\ln(1-\beta)}{\ln(\beta)} \leq \frac{\ln(Cn^{-\epsilon}(1-\frac{2}{p+1}))}{\ln(1-Cn^{-\epsilon}(1-\frac{2}{p+1}))} \leq \frac{\epsilon\ln(n) - \ln(C) - \ln(1-\frac{2}{p+1})}{\ln(\frac{1}{1-Cn^{-\epsilon}(1-\frac{2}{p+1})})}.$$

By Taylor expansion we know that

$$\ln(\frac{1}{1-x}) = x + \sum_{i=2}^{\infty}\frac{1}{i}x^i \geq x.$$

Therefore

$$\frac{\ln(1-\beta)}{\ln(\beta)} \leq C^{-1}n^{\epsilon}\frac{p+1}{p-1}(\epsilon\ln(n) - \ln(C) - \ln(\frac{p-1}{p+1})).$$

To conclude, the expected number of invariant factors is less than or equal

$$C^{-1}n^{\epsilon}\frac{p+1}{p-1}(\epsilon\ln(n) - \ln(C) - \ln(\frac{p-1}{p+1})) + \frac{4}{3}.$$

Remark 5.3.6 In the asymptotic case, the condition $\gamma(n) \leq Cn^{-\epsilon}$ can be replaced by a weaker asymptotic condition $\gamma(n) \in O(n^{-\epsilon})$.

5.3.2 Case of Dense Matrices

In what follows we would like to evaluate the expected number of invariant factors for the integer Smith normal form. To do this, we have to put together the information on all primes. The information on the Smith form at each prime p is encoded in the local Smith form, see e.g. [39] for definition.

Definition 5.3.7 (Local Smith Form [39]) Let p be a prime. Let $k, n \in \mathbb{N}, k, n > 0$ and let A be a $k \times n$ matrix over $\mathbb{Z}$. Let $SF(A) = \mathrm{diag}(s_1(A), \ldots, s_{\min(k,n)}(A))$ be the Smith normal form of A. Let l denote the number of non-zero invariant factors and let $\mathrm{ord}_p(x)$ denote the order modulo p of integer x.

The local Smith form SF_p of A at p is a $k \times n$ diagonal matrix equal to

$$SF_p(A) = \mathrm{diag}(p^{\mathrm{ord}_p(s_1(A))}, \ldots, p^{\mathrm{ord}_p(s_l(A))}, 0, \ldots, 0).$$

Informally speaking, local Smith form contains all information on A at prime p, including the $\mathrm{rank}(A) \mod p$ and the Smith normal forms mod p for all $s \in \mathbb{N}, s > 0$. The (integer) Smith form of the matrix is equal to the product of its local Smith forms. Therefore the number of non-trivial invariant factors of A is the maximum of the number of non-trivial invariant factors of all local Smith forms of A at all primes p, which at its turn is equal to the number of invariant factors divisible by p. However, the maximum of the expected number of invariant factors divisible by p (given by Lem. 5.3.2) is not equal to the expected number of non-trivial invariant factors, as it constitutes only a lower bound for this value. Yet we may use the methods analogous to [44, Thm. 6.2] to treat all the primes at the same time.

Before we prove the main result (Thm. 5.3.10) on the expected number of invariant factors of an integer matrix, we need a lemma on prime summation.

Lemma 5.3.8 (Prime Summation) *Let $\lambda \in \mathbb{N}$ and let $j \in \mathbb{N}$, $j > 1$.*

We have the following inequality

$$\sum_{p \text{ prime}, 8<p<\lambda} \left(\frac{1}{\lambda}\lceil\frac{\lambda}{p}\rceil\right)^j \leq \left(\frac{1}{4}\right)^{j-1}.$$

PROOF First, we may assume that $\lambda > 11$ as the sum is empty in the opposite case.

We will consider separately the primes which fulfill the condition

$$2^k < \frac{\lambda}{p} \leq 2^{k+1} \tag{5.31}$$

for $k = 0, 1, \ldots, k_{max}$. Therefore the primes are divided into intervals

$$\frac{\lambda}{2^{k+1}} \leq p < \frac{\lambda}{2^k}, \quad k = 0, 1, \ldots k_{max}.$$

The value of k_{max} is computed as follows. As we require that $p > 8$ i.e $p \geq 11$, we obtain that k_{max} is the smallest integer to fulfill the condition

$$\frac{\lambda}{2^{k_{max}+1}} \leq 11.$$

By eventually adding some intervals without primes, we compute k_{max} using the condition

$$\frac{\lambda}{2^{k_{max}+1}} \leq 8 < \frac{\lambda}{2^{k_{max}}} \tag{5.32}$$

and obtain $k_{max} = \lceil \log(\lambda) \rceil - 4$.

By Eq. (5.31), in the kth interval, $\lceil \frac{\lambda}{p} \rceil$ is equal to 2^{k+1}. In the interval there are at most $\lceil \frac{\lambda}{2^{k+1}} \rceil$ integers, thus $\lceil \lceil \frac{\lambda}{2^{k+1}} \rceil / 2 \rceil = \lceil \frac{\lambda}{2^{k+2}} \rceil$ odd numbers.

From Eq. (5.32) we may conclude that $\frac{8}{2^2} < \frac{\lambda}{2^{k+2}}$ and consequently there are at least 3 odd numbers in each interval, at least one of them is divisible by 3 and is therefore composite. We may thus conclude that there are at most $\frac{\lambda}{2^{k+2}}$ primes in the kth interval.

We may calculate:

$$\begin{aligned}
&\sum_{p\ prime, 8<p<\lambda} \left(\frac{1}{\lambda} \lceil \frac{\lambda}{p} \rceil \right)^j \leq \sum_{k=0}^{\lceil \log(\lambda) \rceil - 4} \frac{\lambda}{2^{k+2}} \left(\frac{2^{k+1}}{\lambda} \right)^j \leq \sum_{k=0}^{\lceil \log(\lambda) \rceil - 4} \frac{1}{2} \left(\frac{2^{k+1}}{\lambda} \right)^{j-1} \\
&= \frac{1}{2\lambda^{j-1}} \sum_{k=0}^{\lceil \log(\lambda) \rceil - 4} 2^{(k+1)(j-1)} = \frac{1}{2\lambda^{j-1}} 2^{j-1} \frac{2^{(\lceil \log(\lambda) \rceil - 3)(j-1)} - 1}{2^{j-1} - 1} \\
&\leq \frac{1}{2\lambda^{j-1}} 2^{j-1} \frac{2^{(\log(\lambda)-2)(j-1)}}{2^{j-2}} = \frac{\lambda^{j-1} 2^{-2(j-1)}}{\lambda^{j-1}} = \frac{1}{4^{j-1}}.
\end{aligned}$$

which finishes the proof.

Remark 5.3.9 The method of evaluation showed in the previous proof may lead to another inequalities of the same type. For example we obtain

$$\sum_{p\ prime, 4<p<2^l} \left(\frac{1}{2^l} \lceil \frac{2^l}{p} \rceil \right)^j \leq \left(\frac{1}{4} \right)^{j-1}. \tag{5.33}$$

if $\lambda = 2^l$ is a power of 2. Indeed, preserving the notations from the proof of Lem. 5.3.8 the condition on k_{max} becomes

$$\frac{2^l}{2^{k_{max}+1}} \leq 4 < \frac{2^l}{2^{k_{max}}}$$

and we obtain $k_{max} = \lceil \log(2^l) \rceil - 3 = l - 3$. As $\lambda = 2^l$, there are at least $\frac{2^l}{2^{k+2}} = 2^{l-k-2}$ odd numbers in each interval and thus, at least 2^{l-k-2} primes. The final summation gives

$$\begin{aligned}
&\sum_{p\ prime, 4<p<2^l} \left(\frac{1}{2^l} \lceil \frac{2^l}{p} \rceil \right)^j \leq \sum_{k=0}^{l-3} 2^{l-k-2} \left(\frac{2^{k+1}}{2^l} \right)^j = \sum_{k=0}^{l-3} 2^{l-k-2+(k+1-l)j} \\
&= 2^{l-2+j-lj} \sum_{k=0}^{l-3} 2^{k(j-1)} \leq 2^{l-2+j-lj} \frac{2^{(l-2)(j-1)} - 1}{2^{j-1} - 1} \leq 2^{l-2+j-lj} \frac{2^{(l-2)(j-1)}}{2^{j-2}} \\
&= 2^{l-2+j-lj+lj-l-2j+2-j+2} = 2^{-2j+2} \leq \frac{1}{4^{j-1}}
\end{aligned}$$

All powers of primes p are included in the following inequality.

$$\sum_{p\ prime, p>3} \sum_{l:8<p^l<\lambda} \left(\frac{1}{\lambda}\lceil\frac{\lambda}{p^l}\rceil\right)^j \leq \frac{1}{4^{j-1}} \tag{5.34}$$

The proof of Lem 5.3.8 carry on in this case, as by excluding $p = 2, 3$ once again it suffices to count all odd numbers which are not divisible by 3.

The proofs are essentially the same as in 5.3.8 as we maintain the two conditions:

- in the kth interval there are at most $\frac{\lambda}{2^{k+2}}$ numbers to include in the sum;
- the number of intervals k_{max} enables final evaluation.

Main Theorem

Let us now compute the expected number of non-trivial invariant factors for a random dense almost uniformly distributed matrix. Previously known bound on the value for a square $n \times n$ matrix is $29 + 3\lceil\log_\lambda(n)\rceil$, by [44, Thm. 6.2]. In Thm. 5.3.10 be are able to give an asymptotically better result.

Theorem 5.3.10 (Expected Number of Non-trivial Factors) *Let $\mathcal{S}$ be a set of $\lambda > 1$ contiguous integers $\mathcal{S} = \{-\lceil\frac{\lambda}{2}\rceil + 1, \ldots, \lfloor\frac{\lambda}{2}\rfloor\}$. Let $k, n \in \mathbb{N}$ and let A be a $k \times n$ integer matrix with entries chosen randomly and uniformly from set $\mathcal{S}$, independently of each other. Let $k' = \min(k, n)$ and $n' = \max(k, n)$. The expected number of non-trivial invariant factors is at most*

$$\max(2, \left\lceil\sqrt{2\log_\lambda(k')}\right\rceil) + 3.$$

PROOF Without loss of generality we may assume that $k \leq n$ i.e. $k' = k$ and $n' = n$. In the opposite case, the reasoning can be performed for matrix A^T. The idea of the proof is similar to that of [44, Thm. 6.2] but we improve some bounds.

Let Dep_i denote an event that the first i rows of A are linearly **dep**endent (over rationals) and $\mathrm{MDep}_i(j)$, an event that the first i rows of A are **m**odularly **dep**endent and have modular rank $i - j$ i.e. that there exist at least one prime p such that $\mathrm{rank}_p(A_i) \leq i - j$, where A_i denote the submatrix consisting of first i rows of A.

Recall from [44, §6] that

$$\begin{aligned}
&\mathcal{P}(\mathrm{Dep}_1) \leq \lambda^{-n} \\
&\mathcal{P}\left(\mathrm{Dep}_i \wedge \neg\,\mathrm{Dep}_{i-1}\right) \leq \mathcal{P}\left(\mathrm{Dep}_i \mid \neg\,\mathrm{Dep}_{i-1}\right) \leq \lambda^{-n+i-1}.
\end{aligned}$$

This gives $\mathcal{P}(\mathrm{Dep}_i) \leq \frac{1}{\lambda^n} + \cdots + \frac{1}{\lambda^{n-i+1}}$ which is less than $\frac{1}{\lambda^{n-i+1}}\frac{\lambda}{\lambda-1}$.

The fact that the number of non trivial invariant factors is at least $j \geq 1$ (event I_j) implies that either Dep_{k-j+1} or $\mathrm{MDep}_{k-j+i}(i)$ hold, for any $i = 1 \dots j$. The probability $\mathcal{P}\left(\mathrm{MDep}_{k-j+i}(i) \vee \mathrm{Dep}_{k-j+1}\right)$ can be transformed to

$$\mathcal{P}\left(\mathrm{MDep}_{k-j+i}(i) \vee \mathrm{Dep}_{k-j+1}\right) = \mathcal{P}\left((\mathrm{MDep}_{k-j+i}(i) \wedge \neg\,\mathrm{Dep}_{k-j+1}) \vee \mathrm{Dep}_{k-j+1}\right),$$

and both $\mathcal{P}\left(\mathrm{MDep}_{k-j+i}(i) \wedge \neg\,\mathrm{Dep}_{k-j+1}\right)$ and $\mathcal{P}\left(\mathrm{Dep}_{k-j+1}\right)$ can be treated separately. To estimate $\mathcal{P}\left(\mathrm{MDep}_{k-j+i}(i) \wedge \neg\,\mathrm{Dep}_{k-j+1}\right)$ we will sum over all possible primes to obtain

$$\mathcal{P}\left(\mathrm{MDep}_{k-j+i}(i) \wedge \neg\,\mathrm{Dep}_{k-j+1}\right) \leq \sum_{p\ prime} \mathcal{P}\left(\mathrm{rank}_p(A_{k-j+i}) \leq k-j\right).$$

Since Dep_{k-j+1} does not hold, there exists a $(k-j+1) \times (k-j+1)$ non-zero minor, and we have to sum over the primes which divide it. We will treat separately primes $p < \lambda$ and $p \geq \lambda$.

We set $\beta_p = \frac{2}{p+1}$ for $p < \lambda$ and $\beta_p = \frac{1}{\lambda}$ for $p \geq \lambda$. Then, let $B_{i,p}$ denote

$$B_{i,p} = \prod_{l=1}^{i} \frac{1}{1-\beta_p^l}.$$

We notice that $B_{i,p_1} \geq B_{i,p_2}$ for $p_1 \leq p_2$ and that $B_{i,p} \leq (\frac{1}{1-\beta_p})^i$. By applying Lem. 5.2.3 for each prime p, we have

$$\mathcal{P}(\mathrm{rank}_p(A_{k-j+i}) \leq k-j) \leq \beta_p^{(n-k+j)i} B_{i,p}$$

and therefore for $\lambda > 11$

$$\sum_{p<\lambda} \mathcal{P}(\mathrm{rank}_p(A_{k-j+i}) \leq k-j) \leq B_{i,2}\beta_2^{(n-k+j)i} + B_{i,3}\beta_3^{(n-k+j)i} + B_{i,5}\beta_5^{(n-k+j)i} + B_{i,7}\beta_7^{(n-k+j)i} + \sum_{8<p<\lambda} B_{i,p}\beta_p^{(n-k+j)i}. \tag{5.35}$$

Thanks to Lem. 5.3.8, for $i \geq 2$ the sum $\sum_{8<p<\lambda} B_{i,p}\beta_p^{(n-k+j)i}$ can be bounded by $B_{i,11}\left(\frac{1}{4}\right)^{(n-k+j)i-1}$. Notice, that we include only those terms, for which $p < \lambda$. Therefore for $\lambda \leq 11$ the sum will contain less numbers. The bounds for $\sum_{p<\lambda} \mathcal{P}(\mathrm{rank}_p(A_{k-j+i}) \leq k-j)$ are summarized in Tab. 5.1.

For primes $p \geq \lambda$ we should estimate the number of primes dividing the $(k-j+1)$th minor. By the Hadamard's bound (notice that Dep_{k-j+1} does not hold), the minors are bounded in absolute value by $\left((k-j+1)\left(\frac{\lambda}{2}\right)^2\right)^{\frac{k-j+1}{2}}$. Therefore the number of primes $p \geq \lambda$ dividing the minor is at most $\frac{k}{2}(\log_\lambda(k)+2)$. This results in a bound

$$\sum_{p\geq\lambda} \mathcal{P}(\mathrm{rank}_p(A_{k-j+i}) \leq k-j) \leq \frac{k}{2}\left(\log_\lambda(k)+2\right)\frac{1}{\lambda^{(n-k+j)i}}\left(\frac{\lambda}{\lambda-1}\right)^i$$

λ	$\sum_{p<\lambda} \mathcal{P}(\mathrm{rank}_p(A_{k-j+i}) \leq k-j)$ is less than or equal
$2 < \lambda \leq 3$	$B_{i,2}\beta_2^{(n-k+j)i}$
$3 < \lambda \leq 5$	$B_{i,2}\beta_2^{(n-k+j)i} + B_{i,3}\beta_3^{(n-k+j)i}$
$5 < \lambda \leq 7$	$B_{i,2}\beta_2^{(n-k+j)i} + B_{i,3}\beta_3^{(n-k+j)i} + B_{i,5}\beta_5^{(n-k+j)i}$
$7 < \lambda \leq 11$	$B_{i,2}\beta_2^{(n-k+j)i} + B_{i,3}\beta_3^{(n-k+j)i} + B_{i,5}\beta_5^{(n-k+j)i} + B_{i,7}\beta_7^{(n-k+j)i}$
$11 < \lambda$	$B_{i,2}\beta_2^{(n-k+j)i} + B_{i,3}\beta_3^{(n-k+j)i} + B_{i,5}\beta_5^{(n-k+j)i} + B_{i,7}\beta_7^{(n-k+j)i} + B_{i,11}\left(\frac{1}{4}\right)^{(n-k+j)i-1}$

Table 5.1: Bounds for $\sum_{p<\lambda} \mathcal{P}(\mathrm{rank}_p(A_{k-j+i}) \leq k-j)$.

Summarizing,

$$\mathcal{P}\left(\left(\mathrm{MDep}_{k-j+i}(i) \wedge \neg \mathrm{Dep}_{k-j+1}\right) \vee \mathrm{Dep}_{k-j+1}\right) \leq \sum_{p<\lambda} \mathcal{P}(\mathrm{rank}_p(A_{k-j+i}) \leq k-j)$$
$$+ \frac{k}{2}\left(\log_\lambda(k)+2\right)\left(\frac{\lambda}{\lambda-1}\right)^i \frac{1}{\lambda^{(n-k+j)i}} + \frac{\lambda}{\lambda-1}\lambda^{-(n-j+1)}, \tag{5.36}$$

where the bounds for $\sum_{p<\lambda} \mathcal{P}(\mathrm{rank}_p(A_{k-j+i}) \leq k-j)$ are given in Tab. 5.1

We can now compute the expected number of non-trivial invariant factors.

Let us fix $h = \max(2, \left\lceil \sqrt{2\log_\lambda(k)} \right\rceil)$. In particular, we have that $\mathcal{P}(I_j)$ is less than $\mathcal{P}\big((\mathrm{MDep}_{k-j+h}(h) \wedge \neg \mathrm{Dep}_{k-j+1}) \vee \mathrm{Dep}_{k-j+1}\big)$. We can check that $h^2 \geq \log_\lambda(k) + \log_\lambda(\log_\lambda(k)+2)$. This gives also

$$\lambda^{h^2} \geq k\left(\log_\lambda(k)+2\right) \tag{5.37}$$
$$1 \geq \frac{k}{2}\left(\log_\lambda(k)+2\right)\left(\frac{\lambda}{\lambda-1}\right)^h \left(\frac{\lambda^h}{\lambda^h-1}\right)\frac{1}{\lambda^{h(h+1)}}. \tag{5.38}$$

The expected number of non trivial invariant factors is bounded by:

$$\sum_{j=1}^{h} 1 + \sum_{j=h+1}^{k} \mathcal{P}\left(\left(\mathrm{MDep}_{k-j+h}(h) \vee \neg \mathrm{Dep}_{k-j+1}\right) \wedge \mathrm{Dep}_{k-j+1}\right),$$

see Eq. (5.36). The sum is maximal in the case of a square matrix, for $n-k=0$, and we may continue the evaluation in this case. For $\lambda > 11$:

$$h + B_{h,2}\frac{\beta_2^{h(h+1)}}{1-\beta_2^h} + B_{h,3}\frac{\beta_3^{h(h+1)}}{1-\beta_3^h} + B_{h,5}\frac{\beta_5^{h(h+1)}}{1-\beta_5^h} + B_{h,7}\frac{\beta_7^{h(h+1)}}{1-\beta_7^h} + B_{h,11}\left(\frac{1}{4}\right)^{h(h+1)-1}\frac{4^h}{4^h-1}$$
$$+ \frac{1}{\lambda}\left(\frac{\lambda}{\lambda-1}\right)^2 + \frac{k}{2}\left(\log_\lambda(k)+2\right)\left(\frac{\lambda}{\lambda-1}\right)^h \frac{1}{\lambda^{h(h+1)}}\left(\frac{\lambda^h}{\lambda^h-1}\right) \leq^{(5.38)} h + f(h,\lambda) + 1.$$

λ	$f(h,\lambda)$	bound
$\lambda = 2$	$\frac{1}{\lambda}\left(\frac{\lambda}{\lambda-1}\right)^2$	2
$\lambda = 3$	$\frac{1}{\lambda}\left(\frac{\lambda}{\lambda-1}\right)^2 + B_{h,2}\frac{\beta_2^{h(h+1)}}{1-\beta_2^h}$	1.61
$\lambda = 4,5$	$\frac{1}{\lambda}\left(\frac{\lambda}{\lambda-1}\right)^2 + B_{h,2}\frac{\beta_2^{h(h+1)}}{1-\beta_2^h} + B_{h,3}\frac{\beta_3^{h(h+1)}}{1-\beta_3^h}$	1.36
$\lambda = 6,7$	$\frac{1}{\lambda}\left(\frac{\lambda}{\lambda-1}\right)^2 + B_{h,2}\frac{\beta_2^{h(h+1)}}{1-\beta_2^h} + B_{h,3}\frac{\beta_3^{h(h+1)}}{1-\beta_3^h} + B_{h,5}\frac{\beta_5^{h(h+1)}}{1-\beta_5^h}$	1.16
$7 < \lambda \leq 11$	$\frac{1}{\lambda}\left(\frac{\lambda}{\lambda-1}\right)^2 + B_{h,2}\frac{\beta_2^{h(h+1)}}{1-\beta_2^h} + B_{h,3}\frac{\beta_3^{h(h+1)}}{1-\beta_3^h} + B_{h,5}\frac{\beta_5^{h(h+1)}}{1-\beta_5^h} + B_{h,7}\frac{\beta_7^{h(h+1)}}{1-\beta_7^h}$	1.08
$11 < \lambda$	$\frac{1}{\lambda}\left(\frac{\lambda}{\lambda-1}\right)^2 + B_{h,2}\frac{\beta_2^{h(h+1)}}{1-\beta_2^h} + B_{h,3}\frac{\beta_3^{h(h+1)}}{1-\beta_3^h} + B_{h,5}\frac{\beta_5^{h(h+1)}}{1-\beta_5^h}$ $+B_{h,7}\frac{\beta_7^{h(h+1)}}{1-\beta_7^h} + B_{h,11}\left(\frac{1}{4}\right)^{h(h+1)-1}\frac{4^h}{4^h-1}$	0.4

Table 5.2: Bounds for $f(h,\lambda)$.

Analogous bounds are obtained for $\lambda \leq 11$, where the sum $\sum_{p<\lambda}\mathcal{P}(\mathrm{rank}_p(A_{k-j+h}) \leq k-j)$ contains less values. For $\lambda > 11$, $f(h,\lambda)$ is equal to

$$f(h,\lambda) = B_{h,2}\frac{\beta_2^{h(h+1)}}{1-\beta_2^h} + B_{h,3}\frac{\beta_3^{h(h+1)}}{1-\beta_3^h} + B_{h,5}\frac{\beta_5^{h(h+1)}}{1-\beta_5^h}$$
$$+ B_{h,7}\frac{\beta_7^{h(h+1)}}{1-\beta_7^h} + B_{h,11}\left(\frac{1}{4}\right)^{h(h+1)-1}\frac{4^h}{4^h-1} + \frac{1}{\lambda}\left(\frac{\lambda}{\lambda-1}\right)^2 \leq 0.4.$$

The function $f(h,\lambda)$ is decreasing with h and λ. By taking minimal $h = 2$, $beta_p = \frac{2}{p+1}$ and $B_{2,p} = \frac{1}{(1-\beta_p)(1-\beta_p^2)}$ and $\lambda = 12$, we obtain the bound. For $\lambda \leq 11$ the sum contains less numbers. Tab. 5.2 summarizes the value of $f(h,\lambda)$ and the bounds for all cases of λ. We may conclude that $f(h,\lambda) \leq 2$.

For a rectangular matrix, $n-k > 0$, the result is even sharper.

Putting it together. The expected number of non-trivial invariant factors is less than or equal $h+3 = \max(2, \lceil\sqrt{2\log_\lambda k}\rceil) + 3$.

Corollary 5.3.11 Let $\mathcal{S}$ be a set of $\lambda > 1$ contiguous integers $\mathcal{S} = \{-\lceil\frac{\lambda}{2}\rceil + 1, \ldots, \lfloor\frac{\lambda}{2}\rfloor\}$. Let $n \in \mathbb{N}$ and let A be a $n \times n$ integer matrix with entries chosen randomly and uniformly from set $\mathcal{S}$, independently of each other. Let $h = \max(2, \lceil\sqrt{2\log_\lambda(n)}\rceil)$. The probability that the number of non-trivial invariant factors is at least $J = \max(\sqrt{3/2}h + 1, h+2)$ is

$$\mathcal{P}(I_J) \in O(n^{-1})$$

PROOF Let us consider the notions of the proof of Thm. 5.3.10. Let A_i denote the submatrix consisting of first i rows of A. Let I_J denote the event that at least J invariant factors of A are non-trivial. Eq. (5.36) implies that for any $i = 1 \ldots J$

$$\mathcal{P}(I_J) \leq \sum_{p<\lambda}\mathcal{P}(\mathrm{rank}_p(A_{n-J+i}) \leq n-J) + \frac{n}{2}(\log_\lambda(n)+2)\left(\frac{\lambda}{\lambda-1}\right)^i\frac{1}{\lambda^{Ji}} + \frac{\lambda}{\lambda-1}\lambda^{-(n-J+1)}.$$

p	β_p	$B_{J,p}$	$\beta_p^{J^2} B_{J,p} \leq$
2	$\frac{2}{3}$	3^J	$\left(\frac{2}{3}\right)^{(J-2)^2}$
3	$\frac{1}{2}$	2^J	$\left(\frac{1}{2}\right)^{(J-1)^2}$
5	$\frac{1}{3}$	$\left(\frac{3}{2}\right)^J$	$\left(\frac{1}{3}\right)^{(J-1)^2}$
7	$\frac{1}{4}$	$\left(\frac{4}{3}\right)^J$	$\left(\frac{1}{4}\right)^{(J-1)^2}$
11	$\frac{1}{6}$	$\left(\frac{6}{5}\right)^J$	$\frac{\left(\frac{1}{6}\right)^{(J-1)^2}}{\left(\frac{1}{4}\right)^{J^2-1}}$

Table 5.3: Bounds for $p = 2, 3, 5, 7, 11$.

Let us take $i = J$. Then

$$\mathcal{P}(I_J) \leq \sum_{p<\lambda} \mathcal{P}(\operatorname{rank}_p(A) \leq n - J) + \frac{n}{2}\left(\log_\lambda(n) + 2\right)\left(\frac{\lambda}{\lambda - 1}\right)^J \frac{1}{\lambda^{J^2}} + \frac{\lambda}{\lambda - 1}\lambda^{-(n-J+1)}.$$

We will bound the terms separately.

By Eq. (5.35) for $\lambda > 11$

$$\sum_{p<\lambda} \mathcal{P}(\operatorname{rank}_p(A) \leq n - J) \leq B_{J,2}\beta_2^{J^2} + B_{J,3}\beta_3^{J^2} + B_{J,5}\beta_5^{J^2} + B_{J,7}\beta_7^{J^2} + B_{J,11}\left(\frac{1}{4}\right)^{J^2-1},$$

where $\beta_p = \frac{2}{p+1} < 1$ for $p < \lambda$ and $B_{J,p} = \prod_{l=1}^{J} \frac{1}{1-\beta_p^l}$. See Tab. 5.1 for the case $\lambda \leq 11$. We bound the terms in Tab. 5.3. The values for p assume that $p < \lambda$, otherwise, the corresponding value is not included in the bound anyway. We have

$$\left(\frac{2}{3}\right)^{(J-2)^2} \geq \left(\frac{1}{2}\right)^{(J-1)^2} \geq \left(\frac{1}{3}\right)^{(J-1)^2} \geq \left(\frac{1}{4}\right)^{(J-1)^2} \geq \left(\frac{1}{6}\right)^{(J-1)^2}.$$

For $J = \max(\sqrt{3/2}h + 1, h + 2)$, $h = \max(2, \lceil\sqrt{2\log_\lambda(n)}\rceil), \lambda > 1$ we obtain that

$$\left(\frac{2}{3}\right)^{(J-2)^2} \leq \left(\frac{2}{3}\right)^{h^2} \leq \left(\frac{2}{3}\right)^{2\log_\lambda(n)} \leq n^{2\log_\lambda(\frac{2}{3})} \leq n^{-1}$$

Therefore $\sum_{p<\lambda} \mathcal{P}(\operatorname{rank}_p(A) \leq n - J) \in O(n^{-1})$.

Let us consider

$$\frac{n}{2}\left(\log_\lambda(n) + 2\right)\left(\frac{\lambda}{\lambda - 1}\right)^J \frac{1}{\lambda^{J^2}}$$

for $J = \max(\sqrt{3/2}h + 1, h + 2)$. By Eq. (5.37) $\frac{n}{2}\left(\log_\lambda(n) + 2\right) \leq \lambda^{h^2}$. Therefore

$$\frac{n}{2}\left(\log_\lambda(n) + 2\right)\left(\frac{\lambda}{\lambda - 1}\right)^J \frac{1}{\lambda^{J^2}} \leq \lambda^{h^2}\left(\frac{\lambda}{\lambda - 1}\right)^{\sqrt{3/2}h+1} \frac{1}{\lambda^{\frac{3}{2}h^2+2\sqrt{3/2}h+1}} \leq \frac{1}{\lambda^{\frac{h^2}{2}}}$$

For $h = \max(2, \lceil \sqrt{2\log_\lambda(n)} \rceil), \lambda > 1$ this is less than or equal to

$$\frac{1}{\lambda^{\frac{h^2}{2}}} \leq \frac{1}{\lambda^{\frac{2\log_\lambda(n)}{2}}} = n^{-1}.$$

Hence $\frac{n}{2}\left(\log_\lambda(n) + 2\right)\left(\frac{\lambda}{\lambda-1}\right)^J \frac{1}{\lambda^{J^2}} \in O(n^{-1})$ as well.

Finally,

$$\frac{\lambda}{\lambda - 1}\lambda^{-(n-J+1)} \leq \lambda^{-n/2} \in O(n^{-1})$$

Hence, we conclude that the sum (18) is $O(n^{-1})$, which finishes the proof.

5.3.3 Case of Sparse Matrices

In [7] the authors consider a special case of sparse distributions modulo p, where the probability of getting a 0 entry is considerably higher than the probability of getting a nonzero entry and non-zero entries are uniformly distributed modulo p. Namely, for $0 \leq \gamma \leq 1$ and a prime p they consider a probability distribution on $\mathbb{Z}_p$ given by a density function ξ

$$\begin{aligned} \xi(0) &= 1 - \gamma \\ \xi(x_0) &= \frac{\gamma}{p-1}, \quad x_0 \neq 0. \end{aligned}$$

By putting $\gamma = (p-1)/p$ one obtains the uniform distribution on $\mathbb{Z}_p$. In [7, Cor. 2.3] the authors prove, that the expected rank defect (i.e. the difference between the dimension and the rank) for a square $n \times n$ matrix is finite, provided that $\gamma \geq \frac{ln(n)-c}{n}$ for some constant c. Notice, that the defect is exactly the number of zero invariant for the Smith form modulo p in this case. In other words, the defect modulo p is the number invariant factors divisible by p.

The authors relate the defect of the matrix to the number of linear dependencies of the rows of the matrix, which is an effective approach for this kind of distributions. The expected number of linear dependencies M is

$$\mathbf{E}_\xi(M) = \sum_{i=1}^{n} \binom{n}{k} (1 - \frac{1}{p})^k \frac{1}{p^{n-k}} \left(1 + (q-1)(1 - \frac{p\gamma}{p-1})^k\right)^n$$

and by [7, Lem. 3.2] $p^{n-\mathrm{rank}(A)} - 1 = M(A)$ for any $n \times n$ matrix A.

This leads to the following result on the expected number of non-trivial invariant factors modulo a prime p for sparse matrices.

Theorem 5.3.12 (Cor. 2.3. of [7]) *Let $n \in \mathbb{N}$ and p be a prime. Let $0 \leq c(n) \leq \log(n)$ be a bounded function and let $1 \geq \gamma > 0$, $\gamma \geq \frac{\log(n)-c(n)}{n}$. Let $M = [m_{ij}]$ be a $n \times n$ matrix over a finite field $\mathbb{Z}_p$ with entries chosen randomly, independently of each other, with a probability $\mathcal{P}$ such that*

$$\mathcal{P}(m_{ij} = 0) = 1 - \gamma$$
$$\mathcal{P}(m_{ij} = x_0) = \frac{\gamma}{p-1}, \quad x_0 \neq 0.$$

for each entry. Then the expected number of non-trivial invariant factors ($\mod p$) of M is $O(1)$.

PROOF See [7, Cor 2.3]. See also Thm. 2.1, Thm. 2.2 and Thm. 6.1 of [7] for associated results.

Now, let us consider a sparse integer matrix. We have the following corollary.

Corollary 5.3.13 Let $n \in \mathbb{N}$ and $1 \geq \gamma > 0$, $\gamma \geq \frac{\log(n)-c(n)}{n}$. Let $\mathcal{S}$ be a set of $\lambda > 1$ contiguous integers, $\mathcal{S} = \{-\lceil \frac{\lambda}{2} \rceil + 1, \ldots, \lfloor \frac{\lambda}{2} \rfloor\}$. Let us consider a random $n \times n$ integer matrix $A = [a_{ij}]$ with entries chosen independently of each other, in such a way, that the non-zero entries are with a probability $\mathcal{P}$ such that

$$\mathcal{P}(a_{ij} = 0) = 1 - \gamma$$
$$\mathcal{P}(a_{ij} = x_0) = \frac{\gamma}{\lambda - 1}, \quad x_0 \in \mathcal{S} \setminus \{0\}.$$

for each entry. Then the expected number of invariant factors of A not divisible by $p = 2$ is $O(1)$.

PROOF The distribution $\mathcal{P}$ of entries over $\mathbb{Z}$ induces a distribution of entries modulo 2 such that $\mathcal{P}(a_{ij} = 1 \mod 2) = p_1 = \gamma \lfloor \frac{\lambda}{2} \rfloor \geq \gamma$ and $\mathcal{P}(a_{ij} = 0 \mod 2) = 1 - p_1$.

Thm. 5.3.12 can be applied directly in this case and allows us to conclude that the expected number of non-trivial invariant factors of $A \mod 2$ is finite. This is equal to the expected number of invariant factors of A divisible by 2.

The approach of [7] is indeed equivalent to the approach of Sec. 5.1.3. Namely, we can define the probability density function η by

$$\eta(0) = 1 - \gamma' + \gamma' \frac{1}{p}$$
$$\eta(x_0) = \frac{\gamma'}{p}, \quad x_0 \neq 0.$$

The two distributions η and ξ are equivalent for $\gamma' = \gamma \frac{p}{p-1}$. For $p > 2$ similar reasoning as in [7] yields

$$\mathbf{E}_\xi(M) = \sum_{i=1}^{n} \binom{n}{k} (1 - \frac{1}{p})^k \frac{1}{p^{n-k}} \left(1 + (q-1)(1-\gamma')^k\right)^n.$$

The sparse distribution naturally extends to the case of non-uniform (α, β) distributions on $\mathbb{Z}_p$ by

$$\begin{aligned} 1-\gamma'+\gamma'\alpha \leq \xi(0) \leq 1-\gamma'+\gamma'\beta \\ \gamma'\alpha \leq \xi(x_0) \leq \gamma'\beta, \quad x_0 \neq 0, \end{aligned}$$

where $p\beta \geq 1$. This lead to estimations of the expected number of linear dependencies by the sum

$$\mathbf{E}_\xi(M) = \sum_{i=1}^{n} \binom{n}{k} (1-\frac{1}{p})^k \frac{1}{p^{n-k}} \left(p\beta + p(1-\beta)(1-\gamma')^k\right)^n,$$

which diverges as n goes to infinity. This means that sparse distribution does not generalize well to the non-uniform case. In [21, 20] some further analysis and asymptotic results are presented, but it does not seem to generalize to the non-uniform case neither.

Instead, we will prove that the expected number of non-trivial invariant factors divisible by $p > 2$ is less than or equal to the expected number of non-trivial invariant factors divisible by 2.

We will need the following assumption, which seems natural yet might be difficult to prove rigorously. In short, we require that the probability that a prime p divides the determinant divisors of a random matrix A is decreasing as p increases.

Let ξ be a random distribution of integers. Let A be a ξ-distributed random matrix. For $i = 1, \ldots, n$, let d_i be the gcd of all $i \times i$ minors of A. Then for $i = 1, \ldots, n$ and for primes $p > p'$, the probability

$$\mathcal{P}_\xi(p \mid d_i) \leq \mathcal{P}_\xi(p' \mid d_i). \tag{5.39}$$

Theorem 5.3.14 *Let ξ be a random distribution of integers. Let $n \in \mathbb{N}, n > 0$ and let A be $n \times n$ ξ-distributed random matrix. Let us assume that Eq. (5.39) holds. Let $p > p'$ be two primes. Let I_p and $I_{p'}$ denote the number of non-trivial invariant factors divisible by p and p' respectively. Then the expected value of $I_p - I_{p'}$ is non-negative.*

PROOF We have

$$\mathbf{E}(I_{p'} - I_p) = \sum_{i=-n}^{n} i\,\mathcal{P}(I_{p'} - I_p = i) = \sum_{i=1}^{n} \mathcal{P}(I_{p'} - I_p \geq i) - \sum_{i=1}^{n} \mathcal{P}(I_p - I_{p'} \geq i).$$

The event $I_{p'} - I_p \geq i$ means that there exists $k = 0, \ldots, n-i-1$ such that $d_1 = \cdots = d_k = 1$ and $p' \mid d_{k+1}$ (i.e. $p' \mid s_{k+1} = d_{k+1}$), and $p \nmid d_{k+1}, \ldots, p \nmid d_{k+i}$. We have

$$\mathcal{P}(I_{p'} - I_p \geq i) = \mathcal{P}(\exists k = 0, \ldots, n-i-1 : d_1 = \cdots = d_k, p' \mid d_{k+1}, p \nmid d_{k+1}, \ldots, p \nmid d_{k+i}).$$

By Eq. (5.39) this is less than or equal to

$$\mathcal{P}(\exists k = 0, \dots, n-i-1 : d_1 = \dots = d_k, p \mid d_{k+1}, p' \nmid d_{k+1}, \dots, p' \nmid d_{k+i}) = \mathcal{P}(I_p - I_{p'} \geq i).$$

Thus,

$$\mathbf{E}(I_{p'} - I_p) = \sum_{i=1}^{n} [\mathcal{P}(I_{p'} - I_p \geq i) - \mathcal{P}(I_p - I_{p'} \geq i)] \geq 0,$$

which finishes the proof.

Corollary 5.3.15 Let $n \in \mathbb{N}$ and $1 \geq \gamma > 0$, $\gamma \geq \frac{\log(n) - c(n)}{n}$. Let $\mathcal{S}$ be a set of $\lambda > 1$ contiguous integers, $\mathcal{S} = \{-\lceil \frac{\lambda}{2} \rceil + 1, \dots, \lfloor \frac{\lambda}{2} \rfloor\}$. Let us consider a random $n \times n$ integer matrix $A = [a_{ij}]$ with entries chosen independently of each other, in such a way, that the non-zero entries are with a probability $\mathcal{P}$ such that

$$\begin{aligned} \mathcal{P}(a_{ij} = 0) &= 1 - \gamma \\ \mathcal{P}(a_{ij} = x_0) &= \frac{\gamma}{\lambda - 1}, \quad x_0 \in \mathcal{S} \setminus \{0\}. \end{aligned}$$

for each entry. Then the expected number of invariant factors of A not divisible by a prime $p > 2$ is finite, under the assumption that Eq. (5.39) holds for $\mathcal{P}$.

PROOF Let I_p denote the number of non-trivial invariant factors divisible by p. Then

$$\mathbf{E}(I_p) = \mathbf{E}(I_2) - \mathbf{E}(I_2 - I_p).$$

By Cor. 5.3.13, $\mathbf{E}(I_2)$ is finite and by Thm. 5.3.14, $\mathbf{E}(I_2 - I_p)$ is non negative. Hence, $\mathbf{E}(I_p)$ is not greater than $\mathbf{E}(I_2)$ and finite.

Cor. 5.3.13 and 5.3.15 are not sufficient to give a nice estimation on the expected number of non-trivial invariant factors of a sparse integer matrix. Yet, under the assumption of Eq. (5.39), they provide an important clue that the expected number of non-trivial invariant factors is not extremely large in this case.

Notice, that the matrix cannot be too sparse. In the case when the expected number of non-zero elements in row/colum is finite, the expected number of invariant factors modulo 2 is linear in n. Indeed, a row/column of a matrix with k non-zero entries is 0 mod 2 with probability β_2^k, if β_2 is the probability that a non-zero entry is 0 mod 2. This leads to a linear number of 0 rows modulo 2. This might be the case of diagonal and bi-diagonal etc. matrices, which thus could have a significant number of non-trivial invariant factors.

5.4 The Determinant of Random Almost Uniformly Distributed Matrices

In his thesis, Wan [127] gives some results on almost uniformly distributed variables modulo p, where p is a prime. This allows him to use multiplicative preconditioning in the Smith form adaptive algorithm of [44] and obtain a Monte Carlo type algorithm.

In what follows, we are going to follow the reasoning of [127] in order to characterize the expected behavior of his preconditioning. The ultimate goal of this section is to bound the probability that p^s divides the determinant of a random matrix, see Thm. 5.4.5. To prove this, we have to restate the results of Wan's thesis in the case of almost uniformly distributed variables modulo p^s, where p is a prime and $s \in \mathbb{N}, s > 1$. The major difference between those cases is that $\mathbb{Z}_p$ is a field, and $\mathbb{Z}_{p^s}$, for $s > 1$ is only a ring with several 0 divisors.

In the case where $s = 1$, the problem of the determinant modulo p has been consider several times, see for example [127] and the references therein. He achieves the bound $\frac{3}{p}$ for the probability that a prime $p \leq \lambda$ divides the determinant of a random integer matrix. This bound will be the departure point in our consideration.

In the case of uniform distribution, Brent and McKay give in [10, Cor. 2.2] a bound that the determinant is determined over p^s, for $s \in \mathbb{N}$. It is however not possible to transform this result to the case of almost-uniform distribution without revisiting their proofs and theorems. In fact, we performed our analysis independently of [10], it seems however that the tools that we use are similar.

The setup for almost uniform distribution has been given in Sec. 5.1.2. The notion of random matrices is detailed in Sec. 5.1.3.

The outline of this section is as follows. In Sec. 5.4.1 we present some preliminary results that will allow us to proof Thm. 5.4.5 in Sec. 5.4.2. In Lem. 5.4.1 we restate the basic results of [127] in the mod p^s case for almost uniformly distributed vectors. Then in Lem. 5.4.3 and 5.4.4 we extend the result to almost uniformly distributed matrices to obtain characterization that we need in the proof of Thm. 5.4.5. The proof of Thm. 5.4.5 is given in section 5.4.2.

5.4.1 Preliminary Results

The following lemma gives analogues to lemmas 5.10, 5.11 in [127] in the case of ring $\mathbb{Z}_{p^s}$. It proves in particular that the results of dot product and matrix-vector product are almost uniformly distributed.

For the rest of this section let us assume that $=_{p,s}$ denote an equality mod p^s. It requires that all operations in the $=_{p,s}$ equation can be performed modulo p^s, in particular, this is the case of the inverse operation.

Lemma 5.4.1 (Distribution of $\boldsymbol{t} \cdot \boldsymbol{x}$ and $\boldsymbol{A}\boldsymbol{x}$) *Let $\mathcal{S}$ be a set of λ contiguous integers $\mathcal{S} = \{a, a+1, \ldots, a+\lambda-1\}$, where $a \in \mathbb{Z}$ and $\lambda > 0$. Let p be a prime and $s \in \mathbb{N}, s > 0$.*

Let $\mathcal{P}_{\mathcal{S}}$ denote the uniform distribution on $\mathcal{S}$. Let $\mathcal{P}$ denote the distribution of vectors of $\mathcal{S}^n$ such that the entries of the vector are chosen independently of each other according to distribution $\mathcal{P}_{\mathcal{S}}$. Let

$$\alpha_{p,s} = \frac{1}{\lambda}\lfloor\frac{\lambda}{p^s}\rfloor, \quad \beta_{p,s} = \frac{1}{\lambda}\lceil\frac{\lambda}{p^s}\rceil.$$

(i) cf. [127, Lem. 5.10] *Let $n \in \mathbb{N}, n > 0$. Let $t \in \mathbb{Z}^n$, $t \neq 0 \mod p$ and $d \in \mathbb{Z}_{p^s}$ be given. Then the probability that a random vector $x \in \mathcal{S}^n$ is chosen according to distribution $\mathcal{P}$, such that $t \cdot x = d \mod p^s$ is $\alpha_{p,s} \leq \mathcal{P}(x : t \cdot x = d \mod p^s) \leq \beta_{p,s}$.*

(ii) cf. [127, Lem. 5.11] *Let $r, m, n \in \mathbb{N}, m, n \geq r > 0$. Let $A \in \mathbb{Z}^{m\times n}$ be a matrix over $\mathbb{Z}$ such that the local Smith form of A at p is trivial i.e. that $SF_p(A) = \mathrm{diag}(\underbrace{1,\ldots,1}_{r},0,\ldots,0)$; let $b \in \mathbb{Z}_{p^s}^m$ be given. Then the probability that a random vector $x \in \mathcal{S}^n$ is chosen according to distribution $\mathcal{P}$, such that $Ax = b \mod p^s$ is*

$$\alpha_{p,s}^r \leq \mathcal{P}\left(x : Ax = b \mod p^s\right) \leq \beta_{p,s}^r.$$

PROOF For (i) the proof of 5.10 from [127] carry on as, by assumption on t it is possible to choose an entry invertible mod p^s.

For (ii) we slightly modify the proof of 5.11 from [127]. As $\mathbb{Z}_{p^s}, s > 1$ is no longer a field, the proof is slightly more technical. Since A has a trivial Smith form modulo p there exist two matrices L and R in $\mathbb{Z}_{p^s}^{m\times m}$ and $\mathbb{Z}_{p^s}^{n\times n}$ respectively, such that $\det(L), \det(R) \neq 0 \mod p$ and $A =_{p,s} L \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} R' \\ R'' \end{bmatrix}$, where $R' = [R_{ij}]_{i=1..r, j=1..n}$ is a $r \times n$ upper minor of R and R'' is the $(n-r)\times n$ lower minor. We may therefore transform

$$\mathcal{P}(x \in \mathcal{S}^n : Ax =_{p,s} b) = \mathcal{P}(x \in \mathcal{S}^n : R'x =_{p,s} [L^{-1}b]_{1..r})$$

Since the determinant of R is non-zero modulo p there exist a $r \times r$ minor R_1 of R', which is non-zero modulo p. This means that we can find elements $R_{1i_1} \ldots R_{ri_r}$ of R_1, where i_j are pairwise distinct, $j = 1 \ldots r$, such that R_{ji_j} are non-zero modulo p. Let $d =_{p,s} L^{-1}b$. The probability can further be rewritten:

$$\begin{aligned}
&\mathcal{P}(x \in \mathcal{S}^n : R'x =_{p,s} [d]_{1..r}) = \\
&\sum_{y_1,\ldots,y_r \in \mathbb{Z}_{p^s}} \mathcal{P}\left(x_i \in \mathcal{S},\ \begin{matrix} i = 1..n, \\ i \neq i_j, \\ j = 1..r \end{matrix} : \begin{matrix} R_{11}x_1 + \ldots \hat{i_1} + R_{1n}x_n =_{p,s} y_1 \\ R_{21}x_1 + \ldots \hat{i_2} + R_{2n}x_n =_{p,s} y_2 \\ \ldots \\ R_{r1}x_1 + \ldots \hat{i_r} + R_{rn}x_n =_{p,s} y_r \end{matrix} \right) \\
&\cdot \mathcal{P}\left(x_{i_1}..x_{i_r} \in \mathcal{S} : \begin{matrix} x_{i_1} =_{p,s} (d_1 - y_1)R_{1i_1}^{-1} \\ x_{i_2} =_{p,s} (d_2 - y_2)R_{2i_2}^{-1} \\ \ldots \\ x_{i_r} =_{p,s} (d_r - y_r)R_{ri_r}^{-1} \end{matrix} \right).
\end{aligned} \tag{5.40}$$

We use $\ldots^{\hat{j}}$ to denote that the element with index j is omitted in the sum, $j = 1 \ldots r$.

As the entries x_{i_j} are chosen independently of each other we have

$$\mathcal{P}\left(x_{i_1}..x_{i_r} \in \mathcal{S} : \begin{array}{c} x_{i_1} =_{p,s} (d_1 - y_1)R^{-1}_{1i_1} \\ x_{i_2} =_{p,s} (d_2 - y_2)R^{-1}_{2i_2} \\ \ldots \\ x_{i_r} =_{p,s} (d_r - y_r)R^{-1}_{ri_r} \end{array} \right) = \prod_{j=1}^{r} \mathcal{P}_{\mathcal{S}}(x_{i_j} \in \mathcal{S} : x_{i_j} =_{p,s} (d_j - y_j)R^{-1}_{ji_j})$$

Moreover, by Prop. 5.1.5, uniform distribution $\mathcal{P}_{\mathcal{S}}$ on $\mathcal{S}$ induces almost uniform distribution mod p^s on $\mathbb{Z}_{p^s}$. Therefore, notice that $\alpha_{p^s} = \alpha_{p,s}$ and $\beta_{p^s} = \beta_{p,s}$, we obtain that

$$\alpha_{p,s} \leq \mathcal{P}_{\mathcal{S}}(x_j : x_j =_{p,s} (d_j - y_j)R^{-1}_{ji_j} \mod p^s) \leq \beta_{p,s}$$

for $j = 1 \ldots r$. As

$$\sum_{y_1,\ldots,y_r \in \mathbb{Z}_{p^s}} \mathcal{P}\left(x_i \in \mathcal{S},\ \begin{array}{c} i = 1..n, \\ i \neq i_j, \\ j = 1..r \end{array} : \begin{array}{c} R_{11}x_1 + \ldots^{\hat{i_1}} + R_{1n}x_n =_{p,s} y_1 \mod p^s \\ R_{21}x_1 + \ldots^{\hat{i_2}} + R_{2n}x_n =_{p,s} y_2 \mod p^s \\ \ldots \\ R_{r1}x_1 + \ldots^{\hat{i_r}} + R_{rn}x_n =_{p,s} y_r \mod p^s \end{array} \right) = 1,$$

we can bound the probability by

$$\alpha^r_{p,s} \leq \mathcal{P}(x : Ax =_{p,s} b \mod p^s) \leq \beta^r_{p,s}.$$

which finishes the proof.

Remark 5.4.2 The distribution on $\mathbb{Z}_{p^s}$ induced by the dot product $t \cdot x$ is not the same as $\mathcal{P}_{\mathcal{S},p^s}$. Yet, it has the same probability bounds $\alpha_{p,s}, \beta_{p,s}$.

The following lemmas show that the entries of a preconditioned matrix LXR, where entries of X are chosen from $\mathcal{S}$, are almost uniformly distributed.

Lemma 5.4.3 (Distribution of LXR) *Let $\mathcal{S}$ be a set of λ contiguous integers $\mathcal{S} = \{a, a+1, \ldots, a+\lambda-1\}$, where $a \in \mathbb{Z}$ and $\lambda > 0$. Let p be a prime, $s \in \mathbb{N}, s > 0$. Let $\mathcal{P}_{\mathcal{S}}$ denote the uniform distribution on $\mathcal{S}$. Let $\mathcal{P}$ denote the distribution of matrices of $\mathcal{S}^{n\times n}$ such that the entries of the matrix are chosen independently of each other according to distribution $\mathcal{P}_{\mathcal{S}}$. Let*

$$\alpha_{p,s} = \frac{1}{\lambda}\lfloor\frac{\lambda}{p^s}\rfloor, \quad \beta_{p,s} = \frac{1}{\lambda}\lceil\frac{\lambda}{p^s}\rceil.$$

Let $L, R \in \mathbb{Z}^{n\times n}$ be matrices such that $p \nmid \det(L), p \nmid \det(R)$. Let $\mathcal{I}$, $\mathcal{J}$ be any disjoint subsets of $\{1 \ldots n\}^2$ (sets of index pairs). Let $d_{ij}, (i,j) \in \mathcal{I}$ (resp. $D_{kl}, (k,l) \in \mathcal{J}$) be any values (resp. subsets) from $\mathbb{Z}_{p^s}$. We consider the probability of choosing a matrix

$X \in \mathcal{S}^{n\times n}$, according to distribution $\mathcal{P}$, such that $(LXR \mod p^s)_{ij} = d_{ij}$ for $(i,j) \in \mathcal{I}$ subject to event $(LXR \mod p^s)_{kl} \in D_{kl}$ for $(k,l) \in \mathcal{J}$. We have

$$\alpha_{p,s}^{|\mathcal{I}|} \leq \mathcal{P}\left((LXR \mod p^s)_{ij} = d_{ij} \mid (LXR \mod p^s)_{kl} \in D_{kl}\right) \leq \beta_{p,s}^{|\mathcal{I}|}.$$

PROOF By the definition of conditional probability we have

$$\begin{aligned}\mathcal{P}((LXR)_{ij} =_{p,s} d_{ij} \mid (LXR \mod p^s)_{kl} \in D_{kl}) \\ = \frac{\mathcal{P}((LXR)_{ij} =_{p,s} d_{ij} \wedge (LXR \mod p^s)_{kl} \in D_{kl})}{\mathcal{P}((LXR \mod p^s)_{kl} \in D_{kl})}\end{aligned} \tag{5.41}$$

Let $\mathcal{D}'$ denote the set of all possible matrices $[a_{kl}] \in \mathcal{S}^{n\times n}$ such that $(a_{kl} \mod p^s) \in D_{kl}$ if $(k,l) \in \mathcal{J}$. Let $\mathcal{D}$ denote the set of matrices from $\mathcal{D}'$ for which additionally $a_{kl} =_{p,s} d_{kl}$ if $(k,l) \in \mathcal{I}$. Let L^{-1} and R^{-1} denote the inverse mod p^s of L and R respectively. Then Eq. (5.41) can be transformed to

$$\frac{\mathcal{P}\left(X : (X \mod p^s) \in L^{-1}\mathcal{D}R^{-1}\right)}{\mathcal{P}\left(X : (X \mod p^s) \in L^{-1}\mathcal{D}'R^{-1}\right)},$$

where $L^{-1}\mathcal{D}R^{-1}$ (resp. $L^{-1}\mathcal{D}'R^{-1}$) is a subset of $\mathbb{Z}_{p^s}^{n\times n}$ equal to $\{L^{-1}MR^{-1} \mid M \in \mathcal{D}\}$ (resp. $\{L^{-1}MR^{-1} \mid M \in \mathcal{D}'\}$). Notice, that as L, R are invertible, sets $\mathcal{D}$ and $L^{-1}\mathcal{D}R^{-1}$ (resp. $\mathcal{D}'$ and $L^{-1}\mathcal{D}'R^{-1}$) have the same number of elements.

As every choice of X has the same probability, by the classic definition of probability it suffices to count the number of elements in $\mathcal{D}$ and $\mathcal{D}'$. The proportion is determined by the number of possible choices of elements at the $(k,l) \in \mathcal{I}$ position. The entries at the (k,l)th position, $(k,l) \in \mathcal{I}$, of $\mathcal{D}$ are determined modulo p^s and thus there are at least $\lfloor \frac{\lambda}{p^s} \rfloor$ and at most $\lceil \frac{\lambda}{p^s} \rceil$ possible choices for each entry a_{kl}, $(k,l) \in \mathcal{I}$. At the same time, $\mathcal{I} \cap \mathcal{J} = \emptyset$, therefore there are λ possibilities of choice for the corresponding entry for elements of $\mathcal{D}'$. Therefore the proportion of matrices is

$$\alpha_{p,s}^{|\mathcal{I}|} = \left(\frac{\lfloor \frac{\lambda}{p^s} \rfloor}{\lambda}\right)^{|\mathcal{I}|} \leq \frac{\mathcal{P}\left(X : (X \mod p^s) \in L^{-1}\mathcal{D}R^{-1}\right)}{\mathcal{P}\left(X : (X \mod p^s) \in L^{-1}\mathcal{D}'R^{-1}\right)} \leq \left(\frac{\lceil \frac{\lambda}{p^s} \rceil}{\lambda}\right)^{|\mathcal{I}|} = \beta_{p,s}^{|\mathcal{I}|}.$$

which gives the required inequality.

The methods used to prove Lemmas 5.4.1 and 5.4.3 can applied to prove the following lemma.

Lemma 5.4.4 (Distribution of AXA') *Let $\mathcal{S}$ be a set of λ contiguous integers $\mathcal{S} = \{a, a+1, \ldots, a+\lambda-1\}$, where $a \in \mathbb{Z}$ and $\lambda > 0$. Let p be a prime, $s \in \mathbb{N}, s > 0$. Let $\mathcal{P}_{\mathcal{S}}$ denote the uniform distribution on $\mathcal{S}$. Let $\mathcal{P}$ denote the distribution of matrices of $\mathcal{S}^{n\times m}$*

such that the entries of the matrix are chosen independently of each other according to distribution $\mathcal{P}_{\mathcal{S}}$. Let

$$\alpha_{p,s} = \frac{1}{\lambda}\lfloor\frac{\lambda}{p^s}\rfloor, \quad \beta_{p,s} = \frac{1}{\lambda}\lceil\frac{\lambda}{p^s}\rceil.$$

Let $n, m \in \mathbb{N}$, $0 < m \leq n$. Let $A \in \mathbb{Z}^{m\times n}$ be a matrix such that $\operatorname{rank}(A) = m$ *and the local Smith form of A at p is trivial i.e. $SF_p(A) = \operatorname{diag}(1, \ldots, 1)$. Let $A' \in \mathbb{Z}^{n\times n}$ be a matrix such that $p \nmid \det(A')$.*

Let $\mathcal{I}$, $\mathcal{J}$ be any disjoint subsets of $\{1 \ldots m\}^2$ (sets of index pairs). Let $b_{ij}, (i,j) \in \mathcal{I}$ ($B_{kl}, (k,l) \in \mathcal{J}$) be any values (resp. subsets) from Z_{p^s}. We consider the probability of choosing a random matrix $X \in \mathcal{S}^{n\times m}$ according to distribution $\mathcal{P}$, such that $(AXA')_{ij} = b_{ij}$ $\bmod p^s$ *for $(i,j) \in \mathcal{I}$ subject to event $(AXA' \mod p^s)_{kl} \in B_{kl}$ for $(k,l) \in \mathcal{J}$. We have*

$$\alpha_{p,s}^{|\mathcal{I}|} \leq \mathcal{P}\left((AXA')_{ij} = b_{ij} \mod p^s \mid (AXA' \mod p^s)_{kl} \in B_{kl}\right) \leq \beta_{p,s}^{|\mathcal{I}|}. \tag{5.42}$$

PROOF As in the proof of 5.4.1, A has a trivial Smith form modulo p and therefore there exist two matrices L and R in $\mathbb{Z}_{p^s}^{m\times m}$ and $\mathbb{Z}_{p^s}^{n\times n}$ respectively, s.t. $\det(L), \det(R) \neq_p 0$ and

$$A =_{p,s} L \begin{bmatrix} I_m & 0 \end{bmatrix} \begin{bmatrix} R' \\ R'' \end{bmatrix},$$

where $R' = [R_{ij}]_{i=1..m, j=1..n}$ is a $m \times n$ upper minor of R and R'' is the $(n-m)\times n$ lower minor. Moreover, matrix A' is invertible modulo p^s.

We have that $AXA' =_{p,s} B \Leftrightarrow L \begin{bmatrix} I_m \\ 0 \end{bmatrix} RX =_{p,s} A'^{-1}B \Leftrightarrow LR'X =_{p,s} A'^{-1}B$.

By taking $A'^{-1}B$ instead of B, we may assume without loss of generality that $A' = I$.

As in the proof of 5.4.3, we construct the sets of matrices $\mathcal{B}$, $\mathcal{B}'$. Let $\mathcal{B}'$ denote the set of all possible matrices $D \in \mathcal{S}^{m\times m}$, $D = [d_{kl}]_{kl=1..n}$, such that $(d_{kl} \mod p^s) \in B_{kl}$ if $(k,l) \in \mathcal{J}$. Let $\mathcal{B}$ denote the set of matrices from $\mathcal{B}'$ for which additionally $d_{kl} =_{p,s} b_{kl}$ if $(k,l) \in \mathcal{I}$. Let L^{-1} denote the inverse mod p^s of L.

We have that $\mathcal{P}\left((AX)_{ij} =_{p,s} b_{ij} \mid (AX \mod p^s)_{kl} \in B_{kl}\right)$ is $\frac{\mathcal{P}(R'X \in L^{-1}\mathcal{B})}{\mathcal{P}(R'X \in L^{-1}\mathcal{B}')}$.

As the entries are chosen independently of each other, we may consider the choice of each column separately and obtain

$$\mathcal{P}\left((AX)_{ij} =_{p,s} b_{ij} \mid (AX \mod p^s)_{kl} \in B_{kl}\right) = \frac{\prod_{i=1\ldots m}\mathcal{P}(R'X_i \in L^{-1}\mathcal{B}_i)}{\prod_{i=1\ldots m}\mathcal{P}(R'X_i \in L^{-1}\mathcal{B}'_i)}, \tag{5.43}$$

where X_i denote the ith column of X and $\mathcal{B}_i (resp. \mathcal{B}'_i)$, the set of all possible ith columns for matrices from $\mathcal{B}$ (resp. $\mathcal{B}'$).

Let us consider the probability of choosing the ith column separately and let us set $x = X_i$,$b = L^{-1}\mathcal{B}_i$, and $b' = L^{-1}\mathcal{B}'_i$. This puts us in the situation of Lem. 5.4.1 (ii). For the

ith column, let us define sets $\mathcal{I}_i = \{j : (j,i) \in \mathcal{I}\}$ and $\mathcal{J}_i = \{j : (j,i) \in \mathcal{J}\}$. As $\mathcal{I} \cap \mathcal{J} = \emptyset$, sets $\mathcal{I}_i$ and $\mathcal{J}_i$ are also disjoint for $i = 1..m$. Without loss of generality we may assume that there exists $0 \le k_i \le l_i \le m$, such that $\mathcal{I}_i = \{1, \dots, k_i\}$ and $\mathcal{J}_i = \{k_i + 1, \dots, l_i\}$. Moreover, we have $\sum_{i=1}^m k_i = |\mathcal{I}|$. In the case of the ith column let us set $k = k_i$ and $l = l_i$.

As before, since the determinant of R is non-zero modulo p there exist a $m \times m$ minor R_1 of R', which is non-zero modulo p. This means that we can find elements $R_{1i_1} \dots R_{mi_m}$ of R_1, where i_j are pairwise distinct, $j = 1 \dots m$, such that R_{ji_j} are non-zero modulo p.

We obtain

$$\begin{aligned}
\mathcal{P}(x \in \mathcal{S}^n : (R'x \mod p^s) \in b) = &\sum_{y_1,\dots,y_l \in \mathbb{Z}_{p^s}} \\
&\mathcal{P}\left(x_i \in \mathcal{S},\ \begin{matrix} i = 1..n, \\ i \neq i_j, \\ j = 1..m \end{matrix} : \begin{matrix} R_{11}x_1 + \dots^{\hat{i_1}} + R_{1n}x_n =_{p,s} y_1 \\ R_{21}x_1 + \dots^{\hat{i_2}} + R_{2n}x_n =_{p,s} y_2 \\ \dots \\ R_{l1}x_1 + \dots^{\hat{i_l}} + R_{ln}x_n =_{p,s} y_l \end{matrix} \right) \\
&\cdot \mathcal{P}\left(x_{i_1}, \dots, x_{i_m} \in \mathcal{S} : \begin{matrix} (x_{i_1} \mod p^s) = (b_1 - y_1)R_{1i_1}^{-1} \\ \dots \\ (x_{i_k} \mod p^s) = (b_k - y_k)R_{ki_k}^{-1} \\ (x_{i_{k+1}} \mod p^s) \in (b_{k+1} - y_{k+1})R_{k+1,i_{k+1}}^{-1} \\ \dots \\ (x_{i_l} \mod p^s) \in (b_l - y_l)R_{li_l}^{-1} \end{matrix} \right).
\end{aligned} \tag{5.44}$$

We use $\dots^{\hat{j}}$ to denote that the element with index j is omitted in the sum, $j = 1..m$. As the entries $x_{i_1}, \dots x_{i_m}$ are chosen independently of each other, the probability is

$$\begin{aligned}
\mathcal{P}(x \in \mathcal{S}^n :&(R'x \mod p^s) \in b) = \sum_{y_1,\dots,y_l \in \mathbb{Z}_{p^s}} \\
&\mathcal{P}\left(x_i \in \mathcal{S},\ \begin{matrix} i = 1..n, \\ i \neq i_j, \\ j = 1..m \end{matrix} : \begin{matrix} R_{11}x_1 + \dots^{\hat{i_1}} + R_{1n}x_n =_{p,s} y_1 \\ R_{21}x_1 + \dots^{\hat{i_2}} + R_{2n}x_n =_{p,s} y_2 \\ \dots \\ R_{l1}x_1 + \dots^{\hat{i_l}} + R_{ln}x_n =_{p,s} y_l \end{matrix} \right) \\
&\cdot \prod_{j=1}^{k} \mathcal{P}_{\mathcal{S}} \left(x_{i_j} \in \mathcal{S} : (x_{i_j} \mod p^s) = (b_j - y_j)R_{ji_j}^{-1} \right) \\
&\cdot \prod_{j=k+1}^{l} \mathcal{P}_{\mathcal{S}}((x_{i_j} \mod p^s) \in (b_j - y_j)R_{ji_j}^{-1}).
\end{aligned}$$

Moreover, by Prop. 5.1.5, uniform distribution $\mathcal{P}_{\mathcal{S}}$ on $\mathcal{S}$ induces almost uniform distribution mod p^s on $\mathbb{Z}_{p^s}$. Therefore, for $j = 1 \dots k$, we obtain

$$\alpha_{p,s} \le \mathcal{P}_{\mathcal{S}}(x_j : x_j =_{p,s} (b_j - y_j)R_{ji_j}^{-1} \mod p^s) \le \beta_{p,s}. \tag{5.45}$$

By Eq. (5.45)

$$\alpha_{p,s}^k \leq \sum_{y_1,\ldots y_k \in \mathbb{Z}_{p^s}} \mathcal{P}\left(x_i \in \mathcal{S},\ \begin{matrix} i = 1..n, \\ i \neq i_j, \\ j = 1..m \end{matrix} : \begin{matrix} R_{11}x_1 + \ldots \hat{i_1} + R_{1n}x_n =_{p,s} y_1 \\ R_{21}x_1 + \ldots \hat{i_2} + R_{2n}x_n =_{p,s} y_2 \\ \ldots \\ R_{k1}x_1 + \ldots \hat{i_k} + R_{kn}x_n =_{p,s} y_k \end{matrix} \right)$$
$$\underbrace{\cdot \prod_{j=1}^{k} \mathcal{P}_{\mathcal{S}}\left(x_{i_j} \in \mathcal{S} : (x_{i_j} \mod p^s) = (b_j - y_j) R_{ji_j}^{-1} \right)}_{\mathcal{P}_{rob}} \leq \beta_{p,s}^k.$$

Let us denote the above expression by $\mathcal{P}_{rob}$. We have

$$\alpha_{p,s}^k \mathcal{P}_{rob} \leq \mathcal{P}(x \in \mathcal{S}^n : (R'x \mod p^s) \in b) \leq \beta_{p,s}^k \mathcal{P}_{rob}\,.$$

As $b_j = b'_j$ for $j = k+1..l$, $\mathcal{P}_{rob}$ is equal to $\mathcal{P}(x \in \mathcal{S}^n : (R'x \mod p^s) \in b') = \mathcal{P}_{rob}$. as well.

Therefore for the ith column ($k = k_i$) we have the final evaluation

$$\alpha_{p,s}^{k_i} \leq \frac{\mathcal{P}(R'X_i \in L^{-1}\mathcal{B}_i)}{\mathcal{P}(R'X_i \in L^{-1}\mathcal{B}'_i)} \leq \beta_{p,s}^{k_i}.$$

Summarizing, by Eq. (5.43) and by the fact that $\sum_{i=1}^n k_i = |\mathcal{I}|$ we have

$$\alpha_{p,s}^{|\mathcal{I}|} \leq \mathcal{P}\left((AX)_{ij} =_{p,s} b_{ij} \mid (AX \mod p^s)_{kl} \in B_{kl} \right) \leq \beta_{p,s}^{|\mathcal{I}|},$$

which finishes the proof.

5.4.2 The Determinant of a Random Matrix

Theorem 5.4.5 (Determinant Modulo p^s) *Let $\mathcal{S}$ be a set of λ contiguous integers $\mathcal{S} = \{a, a+1, \ldots, a+\lambda-1\}$, where $a \in \mathbb{Z}$ and $\lambda > 0$. Let p be a prime, $s \in \mathbb{N}, s > 0$. Let $\mathcal{P}_{\mathcal{S}}$ denote the uniform distribution on $\mathcal{S}$. Let $\mathcal{P}$ denote the distribution of matrices of $\mathcal{S}^{n\times k}$ such that the entries of the matrix are chosen independently of each other according to distribution $\mathcal{P}_{\mathcal{S}}$.*

Let $k, n \in \mathbb{N}, 0 < k \leq n$. Let U be an $k \times n$ matrix, $k \leq n$, such that the Smith form of U is trivial and U has a full rank i.e. $SF(U) = \mathrm{diag}(1, \ldots, 1)$. Let V be a $n \times n$ matrix, such that $p \nmid \det(V)$. Let M be an $n \times k$ matrix chosen according to distribution $\mathcal{P}$. Then the probability that $p^s < \lambda$ divides the determinant $\det(UMV)$ is at most $\frac{3}{p^s}$.

Proof of Thm. 5.4.5

The proof is divided into several parts. First we present the LRE elimination mod p^s algorithm, see [38, 39], which we will use in our reasoning. Then we consider the distribution of entries at each step of the algorithm. Last part is consecrated to the inductive proof which is done with respect to matrix size k and prime power s.

The fact that p^s divides the determinant $\det(UMV)$ means that $\mathrm{ord}_p(\det(UMV))$, the order mod p, is greater than or equal to s. We may evaluate $\mathrm{ord}_p(\det(UMV))$ by the LRE algorithm of [38, 39]. The Algorithm, adapted to the order modulo p evaluation, is given in Alg. 5.4.1.

1. *LRE Algorithm*

   To check whether $\mathrm{ord}_p\left(\det\left(UMV\right)\right) \geq s$ can run Alg. 5.4.1 with input $M(0) = UMV$, p and s. The algorithm consists of elimination and reduction steps. All operations are performed modulo p^s. At the rth elimination step, the next invertible entry, if it exists, is placed in the (r, r) pivot position and the rth column and row are zeroed. Elimination is done by the elementary row and column operations, which are encoded in L and R matrices. Matrices L, R are invertible modulo p^s and are determined by the entries of $M(0)$. Surely, the original values of $M(0)$ can be reconstructed from $M(i) = LM(0)R$ and L, R. If no invertible entry is found, it means that all remaining entries are divisible by p, so we proceed with a reduction step i.e. we consider the remaining $(k-r, k-r)$ submatrix divided by p. The problem now reduces to determining whether ord_p of an $(k-r, k-r)$ matrix is greater than $s-k+r$.

   Proposition 5.4.6 (Correctness of Alg. 5.4.1) Let $k \in \mathbb{N}$ and let M be a $k \times k$ matrix of integers. Let p be a prime and $s \in \mathbb{N}, s > 0$. Algorithm 5.4.1 stops after at most $\lceil \frac{s}{2} \rceil$ steps of the **while** loop. It returns TRUE if and only if $\mathrm{ord}_p(\det(M)) \geq s$.

   PROOF The algorithm stops if $s_{i+1} \leq 0$ or if the remaining submatrix is of size 0 or 1. Notice, that at each step of the **while** loop, except the last one, s_i is decremented by at least $2 \leq k_{i-1} - r_i$. Therefore after $\lceil \frac{s_1}{2} \rceil$ steps $s_{i+1} \leq s_1 - 2\lceil \frac{s_1}{2} \rceil \leq 0$ and the algorithm stops.

   To proof the correctness of the algorithm notice, that in the ith step we have a recursive formula

   $$\det(M(i-1)) = p^{k_{i-1}-r_i} \det(M_i'). \tag{5.46}$$

   Therefore $\mathrm{ord}_p(\det(M(i-1))) \geq s_i$ iff $\mathrm{ord}_p(\det(M_i')) \geq s_{i+1}$. Let us analyze the stopping conditions:

   1. $s_i > 0$, $k_{i-1} - r_i = 0$; M_i' is an empty matrix, $\mathrm{ord}_p(\det(M(i-1))) = 0 < s_i$. The algorithm returns FALSE are required.
   2. $s_i > 0$, $k_{i-1} - r_i = 1$; $M_i' = (L_i M(i-1) R_i)_{k_{i-1}k_{i-1}}$ is the last entry of the matrix. $\mathrm{ord}_p(\det(M(i-1))) = \mathrm{ord}_p(M_i')$. The algorithm returns TRUE and FALSE correctly, depending on $\mathrm{ord}_p(M_i')$.

Algorithm 5.4.1 LRE Algorithm of [39] for $\mathrm{ord}_p(\det(A))$

Require: $k \times k$ matrix M;
Require: prime p;
Require: integer $s > 0$;
Ensure: TRUE if $\mathrm{ord}_p(\det(M)) \geq s$, FALSE otherwise.

$M(0) = M \mod p^s$; $k_0 = k$; $s_1 = s$;$i = 1$;
while $s_i > 0$ **do**
  Find $r_i \geq 0$;$L_i, R_i, \in \mathbb{Z}^{k_{i-1} \times k_{i-1}}$; $M_i' \in \mathbb{Z}^{(k_{i-1}-r_i)\times(k_{i-1}-r_i)}$,$p \mid M_i'$ such that

$$L_i M(i-1) R_i = \mathrm{diag}(\underbrace{1, \ldots, 1}_{r_i}, M_i') \mod p^{s_i} \qquad \text{\#Elimination step;}$$

  $k_i = k_{i-1} - r_i$;
  if $k_i = 0$ **then** **Return** FALSE;
  else if $k_i = 1$ **then**
    if $p^{s_i} \mid (M_i')_{k_i k_i}$ **then** **Return** TRUE;
    else **Return** FALSE;
  else
    $s_{i+1} = s_i - k_i$;
    if $s_{i+1} \leq 0$ **then** **Return** TRUE;
    $M(i) = L_i M(i-1) R_i[r_i + 1..k_{i-1}, r_i + 1..k_{i-1}]/p$; #Reduction step
    $i = i + 1$;
end while

3. $s_i > 0$, $k_{i-1} - r_i > 1$; either $s_{i+1} < 0 \leq \mathrm{ord}_p(\det(M_i'))$ and the algorithm returns TRUE, or we recursively run the algorithm for $s_{i+1} = s_i - k_{i-1} + r_i$ and $M(i)$ to determine whether $\mathrm{ord}_p(\det(M(i))) \geq s_{i+1}$. By the recursive formula (5.46) the final answer is correct.

Let us now recover the relation between $M(0)$, M_i' and $M(i)$ from the recursive formulas. First, we have

$$L_i M(i-1) R_i = \begin{bmatrix} I_{r_i} & 0 \\ 0 & pM(i) \end{bmatrix} \mod p^{s_i},$$

which results in the following matrix equation for $M(0)$.

$$\begin{bmatrix} I_{\sum_{t=1}^{i-1} r_t} & 0 \\ 0 & L_i \end{bmatrix} \cdots \begin{bmatrix} I_{r_1} & 0 \\ 0 & L_2 \end{bmatrix} L_1 M(0) R_1 \begin{bmatrix} I_{r_1} & 0 \\ 0 & R_2 \end{bmatrix} \cdots \begin{bmatrix} I_{\sum_{t=1}^{i-1} r_t} & 0 \\ 0 & R_i \end{bmatrix}$$
$$= \begin{bmatrix} I_{r_1} & & & & \\ & pI_{r_2} & & & \\ & & \ddots & & \\ & & & p^{i-1} I_{r_i} & \\ & & & & M_i' \end{bmatrix} \mod p^s \tag{5.47}$$

For the ith elimination step to be completed we additionally require that $M'_i = p^i M(i)$.

2. *Distribution of entries at each step of the* **while** *loop*

To determine the probability $\mathrm{ord}_p(\det(UMV) \geq s$ we need to run Alg. 5.4.1 for a random input M. As every matrix UMV corresponds to matrices L_1, R_1 and vice versa, we will use the law of total probability to sum over all possible choices of L_1, R_1 and then determine the possibility of each **if** branch. This requires a recursive call to the algorithm to determine whether $\mathrm{ord}_p(\det(M(1))) \geq s_2$, or, inductively, whether $\mathrm{ord}_p(\det(M(i))) \geq s_{i+1}$, $i = 1, \ldots i_{max}$, subject to the event that $M(i)$ is obtained. Here, i_{max} denote the total number of steps of the **while** loop. We need to prove that the theorem can be applied recursively to matrix $M(i)$, whose entries are distributed according to the conditional probability, subject to the event that $M(i)$ is obtained.

The detailed scheme of the inductive procedure is given in step 3. In the following lemma we characterize the conditional distribution of entries of $M(i)$.

Lemma 5.4.7 (Distribution of $M(i)$ in Alg. 5.4.1) *Let $\mathcal{S}$ be a set of λ contiguous integers $\mathcal{S} = \{a, a+1, \ldots, a+\lambda-1\}$, where $a \in \mathbb{Z}$ and $\lambda > 0$. Let p be a prime, $s \in \mathbb{N}, s > 0$. Let $\mathcal{P}_{\mathcal{S}}$ denote the uniform distribution on $\mathcal{S}$. Let $\mathcal{P}$ denote the distribution of matrices of $\mathcal{S}^{n\times k}$ such that the matrix entries of are chosen independently of each other according to distribution $\mathcal{P}_{\mathcal{S}}$.*

Let $k, n \in \mathbb{N}, 0 < k \leq n$. Let U be an $k \times n$ matrix, $k \leq n$, such that the Smith form of U is trivial and U has a full rank i.e. $SF(U) = \mathrm{diag}(1, \ldots, 1)$. Let V be a $n \times n$ matrix, such that $p \nmid \det(V)$.

Let M be an random element of $\mathcal{S}^{n\times k}$. Suppose that $M(0) = UMV$ is given as an input to Alg. 5.4.1. Let i_{max} be the number of steps of the **while** *loop in the algorithm and let $0 < i \leq i_{max}$ be one of the steps. Let $M(i-1)$ be the matrix on which elimination is performed in the ith step.*

Let $\mathcal{P}^{i-1}$ denote the conditional probability distribution under the assumption that matrix $M(i-1)$ was obtained in the $i-1$th step of the algorithm and $\mathcal{P}^{i-1}_{L,R}$ denote the conditional probability that matrix $M(i-1)$ was obtained and, additionally, that it is possible to partially diagonalize the matrix $\mod p^{s_i}$ by performing $LM(i-1)R$ (which is the goal of the ith step). Notice that $\mathcal{P}^{i-1} = \mathcal{P}^{i-1}_{I,I}$. Let k_{i-1} be the dimension of $M(i-1)$, r be the number of 1 at the beginning of the diagonal of $LM(i-1)R$ and let $k_i = k_{i-1} - r$. Let $\alpha \leq s_i$, $\mathcal{I} \subset \{1, \ldots, k_i\}^2$ be a set of index pairs and $d_{tt'} \subset \mathbb{Z}_{p^\alpha}, (t, t') \in \mathcal{I}$ be a given subset of $\mathbb{Z}_{p^\alpha}$.

We have that

$$\mathcal{P}^{i-1}_{L,R}((LM(i-1)R)_{r+t,r+t'} = d_{tt'} \mod p^\alpha, (t,t') \in \mathcal{I}) \leq \begin{cases} \left(\frac{p^{s_i-1}+1}{p^{s_i}+1}\right)^{|\mathcal{I}|} & \alpha = 1 \\ \left(\frac{2}{p^\alpha}\right)^{|\mathcal{I}|} & \alpha \geq 1 \end{cases}.$$

PROOF First, let us suppose, that multiplication by L and R leads to the elimination of $r = r_i$ rows, $r \geq 0$. Then by Eq. (5.47) we can set

$$A = \begin{bmatrix} I_{\sum_{t=1}^{i-1} r_t} & 0 \\ 0 & L_i \end{bmatrix} \cdots \begin{bmatrix} I_{r_1} & 0 \\ 0 & L_2 \end{bmatrix} L_1 U$$
$$A' = V R_1 \begin{bmatrix} I_{r_1} & 0 \\ 0 & R_2 \end{bmatrix} \cdots \begin{bmatrix} I_{\sum_{t=1}^{i-1} r_t} & 0 \\ 0 & R_i \end{bmatrix}.$$

Then we can set

$$B = \begin{bmatrix} I_{r+\sum_{t=1}^{i-1} r_i} & 0 \\ 0 & L \end{bmatrix} A$$
$$B' = A' \begin{bmatrix} I_{r+\sum_{t=1}^{i-1} r_i} & 0 \\ 0 & R \end{bmatrix}.$$

Matrices B, B' fulfill the assumptions of Lemma 5.4.4 and $BMB' = B[*]B' \mod p^s$, where $[*]$ is the right-hand side of Eq. (5.47). $C = B[*]B' \mod p^s$ is of the form

$$\begin{bmatrix} I_{r1} & & & & \\ & pI_{r2} & & & \\ & & \ddots & & \\ & & & p^{i-1} I_r & \\ & & & & p^{i-1} M' \end{bmatrix}. \tag{5.48}$$

We should remark that at this point it is not yet decided whether $p \mid M'$ i.e. $L_i = L, R_i = R$ or whether the elimination continues.

Let us consider the distribution modulo $p^{i-1+\alpha}$. Let us set $\mathcal{J} = \{1..k\} \times \{1..k\}$ to be the condition set of all index pairs, and let $\mathcal{I}' = \{(t+r+\sum_{j=1}^{i-1} r_j, t'+r+\sum_{j=1}^{i-1} r_j), (t,t') \in \mathcal{I}\}$.

Let us define $\mathcal{C}_{tt'} \mod p^{i-1+\alpha}$:

$$\mathcal{C}_{tt'} = \begin{cases} \{0\}, & t \neq t', t \leq \sum_{j=0}^{i-1} r_j + r \text{ or } t' \leq \sum_{j=0}^{i-1} r_j + r \\ \{p^j\}, & \sum_{j=0}^{x} r_j < t = t' \leq \sum_{j=0}^{x+1} r_{j+1}, x = 0, \ldots i-1 \\ \{p^{i-1}, \ldots, p^{i-1}(p^\alpha - 1)\}, & t, t' > \sum_{j=1}^{i} r_j \end{cases}.$$

We want to compute the conditional probability $\mathcal{P}_{L,R}^{i-1}((LM(i-1)R)_{k_i+t,k_i+t'} = d_{tt'} \mod p^\alpha, (t,t') \in \mathcal{I})$. By Eq. (5.48) and using introduced notations this is equivalent to the computation of the probability

$$\mathcal{P}(C_{tt} = p^{i-1} d_{tt'}, (t,t') \in \mathcal{I}' \mid C_{xx'} \in \mathcal{C}_{xx'}, (x,x') \in \mathcal{J}). \tag{5.49}$$

However, we cannot use Lemma 5.4.4 directly to compute this probability as the condition set $\mathcal{J} = \{1..k\} \times \{1..k\}$ contains all index pairs and $\mathcal{I}' \subset \mathcal{J}$. Yet in this particular case, the idea of the proof carries on. First, we apply the definition of the conditional probability and obtain

$$\frac{\mathcal{P}(C_{tt'} = p^{i-1}d_{tt'}, (t,t') \in \mathcal{I}' \wedge C_{tt'} \in \mathcal{C}_{tt'}, (t,t') \in \mathcal{J} \setminus \mathcal{I}')}{\mathcal{P}(C_{tt'} \in \mathcal{C}_{tt'}, (t,t') \in \mathcal{J})}.$$

As $p^{i-1}d_{tt'} \in \mathcal{C}_{tt'}$ we only need to estimate which fraction of cases corresponding to $C_{tt'} \in B_{tt'}$ corresponds to $C_{tt'} = p^{i-1}d_{tt'}$, $(t,t') \in \mathcal{I}'$. We will analyze the case when $\mathcal{I}'$ consist of one element in detail. Then the result generalized to the case when the elements of $\mathcal{I}'$ lie in one column, and to the general $\mathcal{I}'$, see the arguments in the proof of Lem.5.4.4.

Suppose that $\mathcal{I}'$ has one element (t,t'). $C_{xx'} \in \mathcal{C}_{xx'}$ for $(x,x') \in \mathcal{J} \setminus \{(t,t')\}$ leave the choice of only one entry of M free. Condition $C_{tt'} = 0$ modulo p^{i-1} corresponds to a choice of S equally spaced elements from $\mathcal{S}$ (see Eq. (5.44)) for this element. At most $\lceil \frac{S}{p^\alpha} \rceil$ of those have a certain value $d_{tt'}$ modulo p^α. Therefore the fraction in Eq. (5.49) is bounded by

$$\frac{1}{S}\lceil \frac{S}{p^\alpha} \rceil \leq \frac{1}{p^\alpha} + \frac{p^\alpha - 1}{Sp^\alpha}. \tag{5.50}$$

We have that $S = \lfloor \frac{\lambda}{p^{i-1}} \rfloor$ or $S = \lceil \frac{\lambda}{p^{i-1}} \rceil$. Therefore for $i \leq \lceil \frac{s_1}{2} \rceil$ we have

$$p^\alpha \leq p^{s_i} \leq p^{s_1 - 2(i-1)} < \frac{\lambda}{p^{2(i-1)}} \leq \lfloor \frac{\lambda}{p^{i-1}} \rfloor \leq S.$$

This leads to a bound for $\frac{1}{S} \leq \frac{1}{p^\alpha + 1}$ and for $\alpha = 1$ we obtain a better bound $\frac{1}{S} \leq \frac{1}{p^{s_i}+1}$. By including this bound in Eq. (5.50) we obtain

$$\frac{1}{S}\lceil \frac{S}{p^\alpha} \rceil \leq \frac{2}{p^\alpha + 1}$$
$$\frac{1}{S}\lceil \frac{S}{p} \rceil \leq \frac{p^{s_i - 1} + 1}{p^{s_i} + 1}$$

3. *Inductive proof*

For the inductive scheme to work we will have to prove a generalized version of 5.4.5, where the almost uniform distribution $\mathcal{P}$ of the entries of M is replaced by a distribution $\mathcal{P}'$ such that

$$\mathcal{P}'(M_{tt'} = d_{tt'} \mod p^\alpha, (t,t') \in \mathcal{I}) \leq \beta_{p,\alpha}^{|\mathcal{I}|}.$$

for any index set $\mathcal{I}$, α, and a bound $\beta_{p,\alpha}$.

By Lem. 5.4.7 this is the case of the conditional distribution $\mathcal{P}^{i-1}_{L,R}$ of the entries of $M(i-1)$, $i = 1, \ldots, i_{max}$, for any admissible matrices L, R, $\alpha \leq s_i$ and

$$\beta_{p,\alpha} = \begin{cases} \frac{2}{p^{\alpha}+1} & \alpha \geq 1 \\ \frac{p^{s_i-1}+1}{p^{s_i}+1} & \alpha = 1 \end{cases}.$$

The following lemma suffices to prove Thm. 5.4.5.

Lemma 5.4.8 (Determinant mod p^s of $M(i-1)$) *Let UMV be such as in Thm. 5.4.5. Suppose that UMV is the input to Alg. 5.4.1. Let us consider the notions of Alg. 5.4.1 i.e. let $M(i-1)$ denote the $k_{i-1} \times k_{i-1}$ matrix in the ith step of the algorithm, for which the computation is performed modulo s_i. Let $\mathcal{P}^{i-1}$ be defined as in Lem. 5.4.7. Then*

$$\mathcal{P}^{i-1}(p^{s_i} \mid \det(M(i-1))) \leq \frac{3}{p^{s_i}}$$

for steps $i = 1, \ldots, i_{max} - 1$ of the **while** *loop.*

The proof is recursive. We will proof the theorem in several steps.

a) Initialization step $s_i = 1$, $i = i_{max}$.
In this step we analyze the situation when the modulus p^{s_i} is the smallest possible i.e. $s_i = 1$. This has to be the last step, therefore we may assume that $i = i_{max}$.
First, for $s_i = 1$, [127, Thm 5.13] gives

$$\mathcal{P}^{i-1}\left(p \nmid \det\left(M(i-1)\right)\right) \leq \prod_{t=1}^{k_{i-1}} \left(1 - \beta^t_{p,1}\right).$$

This transforms to

$$\mathcal{P}^{i-1}\left(p \mid \det\left(M(i-1)\right)\right) \leq \sum_{t=1}^{k_{i-1}} \beta^t_{p,1} \leq \frac{\beta_{p,1}}{1-\beta_{p,1}}. \tag{5.51}$$

Thus, the probability can be bounded by $\min\left(1, \frac{\frac{2}{p+1}}{1-\frac{2}{p+1}}\right)$ and therefore by $\frac{2}{p-1} \leq \frac{3}{p}$.

b) Initialization step $k_{i-1} = 2$, $i = 1, \ldots i_{max}$.
In this step we analyze the situation when the recursive call would be performed on the smallest possible matrix i.e. $k_{i-1} = 2$.

i. Inner initialization step $k_{i-1} = 2$, $s_i = 2$, $i = i_{max}$.
This has to be the last step, therefore we may assume that $i = i_{max}$. For $k_{i-1} > 1$ we will sum over all possible choices of L_i and R_i. We will divide the sum on the cases when applying L_i and R_i to $M(i-1)$ leads to the elimination of at least r rows/columns. We call such event E_r.

First, for $k_{i-1} = 2, s_i = 2$:

$$\mathcal{P}^{i-1}(p^2 \mid \det(M(i-1)) \leq \sum_{L,R\in E_1} \mathcal{P}^{i-1}_{L,R}(p^2|(LM(i-1)R)_{22})\mathcal{P}^{i-1}(L,R\in E_1)$$
$$+\mathcal{P}^{i-1}(p|M(i-1)_{tt'}, t,t' = 1,2) \leq \beta_{p,2} + \beta^4_{p,1} \leq \frac{2}{p^2+1} + (\frac{2}{p+1})^4 \leq \frac{3}{p^2}$$

ii. Inner Induction Step $k_{i-1} = 2$, $s_i > 2$.
Now we suppose inductively that $\mathcal{P}^j\left(p^\alpha \mid \det(M(j))\right) \leq \frac{3}{p^\alpha}$ for all $\alpha \leq s_j < s_i$, $j = i+1, \ldots, i_{max}$.
Then for $k_{i-1} = 2, 2 < s_i$,

$$\mathcal{P}^{i-1}\left(p^{s_i} \mid \det(M(i-1))\right) \leq \sum_{L,R\in E_1} \mathcal{P}^{i-1}_{L,R}\left(p^{s_i}|\left(LM(i-1)R\right)_{22}\right)\mathcal{P}^{i-1}\left(L,R\in E_1\right)$$
$$+\mathcal{P}^{i-1}\left(p|M(i-1)_{tt'}, t,t' = 1,2\right)\mathcal{P}^{i-1}_{I,I}\left(p^{s_i-2}|\det(M(i-1))\right)$$

As $p \mid M(i-1)_{tt'}$ for $t,t' = 1,2$, we go to step $i+1$. In this case we have $M(i) = M(i-1)$, $s_{i+1} = s_i - 2$ and by the recursive argument the probability

$$\mathcal{P}^{i-1}_{I,I}\left(p^{s_i-2}|\det(M(i-1))\right) = \mathcal{P}^i(p^{s_{i+1}}|\det(M(i)) \leq \frac{3}{p^{s_{i+1}}}.$$

We may recursively bound the probability by

$$\mathcal{P}^{i-1}\left(p^{s_i} \mid \det(M(i-1))\right) \leq \beta_{p,s} + \beta^4_{p,1}\frac{3}{p^{s_i-2}}.$$

The latter is less than $\frac{3}{p^{s_i}}$ when

$$\beta^4_{p,1}3p^2 \leq 1. \tag{5.52}$$

With $\beta_{p,1} \leq \frac{2}{p+1}$ this means that $\frac{48}{(1+1/p)^2(p+1)^2} = \frac{48}{(p+2+1/p)^2} \leq 1$ which is fulfilled for $p > 3$. For primes $p = 2,3$ we have to use a sharper bound for $\beta_{p,1} \leq \frac{p^{s_i-1}+1}{p_i^s+1} \leq \frac{p+1}{p^2+1}$.
This allows us to prove the inequality (5.52) for $p = 3$ since $\left(\frac{2}{5}\right)^4 27 < 0.7$. For $p = 2$ and $s_i > 3$ also $\left(\frac{9}{17}\right)^4 12 < 0.95$. For the remaining case $p = 2$, $s_i = 3$ we can bound $\mathcal{P}^{i-1}_{I,I}\left(p \mid \det\left(M\left(i\right)\right)\right)$ by 1 and then one can prove that $\beta_{p,3} + \beta^4_{p,1} \leq \frac{2}{9} + \left(\frac{5}{9}\right)^4 < 0.32 \leq \frac{3}{8}$.

c) Induction step $k_{i-1} > 2$
Again, let us suppose inductively that $\mathcal{P}^j\left(p^\alpha \mid \det(M(j))\right) \leq \frac{3}{p^\alpha}$ for all $\alpha \leq s_j < s_i$, $j = i+1, \ldots, i_{max}$. Notice that, for $j > i$ we have that $k_{j-1} \leq k_{i-1}$ and $s_j < s_i$.

We will consider $k_{i-1} > 2$. Again we can sum over all possible elimination and reduction steps combinations. The resulting probability is, for $s_i \leq k_{i-1}$:

$$\begin{aligned}
&\mathcal{P}^{i-1}\left(p^{s_i} \mid \det\left(M\left(i-1\right)\right)\right) \leq \mathcal{P}^{i-1}\left(p|M\left(i-1\right)_{tt'} \forall_{t,t' \leq k_{i-1}}\right) \\
&+ \sum_{r=1}^{k_{i-1}-s_i} \sum_{L,R \in E_r} \mathcal{P}^{i-1}_{L,R}\left(p|\left(LM\left(i-1\right)R\right)_{tt'} \forall_{r<t,t' \leq k_{i-1}}\right) \mathcal{P}^{i-1}\left(L,R \in E_r\right) \\
&+ \sum_{r=k_{i-1}-s_i+1}^{k_{i-1}-2} \sum_{L,R \in E_r} \mathcal{P}^{i-1}_{L,R}\left(p|\left(LM\left(i-1\right)R\right)_{tt'} \forall_{r<t,t' \leq k_{i-1}}\right) \cdot \\
&\qquad\qquad \mathcal{P}^{i-1}\left(L,R \in E_r\right) \mathcal{P}^{i-1}_{L,R}\left(p^{s_i-k_{i-1}+r}|\det\left(LM(i-1)R\right)\right) \\
&+ \sum_{L,R \in E_{k_{i-1}-1}} \mathcal{P}^{i-1}_{L,R}\left(p^{s_i}|\left(LM\left(i-1\right)R\right)_{k_{i-1}k_{i-1}}\right) \mathcal{P}^{i-1}\left(L,R \in E_{k_{i-1}-1}\right) \leq \\
&\sum_{r=0}^{k_{i-1}-s_i} \beta_{p,1}^{(k_{i-1}-r)^2} + \sum_{r=k_{i-1}-s_i+1}^{k_{i-1}-2} \sum_{L,R \in E_r} \beta_{p,1}^{(k_{i-1}-r)^2} \mathcal{P}^{i-1}\left(L,R \in E_r\right) \\
&\qquad \cdot \mathcal{P}^{i-1}\left(p^{s_i-k_{i-1}+r}|\det\left(LM(i-1)R\right)\right) + \beta_{p,s_i},
\end{aligned}$$

and similarly for $s_i > k_{i-1}$

$$\begin{aligned}
&\mathcal{P}^{i-1}\left(p^{s_i} \mid \det(M(i-1))\right) \leq \\
&\mathcal{P}^{i-1}\left(p|M(i-1)_{tt'} \forall_{t,t' \leq k}\right) \mathcal{P}^{i-1}_{I,I}\left(p^{s_i-k_{i-1}}|\det\left(LM(i-1)R\right)\right) \\
&+ \sum_{r=1}^{k_{i-1}-2} \sum_{L,R \in E_r} \mathcal{P}^{i-1}_{L,R}\left(p|\left(LM\left(i-1\right)R\right)_{tt'} \forall_{r<t,t' \leq k_{i-1}}\right) \cdot \\
&\qquad\qquad \mathcal{P}^{i-1}\left(L,R \in E_r\right) \mathcal{P}^{i-1}_{L,R}\left(p^{s_i-k_{i-1}+r}|\det\left(LM(i-1)R\right)\right) \\
&+ \sum_{L,R \in E_{k_{i-1}-1}} \mathcal{P}^{i-1}_{L,R}\left(p^{s_i}|\left(LM\left(i-1\right)R\right)_{k_{i-1}k_{i-1}}\right) \mathcal{P}^{i-1}\left(L,R \in E_{k_{i-1}-1}\right) \\
&\leq \beta_{p,1}^{k_{i-1}^2} \mathcal{P}^{i-1}\left(p^{s_i-k_{i-1}}|\det\left(M\left(i-1\right)\right)\right) + \beta_{p,s_i} \\
&+ \sum_{r=1}^{k_{i-1}-2} \sum_{L,R \in E_r} \beta_{p,1}^{r^2} \mathcal{P}^{i-1}\left(L,R \in E_r\right) \mathcal{P}^{i-1}_{L,R}\left(p^{s_i-k_{i-1}+r}|\det\left(LM(i-1)R\right)\right).
\end{aligned}$$

For every $r = 0, \ldots, k_{i-1}-2$ for every choice of matrices L, R, the fact that p divides $(LM(i-1)R)_{tt'}$, for $r < t, t' \leq k_{i-1}$ means that either $s_i - k_{i-1} + r \leq 0$ and the algorithm stops or we proceed to the $i+1$ step of the algorithm. In the latter case we have $M(i) = LM(i-1)R$, $s_{i+1} = s_i - k_{i-1} + r$. By the recursive argument the probability

$$\mathcal{P}^{i-1}_{L,R}\left(p^{s_i-2}|\det(LM(i-1)R)\right) = \mathcal{P}^{i}(p^{s_{i+1}}|\det(M(i))$$

is less than $\frac{3}{p^{s_i+1}}$. Therefore we can then bound both sums by

$$\mathcal{P}^{i-1}\left(p^{s_i} \mid \det\left(M\left(i-1\right)\right)\right) \leq \sum_{t=2}^{\infty} \beta_{p,1}^{t^2} \frac{3}{p^{s_i-t}} + \beta_{p,s_i} \leq \frac{3\beta_{p,1}^4}{p^{s_i-2}} \frac{1}{\left(1-\beta_{p,1}^5 p\right)} + \beta_{p,s_i}. \tag{5.53}$$

To prove the inequality $\mathcal{P}^{i-1}\left(p^{s_i} \mid \det\left(M\left(i-1\right)\right)\right) \leq \frac{3}{p^{s_i}}$, we have to consider several cases. For $p > 3$ we use the bounds $\beta_{p,1} \leq \frac{2}{p+1}$ and $\beta_{p,s_i} \leq \frac{2}{p^{s_i}+1}$.
Then we have

$$\frac{3 \cdot 2^4 p^2 (p+1)}{p^{s_i}\left((p+1)^5 - p2^5\right)} + \frac{2}{p^{s_i}+1} < \frac{2}{p^{s_i}} + \frac{1}{p^{s_i}} \frac{48p^2(p+1)}{(p+1)^5 - (p+1)2^4} < \frac{2}{p^{s_i}} + \frac{1}{p^{s_i}} \frac{48p^2}{(p+1)^4 - 2^4}$$
$$< \frac{2}{p^{s_i}} + \frac{1}{p^{s_i}} \frac{48p^2}{25p^2 + 4 \cdot 5p^2 + 6 \cdot p^2 + 4 \cdot 5 + 1 - 16} < \frac{2}{p^{s_i}} + \frac{1}{p^{s_i}} \frac{48p^2}{51p^2} < \frac{3}{p^{s_i}}.$$

For $p = 3$ it can be explicitly checked that $\mathcal{P}^{i-1}\left(p^{s_i} \mid \det\left(M(i-1)\right)\right) < \frac{3}{p^{s_i}}$ using the bound $\frac{p+1}{p^2+1}$ for $\beta_{p,1}$ (notice that $s_i \geq 2$). In this case we get $\frac{1}{3^{s_i}} \frac{3\left(\frac{2}{5}\right)^4 3^2}{\left(1-\left(3\frac{2}{5}\right)^5\right)} + \frac{2}{3^{s_i}} < \frac{1}{3^{s_i}} 2.75$.
For $p = 2$ we have to consider $2^2, 2^3, 2^4$ and 2^{s_i} for $s_i > 4$ separately and use the sharper bound $\beta_{p,1} \leq \frac{p^{s_i-1}+1}{p^{s_i}+1}$. Let us rewrite (3c) in this cases.

- $s_i = 2$:

$$\mathcal{P}^{i-1}\left(2^2 \mid \det\left(M\left(i-1\right)\right)\right) \leq \sum_{t=2}^{k_{i-1}} \beta_{p,1}^{t^2} + \beta_{p,2} \leq \beta_{p,1}^4 \frac{1}{1-\beta_{p,1}^5} + \beta_{p,2}.$$

  As $\beta_{p,1} \leq \frac{2+1}{4+1}$ we have $0.65 < 0.75$.
- $s_i = 3$:

$$\begin{aligned}\mathcal{P}^{i-1}\left(2^3 \mid \det\left(M\left(i-1\right)\right)\right) &\leq \sum_{t=3}^{k} \beta_{p,1}^{t^2} + \beta_{p,1}^4 \cdot 1 + \beta_{p,3} \\ &\leq \beta_{p,1}^9 \frac{1}{1-\beta_{p,1}^7} + \beta_{p,1}^4 + \beta_{p,3}.\end{aligned}$$

  As $\beta_{p,1} \leq \frac{4+1}{8+1}$ we have $0.33 < 0.375$.
- $s_i = 4$:

$$\begin{aligned}\mathcal{P}^{i-1}\left(2^4 \mid \det\left(M(i-1)\right)\right) &\leq \sum_{t=4}^{n} \beta_{p,1}^{t^2} + \beta_{p,1}^9 \cdot 1 + \beta_{p,1}^4 \mathcal{P}^i\left(2^2 \mid \det\left(M(i)\right)\right) + \beta_{p,4} \\ &\leq \beta_{p,1}^{16} \frac{1}{1-\beta_{p,1}^9} + \beta_{p,1}^9 + \beta_{p,1}^4 \frac{3}{4} + \beta_{p,4}.\end{aligned}$$

  As $\beta_{p,1} \leq \frac{8+1}{16+1}$ we have $0.18 < 0.1875$.

- $s_i > 4$:
  We use inequality (5.53) with $\beta_{p,1}$ bounded by $\frac{p^4+1}{p^5+1}$. We get that

$$\mathcal{P}^{i-1}(2^{s_i} \mid \det(M(i-1))) \leq$$
$$\frac{1}{2^{s_i}}\left(\frac{3\left(2^4+1\right)^4 2^2}{\left(2^5+1\right)^4\left(\left(2^5+1\right)^5\right)-2\left(2^4+1\right)}+2\right) < 2.92\frac{1}{2^{s_i}} < \frac{3}{2^s}.$$

We have thus proven that $\mathcal{P}^{i-1}(p^{s_i}|\det(M(i-1))) \leq \frac{3}{p^{s_i}}$ for every $s_i > 0$ and every size k_{i-1} of $M(i-1)$. The recursive scheme is possible as while i increases, the size k_{i-1} of matrix $M(i-1)$ gets smaller or stays the same and the power s_i decreases.

Thus, by induction, we conclude that $\mathcal{P}(p^s|\det(UMV))$ is also less than or equal $\frac{3}{p^s}$.

5.4.3 Comparison with [10]

In the situation of Thm. 5.4.5 let us consider the case $p^s \mid \lambda$ i.e. the case when the distribution is uniform. Let us take $q = \frac{1}{p}$. Suppose that $k = n$ and $U, V = \mathrm{Id}$. In this case we may apply [10, Cor. 2.2]. We obtain

$$\mathcal{P}(\mathrm{rank}(A) = n) = \prod_{i=s}^{n+s-1}(1-q^i).$$

A simple inductive reasoning shows that

$$\prod_{i=s}^{n+s-1}(1-q^i) \leq \sum_{i=s}^{n+s-1} q^i \leq q^s\frac{1}{1-q} = \frac{1}{p^s}\frac{p}{p-1} \leq \frac{1}{p^{s-1}(p-1)}.$$

Yet, it is not immediately visible how to generalize the result of [10] to the non-uniform case. In particular, it is problematic whether recursion can be based on [10, Lem 2.2].

6

Preconditioned Chinese Remaindering Algorithm for the Determinant Computation

6.1 Classic Chinese Remaindering Algorithm

A classical method to find a large integer number x is to use the Chinese Remaindering (CR) Algorithms and reconstruct the result from its modular images using the Chinese remaindering theorem. If x is computed modulo several distinct primes p_t, then its value x_t modulo $M_t = p_0 \cdots p_t$ in the symmetric range is reconstructed based on the Chinese Remaindering theorem.

The value of x is thus found as soon as the product of p_i exceeds $2|x|$. We know that the product is sufficiently big if it exceeds some upper bound $2H$ on this value, $|x| \leq H$. This leads to a certified algorithm, which returns x with surety.

Probabilistically, we may assume that x has been obtained, if the reconstructed result remains identical for several successive additions of modula. The principle of this latter early termination (ET) is thus to stop the reconstruction before reaching the upper bound, as soon as the result remains the same for several steps, see [39]. This leads to a Monte Carlo type algorithm, which returns x with high probability.

More details on the procedure can be found in [70, Sec.5.4,5.5]. In Chapter 10, a survey on CR Algorithm can be found, which is an attempt to summarize and generalize existing results on this problem. In this chapter we restrict our interest to the case of determinant computation and only provide the results that are necessary to introduce the preconditioned CR Algorithm.

6.1.1 Requirements for CRA

The algorithm takes as input matrix A and the probability of error $0 \leq \epsilon < 1$. If $\epsilon > 0$ early termination strategy is used and the algorithm is Monte Carlo type. The result returned is correct with a probability at least $1 - \epsilon$.

The algorithm requires the following functions: $reconstruct(x_t, y_0, \ldots, y_t, p_0, \ldots, p_t)$ and $terminated(\epsilon, P, H)$ and $iteration(y_t, p_t)$. In a CR Algorithm to compute $x \in \mathbb{Z}$, H is a bound such that $|x| \leq H$. Additionally, the algorithm requires a set P of primes greater than l for a given $l \in \mathbb{R}$, and procedure to choose a prime from P randomly and uniformly in negligible time.

The functions *iteration*, *reconstruct* and *terminated* fulfill the requirements

1. $terminated(\epsilon, P, H)$ returns true if $x_t = x$ with probability at least $1 - \epsilon$.
2. $iteration(y_t, p_t)$ returns $y_t = x \mod p_t$.
3. $reconstruct(x_t, y_0, \ldots, y_t, p_0, \ldots, p_t)$ returns $-\lceil \frac{\prod_{i=0}^{t} p_i}{2} \rceil + 1 \leq x_t \leq \lfloor \frac{\prod_{i=0}^{t} p_i}{2} \rfloor$ such that x_t is the solution to the equation

$$\begin{aligned} y_0 &= x \mod p_0 \\ &\ldots \\ y_t &= x \mod p_t; \end{aligned}$$

   By the Chinese Remaindering Theorem, x_t exists and is unique.

One step of the CRA loop consist of

1. choosing a prime p_t from the set P, coprime with $\prod_{i=0}^{t-1} p_i$. We assume that this can be done in $O(1)$ bit operations.
2. computing $y_t = x \mod p_t$ by $iteration(y_t, p_t)$.
3. reconstructing $d_t = x \mod \prod_{i=0}^{t} p_i$ by $reconstruct(d_t, y_0, \ldots, y_t, p_0, \ldots, p_t)$. This can be realized in $O(t \log(t) \log(\log(t)))$ bit operations, assuming that previous results d_{t-1} and $\prod_{i=0}^{t-1} p_i$ are known.
4. checking for early termination by $terminated(\epsilon, P, H)$. This requires checking if $d_t = d_{t-1} = \cdots = d_{t-k_{max}}$ or $t > N$, where k_{max}, N are computed depending on the parameters and current state of the algorithm. This can be done in $O(1)$ bit operations.

CRA steps are run until *terminated* returns true.

In general, there are at most $O(\log(|x|))$ steps of the CRA loop and the cost of the algorithm is $\log(|x|)$ times the cost of one iteration.

6.1.2 Reconstruction and Termination in CRA

Let $M_t = \prod_{i=0}^{t} p^i$. Function *reconstruct* can be realized by the following incremental formula:

$$d_t = d_{t-1} - ((d_{t-1} - y_t)_{p_t} (M_{t-1}^{-1})_{p_t})_{p_t} \cdot M_{t-1} \tag{6.1}$$

where $(-)_{p_t}$ denote a computation modulo p_t. We assume that all modular operations are done in the symmetric range i.e. $\mathbb{Z}_q = \{-\lceil \frac{q}{2} \rceil + 1, \ldots, \lfloor \frac{q}{2} \rfloor\}, q \in \mathbb{N}$. For the cost estimation and other possibilities, see Sec. 10.4.

The following theorem gives the conditions for certification and early termination of the CR loop.

Lemma 6.1.1 (Termination in CRA) *Let $x \in \mathbb{Z}$. Let $l \in \mathbb{R}, l > 1$.*

1. *Suppose that distinct primes p_i, $i = 0, 1, \ldots$ greater than l are sampled from a set of primes P. Let $t \in \mathbb{N}$ and let x_t be the value of x modulo $p_0 \cdots p_t$ computed in the symmetric range. Then x equals x_t if*

$$2|x| < \prod_{i=0}^{t} p_i \tag{6.2}$$

and consequently

$$x_t = x, \text{ if } t \geq N' = \begin{cases} \lceil \log_l(|x|) \rceil & \text{if } x \neq 0 \\ 0 & \text{if } x = 0 \end{cases}. \tag{6.3}$$

2. *Let $H \in \mathbb{R}$ be such that $|x| \leq H$. Let $N = \lceil \log_l(H) \rceil$. Suppose that distinct primes $p_0, \ldots, p_N$ greater than l are uniformly and randomly sampled from a set of primes P, where $|P| > 2\log_l(H)$. Denote by $\mathcal{P}$ the distribution of finite sequences $p_0, \ldots, p_N \in P^N$ such that $p_i \neq p_j$ for $i, j = 0, \ldots, N, i \neq j$.*

   Let x_t be the value of x modulo $p_0 \cdots p_t$ computed in the symmetric range and let $0 < \epsilon < 1$. We have:

   (i) suppose that $t, k \in \mathbb{N}, t + k \leq N$;
   if $x_t = x_{t+1} = \cdots = x_{t+k}$ and

$$\frac{R'(R'-1)\ldots(R'-k+1)}{(|P|-t-1)\ldots(|P|-t-k)} < \epsilon, \tag{6.4}$$

   where

$$R' = \lfloor \log_l \left(\frac{H + |x_t|}{p_0 p_1 \ldots p_t} \right) \rfloor,$$

   then $\mathcal{P}(x_t \neq x) < \epsilon$.

   (ii) suppose that $t, k \in \mathbb{N}, t + k \leq N$;
   suppose that $x_t = x_{t+1} = \cdots = x_{t+k}$ and $k \geq k_{max}$, where

$$k_{max} = \lceil \frac{\log(1/\epsilon)}{\log(P') - \log(\log_l(H))} \rceil, \tag{6.5}$$

   and $P' = |P| - \lfloor \log_l(H) \rfloor$;
   then $\mathcal{P}(x_t \neq x) < \epsilon$.

PROOF See Sec. 10.6 and Lem. 10.6.1 and Lem. 10.6.3 for the proof.

Lemma 6.1.1 defines $N' = \log_l(|x|) \leq N = \log_l(H)$ that should be used in *terminated* provided that set P of primes fulfills certain requirements. Namely, l and P should be such that $|P| > 2\log_l(H)$. From the implementational point of view, set P have to be finite and it is convenient to include primes of the same bit size in the set. We will assume that given H, $l \in \mathbb{R}$ can be determined, such that the set P of primes $l < p_i < 2l$ has the cardinality $|P| > 2\log_l(H)$. We will also assume that it can be done in negligible time and, for simplicity, that the resulting set contains word-size primes. This allows us to assume that the time of modular arithmetic operations is $O(1)$. Complexity considerations are done subject to this condition.

Notice that, k_{max} can be updated 'on-the-fly', by Eq. (6.4), resulting in an output-dependent early termination. For more details on the termination of CR Algorithm see Sec. 10.6

6.1.3 Modular Determinant Computation

Let us consider the choice of *iteration* and its complexity for the determinant computation. The CR Algorithm can be applied to the determinant computation which results in a simple algorithm. The effectiveness of the algorithm is due to the fact that fast methods exist for the computation of the determinant modulo a prime. Moreover, the use of early termination techniques make the algorithm ideal for the computation of small determinants. The CRA determinant algorithm is easy adjust to both sparse and dense matrix cases.

Let $n \in \mathbb{N}, n > 0$ and let us suppose that we are computing the determinant of a $n \times n$ matrix A. Moreover, let $\|A\| = \max_{i,j=1..n}(|A_{ij}|)$ denote the maximal entry of A in absolute value.

In the dense case, the computation of determinant over a modular field is tied to matrix multiplication via block recursive matrix factorizations [68]. Therefore, using BLAS routines we obtain an optimal algorithm for the modular determinant computation which has the complexity of $O(n^\omega)$ bit operations, where ω is the exponent of matrix multiplication. The value of ω is 3 for the classical algorithm, and 2.375477 for the Coppersmith-Winograd method, see [22]. The additional cost of taking a modular image of a $n \times n$ matrix A, whose entries are bounded in absolute value by $\|A\|$ is $O(n^2 M(\log(\|A\|)))$. By $M(x)$ we denote the cost of multiplication of two x-bit integers. $M(x)$ is equal $O(x^2)$ for the classic multiplication and the best complexity by the fast multiplication algorithm of Schönhage-Strassen is $M(x) = O(x\log(x)\log(\log(x)))$, see [70, Tab. 8.6].

In the sparse case, black box methods offer a space effective solutions of the computational complexity $O(\Omega n)$, where Ω is the number of non-zero entries of a matrix, see [40, Table 4]. In some cases, elimination techniques offer faster solutions in the sparse case, see [40]. Adaptive solutions is suggested in [42]. The additional cost of taking a modular image of A is $O(\Omega M(\log(\|A\|)))$.

6.1.4 The Number of Steps in the CRA Loop

In the case of determinant computation the bound H for the absolute value of the determinant is computed with regards to matrix size and the maximal value of matrix entries.

Proposition 6.1.2 (Hadamard's Bound, cf. [70]) Let $n \in \mathbb{N}, n > 0$ and let A be a $n \times n$ matrix. Let $\|A\|$ denote the maximal entry of A in the absolute value i.e. $\|A\| = \max_{i,j=1..n}(|A_{ij}|)$. Then

$$\det(A) \leq H = n^{\frac{n}{2}} \|A\|^n \tag{6.6}$$

and H is called the Hadamard's bound on A. Let log denote the binary logarithm. We have

$$\log(H) \in O(n \log(n\|A\|)).$$

PROOF The theorem can be found for example in [70, Thm. 16.6].

Lemma 6.1.1 defines $N = \log_l(H)$ as the upper bound on the number of iterations for *terminated* function in the CR Algorithm for the determinant computation.

6.2 Preconditioned Chinese Remaindering Algorithm

The early terminated CRA scheme is optimal in the sense that the state-of-the art algorithms offer optimal solutions for the modular determinant computation and early termination results in sensitivity to the actual output. Therefore a way to improve this scheme lies in reducing the number of iterations, which can be done by reducing the value x that is to be reconstructed. In the case of the determinant computation, this idea was first developed by Abbott et al. in [2].

Let $n \in \mathbb{N}, n > 0$ and A be a $n \times n$ integer matrix. Denote by s_n the largest invariant factor of A. The idea of [2] is to combine CRA with the approach of [103], i.e. to use system solving to approximate the determinant by s_n and recover only the remaining part $(\frac{\det(A)}{s_n})$ via Chinese remaindering.

This can be realized by means of *preconditioned* CRA loop. In addition to regular CRA requirements, see Alg. 10.1.1, preconditioned CRA requires an integer preconditioner D, such that it can be guaranteed that $\frac{\det(A)}{D}$ is integer. The behavior of the algorithm is modified in only one place: the modular determinant y_t returned by $iteration(y_t, p_t)$ is divided by D in modular arithmetics. Additional cost at this step is only that of modular division.

Alg. 6.2.1 is an outline of a procedure to compute the determinant using CRA loops with early termination, correctly with probability $1 - \epsilon$. The running time of the algorithm is output dependent. The termination condition is computed on-the-fly and is also output-dependent.

Algorithm 6.2.1 Early Terminated Preconditioned CRA

Require: $n \times n$ integer matrix A,
Require: H - Hadamard's bound ($H = (\sqrt{n}\|A\|)^n$ by Prop. 6.1.2),
Require: $D > 0$, such that $\det(A)/D \in \mathbb{Z}$,
Require: $l > 1$, a set P of primes greater than l and less than $2l$, $|P| > 2\log_l(H/D)$,
Require: $0 \leq \epsilon < 1$,
Require: Function $iteration(y_t, p_t)$ which produce $y_t = \det(A) \mod p_t$,
Ensure: The integer determinant of A, correct with probability at least $1 - \epsilon$;

```
t = 0;
repeat
    Choose uniformly and randomly a prime p_t from the set P;
    P = P\{p_t} ;
    Compute iteration(y_t, p_t);
    y_t = y_t/D  mod p_t;
    Compute reconstruct(x_t, x_{t-1}, y_t, p_0 ··· p_{t-1}, p_t) by Eq. (6.1);
    k = max{s : x_{t-s} = ··· = x_t};
    R' = ⌊log_l (H/D+|x_t|)/(p_0 p_1 ... p_{t-k})⌋; # see 6.1.1(2i)
    t = t + 1;
until (R'(R'-1)...(R'-k+1))/((|P|+k)...(|P|+1)) < ε or ∏_{i=0}^{t-1} p_i > 2H/D
Return D · x_{t-1};
```

A difference between *preconditioned* CRA loop for $\det(A)$ with preconditioner D and a CRA loop for $\frac{\det(A)}{D}$ is minor and to some extent only lexical. The procedures differ in the definition of *iteration* function and in the returned result ($\det(A)$ vs. $\frac{\det(A)}{D}$) but are computationally equivalent. Therefore any statement that is true for the CR Algorithm computing $\frac{\det(A)}{D}$ are true for the preconditioned CRA computing $\det(A)$. By putting $D = 1$, we obtain in Alg. 6.2.1 the classic CR Algorithm for the determinant computation.

6.2.1 Correctness of the Algorithm

Theorem 6.2.1 (Correctness of Preconditioned CRA) *Let $n \in \mathbb{Z}, n > 0$, and A be a $n \times n$ matrix. Let H be the Hadamard's bound for A and let $D > 0$ be such that $\det(A)/D \in \mathbb{Z}$. Let $l > 1$ and let P be a set of more than $2\log_l(H/D)$ primes greater than l. Suppose that primes are sampled randomly and uniformly from P.*

Let $0 \leq \epsilon < 1$. Let n, A, H, D, l, P and ϵ be the input of Alg. 6.2.1. Then Alg. 6.2.1 returns $\det(A)$ correctly with probability at least $1 - \epsilon$.

PROOF Notice that t is changed to $t + 1$ before the termination conditions are checked.

If $\epsilon = 0$ i.e. no early termination is allowed, the algorithm stops if $\prod_{i=0}^{t-1} p_i$ is greater than $2H/D$. This means that $\prod_{i=0}^{t-1} p_i$ fulfills Eq. (6.2), and therefore $x_{t-1} = \det(A)/D$ by Lem. 6.1.1 with probability 1.

If $0 < \epsilon < 1$ the termination is also possible as soon as $\frac{R'(R'-1)...(R'-k+1)}{(|P|+k)...(|P|+1)} < \epsilon$, which is consistent with Eq. (6.4), if we notice that the current size of P is updated by the

algorithm. Therefore by Lem. 6.1.1 (2i) the probability that $x_{t-1} = \det(A)/D$ is at least $1 - \epsilon$. Thus, the algorithm returns $\det(A)$ with the required probability.

6.2.2 The Complexity of the Algorithm

Let us now estimate the complexity of Alg. 6.2.1. We assume that the cost of constructing l and P and the cost of generating random primes is negligible compared with the other elements of the algorithm. We denote by $M(x)$ the cost of multiplication of two x-bit integers.

Theorem 6.2.2 (Cost of Preconditioned CRA) *Let $n \in \mathbb{Z}, n > 0$, and A be a $n \times n$ matrix. Let H be the Hadamard's bound for A and let D be an integer, such that $\det(A)/D \in \mathbb{Z}$. Let $l > 1$ and let P be a set of more than $2\log_l(H/D)$ randomly distributed primes greater than l and less than $2l$. Let $0 < \epsilon < 1$. Suppose that Alg. 6.2.1 is run on input A, H, D, l, P and ϵ.*

We assume that $l \in O(1)$ and that the cost of choosing a prime and updating P is also negligible.

1. *Suppose than A is a dense matrix and an algorithm of complexity $MDet(A) = O(n^\omega)$ is used to compute $\det(A) \mod p, p \in P$ in iteration.*

   Then the complexity of Alg. 6.2.1 is

$$O((\log(|\det(A)/D|) + \log(1/\epsilon))(n^2 M(\log(\|A\|)) + n^\omega).$$

2. *Suppose that A is a sparse matrix with Ω entries and a black box algorithm of complexity $MDet(A) = O(n\Omega)$ is used to compute $\det(A) \mod p, p \in P$ in iteration.*

   Then the complexity of Alg. 6.2.1 is

$$O((\log(|\det(A)/D|) + \log(1/\epsilon))(M(\log(D)) + \Omega M(\log(\|A\|)) + n\Omega)).$$

PROOF Let us estimate the cost of one iteration.

By assumption, the cost of choosing p_t and updating P is constant.

Computation of $iteration(y_t, p_t)$ requires a computation of $A_t = A \mod p_t$ and a computation of $\det(A_t) \mod p_t$. The cost of modular image computation in the case of dense matrix A is $O(n^2 M(\log(\|A\|)))$. In the case of sparse matrix, the cost is $O(\Omega M(\log(\|A\|)))$. The cost of determinant computation is $O(n^\omega))$ or $O(\Omega n)$ respectively, see Sec. 6.1.3.

In step 6 we have to divide y_t by D in the modular arithmetic. First, an image $D \mod p_t$ has to be computed, which costs $O(M(\log(D)))$ bit operations. As we assume that $l \in O(1)$, the cost of modular division can be ignored. As $D \leq H$, this is less than $M(n\log(n\|A\|)) \in M(\log(\|A\|)) \times O^{\approx}(n\log(n))$.

By Prop. 10.4.3, the cost of reconstruction of x_t is $O(M(t))$, where t is the index of iterations. As the product of primes used in the algorithm is at most $2H/D$ (see line 11),

t is $O(\log(H))$ in the asymptotically worst case. Therefore the cost of reconstruction is at most $M(n\log(n\|A\|)) \in M(\log(\|A\|)) \times O^{\approx}(n\log(n))$.

Finally, we may assume that the cost of computing k is constant, as updating the maximum requires just one comparison, see 10.6.4. As R' may be computed in floating point arithmetics, we may also assume that the cost of its computation is $O(1)$.

The total number of iterations needed to obtain $\det(A)$ is less than or equal $\lceil \log_l(|\frac{\det(A)}{D}|)\rceil$ by Lem. 6.1.1. However, additional $k_{max}\lceil \frac{\log(1/\epsilon)}{\log(P')-\log(\log_l(H/D))}\rceil = O(\log(1/\epsilon))$ iterations are needed to confirm the result with probability $1-\epsilon$, see Lem. 6.1.1(2ii).

Let us evaluate the cost of the algorithm in the dense and sparse case.

1. In the dense case the cost is dominated by the cost of *iteration*. Indeed, the cost of division by D
$$O(M(\log(D))) = O(n^2 M(\log(\|A\|))),$$
and the cost of reconstruction
$$M(\log(\|A\|)) \times O^{\approx}(n\log(n)) \in O(n^2 M(\log(\|A\|)))$$
is bounded by the cost of taking a modular image of A. Therefore the complexity is
$$O((\log(|\det(A)/D|) + \log(1/\epsilon))(n^2 M(\log(\|A\|)) + n^{\omega}).$$

2. In the sparse case analogous reasoning yields
$$O((\log(|\det(A)/D|) + \log(1/\epsilon))(M(D) + \Omega M(\log(\|A\|)) + n\Omega)).$$

6.3 Preconditioning in Abbott's at al. Algorithm

In [2] the authors propose to take D equal to the largest invariant factor of the matrix, or to be explicit, its factor computed as the least common multiple of the denominators of solutions to several linear systems.

6.3.1 Correctness of Preconditioner

Proposition 6.3.1 ensures that we will always obtain a divisor of the largest invariant factor in this way.

Proposition 6.3.1 (Solution to a Linear System vs. s_n) Let A be an invertible $n\times n$ integer matrix. Let s_n be the largest invariant factors of A. Let $b \in \mathbb{Z}^n$ be an integer vector. Let x be a solution to the system of equations $Ax = b$. Then $s_n \cdot x \in \mathbb{Z}$.

PROOF Let $S = \mathrm{diag}(s_1, \ldots, s_n)$ be the Smith form of A. There exist invertible matrices U and V, $|\det(U)| = |\det(V)| = 1$, such that $A = USV$, we can equivalently solve $SVx = U^{-1}b$ for $y = Vx$, and then solve the equation for x. As the determinants of U and V are equal ± 1, the least common multiple of the denominators of x and y, $d(x)$ and $d(y)$ satisfies $d(x) = d(y)$. Indeed,

$$y_i = \sum_{j=1}^{n} V_{ij} x_j = \frac{1}{d(x)} \sum_{j=1}^{n} V_{ij} d(x) x_j$$

and $\sum_{j=1}^{n} V_{ij} d(x) x_j \in \mathbb{Z}$. Therefore the denominator of y_i, $d(y_i)$ divides $d(x)$ for every $i = 1..n$, which means that $d(y) \mid d(x)$. As V^{-1} is also an integer matrix, by repeating the reasoning for $x = V^{-1}y$ we obtain that $d(x) \mid d(y)$ at the same time. Therefore we may conclude that $d(x) = d(y)$.

As S is a diagonal matrix, $y_i, i = 1..n$, can be computed explicitly as $\frac{(U^{-1}b)_i}{s_i}$. U^{-1} is an integer matrix and consequently $(U^{-1}b)_i$ is integer. Therefore $d(y_i) \mid s_i$ and $s_i \mid s_n$, for every $i = 1..n$. We may conclude that $d(y) \mid s_n$.

Finally we have that $d(x) = d(y) \mid s_n$, which means that $s_n \cdot x \in \mathbb{Z}$.

See also [2, Sec.2] for the proof.

6.3.2 Computation of Preconditioner

A method to compute s_n for integer matrices was first stated by V. Pan [103] and later in the form of the *LargestInvariantFactor* procedure (LIF) in [2, 44, 39, 112].

Algorithm 6.3.1 LIF cf. [2, 44, 39, 112]

Require: A $n \times n$ integer matrix A,
Require: Parameter $\beta \in \mathbb{N}$, $\beta > 1$,
Require: A number of iterations $r \in \mathbb{N}$, $r > 0$,
Require: A stream S_β of random integers uniformly chosen from the set $\{-\lceil \frac{\beta}{2} \rceil + 1, \ldots, \lfloor \frac{\beta}{2} \rfloor\}$,
Require: Function $Solver(x, A, b)$ which solves the equation $Ax = b$ for $x \in \mathbb{Q}^n$,
Ensure: $\tilde{s}_n$, a factor of s_n;
$\tilde{s}_n$ equals s_n with probability depending on r and β given by Thm. 6.3.2

1: $\tilde{s}_n = 1$;
2: **for** $i = 1$ to r **do**
3: Generate b_i a random vector of dimension n from the stream S_β;
4: Run $Solver(x_i, A, b_i)$;
5: $d = \mathrm{lcm}(\text{denominators of entries of } x_i)$;
6: $\tilde{s}_n = \mathrm{lcm}(\tilde{s}_n, d)$;
7: **end for**
8: **Return** : $\tilde{s}_n$.

Algorithm 6.3.1 takes as input parameters $\beta \in \mathbb{N}, \beta > 1$ and $r \in \mathbb{N}, r > 0$ which are used to control the probability of correctness; r is the number of successive solvings and β is the size of the set from which the values of a random vector b are chosen, i.e. a bound for $\|b\|$. Theorem 6.3.2 characterizes the probabilistic behavior of the LIF procedure for several choices of parameters. This theorem is a summary of known results and some new evaluations.

6.3.3 Probability of LIF Algorithm

The idea behind the algorithm of Abbott et al. [2] is that it enables us to get the largest invariant factor with high probability. Indeed, the authors are able to obtain quantitative results, that enable then to conclude that the average complexity of their algorithm is proportional to $\frac{\det(A)}{s_n}$.

Theorem 6.3.2 (Probability of LIF Algorithm) *Let $n \in \mathbb{N}, n > 0$. Let A be a $n \times n$ integer matrix, H its Hadamard's bound, and let $r > 0$ and $\beta > 1$. Let S_β be a stream of random integers randomly and uniformly chosen from the set $\{-\lceil \frac{\beta}{2} \rceil + 1, \ldots \lfloor \frac{\beta}{2} \rfloor\}$. Denote by $\mathcal{P}$ the distribution of results $\tilde{s}_n$ of Algorithm 6.3.1 with input A, β, r, S_β.*

Then the output $\tilde{s}_n$ of Algorithm 6.3.1 with input A, β, r, S_β divides s_n and is characterized by the following properties.

(i) Thm. 2 of [2]

If $r = 1$, p is a prime, $l \geq 1$, then $\mathcal{P}(p^l | \frac{s_n}{\tilde{s}_n}) \leq \frac{1}{\beta} \lceil \frac{\beta}{p^l} \rceil$;

(ii) Thm 2.1 of [44]

if $r = 2$, $\beta = 6 + \lceil 2 \log(H) \rceil$ then $s_n = \tilde{s}_n$ with probability at least 1/3;

(iii) if $r = 2$, $\beta = \lceil \log(H) \rceil$ then $\mathbf{E}(\log\left(\frac{s_n}{\tilde{s}_n}\right)) = O(1)$;

(iv) if $r = \lceil \log(\log(H)) + \log\left(\frac{1}{\epsilon}\right) \rceil$, $2 \mid \beta$ and $\beta \geq 2$ then $s_n = \tilde{s}_n$ with probability at least $1 - \epsilon$;

(v) Lem. 2 of [2]

if $r = \lceil 2 \log(\log(H)) \rceil$, $\beta \geq 2$ then $\mathbf{E}(\log\left(\frac{s_n}{\tilde{s}_n}\right)) = O(1)$;

PROOF By Prop. 6.3.1 the denominator $den(x_i)$ divides s_n. Thus, $\tilde{s}_n$ is indeed a fraction of s_n.

The distribution $\mathcal{P}$ of $\tilde{s}_n$ results from the distribution of vectors $b_1, \ldots, b_r$, which are sampled randomly and uniformly from S_β.

The proofs of (*i*) and (v) are in [2, Thm. 2, Lem. 2]. The proof of (ii) is in [44, Thm. 2.1].

To prove (iii) we adapt the proof of (ii). Let us notice that

$$\mathbf{E}(\log\left(\frac{s_n}{\tilde{s}_n}\right)) \leq \sum_{p\ prime, p|s_n} \sum_{k=1}^{\lceil \log_p(s_n)\rceil} k\log(p)\,\mathcal{P}(ord_p(\frac{s_n}{\tilde{s}_n}) = k)$$
$$= \sum_{p\ prime, p|s_n} \sum_{k=1}^{\lceil \log_p(s_n)\rceil} k\log(p)(\mathcal{P}(ord_p(\frac{s_n}{\tilde{s}_n}) \geq k) - \mathcal{P}(ord_p(\frac{s_n}{\tilde{s}_n}) \geq (k-1)))$$

This transforms to

$$\mathbf{E}(\log\left(\frac{s_n}{\tilde{s}_n}\right)) \leq \sum_{p\ prime, p|s_n} \sum_{k=1}^{\lceil \log_p(s_n)\rceil} \log(p)\,\mathcal{P}(ord_p(\frac{s_n}{\tilde{s}_n}) \geq k)$$
$$= \sum_{p\ prime, p|s_n} \sum_{k=1}^{\lceil \log_p(s_n)\rceil} \log(p)\,\mathcal{P}(p^k \mid \frac{s_n}{\tilde{s}_n})$$

By (i), taking into account that $r = 2$, this is less than or equal to

$$\mathbf{E}(\log\left(\frac{s_n}{\tilde{s}_n}\right)) \leq \sum_{p\ prime, p|s_n} \sum_{k=1}^{\lceil \log_p(s_n)\rceil} \log(p)\left(\frac{1}{\beta}\lceil\frac{\beta}{p^k}\rceil\right)^2$$

As $\frac{1}{\beta}\lceil\frac{\beta}{p^k}\rceil$ is bounded by $\frac{1}{\beta} + \frac{1}{p^k}$ this can be further expressed as

$$\sum_{p\ prime, p|s_n} \sum_{k=1}^{\lceil \log_p(s_n)\rceil} \log(p)\left(\frac{1}{\beta}\lceil\frac{\beta}{p^k}\rceil\right)^2 \leq \sum_{p\ prime, p|s_n} \sum_{k=1}^{\lceil \log_p(s_n)\rceil} \log(p)\left(\frac{1}{\beta} + \frac{1}{p^k}\right)^2$$
$$\leq \sum_{p\ prime} \sum_{k=1}^{\infty} \log(p)\frac{1}{p^{2k}} + \frac{2}{\beta}\sum_{p|s_n}\sum_{k=1}^{\infty}\log(p)\frac{1}{p^k} + \frac{1}{\beta^2}\sum_{p|s_n}\sum_{k=1}^{\lfloor \log_p(s_n)\rfloor}\log(p) \leq$$
$$\sum_{p\ prime} \log(p)\frac{1}{p^2-1} + \frac{2}{\beta}\sum_{p|s_n}\log(p)\frac{1}{p-1} + \frac{1}{\beta^2}\sum_{p|s_n}\log(p)\log_p(s_n)$$
$$\leq \sum_{p\ prime} \log(p)\frac{1}{p^2-1} + \frac{2}{\beta}\log(s_n) + \frac{1}{\beta^2}\log^2(s_n) \leq 5 \in O(1).$$

The choice of $\beta = \lceil\log(H)\rceil \geq \log(|\det(A)|) \geq \log(s_n)$ ensures that $\frac{\log(s_n)}{\beta} \leq 1$. Moreover, the sum $\sum_{p\ prime}\log(p)\frac{1}{p^2-1}$ can numerically be bounded by 1.78. Therefore the expected value is bounded by $5 \in O(1)$.

To prove (iv) we slightly modify the proof of (v) in the following manner. From (i) we notice that for every prime p dividing s_n, the probability that it divides the missed part of s_n satisfies:

$$\mathcal{P}(p \mid \frac{s_n}{\tilde{s}_n}) \leq \left(\frac{1}{\beta}\lceil\frac{\beta}{p}\rceil\right)^r \leq \left(\frac{1}{2}\right)^r.$$

Indeed,

$$\left(\frac{1}{\beta}\lceil\frac{\beta}{p}\rceil\right)^r \leq \begin{cases} \frac{1}{\beta}\frac{\beta}{2} & p=2 \\ \frac{2}{p+1} & 2<p<\beta \\ \frac{1}{\beta} & p\geq\beta\geq 2 \end{cases}$$

which is not more than $\frac{1}{2}$ in every case.

As there are at most $\log(H)$ such primes, we get

$$\mathcal{P}(s_n=\tilde{s}_n)\geq 1-\log(H)(1/2)^r \geq 1-\log(H)2^{-\log(\log(H))-\log\left(\frac{1}{\epsilon}\right)} = 1-\log(H)\frac{1}{\log(H)}\epsilon.$$

6.3.4 Algorithm of Abbott et al.

To summarize this section let us describe the algorithm of Abbott et al. [2].

Let $n\in\mathbb{N}, n>0$, A a $n\times n$ integer matrix, H its Hadamard's bound (see Prop. 6.1.2) and ϵ be given. The determinant of A is computed by Alg. 6.3.2 with probability at least $1-\epsilon$.

Algorithm 6.3.2 The Algorithm of Abbott et al. [2]

Require: $n\times n$ integer matrix A;
Require: H - Hadamard's bound ($H=(\sqrt{n}\|A\|)^n$);
Require: $0<\epsilon<1$;
Require: Function $Solver(x,A,b)$ which solves the equation $Ax=b$ for $x\in\mathbb{Q}^n$;
Require: Function $iteration(y_i,p_i)$ which produce $y_i=\det(A) \mod p_i$;
Ensure: The integer determinant of A, correct with probability at least $1-\epsilon$.

1: Set $r=2,\beta=\lceil\log(H)\rceil$.
2: Run Alg. 6.3.1 for A,β,r,S_β to obtain$\tilde{s}_n$.
3: Determine integer $l>0$, set of random primes P such that $l<p_i<2l$, for every $p_i\in P$ and $|P|>2\log_l(H/\tilde{s}_n)$.
4: Run Alg. 6.2.1 for $A,H,\tilde{s}_n,l,P$ and ϵ.

The algorithm requires a solver which allows to solve equation $Ax=b$ in rational numbers for an integer vector b (see requirements of Alg. 6.3.1) and a procedure *iteration* which computes the determinant modulo a prime. The choice of *iteration* was already discussed in section 6.1.3. We will discuss the choice of *Solver* in the following section.

6.3.5 Choosing a Solver

Let $n\in N, n>0$ and let A be a $n\times n$ integer matrix, H its Hadamard's bound (see Prop. 6.1.2), let b be an integer vector of size n such that the entries of b are bounded in absolute value by $\beta=\lceil\log(H)\rceil$.

By Thm. 6.3.2(iii), the choice of $r = 2$ and $\beta = \lceil \log(H) \rceil \in O(n \log(n\|A\|))$ suffices to obtain $\frac{s_n}{\tilde{s}_n} \in O(1)$ by Alg. 6.3.1, where s_n is the largest invariant factor of A. There exist several algorithms, that can be used to solve the integer equation $Ax = b$ for $x \in \mathbb{Q}^n$ in this case.

In the case of dense matrix, the most widely used algorithm is the p-adic solver of Dixon [27]. The complexity of one linear solving by Dixon p-adic lifting is $O(n^3 \log^2(n\|A\|) + n \log^2(\beta))$ (see [97]), which for $\beta \in O(n \log(n\|A\|))$ is $O(n^3 \log^2(n\|A\|))$.

There exists also an early terminated version of p-adic solver, which has the output dependent complexity of $O(n^\omega + Kn^2 \log(\|A\|n) + K \log(\beta)) \in O(Kn^2 \log(\|A\|n))$ if only K terms of p-adic approximation of x are needed to reconstruct the result. However, an erroneous answer $\tilde{s}_n \nmid \det(A)$ would force preconditioned CRA to loop until the maximal number of iterations is reached. We regard this as a main drawback of the early termination strategy and would restrain from using this variant of solving in preconditioned CRA algorithm.

Finally, the paper of Storjohann [122] gives a recipe for a linear solver of best theoretical complexity of $O(n^\omega B(\log(n\|A\|)) \log(n))$, see [122, Thm. 38]. $B(x)$ is the cost of gcd-like operations on x bit integers, see Ex. 2.1.6, for the complexity. $B(\log(n\|A\|)$ is $O^\sim(\log(n\|A\|))$. The possibility of using this algorithm could be consider in the theoretic considerations. However, no implementation is available these days, which prevents us to evaluate this possibility experimentally. Without using fast matrix multiplications, the algorithm obtains the complexity of $O(n^3 B(\log(n\|A\|)))$ bit operations.

Let us now suppose that A is a sparse matrix with Ω entries. The classic approach of Wiedemann [133] has the complexity $O(n^2 \Omega \log(n\|A\|))$, same as the determinant computation. A combination of Wiedemann's method with the p-adic approach was studied in [77] but does not lead to an improvement in terms of worst case complexity.

However, more recently, the sparse solver [43], which up-to-date has the best complexity can be used solve a system of equations, and consequently to obtain s_n at a lower cost than the determinant computation. The cost of the solver, see [43, Thm. 3.3] is that of $O^\sim(n^{1.5} \log(\|A\| + \beta))$ matrix-vector products and $O^\sim(n^{2.5} \log(\|A\| + \beta))$ additional arithmetic operations. By similar reasoning as in [97], we are able to reduce this estimation to $O(n^{1.5} \log(n\|A\|) + n^{0.5} \log(\beta))$ matrix-vector products and $O(n^2 \log(n\|A\|) \times (O^\sim(n^{0.5}) + \log(\|A\|)) + n^2 \log(\beta)(O^\sim(n^{0.5}) + \log(n\|A\|)) + n \log^2(\beta))$ additional arithmetic operations. Indeed, the p-adic scheme of [43] requires $O(n \log(n\|A\| + \log(\beta)))$ iterations which cost $O(n(\log(\|A\|) + \log(\beta)) + m\mu(n) + nO^\sim(n^{0.5}))$ each. Here, the exact complexity of $O^\sim(n^{0.5})$ depends on the type of block projections used in the algorithm.

The cost of one matrix-vector product is $O(\Omega)$. Therefore for $\beta \in O(n \log(n\|A\|))$ and $\Omega \geq n$ the cost of the solver is

$$O(\Omega n^{1.5} \log(n\|A\|) + n^2 \log(n\|A\|) \log(\|A\|))$$

bit operations.

In [128], the author presents another possibility, which could lead to better running times in the case of well-conditioned matrices. In this case, numerical procedures can be used to quickly approximate the solution, while symbolic methods such as rational reconstruction allows to deduce the exact result from it. If ill-conditioned input is detected, one of the above-mentioned exact algorithms can be run. The author remarks that the algorithm has very good running times for well conditioned matrices. While the worst case complexity of the algorithm is the same as for the exact algorithm, the use of this solver can be considered for practical speed-up.

6.3.6 Complexity of the Determinant Algorithm of Abbott et al.

To sum up, let us now evaluate the average complexity of Abbott's et al. algorithm in the case of three solvers which are possible to be used. The p-adic algorithm of [27] is used in most the implementation for the dense matrix case. The algorithm of Storjohann [122] enables us to give the best worst case complexity. The algorithm of Eberly et al. [43] can be used in the case of a sparse matrix.

Theorem 6.3.3 (Complexity of Abbott's et al. Algorithm) *Let $n \in \mathbb{Z}, n > 0$, and A be a $n \times n$ matrix. Let H be the Hadamard's bound for A. Let $0 < \epsilon < 1$.*

For the cost computation, let us assume that $M(x)$ denote the cost of multiplication of two x-bit integers and $B(x)$ denote the cost of gcd*-like operations on x-bit integers.*

Let us consider Alg. 6.3.2 and let us specify functions Solver and iteration as below. Let us assume that l and P in Alg. 6.3.2 can be found in $O(1)$ and that the cost of prime generation and the cost of updating P is also constant.

1. *Suppose than A is a dense matrix and in iteration an algorithm of complexity $MDet(A) = O(n^\omega)$ is used to compute $\det(A) \mod p$ for $p \in P$. Suppose that Dixon p-adic lifting algorithm [27] is used in Solver.*

   Then the average complexity of Alg. 6.3.2 is

$$O(n^3 \log^2(n\|A\|) + (\log(|\frac{\det(A)}{s_n}|) + \log(\frac{1}{\epsilon}))(n^2 M(\log(\|A\|)) + n^\omega))$$

2. *Suppose than A is a dense matrix and in iteration an algorithm of complexity $MDet(A) = O(n^\omega)$ is used to compute $\det(A) \mod p$ for $p \in P$. Suppose that Storjohann's algorithm [122] is used is used in Solver.*

   Then the average complexity of Alg. 6.3.2 is

$$\begin{aligned} &O(n^\omega \log(n) B(\log(n\|A\|)) \\ &\qquad + (\log(|\frac{\det(A)}{s_n}|) + \log(\frac{1}{\epsilon}))(n^2 M(\log(\|A\|)) + n^\omega)) \end{aligned}$$

3. *Suppose that A is a sparse matrix with Ω entries and in iteration a black box algorithm of complexity $MDet(A) = O(n\Omega)$ is used to compute* $\det(A) \mod p$. *Suppose that the algorithm of [43] is used in Solver.*

   Then the average complexity of Alg. 6.3.2 is

$$O(\Omega n^{1.5}\log(n\|A\|) + n^2\log(n\|A\|)\log(\|A\|) \\ + (\log(|\frac{\det(A)}{s_n}|) + \log(\frac{1}{\epsilon}))(M(\log(s_n)) + \Omega M(\log(\|A\|)) + n\Omega)).$$

PROOF In Alg. 6.3.2 r is set to 2 and β is $O(n\log(n\|A\|))$. Therefore the cost of Alg. 6.3.1 used in step 2 of the algorithm is

1. $O(n^3\log^2(n\|A\|))$
2. $O(n^\omega\log(n)B(\log(n\|A\|)))$
3. $O(\Omega n^{1.5}\log(n\|A\|) + n^2\log(n\|A\|)\log(\|A\|))$

respectively.

By Thm 6.3.2(iii), the $D = \tilde{s}_n$ is such that the expected value $\mathbf{E}\left(\log(\frac{s_n}{\tilde{s}_n})\right)$ is $O(1)$. Therefore the expected value of $\log(\frac{\det(A)}{D})$ is $\log(\frac{\det(A)}{s_n}) + O(1)$. Moreover, $D \leq s_n$.

By Thm. 6.2.2 we obtain, that the cost of the preconditioned CRA algorithm is

1. $O((\log(|\frac{\det(A)}{s_n}|) + \log(\frac{1}{\epsilon}))(n^2M(\log(\|A\|)) + n^\omega))$ in the dense matrix case (cases 1. and 2.)
2. $O((\log(|\frac{\det(A)}{s_n}|) + \log(\frac{1}{\epsilon}))(M(\log(s_n)) + \Omega M(\log(\|A\|)) + n\Omega))$ in the sparse matrix case (case 3.)

respectively. Putting all together, we obtain the result.

When compared with the original CRA loop algorithm one have potentially reduced the average complexity of the algorithm in terms of n on the cost of rising the complexity in terms of $\log(\|A\|)$. No gain is guaranteed in terms of the worst case complexity, as in the worst case the value of $\frac{\det(A)}{\tilde{s}_n}$ can be $\det(A)$, and anyway, $\frac{\det(A)}{s_n} \in O(n\log(n\|A\|))$.

The convincing analysis of $\frac{\det(A)}{s_n}$ has yet to be performed. However, experiments suggest that it is often 1 or a small constant. The works of [2], [44], and our study on the expected number of invariant factors try to overcome this problems.

7

Bonus and Extended Bonus Idea

In their article [44], Eberly, Giesbrecht and Villard proposed another way of computing the determinant as a product of all invariant factors of a matrix. The main contribution of the paper is a procedure OIF(i) to compute an arbitrary invariant factor s_i, $i = 1, \ldots, n$ for a $n \times n$ matrix A. We describe this algorithm in Ch. 3.

Let A be a $n \times n$ matrix. For $i < n$, the output $\tilde{s}_i$ computed by OIF(i) can be larger than the actual factors s_i. This prevents us from coupling Abbott's preconditioned CRA, see Ch. 6, and Eberly's ideas directly i.e. computing consecutive factors $\tilde{s}_{n-1}, \tilde{s}_{n-2}, \ldots$ by means of OIF algorithm and reconstructing only the remaining factors $\frac{\det(A)}{s_n s_{n-1} \ldots}$ by means of the CRA loop.

The problem has to some extend been solved by Saunders and Wan in [112], as they proposed a way to compute an approximation of the penultimate factor $\tilde{s}_{n-1}$ which is always a factor of s_{n-1}. Indeed, the authors introduce the idea of computing the penultimate invariant factor (i.e. s_{n-1}) of A while computing s_n using two system solvings. The additional cost is comparatively small, therefore s_{n-1} is referred to as a bonus. The algorithm is presented in [112, Alg. 1], [113, Alg. 2.1] and [127, Alg.4].

Here, we extend this idea to the computation of the product of k last invariant factors by means of k linear system solvings. We start by remaindering the new preconditioning of Saunders and Wan in Sec. 7.1, which leads to a different variant of OIF procedure. Then starting from Sec. 7.2 we introduce the idea of an extended bonus.

7.1 New Preconditioning of Saunders and Wan

In a series of papers ([112, 113, 127]) D. Saunders and Z. Wan propose to use multiplicative preconditioners to compute ith invariant factor of matrix A.

Lemma 5.26 of [127] states what follows.

Lemma 7.1.1 (Lemma 5.26 of [127]) *Let $n, m, k \in \mathbb{N}$, $n, m \leq k > 0$. Let A be an $n \times m$ matrix and let R (resp. L) be a $m \times k$ (resp. $k \times n$) matrix. Then s_i divides $s_i(AR)$ (resp. $s_i(LA)$) for every $i = 1, \ldots, \min(n, m, k)$.*

7.1.1 Properties of Preconditioning

In [127, Thm. 5.8] the authors state that for random preconditioners L and R, $s_k(LAR)$ equal $s_k(A)$ with high probability. More precisely, [127, Lem. 5.17] characterize cases when the preconditioning is successful. For matrix M, $k \in \mathbb{N}, k > 0$ we denote by $\mu(M)$ the product $\mu_k = s_1 s_2 \cdots s_k$ of k smallest invariant factors. In our context it will be essential to improve [127, Lem. 5.17] in order to compare $\mu_k(A)$ with $\mu_k(AR)$ (resp. $\mu_k(LA)$.

First, let us recall that Thm. 5.4.5 gives the probability that p^s divides the determinant of a random matrix. We can use this result to prove the following theorem.

Theorem 7.1.2 ($\mu_k(A)$ vs. $\mu_k(AR)$) *Let $n, m, k \in \mathbb{N}, n, m \geq k > 0$ and let A be a $n \times m$ integer matrix such that the product $\mu_k(A)$ of k smallest invariant factors is non-zero. Let R be a $m \times k$ matrix. Then there exist a $k \times m$ matrix V_1, which has a trivial Smith form, such that for every prime p, $s \in \mathbb{N}, s > 0$, we have an implication*

$$If \;\; \mathrm{ord}_p\left(\frac{\mu_k(AR)}{\mu_k(A)}\right) \geq s \;\; then \;\; \mathrm{ord}_p(\det(V_1 R)) \geq s.$$

Let us now assume that $\mathcal{S}$ is a set of λ contiguous integers $\mathcal{S} = \{a, a+1, \ldots, a+\lambda-1\}$, where $a \in \mathbb{Z}$ and $\lambda > 0$. Let $\mathcal{P}_{\mathcal{S}}$ denote the uniform distribution on $\mathcal{S}$. Let $\mathcal{P}$ denote the distribution of matrices of $\mathcal{S}^{m\times k}$ such that the entries of the matrix are chosen independently of each other according to distribution $\mathcal{P}_{\mathcal{S}}$. Let R be chosen according to distribution $\mathcal{P}$. Then

(i) if $p^s < \lambda$:

$$\mathcal{P}(\mathrm{ord}_p\left(\frac{\mu_k(AR)}{\mu_k(A)}\right) \geq s) \leq \frac{3}{p^s}$$

(ii) if $p^s \geq \lambda$:

$$\mathcal{P}(\mathrm{ord}_p\left(\frac{\mu_k(AR)}{\mu_k(A)}\right) \geq s) \leq \frac{3}{\sqrt{\lambda}}.$$

PROOF Let U, V be matrices in $\mathbb{Z}^{n\times n}$ and $\mathbb{Z}^{m\times m}$ respectively, such that $A = USV$, where S is the Smith form of A. Let us rewrite the equation $AR = USVR$ in a block form. We obtain:

$$U \begin{bmatrix} S_1 & 0 \\ 0 & S_2 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} R = U \begin{bmatrix} S_1 V_1 R \\ S_2 V_2 R \end{bmatrix}$$

where S_1, S_2 are $k \times k$ and $(n-k)\times(m-k)$ matrices and V_1, V_2 are $k \times m$ and $(m-k)\times m$ matrices respectively. Moreover, since $|\det(V)| = 1$, V_1 has a trivial Smith form.

We notice that the local Smith form at p of $\begin{bmatrix} S_1 V_1 R \\ S_2 V_2 R \end{bmatrix}$ is the same as the local Smith form at p of AR.

We have that $\det(S_1V_1R) = \det(S_1)\det(V_1R)$. On the other hand $\det(S_1) = \mu_k(A)$ and $\mu_k(AR)$ divides $\det(S_1V_1R)$. Therefore

$$\mu_k(AR) \mid \mu_k(A) \cdot \det(V_1R).$$

If $\mathrm{ord}_p(\frac{\mu_k(AR)}{\mu_k(A)} \geq s)$ this implies that $\mathrm{ord}_p(V_1R)$ also has to be greater than s.

We notice that matrices V_1 and R fulfill the requirements of Thm. 5.4.5. Then the probability evaluation for $p^s < \lambda$ is a consequence of Thm. 5.4.5 for matrices V_1 and R.

In the case when $p^s \geq \lambda$, $\mathcal{P}\left(p^s | \det\left(V_1R\right)\right)$ is less than $\mathcal{P}\left(p^{\max(1,\lfloor \log_p(\lambda)\rfloor)} | \det\left(V_1R\right)\right)$.

By [127, Thm. 5.13] the bound on $\mathcal{P}\left(p \mid \det\left(V_1R\right)\right)$ for $p \geq \lambda$ is $\frac{1}{\lambda-1}$ which is less than $\frac{1}{\sqrt{\lambda}}$.

For $p < \lambda$, the bound on $\mathcal{P}\left(p^{\lfloor \log_p(\lambda)\rfloor} \mid \det\left(V_1R\right)\right)$ is $\frac{3}{p^{\lfloor \log_p(\lambda)\rfloor}}$ by Thm. 5.4.5. We have

$$\frac{3}{p^{\lfloor \log_p(\lambda)\rfloor}} \leq 3\min(\frac{1}{p}, \frac{1}{p^{\log_p(\lambda)-1}}) \leq 3\min(\frac{1}{p}, \frac{p}{\lambda})$$

This is less than $\frac{3}{\sqrt{\lambda}}$ as well, which leads to the final result.

By symmetry, the dual of the theorem for a left-hand size multiplicative preconditioner L is also true. By Thm. 7.1.2 we obtain [127, Lem.5.17] for free.

Lemma 7.1.3 (Lemma 5.17 of [127]) *Let $n, m, k \in \mathbb{N}$, $n, m \geq k > 0$. Let A be a $n \times m$ matrix. Let k be such that the kth invariant factor of A, s_k, is non-zero. Let p be a prime. Then there exists a full column ranked matrix M, such that for any $i \times n$ integer matrix L*

$$p \nmid \frac{s_k(LA)}{s_k} \text{ if } \det(LM) \neq 0 (\mod p).$$

PROOF Let us consider the dual of Thm. 7.1.2 for $s = 1$. Let V_1 be the matrix given by the theorem and let us set $M = V_1$. We have the following statement.

$$If \ \mathrm{ord}_p\left(\frac{\mu_k(LA)}{\mu_k(A)}\right) \geq 1 \ then \ \mathrm{ord}_p(\det(LM)) \geq 1.$$

By transposition we have

$$If \ \mathrm{ord}_p(\det(LM)) = 0 \ then \ \mathrm{ord}_p\left(\frac{\mu_k(LA)}{\mu_k(A)}\right) = 0.$$

$\mu_k(LA)$ (resp. $\mu_k(A)$) is a product of $s_1 \cdots s_k(LA)$ (resp. $s_1 \cdots s_k(A)$). By Lem. 7.1.1, $s_i(LA)$ divides $s_i(A)$ for $i = 1, \ldots, k$. Thus, also $\mathrm{ord}_p\left(\frac{s_k(LA)}{s_k(A)}\right) = 0$, which finishes the proof.

7.2 Extended Bonus Idea

Let $n, k \in \mathbb{N}, n, k > 0$, and let A be a $n \times n$ invertible matrix. Let X be a (matrix) rational solution of the equation $AX = B$, where $B = [b_i], i = 1, \ldots, k$, is a random $n \times k$ matrix. Then the coordinates of X have a common denominator $\tilde{s}_n$ and we let $N = [n_i], i = 1, \ldots, k$, denote the matrix of numerators of X. Thus, $X = \tilde{s}_n^{-1} N$ and $\gcd(N, \tilde{s}_n) = 1$. Suppose that the Smith form of A is equal to $S = \operatorname{diag}(s_1, \ldots, s_n)$.

Following Wan, we notice that $s_n A^{-1}$ is an integer matrix, the Smith form of which is equal to

$$diag\left(\frac{s_n}{s_n}, \frac{s_n}{s_{n-1}}, \ldots, \frac{s_n}{s_1}\right). \tag{7.1}$$

Therefore, we may compute s_{n-k+1} when knowing $s_k\left(s_n A^{-1}\right)$. The trick is that the computation of A^{-1} is not required: we can perturb A^{-1} by right multiplying it by B. Then, $s_k\left(s_n A^{-1} B\right)$ is a multiple of $s_k\left(s_n A^{-1}\right)$ by Lem. 7.1.1. Instead of $M = s_n A^{-1} B$ we would prefer to use $\tilde{s}_n A^{-1} B$ which is already computed and equal to N. The relation between A and N is as follows.

Lemma 7.2.1 (Smith Form of N vs. Smith Form of A) *Let $n, k \in \mathbb{N}, n, k > 0$, and let A be a $n \times n$ invertible matrix. Suppose that the Smith form of A is equal to $S = \operatorname{diag}(s_1, \ldots, s_n)$. Let X be a (matrix) rational solution of the equation $AX = B$, where $B = [b_i], i = 1, \ldots, k$, is a $n \times k$ matrix. Suppose that $X = \tilde{s}_n^{-1} N$, where $\gcd(\tilde{s}_n, N) = 1$. Let L be $k \times n$ matrix. Then*

$$\left.\frac{\tilde{s}_n}{\gcd\left(s_i\left(N\right), \tilde{s}_n\right)}\right| s_{n-i+1} \text{ and } \left.\frac{\tilde{s}_n}{\gcd\left(s_i\left(LN\right), \tilde{s}_n\right)}\right| s_{n-i+1}, i = 1 \ldots, k.$$

PROOF The Smith forms of $s_n A^{-1} B$ and N are connected by the relation

$$\frac{s_n}{\tilde{s}_n} s_i\left(N\right) = s_i\left(s_n A^{-1} B\right),$$

$i = 1, \ldots, k$. Moreover, $s_i\left(N\right)$ is a factor of $s_i\left(LN\right)$. We notice that $\frac{s_n}{s_i(s_n A^{-1} B)}$ equals $\frac{\tilde{s}_n}{s_i(N)}$, and thus, by Eq. (7.1). $\frac{\tilde{s}_n}{\gcd(s_i(LN), \tilde{s}_n)}$ is an (integer) factor of s_{n-i+1}.

Remark 7.2.2 Taking $\gcd(s_i(LN), \tilde{s}_n)$ and $\gcd(s_i(N), \tilde{s}_n)$ is necessary as $\frac{\tilde{s}_n}{s_i(LN)}$, $\frac{\tilde{s}_n}{s_i(N)}$ may be rational numbers.

7.3 Last k Invariant Factors

Let $n \in \mathbb{N}, n > 0$ and let A be a $n \times n$ matrix. In fact we are interested in computing the product $\pi_k = s_n s_{n-1} \cdots s_{n-k+1}$ of the k biggest invariant factors of A rather than the factors itselves. Then, following the idea of [2], we would like to reduce the computation

of the determinant to the computation of $\frac{\det(A)}{\tilde{\pi}_k}$, where $\tilde{\pi}_k$ is a factor of π_k that we have obtained. Therefore in our application it is essential that $\tilde{\pi}_k$ is a divisor of π_k.

Thanks to Lem. 7.2.1 we can propose the following procedure to approximate π_k with fulfills this requirement.

Algorithm 7.3.1 k-Bonus Algorithm

Require: $n \times n$ integer matrix A, $k \in \mathbb{N}$,
Require: Parameter $\beta \in \mathbb{N}$, $\beta > 1$,
Require: A number of iterations $r \in \mathbb{N}$, $r > 0$,
Require: A stream S_β of random integers uniformly chosen from the set $\{-\lceil \frac{\beta}{2} \rceil + 1, \ldots, 0, \ldots, \lfloor \frac{\beta}{2} \rfloor\}$,
Ensure: $\tilde{\pi}_k$, a factor of $\pi_k = s_n s_{n-1} \cdots s_{n-k+1}$;

```
for i = 1 to r do
    s̃_n = 1;
    Create a random n × k matrix B_i from stream S_β;
    Solve AX = B_i for a rational matrix X_i, represent X_i = (1/s̃_n(i)) N_i;
    s̃_n = lcm(s̃_n, s̃_n(i));
    Create a random k × n matrix L_i;
    Compute D_i = det(L_i N_i);
end for
Compute π̃_k(i) = (s̃_n^k / gcd(s̃_n^k, D_i)), i = 1, ..., r;
Return π̃_k = lcm_{i=1..r}(π̃_k(i));
```

7.4 Expected Overestimation

Let us consider Alg. 7.3.1. In the following lemmas we show that by repeating the choice of random matrices B_i and L_i twice (i.e. for $r = 2$), we will omit only a finite number of bits in π_k. Let us start with a preliminary result.

Lemma 7.4.1 (Expected Underestimation of π_k) *Let $n, k \in \mathbb{N}, n > 0, k > 1$, let A be an $n \times n$ integer matrix, H its Hadamard's bound, and $\beta > 1$. Let S_β be a stream of random integers uniformly chosen from the set $\{-\lceil \frac{\beta}{2} \rceil + 1, \ldots, 0, \ldots, \lfloor \frac{\beta}{2} \rfloor\}$. Let $\mathcal{P}$ denote the distribution of $n \times k$ and $k \times n$ matrices, whose entries are sampled from the stream S_β. Let B_i and R_i be $n \times k$ (resp. $k \times n$) matrices given by distribution $\mathcal{P}$. Suppose that the Smith form of A is equal to $S = \mathrm{diag}(s_1, \ldots, s_n)$. Denote by μ_k the product $s_1 \ldots s_k$ of the k smallest invariant factors and by π_k the product of the k biggest factors of a matrix. Then for $M = s_n A^{-1}$*

$$\mathbf{E}\left(\log\left(\frac{\pi_k(A)}{s_n^k}\gcd(\mu_k(L_1 M B_1), \mu_k(L_2 M B_2), s_n^k)\right)\right) \in \; O(1) + O(\frac{k^3 \log^4(H)}{\beta}),$$

where the expected value is computed with respect to distribution $\mathcal{P}$.

PROOF First, notice that $\frac{\pi_k(A)}{s_n^k} = \frac{1}{\mu_k(M)}$. Therefore the expected value is less than or equal

$$\begin{aligned}
&\mathbf{E}\left(\log\left(\frac{\pi_k(A)}{s_n^k}\gcd\left(\mu_k(L_1MB_1),\mu_k(L_2MB_2),s_n^k\right)\right)\right)\\
&=\sum_{p\ prime}\sum_{s=1}^{\infty}\log(p)\,s\mathcal{P}\left(\mathrm{ord}_p\left(\frac{\gcd\left(\mu_k(L_1MB_1),\mu_k(L_2MB_2),s_n^k\right)}{\mu_k(M)}\right)=s\right)\\
&=\sum_{p\ prime}\sum_{s=1}^{\infty}\log(p)\mathcal{P}\left(\mathrm{ord}_p\left(\frac{\gcd\left(\mu_k(L_1MB_1),\mu_k(L_2MB_2),s_n^k\right)}{\mu_k(M)}\right)\geq s\right)\\
&\leq\sum_{p\ prime,p|s_n}\sum_{s=1}^{k\log(s_n)}\log(p)\Pi_{i=1,2}\mathcal{P}\left(\mathrm{ord}_p\left(\frac{\mu_k(L_iMB_i)}{\mu_k(M)}\right)\geq s\right)\\
&\leq\sum_{p|s_n}\sum_{s=1}^{k\log(s_n)}\log(p)\Pi_{i=1,2}\left(\sum_{j=0}^{s}\mathcal{P}\left(\begin{array}{c}\mathrm{ord}_p\left(\frac{\mu_k(MB_i)}{\mu_k(M)}\right)\geq j\wedge\\ \mathrm{ord}_p\left(\frac{\mu_k(L_iMB_i)}{\mu_k(MB_i)}\right)\geq(s-j)\end{array}\right)\right).
\end{aligned}$$

Thanks to Thm. 7.1.2 we can estimate this probability for every prime p and power s. We only consider $p|s_n$ and powers $p^s < s_n^k$. We will divide the sum on two disjoint parts, for p, s such that $p^s < \beta$ and $p^s \geq \beta$.

For $p^s < \beta$ we have

$$\begin{aligned}
&\sum_{j=0}^{s}\mathcal{P}\left(\mathrm{ord}_p\left(\frac{\mu_j(MB_i)}{\mu_j(M)}\right)\geq j\wedge\mathrm{ord}_p\left(\frac{\mu_j(L_iMB_i)}{\mu_j(MB_i}\right)\geq(s-j)\right)\leq\\
&\sum_{j=0}^{s}\mathcal{P}\left(B_i:\mathrm{ord}_p\left(\det\left(VB_i\right)\right)\geq j\right)\mathcal{P}\left(L_i:\mathrm{ord}_p\left(\det\left(L_iU\right)\right)\geq j\right)\leq(s+1)\frac{3}{p^s}.
\end{aligned}$$

Now the expected size of the under-estimation is less than or equal to

$$\begin{aligned}
&\log(2)\left(3+\sum_{s=4}^{\infty}\left((s+1)^2\frac{3}{2^s}\right)^2\right)+\log(3)\left(2+\sum_{s=3}^{\infty}\left((s+1)\frac{3}{3^s}\right)^2\right)\\
&+\log(5)\left(1+\sum_{s=2}^{\infty}\left((s+1)^2\frac{3}{5^s}\right)^2\right)+\log(7)\left(\sum_{s=2}^{\infty}\left((s+1)^2\frac{3}{7^s}\right)^2\right)\\
&+\sum_{10<p\leq H}\sum_{s=1}^{\infty}\log(p)\left(\frac{3(s+1)}{p^s}\right)^2\leq{}^{1}4.36+2.24+1.14+0.77\\
&+\sum_{10<p\leq H}\log(p)\frac{-27p^2+36p^4+9}{(p-1)^3(p+1)^3}\leq 8.51\\
&+\int_{10}^{\infty}\log(x)\frac{-27x^2+36x^4+9}{(x-1)^3(x+1)^3}dx\leq 8.51+11.97\leq 21
\end{aligned}$$

which is $O(1)$. We use the estimation $\sum_{s=1}^{\infty}\left(\frac{3(s+1)}{p^s}\right)^2 = \frac{-27p^2+36p^4+9}{(p-1)^3(p+1)^3}$.

For $p^s \geq \beta$ the expected size of the underestimation is

$$\sum_{p|s_n}\sum_{s=\lceil\log_p(\beta)\rceil}^{k\log_p(H)}(s+1)^2\log(p)\left(\frac{3}{\sqrt{\beta}}\right)^2 \leq {}^2\sum_{p|s_n}\frac{9\log(p)}{\beta}\Big(\frac{13}{6}k\log_p(H)+\frac{3}{2}\left(k\log_p(H)\right)^2$$
$$+\frac{1}{3}\left(k\log_p(H)\right)^3\Big) \leq {}^3k\log^2(H)\frac{9}{\beta}\left(\frac{13}{6}+\frac{3}{2}k\log(H)+\frac{1}{3}k^2\log^2(H)\right) \leq \frac{36k^3\log^4(H)}{\beta}.$$

We use the estimation $\sum_{i=1}^{N}(s+1) = \frac{13}{6}N+\frac{3}{2}N^2+\frac{1}{3}N^3$. This is $O\left(\frac{k^3\log^4(H)}{\beta}\right)$, which gives the result.

7.5 Correctness of k-Bonus Algorithm

Theorem 7.5.1 (Expected Underestimation in Alg. 7.3.1) *Let $n, k \in \mathbb{N}, n > 0, k > 1$, let A be an $n \times n$ integer matrix, and H its Hadamard's bound. Let $r = 2$ and let and $\beta = 36k^3\log^4(H)$. Let S_β be a stream of random integers uniformly chosen from the set $\{-\lceil\frac{\beta}{2}\rceil+1,\ldots,0,\ldots,\lfloor\frac{\beta}{2}\rfloor\}$. Let $\mathcal{P}$ denote the distribution of $n\times k$ and $k\times n$ matrices, whose entries are sampled from the stream S_β.*

Let A, k, β, r, S_β be the input of Alg. 7.3.1. Then $\tilde{\pi}_k(i) \mid \tilde{\pi}_k$ and divides π_k. π_k is the product of k biggest invariant factors of A and $\tilde{\pi}_k(i), \tilde{\pi}_k$ are the (partial) results of the algorithm at step $i = 1,\ldots r$. Moreover, the algorithm outputs $\tilde{\pi}_k$ such that the expected value of $\log(\frac{\pi_k}{\tilde{\pi}_k})$ is $O(k)$.

PROOF For $i = 1, 2$ let L_i, N_i be defined as in the algorithm. By Lem. 7.2.1 we have

$$\frac{\tilde{s}_n}{\gcd\left(s_j\left(L_iN_i\right),\tilde{s}_n\right)}\Bigg| s_{n-j+1}, j = 1..k.$$

Therefore

$$\frac{\tilde{s}_n^k}{\pi_{j=1}^k\gcd\left(s_j\left(L_iN_i\right),\tilde{s}_n\right)}\Bigg|\pi_k$$

We have that $\pi_{j=1}^k\gcd\left(s_j\left(L_iN_i\right),\tilde{s}_n\right)$ divides $\gcd(\pi_k(L_iN_i),\tilde{s}_n^k) = \gcd(\det(L_iN_i),\tilde{s}_n^k)$ and therefore $\frac{\tilde{s}_n^k}{\gcd(\det(L_iN_i),\tilde{s}_n^k)}$ divides π_k. Therefore $\tilde{\pi}_k(i)$ divides π_k and as a consequence $\tilde{\pi}_k = \mathrm{lcm}(\tilde{\pi}_k(1),\tilde{\pi}_k(2))$ divides π_k as well.

Let us now consider the expected overestimation. Notice that $N_i = MB_i$. By Lem. 7.4.1

$$\mathbf{E}\left(\log\left(\frac{\pi_k}{s_n^k}\gcd\left(\det(L_1N_1),\det(L_2N_2),s_n^k\right)\right)\right) \in O(1).$$

We have

$$\begin{aligned}\mathbf{E}(\log(\frac{\pi_k}{\tilde{\pi}_k})) &= \mathbf{E}(\log(\frac{\pi_k}{\mathrm{lcm}(\tilde{\pi}_k(1),\tilde{\pi}_k(2))})) = \mathbf{E}(\log(\frac{\pi_k}{\mathrm{lcm}(\frac{\tilde{s}_n^k}{\gcd(\tilde{s}_n^k,\det(L_1N_1))},\frac{\tilde{s}_n^k}{\gcd(\tilde{s}_n^k,\det(L_2N_2))})})) \\ &\leq \mathbf{E}(\log(\frac{\pi_k}{\tilde{s}_n^k}\gcd(\tilde{s}_n^k,\det(L_1N_1),\det(L_2N_2)))) \\ &\leq \mathbf{E}(\log(\frac{\pi_k}{s_n^k}\gcd(s_n^k,\det(L_1N_1),\det(L_2N_2)))) + k\mathbf{E}(\frac{s_n}{\tilde{s}_n})\end{aligned}$$

The choice of β and r ensures that $\mathbf{E}(\frac{s_n}{\tilde{s}_n}) \leq 5 \in O(1)$, see 6.3.2(v) Therefore we may conclude that

$$\mathbf{E}(\log(\frac{\pi_k}{\tilde{\pi}_k})) \leq 21 + 5k \in O(k).$$

Remark 7.5.2 Another method to compute the product μ_k of k first invariant factors of a rectangular matrix N would be to compute several minors of the matrix and to take the gcd of them. However, it is difficult to judge the impact of choosing only a few minors (instead of all) in the gcd computation. Indeed, no result is known on the distribution of entries of N. An experimental evaluation whether for random A and random B the minors of N are sufficiently "randomly" distributed remains to be done. The advantage of this method would be to get rid of random matrix L and the overestimation it may cause.

7.5.1 Cost of Bonus Computation

Let us now estimate k for which the use of term "bonus" is justified, i.e. the cost of computing $\pi_k(i)$ once $\tilde{s}_n$ is known is considerably smaller than the cost of one solving.

Lemma 7.5.3 (Cost of Alg. 7.3.1) *Let $n, k \in \mathbb{N}, n > 0, k > 1$, let A be an $n \times n$ integer matrix, and H its Hadamard's bound. Let $r = 2$ and let and $\beta = 36k^3 \log^4(H)$. Let S_β be a stream of random integers uniformly chosen from the set $\{-\lceil\frac{\beta}{2}\rceil + 1, \ldots, 0, \ldots, \lfloor\frac{\beta}{2}\rfloor\}$. Let $\mathcal{P}$ denote the distribution of $n \times k$ and $k \times n$ matrices, whose entries are sampled from the stream S_β.*

Let A, k, β, r, S_β be the input of Alg. 7.3.1. Let us assume that the cost of Solver procedure is more than $C_1 n^{\omega'} M(n \log(n\|A\|))$, where ω' is non-negative constant and $\omega' \geq \omega$ and $M(x)$ is the cost of multiplication of x bit integers. Moreover, let us assume that Determinant is used to compute $\det(L_iN_i)$ in Alg. k-Bonus and that the cost of this algorithm is less than $C_2 k^3 n \log(n\|A\|) M(n \log(n\|A\|))$.

Let us assume that

$$k \leq \sqrt[3]{\frac{C_1}{C_2}} \frac{n^{\frac{\omega'-2}{3}}}{\sqrt[3]{\log(n\|A\|)}}. \tag{7.2}$$

The cost Alg. k-Bonus is $k+1$ times the cost of Solver .

PROOF For $k \leq \sqrt[3]{\frac{C_1}{C_2}} \frac{n^{\frac{\omega'-2}{3}}}{\sqrt[3]{\log(n\|A\|)}}$ the cost of CR algorithm is less than or equal

$$C_2 \frac{C_1}{C_2} \frac{n^{\omega'-2}}{\log(n\|A\|)} n \log(n\|A\|) M(n \log(n\|A\|)) \leq C_1 n^{\omega'-1} M(n \log(n\|A\|)),$$

which, by assumption, is the cost of one *Solver*. Therefore the cost of the algorithm is at most that of $k+1$ solving by *Solver*.

Remark 7.5.4 Let us now explained where did the bound on k came from. Let us assume that A is a $n \times n$ matrix with entries bounded in absolute value be $\|A\|$. Suppose that $1 < k \leq n$, and k-Bonus, see Alg. 7.3.1, is to be computed and Let us adopt its notations.

Let us assume cost of one solving is more than $C_1 n^{\omega'} M(n \log(n\|A\|))$. The values ω' is 3 for p-adic lifting and ω for [122]. As $M(n \log(n\|A\|)) \leq B(n \log(n\|A\|)) \leq (n \log(n\|A\|))^2$, the bound also holds for the algorithm of [122] if fast multiplication $M(x) = O^{\approx}(x \log(x))$ is used.

The cost of "bonus" computation is the cost of matrix multiplication $L_i N_i$ plus the cost of determinant computation for matrix $L_i N_i$. The cost of certified CR Algorithm to compute $\det(L_i N_i)$ is $O(k^2 n M(\log(k\|L_i N_i\|)) + k \log(k\|L_i N_i\|)(k^\omega + k^2 M(\log(\|L_i N_i\|)))$, which includes the cost of matrix multiplication and the cost of taking the modular image.

Let us now estimate $\|L_i N_i\|$. Let us notice that $\|L_i\| \leq \frac{\beta}{2}$, $N_i = \tilde{s}_n(i) A^{-1} B_i$, where $\|N_i\| \leq \|s_n A^{-1} B_i\|$ and $\|B_i\| \leq \frac{\beta}{2}$. Therefore $\|N_i\| \leq \frac{nH\beta}{2}$ and $\|L_i N_i\| \leq \frac{n^2 H \beta^2}{4}$, which implies that $\log(\|L_i N_i\|)) = O(n \log(n\|A\|))$. As $k \leq n$ and $\omega \leq 3$, we see that the cost of imaging prevails in each CRA iteration and the complexity of the CRA is $O(k^3 n \log(n\|A\|) M(n \log(n\|A\|))) \leq C_2 k^3 n \log(n\|A\|) M(n \log(n\|A\|))$ for some $C_2 > 0$. By comparing the complexities we get the bound on k.

Remark 7.5.5 Both C_1 and C_2 can roughly by approximated when the procedures *Solver* and *Determinant* are run. Clearly, constants C_1 and C_2 are not important from the point of view of the asymptotic complexity of the algorithm.

8

Adaptive Determinant Algorithm

In this section we would like to introduce an adaptive algorithm for the computation of determinant of a dense matrix. The algorithm is dynamic, baroque, and introspective in the sense of the classification of Sec. 1.4.3.

8.1 General Idea and Components of the Algorithm

The algorithm combines the ideas of the following algorithms

- the classic CRA algorithm with or without early termination, see [70, Sec.5.4,5.5];
- the idea of Abbott et al.[2] of using preconditioned ET CRA with early termination, the idea of using s_n and LIF algorithm to get the preconditioner, see Ch. 6;
- the idea of Eberly et al. [44] to compute the determinant as a product of invariant factors and to estimate the expected number of those, see [44];
- the idea of Saunders and Wan [112, 127, 113] to compute a "bonus" factor by additional system solvings, developed by us to the computation of k last factors by k system solvings in kBonus procedure, see Ch. 7;
- any algorithm of [78, 122] or other, which can be chosen to complete the algorithm;

The algorithm is a version of an adaptive preconditioned CRA loop (cf. Alg. 6.2.1, Abbott's algorithm [2]), in which the preconditioner can change during execution. In the adaptive part of the algorithm we repeatedly run several CRA steps and approximate the product of biggest invariant factors of A (denoted π_k for k largest factors). Then the preconditioned CRA loop is recomputed with the new bigger preconditioner at low cost. Moreover, based on certain properties of the output that are detected on the run-time, the algorithm may choose to stop updating the preconditioner, therefore emphasizing the CRA part. We suppose that the product of at most r_{max} factors can be computed (by r_{max}-Bonus algorithm), where r_{max} is chosen in a way to control the total probability of the adaptive part of the algorithm.

We prove that in the average case, the preconditioned CRA loop early terminates before r_{max} is achieved, which results in a good expected case complexity of the algorithm. Moreover, we are able to detect cases when CRA loop may take to long to terminate and turn to the asymptotically better algorithm (e.g. [2, 44, 78, 122] or any other) in this case, thus keeping the worst case complexity at bay.

The algorithm is *dynamic*, as the top level procedures do not depend on the architecture. However, we allow for the lower level subroutines, e.g. FFLAS matrix multiplication or integer multiplication to depend on it.

The algorithm is *adaptive*, as the switches between the running the CRA and updating the preconditioner (cf. the LIF alg. 6.3.1 and k-Bonus alg. 7.3.1) are computed at the runtime. It is *introspective*, as the timing comparison between the two parts is taken into account while deciding between k-Bonus computation and running CRA iterations.

The algorithm is *baroque* as there are possible $O(r_{max})$ switches, where r_{max} is the bound on the number of invariant factors as a function of matrix size n. In order to obtain good expected complexity, we require that r_{max} is greater than $O(\sqrt{\log(n)})$ i.e. greater the expected number of non-trivial invariant factors for a random dense matrix, see Thm 5.3.10. Moreover, the choice of r_{max} have to ensure, that the adaptive part of the algorithm does not exceed the worst case complexity bound imposed by the last switch (worst case) algorithm.

8.2 Requirement of the Algorithm

The algorithm inherits its requirements from the components. As input, it requires

- $n \times n$ integer matrix A,
- $0 < \epsilon < 1$, the error tolerance of the algorithm.

As a variant of early terminated preconditioned CRA loop it requires

- $H = (\sqrt{n}\|A\|)^n$, which is a bound for the determinant
- $l \in \mathbb{N}, l > 1$, P a set of primes, such that $|P| > 2\log_l(H)$ and $l < p < 2l, \quad \forall p \in P$, primes are randomly and uniformly sampled from P;
- functions $iteration, reconstruct, terminated$ as defined in Sec. 6.1
- $k_{max} = \lceil \frac{\log(1/\epsilon)}{\log(P') - \log(\lceil \log_l(H) \rceil)} \rceil$, where $P' = |P| - \lfloor \log_l(H) \rfloor$, see Eq. (6.5). This is the maximum bound on the number of iterations needed to early terminate

For the computation of the preconditioner by LIF and k-Bonus algorithms, it requires

- integer $r_{min} > 0$, which denotes the minimal number of repetition needed to ensure the computation of $\tilde{\pi}_k \approx \pi_k$ with sufficient probability; the choice $r_{min} = 2$ suffices in our case, see Thm. 7.5.1;

- integer $\beta > 1$, such that entries of random matrices are uniformly sampled from the set $\{-\lceil\frac{\beta}{2}\rceil + 1, \ldots, 0, \ldots, \lfloor\frac{\beta}{2}\rfloor\}$; the choice $\beta = 36r_{max}^3 \log^4(H)$ suffices in our case, see Thm. 7.5.1;
- Function $Solver(x, A, b_i)$ to solve equation $Ax = b_i$; one may choose between variants of p-adic lifting [27, 95] and fast solver of [122].
- Function $Determinant(0, M)$ to compute the determinant of a $i \times i$ matrix M, $i \leq r_{max}$ with certainty; if r_{max} is sufficiently small, classic CRA algorithm may be used, see Lem. 7.5.3.

In order to finish the algorithm, an implementation of $Determinant(\epsilon, A)$ has to be available. The particular implementation may take into account the partial results obtained so far.

During the course of the algorithm we separate the $r_{min} \times r_{max}$ solving of r_{max}-Bonus computation by several CRA steps. Therefore we introduce variables $k_{app}(i)$, $i = 1..r_{min}$, which store the number of solvings computed so far in the ith iteration of r_{max}-Bonus algorithm.

8.2.1 Evaluation of r_{max} - the Number of Iterations

Let us finish this section by the evaluation of r_{max}. In order to make the algorithm applicable to a large class of generic matrices, we will require that r_{max} is greater than the expected number of non-trivial invariant factors of a random dense matrix. Precisely, we consider dense $n \times n$ matrices here, with entries chosen randomly and uniformly from the set of consecutive integers $\{-\lceil\frac{\lambda}{2}\rceil + 1, \ldots, 0, \ldots, \lfloor\frac{\lambda}{2}\rfloor\}$ for which the expected number of invariant factors is bounded by $\max(2, \left\lceil\sqrt{2\log_\lambda(n)}\right\rceil)+3$ by Thm. 5.3.10. In no additional information on the distribution of input is provided, this prompts us anyway to set

$$r_{max} \geq \left\lceil\sqrt{2\log_{2\|A\|}(n)}\right\rceil + 3.$$

To keep the cost of bonus computation at bay, we decide to take

$$r_{max} \in O\left(\frac{n^{\frac{\omega'-2}{3}}}{\sqrt[3]{\log(n\|A\|)}}\right),$$

where ω' is the exponent of n in the complexity of $Solver$, see Lem. 7.5.3. This choice ensures that the product $\tilde{\pi}_k$ is updated at the cost of 2 $Solvers$ at each iteration, see the complexity section Sec. 7.5.1 for details.

Still, we need to compare the cost of the adaptive part, which takes at most the time of $O(r_{min} \times r_{max})$ $Solver$s with the cost of the last step algorithm. In fact, the worse the complexity of the last step $Determinant(\epsilon, A)$ algorithm, the bigger bound on r_{max} we

may take. In the case of [122] algorithm, the *Solver* and *Determinant* differ only by a $O(\log(n))$ factor, which is the smallest possible value for the algorithms currently in use. Therefore, $O(\log(n))$ *Solver*s may be run in the adaptive part.

We propose to set the value of r_{max} to

$$r_{max} = \max(\log(n), \left\lceil \sqrt{\log_{2\|A\|}(n)} \right\rceil + 3).$$

which allows to keep the asymptotic complexity of the adaptive part below the complexity of the worst case algorithm. This value can be increased in the case when slower *Determinant* algorithm is used in the worst-case part. This requires a comparison (asymptotic or exact) between complexities of the *Determinant* algorithm and the *Solver*.

8.2.2 Summary - Parameters of the Algorithm

To summarize, let us recall all parameters of the algorithm.

- $n \in \mathbb{N}$ - the size of the matrix,
- $A \in \mathbb{Z}^{n\times n}$ - the integer matrix,
- $\|A\| = \max_{i,j=1..n}(A_{ij})$ - the maximum norm of the matrix,
- $0 < \epsilon < 1$ - probability of correctness of the algorithm,
- $H = (\sqrt{n}\|A\|)^n$, - the bound for the determinant of the matrix,
- $l \in \mathbb{N}$ - the size of primes used in the CRA part,
- P - a set of primes used in the CRA part,
- $k_{max} = \lceil \frac{\log(1/\epsilon)}{\log(P') - \log(\lceil \log_l(H) \rceil)} \rceil$, - the threshold for early termination in CRA part,
- $\beta = 36 r_{max}^3 \log^4(H)$ - the bound on random matrix/vector entries in k-Bonus part,
- $k_{app}(i) = 0, i = 0, 1..r_{min}$ - the current number of invariant factors computed in each repetition,
- $r_{min} = 2$ - the number of iterations in k-Bonus part,
- $r_{max} = \max(\log(n), \left\lceil \sqrt{2\log_\lambda(n)} \right\rceil + 3)$ - the maximum number of invariant factors to compute,

The algorithm will use the following subroutines.

- $iteration(A, p)$ - to compute the modular determinant;
- $reconstruct(x_t, y_0, \ldots, y_t, p_0, \ldots, p_t)$ - to perform reconstruction based on Chinese Remaindering Theorem, see Eq. (6.1);

- $terminated(\epsilon, P, H)$ - to check for early termination in the CRA part, see Alg. 6.2.1;
- $Solver(x, A, b)$ - to solve $Ax = b$ over rational field, see Sec. 6.3.5;
- $Determinant(\epsilon, A)$ - one of existing algorithms to compute the determinant with probability at least $1 - \epsilon$, see Sec. 4.1

8.3 Main Parts of Algorithm

The algorithm is divided in three main parts.

1. The CRA part consist of several repeated CRA loop steps i.e. given t_{min} and t_{max} and given the preconditioner D we perform operations as in Alg. 8.3.1.

Algorithm 8.3.1 CRA part of Adaptive Determinant Algorithm 8.4.1

```
for t = t_min to t_max − 1 do
    if d_t not stored then
        Choose uniformly and randomly a prime p_t from the set P;
        Run iteration(d_t, p_t) to get d_t = det(A) mod p_t; store d_t;
    y_t = d_t/D mod p_t
    Run reconstruct(x_t, y_0, ..., y_t, p_0, ..., p_t)
    if terminated(ε, H) then
        Return D · x_t;
end for
```

The CRA part is performed every time D is updated or the bounds t_{min}, t_{max} are changed. t_{max} may not be known in advance, it may be computed based on partial timings of the algorithm.

2. The LIF&Bonus part consists of solving one system of equations of a form $Ax = b$, where b is a random vector; at the ith execution of LIF&Bonus we use the last results together with the results of previous $i - 1$ calls to LIF&Bonus to compute an approximation $\tilde{\pi}_i$ of the product π_i of i biggest invariant factors. Our approximation is always a divisor of π_i; in order to obtain $\tilde{\pi}_i$ sufficiently close to π_i, we have to repeat i-Bonus scheme r_{mix} times and take $\tilde{\pi}_i = \mathrm{lcm}_{i=1..r_{min}}(\tilde{\pi}_i(j))$, where $\tilde{\pi}_i(j)$ is the result of jth repetition of the procedure, $j = 1..r_{min}$.

   Let us assume that $i, j \in \mathbb{N}, i, j > 0$ and let us consider the ith call to LIF&Bonus in the jth repetition of the procedure, assume that the results of previous $i - 1$ calls of the jth procedure are $x_1(j), \ldots, x_{i-1}(j)$. The operations are shown in Alg. 8.3.2.

   The LIF&Bonus part is executed at most r_{max} times and intertwines with the CRA part. The aim of this part is to update the preconditioner D needed by the CRA part, by putting $D \approx \pi_i$ (see Alg. 8.4.1 for details). If D is not changed by the procedure it might be either due to a possible underestimation of the product $\tilde{\pi}_i$ or to the fact that the ith invariant factor s_{n-i+1} is trivial i.e. equal to 1. To rule out a possible

Algorithm 8.3.2 LIF&Bonus Part of Adaptive Determinant Algorithm 8.4.1

1: Generate b_i a random vector of dimension n from the stream $\mathcal{S}_\beta$;
2: Run $Solver(x_i(j), A, b_i)$ to solve $Ax_i(j) = b_i$;
3: Update $\tilde{s}_n(i) = \mathrm{lcm}(\tilde{s}_n(i), \mathrm{den}(x_i(j)))$, where $\mathrm{den}(x_i(j))$ is the common denominator of the entries of $x_i(j)$;
4: Set $N_k = \tilde{s}_n(i)x_i(j), k = 1..i,\ N = [N_k]_{k=1..i}$
5: Generate $i \times n$ matrix L from the stream $\mathcal{S}_\beta$;
6: $d = Determinant(0, LN)$ to get $\det(LN)$ with certainty;
7: **Return** $\tilde{\pi}_i(j) = \frac{\tilde{s}_n(i)^k}{\gcd(d, \tilde{s}_n(i)^k)}$.

underestimation, one need to ensure that the whole scheme is repeated at least r_{min} times before a conclusion is made.

3. The Worst Case part consists of running a determinant algorithm $Determinant(\epsilon, A)$ chosen by the user. The complexity of *Determinant* will determines the worst case complexity of the algorithm. This is consider as the exceptional case or "failure" of the adaptive scheme.

8.4 Procedure

Formally, the adaptive determinant algorithm is as follows. The requirement and parameters of the algorithm are given in Sec. 8.2.

8.5 Correctness of Algorithm

Theorem 8.5.1 (Preconditioning in Alg. 8.4.1) *During the course of Alg. 8.4.1, the preconditioner D is always a divisor of the determinant* $\det(A)$.

PROOF Let us first notice, that if $\det(A) = 0$ then the algorithm may stop after first k_{max} iterations by early termination principle. Therefore any system solving is performed on a non-singular matrix.

Preconditioner D is updated at each iteration by the LIF&Bonus part. All iterations of LIF&Bonus part put together are equivalent to the k-Bonus algorithm, therefore Thm. 7.5.1 may be applied. For iterations i, j, $i = 1..r_{min}, j = 1..r_{max}$ in the LIF&Bonus part, $\tilde{\pi}_j(i)$ is computed. Therefore we may conclude that D is a divisor of the determinant.

Theorem 8.5.2 (Correctness of Alg. 8.4.1) *Algorithm 8.4.1 correctly computes the determinant with probability* $1 - \epsilon$.

PROOF Termination is possible only by the early terminated CRA loop, see Alg. 8.3.1, or by the determinant algorithm used in the last step. The choice of k_{max} from Lem. 6.1.1(2ii) and the choice of the worst case determinant algorithm ensures that $1 - \epsilon$ probability is obtained.

Algorithm 8.4.1 Introspective Determinant Algorithm

Ensure: $\det(A)$ correct with probability at least $1-\epsilon$.

```
t_max = k_max + 1;
Run CRA part for t = 0..t_max − 1;
Compute the time t_1 of one step of CRA;
repeat
  for j = 1 to r_min do
    for i = k_app(j) + 1 to r_max do
      Run LIF&Bonus part for π̃_i(j);
      Compute the time t_2 of previous call;
      k_iter = max(⌈5 / (r_min log(l))⌉, t_2/t_1);
      if i = 1 then k_iter = k_iter + ⌈21 / (r_min log(l))⌉;
      π̃_i = lcm(π̃_i, π̃_i(j));
      D = lcm(D, π̃_i);
      if π̃_i = π̃_{i−1} and k_app(j) ≥ k_app(j − 1) then
        Run new iterations of CRA for t = t_max..t_max + k_iter − 1 iterations, update
        t_max = t_max + k_iter;
        k_app(j) = i;
        if j = r_min then
          k_app(j) = r_max for all j = 1..r_min;
          k_iter = (r_max − i) × t_2/t_1;
          Run new iterations of CRA for t = t_max..t_max + k_iter − 1 iterations, update
          t_max = t_max + k_iter; ;
        break;
      else
        Rerun CRA for t = 0..t_max − 1;
        Run new iterations of CRA for t = t_max..t_max + k_iter − 1 iterations, update
        t_max = t_max + k_iter;
    end for
  end for
until k_app(j) = r_max for all j = 1..r_min
Return Determinant(ϵ, A)
```

8.6 Expected Behavior of Algorithm

Thm. 7.5.1 states that the expected underestimation of $\tilde{\pi_k}$ is at most $21 + 5k \in O(k)$. This result may be used to analyze the "expected" behavior of Alg. 8.4.1.

During the course of the algorithm, in the LIF&Bonus part, we try to compute a divisor of the determinant, namely the product of r_{max} largest invariant factors $\pi_{r_{max}}$. The idea of the algorithm is to approach $\pi_{r_{max}}$ gradually, by computing the products $\tilde{\pi}_1, \tilde{\pi}_2, \ldots$ etc., which approximate $\pi_1, \pi_2, \ldots$ etc. This is implemented in the innermost **for** loop of the algorithm. The middle-lever **for** loop is conceive to update the products by repeating the computation with new left and right-hand side preconditioners.

The idea is to first, add new factors, and then, to correct the products by repeating the computation. In this way we may detect cases, when $\pi_k = \pi_{r_{max}}$ for $k < r_{max}$. We can also detect the cases when π_k is sufficiently close to the determinant. As it can be seen on the example of Pan [103] and Abbott's et al. [2] algorithms, sometimes $\pi_1 = s_n$ may be a sufficient.

The idea is purely heuristic, but by Thm. 7.5.1 some rigorous evaluations can be made. First, suppose that the **repeat** loop of the algorithm finishes. If we ignore all CRA steps that occur during the course of the algorithm, we obtain a procedure equivalent to the r_{max}-Bonus computation. By Thm. 7.5.1 we obtain $\pi_{r_{max}}$ up to $O(r_{max})$ bits.

Notice, however, that we performed $r_{min} \times r_{max} \times k_{iter}$ iterations of the CRA part, and compute at least $\log(l)$ bits of the determinant in each iteration. This allows us to obtain at least $r_{max} \times 5 + 22$ bits of the determinant, see the definition of k_{iter} in Alg. 8.4.1. This is exactly the expected under-estimation of $\pi_{r_{max}}$.

Remark 8.6.1 In general, if the number of non-trivial invariant factors is $k \leq r_{max}$, the innermost **for** loop will break at $i \leq k$. Some erroneous computation may lead the innermost **for** loop to break earlier i.e. when $i < k$. However, incrementing j in the middle-level **for** loop ensures that the computation is repeated. Therefore the error is corrected up to $O(k)$ bits as soon as j achieves r_{min}. Indeed, if $k_{app}(i) \geq k$ for all $i = 1..r_{min}$ (see line 13 of Alg. 8.4.1) ensures, that we have computed at least $\log(\pi_k)$ bits of the determinant. Therefore, once k equals to the number of non-trivial invariant factors, we get all bits of the result. The choice of k_{max} at the beginning of the algorithm ensures that CRA part should early terminate with probability at least $1 - \epsilon$.

Thm. 5.3.10 states that the expected number of invariant factors for random integer matrix with entries chosen uniformly and randomly from set of $\lambda > 1$ consecutive integers $\mathcal{S} = \{-\lceil \frac{\lambda}{2} \rceil + 1, \ldots \lfloor \frac{\lambda}{2} \rfloor\}$ is at most

$$\max(2, \left\lceil \sqrt{2 \log_\lambda (n)} \right\rceil) + 3.$$

The choice of r_{max}, greater than the expected number of non-trivial invariant factors, together with the above result ensures that "for an average matrix" the algorithm will stop in the adaptive part.

We will use this fact in the complexity considerations in the next section.

8.7 Complexity of Algorithm

We will consider the complexity of the adaptive algorithm.

8.7.1 Output Dependent Complexity

Suppose that the parameters Alg. 8.4.1 are defined as in Sec. 8.2.2.

Theorem 8.7.1 (Output-dependent Complexity of Alg. 8.4.1) *Let A be a $n \times n$ integer matrix with entries bounded by $\|A\|$ in absolute value. Suppose that A has k non-trivial invariant factors. Let the parameters for Alg. 8.4.1 be as in Sec. 8.2.2*

Denote by $C(Solver(n, \|A\|))$ the complexity of Solver procedure. The expected output dependent complexity of Alg. 8.4.1 is

$$O\left(n^{\omega} \log\left(1/\epsilon\right) + C(Solver(n, \|A\|))k\right)$$

as long as $k \leq r_{max}$.

PROOF For a matrix A, with k_{max} defined as in Eq. (6.5), the complexity of initial CRA iterations is $O\left(n^{\omega} \log\left(1/\epsilon\right)\right)$.

The cost of the remaining part of the algorithm is the cost of all calls to $LIF\&Bonus$ part 8.3.2, the cost of updating D in line 12, the cost of re-running the CRA loop in line 23, and the cost of all CRA iterations.

The cost of updating D by lcm is negligible in each iteration. To rerun the CRA loop, we need to recompute modular result $d_t/D \mod p_t$ and reconstruct $\det(A)/D \mod \prod_{i=0}^{t_{max}}$. By using fast integer reconstruction of [70, Ch.10.3], as $\det(A)/D = O(n \log(n\|A\|))$ in the worst case, this can be done in negligible cost.

If $k \leq r_{max}$, the algorithm is expected to stop when k factors are computed in line 14 when $i = k, j = r_{min}$ in the worst case. See remark 8.6.1 for explication. It is not expected to enter the CRA loop in line 19, as CRA iterations performed so far are expected to compensate for underestimation of $\tilde{\pi}_k$ before it happens.

At each iteration (i, j), either a $O(1)$ number of CRA iteration is run, or, the overall time of CRA iterations is bounded by $C(Solver(n, \|A\|))$ by a direct timing comparison, see the definition of k_{iter} in Alg. 8.4.1. As the complexity of modular determinant computation of *iteration* is asymptotically less than $C(Solver(n, \|A\|))$ in both cases, the summarized complexity of CRA iterations is $O(C(Solver(n, \|A\|)))$ at each iteration of the inner-most **for** loop.

Finally, the choice of r_{max} in Sec. 8.2.1 ensures that the cost all calls to $LIF\&Bonus$ part is at most that of $r_{min}(k+1)C(Solver(n, \|A\|))$, see Lem. 7.5.3.

Thus, the expected output dependent complexity is $O\left(n^{\omega} \log\left(1/\epsilon\right) + kC(Solver(n, \|A\|))\right)$ as required.

8.7.2 Expected Complexity

We may now estimate the expected complexity of the algorithm.

Theorem 8.7.2 (Expected Complexity of Alg. 8.4.1) *Let A be a random $n \times n$ integer matrix with entries chosen uniformly and randomly from the set $\mathcal{S}$ of λ consecutive integers, $\mathcal{S} = \{\lceil \frac{\lambda}{2} \rceil - 1 \ldots \lfloor \frac{\lambda}{2} \rfloor$. Let the parameters for Alg. 8.4.1 be as in Sec. 8.2.2.*

Denote by $C(Solver(n, \|A\|))$ the complexity of Solver procedure. Then the expected complexity of Alg. 8.4.1 is

$$O\left(n^{\omega} \log(1/\epsilon) + C(Solver(n, \|A\|))\sqrt{\log_{\lambda}(n)}\right)$$

PROOF By Theorem 5.3.10 $O(\sqrt{log_{\lambda}(n)})$ is the bound on the expected number of invariant factors, and this value is less than or equal to r_{max}, see Sec. 8.2.1. Let us assume that the number of non-trivial invariant factors of A is as expected. Then in Thm. 8.7.1 we may take $k = O(\sqrt{log_{\lambda}(n)})$ and the result follows.

The expected running time on a particular given random matrix can be re-estimated once $\tilde{s}_n$ is known. The trick is to approximate the expected number of invariant factors again, using the factorization of $\tilde{s}_n$ and following the proof of Thm. 5.3.10.

To compute the average complexity rigorously it not sufficient to replace it with the expected-case complexity. One should take into account the impact of all less probable cases. First, CRA iterations may fail to compensate for the k-bonus errors. This possibility can be excluded with hight probability by analysis in the spirit of Theorems [127, Thm. 5.8], 7.5.1 or 7.1.2. To obtain a good average complexity for the compensation may require increasing the values of r_{min} and/or β. The exact analysis is overwhelmed with technical details and in our opinion will not improve our understanding of Alg. 8.4.1 and its behavior. Instead, we try ro computing as many CRA as possible without time handicap. Notice, that if k_{iter} is set to $\frac{t_2}{t_1}$, it is asymptotically at least $O(\log(n))$, which should be more than necessary to compensate for even a significant under-approximation,. Therefore even a highly erroneous computation of $\tilde{\pi}_k$ could be compensated by the CRA steps and this case should not contribute to the average complexity.

It is more instructive to consider the impact of bigger number of non-trivial invariant factors on the algorithm. By 5.3.11, the probability that the number of invariant factors is greater than $\max(\sqrt{3/2}h + 1, h + 2)$, where $h = \max(2, \sqrt{\log_{\lambda}(n)})$ (i.e. about 1.23 times the expected value) is $O(n^{-1})$. If $r_{max} > \max(\sqrt{3/2}h + 1, h + 2)$, the complexity of reaching *Determinant* in line 29 is compensated by its low probability and the average complexity of the algorithm is equal to the expected one, provided that we can assume that the approximation of π_k is sufficiently exact.

8.7.3 Worst Case Complexity

Finally, we may estimate the worst case complexity of the algorithm.

Theorem 8.7.3 (Worst Case Complexity of Alg. 8.4.1) *Let A be a $n \times n$ integer matrix with entries bounded by $\|A\|$ in absolute value. Let the parameters for Alg. 8.4.1 be as in Sec. 8.2.*

Denote by $C(Determinant(\epsilon, n, \|A\|)$ the complexity of Determiant algorithm used in the worst-case part. The worst case complexity of Alg. 8.4.1 is

$$O\left(n^{\omega} \log\left(1/\epsilon\right) + C(Determinant(\epsilon, n, \|A\|))\right).$$

PROOF The choice of r_{max} in Sec. 8.2.1 is done in a way, that ensures the running time of the adaptive phase is asymptotically equal to the running time of the worst-case part (i.e. $C(Determinant(\epsilon, n, \|A\|))$). By adding the cost of initial CRA iterations we get the result.

8.8 Experimental Evaluation

Algorithm 8.4.1 is implemented in the LinBox in "linbox/algorithms/hybrid-det.h".

In the implementation, we have relaxed some of the requirements, setting $k_{max} = 3$, $r_{min} = 1$ and $r_{max} = 1$ or 2. We are using word-size primes in CRA part of the algorithm, which permits fast matrix multiplication in FFLAS routines, see [35, 36, 109]. We use LU factorization for *iteration* procedure. For *Solver* procedure we use p-adic lifting, see [27]. Random vectors and random matrices in the LIF&Bonus part are sampled from a stream of word-size integers, i.e. we assume $\beta \in O(1)$. The *Determinant* algorithm in the last, worst case, part is the Abbott's preconditioned CRA algorithm [2]. Yet, using this simplified setup, we were able to obtain timings, that are better than the state-of-the-arts for a wide class of matrices.

8.8.1 Generic Case

By Thm. 8.7.2 our adaptive algorithm claims the best up-to-date complexity for random dense matrices. This result is proved by our experimentations.

For a generic case of random dense matrices our observation is that the bound on the expected number of non-trivial invariant factors is quite crude. Indeed, for random dense matrices, the algorithm nearly always stopped with early termination after one system solving, which corresponds to the case $\log\left(\frac{\det(A)}{s_n}\right) \in O(1)$, or more precisely $\log\left(\frac{\det(A)}{\tilde{s}_n}\right) \in O(1)$, where $\tilde{s}_n$ is the result of the LIF algorithm, see Alg. 6.3.1. This together with fast underlying arithmetics of FFLAS [35] accounted for the superiority of our algorithm as seen in figure 8.1 and 8.2 where comparison of timings for different algorithms is presented.

Let us focus on the comparison between our adaptive algorithm (see Hybrid algorithm in Fig. 8.1) an the state-of-the-art of [122], which claims currently the best theoretical complexity (see Storjohann-Giorgi-Olesh in Fig. 8.1). Our algorithm beats both the uncertified (Monte Carlo) and certified version of the algorithm. See [123] for the results

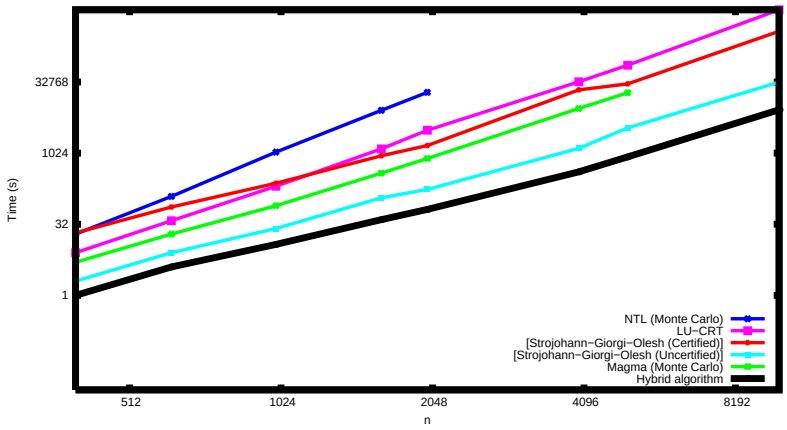

Figure 8.1: Comparison of our algorithm with other existing implementation. Tested on random dense matrices of the order 400 to 10000, with entries {-8,-7,. . . ,7,8} Using fast modular routines puts our algorithm several times ahead of the others. Scaling is logarithmic.

of algorithm [122]. The implementation of the algorithm does not use the fast rational solver, and nor is ours.

As it can be seen in figure 8.2, our algorithm is asymptotically better than the classic CRA algorithm. Moreover, our implementation is faster that Abbott's et al. algorithm [2] by a constant factor. This is due to the fact, that our implementation can make use of *a priori* erroneous output $\tilde{s}_n$ and therefore, in a generic case, needs less executions of p-adic lifting than the algorithm of [2].

8.8.2 Malicious Matrices

Thanks to the introspective approach our algorithm can detect the cases when the number of invariant factors is small and equal to $k < r_{max}$, see Sec. 8.6, and has good complexity in this case, see Thm. 8.7.1. To test the ability of our algorithm to detect propitious cases, we have run it on various sets of structured and engineered matrices. The adaptive approach allowed us to obtain very good timings which motivates us to encourage the use of this algorithms in the situations which go further beyond the random dense matrix case.

In Figure 8.3 we present the results of the determinant computation for sparse matrices of N. Trefethen[1], which have a relatively high number of non-trivial invariant factors. Our algorithm gives the best timings for this class of matrices.

In what follows we choose to evaluate our algorithm in the case of matrices, for which the number of non-trivial invariant factors is even bigger. In Table 8.1 we give the timings for our algorithm with $r_{max} = 1$ (equivalent [2]) and 2. The algorithms was run on a set of specially engineered matrices which have the same Smith form as $diag\{1, 2 \ldots, n\}$ and the number of invariant factors of about $\frac{n}{2}$. We notice that the algorithm with $r_{max} = 1$

[1] http://ljk.imag.fr/membres/Jean-Guillaume.Dumas/Matrices/Trefethen/

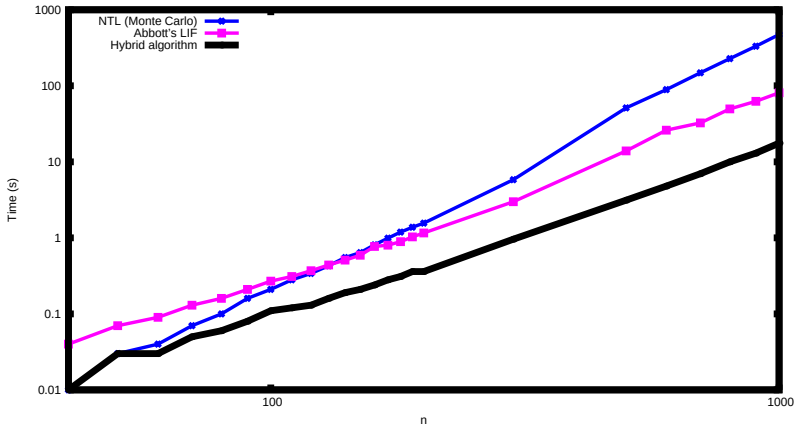

Figure 8.2: Comparison of our algorithm with early terminated Chinese remaindering algorithm (LU) and the algorithm of Abbott *et al.* [2] (LIF). Tested on random dense matrices of the order 40 to 1000, with entries {-100,-99,...,99,100}. When matrix size exceeds 80 the adaptive algorithm wins. Scaling is logarithmic.

n	$r_{max} = 1$	$r_{max} = 2$	n	$r_{max} = 1$	$r_{max} = 2$
100	0.17	0.22	300	5.65	5.53
120	0.29	0.33	350	9.76	9.64
140	0.48	0.55	400	14.99	14.50
160	0.73	0.78	600	57.21	54.96
180	1.07	1.16	800	154.74	147.53
200	1.49	1.51	1000	328.93	309.61
250	2.92	3.00	2000	3711.26	3442.29

Table 8.1: Comparison of the performance of Alg. 8.4.1 with r_{max} set to 1 and 2 on engineered matrices.

runs better for small n i.e. the computation of s_{n-1} in this case, despite the fact that $s_{n-1} > 1$. This motivated us to develop an even more adaptive approach, which we describe in Section 8.9.

The results encouraged us to construct a sparse variant of our algorithm, which we shortly describe in Section 8.10.1. Figure 8.3 gives a comparison of the performance of sparse and dense variants. We used the sparse solver of [43] and a sparse modular determinant algorithm implemented in LinBox. Using the algorithm with the dense solver outperforms using the sparse solver by a factor of 3.3 to 2.3, and decreasing with the matrix size n. Thanks to the space-efficiency of the sparse algorithm we are able to compute the determinant for 20000×20000 matrix for which the dense solver resulted in memory trashing.

In figure 8.4 we compare the performance of dense and sparse variants of the algorithm with the CRA algorithm (sparse variant) for random sparse matrices. The matrices are very sparse (20 non-zero entries per row). To ensure that the determinant is non-zero

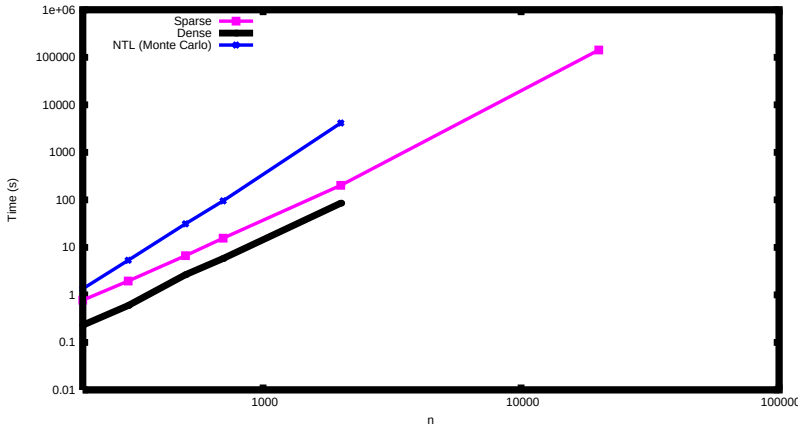

Figure 8.3: Comparison of sparse and dense variants of our determinant algorithm for Trefethen's matrices. Scaling is logarithmic.

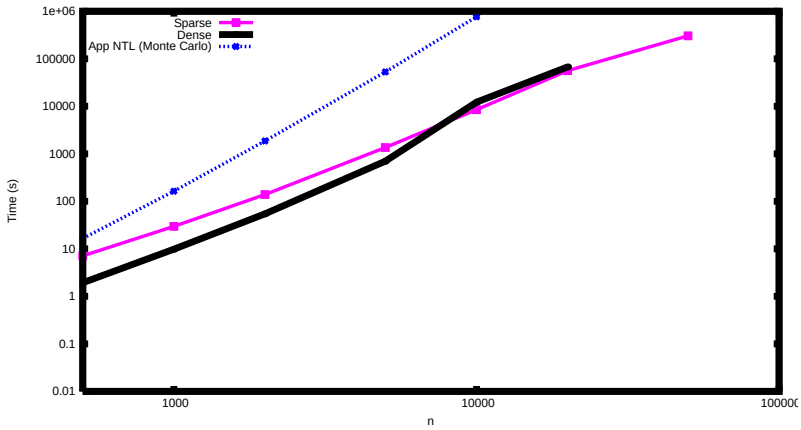

Figure 8.4: Comparison of sparse and dense variants of our determinant algorithm with the CRA algorithm for random sparse matrices. Scaling is logarithmic. The running time of the CRA algorithm has been approximated based on the timings for one iteration

we put 1 on the diagonal. Both dense and sparse variants of the algorithm have better running times than the CRA, which proves that we can detect propitious cases for sparse matrices. Furthermore, sparse variant is best for bigger matrices and again lets us solve the problem when the dense variant fails due to insufficient memory.

8.9 Further Introspective Modifications of Algorithm

We start with a simple remark. For every matrix, with growing i, the size of s_{n-i} decreases whilst the cost of its computation increases. In Table 8.1, this accounts for better performance of Abbott's algorithm ($r_{max} = 1$), which computes only the last factor s_n, in the case of small n. For bigger n calculating the penultimate factors s_{n-1} starts to pay out. The same pattern could repeat for s_{n-1} and s_{n-2} etc. as matrix size increases.

The switch between winners in Table 8.1 can be explained by the fact that, in some situations, obtaining s_{n-i} by CRA (which costs $\frac{\log(s_{n-i})}{\log(l)} \times t_1$, where t_1 is the time of LU) outperforms system solving. Then, this also holds for all consecutive factors and the algorithm based on CRA wins. The condition can be checked *a posteriori* by approximating the time of LUs needed to compute the actual factor. We can therefore construct a condition that would allow us to turn to the CRA loop in the appropriate moment. This can be done by changing the condition in line 13 ($\tilde{\pi}_i = \tilde{\pi}_{i-1}$) to

$$\log\left(\frac{\pi_i}{\pi_{i-1}}\right) \leq \frac{t_1}{t_2}\log(l)\,,$$

if the primes used in the CRA loop are greater than l and t_1, t_2 are the times of the *Solver* and LU. This would result with a performance close to the best and yet flexible.

If, to some extend, $s_{n-i}, i > 0$ could be approximated *a priori*, this condition could be checked before the computation of s_{n-i}. This would require a partial factorization of s_{n-i+1} and probability considerations as in Thm. 5.3.10 and [44]. This method cannot however by applied to better approximate s_n.

Notice, that at least one system of linear equations has to be solved during the course of the algorithm in order to approximate s_n. This means that the minimal and inevitable cost of the algorithm, for an $n \times n$ matrix with entries bounded by $\|A\|$ in absolute value, is $O(n^3(\log^2(n\|A\|))^2)$ for the p-adic lifting of [27]. In the [27] algorithm, this cost does not depend on the computed size of the solution, but an output dependent solver, which uses early terminated rational reconstruction, can be envisaged, see Sec. 11.3 and Sec. 11.6.2 for experiments.

The p-adic solver could find $\tilde{s}_n$ in roughly $K \sim 2\rceil \log_p(\tilde{s}_n)\rceil$ iterations of $O^{\sim}(n^2\log(\|A\|n)+\log(\beta))$ bit operations each, assuming that $\log_p$ for the vector of numerators of the solution vector is less than or equal to $\rceil \log_p(\tilde{s}_n)\rceil$ as well, see Sec. 11.3. Depending on the size of output on early termination, we may either run the CRA loop or continue the p-adic solver until certification if necessary. While considering the use of early terminated p-adic lifting, one should be aware of using the uncertified output as preconditioner D might violate $D \mid \det(A)$ condition.

A direct comparison of times of one modular LU and one lifting step can thus heuristically be used to decide whether to perform the first solving. Informally speaking, one LU iteration $\mod p$ gives us $\log_p$ bits of $\det(A)$, whereas one p-adic step gives us $\log_p/2$ bits. Thus, early terminated p-adic lifting would recover $\tilde{s}_n$ quicker than the CRA scheme provided that the time t_3 is less than $\frac{t_1}{2}$, where t_1 denote the time of one modular LU. This reasoning ignores the time of integer and rational reconstruction in CRA and p-adic lifting, hence, in practice, condition should be rather be of a form $t_3 < \frac{t_1}{\alpha}$ for $\alpha > 2$. Notice however, that in the asymptotic case and for dense matrices, we expect $t_3 \ll t_1$. We recall here, that the problem of rational reconstruction and early termination in p-adic lifting is much more complex than the integer reconstruction in CRA. We will discussed this in Ch. 10 and Ch. 11.

8.10 Beyond Random Dense Matrices - Application to Structured Matrices

Structured matrices has to be consider from two different points of view in the case of Alg. 8.4.1. First, dense implementations may be used for structured matrices. Thm. 8.7.1 gives the output-dependent complexity in this case, but we need expected complexity to evaluate how often adaptive approach pays out. Therefore, there is a need for probabilistic analysis on the number of non-trivial invariant factors in the case of a particular structure.

Secondly, other linear solving and determinant algorithms dedicated to structured matrices can be used, which implies new evaluation for both output-dependent and expected complexities.

In this manner we will analyze the case of structured matrices on the example of sparse matrices.

8.10.1 Sparse Case

Sparse matrices are matrices, which have more entries equal to 0 than average matrix. Therefore, we introduce a parameter Ω - the number of non-zero factors and require that Ω is considerably smaller than n^2 for $n \times n$ matrices. In the implementation, only the Ω non-zero entries are stored. The basic operation on this type of matrices is matrix-vectors product, which costs $O(\omega)$.

Probability Revisited

The definition of random sparse matrices is given in Sec. 5.1.3. In Sec. 5.3.3 it is proven that in the case of integer sparse matrices the expected number of invariant factors divisible by 2 is finite, see Cor. 5.3.13, and that, under some additional assumptions, the expected number of invariant factors divisible by $p > 2$ is finite for every prime p separately, see Cor. 5.3.15. Another weaker characterization is also given in Cor. 5.3.5.

In particular, Cor. 5.3.5 implies that the number of invariant factors can be quite big. This is clearly visible in the case when $\epsilon = 1$, i.e. when the expected number of non-zero entries in a row of the matrix is finite. The expected number of non-zero invariant factors is $O(n)$ at each prime p in this case. Indeed, Cor. 5.3.13,5.3.15 exclude this case.

Intuitively, larger number of invariant factors divisible by a prime p is due to occurrence of rows (resp. columns) which are divisible by p. Such columns might be detected beforehand by row (resp. column) gcd computation. Let g_i denote the gcd of all entries in a row (resp. column). Notice, that g_i divides the determinant in this case. Thus, Alg. 8.4.1 can be considered for a new matrix $\tilde{A} = A\,\mathrm{diag}(\frac{1}{g_i})$ (resp. $\mathrm{diag}(g_i)A$), which will have lower number of non-trivial invariant factors. Some results for modular rank in this case are presented in [20]. The question of the expected number of invariant factors for a sparse matrix in this case remains open.

Algorithms Revisited

To apply Alg. 8.4.1 we need a dedicated sparse methods for modular determinant computation and system solving. Additionally, we need a dense routine to compute the determinant of a small at most $r_{max} \times r_{max}$ matrix with entries bounded in size by $O(n \log(n\|A\|)$.

Instead of the dense LU, sparse elimination can be used in practice e.g. for extremely sparse matrices [40]. In general, black box method are preferred. The idea is to precondition the matrix so that its characteristic polynomial equals its minimal polynomial [45, 14]; and then to compute the minimal polynomial via Wiedemann's algorithm [133]. The complexity of the sparse modular determinant computation is then $O(n\Omega)$ [40, Table 4]. Adaptive solutions is suggested in [42].

For solving a sparse system of linear equations the solver of [43] can be used. By similar reasoning as in [97], the cost of one solving is that of $O(n^{1.5}\log(n\|A\|)+n^{0.5}\log(\beta))$ matrix-vector products and $O(n^2\log(n\|A\|)(n^{0.5}+\log(\|A\|))+n^2\log(\beta)\log(n\|A\|)+n\log^2(\beta))$ additional arithmetic operations.

Let us now analyze the existing integer determinant algorithms, that can be used in the worst-part switch. Possible solutions include a CR and preconditioned CR Algorithm, which has the worst-case complexity of about $O(\Omega n \log(|\det(A)|))$ bit operations, see Thm. 6.3.3. Also, the algorithm of [44] can by adapted to the sparse case, by using the sparse solver of [43] in the LIF procedure. In this way, we obtain an algorithm with complexity $O\big(n^{1.5}\Omega\log(n\|A\|)\log(\|A\|)\log^2(n)\sqrt{\log(|\det(A)|)}\big)$, which, unfortunately, does not improve the complexity of CRA scheme.

Let us now analyze the choice of r_{max} in the adaptive Algorithm 8.4.1. By analyzing the complexities of subprocedures, as in Sec. 8.2.1 and Lem. 7.5.3 we obtain that r_{max} should fulfill the following conditions.

By comparing the cost of one system solving and the cost of "bonus" computation i.e. the determinant of $r_{max} \times r_{max}$ matrix LN, where $\|LN\| \in O(n\log(n\|A\|))$, LN is a dense matrix, we obtain the condition:

$$\begin{aligned} r_{max}^2 nM(n\log(n\|A\|)) + r_{max}\log(r_{max}\|LN\|)(r_{max}^{\omega} + r_{max}^2 M(\|LN\|)) \\ \in O\big(n^{1.5}\Omega\log(n\|A\|)\log(\|A\|)\big) \end{aligned}$$

where $M(x)$ is the cost of multiplication of two x-bit integers.

By comparing the cost of the adaptive part with the cost of computing the determinant in the worst case, we obtain

$$r_{max}n^{1.5}\Omega\log(n\|A\|)\log(\|A\|) \leq n^2\Omega\log(n\|A\|)$$

This allows us to take

$$r_{max} = \min\left(\sqrt[3]{\frac{\Omega}{n^{0.5}\log(n)\log(n\|A\|)}}, \frac{n^{0.5}}{\log(\|A\|}\right)$$

In the propitious case, when the number of non-trivial invariant factors is smaller than r_{max} we obtain an algorithm with the running time better than currently known algorithms. In the malicious case when this condition is not fulfilled, the running time of the algorithm is the same as for the worst case CRA algorithm. Additionally, adaptive modifications of Sec. 8.9 may help us obtain satisfactory running time in this case as well. This is suggested by good experimental results, see Fig. 8.4.

Practical Speedup

In fact, by the adaptive modification of Sec. 8.9 we only need to consider the number of "big" invariant factors, i.e. the number of invariant factors which are bigger than a certain parameter C. The parameter has to be chosen in a way, that the product of all smaller factors can be computed by modular CRA loop quicker than another rational solution of a system of equation. We could exploit here the difference in complexity between system solving and one modular routine which is $O(n^{1.5}\Omega)$ to $O(\Omega n)$ in the case of sparse procedures. This could enable us to recover $O(n^{0.5})$ bits of the determinant by running modular routines without exceeding the cost of one linear system solving, see the definition of k_{iter} in line 9 of Alg. 8.4.1.

Memory Usage

One of the reason of using the sparse variants of the algorithm is the reduced memory usage. In the case of our algorithm, the memory used is dominated by the "bonus" computation, where storing matrix LN takes $O(r_{max}n^2 \log(n\|A\|))$ bits.

8.10.2 Toeplitz and Hankel Matrices

Definition 8.10.1 (Toeplitz and Hankel Matrices) Let $n \in \mathbb{N}, n > 0$. Let $A = [a_{i,j}]_{i,j=1..n}$ be a $n \times n$ matrix. We say that A is a

1. Toeplitz matrix, if $a_{i,j} = a_{i-1,j-1}$ for all $i, j = 2..n$.
2. Hankel matrix, if $a_{i,j} = a_{i-1,j+1}$ for all $i, j = 2..n$.

The analysis of this case is similar to the sparse case. The main feature of Toeplitz and Hankel matrices (for fixed $\|A\|$) is that the cost of matrix-vector product is $O^{\sim}(\log(n))$. In [104, 106, 105, 107] a rational solver of complexity $O(n^2 \log^2(n))$ is proposed and possible determinant algorithms are discussed. Experiments suggest, that the expected number of invariant factors might be significant in this case, see [107] for the study of modular rank degeneration.

Part III

Adaptive Rational Algorithms

9

Motivations

To our knowledge, the problem of the exact solution to linear algebra problems for rational matrices (i.e a matrix with rational entries) has not been widely studied so far. In general, exact algorithms can be used everywhere where large precision is required. For example, the determinant can be too close to 0 or $\pm\infty$ and thus cannot be computed by floating point precision algorithms. In the case of ill-conditioned matrices, symbolic methods can be preferred as rounding errors can spoil the computation.

Standard linear algebra routines often fail in the case of ill-conditioned matrices. As a remedy, the use of other methods such as interval arithmetics or multiple-precision floating point computation has been proposed. The other possibility is the use of exact methods, which would perform the computation on arbitrary precision integers or rational numbers (treated as a pair consisting of a numerator and a denominator).

One should notice here that floating point representation allows to represent exactly only the fractions with denominators being a power of 2. Rational field implementations would allow us to represent exactly all fractions, and therefore any real-life values such as measurement results (decimal fractions). better to our problem. This is an opportunity that should not be sneezed at, as ill-conditioned problem are sensitive to data modifications. Another interesting possibility might be to compare the decimal/binary fractions approximation of our real-valued data with continued fractions approximation. Continued fractions are the best approximants with small denominators, see [70, Ch. 4].

Rational field arithmetics is implemented in GMP[1] and Givaro[2] libraries. In general, rational numbers are difficult to treat from exact computation point of view. Mainly, the size of numerator and denominator can increase very quickly with every addition and multiplication. When we add or multiply two fractions with numerators and denominators bounded by M, the numerator and denominator of the result are bounded by $O(M^2)$. Moreover, one addition requires 3, and one multiplication requires 2 integer products, as well as a gcd computation. Therefore, the cost of an exact matrix-vector or matrix-matrix product can be prohibitive in practice. This prohibits the use of the rational field $\mathbb{Q}$ in most exact linear algebra algorithms which rely on matrix-matrix or matrix-vector products. In

[1] http://gmplib.org/
[2] http://www-ljk.imag.fr/CASYS/LOGICIELS/givaro/

particular, algorithms might be correct for matrices over a field, but not practical for matrices over $\mathbb{Q}$. For example, this is the case of elimination-based algorithms.

9.1 Matrix Storage

To perform computations on a rational matrix $A = \left[\frac{a_{ij}}{b_{ij}}\right], b_{ij} > 0$ the problem of matrix representation for storage has to be considered. First, a matrix of rational numbers can be stored, where entries are elements of the GMP or Givaro rational field. Yet, there exist also other possibilities.

Suppose that $A = \frac{a_{ij}}{b_{ij}}$. We call $D(A)$ the common denominator of all entries of A i.e. $D(A) = \mathrm{lcm}(b_{ij})$. Then, matrix A can be stored as a pair $(A', D(A))$, where

$$A' = D(A) \cdot A \tag{9.1}$$

is an integer matrix. Let $\|A\|_r = \max(\|a_{ij}|, b_{ij})$ be the largest absolute value of a numerator or denominator of A. When the entries of A are decimal or binary fractions, $D(A)$ can be set to a small power of 10 or 2, and $\|A'\|$ is close to $\|A\|_r$. Unfortunately, if we only assume that the values $|a_{ij}|, b_{ij}$ are less than M, both $D(A)$ and $\|A'\|$ are bounded by $O(M^{n^2})$ and storage of A' requires more memory.

Instead of taking the common denominator of all matrix entries, we may take the common denominators by rows (columns). We define $D_i(A) = \mathrm{lcm}(b_{ij})$ and $E_j(A) = \mathrm{lcm}(b_{ij})$. Then, diagonal matrices $\tilde{D} = \mathrm{diag}\, D_i(A)$ and $\tilde{E} = \mathrm{diag}\, E_i(A)$ can be formed and we define integer matrices

$$\tilde{A}_1 = \tilde{D}A, \quad \tilde{A}_2 = A\tilde{E}. \tag{9.2}$$

A can be represented by a pair of matrices $(\tilde{D}, \tilde{A}_1)$ or $\tilde{E}, \tilde{A}_2$. In what follows, we will use the notion $\tilde{A}$ to denote either of the preconditioning $\tilde{A}_1$ or $\tilde{A}_2$ if distinction between the two is not necessary and both cases may be analyzed at the same time.

This representation is equivalent to the previous one if and only if $D_i(A) = D(A)$ ($E_i(A) = D(A)$ resp.) for all i. However, contrary to multiplying by $D(A)$, multiplying by a diagonal matrix can disturb matrix structure (for example symmetric, Hankel or Toeplitz). In general, if we assume that the values $|a_{ij}|, b_{ij}$ are less than M, then $\|\tilde{A}\|$ is $O(M^n)$, thus less than $\|A'\|$.

In this chapter, we will ignore the cost of generation of A' and $\tilde{A}$ from A. We will assume that one of the representation $A, (D(A), A'), (\tilde{D}, \tilde{A})$ is given at the input of the algorithm. In practice, representations might be computed while data are read at relatively small cost when compared to the main algorithm.

9.2 Modular Computation for Rational Matrices

The cost of computing the modular image of a fraction $\frac{a}{b}$, where a, b are of moderate size, should be comparable with the cost of computing the modular image of a large integer

number. This allows us to compute the modular image of a rational matrix A as well as for its representations by A' and $\tilde{A}$ at comparable cost.

In past decade modular linear algebra routines have undergo major enhancements so that they are nowadays quite as fast as numerical routines see e.g. [36]. This is mostly due to the effective memory management. It has been demonstrated with the BLAS and LAPACK development [29, 5] that this can be the key factor in the performance of algorithms. BLAS and LAPACK routines have been successfully applied to the exact modular computation see e.g. [35, 36, 109], thus making matrix multiplication in LinBox practically subcubic and fast. The same is not yet the case for multiprecision or interval arithmetics. This motivates our concentration on exact linear algebra as a solution to numerical ill-conditioned applications.

Considering the good performance of modular routines, this encourages us to use repeatable modular computation in order to perform rational computation. Chinese remaindering algorithm coupled with rational reconstruction has long been considered e.g. in the case of solving a system of linear equations with rational coefficients, see e.g [12], or for polynomial gcd computation, see [19]; in these situations the input is usually integer but the result is rational, however, the method carry on to rational input as well.

In this part of the thesis we would like to revisit this concept and evaluate its practical applicability. We will consider the problem of system solving, determinant, characteristic and minimal polynomial computation for dense and sparse rational matrices. Our analysis takes into account recently developed ideas such as preconditioning for CRA, see Chapter 6. We will consider the applicability of fast rational reconstruction, which was first given by X. Wang and Pan [131], in 2002 and first implemented by Lichtblau, [88] in 2005. We will focus on early termination techniques for CRA, which were first given by P.S. Wang in [129, 130] and, recently by Monagan, [90] in 2004.

A number of possible variants of the algorithm is large and includes treating matrices $A, A', \tilde{A}$. This, eventually makes the use of integer algorithms envisageable, depending on the problem. As the ultimate goal of this part, we would like to propose an adaptive, introspective strategy, which would try to choose the best variant for a particular input instance.

9.3 Outline of this Part

We will approach the problem of rational computation in a generic manner. In order to do that, in Ch. 10 we present a generic Chinese Remaindering Algorithm and present the state-of-the-art implementation of its components, with a special emphasis on the scalar and vector case. In Ch. 11 we use the fast rational reconstruction ideas of [131] and provide its implementation. Then we change the algorithm in order to be able to apply the early termination technique of [90], which we evaluate experimentally. The implementation of Monagan's [90] strategy has been done in the case of polynomials, see [82], but to our knowledge, has not been performed in the case of integers.

Then in Ch. 13 we analyze the preconditioning strategies for rational matrices, which can be used for a particular problem. In Ch. 14 we present possible strategies which leads to the construction of an adaptive rational algorithm. In Sec. 14.2 we analyze the complexity of the algorithms and evaluate it experimentally in Sec. 14.3.

10

Chinese Remaindering Algorithm - Survey

The Chinese Remaindering Algorithm is a core procedure for many integer and rational algorithms. For example, CRA coupled with rational reconstruction could be the first choice for the computation of the determinant of a rational matrix, for solving a system of linear equations with rational coefficients etc. The rational case seems more difficult than the integer one, as the number of iterations of the CRA loop might be by far more significant. Therefore in this chapter we would like to analyze the existing state-of-the-art approaches to the CR Algorithm in a most general setting. Our analysis includes early terminated and certified variants of the algorithm, together with the probability considerations, scalar and vector case analysis and parallel implementation among other problems. For a presentation of basic ideas of the CR algorithm we refer to [70, Sec.5.4,5.5,10.3]. The problem of (integer) early termination has also been considered in [75, 38, 39, 4].

10.1 CR Algorithm

In Alg. 10.1.1 we would like to present a generic ET CR Algorithm for computation with rational matrices. Let $n, m \in \mathbb{N}, n > 0$ and let A - a $n \times m$ matrix over $\mathbb{Q}$, be given at the input of the algorithm[1]. Let $0 \leq \epsilon < 1$ be the probability of error of the algorithm. Let X be the computational problem that we would like to solve i.e. we are looking for the value of a rational function $X(A)$. As an examples, X can be the determinant, minimal or characteristic polynomial or a solution to equation $Ax = b$, for a given vector $b \in \mathbb{Q}^m$.

Suppose that $X(A)$ is a vector of size $s \geq 1$ and let $\|X(A)\|$ be its maximum norm. Let $Den(X(A)), Num(X(A))$ be the vectors of numerators and denominators of $X(A)$. In the preliminaries, we start with an estimation of a bound $H = H(A) \in \mathbb{Z}$ which is such that

$$\|X(A)\|_r = \|X(A)\| \leq H \quad if \quad X(A) \in \mathbb{Z}^s$$
$$\|X(A)\|_r = \max(Den(X(A)), |Num(X(A))|) \leq H \quad if \quad X(A) \in \mathbb{Q}^s\,.$$

[1] Analogously, other object such as polynomials might be considered here

Moreover, CRA algorithm requires a subset of primes P and function $get\_prime(p)$, which chooses a prime out from P and removes it. In the case when $\epsilon > 0$, get_prime generates a random element of P. Additionally, a preconditioned $D \in \mathbb{Q}_+^s$ might be given. We define D^{-1} as a vector of inverses of elements of D, and $\cdot * -$ defines a piecewise multiplication.

We assume that the following functions are given and fulfill the requirements:

1. $get\_prime(p)$, which chooses $p \in P$ and updates P by setting $P = P \setminus \{p\}$;

2. $image(A_p, A, p)$, which returns $A_p = A \mod p$ for a prime $p \in P$;

3. $iteration(y, A_p)$, which returns $y = X(A) \mod p$ for a prime $p \in P$;

4. $reconstruct(x_t, y_0, \ldots, y_t, p_0, \ldots, p_t)$, which either returns $x_t = X(A) \mod \prod_{i=0}^{t} p_i$ or **false**. If $reconstruct$ returns an integer value it computes

$$-\lceil \frac{\prod_{i=0}^{t} p_i}{2} \rceil + 1 \leq x_t \leq \lfloor \frac{\prod_{i=0}^{t} p_i}{2} \rfloor$$

   such that x_t is the solution to the equation

$$\begin{aligned} y_0 &= X(A) \mod p_0 \\ &\ldots \\ y_t &= X(A) \mod p_t; \end{aligned}$$

   By the Chinese Remaindering Theorem, x_t exists and is unique. In the when **false** is returned, there $reconstruct$ might perform partial integer reconstruction and store results in a structure. By returning **false**, $reconstruct$ incorporates scheduling for integer reconstruction;

5. $get\_rational(d_t, x_t, \prod_{i=0}^{t} p_i)$, which returns a rational d_t, such that $d_t = x_t \mod \prod_{i-0}^{t} p_i$ or **false** if reconstruction fails. If it can be proven that $X(A) \in \mathbb{Z}$, this function should be skipped.

6. $terminated(\epsilon, P, H)$, which returns true if $x_t = x$ with probability $1 - \epsilon$.

The algorithm is given in a general form in Alg. 10.1.1.

The cost of $get\_prime, image, iteration, terminated$ is the same for each iteration of Alg. 10.1.1. The cost of $reconstruct, get\_rational$ increases with each iteration.

Let us now discuss the implementation of functions $image$, $iteration$, $reconstruct$ and $get\_rational$, and $terminated$. We need to distinguish between the cases when $X(A)$ is a scalar or a vector.

Algorithm 10.1.1 ET CRA to compute $X(A) \in \mathbb{Q}^s$

Require: $n \times m$ rational matrix A,
Require: $H \in \mathbb{R}$, $|X(A)|_r \leq H$,
Require: $0 \leq \epsilon < 1$ the probability of failure of the algorithm,
Require: P a subset of primes,
Require: $D \in Q^s$ a preconditioner,
Ensure: $X(A) \in \mathbb{Q}^s$ with probability at least $1 - \epsilon$;

```
t = 0;
repeat
    get_prime(p_t);
    image(A_{p_t}, A, p_t);
    iteration(y_t, A_{p_t});
    y_t = D * y_t;
    if reconstruct(x_t, y_0, ..., y_t, p_0, ..., p_t) then
        get_rational(d_t, x_t, ∏_{i=0}^{t} p_i)
    t = t + 1;
until terminated(ε, P, H)
Return X(A) = D^{-1} * get_rational(d_{t-1}, x_{t-1}, ∏_{i=0}^{t-1} p_i);
```

10.2 Image of a Rational Matrix

Let us define the cost of *image* procedure as a function $Im(A, p)$, where $p = \max(P)$ is the maximal prime used in Alg. 10.1.1.

Let us consider the case of rational matrices. Let $n, m \in \mathbb{N}$ and let $A = [\frac{a_{ij}}{b_{ij}}]$ be a $n \times m$ rational matrix, with a Ω non zero entries. If A is a dense matrix, $\Omega = nm$ may be taken. Let us consider matrix representations $(D(A), A'), (\mathrm{diag}(D_i), \tilde{A})$ be $(\mathrm{diag}(E_i), \tilde{A})$ from Sec. 9.1 given by Eq. (9.1), (9.2). Recall that $\|A\|_r = \max(|a_{ij}|, |b_{ij}|)$ is the bound for entries of A, $D(A) = \mathrm{lcm}(b_{ij})$ is the common denominator and D_i (resp. E_i) are common denominators for of each row (resp. column) of A.

The complexity of modular imaging depends on the matrix representation chosen. It is expressed in terms of $p, \Omega, \|A\|_r, D(A), \|A'\|, D_i$ and $\|\tilde{A}\|$, see Lem. 10.2.1. The structure of matrices $A, A', \tilde{A}$ should be also taken into consideration. These parameters are known in the course of the algorithm so the choice of the best representation can be made either by comparison of the derived complexities or by direct comparison of imaging times for each representation. The following lemma gives the cost of image computation.

Lemma 10.2.1 (Cost of Image) *Let p be a prime. Let $n, m \in \mathbb{N}$ and let $A = [\frac{a_{ij}}{b_{ij}}]$ be a $n \times m$ rational matrix, with a Ω non zero entries. Let $(D(A), A')$ and $(\mathrm{diag}(D_i), \tilde{A})$, where $\tilde{A} = \mathrm{diag}(D_i)A$ be its representations. Let* EEA *stand for the cost of Extended Euclidean Algorithm, where* $\mathrm{EEA}(p) = O(\log(p) \log^2 \log(p) \log(\log(\log(p))))$ *if the fast Euclidean algorithm is used.*

1. *The complexity of $image(A, p)$ for a rational matrix A is $O(\Omega \log(\|A\|_r))$ operations modulo p + $O(\Omega\, \mathrm{EEA}(p))$ bit operations;*

2. *The complexity of image*(A, p) *for a rational matrix* $(D(A), A')$ *is* $O(\Omega(\log(\|A'\|)) + \log(D(A)))$ *operations modulo* p. *In the worst case* $D(A) = \|A'\| = O(\|A\|_r^{\Omega})$.

3. *The complexity of image*(A, p) *for a rational matrix* $(\mathrm{diag}(D_i), \tilde{A})$ *is* $O(\Omega(\log(\|\tilde{A}\|)) + \sum_i \log(D_i))$ *operations modulo* p. *In the worst case* $D(A) = \prod_i D_i$ *and* $\|\tilde{A}\| = \|A\|_r^m$.

PROOF For a matrix without a pattern we compute an image for all Ω entries. For a rational fraction the cost is $O(\log(\|A\|_r))$ for the computation of the modular image of the numerator and denominator and EEA(p) for the modular inverse computation. In the case of integer matrices $A', \tilde{A}$, the cost is that of modular image computation for the entries, which is $O(\Omega \log(\|A'\|))$ and $O(\Omega \log(\|\tilde{A}\|))$ respectively. Additionally the cost of taking $D(A), D_i$ modulo p has to be considered.

In the worst case, $D(A)$ might be $\|A\|_r^{\Omega}$ as a lcm of Ω co-prime numbers; also $\prod D_i = D(A)$ in this case. At the same time, $\|A'\| = \|D(A)A\|$ might be about the same value as $D(A)$. As D_i is a lcm of all row denominators it is the lcm of at most m numbers. The maximal entries of $D_i A_i$ for the row A_i of A might therefore be integers reaching $\|A\|_r^m$.

In view of Lem. 10.2.1, the cost of division in the modular imaging of A has to be compared to the cost of taking modular images of possibly many times larger integers of $A', \tilde{A}$. The relation between the cost depends heavily on the particular matrix instance. Moreover, whereas A' in general preserves matrix structure, some favorable properties (symmetry, Hankel/Toeplitz type) may be lost while considering $\tilde{A}$. Due to this many factors and taking into account that *image* procedure is repeated at each iteration, we suggest that a direct timing comparison might be the best way of choosing the optimal imaging scheme. Some experimental result are presented in Sec. 14.3.1.

10.3 Iteration in CRA

Let us define the cost of *iteration* procedure as a function $It(A, p)$, where $p = \max(P)$ is the maximal prime used in Alg. 10.1.1.

We will present the cost estimations in term of modulo p operations for the linear algebra problems that we will consider later on. In the following subsections we will analyze the cost of system solving, minimal and characteristic polynomial and determinant computation for dense and sparse matrices over a modular field.

Let A be a $m \times m$ matrix over a field and let b be a vector of size m. We consider modulo p computations of the determinant, minimal/characteristic polynomial or linear system solution. The complexities are expressed in the number of modulo p operations.

1. Characteristic and Minimal Polynomial

   In his Ph.D. thesis [109] Pernet gives a throughout description of the existing algorithms to compute the modular minimal polynomial in the case of dense and sparse matrices. The work covers both Krylov-Lanczos approaches for dense matrices and

black box approach of Wiedemann [133] for sparse matrices. The complexity of the algorithm is $O(m^\omega)$ in the dense case and $O(m\Omega)$ in the sparse case. See [109] and the references therein for results.

2. System Solving

   Let us consider a system of m linear equations $Ax = b$. If A is dense, then the solution can be computed from the LU factorization of A, where L and U are lower and upper triangular matrices. The complexity of computing the factorization is $O(m^\omega)$, as BLAS matrix multiplication might be used. In the case of a sparse matrix, one can compute the minimal polynomial for Ab, i.e. a minimal polynomial $p_{Ab}(x) = \sum_{i=0}^{d} c_i x^i$ such that $p_{Ab}(A)b = 0$. Then the solution x is given by

$$x = -\sum_{i=1}^{d} c_i A^{i-1} b.$$

   The complexity for the minimal polynomial computation is $O(m\Omega)$, where Ω is the number of nonzero entries of the matrix.

   For the case of non-square matrices, see e.g. [97, 98, 95, 96]

3. Determinant

   As in the case of system solving, LU factorization can be used to compute the determinant in the dense case. In the sparse case, the computation of the characteristic polynomial is necessary. The complexity is thus $O(m^\omega)$ or $O(m\Omega)$ for dense and sparse matrix respectively.

In [35, 36, 109] the authors show how to incorporate fast matrix multiplication in exact algorithms and evaluate the computation of LU factorization and other decompositions experimentally. Results suggest that sub-cubic complexities are reached for matrices of average size matrices.

To sum up, for the problems considered in this chapter, $It(A, p)$ is $O(m^\omega)$ or $O(m\Omega)$ for a dense or sparse general matrix. For structured matrices Hankel/Toeplitz type the cost can be significantly smaller e.g. $O(n^2 \log^2(n))$ for linear solving and determinant computation, see [104, 106, 105, 107].

10.4 Reconstruction in CRA

Let $p = \max(P)$ be the maximal prime used in Alg. 10.1.1. Let us define the cost of *reconstruct* procedure at iteration t as a function $rec(p, t)$ and the overall time of reconstruction $Rec(p, t) = \sum_{i=0}^{t} rec(p, t)$. First, we will consider $rec(p, t)$ and $Rec(p, t)$ in the scalar case. In Sec. 10.4.3 the modifications for the vector case are considered.

Proposition 10.4.1 (Reconstruction in CRA) Let x be an integer. Let $t, s > 0$ and $p_0, \ldots, p_{t+s}$ be distinct primes. Let M_i denote the product $\prod_{j=0}^{i} p_j$ and $M'_s = \prod_{j=t+1}^{t+s} p_j$.

Let $x_t \in \mathbb{Z}_{M_t}, y_s \in \mathbb{Z}_{M'_s}$ be such that $x_t = x \mod M_t$ and $y_s = x \mod M'_s$. Suppose that $M_t > M'_s$ and $2 \nmid M'_s$. Then the value of $x_{t+s} \in \mathbb{Z}_{M_{t+s}}$, $x_{t+s} = x \mod M_{t+s}$ can be computed by the formula

$$x_{t+s} = x_t + ((y_s - x_t)_{M'_s}(M_t^{-1})_{M'_s})_{M'_s} \cdot M_t$$

where $(-)_{M'_s}$ denote a computation modulo M'_s. We assume that all modular operations are done in the symmetric range i.e. in $\mathbb{Z}_q = \{-\lceil \frac{q}{2} \rceil + 1, \ldots, \lfloor \frac{q}{2} \rfloor\}, q \in \mathbb{N}$.

PROOF One can easily check that

$$x'_{t+s} = x_t + ((y_s - x_t)_{M'_s}(M_t^{-1})_{M'_s})_{M'_s} \cdot M_t \tag{10.1}$$

is equal to y_s mod M'_s and x_t mod M_t. Therefore, by the Chinese Remaindering Theorem x'_{t+s} equals x modulo $M_{t+s} = M_t M'_s$. We have

$$\begin{aligned} -\lceil \frac{M_t}{2} \rceil + 1 \leq x_t \leq \lfloor \frac{M_t}{2} \rfloor \\ (-\lceil \frac{M'_s}{2} \rceil + 1)M_t \leq ((y_s - x_t)_{M'_s}(M_t^{-1})_{M'_s})_{M'_s} \cdot M_t \leq \lfloor \frac{M'_s}{2} \rfloor M_t \end{aligned}$$

Thus

$$-\lceil \frac{M_t}{2} \rceil + 1 + (-\lceil \frac{M'_s}{2} \rceil + 1)M_t \leq x'_{t+s} \leq \lfloor \frac{M_t}{2} \rfloor + \lfloor \frac{M'_s}{2} \rfloor M_t.$$

After evaluation we obtain

$$-\lceil \frac{M_{t+s}}{2} \rceil + 1 \leq x'_{t+s} \leq \lfloor \frac{M_{t+s}}{2} \rfloor.$$

Therefore $x_{t+s} = x'_{t+s} \in \mathbb{Z}_{M_{t+s}}$.

Remark 10.4.2 As a modification to Prop. 10.4.1 one may also consider the situation where $-\lceil \frac{M_t}{2} \rceil + 1 \leq x_t \leq \lfloor \frac{M_t}{2} \rfloor$ (i.e. x is evaluated modulo M_t in the symmetric range) but all results modulo M'_s are obtained in the positive range, as used in many implementations of word-size modular fields. In this case from Eq. (10.1) we have

$$-\lceil \frac{M_t}{2} \rceil + 1 \leq x'_{t+s} \leq \lfloor \frac{M_t}{2} \rfloor + (M'_s - 1)M_t.$$

We obtain

$$-\lceil \frac{M_{t+s}}{2} \rceil + 1 \leq x'_{t+s} \leq M_{t+s} - \lceil \frac{M_t}{2} \rceil.$$

In order to obtain x_{t+s} equal to $x \mod M_{t+s}$ in the symmetric range we eventually have to correct x'_{t+s} by eventually subtracting M_{t+s}.

The reconstruction proposed in Prop. 10.4.1 is the optimal use of Chinese Remaindering Theorem in the case of CR Algorithm. It allows directly to compute x_{t+s} in the required range, without using mod M_{t+s} operations. Let us evaluate the cost of the reconstruction.

Proposition 10.4.3 (Cost of Reconstruction) Let $p, t, s > 0$ be integers, $t > s$. Let x_t, y_s, M_t, M'_s be integers, such that

$$\begin{aligned} 0 < M_t &< p^{t+1}, \\ -\lceil \frac{M_t}{2} \rceil + 1 \leq x_t &\leq \lfloor \frac{M_t}{2} \rfloor, \\ 0 < M'_s &< p^s, \\ -\lceil \frac{M'_s}{2} \rceil + 1 \leq y_s &\leq \lfloor \frac{M'_s}{2} \rfloor. \end{aligned}$$

The cost of computing x_{t+s} by Eq. (10.4.1) is

$$O^{\approx}((t+s)\log(t+s)\log(p) + (t+s)\log(p)\log(\log(p)) + s\log(p)(\log^2(s) + \log^2(\log(p)))).$$

bit operations.

PROOF Let x'_{t+s} be defined as in Eq. (10.1). Let us consider all operations involved in the computation of x'_{t+s}.

$$x'_{t+s} = x_t +^{(1)} ((y_s - x_t)^{(2)}_{M'_s}(M_t^{-1})^{(3)}_{M'_s})^{(4)}_{M'_s} \cdot^{(5)} M_t$$

(1) is an addition of two integers bounded in absolute value by $\frac{p^{t+1}}{2}$ and $\frac{p^{t+s+1}}{2}$.

(2) is a subtraction modulo M'_s; additionally $x_t \mod M'_s$ has to be computed.

(3) is an inverse modulo M'_s; additionally $M_t \mod M'_s$ has to be computed.

(4) is a multiplication modulo M'_s

(5) is a multiplication of two integers bounded in absolute value by $\frac{p^s}{2}$ and p^{t+1}.

The cost of operations is as follows. By $M(x)$ we denote the cost of multiplication of two x-bit integers. $M(x)$ is equal $O(x^2)$ for the classic multiplication and the best complexity by the fast multiplication algorithm of Schönhage-Strassen is $M(x) = O(x\log(x)\log(\log(x)))$, see [70, Tab. 8.6].

(1) $O((t+s)\log(p))$ for addition of integers,

(2) $O(M(t\log(p)))$ for image computation and $O(s\log(p))$ for subtraction,

(3) $O(M(t\log(p)))$ for image computation and $O(M(s\log(p))\log(s\log(p)))$ for the inverse,

(4) $O(M(s\log(p)))$ for modular multiplication,

(5) $O(M((t+s)\log(p)))$ for integer multiplication.

By using fast integer multiplication we may obtain that the cost of computation of x'_{t+s} is

$$\begin{aligned}&O((t+s)\log(p)\log((t+s)\log(p))\log(\log((t+s)\log(p)))\\&\qquad + s\log(p)\log^2(s\log(p))\log(\log(s\log(p))))\\&= O^{\approx}((t+s)\log(t+s)\log(p) + (t+s)\log(p)\log(\log(p)) + s\log(p)(\log^2(s) + \log^2(\log(p)))).\end{aligned}$$

in the *doubly soft Oh* notation. The eventual additional cost of computing x_t from x'_t is $O((t+s)\log(p))$ and does not change the asymptotic complexity.

10.4.1 Incremental Reconstruction

The ET strategy allows for optimal termination if the reconstruction is performed at each addition of a new modulus i.e. when $s = 1$ in Prop. 10.4.1. This can be done if the cost of the reconstruction given by Prop. 10.4.1 is negligible compared to the cost $Im(A,p) + It(A,p)$. Let us consider the notation of Prop. 10.4.1. We may implement function $reconstruct(x_t, y_0, \ldots, y_t, p_0, \ldots, p_t)$ recursively, as $reconstruct(x_t, x_{t-1}, y_t, M_{t-1}, p_t)$. The result x_t of this function is given by Eq. (10.4.1) of Prop. 10.4.1 with $s = 1$. By Prop. 10.4.3, the cost of $rec(p,t)$ is $O(M(t)) = O^{\approx}(t\log(t))$, if $p \in O(1)$. The cost $Rec(p,t)$ of all calls to *reconstruct* needed to compute x_t in t iterations is thus $O^{\approx}(t^2\log(t))$.

10.4.2 Delayed Reconstruction

In the case of certified CRA with $\epsilon = 0$ it is not necessary to reconstruct the result at every step. Instead, the modular results $x_0, \ldots, x_t$ should be stored and fast reconstruction can be performed at the end of the algorithm.

In [70, Sec. 10.3] fast integer reconstruction of the final result from t modular results is described. Using the Lagrangian interpolation formula and a tree-like evaluation scheme, the cost $Rec(p,t)$ can be reduced to $O(t\log^2(t)\log\log(t))$ if $p \in O(1)$.

Delayed reconstruction can also be carried out by means of a radix list. This is a classic approach for computing the amortized cost of a multistep incremental operation, see e.g. [23, Ch. 17]. The approach is characterized by the following properties.

1. At iteration $t, t = 0, 1, \ldots,$ partial results are stored in a list of length $l = \lfloor \log(t+1) \rfloor + 1$ in such a way that for $t + 1 = \sum_{i=0}^{l-1} c_i 2^i$, where $c_i \in \{0,1\}$, only cells indexed by i s.t. $c_i = 1$ are occupied.

2. The result $C[i,t]$ stored in cell i at iteration t is bounded by $\frac{p^{2^i}}{2}$. $C[l-1,t]$ is equal to $x \mod p_0 \ldots p_{2^{l-1}-1}$ and for $i = 0, \ldots, l-1$ s.t. $c[i] = 1$

$$C[i,t] = x \mod p_{\sum_{j=i+1}^{l-1} c_j 2^j} \cdots p_{-1+\sum_{j=i}^{l-1} c_j 2^j}.$$

3. In iteration $t+1$, result y_{t+1} is inserted in the list as follows. If $C[0,t]$ is free, then $C[0,t+1] = y_{t+1}$ and $C[i,t+1] = C[i,t]$ for $i = 1,2\ldots$. Otherwise, let $C[i,t]$ be the free cell of lower index. Then $z_0 = y_{t+1}, z_1 = x \mod p_t p_{t-1}$, $z_2 = x \mod p_t p_{t-1} p_{t-2} p_{t-3}, \ldots, z_i = x \mod p_t \ldots p_{t-2^i+1}$ can gradually be reconstructed from z_{i-1} and $C[i-1,t]$. We set $C[0,t+1] = \cdots = C[i-1,t+1] = empty$, $C[i,t+1] = z_i$ and $C[i+1,t+1] = C[i+1,t], \ldots, C[l-1,t+1] = C[l-1,t]$. By Prop. 10.4.3 (notice, that $s = t$) the cost of the reconstruction in this step is $O^{\approx}(\sum_{j=0}^{i-1} 2^j \log^2(2^j)) = O^{\approx}(2^i \log^2(2^i))$.

4. The amortized cost $Rec(p, 2^{l-1}-1)$ is $O^{\approx}(\sum_{i=0}^{l-2} 2^{l-2-i} 2^i \log^2(2^i)) = O^{\approx}(2^l l^3)$, as in iterations $0,1,\ldots,2^{l-1}-1$, cell i is the first cell free 2^{l-2-i} times; thus, $R(p,t) = O^{\approx}(t \log^3(t)) = O^{\sim}(t)$.

10.4.3 Reconstruction in Vector Case

The reconstruction of a vector $X(A)$ of size s takes s times the reconstruction of a scalar, which makes it prohibitive to reconstruct the whole vector at each iterations. Instead, the reconstruction of the vector should be delayed until the CR loop terminates. Then, vector reconstruction is performed once by the method of Sec. 10.4.2. The question remains as how to decide on the early termination.

In e.g. [4] the following method is proposed. Let r be a random vector of size s. CR Algorithm can be run concurrently in order to compute $X(A)$ and the scalar product $r \cdot X(A)$. The value $r \cdot X(A)$ should be computed in the standard way for scalar CRA and all vector entries at primes $p_0, p_1 \ldots$ should be stored (e.g. in a radix list). When termination is detected for $r \cdot X(A)$ at iteration t, $X(A) \mod \prod_{i=0}^{t} p_i$ is reconstructed and checked for early termination. In case of failure, CRA is recalculated for a new random vector r' and resumed. In the case when the entries $X(A)$ alternate in sign, several random vector might be required.

An alternative method would be to check for termination of only one entry of $X(A)$ at a time. If termination is detected, another entry is reconstructed and checked for termination, until all entries are reconstructed with a sufficient probability, see Sec. 10.6.3.

10.5 Getting Rational Result

In the case when $X(A)$ is not integer, reconstruction by *reconstruct* of $X(A) \mod \prod_{i=0}^{t} p_i$ will generally return a different answer for each t. In this case, rational reconstruction is necessary in order to detect early termination, see Sec. 10.6.2. In Ch. 11 we discuss the algorithms for rational reconstruction in details. In general, rational reconstruction is slower than integer reconstruction. For each iteration $i = 0, \ldots, t$ it is an independent problem and 'incremental' reconstruction as in the case of integer reconstruction in Sec. 10.4.1 cannot be envisaged.

Let $rat(p,t)$ denote the cost of *get_rational* in the tth iteration, and $Rat(p,t)$ denote the overall time of rational reconstruction in iterations $0,1,\ldots,t$. By fast rational reconstruction algorithm of [131], $rat(p,t)$ is asymptotically equal to $O^{\approx}(t\log^2(t))$ which is comparable to $Rec(p,t)$ by the delayed reconstruction of [70, Sec. 10.3], see Sec. 10.4.2.

Due to the cost of rational reconstruction, a scheduling has to be introduced by *reconstruct* returning **false**. Indeed, in [15] the authors suggest that rational reconstruction could be performed only in iterations numbered $1,4,9,16,25$ etc. Others approaches consider using radix lists and performing full (integer) reconstruction only on iterations $1,2,4,8,16,32$,etc. We return to this problem in Sec. 10.7.

10.6 Termination in CR Algorithm

In this section we will give the stopping condition for certified and early terminated CRA loop by *terminated*. We start by the scalar integer case. Then in Sec. 10.6.3 and Sec. 10.6.2 we discuss the modifications in the vector case and in the case of rational computation.

Here, we propose an output dependent early termination, which does not only depend on the static parameters of the algorithm such as the bound H, but also involves the size of output x_t and modulus $\prod_{i=0}^{t} p_i$ at iteration t. In Sec. 10.6.4 we discuss the case when the bound is not known i.e. $H=\infty$.

The problem of early termination of the CRA algorithm was treated in [75, 38, 39, 4]. Lemmas 10.6.1,10.6.3 summarize the known results.

10.6.1 Termination in Scalar Case

Lemma 10.6.1 (Certified Termination in CRA) *Let $x\in\mathbb{Z}$. Let $l\in\mathbb{R}, l>1$. Suppose that distinct primes p_i greater than l are sampled from a set of primes P. Let $t\in\mathbb{N}$ and let x_t be the value of x modulo $p_0\cdots p_t$ computed in the symmetric range. We have that x equals x_t if*

$$2|x| < \prod_{i=0}^{t} p_i \tag{10.2}$$

and consequently,

$$x_t = x,\ \textit{for } t \geq N = \begin{cases} \lceil \log_l(|x|)\rceil & \textit{if } x\neq 0 \\ 0 & \textit{if } x = 0 \end{cases}. \tag{10.3}$$

PROOF In the case when x is positive, the prove is given in [38, 39]. To prove Eq. (10.2),(10.3), notice that

$$-\lceil\frac{p_0\cdots p_t}{2}\rceil + 1 \leq x_t < \lfloor\frac{p_0\cdots p_t}{2}\rfloor$$

and $x_t = x \mod p_0 \cdots p_t$. If $2|x| < \prod_{i=0}^{t} p_i$ we have

$$-\lceil \frac{p_0 \cdots p_t}{2} \rceil + 1 \leq -\lceil \frac{2|x|+1}{2} \rceil + 1 \leq x$$

and

$$x \leq \lfloor \frac{2|x|}{2} \rfloor \leq \lfloor \frac{p_0 \cdots p_t}{2} \rfloor.$$

Therefore $x \mod \prod_{i=0}^{t} p_i$ equals x.

With l being the lower bound for p_i, in order to fulfill Eq. (10.2) it suffices that $l^{t+1} \geq 2|x|$, which transforms to $t \geq \log_l(2|x|) - 1$ when $x \neq 0$. To fulfill this equation it suffices that $t \geq \lceil \log_l(|x|) \rceil$.

Corollary 10.6.2 (Certified CRA cf. [39]) Let $x \in \mathbb{Z}$, $H \in \mathbb{R}$ such that $|x| \leq H$. Let $l \in \mathbb{R}, l > 1$ and let $N = \lceil \log_l(H) \rceil$. Define P as a finite set of primes greater than l, such that $|P| > N$. Suppose that *get_prime* picks up a prime from P. Let us define $terminated(0, P, H)$ as a condition

$$\text{if } 2H < \prod_{i=0}^{t} p_i \text{ then } break;$$

Then the $terminated(0, P, H)$ is well defined. Moreover, the **repeat** loop of Alg. 10.1.1 for $x = X(A)$ performs ar most $N + 1$ steps.

PROOF In Eq. (10.2) let us replace $|x|$ by the bound H. As $2|x| < 2H$, by Eq. (10.2), x_t is equal x. Therefore $terminated(0, P, H)$ is defined as required. By Eq. (10.3), at most $p_0, \ldots, p_N$,should be taken at most, i.e. $N + 1$ primes are sufficient.

The following lemma gives the basis for constructing $terminated(\epsilon, P, H)$ for $\epsilon > 0$.

Lemma 10.6.3 (Early Termination in CRA) *Let $x \in \mathbb{Z}$, $H \in \mathbb{R}$ such that $|x| \leq H$. Let $l \in \mathbb{R}, l > 1$ and let $N = \lceil \log_l(H) \rceil$.*

Suppose that distinct primes $p_0, \ldots, p_N$ greater than l are uniformly and randomly sampled from a set of primes P, where $|P| > 2\log_l(H)$. Denote by $\mathcal{P}$ the distribution of finite sequences $p_0, \ldots, p_N \in P^N$ such that $p_i \neq p_j$ for $i, j = 0, \ldots, N, i \neq j$.

Let x_t be the value of x modulo $p_0 \cdots p_t$ computed in the symmetric range and let $0 < \epsilon < 1$. We have:

(i) suppose that $t, k \in \mathbb{N}, t + k \leq N$;
if $x_t = x_{t+1} = \cdots = x_{t+k}$ and $\frac{R'(R'-1)\ldots(R'-k+1)}{(|P|-t-1)\ldots(|P|-t-k)} < \epsilon$, where

$$R' = \lfloor \log_l \left(\frac{H + |x_t|}{p_0 p_1 \ldots p_t} \right) \rfloor,$$

then $\mathcal{P}(x_t \neq x) < \epsilon$.

(ii) suppose that $t, k \in \mathbb{N}, t + k \leq N$;

suppose that $x_t = x_{t+1} = \cdots = x_{t+k}$ and $k \geq k_{max}$, where

$$k_{max} = \lceil \frac{\log(1/\epsilon)}{\log(P') - \log(\log_l(H))} \rceil, \tag{10.4}$$

and $P' = |P| - \lfloor \log_l(H) \rfloor$. Then $\mathcal{P}(x_t \neq x) < \epsilon$.

PROOF Let us suppose that $x = x_t \mod \prod_{i=0}^{t} p_i$ and let us estimate the number of primes p_{t+1} coprime with $\prod_{i=0}^{t} p_i$ such that $x_t = x_{t+1}$.

There exists $K \in \mathbb{Z}$, such that

$$x = x_t + K p_0 \cdots p_t \tag{10.5}$$

and it suffices to estimate the number of primes greater than l dividing K. We have

$$K = \frac{x - x_t}{p_0 \cdots p_t}.$$

As the primes in P are greater than l, the number of primes is bounded by

$$R = \lfloor \log_l \left(\frac{|x - x_t|}{p_0 \cdots p_t} \right) \rfloor,$$

which is less than or equal to

$$R' = \lfloor \log_l \left(\frac{H + |x_t|}{p_0 p_1 \ldots p_t} \right) \rfloor.$$

For (i) we notice that k primes $p_{t+1}, \ldots p_{t+k}$ have to be chosen from $P \setminus \{p_0, \ldots p_t\}$ among primes dividing K. There are at most R' of those and the probability is bounded by

$$\frac{\binom{R'}{k}}{\binom{|P|-(t+1)}{k}} = \frac{R'(R'-1)\ldots(R'-k+1)}{(|P|-t-1)\ldots(|P|-t-k)} < \epsilon. \tag{10.6}$$

For (ii) suppose that condition $\prod_{i=0}^{t+k} p_i > 2H$ does not hold which implies that $t + k < \log_l(H) \leq N$. In the opposite case, $x_{t+k} = x$ by Lem. 10.6.1 and we are done. We want to find k such that

$$\frac{R'(R'-1)\ldots(R'-k+1)}{(|P|-t-1)\ldots(|P|-t-k)} < \epsilon,$$

where $R' = \lfloor \log_l \left(\frac{H+|x_t|}{p_0 p_1 \ldots p_t} \right) \rfloor$. We have that $|P| - t - k$ is greater than or equal

$$|P| - t - k \geq |P| - \lfloor \log_l(H) \rfloor = P'.$$

Therefore

$$\frac{R'(R'-1)\dots(R'-k+1)}{(|P|-t-1)\dots(|P|-t-k)} \leq \left(\frac{R'}{P'}\right)^k.$$

Moreover, the fraction $\frac{R'}{P'}$ is less than 1. Indeed,

$$R' = \lfloor \log_l \left(\frac{H+|x_t|}{p_0 p_1 \dots p_t}\right) \rfloor \leq \lfloor \log_l \left(\frac{2H}{l^{t+1}}\right) \rfloor \leq \log_l(H) \leq 2\log_l(H) - \lfloor \log_l(H) \rfloor < P'.$$

Solving for k the inequality $\left(\frac{R'}{P'}\right)^k < \epsilon$ gives

$$k > \frac{\log(\epsilon)}{\log\left(\frac{R'}{P'}\right)} = \frac{\log\left(\frac{1}{\epsilon}\right)}{\log(P') - \log(R')} \geq \frac{\log\left(\frac{1}{\epsilon}\right)}{\log(P') - \log(\log_l(H))}.$$

Therefore we may take

$$k_{max} = \lceil \frac{\log(1/\epsilon)}{\log(P') - \log(\log_l(H))} \rceil.$$

Corollary 10.6.4 (Early terminated in CRA) Let $x \in \mathbb{Z}$, $H \in \mathbb{R}$ such that $|x| \leq H$. Let $l \in \mathbb{R}, l > 1$ and let $N = \lceil \log_l(H) \rceil$. Define P as a finite set of primes greater than l, such that $|P| > 2\log_l(H)$. Suppose that *get_prime* randomly and uniformly picks up a prime from P, and let $P' = |P| - \lfloor \log_l(H) \rfloor$. Denote by $\mathcal{P}$ the distribution of finite sequences $p_0, \dots, p_N \in P^N$ such that $p_i \neq p_j$ for $i, j = 0, \dots, N, i \neq j$.

For $i = 0, \dots, N$, let x_i be the value of x modulo $p_0 \cdots p_i$ computed in the symmetric range. Let us consider the tth step of the CRA loop, see Alg. 10.1.1. Let $0 < \epsilon < 1$ and let $k_{max} = \lceil \frac{\log(1/\epsilon)}{\log(P') - \log(\log_l(H))} \rceil$.

We may define $terminated(\epsilon, P, H)$ as follows: The last **if** may be replaced by the output-

---

if $2H < \prod_{i=0}^{t} p_i$ **then** break;
$k = \max\{s : x_{t-s} = \dots = x_t\}$;
if $k \geq k_{max}$ **then** break;

---

dependent condition: The cost of calling $terminated(\epsilon, P, H)$ in Alg. 10.1.1 if $O(1)$ at

---

$R' = \lfloor \log_l \frac{H+|x_t|}{p_0 p_1 \dots p_{t-k}} \rfloor$
if $\frac{R'(R'-1)\dots(R'-k+1)}{(|P|+k)\dots(|P|+1)} < \epsilon$ **then** break;

---

each iteration t.

PROOF This definition of $terminated(\epsilon, P, H)$ ensures that Lemma 10.6.3 can be applied. The two variants of termination corresponds to (i) and (ii) cases of the lemma. Notice, that the size of P is updated in the algorithm by *get_prime*. To estimate the cost of $termianted(\epsilon, P, H)$ notice, that the value of maximum $k(t)$ at tth iteration is $k(t-1)+1$ if $x_t = x_{t-1}$ or 0 otherwise, therefore it may be computed at $O(1)$ cost, if the results from previous iteration are available.

Remark 10.6.5 Function *reconstruct* and *terminated* might be run for a set of several preconditoners $\{D_1, \ldots, D_i\}$ at the same time. Lem. 10.6.3 ensures $1-\epsilon$ probability of correctness as soon as early termination happens for one of the preconditioners.

10.6.2 Early Termination in Rational Case

In Ch. 11 we compare the early termination schemes of P.S. Wang [129] and Monagan [90], which can be applied to the case of p-adic lifting. Yet in the case of CR Algorithm another termination strategy is possible. One can ignore the assumptions on which the early termination schemes of [129, 90] rely and accelerate the convergence rate, based on the concept of preconditioned CRA loop.

Lem. 10.6.3 induces the following corollary.

Corollary 10.6.6 (Early termination in rational CRA) Let $x = \frac{a}{b} \in \mathbb{Q}$, $H \in \mathbb{R}$ such that $|x|_r \leq H$. Let $l \in \mathbb{R}, l > 1$.

Suppose that distinct primes $p_0, p_1, \ldots$ greater than l are uniformly and randomly sampled from a finite set of primes P. Denote by $\mathcal{P}$ the distribution of permutations of P i.e. finite sequences $p_0, p_1, \ldots, p_{|P|}$ such that $p_i \in P, p_i \neq p_j$ for $i, j = 0, \ldots, N, i \neq j$.

Let $x_t = \frac{a_t}{b_t}, a_t, b_t \in \mathbb{Z}, b_t > 0$ be the value of x modulo $p_0 \cdots p_t$ computed by rational reconstruction and let $0 < \epsilon < 1$. We have:

(i) if $x_t = x_{t+1} = \cdots = x_{t+k}$ and $\frac{R'(R'-1)\ldots(R'-k+1)}{(|P|-t-1)\ldots(|P|-t-k)} < \epsilon$, where

$$R' = \lfloor \log_l \left(\frac{H(|a_t| + b_t)}{p_0 p_1 \cdots p_t} \right) \rfloor,$$

then $\mathcal{P}(x_t \neq x) < \epsilon$.

(ii) suppose that $t, k \in \mathbb{N}$ are such that $x_t = x_{t+1} = \cdots = x_{t+k}$ and $|P| - t - k > \log(\log_l(H^2))$;

suppose that $k \geq k_{max}$, where

$$k_{max} = \lceil \frac{\log(1/\epsilon)}{\log(|P| - t - k) - \log(\log_l(H^2))} \rceil,$$

then $\mathcal{P}(x_t \neq x) < \epsilon$.

PROOF The idea of this proof is the same as in Lem. 10.6.3.

Let us suppose that $\frac{a}{b} = \frac{a_t}{b_t} \mod \prod_{i=0}^{t} p_i$ and let us estimate the number of primes p_{t+1} coprime with $\prod_{i=0}^{t} p_i$ such that $a_t = a_{t+1}, b_t = b_{t+1}$.

There exists $K \in \mathbb{Z}$, such that

$$ab_t = ba_t + Kp_0 \cdots p_t \tag{10.7}$$

and it suffices to estimate the number of primes greater than l dividing K. We have

$$K = \frac{ab_t - ba_t}{p_0 \cdots p_t}.$$

As the primes in P are greater than l, the number of primes is bounded by

$$R' = \lfloor \log_l \left(\frac{H(|a_t| + b_t)}{p_0 p_1 \ldots p_t} \right) \rfloor.$$

As in Lem. 10.6.3 we notice that k primes $p_{t+1}, \ldots p_{t+k}$ have to be chosen from $P \setminus \{p_0, \ldots p_t\}$ among primes dividing K. There are at most R' of those and the probability is bounded by

$$\frac{\binom{R'}{k}}{\binom{|P|-(t+1)}{k}} = \frac{R'(R'-1)\ldots(R'-k+1)}{(|P|-t-1)\ldots(|P|-t-k)} < \epsilon.$$

For (ii), the proof of Lem. 10.6.3(ii) carries on.

Cor. 10.6.6 ensures that whenever a result reoccurs, the probability of its correctness can be treated as in the case of integer CRA. Now, suppose that $d_t = \frac{a_t}{b_t}$ is the reconstructed result in the tth iteration. Let ϵ be the required error tolerance of the rational reconstruction scheme. Let us consider a procedure which solves the problem modulo a prime. Let us set the preconditioner $D = \frac{b}{a}$ and let us run preconditioned CRA scheme for the preconditioned problem of computing $DX(A)$. Supposing that the result $\frac{a}{b}$ is correct, CRA will terminate with result 1. If d_t is different from $X(A)$ then, in general, $DX(A)$ is rational, and no early termination occurs. Lemma 10.6.3 and Cor. 10.6.6 tells us that a small number of repetitions of 1 is needed to confirm the result with probability $1 - \epsilon$ as long as the set of primes is large enough. With each repetition the probability of correctness is improved, and the convergence is fast.

10.6.3 Early Termination in Vector Case

In Sec. 10.4.3 we describe two ways to anticipate early termination of the CRA for the computation of vector $X(A)$. Namely, a scalar CRA can be run in order to compute $r \cdot X(A)$, where r is a random vector, or to compute $X(A)_1$, i.e one entry of $X(A)$. In both cases, even the successful computation of $r \cdot X(A)$ or $X(A)_1$ resp. at iteration t does not guarantee that $X(A)$ can be reconstructed in the same iteration.

Thus, the early termination with probability of success $1 - \epsilon$ is only guaranteed if all entries of the vector $X(A)$ are reconstructed with probabilities of error $\epsilon_1, \ldots, \epsilon_s$ such that $\sum_{i=0}^{s} \epsilon_i < \epsilon$. For each entry, ϵ_i can be computed by counting the number of repetitions and evaluating Eq. (10.6), see also [39, Lem. 3.1]. The **repeat** loop of Alg. 10.1.1 must continue until $\sum_{i=0}^{s} \epsilon_i < \epsilon$.

Suppose that $X(A)$ is a vector of size s. In the worst case, this requires that each entry is computed with the probability of error at most $1 - \frac{\epsilon}{s}$. By Lem. 10.6.3(ii), this means that a

$$\lceil \frac{\log(s/\epsilon)}{\log(P') - \log(\log_l(H))} \rceil \in O(\log(\frac{s}{\epsilon})) \tag{10.8}$$

repetitions have to occur for each entry of the vector.

10.6.4 Extremely High Number of Iterations

Apart from the growing cost of reconstruction, the enormous number of steps forces us to provide a large set P of possible primes which affects the choice of l, and thus, the size of primes. Suppose that P consist of primes of the same bit size i.e. $l < p < 2l$ for $p \in P$. The number of primes less than $2l$ is given by the prime counting function $\pi(2l)$, which is asymptotically $\frac{2l}{\ln(2l)}$. Therefore $\log(2l)$ should be equal $O(\ln(H))$, where H is the bound for $X(A) \in \mathbb{Z}$. In the worst case for large H, this means that using all word-size primes may not be sufficient for the CRA to terminate. Although modular algorithms are generally not optimized for the performance on bigger primes, their asymptotic complexity change only by a $\log(2l)$ factor.

The case, when the bound is not known, i.e. the value of H is ∞, requires a slightly different approach. The difficulty lies in the fact, that the size of set P, and thus, l, cannot be computed beforehand. With no other information, we may assume that $P = P_1$ consists of primes p of the same bit size i.e. $l < p < 2l$, where $l = 2^k$ for $k \in \mathbb{N}$. Yet, we may use all primes and the same result $x_t = x_{t+1}$ might not reoccur.

Still, a new set $P = P_2$ of primes of size $2l < p < 4l$ and etc. can be constructed. Current results are kept. When early termination happens, i.e. $x_t = x_{t-1} = \cdots = x_{t+k}$, the output of the algorithm may be characterized as a follows.

Lemma 10.6.7 (Probability of Unbounded CRA) *Let $x \in \mathbb{Z}$, $M \in \mathbb{N}, N > 0$, $l \in \mathbb{R}, l > 1$ and let P be a set of primes p such that $l < p < 2l$ for $p \in P$. Let $p_0, \ldots, p_N$ be a random permutation of primes from P. Denote by $\mathcal{P}$ the distribution of the permutations.*

Let x_t be the value of x modulo $Mp_0 \cdots p_t$ computed in the symmetric range. Suppose that $x_t = \cdots = x_{t+k}$. Let $\alpha \in \mathbb{R}$ be such that $\lfloor \log_l(\frac{\alpha|x_t|}{\prod_{i=0}^{t} p_i}) \rfloor < |P| - t - k$. Then either $|x| > (\alpha - 1)|x_t|$ or the probability that $x = x_t$ is at least $1 - \left(\frac{\lfloor \log_l(\frac{\alpha|x_t|}{\prod_{i=0}^{t} p_i}) \rfloor}{|P|-t-k} \right)^k$.

PROOF For the proof, repeat the reasoning of 10.6.3 (i) and (ii).

10.7 In Search for Ideal Reconstruction Strategy

The idea of early termination is to stop the reconstruction shortly after the minimal number of steps has been reached. Suppose $s \in \mathbb{N}$ and the searched value $X(A)$ is a vector

of size s. If $X(A) \in \mathbb{Z}$ we define $\|X(A)\|$ as the maximal absolute value of its entries. Then Eq. (10.3) says that the minimum number of $N = O(\log_l(\|X(A)\|))$ is needed, plus additional $\log(s)k_{max}$ steps to confirm early termination, see Eq. (10.4) and (10.8) for the maximal number of steps in the scalar and vector case, depending on ϵ.

Analogously, if $X(A) \in \mathbb{Q}$, let us define $D(X(A))$ as the maximal denominator and $N(X(A))$ as the maximal numerator of $|X(A)|$. Then, the minimal number of steps is about $N = O(\log_l(D(X(A)) \cdot N(X(A))))$, see [70, Sec. 5.10] plus the additional cost due to termination detection, see Sec. 11.3 for more information.

Without loss of generality we will assume that $\log(s)k_{max} \in O(N)$. In the ideal situation, the cost of early terminated CRA, in the case where N steps suffice should be $O(N(Im(A,p) + It(A,p)) + Rec(p,N) + Rat(p,N)$. The cost $O(N(Im(A,p) + It(A,p))$ is inevitable whereas the cost $Rec(p,N) + Rat(p,N)$ depends on how often we perform the reconstruction.

For $X(A) \in \mathbb{Z}^s$, the lower bound on cost of reconstruction is $O^{\approx}(sN\log^2(N))$, which is the cost of fast delayed reconstruction and the upper bound is $O^{\approx}(N^2\log(N) + sN\log(N))$, which is the cost of incremental reconstruction, see Sec. 10.4.

For $X(A) \in \mathbb{Q}^s$ an additional cost of rational reconstruction has to be included. The lower cost bound is asymptotically $O^{\approx}(sN\log^2(N))$ using fast arithmetics and $O(sN^2)$ in standard arithmetics, which corresponds to performing one (successful) rational reconstruction in the Nth iteration only.

As long as the cost of integer and rational reconstruction at step t is smaller than the cost of $Im(A,p) + It(A,p)$, the reconstruction can be performed at each step without any asymptotic handicap. This results with optimal termination and complexity $O(N(Im(A,p) + It(A,p)))$. Yet in general, this cannot be assumed and a scheduling of integer and rational reconstruction should be envisaged. The scheduling function might be determined a priori, see Sec. 10.5.

Here, we propose another approach which generalizes to the abstract situation when the cost of reconstruction dominates the cost of *iteration* and *image*. Our introspective scheduling performs the reconstruction as often as possible in order to enable optimal early termination. We will require that at most $\tilde{N} = O^{\sim}(N)$ iterations are performed, where N is the minimal number of iterations needed. At the same time, introspective reconstruction scheduling algorithm balances the cost of all reconstructions $Rec(p,\tilde{N}) + Rat(p,\tilde{N})$ in such a way that at the end, the cost of CR Algorithm is $O^{\approx}(\tilde{N}(Im(A,p) + It(A,p)) + s\tilde{N}\log^2(\tilde{N}))$.

At the same time, the experiments in Ch. 14.3 show that indeed, scheduling of rational reconstruction can improve the performance in the case of rational computation.

Let us analyze Alg. 10.1.1. The idea is to measure the time $time_1(t) = t(Im(a,p) + It(A,p))$ of operations *image* and *iteration* computed in steps $0,1,2,\ldots,t-1,t$ and compare it to $time_2(t-1)$ of *reconstruct* and *get_rational* computed in iterations $0,1,2,\ldots,t-1$. If $time_1(t) > time_2(t-1)$, *reconstruct* and *get_rational* are computed so that x_t is known. This allows us to check for termination.

Let $\tilde{N}$ be the iteration in which early termination was reached and let N_1, N_2 be the iterations in which *reconstruct* and *get_rational* were computed before. To see that we have the required complexity, it suffices to notice that all reconstruction costs except the last one are absorbed in $time_1(\tilde{N}-1) = O(\tilde{N}(Im(A,p)+It(A,p)))$. Then, the additional cost of the last reconstruction is $O^{\approx}(\tilde{N}\log^2(\tilde{N}))$.

Now, it rests to prove that the number of steps $\tilde{N}$ is indeed $O^{\sim}(N)$. The reasoning is as follows. Assuming that early termination was detected in step $\tilde{N}$, this means that $\tilde{N} > N_2 \geq N > N_1$. From the time estimations we may conclude that $N_2 = N_1 + O(\frac{time_2(N_1)-time_2(N_1-1)}{Im(A,p)+It(A,p)})$. Analogously, $\tilde{N} = N_2 + O(\frac{time_2(N_2)-time_2(N_2-1)}{Im(A,p)+It(A,p)})$. This means that $N_2 = O^{\sim}(N_1)$ and $\tilde{N} = O^{\sim}(N_2)$ which allows us to conclude that $\tilde{N} = O^{\sim}(N)$.

10.8 Parallel CRA

In the attempt to provide parallel routines for the LinBox library, providing a parallel CRA loop is of particular interests. A CRA parallel implementation using MPI[2] is available in LinBox. Also, parallel implementation using an adaptive scheduling library Kaapi[3] is included in LinBox. The Work stealing idea of Kaapi, makes integrating the libraries an good place of development, see [52, 26] for some ideas. Optimization of parallel CRA in terms of parallel time and equivalent sequential time, is an on-going work. See also [13] for perspectives of parallelization in LinBox.

[2] e.g. http://www.open-mpi.org/
[3] http://kaapi.gforge.inria.fr/

11

Rational Reconstruction

The goal of this chapter is the comparison of early termination strategies for rational reconstruction and the evaluation of fast rational reconstruction algorithm of [131]. In Sec. 11.1 we shortly introduce the problem and comment on the existing algorithms in Sec. 11.2. In Sec. 11.3 we discuss the applicability of rational reconstruction to p-adic lifting and CR Algorithm.

Then in Sec. 11.4 we solve implementational issues of the algorithm of [131], which includes fixing of an important bug. In Sec. 11.5 we show how to adapt this algorithm to Maximal Quotient Rational Reconstruction of [90]. In Sec. 11.6 experimental comparison of strategies of [129] and [90] is given.

11.1 Problem of Rational Reconstruction

A modular image of a rational number $\frac{a}{b} \bmod p$ can be computed by taking the modular images of a and b and applying the modular division. This fact can be written as

$$\frac{a}{b} = u \bmod p \Leftrightarrow a = bu \bmod p. \tag{11.1}$$

It should be noticed is that the opposite procedure can also be performed. One can reconstruct the fraction $\frac{a}{b}$ where $\gcd(a, b) = 1, b > 0$ from it modular image u. The solution is usually not unique but if we impose a bound q and additionally require that $|a| < \frac{q}{2}$, $b \leq \frac{p}{q}$, then there exists at most one solution, see [70, Sec. 5.10].

This operation is called rational reconstruction and is a core procedure for exact computation where rational results are expected, such as e.g. solving a linear system of integer equations. The concept of (integer) rational reconstruction was first developed by P.S. Wang in [129, 130]. For a description of the problem, see [70, Sec. 5.10].

11.2 Existing Algorithms for Rational Reconstruction

The solution to the rational reconstruction problem can be computed by applying the extended Euclidean algorithm EEA which searches for the gcd of p and u. Let us set

$r_0 = p, r_1 = u, s_0 = 1, t_0 = 0, s_1 = 0, t_1 = 1$. In the $(i-1)$th step of the algorithm, $i > 1$ we determine $r_i = r_{i-2} \mod r_{i-1}$ and s_i, t_i such that $s_i r_0 + t_i r_1 = r_i$. Indeed, every pair $(a, b) = (\mathrm{sgn}(t_i) r_i, |t_i|)$ fulfills Eq. (11.1).

The procedure $Ratrec(a, b, u, p, q, D)$ takes as the input modulus p, $u \in \mathbb{Z}$ and the bounds q and D, and returns a fraction $\frac{a}{b} = u \bmod p$ such that $|a| < q, b < D$ or FAIL if no such solution exists.

The worst case complexity of *Ratrec* is thus the same as for the EEA algorithm i.e. $O\left(k^2\right)$, with $k = \log(p)$ for the classical algorithm. It has been show by [131] that the fast Euclidean algorithm, which have the complexity of $O(k \log^2(k) \log(\log(k)))$, can also be adapted for the rational reconstruction. In Sec. 11.4 we show how to correct and implement the algorithm and comment on its practical applicability. To our knowledge, this is the first implementation of [131]. We are aware of only one other implementation of fast rational reconstruction algorithms, which claims the same complexity in [88]. In [90], the author recalls the algorithm of Steel, which uses fast integer multiplication to speed up the reconstruction.

We should also notice that there exist other algorithms for rational reconstruction, such as [90], which do not need bounds q, D in order to perform rational reconstruction. The algorithm given in [90] is restricted to the classic Euclidean algorithm, thus having the complexity of $O(k^2)$. In [82] the authors shown how to adapt the idea to the polynomial case and mention that the use of fast integer Euclidean algorithm would require more work. In Sec. 11.5 we show that this can indeed be done and show how to adapt the algorithm [131] to the maximal quotient reconstruction of [90].

In Sec. 11.6 we present the experimental result to compare deterministic, P.S. Wang's early terminated and Monagan's early terminated strategies with and without using fast rational reconstruction.

11.3 Rational Reconstruction in CR Algorithm and p-adic Lifting

In many application, the cost of rational reconstruction is usually small compared with the cost of computing u and p. The general scheme for the computation of $x = \frac{a}{b}$ is to recursively compute u_k, M_k, where $u_k = x \mod M_k$ and $M_k = p_1 p_2 \cdots p_k$ (in Rational Chinese Remaindering scheme) or $M_k = p^k$ (in p-adic lifting scheme, see [27]) until $M_k > 2|a|b$ and then to apply the rational reconstruction. If bounds q and D for a and b, we have the classic version of the reconstruction scheme. However, early terminated version of the schemes would be of particular interest in our case.

The idea is to look for the 'best' pair of possible candidates for a and b, where by 'best' we refer to the one that minimizes the product $|a|b$. Two probabilistic algorithms are known that can solve the problem. First, the strategy of [129] point us to run $Ratrecon(a, b, u_k, \sqrt{\frac{M_k}{2}}, \sqrt{\frac{M_k}{2}})$. Second, the Maximal Quotient algorithm of [90] tells us to pick a pair that correspond to the maximal (or sufficiently large) quotient in the

Euclidean algorithm. If this pair is sufficiently 'small' it is a solution to the problem with certain probability, see [90, Alg. MQRR].

Let us shortly compare these approaches. First, for input u, p, if both Wang's and Monagan's algorithms are based on the same Extended Euclidean Algorithm, then Wang's algorithm could take significantly less time to compute, as it only needs to obtain approximately half of quotients. This can be compensated by the fact, that the Maximal Quotient Rational Reconstruction requires less steps in CRA/p-adic scheme. Indeed, if $\frac{a}{b}$ is the result, then early termination strategy of Wang requires p to be greater than $2\max(a^2, b^2)$. On the other hand, the strategy of Monagan requires M_k to be only a moderate number of bits greater than $2|a|b$ on average. Yet, in the worst case, Monagan's scheme in only guaranteed to terminate when $M_k > 9a^2b^2$. The authors claim that this is a rare situation.

A purely heuristic improvement to the Wang's algorithm is to use the bounds $\sqrt{\frac{M_k}{2}\frac{q}{D}}$, $\sqrt{\frac{M_k}{2}\frac{D}{q}}$ instead of $\sqrt{\frac{M_k}{2}}$, if the bound q, D for the numerator and denominator are known. Also, if a vector of elements with similar denominators is to be reconstructed, the number of entries that need to be reconstructed can also be reduced by preconditioning, see remark in e.g. [102]. This leads to the case when the numerator can be significantly bigger than the denominator, making it suitable to apply Maximal Quotient algorithm.

11.3.1 Probability of Early Termination

To determine the probability of correctness of the termination of CRA/p-adic scheme using Wang's and Monagan's approach, random distribution on u is assumed. This leads to estimated $6/\pi^2$ probability of correctness in the case of Wang's algorithm, see [19]. In [90], Monagan presents asymptotic and experimental claims about his strategy. Experiments and practice show the assumption might not be correct for a particular problem. Measures can be taken to improve the probability, including taking bounds q and D smaller that $\sqrt{M_k/2}$. For another approach in the case of rational CR Algorithm, see Sec. 10.6.2.

11.4 Fast Rational Reconstruction - Implementation

In this section we will present details on the implementation of the algorithm of [131]. Another algorithm for fast rational reconstruction was presented in [88]. Unfortunately, the algorithm, as it is presented in [131, Alg.1] is not correct, which we will show in Ex. 11.4.5. Yet it is easy to correct the algorithm, which we will try to clarify now.

Let us first introduce the notations that are used in the paper of Pan and Wang [131]. At the input of the Fast Extended Euclidean Algorithm FEEA we are given a pair of integers r_0, r_1, $r_0 > r_1 \geq 0$. For $i = 1, \ldots, l$ we define $r_{i+1} = r_i \mod r_{i-1}$ and $q_i = \lfloor \frac{r_{i-1}}{r_i} \rfloor$, until $r_{l+1} = 0$. Moreover, a matrix $Q_i = \begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix}$ is computed such that

$$\begin{pmatrix} r_0 \\ r_1 \end{pmatrix} = Q_i \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix}. \tag{11.2}$$

Additionally, we define $Q_0 = \mathrm{Id}$ and $Q_{l+1} = \begin{bmatrix} \infty & \infty \\ \infty & \infty \end{bmatrix}$. Recalling the notations of [70, Sec. 5.10], we have that $Q_i^{-1} = \begin{bmatrix} s_i & t_i \\ s_{i+1} & t_{i+1} \end{bmatrix}$, where s_i, t_i, $i = 0, \ldots, l$ are the resultants of the extended Euclidean Algorithm i.e.

$$s_i r_0 + t_i r_1 = r_i.$$

Notice, that Eq. (11.1), $a = \mathrm{sgn}(t_i) r_i, b = t_i$, $i = 0, \ldots, l$ is one of the solutions to the rational reconstruction problem defined by Eq. (11.1).

Remark 11.4.1 Other authors e.g. [70, 88, 90], prefer to use the dual notation in their presentation of extended Euclidean algorithm, by computing Q_i^{-1}. The aim of this section being the experimental validation of [131] algorithm and its extension points us to adapt his notions instead.

In [131, Def 2.3] the definition of matrices Q_i is given.

Definition 11.4.2 (Def. 2.3 of [131]) Let $r_0 \geq r_1 > 0$ be integers and let $q_i, i = 0, \ldots, l$ be the quotient sequence in the Euclidean algorithm for r_0, r_1. Define $Q_0 = \mathrm{Id}$ and $Q_{l+1} = \begin{bmatrix} \infty & \infty \\ \infty & \infty \end{bmatrix}$ and $Q_i = Q_{i-1} \begin{bmatrix} q_i & 1 \\ 1 & 0 \end{bmatrix}$. Then Q_i is the matrix sequence for r_0, r_1.

In [131, Thm. 2.6] the authors show how to obtain the next and the previous matrix Q_{i-1}, Q_{i+1} from matrix $Q_i, i = 1, \ldots, l-1$. We cite the theorem in 11.4.3. Notice that part (ii) of the theorem was corrected in the border cases for $i = 0, 1, 2$.

Theorem 11.4.3 (Thm. 2.6 of [131]) *Let r_i, $i = 0, \ldots, l$ be the remainder sequence of the Euclidean Algorithm, let $Q_i = \begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix}$ be the matrix sequence for r_0, r_1. We have*

(i) $b_i = a_{i-1}$, $d_i = c_{i-1}$ *for* $i = 1, \ldots, l$.

(ii) $a_i = a_{i-1} q_i + a_{i-2}$ *and* $c_i = c_{i-1} q_i + c_{i-2}$ *for* $i = 2, \ldots, l$. *Moreover* $a_1 \geq a_0$, $a_i > a_{i-1}$ *for* $i = 2, \ldots, l$ *and* $c_2 \geq c_1 > c_0$, $c_i > c_{i-1}$ *for* $i = 3, \ldots, l$.

(iii) $a_{i-2} = a_i \mod a_{i-1}$, $c_{i-2} = c_i \mod c_{i-1}$ *for* $i = 3, \ldots, l$.

(iv) $a_0 > c_0, a_1 \geq c_1, a_i > c_i$ *for* $i = 2, \ldots, l$.

PROOF In [131] no proof of Thm. 2.6 is given as it is considered basic. Due to the issue in part (ii) of the theorem and for sake of completeness we would like to include it here. Part (i) follows directly from Def. 11.4.2 as are the formula for a_i and b_i in part (ii). Moreover, $a_0 = 1 \leq q_1 = a_1$ and $c_0 = 0 < c_1 = 1 \leq q_2 = c_2$. Hence $a_i > a_{i-1}$ and $c_{i+1} > c_i$ for $i = 2, \ldots, l$. Equations in part (iii) follow as soon as $a_{i-1}, c_{i-1} > 1$ thus for $i = 3, \ldots, l$ as claimed. Also $a_0 = 1 > 0 = c_0$, $a_1 = q_1 \geq 1 = c_1$ and $a_i > c_i$ for $i = 2, \ldots, l$.

Remark 11.4.4 Part (*iii*) of Thm. 11.4.3 allows us to compute Q_{i-1} from Q_i for $i = 3, \ldots, l$. As equalities between a_1 and a_0, c_1 and c_2 and a_1 and c_1 cannot be ruled out, several cases are possible in the computation of Q_1 from Q_2, see Alg. 11.4.1. The case when $i = 1$ can easily be detected by checking whether $d_i = 0$, which points us to take $Q_{i-2} = \mathrm{Id}$ in this case.

Theorem 11.4.3 gives raise to two procedures: $PrevEuclideStep(Q)$ in Alg. 11.4.1 and $NextEuclideStep(Q, r_i, r_{i+1})$ in Alg. 11.4.2.

Algorithm 11.4.1 PrevEuclideanStep

Require: $Q = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, an element of the matrix sequence, $Q \neq \mathrm{Id}$.

Ensure: $Q' = \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix}$ the element previous to Q in the matrix sequence.

$a' = b; , c' = d;$
if $b = 1$ and $d = 1$ **then** $b' = 1; d' = 0;$ return; ;
$q = \lfloor \frac{a}{b} \rfloor;$
$b' = a - qb, d' = c - qd;$
$Q' = \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix};$
Return $Q';$

Algorithm 11.4.2 NextEuclideanStep

Require: integers $r_i > r_{i+1} \geq 0$,

Require: $Q = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ an element of the matrix sequence for r_0, r_1, $Q \neq \begin{bmatrix} \infty & \infty \\ \infty & \infty \end{bmatrix}$,

Ensure: $Q' = \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix}$ the element next to Q in the matrix sequence.

$b' = a, d' = c;$
if $r_{i+1} > 0$ **then**
  $q = \lfloor \frac{r_i}{r_{i+1}} \rfloor$
  $r_{i+2} = r_i - qr_{i+1}$
  $a' = aq + b, c' = cq + d;$
else $a' = b' = c' = d' = \infty$
$Q' = \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix}$
Return (Q', r_{i+1}, r_{i+2})

Let us now recall the original algorithm 4.1 of [131].

Alg. 4.1 of [131] is as follows. On the input, we have integers $0 < r_1 \leq r_0$, $h \geq 0$, $d = \lfloor \log(r_0) \rfloor$. On output, we get the unique index k such that matrix Q_k from the matrix sequence for r_0, r_1 fulfills $Q_k \leq 2^h < Q_{k+1}$.

1. If $h \leq \lfloor d/2 \rfloor - 1$ then recursively apply the algorithm to $r_0^* = \lfloor \frac{r_0}{2^{d-2h-1}} \rfloor, r_1^* = \lfloor \frac{r_1}{2^{d-2h-1}} \rfloor$ in order to obtain matrix Q_i^* from the matrix sequence Q_i^* for r_0^*, r_1^*. Then perform 2 reverse Euclidean steps in order to obtain $Q_{i-2} = Q_{i-2}^*$. Perform a finite number of forward Euclidean steps in order to find Q_k, Q_{k+1}.

2. If $\lfloor d/2 \rfloor \leq h \leq d-1$, apply the algorithm to $r_0, r_1, \lfloor h/2 \rfloor$ in order to obtain matrix Q_i from the matrix sequence Q_i for r_0, r_1. Then apply the algorithm again to $r_i, r_{i+1}, \lfloor h/2 \rfloor$ in order to obtain matrix $\tilde{Q}_j$ from the matrix sequence $\tilde{Q}_j$ for r_i, r_{i+1}. Set $Q_{i+j} = Q_i \tilde{Q}_j$ and compute Q_k, Q_{k+1} in a finite number of forward Euclidean steps.

3. If $k \geq d$, apply the algorithm to $r_0, r_1, d-1$ in order to obtain matrix Q_i from the matrix sequence Q_i for r_0, r_1. Then compute Q_k, Q_{k+1} in a finite number of forward Euclidean steps.

In step 1 the authors apply [131, Thm. 3.3] in order to prove that $Q_{i-1} = Q_{i-2}^*$, and that Q_k can be found in 2 reverse steps and at most 4 forward steps. In this step the actual reduction in size of the input entries is performed. Most of the computation is done on integers r_0^*, r_1^* of reduced size. Similarly, in step 3 at most 4 forward steps are needed to find Q_k.

In step 2 the decomposition on two smaller size tasks is performed. The authors claim that $Q_{i+j+2} > 2^{h-1}$. Unfortunately, this claim is not true, as the following example shows.

Example 11.4.5 Let $r_0, r_1, h, h > 3$ be the input in the 2 step of [131, Alg. 4.1]. Let

$$Q_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix},$$

$$Q_2 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2^{\lfloor \frac{h}{2} \rfloor} + 1 & 1 \\ 1 & 0 \end{bmatrix},$$

$$Q_3 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2^{\lfloor \frac{h}{2} \rfloor} + 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

and for $i = 4, 5, \ldots,$

$$Q_i = Q_{i-1} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

be the elements of the matrix sequence for r_0, r_1. The first recursive call to FEEA outputs Q_1 as for Q_2, $a_2 = 2^{\lfloor \frac{h}{2} \rfloor} + 2 > 2^{\lfloor \frac{h}{2} \rfloor}$ and the second recursive call to FEEA outputs $\tilde{Q}_0 = \mathrm{Id}$ as $\tilde{Q}_1 = \begin{bmatrix} 2^{\lfloor \frac{h}{2} \rfloor} + 1 & 1 \\ 1 & 0 \end{bmatrix}$. Yet

$$Q_3 = \begin{bmatrix} 2^{\lfloor \frac{h}{2} \rfloor} + 3 & 2^{\lfloor \frac{h}{2} \rfloor} + 2 \\ 2^{\lfloor \frac{h}{2} \rfloor} + 2 & 2^{\lfloor \frac{h}{2} \rfloor} + 1 \end{bmatrix}$$

is less than 2^{h-1}.

Still worst, $\Theta(h)$ forward Euclidean steps are required in order to find Q_k. Indeed, let $f_0 = 0, f_1 = 1$ and $f_{i+1} = f_i + f_{i-1}$ for $i = 1, \ldots,$ define the Fibonacci sequence. Then for $s > 1$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^s = \begin{bmatrix} f_{s+1} & f_s \\ f_s & f_{s-1} \end{bmatrix}.$$

Thus,

$$Q_{2+s} = \begin{bmatrix} 2^{\lfloor \frac{h}{2} \rfloor} + 2 & 1 \\ 2^{\lfloor \frac{h}{2} \rfloor} + 1 & 1 \end{bmatrix} \begin{bmatrix} f_{s+1} & f_s \\ f_s & f_{s-1} \end{bmatrix}.$$

We need to solve the inequality

$$(2^{\lfloor \frac{h}{2} \rfloor} + 2) f_{s+1} + f_s > 2^h$$

As $f_i \leq 2^{i-2}$, for $i \leq 2$, s is greater than $\lceil \frac{h}{2} \rceil$. As f_i increases exponentially, s is $\Theta(h)$ as claimed.

Also, in the worst case $r_i = r_0$ so same input is given to the second recursive call to FEEA. Therefore, Thm. 4.2 of [131] does not hold for [131, Alg. 4.1].

We might suspect that the erroneous conclusion of [131] might be due to the false assumption that $Q_{i+j+2} \neq Q_{i+1}\tilde{Q}_{j+1}$. At the same time, this means that the algorithm can easily be corrected. In Alg. 11.4.3 we present the correct version of the algorithm.

Before proving the correctness of Alg. 11.4.3 let us first recall two key result of [131]

Corollary 11.4.6 (Cor. 2.8 of [131]) Let r_i, $i = 0, \ldots, l$ be the remainder sequence of the Euclidean Algorithm, let $Q_i = \begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix}$ be the matrix sequence for r_0, r_1. Then $|Q_i| \geq |Q_{i-1}| + |Q_{i-2}|$ for $i = 2, \ldots, l$.

PROOF The inequality follows from Thm. 11.4.3 part (ii).

Theorem 11.4.7 (Thm. 3.3 of [131]) *Let $\lambda > 0$ and $\lambda \leq r_1 \leq r_0$. Suppose that $r_0^* = \lfloor \frac{r_0}{\lambda} \rfloor, r_1^* = \lfloor \frac{r_1}{\lambda} \rfloor$. Let $K > 0$ be such that $r_0^* \geq 2K^2$. Let Q_i^* be the element of the matrix sequence for r_0^*, r_1^* such that $|Q_i^*| \leq K < |Q_{i+1}^*|$. Let Q_j be the matrix sequence for r_0, r_1. Then $Q_{i-2}^* = Q_{i-2}$ and $|Q_k| \leq K < |Q_k|$ for index k which fulfills $i - 2 \leq k \leq i + 2$.*

PROOF See [131, Thm. 3.3].

Now, let us prove the correctness of our algorithm.

Theorem 11.4.8 (Rational Reconstruction by Alg. 11.4.3) *Let $0 < u < p$ be integers, let $K > 0$. Let $Q = FEEA(u, p, \lfloor \log(K) \rfloor)$ be the output of Alg. 11.4.3. Then*

Algorithm 11.4.3 Fast Extended Euclidean Algorithm FEEA cf. [131]

Require: $r_0, r_1, h \in \mathbb{Z}$, $r_0 > r_1 > 0$, $h \geq 0$,

Ensure: $Q_k = \begin{bmatrix} a_k & b_k \\ c_k & d_k \end{bmatrix}$ an element of the matrix sequence for r_0, r_1, s.t. $a_k \leq 2^h < a_{k+1}$.

if $r_0 = r_1$ **then Return** $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$;
if $r_1 = 1$ **then**
  if $r_0 \leq 2^h$ **then Return** $\begin{bmatrix} r_0 & 1 \\ 1 & 0 \end{bmatrix}$;
  else Return $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$;
$q_1 = \lfloor \frac{r_0}{r_1} \rfloor$;
if $h = 0$ **then**
  if $q_1 > 1$ **then Return** $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$;
  else $q_2 = \lfloor \frac{r_1}{r_0 - r_1} \rfloor$; **Return** $\begin{bmatrix} q_2 + 1 & 1 \\ q_2 & 1 \end{bmatrix}$;
$Q = \mathrm{Id}; T = 1$; $d = \lfloor \log(r_0) \rfloor$;
if $r_0 \geq T$ **then**
  if $d - 2h - 1 > 0$ **then** {***** This is step 1 *****}
    $r_0^* = \lfloor \frac{r_0}{2^{d-2h-1}} \rfloor$,$r_1^* = \lfloor \frac{r_1}{2^{d-2h-1}} \rfloor$;
    if $r_1^* > 0$ **then**
      $Q^* = FEEA(r_0^*, r_1^*, h)$;
      $Q = PrevEuclideanStep(Q^*)$; $Q = PrevEuclideanStep(Q)$;
  else {***** This is step 2 *****}
    $Q = FEEA(r_0, r_1, \lfloor \frac{h}{2} \rfloor)$;
    $\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = Q^{-1} \begin{pmatrix} r_0 \\ r_1 \end{pmatrix}$;
    if $r_i < 0$ **then** $r_i = -r_i, r_{i+1} = -r_{i+1}$;
    $(Q_+, r_{i+1}, r_{i+2}) = NextEuclideanStep(Q, r_i, r_{i+1})$;
    if $|Q_+| \leq 2^h$ **then**
      if $r_{i+2} > 0$ **then**
        $h' = \lfloor \log(|Q_+|) \rfloor$;
        $\tilde{Q} = FEEA(r_{i+1}, r_{i+2}, h - h' - 2)$; $Q = Q_+ \tilde{Q}$;
      else Return Q_+
    else Return Q
$\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = Q^{-1} \begin{pmatrix} r_0 \\ r_1 \end{pmatrix}$;
if $r_i < 0$ **then** $r_i = -r_i, r_{i+1} = -r_{i+1}$;
$(Q_+, r_i, r_{i+1}) = NextEuclideanStep(Q, r_i, r_{i+1})$;
while $|Q_+| \leq 2^h$ **do**
  $Q = Q_+$; $(Q_+, r_i, r_{i+1}) = NextEculideanStep(Q_+, r_i, r_{i+1})$;
end while

1. $Q = FEEA(u, p, \lfloor \log(K) \rfloor)$;
$\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = Q^{-1} \begin{pmatrix} r_0 \\ r_1 \end{pmatrix}$;
if $r_i < 0$ **then** $r_i = -r_i, r_{i+1} = -r_{i+1}$;
repeat
$Q = Q_+$, $(Q_+, r_i, r_{i+1}) = NextEculideanStep(Q_+, r_1, r_0 - r_1 q_1)$;
until $|Q_+| \leq 2^h$

outputs $Q = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, Q_+$ *such that* $|Q| \leq K < |Q_+|$ *after at most 2 steps of the* **while** *loop.*

2. $(x, y) = (ua - pc, a)$ *is the solution to the rational reconstruction problem* $\frac{x}{y} = u \mod p$ *with the condition* $|x| < p/K, y \leq K$.

PROOF 1. First, let us prove that for integers $0 < r_1 \leq r_0$, $h \geq 0$, $FEEA(r_0, r_1, h)$ correctly finds Q, Q_+ such that $|Q| \leq 2^h < |Q_+|$. Indeed, the first part of the algorithm correctly find Q in the degenerated cases when $r_1 = 1, r_0 = r_1$ and $h = 0$. Also, the recursive calls to $FEEA$ procedure are correct as the condition $r_0 > 1$ is verified and dealt accordingly. The recursion ends if $r_1 = 1$ or $h = 0$. In step 1, r_1 is reduced at least twice, and in step 2, h is reduced at least two times. This proves that the algorithm terminates.

Step 1 of Alg. 11.4.3 is the same as in the original algorithm of [131]. As $r_0^* = \lfloor r_0/\lambda \rfloor$, $\lfloor \log(r_0^*) \rfloor = \lfloor \log(r_0) \rfloor - d + 2h + 1 = 2h + 1$, Thm. 11.4.7 can be applied. At the end of step 1, Q is an element of the matrix sequence for r_0, r_1 and at most 4 steps of the **while** loop are needed to output the correct result.

We changed Step 2. of Alg. 11.4.3 compared to the algorithm of [131]. Let us analyze the impact it has on the algorithm. First call to FEEA returns $Q = Q_i$, which is an element of the matrix sequence for r_0, r_1. We have

$$|Q_i| \leq 2^{\lfloor h/2 \rfloor} < |Q_{i+1}|.$$

Eq. (11.2) allows us to compute r_i and r_{i+1}. If $|Q_{i+1}| > 2^h$ or $|Q_{i+2}| = \infty$ the step finishes, else FEEA is called again with input $r_{i+1}, r_{i+2}, h - \lfloor \log(|Q_{i+1}|) \rfloor - 2$, where $\lfloor \log(|Q_{i+1}|) \rfloor = h' \geq \lfloor h/2 \rfloor$. At the output to this call $\tilde{Q} = \tilde{Q}_j$ is the jth element of the matrix sequence for r_{i+1}, r_{i+2} such that $|\tilde{Q}_j| \leq 2^{h-h'-2} < |\tilde{Q}_{j+1}|$. Now, $Q_{i+1+j} = Q_{i+1}\tilde{Q}_j$ and $|Q_{i+1+j}| \leq 2 \cdot 2^{h'+1} 2^{h-h'-2} = 2^h$. Moreover, $Q_{i+1+j+1} = Q_{i+1}\tilde{Q}_{j+1}$ and $|Q_{i+1+j+1}| \geq 2^{h'} 2^{h-h'-2} = 2^{h-2}$. As $|Q_{i+j+6}| \geq 4|Q_{i+j+2}|$, we need at most 5 steps of the **while** loop.

This proves that at the end of the FEEA call, we get output Q_i from the matrix sequence of r_0, r_1, such that $|Q_i| \leq 2^{\lfloor \log(K) \rfloor} < |Q_{i+1}|$. By Cor. 11.4.6, $K < 2^{\lfloor \log(K) \rfloor + 1} < |Q_{i+1}| + |Q_{i+2}| \leq |Q_{i+3}|$. Thus, at most 2 steps of the while loop allows us to find the result.

2. For the proof, see [131, Sec. 5]. In short, as $Q_i^{-1} = \begin{bmatrix} s_i & t_i \\ s_{i+1} & t_{i+1} \end{bmatrix}$ we notice that $a_i = |t_{i+1}|$. Moreover for $r_0 = p, r_1 = u$, $\operatorname{sgn}(t_{i+1})r_{i+1} = ua_i - pc_i$.

To estimate the bit complexity of Alg. 11.4.3 we are looking for the asymptotic upper bound on the number of operations for input parameters (d, h). We will consider functions $f(d, h)$ which are increasing in each variable. Let $M(x)$ denote the cost of multiplying/dividing two integers of size x. If fast integer multiplication is used $M(x) = O(x \log(x) \log(\log(x)))$. Analyzing the recursive calls as in [131, Thm. 4.2] we may observe that f has to fulfill

$$f(d,h) \leq \begin{cases} f(2h+1,h) & ,d-2h-1>0 \\ f(d,\lfloor \frac{h}{2} \rfloor) + f(d-h', h-h'-2) & ,d-2h-1 \leq 0 \end{cases} + O(M(d)),$$

where h' is defined as in Alg. 11.4.3 and $f(d, 0) = O(M(d))$. In fact $f(d, h) \leq f(2h + 1, h) + O(M(d))$. Indeed, in step 1 this is due to the recursive call. In step 2, we have that $d - 2h - 1 \leq 0$, thus $d \leq 2h + 1$ and the inequality follows by our assumption on monotonicity. Let us now compute $f(2h + 1, h)$. As $2h + 1 - 2h - 1 = 0$ this falls to step 2. Moreover, $h' \geq \lfloor \frac{h}{2} \rfloor$. Hence $h - h' - 2 \leq \lfloor \frac{h}{2} \rfloor$ and

$$f(2h+1,h) \leq f(2h+1,\lfloor \frac{h}{2} \rfloor) + f(2h+1-h', h-h'-2) + O(M(h)) \leq 2f(2\lfloor \frac{h}{2} \rfloor + 1, \lfloor \frac{h}{2} \rfloor) + O(M(h))$$

From this recursive inequality we obtain that $f(2h + 1, h) \leq O(M(h) \log(h))$. Hence

$$f(d,h) \leq f(2h+1,h) + O(M(d)) \leq O(M(h)\log(h) + M(d))$$

A well balanced fast Euclidean algorithm should intertwine steps 1 and 2. Indeed, in the recursive call in step 1, Alg. 11.4.3 goes to step 2 or recursion stops. The inverse is not generally true, as the first recursive call in step 2 might be to step 2 again. Indeed, $h < d - 1$ implies that $d - 2\lceil \frac{h}{2} \rceil - 1 > h + 1 - 2\lceil \frac{h}{2} \rceil - 1 \geq 0$. As we need only to consider $h \leq d + 1$ this covers most of the cases. Moreover, nothing can be said about the second recursive call in Step 2. An introduction of Step 3 as in [131, 4.1] might helps balance the algorithm from this point of view but does not affect the overall complexity.

The asymptotic complexity of Alg. 11.4.3 is better than that of classic extended Euclidean algorithm (which consists of a **while** loop of consecutive calls to NextEuclideanStep in Alg. 11.4.3). Yet for smaller output, the classic algorithm may have better running time. In particular, for word-size integer entries, one may assume that the cost of integer multiplication is constant, which implies that the number of word-size operations is linear in the size of input in this range. In an adaptive algorithm, the value of T can increased to include this case. T could be set experimentally (see Sec. 11.6), but setting T to the maximal word-size integer value would be a safe choice.

While comparing the cost of classic rational reconstruction and fast rational reconstruction, classic reconstruction has another advantage, as only the upper half of the matrix sequence (i.e. a_i, b_i) has to be computed. From the point of view of the adaptive algorithm, this would imply that an upper lever switch between the algorithms should additionally be considered.

In the actual implementation of the algorithm several other issues have to be consider to avoid re-computation of some values in the algorithm. In particular, this is connected with the computation of the reverse Euclidean steps in Step 1 as well as with the computation of the remainders r_i, r_{i+1} and matrix Q_+ in some cases. This can be solved by storing and passing information between the calls whenever applicable. This is a technical issue, for details, we refer the reader to our implementation in LinBox library.

11.5 Maximal Quotient in Fast Rational Reconstruction

In [82] the authors show how to adapt fast Euclidean algorithm to the Maximal Quotient Rational Reconstruction in the case of polynomials. The algorithm of [131] can be used to obtain Maximal Quotient Fast Rational Reconstruction in the case of polynomials. Indeed, for $d = \lfloor \log(r_0) \rfloor$, $h = d + 1$, Alg. 11.4.3 computes all the quotients of the Euclidean algorithm, its speedup being due to the fact that it operates on smaller operands.

The difference between integer and polynomial algorithm lies in step 1 of Alg. 11.4.3, where an adjustment in a form of at most two reverse Euclidean steps might be necessary. This makes the task of finding the maximal quotient more difficult than in the polynomial case. The maximal quotient cannot be computed until the last quotients computed in the recursive calls in step 1 are corrected or confirmed.

After s recursive calls to step 1, this leads to $2s$ quotients which have not yet been confirmed and might all be incorrect in an unlucky case. As a solution, we propose to store these quotients in a queue-like structure (first in first out) of limited length, which depends on the depth of recursion.

To perform rational reconstruction, one also need to store the elements of matrix sequence for r_0, r_1 corresponding to the quotients. This gives raise to another difficulty in Step 2, where the matrix sequence for r_{i+1}, r_{i+2} has to be left multiplied by Q_{i+1} in order to obtain the matrix sequence for r_0, r_1. As more than one quotient is store, this may lead to a considerable amount of additional work. Therefore we postulate to delay the multiplication stage and to store a sequence of matrices. At the end of the algorithm these matrices have to be multiplied in order to obtain the final matrix from the matrix sequence of r_0, r_1, and to solve the rational reconstruction as in Thm. 11.4.8, part 2.

We store the quotients and matrices in a structure of TruncatedQueue. The structure consists of size delimiter $L.size \geq 0$, a list $L.list$ of at most $L.size$ elements, an elements e_{max} which is equal to the maximal element that has been to the queue through its history. In our case element consist of an integer (a quotient q_i, and the corresponding list of matrices $QList_i$). It permits the following operations.

- *changeSize*(int s') - changes the size of TruncatedQueue from current $L.size$ to s' and returns $L.size - s'$ last elements of $L.list$, if $L.size > s'$, updates e_{max} if applicable;

- *pushpop*(Element e) - adds e to the top of TruncatedQueue; returns and deletes the last element from the structure, if the number of elements increases the size delimiter $L.size$, updates e_{max} if applicable;

- *pop*() - returns the first element from TruncatedQueue and deletes it from the structure, does nothing on an empty queue;

- *merge*(TruncatedQueue L_1) - merges the current TruncatedQueue with the new one by repeating *pushpop*() operation; updates e_{max} if applicable;

- $leftmultiply$(Q) - adds Q to the list of matrices for every element of the queue and e_{max};

- *multiplyall*() - which multiplies the list of matrices for e_{max};

Alg. 11.5.1 presents the maximal quotient algorithm combined with the fast reconstruction.

Algorithm 11.5.1 Fast Maximal Quotient Rational Reconstruction

Require: $0 < u < p$,
Ensure: a pair (n, d) such that $ud = n \mod p$.

1: $L = TruncatedQueue()$; $L.size = 0$;
2: $L.e_{max} = (0, \{\mathrm{Id}\})$; $L.list = empty$;
3: $MaxQ\_FEEA(L, u, p, \lfloor \log(p) \rfloor)$;
4: $Q = L.multiplyall()$; $a_{max-1} = Q_{11}$;
5: $\begin{pmatrix} r_{max-1} \\ r_{max} \end{pmatrix} = Q^{-1} \begin{pmatrix} r_0 \\ r_1 \end{pmatrix}$;
6: **if** $r_{max-1} < 0$ **then** $r_{max-1} = -r_{max-1}, r_{max} = -r_{max}$;
7: **Return** (a_{max-1}, r_{max}).

Alg. 11.5.2 is the recursive procedure called by Alg. 11.5.1. As in Alg. 11.4.3, Alg. 11.5.2 receives integers r_0, r_1, h and stops when k is found such that for Q_k from the matrix sequence for r_0, r_1, $|Q_k| \leq 2^h < |Q_{k+1}|$. Additionally, it receives a TruncatedQueue L, not necessarily empty, with the size delimiter $L.size$.

At the end of the algorithm $e_{max} = (q_{max}, QMaxList)$ and L are updated as follows. If k is greater that $s = L.size$ then q_{max} is set to $\max(q_{max}, L.list.q, q_1, \ldots, q_{k-s})$, where q_{max} is the input value and $L.list.q$ are quotients of the input list $L.list$. $QMaxList$ contains the corresponding list of matrices.$L.list$ contains s lately computed quotients and matrices, namely $(q_k, \{Q_{k-1}\}), (q_{k-1}, \{Q_{k-2}\}), \ldots, (q_{k-s+1}, \{Q_{k-s}\})$. If k is less than s then k last elements of $L.list$ are quotients and matrices $(q_k, \{Q_{k-1}\}), (q_{k-1}, \{Q_{k-2}\}), \ldots, (q_1, \{Q_0\})$ and e_{max} is the maximum of: the input q_{max} and $s - k$ last elements of $L.list.q$, together with the corresponding list of matrices.

Algorithm 11.5.2 MaxQ_FEEA

Require: TruncatedQueue L of size $L.size$, $r_0, r_1, h \in \mathbb{Z}$, $r_0 > r_1 > 0$, $h \geq 0$,
Ensure: Updates L as specified;
Ensure: $Q_k = \begin{bmatrix} a_k & b_k \\ c_k & d_k \end{bmatrix}$ an element of the matrix sequence for r_0, r_1, s.t. $a_k \leq 2^h < a_{k+1}$.

if $r_0 = r_1$ **then** $L.pushpop((1, \{\mathrm{Id}\}))$; **Return** $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$;
if $r_1 = 1$ **then**
  if $r_0 \leq 2^h$ **then** $L.pushpop((r_0, \{\mathrm{Id}\}))$; **Return** $\begin{bmatrix} r_0 & 1 \\ 1 & 0 \end{bmatrix}$;
  else Return $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$;
$q_1 = \lfloor \frac{r_0}{r_1} \rfloor$;
if $h = 0$ **then**
  if $q_1 > 1$ **then Return** $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$;
  else
    $L.pushpop((q_1, \{\mathrm{Id}\}))$; $q_2 = \lfloor \frac{r_1}{r_0 - r_1} \rfloor$; $L.pushpop((q_2, \{\begin{bmatrix} q_1 & 1 \\ 1 & 0 \end{bmatrix}\}))$;
    Return $\begin{bmatrix} q_2 + 1 & 1 \\ q_2 & 1 \end{bmatrix}$;
$Q = \mathrm{Id}; T = 1$; $r_i = r_0; r_{i+1} = r_1$; $d = \lfloor \log(r_0) \rfloor$;
if $r_0 \geq T$ **then**
  if $d - 2h - 1 > 0$ **then** {***** This is step 1 *****}
    $r_0^* = \lfloor \frac{r_0}{2^{d-2h-1}} \rfloor, r_1^* = \lfloor \frac{r_1}{2^{d-2h-1}} \rfloor$; $L.size = L.size + 2$;
    if $r_1^* > 0$ **then**
      $Q^* = MaxQ\_FEEA(L, r_0^*, r_1^*, h)$;
      $(q, Q) = L.pop()$; $(q, Q) = L.pop()$; $L.changeSize(L.size - 2)$;
  else {***** This is step 2 *****}
    $Q = MaxQ\_FEEA(L, r_0, r_1, \lfloor \frac{h}{2} \rfloor)$;
    $\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = Q^{-1} \begin{pmatrix} r_0 \\ r_1 \end{pmatrix}$; **if** $r_i < 0$ **then** $r_i = -r_i, r_{i+1} = -r_{i+1}$;
    $(Q_+, r_{i+1}, r_{i+2}) = NextEuclideanStep(Q, r_i, r_{i+1})$;
    if $|Q_+| \leq 2^h$ **then**
      $L.pushpop((\lfloor \frac{r_i}{r_{i+1}} \rfloor, \{Q\}))$;
      if $r_{i+2} > 0$ **then**
        $h' = \lfloor \log(|Q_+|) \rfloor$; $\tilde{L}.size = L.size$;
        $\tilde{Q} = MaxQ\_FEEA(\tilde{L}, r_{i+1}, r_{i+2}, h - h' - 2)$;
        $\tilde{L}.leftmultiply(Q_+)$; $L.merge(\tilde{L})$; $Q = Q_+ \tilde{Q}$;
      else Return Q_+;
    else Return Q;
$\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = Q^{-1} \begin{pmatrix} r_0 \\ r_1 \end{pmatrix}$; **if** $r_i < 0$ **then** $r_i = -r_i, r_{i+1} = -r_{i+1}$;
$L.pushpop((\lfloor \frac{r_i}{r_{i+1}} \rfloor, \{Q\}))$; $(Q_+, r_i, r_{i+1}) = NextEuclideanStep(Q, r_i, r_{i+1})$;
while $|Q_+| \leq 2^h$ **do**
  $L.pushpop((\lfloor \frac{r_i}{r_{i+1}} \rfloor, \{Q\}))$; $Q = Q_+$;
  $(Q_+, r_i, r_{i+1}) = NextEculideanStep(Q_+, r_i, r_{i+1})$;
end while

11.5.1 Correctness and Additional Cost of Alg. 11.5.1

Let us now analyze the correctness of the algorithm in Thm. 11.5.1.

Theorem 11.5.1 (Rational Reconstruction by Alg. 11.5.1) *Let integers p, u, $p > u > 0$ be input to Alg. 11.5.1. Alg. 11.5.1 outputs the fraction corresponding to the maximal quotient in the quotient series for r_0, r_1.*

PROOF The Algorithm returns Q_k such that $Q_k \leq 2^{\lfloor \log(p) \rfloor + 1} \leq Q_{k+1}$, as the path of the algorithm is the same as in the case of 11.4.3. This implies that $Q_{k+1} = \begin{bmatrix} \infty & \infty \\ \infty & \infty \end{bmatrix}$ and thus $Q_k = Q_l$ is the last matrix of the matrix sequence for u, p. At the same time, this means that all the quotients $q_1, \ldots q_l$ of the quotient series have been computed and entered to the TruncatedQueue at some point of the algorithm. By the definition of *merge* operation and taking into account that $L.size = 0$ at the end of the algorithm, this means that the maximum q_{max} has been determined at some point, and the corresponding sequence of matrices $QMaxList$, after multiplication (see step 2), corresponds to Q_{max-1}. Therefore, after [90], the pair (a_{max-1}, r_{max}) is the solution to the rational reconstruction problem corresponding to the maximal quotient.

In Lem. 11.5.2, let us analyze the depth of recursion of Alg. 11.5.1 and consequently, maximal size of TruncatedQueue L throughout the algorithm.

Lemma 11.5.2 (Depth of Alg. 11.5.1) *Let p, u be integers, $p > u > 0$.*

1. *Let $\lfloor \log(p) \rfloor + 1 \geq h > 0$ be integer. Let p, u, h be input to Alg. 11.4.3. Let us consider the recursion tree for the algorithm. The depth of the tree is $O(\log(h))$. In particular, step 1 (resp. step 2) is run at most $\log(h) + 2$ times in each branch of the recursion tree.*

2. *Let p, u be input to Alg. 11.5.1. The number of elements in the TruncatedQueue in the recursive call to MaxQ_FEEA procedure does not exceed $\log(\log(p) + 1) + 2$.*

PROOF Step 2 results with two branch, in which the third parameter is equal to $\lfloor h/2 \rfloor$ or $h - h' - 2 \leq \lfloor h/2 \rfloor$, as $h' \geq \lfloor h/2 \rfloor$ resp. In the recursive call to FEEA procedure in step 1, step 2 always follow and thus the third parameter is at least halved. Thus, the third parameter is at least halved in at most 2 recursive calls to FEEA procedure. Recursion stops when the parameter reaches 0. Thus, the overall depth of the recursion is at most $2(\log(h) + 2)$, which is $O(\log(h))$. At most half of the steps in each branch are step 1, and there are at most $\log(h) + 2$ calls to step 2 (branches), which gives the result.

Alg. 11.5.1 follow the same path of computation as Alg. 11.4.3 with $h = \lfloor \log(p) \rfloor + 1$ which by previous reasoning means that step 1 is run at most $\log(\log(p) + 1) + 2$ times. As the size of TruncatedQueue is increased only in this step, this gives the result.

By Lem. 11.5.2, additional structure of TruncatedQueue in Alg. 11.5.1 compared to Alg. 11.4.3 requires additional storage of at most $s = \log(\log(p) + 1) + 2$ quotients and matrices, some of which may not belong to the quotient/matrix sequence of the input parameters.

The following lemma allows us to bound the size of matrix entries.

Lemma 11.5.3 (Cor. 2.6 of [131]) *Let r_0, r_1 be integers $r_0 > r_1 > 0$. Let $q_1, \ldots, q_l$ be the quotient series, $r_0, \ldots, r_l$ be the remainder series and $Q_0, \ldots, Q_l$ be the matrix series for r_0, r_1. Then*

$$q_{i+1}|Q_i| \leq q_{i+1}r_{i+1}|Q_i| \leq r_i|Q_i| \leq r_0$$

for $i = 0, \ldots, l-1$.

PROOF The claim that $r_i|Q_i| \leq r_0$ can be found in [131, Cor. 2.6]. The rest of inequalities follow easily. See also [90].

We have the following theorem.

Theorem 11.5.4 (Additional Cost of Alg. 11.5.1) *Let p, u be integers, $p > u > 0$. Let p, u be input to Alg. 11.5.1.*

1. *Throughout Alg. 11.5.1, TruncatedQueue L requires at most $O(\log(\log(p))\log(p))$ bits of memory.*
2. *The cost of operations on TruncatedQueue is $M(\log(p))$.*

PROOF Let p, u be input to Alg. 11.5.1.

1. From Lem. 11.5.3 it follows directly, that the cost of storing a correct quotient and matrix list $(q, QList)$ from quotient series of p, u is $\log(p)$ as $q \cdot |multiplyall(QList)| \leq p$. At the same time, in the recursive call in step 1, $r_i^* \leq p$ and $q \cdot |multiplyall(QList)| \leq r_i^* \leq p$ for, possibly, uncorrect quotients as well. Since at most $s = O(\log(\log(p)))$ pairs $(q, QList)$ are stored, the overall additional storage is $O(\log(\log(p))\log(p))$.
2. As elements are entered to the TruncatedQueue one by one, the maximal quotient is updated in $O(1)$ at most $l = O(\log(p))$ times. At the end of the call to $MaxQ\_FEEA$ the $L.e_{max}.QMaxList$ is a list of 2×2 matrices that have to be multiplied. By Lem. 11.5.2 there are at most $s = O(\log(\log(p)))$ matrices in the list and all matrices have only non negative entries. The result of the multiplication is en element Q_{max-1} of matrix sequence for p, u and thus its entries are less than p. Hence, the cost of multiplication is $O(M(\log(p)))$, where $M(\log(p))$ is the cost of multiplication of integers of size $\log(p)$.

11.6 Rational Reconstruction - Experimental Results

Algorithms 11.4.3 and 11.5.1 are implemented in the LinBox exact linear algebra library in file "linbox/algorithms/fast-rational-reconstruction.h".

In this section we present the experimental evaluation of Alg. 11.4.3. In Sec. 11.6.1 we present the estimation of threshold T, which can be used to speed up Alg. 11.4.3 and Alg. 11.5.1 by turning to classic Euclidean algorithm for smaller inputs. In Sec. 11.6.2

$\log(p) \setminus \log(T)$	∞	32	64	128	256	384	448	512
2236	0.00061	0.00267	0.00200	0.00100	0.00133	0.00100	0.00083	0.00083
4024	0.00161	0.00433	0.00333	0.00200	0.00167	0.00233	0.00183	0.00200
5367	0.00256	0.00667	0.00500	0.00433	0.00333	0.00267	0.00300	0.00283
6699	0.00339	0.00900	0.00533	0.00433	0.00400	0.00417	0.00367	0.00350
7586	0.00400	0.00900	0.00567	0.00533	0.00433	0.00417	0.00417	0.00417
8932	0.00556	0.01200	0.00767	0.00667	0.00600	**0.00500**	**0.00533**	**0.00517**
9819	0.00650	0.01400	0.01000	0.00800	0.00683	0.00583	0.00617	0.00617
11154	0.00800	0.01500	0.01067	0.00867	**0.00717**	0.00633	0.00683	0.00667
12473	0.00972	0.01700	0.01333	0.01100	0.00867	0.00800	0.00833	0.00800
14262	0.01300	0.01867	0.01367	**0.01067**	0.00983	0.00900	0.000883	0.00867
16028	0.01633	0.02033	**0.01533**	0.01267	0.1083	0.01000	0.01100	0.01000
19571	0.02389	0.03033	0.02333	0.01867	0.01583	0.01483	0.01533	0.01433
23553	0.03433	0.03733	0.02833	0.02300	0.01983	0.01817	0.01883	0.01800
28402	0.04811	**0.04233**	0.02966	0.02766	0.02267	0.02133	0.02100	0.02267

Table 11.1: Evaluation of optimal T in Alg. 11.4.3. Average times in seconds for Alg. 11.4.3 with $T = 2^{\log(T)}$ are given. The values of $\log(p)$ are listed in the table, averaging is done for 10 random u. $\log(T) = \infty$ corresponds the classic algorithm. In bold: at about this value of p and T, fast rational reconstruction beats the classic algorithm ($\log(T) = \infty$).

we present the comparison of Wang's [129, 130] and Monagan's [90] early termination strategies for rational CRA. The comparison is express in terms of the number of iterations and timings.

All experiments in this section were performed Intel(R) Core(TM)2 Duo 2.66GHz CPU with 4Gb memory, running Linux.

11.6.1 Computation of Threshold

In Alg. 11.4.3 and Alg. 11.5.1 threshold T, which implements the switch between fast and classic extended Euclidean algorithm, is set to 1. However, we remark that setting T to MAX_INT value should result in better performance. In fact, optimal threshold T could be even bigger. Here, we are going to evaluate the threshold experimentally. We run Alg. 11.4.3 and with $h = \lfloor \sqrt{p/2} \rfloor$ (see Wang's strategy) and compare the timings with classic rational reconstruction, which computes resultants s_i and t_i (i.e. $T = \infty$), see Sec. 11.2. We ran the algorithms for random inputs $(u, p), u < p$ and various thresholds T. Tab. 11.1 presents some of the results. For full comparison see Fig. 11.1.

From Tab. 11.1 and Fig. 11.1 we may conclude, that for $\log(T) = 384, 448, 512$ comparable and best timings can be obtained. Thus, we decided to take the smallest threshold $T = 2^{384}$ in our experiments. We can see that $T = 2^{384}$ performed better for big inputs. Notice, that in this case fast rational reconstruction beats classic rational reconstruction for $\log(p) \simeq 9000$.

In Fig. 11.2 we present the comparison of timings for classic rational reconstruction (EEA) and fast rational reconstruction (FEEA, see Thm. 11.4.8) for a benchmark example of two consecutive Fibbonacci numbers F_{n-1} and F_n. We reconstructed the fraction($a/b = F_{n-1} \bmod F_n$), where $|a| < \sqrt{F_n/2}$ and $b < \sqrt{F_n/2}$ using the classic algorithm and using Alg. 11.4.3. Alg. 11.4.3 is definitely better than the classic algorithm for $n \gtrsim 6500$ which corresponds to the Fibbonacci number F_n, s.t. $\log(F_n) \gtrsim 4500$, which is by half less than it the previous experiment with random numbers.

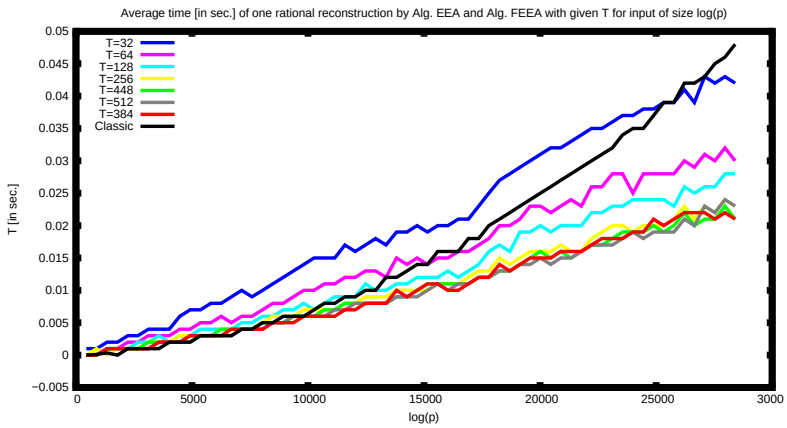

Figure 11.1: Evaluation of optimal T in Alg. 11.4.3. Average times in seconds for Alg. 11.4.3 with $T = 2^{\log(T)}$ are given. The values of $\log(p)$ are listed in the table, averaging is done for 10 random u. $\log(T) = \infty$ corresponds the classic algorithm. In bold: at about this value of p and T, fast rational reconstruction beats the classic algorithm ($\log(T) = \infty$).

11.6.2 Comparison of Wang's and Monagan's Strategies

In [90], Monagan claims that his Maximal Quotient Rational Reconstruction Algorithm can reduces the number of reconstructions in the case of early terminated CRA and p-adic schemes when compared to Wang's [129] termination strategies.

We designed the following experiments to evaluate this claim experimentally. We recall that the number of iterations in certified p-adic lifting for solving equation $Ax = b$, where A is a $n \times n$, is $\log_p(H(A)) + \log_p(H(A, b)) + 1$ where $H(A)$ is given by Eq. (6.6) and $H(A, b) = n^{n/2}\|A\|^{n-1}\|b\|$, see e.g. [27]. The early termination strategies of Wang and Monagan would need about $2\log_p(\max(Num(x), Den(x)))$ or $\log_p(\max(Num(x))) + \log_p(\max(Den(x))) + c$, where c is a small constant, and $Num(x), Den(x)$ denote the vector of numerators and denominators of x respectively, see [129, 90] and Sec. 11.3.

1. Evaluation of Wang's and Monagan's termination for p-adic lifting with matrices D_n, such that $SF(D_n) = \mathrm{diag}(1, 2, \ldots, n)$. Matrices were obtained by performing $2n$ random row and column elementary operations on $\mathrm{diag}(1, 2, \ldots, n)$.

   This is the same example of malicious matrices as in Sec. 8.8.2. From previous experiments, see Tab. 8.1, we knew that the largest invariant factor s_n of A_n is only a small factor of the denominator. Thus, we suspected that the number of iterations in certified p-adic lifting might be largely overestimated and wanted to evaluate the gain of early termination strategies in terms of the number of iterations and timings.

   Tab. 11.2 presents the number of iterations for two variants of Wang's strategies, the strategy of Monagan and certified solving. Fig. 11.3 presents the timings for the same data and Fig. 11.4 shows which fraction of time was consecrated to rational reconstruction. It can be seen that early termination allowed us greatly reduce the number of iterations to 15%, for the smallest matrix and 7% for the biggest matrix,

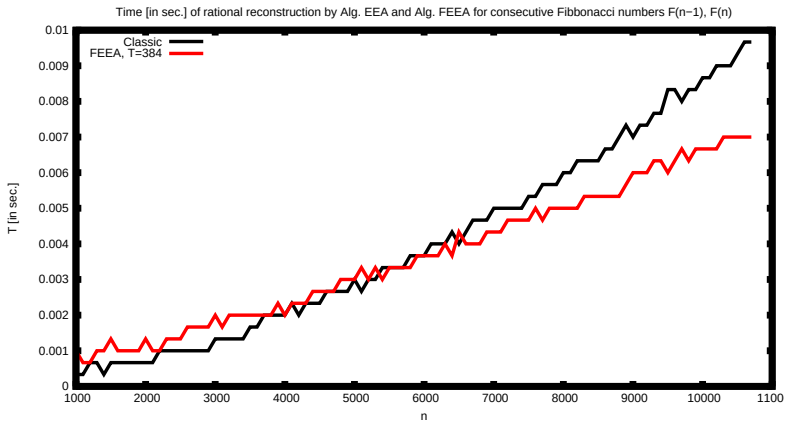

Figure 11.2: Comparison of timings for classic and FEEA rational reconstruction for two consecutive Fibbonacci numbers F_{n-1}, F_n. Times in seconds.

n	$\lceil log_p(N) \rceil$	$\lceil log_p(d) \rceil$	Wang(1)	Wang(2)	Monagan	Certified
100	9	7	13 [17.33%]	17 [22.67%]	11 [14.67%]	75 [100%]
200	17	15	27 [15.88%]	35 [20.59%]	18 [10.59%]	170 [100%]
300	23	22	34 [12.50%]	48 [17.65%]	25 [9.19%]	272 [100%]
400	31	30	49 [12.89%]	63 [16.58%]	34 [8.95%]	380 [100%]
600	47	45	76 [12.71%]	89 [14.88%]	48 [8.03%]	598 [100%]
800	60	59	67 [8.08%]	119 [14.35%]	63 [7.60%]	829 [100%]
1000	74	72	84 [7.79%]	154 [14.29%]	77 [7.14%]	1078 [100%]

Table 11.2: The number of iterations in p-adic lifting for solving equation $D_n x = b$, with random vector b of word-size integers. Prime p s.t. $\log(p) = 20$ was used. The result x was $x = \frac{N}{d}$, where N is a vector of numerators and d is the common denominator. The number of iterations is is given for four strategies: Wang(1) which implements Wang[129] strategy and reconstructs $\frac{n_i}{d_i}$ for every coordinate, Wang(2) which implements Wang[129] and reconstructs $\frac{n_1}{d_1}$, and then preconditions the remaining entries to reconstruct $\frac{d_1 n_2}{d_2}$ etc., Monagan which implements MQRR algorithm of [90] (preconditioning is used as in Wang(2)) and Certified p-adic lifting [27]. Percentage result in brackets.

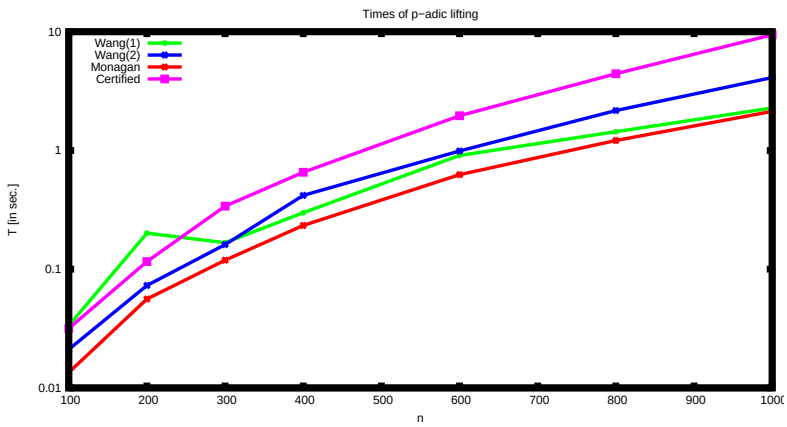

Figure 11.3: Time [in sec.] of early terminated and certified p-adic lifting procedure for solving equation $D_n x = b$, with random vector b of word-size integers. Four strategies are compared Wang(1) which implements Wang[129] strategy and reconstructs $\frac{n_i}{d_i}$ for every coordinate, Wang(2) which implements Wang[129] and reconstructs $\frac{n_1}{d_1}$, and then $\frac{d_1 n_2}{d_2}$ etc., Monagan which implements MQRR algorithm of [90] and Certified p-adic lifting [27]. Reconstruction was performed at every step of p-adic lifting. Scaling is logarithmic.

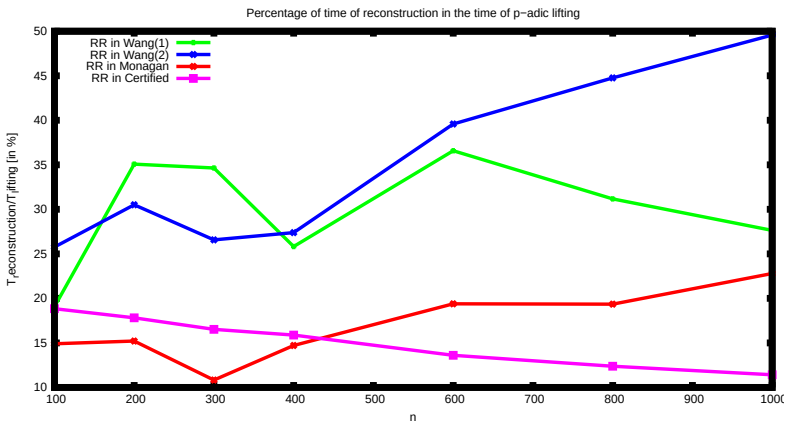

Figure 11.4: Time of rational reconstruction as the percentage of the time of the whole p-adic lifting for D_n matrices. Four strategies are compared Wang(1) which implements Wang[129] strategy and reconstructs $\frac{n_i}{d_i}$ for every coordinate, Wang(2) which implements Wang[129] and reconstructs $\frac{n_1}{d_1}$, and then $\frac{d_1 n_2}{d_2}$ etc., Monagan which implements MQRR algorithm of [90] and Certified p-adic lifting [27]. Reconstruction was performed at every step of p-adic lifting.

in the case Monagan and Wang(1) strategies. Results were checked for correctness. Strategy of Monagan resulted in smallest number of iterations and best timing for all examples tested.

In Wang(1), all vector entries were reconstructed independently, whereas in the remaining strategies the denominator of the first reconstructed entry was used as a preconditioner for the rest of computation. As mentioned in e.g. [102] this is a common approach used in numerous libraries. Interestingly, the number of iterations in Wang's strategy was almost twice smaller if preconditioning was not used (in Wang(1)). This might be explained by the fact, that D_n is obtained from a diagonal matrix in a small number of row and column additions. Thus, denominators d_i in the solution vector $x = [n_i/d_i]$ are not well mixed and might correspond to entries $(1, 2, \ldots, n)$ in the diagonal matrix. This phenomena is not expected to happen in the case of generic dense matrices.

Still, by comparing times in Fig. 11.3 we can see that Wang(2) performs better than Wang(1) for smaller matrices D_n. This can be explained by the fact that rational reconstruction of almost 'integer' fractions $\frac{dn_i}{d_i} = u_t \mod M_t$, where $d = lmc(d_j), j = 1..i$, takes only a few iterations of the Euclidean algorithm, whereas the reconstruction of fraction $\frac{n_i}{d_i} = u_s \mod M_s$ takes more time even if modulus M_s is much smaller than M_t. For D_{200} the time of Wang(1) was even worse than for the Certified method.

In Monagan's strategy, Alg. [90, Alg. MQRR] returned a fraction corresponding to a large quotient $q > \log(M_t) + c$, where $c = 5$ was taken. The probability of correctness for this constant was sufficient, as the whole vector had to be reconstructed correctly. Results were the same if the true maximal quotient was returned, yet the timings in this case were prohibitive, never better and often over 3 times slower than in the case of certified solving.

The number of iterations in Monagan's strategy was always smallest on the examples tested, even though preconditioning might eventually have introduce the same problem as for Wang(1) and Wang(2) strategies. This can be explained by the following reasoning. Suppose that $\frac{n_1}{d_1}$ was reconstructed correctly, which implies that $2n_1d_1 > M_t$ for current modulus M_t in step t of p-adic lifting. Let $d' = \gcd(d_1, d_2)$. Instead of $\frac{n_2}{d_2}$, fraction $\frac{n_2\frac{d_1}{d'}}{\frac{d_2}{d'}}$ is to be reconstructed next. We have

$$2n_2\frac{d_1}{d'}\frac{d_2}{d'} = 2n_2d_2\frac{d_1}{d'2}.$$

Thus, the fraction will be reconstructed as soon as $\frac{d_1}{d'^2} \leq 1$ i.e. as soon as $\gcd(d_1, d_2)$ has at least half the number of bits of d_1. On the contrary for Wang's strategy with preconditioning, the number of iterations can often increase. Indeed it suffices that $n_2 > d_2$ to obtain that

$$2\max((n_2\frac{d_1}{d'})^2, (\frac{d_2}{d'})) \geq 2\max(n_2^2, d_2^2).$$

To conclude, as it was suspected, applying early termination greatly improved the performance of p-adic lifting for both the case of the number of iterations and time. Strategies of Wang(1),Wang(2) and Monagan performed all better than Certified lifting, with strategy of Monagan being slightly ahead. Yet the improvement in the number of iterations does not correspond to the improvement in terms of time. This is due to the fact, that rational reconstruction was performed at each step and, by percentage, was a big time factor in the case of early terminated strategies, see Fig. 11.4. Thus, for certified lifting the relative cost of rational reconstruction fades with time, whereas for early termination strategies of Wang(2) and Monagan it increases fast. In Wang(1), reconstruction generally takes long, but the performance improves in the asymptotic case.

The probability of correctness of early terminated lifting seems to be acceptable and it should be possible to apply early terminated lifting in the adaptive determinant Alg. 8.4.1.

For the p-adic lifting, a random prime p s.t. $\log(p) = 20$ was used in the experiments. Thus from Tab. 11.2, it can be deduced that the sizes of numerators and the common denominator were small and did not exceed 1500 bits for the biggest matrix. Thus, only classic rational reconstruction algorithms were considered.

2. Evaluation of Wang's and Monagan's termination for p-adic lifting for random matrices with integer entries chosen uniformly in the range $[-99, \ldots, 99]$.

   Let A be a random matrix with entries chosen uniformly in the range [-99,...,99]. In Sec. 8.8.1 we confirmed experimentally, that s_n is usually different from $\det(A)$ by only a few bits, see also Thm. 5.3.10 for a theoretical result in this case. In [2, Tab. 1] the authors evaluate experimentally the overestimation $\log(\frac{H(A)}{\det(A)})$, where $H(A)$ is the Hadamard's bound for matrix A and $\det(A)$ is its determinant. The results in [2, Tab.1] and 11.3 imply that the number of iterations in certified p-adic lifting is about 5-10% overestimated.

   In Tab. 11.3 it can be seen that the number of iterations in the case of all early terminated strategies is similar, and slightly better in the case of Monagan's strategy, and slightly worse in the case of Wang(1) strategy (without preconditioning). In Fig. 11.5, it can be seen, that the time for preconditioned reconstruction is significantly better, see also Tab. 11.4 for a percentage result. This implies that in the case of random matrices, preconditioning in the reconstruction of the solution vector is beneficial, as it reduces the number of iterations and improves the time of rational reconstruction. Small difference in the number of iterations of Wang's and Monagan's strategies implies that in the case of random systems of equations, numerators and denominators of result are roughly the same size, thus for $x = \frac{N}{d}$, $\max(\max(N)^2, d^2) \simeq 2\max(N)d$. The size of numerators and denominators suggest that fast rational reconstruction by Alg. 11.4.3 might be considered for bigger matrices.

   Yet the comparison of timings in Fig. 11.5 shows that certified algorithm always performed best and even 2.5 times better than the second best strategy of Monagan. This can be explained by Tab. 11.4 which shows, that rational reconstruction accounted for up to 68.08%,78.61% and 90.16% of the time for Monagan's, Wang(2) and Wang(1)

n	$\log(N)$	$\log(D)$	Wang(1)	Wang(2)	Monagan	Certified
100	871	849	93 [0.98%]	90 [0.95%]	89 [0.94%]	95
200	1808	1785	183 [0.92%]	187 [0.94%]	185 [0.93%]	200
300	3268	3246	341 [0.95%]	339 [0.94%]	339 [0.94%]	359
400	3669	3647	386 [0.95%]	380 [0.94%]	377 [0.93%]	405
600	5678	5655	591 [0.93%]	591 [0.93%]	586 [0.92%]	634
800	7723	7699	801 [0.93%]	801 [0.93%]	781 [0.91%]	859
1000	9809	9786	1012 [0.94%]	1006 [0.93%]	1000 [0.93%]	1077

Table 11.3: The number of iterations in p-adic lifting for solving equation $Ax = b$, where A is a $n \times n$ random matrix with integer entries chosen uniformly in the integer range $[-99, \ldots, 99]$ and b is a random vector of word-size integers. Prime p s.t. $\log(p) = 20$ was used. The result x was $x = \frac{N}{d}$, where N is a vector of numerators and d is the common denominator. The number of iterations is is given for four strategies: Wang(1) which implements Wang[129] strategy and reconstructs $\frac{n_i}{d_i}$ for every coordinate, Wang(2) which implements Wang[129] and reconstructs $\frac{n_1}{d_1}$, and then preconditions the remaining entries to reconstruct $\frac{d_1 n_2}{d_2}$ etc., Monagan which implements MQRR algorithm of [90] (preconditioning is used as in Wang(2)) and Certified p-adic lifting [27]. Percentage result in brackets.

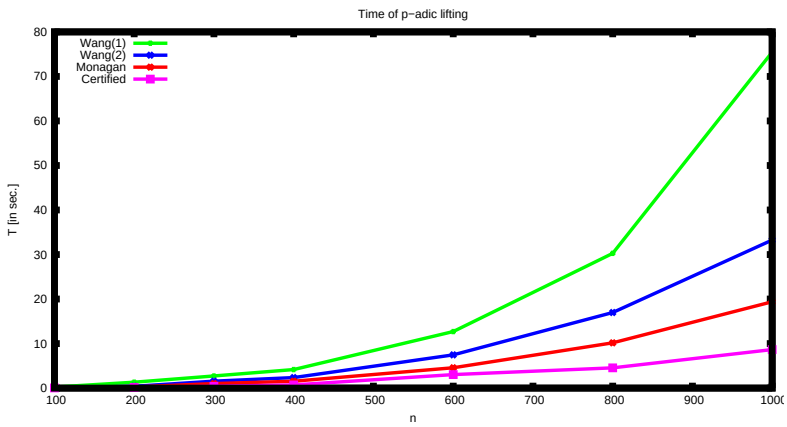

Figure 11.5: Time [in sec.] of early terminated and certified p-adic lifting procedure for solving equation $Ax = b$, where A is a $n \times n$ random matrix with integer entries chosen uniformly in the integer range [-99,. . . ,99] and b is a random vector of word-size integers. Four strategies are compared Wang(1) which implements Wang[129] strategy and reconstructs $\frac{n_i}{d_i}$ for every coordinate, Wang(2) which implements Wang[129] and reconstructs $\frac{n_1}{d_1}$, and then $\frac{d_1 n_2}{d_2}$ etc., Monagan which implements MQRR algorithm of [90] and Certified p-adic lifting [27]. Reconstruction was performed at every step of p-adic lifting. Scaling is logarithmic.

n	Wang(1) [%]	Wang(2)	Monagan	Certified
100	0.17s [74.67]	0.04 [66.31]	0.03 [55.13]	0.01s [22.99]
200	1.07s [81.09]	0.29 [72.94]	0.15 [58.27]	0.03s [22.49]
300	2.40s [87.69]	1.22 [78.61]	0.69 [68.08]	0.09s [22.54]
400	3.54s [85.25]	1.73 [73.84]	0.94 [60.57]	0.14s [20.24]
600	10.91s [85.90]	5.63 [75.44]	2.76 [60.65]	0.70s [23.09]
800	26.23s [86.72]	13.01 [76.70]	6.3 [61.99]	0.74s [16.33]
1000	68.03s [90.16]	26.03 [78.28]	12.23 [63.08]	1.32s [15.22]

Table 11.4: Time of rational reconstruction [in sec. and as the percentage of the time of the whole p-adic lifting] for random matrices. Four strategies are compared Wang(1) which implements Wang[129] strategy and reconstructs $\frac{n_i}{d_i}$ for every coordinate, Wang(2) which implements Wang[129] and reconstructs $\frac{n_1}{d_1}$, and then $\frac{d_1 n_2}{d_2}$ etc., Monagan which implements MQRR algorithm of [90] and Certified p-adic lifting [27]. Reconstruction was performed at every step of p-adic lifting.

strategies compared to 23.0% in the case of certified solving. In fact rational reconstruction in early terminated strategies alone took longer than the whole certified solving. This shows that scheduling of rational reconstruction is necessary if we want to obtain comparable performance in the generic case.

Four main strategies might be envisaged:

- incremental scheduling with threshold T, which schedules the reconstruction to iterations $i = Tk, k = 1, 2, \ldots$;
- quadratic scheduling, which schedules the reconstruction to iterations $i = k^2, k = 1, 2, \ldots$, see [15];
- geometric scheduling, which schedules the reconstruction to iterations $i = 2^k, k = 1, 2, \ldots$;

Notice, that geometric scheduling can effectively be combined with fast polynomial interpolation of $xmodp^{2^k}$ by a divide and conquer method, as in the certified p-adic lifting. Thus, geometric scheduling can be considered the default method.

Still, a quick glance at Tab. 11.3 and a comparison of the numbers of iterations for early terminated and certified strategies reveals that it is debatable whether scheduled rational reconstruction can decrease the number of steps of p-adic lifting in the generic case.

Our experiments have confirmed that the MQRR strategy of Monagan [90] performs better than Wang's early termination [129]. Its is also well suitable for the vector preconditioning strategy used in LinBox and other numerous libraries. Thus, it should also bring reasonable benefits in the case when several linear system are solved for the same matrix A and the denominator of the first solution $\tilde{s}_n$ can be used as preconditioning for the next lifting. This is the case of Alg. 6.3.1 and 7.3.1. Thm. 6.3.2 implies that there are reasonable chances that $\tilde{s}_n A^{-1} b$ is an integer vector.

n	50	100	200	300	500
Monagan's	16	28	49	71	118
Wang's	26	49	92	136	224
α	1.63	1.75	1.88	1.92	1.90

Table 11.5: Comparison of the number of iterations N of rational CRA loop, which used Wang's [129] or Monagan's [90] strategies. The task consisted of solving a linear system of rational equations $H_n x = b$, where b was a random rational vector and H_n was a Hilbert matrix. Rational reconstruction is performed at each step. Wang's strategy performs α times more iterations than Monagan's.

Let us continue with another example.

3. Solving rational equations $H_n x = b$, where H_n is the Hilbert matrix by Rational CRA .

   In Tab. 11.5 we compare the number of iterations of the rational CRA loop in the case of solving a rational system of linear equations $H_n x = b$, where b is a random rational vector and H_n is a Hilbert matrix, see Eq. (13.3) for definition. In is a well known fact see e.g. [17], that the determinant of the Hilbert matrix is equal to $\frac{1}{a_n}$, where $a_n = \prod_k = 1^{n-1}(2k+1) \cdot \big(2k//k\big)^2$. Thus, $\frac{1}{\det(H_n)}$ is integer and the denominator of the result vector x is a factor of $Den(b)$. Thus, we expect the solution $x = \frac{N}{d}$ to consist of a relatively large numerator N and a small denominator d, which means that Monagan's termination strategy is a good candidate.

   Results in Tab. 11.5 were computed assuming that rational reconstruction was performed at each step, which allowed to evaluate the optimal number of iterations independently of scheduling. Experiments prove that the number of iterations needed in the case of Wang's strategy was over 1.5 and almost 2 times bigger than in the case of Monagan's strategy. The ratio is visibly increasing even though the number of iterations is asymptotically $O(\log(a_n))$ in both cases.

12

Implementation of CRA and Rational Reconstruction in LinBox

In this chapter we will present the implementation of the Chinese Remaindering and Rational Reconstruction loop in LinBox. This is the continuation of the work of J.-G. Dumas, on generic Chinese Remaindering design in the scalar and vector case, for integer and rational computation. We have contributed to the implementation by adding the idea of preconditioned CRA with variable preconditioner and by modifying the setting for rational reconstruction to include both Wang's [129] and Monagan's [90] termination. Check Ch. 10 for a survey on (preconditioned) CRA and Ch. 11 for the discussion on rational reconstruction and early termination schemes. See Sec. 11.6 for experimental evaluation.

In what follows, we will present the content and hierarchy of structures in LinBox that realize integer CRA in Sec. 12.1.2, rational CRA in Sec. 12.1.1and Rational Reconstruction 12.1.3.

Trivial and technical parts of the code (e.g. constructors,destructors) were dropped; some template parameter were added to simplify the presentation. The final versions of files in LinBox might differ in details.

12.1 Integer CRA

The main loop of integer CRA is realized by the following structure. See "linbox/-algorithms/cra-domain.h" for the source code.

```
template<class CRABase>
struct ChineseRemainder {
protected:
  CRABase Builder_;
public:
  /*
```

```
   * Runs scalar CRA scheme using functions provided in Builder_ and Iter.
   */
  template<class Int, class Function, class PrimeIter>
  Int& operator() (Int& res, Function& Iter, PrimeIter& p) {
    while ( ! Builder_.terminated() ) {
      ++p;                                  //gets new random prime
      check (! Builder_.noncoprime(*p));    //checks if *p is coprime
      CRABase::Domain D(*p);                //Creates D=Z_p
      Builder_.progress(D, Iter(r, D)); //Computes r = iteration(r,p)
    }
    return res = Builder_.getResult(res);
  }

  /*
   * If (k >=0), resumes CRA for at most k iterations,
   * If (k < 0), resumes CRA until termination occurs,
   * Sets res to the current result at the end of execution,
   * Returns true if termination occurs.
   * To implement: replaced <while> by a <for> loop
   */
  template<class Int, class Function, class PrimeIter>
  bool operator() (const int k, Int& res, Function& Iter, PrimeIter& p);

  /*
   * Analogous functions in the vector case.
   */
  template<class Int, template<class> class Vect, class Function, class PrimeIter>
  Vect<Int> & operator() (Vect<Int>& res, Function& Iter, PrimeIter& p);

  template<class Int, template <class> class Vect, class Function, class PrimeIter>
  bool operator() (const int k, Vect<Int>& res, Function& Iter, PrimeIter& p);

  /*
   *  Changes the preconditioner for Builder_.
   */
  template<class Prec>
  bool& changePrecondtioner(const Prec& P);
};
```

Assume that, according to specification in Sec. 10.1:

- structure `Iter` implements operator(), computing *image* and *iteration* mod `*`p;
- `++`p realizes *get_prime* i.e. returns random primes from P;
- Builder_.progress() implements *reconstruct* and deals with iteration-independent ;

Alg. 10.1.1 can be realized by calling

```
Int res; Function Iter(); PrimeIterator p;
cra(res,Iter,p);
```

Alg. 10.1.1, for which the preconditioner is set and can change during execution can be realized by

```
Int res; Function Iter(); PrimeIterator p;
repeat  {
  Prec P = newPrecondtioner(P);   //computes the new preconditioner
  int k = numberOfIters(k);       //set the number of iterations
  cra.changePrecondtioner(P);
} until ( cra(k, res, Iter, p) )
```

12.1.1 Rational CRA

In the rational case, a similar structure of RationalRemainder is defined, which additionally includes the rational reconstruction. See "linbox/algorithms/rational-cra.h" for the source code.

```
template<class CRABase, class RatRecon>
struct RationalRemainder {
protected:
  CRABase Builder_;
  RatRecon RRec_;
  size_t IterCounter;
public:
  /*
   * Runs scalar CRA scheme using functions provided in Builder_ and Iter.
   * and rational reconstruction scheme implemented by RRec_
   */
  template<class Int, class Function, class PrimeIter>
  Quotient operator() (Int& num, Int& den, Function& Iter, PrimeIter& p) {
    CRABase::Prec P_in = Builder_.getPreconditioner(P_in);
    while ( ! Builder_.terminated() ) {
      ++p;                                  //gets new random prime
      check (! Builder_.noncoprime(*p));    //checks if *p is coprime
      ++IterCounter;
      CRABase::Domain D(*p);                //Creates D=Z_p
      Builder_.progress(D, Iter(r, D));     //Computes r = iteration(r,p)
      if (RRec_.scheduled(IterCounter)) {
          Int m = Builder_.getModulus(m);
          Int u = Builder_.getResidue(u);
          RRec_.reconstructRational(num,den,u,m); //rational reconstruction
          Builder_.changePreconditioner(P_in*Quotient(den,num));
      }
    }
```

```
    CRABase::Prec P = Builder_.getPreconditioner(P);
    Int& res = Builder_.getResult(res);
    num = Num(res*P^{-1}*P_in); den = Den(res*P^{-1}*P_in);
    return Quotient(num, den);
  }

  template<class Int, template<class> class Vect, class Function, class PrimeIter>
  Vect<Int> & operator() (Vect<Int>& num, Int& den, Function& Iter, PrimeIter& p)
  {
    CRABase::Prec P_in = Builder_.getPreconditioner(P_in);
    while ( ! Builder_.terminated() ) {
      ++p;                            //gets new random prime cf. get_prime
      check (! Builder_.noncoprime(*p));             //checks if *p is coprime
      ++IterCounter;
      CRABase::Domain D(*p);                         //Creates D=Z_p
      Builder_.progress(D, Iter(r, D));              //Computes r = iteration(r,p)
      if (RRec_.scheduled(IterCounter)) {
          Int m = Builder_.getModulus(m);
          if (! Builder_.terminated() )
            Int u = Builder_.getResidue(u);              //gets scalar residue
            Int n,d; RRec_.reconstructRational(n,d,u,m); //rational reconstruction
          else {
            Vect<Int> u = Builder_.getResidue(u);        //gets vector residue
            Vect<Int> d; Vect<Int> n;
            RRec_.reconstructRational(n,d,u,m);          //rational reconstruction
          }
          Builder_.changePreconditioner(P_in*Quotient(d,n));
      }
    }
    CRABase::Prec P = Builder_.getPreconditioner(P);
    Int& res = Builder_.getResult(res);
    num = Num(res*P^{-1}*P_in); den = Den(res*P^{-1}*P_in);
    return Quotient(num, den);
  }

   /*
   * Analogous functions as in the integer case
   */
  template<class Int, class Function, class PrimeIter>
  bool operator() (const int k, Int& num, Int& den, Function& Iter, PrimeIter& p);

  template<class Int, template <class> class Vect, class Function, class PrimeIter>
  bool operator() (const int k, Vect<Int>& num, Int& den, Function& Iter, PrimeIter& p);

  //template<class Prec>
  //bool& changePrecondtioner(const Prec& P)
};
```

A standard call to Rational CRA would be

```
Int den, Int num; Function Iter(); PrimeIterator p;
cra(den,num,Iter,p);
```

12.1.2 Implementations of CRABase

In "linbox/algorithm/cra-early-single.h" and "linbox/algorithms/cra-full-multip.h" two base structures for CRABase are implemented, namely EarlySingleCRA and FullMultipCRA.

```
template <class Domain>
struct EarlySingleCRA {
protected:
  Integer         primeProd_=1;      // product of all but last primes
  Integer         nextM_=1;          // lately added prime
  Integer         residue_=0;        // remainder to be reconstructed
  unsigned int    occurrence_ =0;    // number of occurrences of last residue_
  const unsigned int    EARLY_TERM_THRESHOLD;
public:
  void clear() {primeProd_=1;nextM_=1;residue_=0;occurrence_ =0;}
  Integer& getResult(Integer& res) {return res=residue_;}
  Integer& getModulus(Integer& M) {return M=primeProd_*nextM_;}
  Integer& getResidue(Integer& r) {return residue_;}
  bool noncoprime(const Integer& p) {
    return ( (gcd(p, nextM_) != 1) || (gcd(p, primeProd_) != 1) );
  }

  /*
   * Implements early termination strategy,
   */
  bool terminated() {return occurrency_ > EARLY_TERM_THRESHOLD;}
  /*
   * Implements integer reconstruction,
   */
  Integer& progress(Integer& D, Integer& e) {
    Integer u0 = residue_ % D; Integer u1 = e % D;
    Integer m0 = primeProd_*next_M ; Integer m1 = D;
    if ((m1 > m0) || (m1%2==0)) {       //swap
        Integer& tmp = u0; u0 = u1; u1 = tmp;
        tmp = m0; m0 = m1; m1 = tmp;
    }
    primeProd_ =m0; nextM_ = m1;

    if (u0 == u1) ++occurrence_;
    else {
      occurrence_ = 1;
      inv(m0, m0, m1);                    // m0 <-- m0^{-1} mod m1
      u0 = [(u1 - u0)*m0] % m1;
      u0 *=primeProd;
      residue_ += u0;  //residue_ = u0 + [(u1-u0)*m0^{-1}]%m1 * m0
```

```
    }
  }
  /*
   * Same as Integer& progress(Integer& D, Integer& e), when D=Z_p, p is word-size
   * Domain D implements Z_p arithmetics
   * This is a specialization for incremental reconstruction
   */
  Integer& progress(Domain& D, Domain::Element e);

  Integer& getPreconditioner(Integer& P) {return P=1;}
  bool changePreconditioner(const Integer& P) {}
};
```

See Prop. 10.4.1 for the formula of integer reconstruction in `progress()`. See Lem. 10.6.3 for the choice of `EARLY_TERMINATED_THRESHOLD`.

Structure FullMultipCRA implements the delayed reconstruction in the vector case. For the time being, no specialized structure for FullSingleCRA is given, but it may be realized as FullMultiplCRA given a vector of size 1. Triples (residue, modulus, log(modulus)) are stored in table RadixList, see Sec. 10.4.2 and [23] for the description and "linbox/-algorithms/cra-full-multip.h", "linbox/algorithms/lazy_product.h" for the source code.

```
template <class Domain, class RadixList, template<class> class Vect>
struct FullMultipCRA {
protected:
  /*
   * RadixList stores partial residues and prime products
   */
  RadixList RList_;
  RList::modula_iterator modIter_;
  RList::residua_iterator resIter_;
  double LOGARITHMIC_UPPER_BOUND;
public:
  /*
   * getResult is costly
   * should be called after termination() returnes true;
   * reduces the size of RList_ to one;
   */
  Vect& getResult(Vect& res) {RList_.getFrontResidue(res);}
  /*
   * getModulus and getResidue are costly
   * require a call to getResult()
   * getResidue(Integer) is not implemented
   */
  Integer& getModulus(Integer& M) {
    Vect res;
    getResult(res);
```

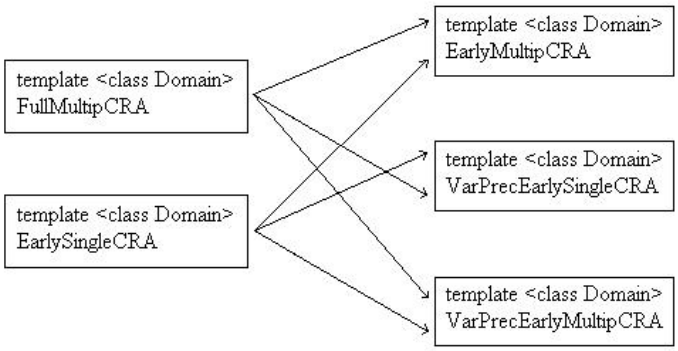

Figure 12.1: Hierarchy for CRABase

```
    RList_.getFrontModulus(M);
  }
  Integer& getResidue(Vect& r) {
    Vect res;
    return getResult(res);
  }

  bool noncoprime() {return RList_.noncoprime();}
  bool terminated() {return RList_.productSize() > LOGARITHMIC_UPPER_BOUND;}
  Integer& progress(Domain D, Vect<Domain::Element> e) {
    return RList_.add(D.characteristic, e, log(D.characteristic));
  }

  Prec& getPreconditioner(Prec& P) {return P=1;}
  bool changePreconditioner(const Integer& P) {}
};
```

Other structures may be constructed as children of the base structures. Fig. 12.1 shows the hierarchy for the presented structures of CRABase.

EarlyMultipCRA implements early terminated CRA in the vector case, as in [4], with a simplified termination technique for easy use, see "linbox/algorithms/cra-early-multip.h".

```
template <class Domain, class RadixList, template<class> class Vect >
struct EarlyMultipCRA: public EarlySingleCRA<Domain>,
                       FullMultipCRA<Domain, RadixList,Vect> {
protected:
  Vect< unsigned long > randv_ = rand();
public:
  Vect<Integer>& getResult(Vect<Integer>& res) {
    return res = FullMultipCRA<Domain>::getResult(res);
  }
```

```
  Integer& getModulus(Integer& M) {
    return EarlySingleCRA<Domain>::getModulus(M);
  }
  Integer& getResidue(Integer& r) {EarlySingleCRA<Domain>::getResidue(r);}
  Integer& getResidue(Vect& r)    {FullMultipCRA<Domain>::getResidue(r);}
  bool noncoprime(const Integer& p) {
    return EarlySingleCRA<Domain>::noncoprime(p);
  }
  /*
   * Checks for termination of randv_*res only.
   */
  bool terminated() {return EarlySingleCRA<Domain>::terminated();}

  Integer& progress(Integer& D,Vect& e) {
    Integer& z = dot(randv_,e) %D;                          //dot product
    EarlySingleCRA<Domain>::progress(D,z);
    FullMultipCRA<Domain>::progress(D,e);
  }
  /*
   * This is a specialization for incremental reconstruction
   */
  Integer& progress(Domain& D,Vect& e);
  bool changeVector() {
    EarlySingleCRA<Domain>::clear();
    randv_ = rand();
    RadixList::residua_iterator resIter_ = FullMultipCRA<Domain>::resIter.begin();
    RadixList::modula_iterator modIter_ = FullMultipCRA<Domain>::modIter.begin();
    for (;resIter != FullMultipCRA<Domain>::resIter.end(); ++resIter; ++modIter; ) {
      Integer& z = *resIter_*randv_ % *modIter;
      EarlySingleCRA<Domain>::progress(*modIter, z);
    }
    return (EarlySingleCRA<Domain>::terminated()) ? true : false;
  }
  Prec& getPreconditioner(Prec& P) {return P=1;}
  bool changePreconditioner(const Integer& P) {}
};
```

Notice that the probability of correctness can not be estimated if `ChineseRemiander::operator()` is called with EarlyMultipCRA. The reoccurrence of vector entries is not checked, see Sec. 10.6.3 for discussion. A proper early termination should separate early termination of `randv_*res` and check the early termination for each entry of the vector separately. Chances of correct termination are also better if several random vectors are used, thus the addition of function `changeVector`.

Let us now present structure VarPrecEarlySingleCRA which can be used to performed CRA with variable preconditioner and should be used with RationalRemainder. The structure is implemented in "linbox/algorithms/varprec-cra-early-single.h".

```
/*
 * Default RadixList and Vect templates are used for FullMultipCRA<Domain>
 * Used to store vectors of size 1 (cf. FullSingleCRA )
 */
template<class Domain, class Prec>
struct VarPrecEarlySingleCRA : public EarlySingleCRA<Domain>,
                               FullMultipCRA<Domain,RadixList,Vect> {
protected:
  Prec precond_;
public:
  Integer& getResult(Integer& res) {
    if (FullMultipCRA<Domain>::terminated()) {
      FullMultipCRA::Vect v;
      return res = FullMultipCRA<Domain>::getResult(v)->front();
    } else {
      return res = EarlySingleCRA<Domain>::getResult(res) * precond_^{-1};
    }
  Integer& getModulus(Integer& M) {return EarlySingleCRA<Domain>::getModulus(M);}
  Integer& getResidue(Integer& r) {return EarlySingleCRA<Domain>::getResidue(r);}
  bool noncoprime(const Integer& p) {return EarlySingleCRA<Domain>::noncoprime();}
  bool terminated() {
    return (EarlySingleCRA<Domain>::terminated()
                || FullMultipCRA<Domain>::terminated();
  }
  Integer& progress(Integer& D,Integer& e) {
    FullMultpiCRA<Domain>::progress(D,Vect(e));
    Integer z = e*precond_ % D;
    return EarlySingleCRA<Domain>::progress(D,z);
  }
  /*
   * This is a specialization for incremental reconstruction
   */
  Integer& progress(Domain& D,Domain::Element& e);
  Prec& getPreconditioner(Prec& P) {return P=precond_;}
  bool changePreconditioner(const Prec& P) {
    EarlySingleCRA<Domain>::clear();
    RadixList::residua_iterator resIter_ = FullMultipCRA<Domain>::resIter.begin();
    RadixList::modula_iterator modIter_ = FullMultipCRA<Domain>::modIter.begin();
    for (;resIter != FullMultipCRA<Domain>::resIter.end(); ++resIter; ++modIter; ) {
      Integer z = *resIter_->front()*precond_ % *modIter->front();
      EarlySingleCRA<Domain>::progress(*modIter, z);
    }
    return (EarlySingleCRA<Domain>::terminated()) ? true : false;
  }
};
```

Analogously, a structure VarPrecEarlyMultipCRA can be defined, which combines the ideas of EarlyMultipCRA and VarPrecEarlySingleCRA. It could be used by RationalRemainder in the vector case.

```
/*
 * Default RadixList template is used for FullMultipCRA<Domain>
 */
template<class Domain, class Prec, template<class> class Vect>
struct VarPrecEarlyMultipCRA : public EarlySingleCRA<Domain>,
                               FullMultipCRA<Domain, RadixList, Vect>
protected:
  Vect< unsigned long > randv_ = rand();
  Prec precond_;
public:
  Vect<Integer>& getResult(Vect<Integer>& res) {
    if (FullMultipCRA<Domain>::terminated()) {
      return res = FullMultipCRA<Domain>::getResult(res);
    } else {
      Vect<Integer> v; FullMultipCRA<Domain>::getResult(v);
      Integer M = EarlySingleCRA<Domain>::getModulus(M);
      res = v*precond_ % M;   //preconditioned result in vector case;
      return res = res*precond_^{-1};                 // real result;
    }
  Integer& getModulus(Integer& M) {return EarlySingleCRA<Domain>::getModulus(M);}
  Integer& getResidue(Integer& r) {return EarlySingleCRA<Domain>::getResidue(r);}
  Vect<Integer>& getResidue(Vect<Integer>& r) {
    r = FullMultipCRA<Domain>::getResidue(r) * precond_ ;
    Integer M = getModulus(M);
    return r = r % M;
  }
  bool noncoprime(const Integer& p) {return EarlySingleCRA<Domain>::noncoprime();}
  bool terminated() {
    return (EarlySingleCRA<Domain>::terminated()
                || FullMultipCRA<Domain>::terminated();
  }
  Integer& progress(Integer& D, Vect<Integer>& e) {
    FullMultpiCRA<Domain>::progress(D,e);
    Integer z = dot(rand_v,(e*precond_)) % D; //dot product
    return EarlySingleCRA<Domain>::progress(D,z);
  }
  /*
   * This is a specialization for incremental reconstruction
   */
  Integer& progress(Domain& D,Domain::Element& e);
  Integer& getPreconditioner(Prec P) {return P=precond_;}
  bool changePreconditioner(const Prec P) {
    EarlySingleCRA<Domain>::clear();
    RadixList::residua_iterator resIter_ = FullMultipCRA<Domain>::resIter.begin();
    RadixList::modula_iterator modIter_ = FullMultipCRA<Domain>::modIter.begin();
    for (;resIter != FullMultipCRA<Domain>::resIter.end(); ++resIter_; ++modIter_; ){
      Vect<Integer> z = *resIter_*precond_ % *modIter;
      EarlySingleCRA<Domain>::progress(*modIter, dot(randv,z));
    }
```

```
    return (EarlySingleCRA<Domain>::terminated()) ? true : false;
  }
  bool changeVector() {
    EarlySingleCRA<Domain>::clear();
    randv_ = rand();
    RadixList::residua_iterator resIter_ = FullMultipCRA<Domain>::resIter.begin();
    RadixList::modula_iterator modIter_ = FullMultipCRA<Domain>::modIter.begin();
    for (;resIter != FullMultipCRA<Domain>::resIter.end(); ++resIter; ++modIter; ) {
      Vect<Integer> z = *resIter_*precond_ % *modIter;
      EarlySingleCRA<Domain>::progress(*modIter, dot(randv,z));
    }
    return (EarlySingleCRA<Domain>::terminated()) ? true : false;
  }
};
```

12.1.3 Implementation of RatRecon

The scheme of repeating rational reconstruction is realized by the following structure. The structure implements scheduling methods (here, `INCREMENTAL`,`QUADRATIC` and `GEOMETRIC`) and calls RRBase to perform the actual rational reconstruction.

```
template <class RRBase, class Method>
struct RReconstruction {
protected:
  RRBase RR_;
  int RecCounter_=0;
  const Method M_;
  const size_t THRESHOLD_;
public:

  bool scheduled(int i) {
    if (M_==INCREMENTAL) {
        if (RecCounter_%THRESHOLD_==0 ) return true;
        else return false;
    }
    else if (M_ == QUADRATIC) {
        if (RecCounter_*RecCounter_ == i) return true;
        else return false;
    }
    else if (M_ == GEOMETRIC) {
        if ((1 << RecCounter_) == i) return true;
        else return false;
    }
    else return true;
  }
```

```
  /*
   * Returns a/b=x mod m by the default method of RRBase
   */
  bool reconstructRational(Element& a, Element& b,
                           const Element& x, const Element& m) const{
    ++RecCounter_;
    Element x_in(x);
    if (x<0) {
      if ((-x)>m) x_in %= m;
      if (x<0) x_in += m;
    } else if (x>m) x_in %= m;
    //Now, 0 < = x_in < m
    return _RR.reconstructRational(a,b,x_in,m);
  }

  /*
   * Returns a/b=x mod m s.t. |a| < a_bound, b>0
   */
  bool reconstructRational(Element& a, Element& b,
                           const Element& x, const Element& m,
                           const Element& a_bound) const {
    ++RecCounter_;
    (...)
    //Now, 0 < = x_in < m analogously
    return _RR.reconstructRational(a,b,x_in,m,a_bound);
  }

  /*
   * Returns a/b=x mod m s.t. |a| < a_bound & 0< b < b_bound
   */
  bool reconstructRational(Element& a, Element& b,
                           const Element& x, const Element& m,
                           const Element& a_bound, const Element& b_bound) const {
    ++RecCounter;
    (...)
    //Now, 0 < = x_in < m analogously
    Element bound = x_in/b_bound;
    _RR.reconstructRational(a,b,x,m,(bound>a_bound?bound:a_bound));
    bool res= (b > b_bound)? false: true;
    return res;
  }
};
```

Currently, four classes are given for RRBase. Fig. 12.2 shows the class hierarchy for RRBase.

In "linbox/algorithms/classic-rational-reconstruction.h", ClassicRationalReconstruction is implemented, which uses classic extended Euclidean algorithm and Wang's [129] bounds

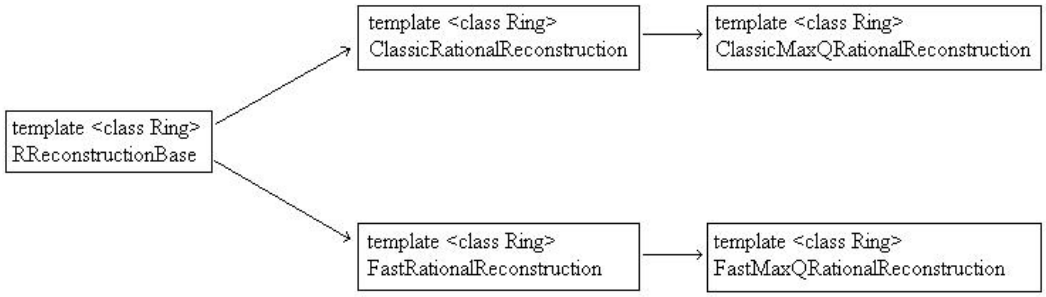

Figure 12.2: Hierarchy for RReconstructionBase

to implement the default method. Namely, the implementation the default reconstructRational is

```
reconstructRational(Element& a, Element& b, const Element& x, const Element& m) {
    reconstructRational(Element& a, Element& b,
                        const Element& x, const Element& m, sqrt(m/2));
    return (b <  sqrt(m/2));
}
```

The classic procedure for `reconstructRational(Element& a, Element& b, const Element& x, const Element& m, a_bound)` is realized by the classic extended Euclidean algorithm EEA, see e.g. [70].

Class ClassicMaxQRationalReconstruction reimplements `reconstructRational(Element& a, Element& b, const Element& x, const Element& m)` using the condition of Monagan [90]. That is, it computes the pair corresponding to sufficiently large quotient in the quotient sequence for (m, x). In our implementation, `reconstructRational(Element& a, Element& b, const Element& x, const Element& m)` outputs pair (a, b) corresponding to quotient `q > m.bitsize() + 5`, see Sec. 11.6.2 for experimental evaluation. It returns false if no such quotient is found.

In "linbox/algorithms/fast-rational-reconstruction.h", class FastRationalReconstruction and its child FastMaxQRationalReconstruction are implemented. The definitions of the default `rationalReconstruction` are analogous as for classic reconstructions. Fast rational reconstruction algorithm of [131] is used, see Alg. 11.4.3,11.5.1,11.5.2 and Ch. 11 for discussion.

13

Preconditioning for Rational Computation

The aim of this chapter is to present the preconditioning methods that are available for the computation with rational matrices in the case of linear system solving, determinant and characteristic or minimal polynomial computation.

First, based on the ideas in Sec. 9.1, we propose matrix preconditioners, that allow us to deal with integer instead of rational matrices. In Sec. 13.1 we discuss relations between the solutions to the initial rational problem and its preconditioned integer variant in the before-mentioned cases. In Sec. 13.2 we discuss the increase in norm, which might be caused by preconditioning.

Then in Sec. 13.3 we present preconditioners for the results, which allow us to run (preconditioner) CR Alg, 10.1.1 for integer instead of rational value. Therefore we do not need to use rational reconstruction, which often leads to simpler early termination strategy in lower number of iterations, compared to the rational variant of the algorithm.

The idea of the preconditioned CRA in the general case has been presented in Ch. 10. In Ch. 6,8 we have shown how to apply it to the computation of the integer determinant. Contrary to Alg. 8.4.1, in the case of rational computations we will need multiplicative preconditioners.

In Sec. 11.3 we discussed the problem of rational CRA and remarked that efficient combination of rational reconstruction an early termination in CRA is more difficult than in the integer case. In particular, scheduling of rational reconstruction is necessary to limit the cost, which comes at the expense of early termination. We recall here that by scheduling the reconstruction, we obtain asymptotically optimal early termination, in the sense of Sec. 10.7. In the case of geometric scheduling, the number of steps can be twice the optimal, whereas incremental integer reconstruction leads to early termination with probability $1 - \epsilon$ after $O(\log(1/\epsilon)) \in O(1)$ supplementary steps. Thus, while computing a result of the same size by integer and rational CRA, we expect integer CRA to perform better in practice.

Sec. 13.3.1 is a case study of result preconditioners for linear system solving, determinant and characteristic or minimal polynomial computation. The aim of preconditioning is

to approximate the denominator of the result. For sake of completeness, in Sec. 13.3.2 we present the bounds on numerators as well. In Sec. 13.4 we evaluate the quality of approximations experimentally.

13.1 Obtaining Integer Matrices by Preconditioning

In Sec. 9.1 we proposed representing rational matrix $A = \left[\frac{a_{ij}}{b_{ij}}\right]$ as $(D(A), A'), (\mathrm{diag}(D_i), \tilde{A}_1)$ or $(\mathrm{diag}(E_i), \tilde{A}_2)$, see Eq. (9.1), (9.2). Recall that $D(A)$ is the common denominator of entries of A and D_i and E_j are common denominators of the entries in the ith row and jth column respectively; $\|A\|_r = \max(|a_{ij}|, |b_{ij}|)$ is the (rational) norm of A.

Matrices $A', \tilde{A}_1, \tilde{A}_2$ are constructed by multiplying A by a scalar or a diagonal matrix. Hence, we will say that they have been obtained by (scalar, matrix) preconditioning of A, or simply that these are preconditioned. For short, we will use the notion $\tilde{A}$ to refer to either $\tilde{A}_1, \tilde{A}_2$ if the distinction between the two is not necessary.

The following propositions explain how to obtain solutions for problems with A by performing computation on preconditioned matrices A',$\tilde{A}_1, \tilde{A}_2$. In Prop. 13.1.1 we analyze the problem of determinant computation and linear system solving and in Prop. 13.1.2 we do the same for the minimal and characteristic polynomial.

Proposition 13.1.1 (Matrix Preconditioning) Let A be a rational $m \times m$ matrix. Let $(D(A), A'), (\mathrm{diag}(D_i), \tilde{A}), (\mathrm{diag}(E_i), \tilde{A})$ be matrix representations defined in Eq. (9.1), (9.2). Then

1. A' and $\tilde{A}$ are integer matrices. If A is symmetric (resp. Hankel, Toeplitz etc.), then A' is symmetric (resp. Hankel, Toeplitz etc.).

2. $$\det(A) = \frac{\det(\tilde{A}_1)}{\prod_i D_i)} = \frac{\det(\tilde{A}_2)}{\det(\prod_i E_i)} = \frac{\det(A')}{D(A)^m}. \tag{13.1}$$

Let $b = [b_i] \in \mathbb{Q}[m]$ be a rational vector of size m, $D(b)$ denote the common denominator of its entries and $D_i(b)$ denote the denominator of b_i. Let $D' = \mathrm{diag}(D_i(b))$. For rational matrix X, let $\tilde{X}_{1(2)}$ denote the matrix obtained by left (right) matrix preconditioning of X as in Eq. (9.2). Then

3. The solution x to the equation $Ax = b$ is given by

$$x = \frac{D(A)}{D(b)} x', \tag{13.2}$$

where x' is the solution to the system $A'x' = b'$, $b' = D(b)b$.

4. Suppose that x_1 is the solution to the integer system $\widetilde{(D'A)}_1 x_1 = D_1 b'$, where D_1 is the diagonal matrix of common denominators for the rows of $D'A$, $b' = D'b$. Then $x = x_1$ is the solution to the rational system $Ax = b$.

5. Suppose that x_2 is the solution to the integer system $\widetilde{(D'A)}_2 x_2 = b'$ and D_2 is the diagonal matrix of common denominators for the columns of $D'A$, $b' = D'b$. Then $x = D_2 x_2$ is the solution to the rational system $Ax = b$.

PROOF 1. In the three cases, all matrix entries are multiplied by a multiple of the denominators. In the case of A', notice that multiplying by a scalar preserves matrix structure.

2. The determinant of a product of matrices is the product of denominators. Notice that $\det(\mathrm{diag}(D_i)) = \prod_i D_i$, $\det(\mathrm{diag}(E_i)) = \prod_i E_i$ and $\det(\mathrm{diag}(D(A))) = D(A)^m$. Thus, Eq. (13.1) follows.

3. We have

$$A'x' = b' \Leftrightarrow D(A)Ax' = D(b)b \Leftrightarrow A\frac{D(A)}{D(b)}x' = b \Rightarrow x = \frac{D(A)}{D(b)}x'.$$

4. We have $\widetilde{(D'A)}_1 = D_1 D'A$. Thus,

$$\widetilde{(D'A)}_1 x_1 = D_1 b' \Leftrightarrow D_1 D'Ax_1 = D_1 D'b \Rightarrow x_1 = x.$$

5. We have $\widetilde{(D'A)}_2 = D'AD_2$. Thus,

$$\widetilde{(D'A)}_2 x_2 = b' \Leftrightarrow D'AD_2 x_2 = D'b \Rightarrow D_2 x_2 = x.$$

In the case of characteristic or minimal polynomial computation the situation is more interesting. We have the following theorem

Proposition 13.1.2 (P_A vs $P_{A'}$) Let $m \in \mathbb{N}$ and let A be a $m \times m$ rational matrix. Let $D(A)$ be the common denominator of entries of A and let $A' = D(A) \cdot A$. Let l, l' be the degrees of the characteristic (minimal) polynomials of A and A' respectively. Suppose that $P_A(x) = \sum_{i=0}^{l} a_i x^i$ where $a_i \in \mathbb{Z}$, and $P_{A'}(x) = \sum_{i=0}^{l'} b_i x^i$ are the characteristic (minimal) polynomials of A and A' respectively. Then $l = l'$ and $a_i = \frac{b_i}{D(A)^{l-i}}$.

PROOF We set $d = D(A)$. Let us consider the case of the characteristic polynomial. We know that the characteristic polynomials of A and dA have the same degree $l = l' = m$ and are unique. From $P_{A'}(A') = 0$ we may deduce that $P_{A'}(dA) = 0$. Therefore $\frac{1}{d^m} P_{A'}(dA) = 0$ and $\frac{1}{d^m} P_{A'}(dx)$ is unitary. By uniqueness we may conclude that $\frac{1}{d^m} P_{A'}(dx) = P_A(x)$ and so $\frac{b_i}{d^{m-i}} = a_i$.

Let us now consider the case of minimal polynomial. By the same reasoning, for minimal polynomial P_A, we may conclude that $P_A(x) \mid \frac{1}{d^{l'}} P_{A'}(dx)$. Similarly, from $P_A(\frac{A'}{d}) = 0$ we may deduce that $P_{A'}(dx) \mid d^l P_A(x)$ and so both polynomials have the same degree $l = l'$ and are in fact equal. We obtain $\frac{b_i}{d^{m-i}} = a_i$ as in the previous case.

Unfortunately, multiplying by diagonal matrix might change the minimal and characteristic polynomial. Thus, computation of the characteristic or minimal polynomial of $\tilde{A}_1, \tilde{A}_2$ does allow us to recover the characteristic or minimal polynomial of A.

13.2 Increase in Norm Due to Preconditioning and Related Problems

Preconditioning of A comes at a cost. After preconditioning, the norm of $A', \tilde{A}$ can be much bigger than $\|A\|_r$. Prop. 13.2.1 gives the worst case bounds on $\|\tilde{A}\|$ and $\|A'\|$.

Proposition 13.2.1 (Norms of $\tilde{A}, A'$) Let A be a $m \times m$ rational matrix with Ω non-zero entries[1], and $M \in \mathbb{R}$ be such that $\|A\|_r \leq M$.

1. Suppose that $m < M$ (resp. $\Omega < M$). Then $\|\tilde{A}\| \leq M^{m+1}$ (resp. $\|A'\| \leq M^{\Omega+1}$).
2. Suppose that $m < M$ (resp. $\Omega < M$). Then $\log(\|\tilde{A}\|) \in O(m \log(M))$ (resp. $\log(\|A'\|) \in O(\Omega \log(M))$).
3. Suppose that $m \geq M$ (resp. $\Omega \geq M$). Then $\|\tilde{A}\| \leq M \cdot \mathrm{lcm}(1, 2, \ldots, M)$ (resp. $\|A'\| \leq M \cdot \mathrm{lcm}(1, 2, \ldots, M)$).
4. Suppose that $m \geq M$ (resp. $\Omega \geq M$). Then $\log(\|\tilde{A}\|) \in O(M)$ (resp. $\log(\|A'\|) \in O(M)$).

PROOF With no loss of generality we assume that $\tilde{A} = \tilde{A}_1 = \mathrm{diag}(D_i)A$, where D_i are the common denominators for entries in ith row of A. Recall that $D(A)$ is the common denominator of entries of A.

In the case when $\Omega < M$, $D(A)$ is the least common multiple of at most Ω denominators and can be as much as M^{Ω}. Multiply this by one numerator (in the case of each entry) to obtain the bound on $\|A'\|$. Similarly, if $m < M$, D_i, is the lcm of at most m denominators[2] and can reach M^m. The bound on $\|\tilde{A}\|$ follows. The bounds in terms of *Big-Oh* notation are a direct consequence.

In the case when $\Omega \geq M$ or $m \geq M$ resp., $D(A)$ and D_i are the lcm of at most M different denominators and do not exceed $LCM_M = \mathrm{lcm}(1, 2, \ldots, M)$. $\log(LCM_M)$ is asymptotically $O(M)$, which gives the required bounds.

In the general case, when the denominators of matrix entries are randomly sampled, Prop. 13.2.1 gives pessimistic bounds on the norms of preconditioned matrices. An increase in size by a factor of $\min(m, \frac{M}{\log(M)})$ (resp. $\min(\Omega, \frac{M}{\log(M)})$) in the case of $\tilde{A}$ (resp. A') is possible.

However, in applications we often encounter matrices, whose entries are of the same type. Often, we deal with decimal or binary fraction of certain precision, where virtually every

[1] This may be replaced by Ω_r, the number of non-integer fractions in the matrix.
[2] m can be replaced by $\Omega_r(i)$, the number of non-integer fractions in the ith row

entry has the form $\frac{a_{ij}}{10^k}$ or $\frac{a_{ij}}{2^k}$. Then it might happen that $\|\tilde{A}\|, \|A'\|$ and $\|A\|_r$ are more or less of the same size.

A great difference in the approach to rational algorithms is expected in the two cases. In the case of matrices with a wide range of different denominators, the increase in norm caused by preconditioning might prohibit us from using preconditioned matrices $\tilde{A}, A'$ in the computation, limiting the possibilities to rational solutions. On the contrary, in the case of decimal or binary fractions, we can use integer algorithms for A', $\tilde{A}$ without a handicap.

Thus, in order to design rational algorithms, we have to bear in mind the differences between the two classes. Also, the analysis of the algorithms has to be performed in the two cases separately as the asymptotic performance may vary significantly. In Ch. 14 we will propose adaptive algorithms that are meant to distinguish between the two classes in order to switch between the strategies. By applying adaptive approach we expect to be able to treat the intermediate cases as well.

We start the analysis by presenting the results on norm growth for random matrices in Sec. 13.2.1 in order to be able to define a general trend. Then we analyze a large set of interesting problems for rational computation in Sec. 13.2.2, chosen from BasisLIB collection of Dan Steffy[3] [118] and some matrices of Matrix Market[4] Harwell-Boeing collection. We choose Hilbert matrices [17] and Lehmer matrices as a benchmark example of rational matrices with very different denominators.

13.2.1 Case of Random Matrices

In this section we will analyze the growth in norm due to preconditioning for random rational matrices. Following the setup of Sec. 5.1, we start with the definition of the distribution of entries.

We start with the case of rational matrices, which potentially have very different denominators. Let us take $\lambda > 0$ and define $\mathcal{I}$ as the set of all (positive) fractions with numerator and denominator bounded in absolute value by λ i.e. $\mathcal{I} = \mathcal{I}_\lambda = \{\frac{p}{q} : 0 < p \leq \lambda, 0 < q \leq \lambda, \gcd(p,q) = 1\}$. For the norm computation, we can restrict ourselves to positive entries.

At first glance, it might seem natural to consider the uniform distribution on $\mathcal{I}$. Yet, it might be difficult to obtain it experimentally, as the computation of $|\mathcal{I}|$ seems a challenging problem itself. That is, the number of elements in $\mathcal{I}$ can be bounded by:

$$\sum_{i=1}^{\lambda} \phi(i) \lfloor \frac{\lambda}{i} \rfloor \leq |\mathcal{I}| \leq \sum_{i=1}^{\lambda} \phi(i) \lceil \frac{\lambda}{i} \rceil,$$

[3] http://www2.isye.gatech.edu/ dsteffy/rational/, see also http://ljk.imag.fr/membres/Jean-Guillaume.Dumas/Matrices/BasisLIB/

[4] http://math.nist.gov/MatrixMarket/

where $\phi(i)$ denote the Euler's totient function. That is, for each denominator $i = 1, \ldots, \lambda$, we count the number of numerators n_i coprime with i, which is $\phi(i)$. Then we consider all numbers $n_i + ik, k = 0, \ldots, \lfloor \frac{\lambda}{i} \rfloor, \lceil \frac{\lambda}{i} \rceil$ s.t. $n_i + ik \leq \lambda$. Asymptotically we have

$$|\mathcal{I}| \approx \lambda \sum_{i=1}^{\lambda} \frac{\phi(i)}{i} = \lambda^2 (\frac{6}{\pi^2} + O(\frac{\log(\lambda)}{\lambda})) \leq 0.6\lambda^2.$$

It might also be debatable, whether matrices encountered in experiments can be approximated by this distribution.

Therefore, let us focus on a simplified 'almost' uniform distribution on $\mathcal{I}$. The distribution is defined in two steps. In the first step, choose a numerator and denominator randomly and uniformly from $\{1, \ldots, \lambda\}$. Then, form a fraction and normalize it if necessary. Return the resulting fraction.

We will denote this distribution by *Rat*. Notice, that the denominator are not uniformly distributed in this case i.e. for random fraction $\frac{n}{d}$: $\mathcal{P}(d = \lambda_1) \neq \mathcal{P}(d = \lambda_2)$ for $\lambda_1 \neq \lambda_2$. The distribution of denominators modulo p in the asymptotic case can be expressed by the following proposition.

Proposition 13.2.2 (Distribution of Denominators for *Rat*) Let p be a prime. Suppose that the numerator a and denominator b of fraction $q = \frac{a}{b}$ are uniformly and randomly sampled from the set $\{1, 2, \ldots, p^k\}$. Then, fraction is normalized if necessary, so that $q = \frac{a'}{b'}$ and $\gcd(a', b') = 1$. Let us consider the probability $a_{p^l,k}$ that the denominator b' is divisible by p^l, $0 < l \leq k$. Then $a_{p^l,k} \to \frac{1}{p^l + p^{l-1} + \cdots + 1}$ if $k \to \infty$.

PROOF For $l = k$, p^l has to divide b and p cannot divide a. In this case, $a_{p^l,l} = \frac{p-1}{p^{l+1}}$. For $l < k$, we have the following possibilities:

- $p^l \nmid b$, which occurs with probability $\frac{p^l - 1}{p^l}$;
- $p^l \mid b$ and $p \nmid a$, which occurs with probability $\frac{p-1}{p^{l+1}}$; this is the favorable case;
- $p^l \mid b$ and $p \mid a$, which occurs with probability $\frac{1}{p^{l+1}}$; in this case, fraction has to be normalized;

In the last case, let $a_1 = \frac{a}{p}, b_1 = \frac{b}{p}$ denote the numerator and denominator of q after 1 step of normalization i.e. division by p; a_1 and b_1 are uniformly distributed in $\{1, 2, \ldots, p^{k-1}\}$. Thus, recursively, the probability that p divides b' is $a_{p^l,k-1}$.

We have the following recursive formula:

$$a_{p^l,k} = \frac{p-1}{p^{l+1}} + \frac{1}{p^{l+1}} a_{p^l,k-1}, \quad a_{p^l,l} = \frac{p-1}{p^{l+1}}.$$

Thus, $a_{p^l,k}$ converges and the limit $\lim_{k\to\infty} a_{p^l,k} = \frac{p-1}{p^{l+1}-1} = \frac{1}{p^l + p^{l-1} + \cdots + 1}$.

A uniform distribution of denominators might also be considered on set $\mathcal{I}' = \mathcal{I}'_\lambda = \{\frac{p}{q} : 0 < q < \lambda, 0 < p < q, \}$. To construct a fraction, we first randomly choose a denominator d for the set $\{1, \ldots, \lambda\}$ and then randomly choose a numerator for the set $\{1, \ldots, d\}$, co-prime with d. We will denote this distribution by *RatUni*. Notice, that $\|A\| < 1$.

The number of elements in $\mathcal{I}' = \mathcal{I}'_\lambda = \{\frac{p}{q} : 0 < q \leq \lambda, 0 < p < q, \}$ is $\sum_{i=1}^{\lambda-1} \phi(i)$ in this case, and it is possible to effectively simulate this distribution. This is a slightly different approach than the previous one. Intuitively, large composite denominators occur with bigger probability, small prime denominators with smaller and large prime denominators with the same probability as in the previous case. Thus, the distribution of lcm of m random denominators might differ in both cases. Yet, we expect that the trends of Prop. 13.2.1 would be confirmed for both distributions.

In the experiments, we analyzed the average growth of $\frac{\log(\|\tilde{A}\|)}{\log(\|A\|_r)}$ and $\frac{\log(\|A'\|)}{\log(\|A\|_r)}$ for random rational matrices A of size $m = 100, 200, \ldots, 1000$ for $\lambda = 100, 10000$ and both distributions *Rat, RatUni*. In each case, average of 5 results for 5 different random matrices was taken. Additionally, we computed the results for random matrices A of size $m = 100, \ldots, 500$ for $\lambda = MAX\_INT$ and both distributions. Fig. 13.1 presents the dependencies we have found.

In Fig. 13.1 it can be seen, that the results are similar for both distributions *Rat* and *RatUni*. This confirms that the asymptotic evaluation of Prop. 13.2.1 is valid, despite the fact that it uses a pessimistic bound on the lcm of m numbers.

In fact, contrary to implications of Prop. 13.2.2, the growth in norm for distribution *RatUni*, is smaller than for *Rat* by a constant close to 1, for all cases except the case of A' and $\lambda = MAX\_INT$. This might be explained by the fact, that in the case of small $\|A\|_r$, the difference in size of numerators influence the result more than the difference in the distribution of denominators. In the case of $\lambda = MAX\_INT$, it might be possible that difference in the distribution of denominators started to play a role.

In the case of $\lambda = 100$ i.e. $\lambda \leq m$, $\|\tilde{A}\|$ and $\|A'\|$ are equal starting from $m = 200$. The ratio stabilizes at 20.4 for *Rat* distribution and 19.4 for *RatUni*. The result of Prop. 13.2.1 yields $\lfloor \log(\mathrm{lcm}(1, \ldots, 100))/\log(100) \rfloor = 50$ in the worst case.

In the case of $\lambda = 10000$ i.e. $m < \lambda, \lambda \leq m^2$ the ratio stabilizes for A' for $m = 300$ at value 1032.8 for *Rat* and 1031.7 for *RatUni* distributions. For $\tilde{A}$ the ratio grows as a function of m from $0.56m$ for $m = 100$ to $0.29m$ for $m = 1000$. This presents, how the accuracy of approximations of Prop. 13.2.1 fades if the ratio between m and $\|A\|_r$ gets smaller.

In the case of $\lambda = MAX\_INT$ i.e. $m \ll \lambda$, both ratios grow in the considered range of m. For $\tilde{A}$ the ratio grows from $0.84m$ for $m = 100$ to $0.73m$ for $m = 500$, so the growth is close to linear in m in this case. For A' the ratio grows from $0.57m^2$ for $m = 100$ to $0.42m^2$ for $m = 500$ which approximates $O(m^2)$ less accurately.

For the case of matrices consisting of decimal fractions, we consider random matrices with entries being random decimal fractions from [0,1) of certain precision k. Given such distribution, A' is equivalent to a random integer matrix with entries randomly and uniformly sampled from the set $\{0, 1, \ldots, 10^k - 1\}$. Thus, results from Ch. 5 apply.

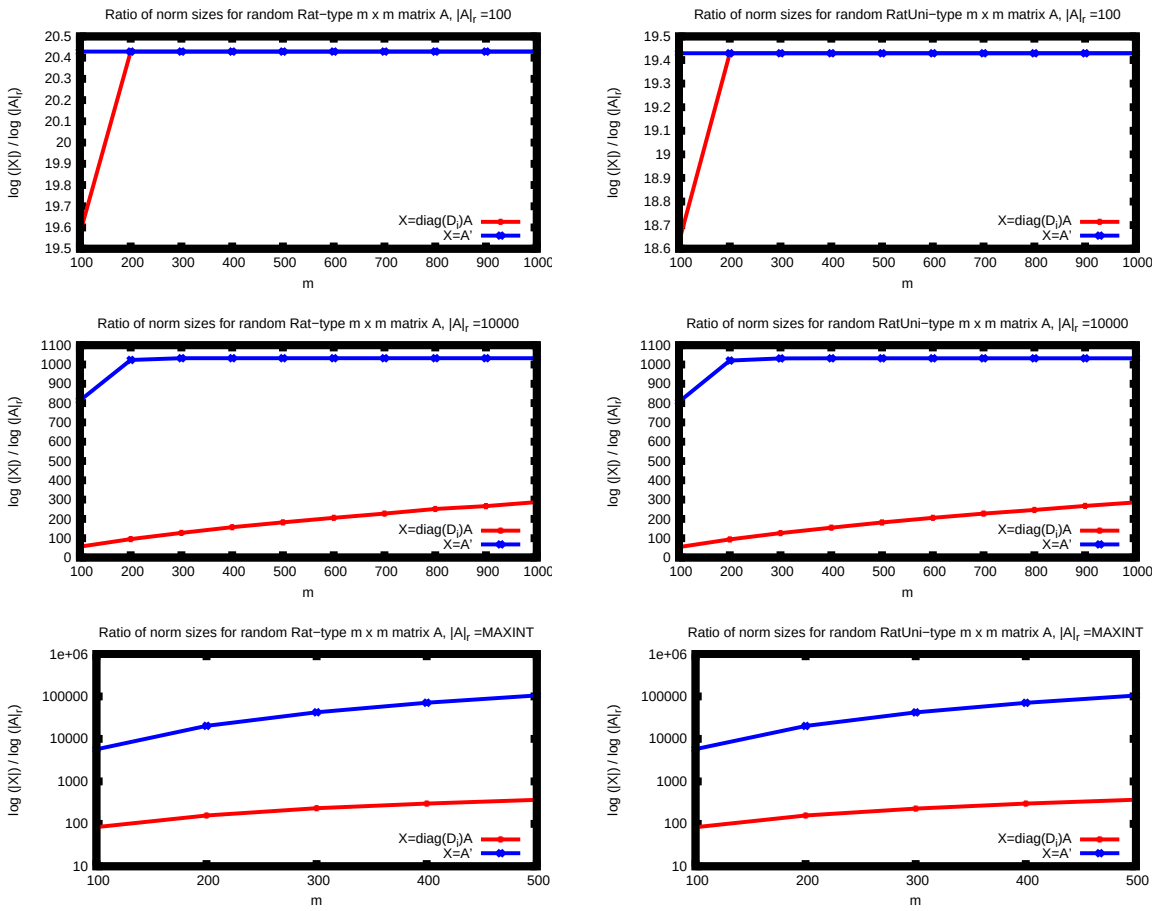

Figure 13.1: The average ratio of $\frac{\log(\|\tilde{A}\|)}{\log(\|A\|_r)}$ and $\frac{\log(\|A'\|)}{\log(\|A\|_r)}$ for random rational matrices A, given according to distribution *Rat* or *RatUni*, for cases $\|A\|_r = 100, 10000, MAX\_INT$. For $\|A\|_r = 100, 10000$ average of 5 result was computed.

Moreover, $\tilde{A}$ is different from A' in an unlikely case, when some rows (resp. columns) of A are reduced when decimal fractions are normalized. Notice, that this is an unlikely case. Indeed, the probability that a all entries in a row of a $m \times m$ random integer matrix A' are divisible by 2 is $\left(\frac{1}{2}\right)^m$. Thus, the probability that $A' \neq \tilde{A}$ is $m\left(\frac{1}{2}\right)^m$.

It is also possible, even less likely that $\|A'\|$ is less than $\|A\|_r = 10^k$. Indeed, this requires that all generated entries are less than $10^k/2$, which occurs with probability $\left(\frac{1}{2}\right)^{m^2}$.

To confirm that this is indeed the case, we generated matrices of decimal fractions of size 100,200,...,1000, for precision $k-1=5$ and 16 decimal places. For each case, 5 matrices were generated. Experimental results confirmed that the size of $\|A\|, \|\tilde{A}\|$ and $\|A'\|$ was the same for all examples tested. Therefore, experimental results confirmed that this a much easier case.

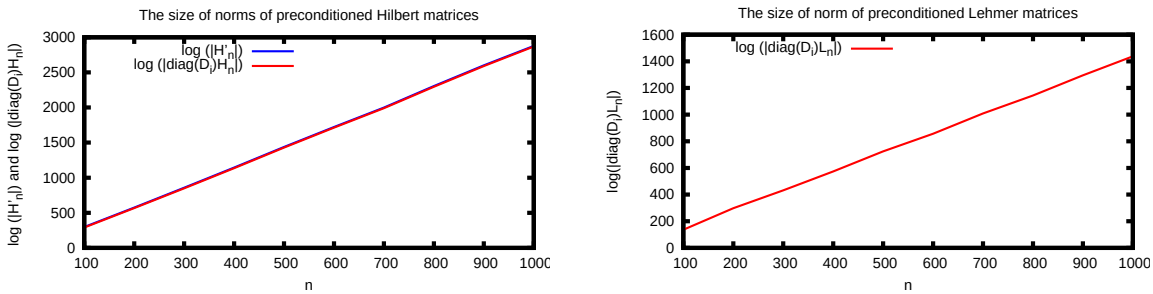

Figure 13.2: Growth of norm for preconditioned Hilbert and Lehmer matrices

13.2.2 Examples of Rational Matrices - Experimental Evaluation

In this section we will analyze the growth in norm due to preconditioning for a wide range of rational matrices, which might be interesting for exact computation. First, let us consider two benchmark examples of ill-conditioned matrices, namely the Hilbert matrices and Lehmer matrices.

The $n \times n$ Hilbert matrix H_n is defined by the equation

$$H_n = \left[\frac{1}{i+j-1}\right]_{i,j=1..n}. \tag{13.3}$$

It is a Hankel-type matrix. The denominators in every row form a set of n consecutive numbers. The rational norm $\|H_n\|_r$ is equal to $2n-1$. In Fig. 13.2 (left), we show the functions $\log(\|\tilde{H}_n\|)$ and $\log(\|H'_n\|)$ as a function of matrix size n. Dependency is linear. In the considered range $n = 100, \ldots, 1000$, functions $\log(\|\tilde{H}_n\|)$ and $\log(\|H'_n\|)$ differ by a small value $c = 7, \ldots, 10$.

The $n \times n$ Lehmer L_n matrix is given by the equation

$$L_n = \left[\frac{\min(i,j)}{\max(i,j)}\right]_{i,j=1..n}. \tag{13.4}$$

The denominators in every row are smaller than n. The rational norm $\|L_n\|_r$ is equal to n. For Lehmer matrices our experiments suggest that, $\|L_n\| = \|L'_n\|$. In Fig. 13.2 (left), we show the functions $\log(\|\tilde{L}_n\|)$ as a function of matrix size n. Dependency is linear in n. In the considered range $n = 100, \ldots, 1000$ functions $\log(\|\tilde{H}_n\|)$ and $\log(\|H'_n\|)$ were equal.

Next, we would like to confront real-live computational examples, with particular emphasis to sparse matrices. Most of the data e.g. in the MatrixMarket collection, consists of matrices with entries being decimal or binary fractions. However, a collection of reasonable size of 157 truly rational matrices BasisLib was recently made available by Dan Stuffy. These matrices arise form problems in linear programming, and some of them are numerically nonstable. For the description of the set see [118]. In some cases, only a few entries are not integer.

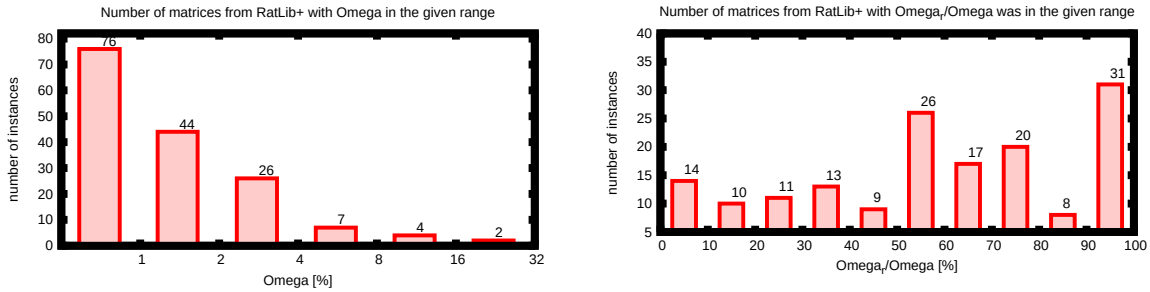

Figure 13.3: Sparsity [in %] and the ratio of non-integer elements to non-zero elements for matrices of BasisLib+ Collection.

In order to make sure that matrices of decimal fractions are represented as well, we added 11 ill-conditioned matrices from the Matrix Market [5] Harwell-Boeing collection, sets: Grenoble, Astroph and Bcsstruc3. Unfortunately, we were not able to compute the norms for 10 biggest matrices due to memory shortage. We refer to the modified collection as BasisLib+.

Fig. 13.3 presents the sparsity of matrices in the collection and the number of rational entries as a function of sparsity. Matrices are very sparse, with 47.8% of the collection having less than 1% non-zero entries, 75.47% less than 2% and 94.34% less than 5%. Only one matrix has 29.45% non-zero entries. For almost 20% of matrices, most of the non-zero entries (i.e. $> 90\%$) are truly rational. At the same time, for 8.81% of matrices less than 10% of entries are rational, which makes them almost integer. Thanks to this sparsity, we expect that preconditioning might be useful here.

Fig. 13.4 presents norm growth for matrices of BasisLib+ collection. For 19 matrices, $\|\tilde{A}\| = \|\operatorname{diag}(D_i)\|A$ was actually smaller than $\|A\|_r$, which is only possible if $\|A\| < 1$. For a majority of matrices, the ratio $\frac{\log(\|\tilde{A}\|)}{\log(\|A\|_r)}$ was 1 or smaller than 2. Then for 97.5% matrices, the ratio was less than 16. The growth in norm is definitely less significant than in the case of random matrices. At the same time for 52 instances, the size of entries of $\tilde{A}$ was bigger than 64. In 48 of those cases, numerators and denominators of A are wordsize.

The ratio $\frac{\log(\|A'\|)}{\log(\|A\|_r)}$ is presented in the right part of Fig. 13.4. The increase in norm is much more significant, although the ratio is still less than 2 for 62.9% of matrices. In the worst case, $\log(\|A'\|)$ was 7244, 1591 and 642 bigger than $\log(\|A\|_r)$.

Based on the ratios, we divided the BasisLib+ collection on three subsets:

1. Easy: for which $\|A\|_r \geq \|\tilde{A}\|$: Total of 76 examples [6]. For 28 of matrices, $\|A\|_r = \|A'\|$.

[5] http://math.nist.gov/MatrixMarket/

[6] List of files: 80bau3b, *UMTS*, baxter, bcsstk21.mtx, bnl2, *car4*, *cont4*, *cr42*, *dano3mip.pre*, *dano3mip*, *danoint*, delf000, dfl001.pre, dfl001, *ganges*, *gen2*, gre_1107.mtx, *gre_216a.mtx*, *gre_216b.mtx*, gre__115.mtx, *gre__185.mtx*, *gre__343.mtx*, *gre__512.mtx*, greenbea, *greenbeb*, *iiasa*, large000, *maros-r7*, mcca.mtx,model11, *model9*, *modszk1*, momentum2, msc98-ip, nemspmm1, neos1, neos7, p010, p05, p19, *qiu*, *qiulp*, r05, *rat1*, *rat5*, *rat7a*, rd-rplusc-21, *rentacar*, roll3000, route, sc205,*scagr7-2r-864*, scfxm1-2b-16, scfxm1-2b-4, scfxm1-2c-4,

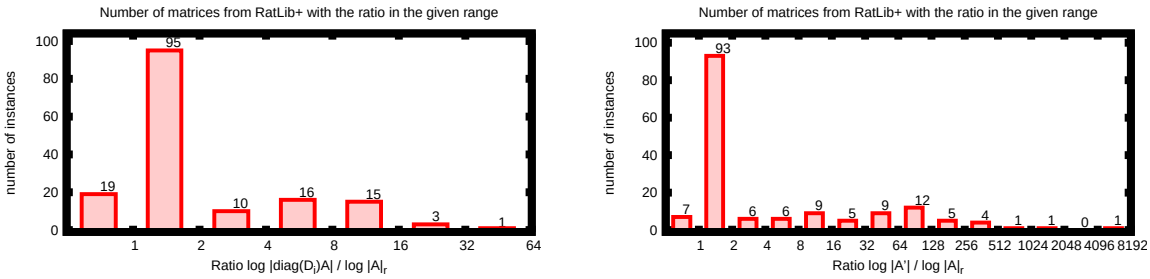

Figure 13.4: Number of instances of matrices from BasisLib+ Collection, for which the growth of norm size for preconditioned matrices (as a ratio of $\log(\|A\|_r)$) was in the given range. Left: ratio $\log \|\tilde{A}\| / \log(\|A\|_r)$. Right: ratio $\log(\|A'\|) / \log(\|A\|_r)$.

2. Intermediate: for which $\|A\|_r < \|\tilde{A}\| = \|A'\|$: Total of 11 examples[7].

3. Difficult: for which $\|A\|_r < \|\tilde{A}\| < \|A'\|$: Total of 71 examples[8].

We have also identified 11 most difficult examples for which the ratio $\frac{\log(\|\tilde{A}\|)}{\log(\|A\|_r)}$ was greater than 12. We summarize the parameters in Tab. 13.1. A high ratio of rational elements to non-zero elements is a common features of the examples. It was also observed that $\log(D(A)) \approx \log(\|A'\|)$ and $\max(\log(D_i)) \approx \log(\|\tilde{A}\|)$.

In their paper, see [118], the authors considered the problem of solving a linear system $Ax = b$, where A was a matrix from BasisLib collection and b was a rational vectors, which can also be found in BasisLib. Exact methods were applied to integer matrices obtained from rational one by scaling of entries. Details of scaling were not given, but scaled integer matrices are provided, which leads to conclusions that scaling followed Prop. 13.1.1 (4). Same approach was considered in [117] for Hilbert and Lehmer matrices.

The authors did experience norm swell in the process which affected the exact computation. The norm growth for the matrix of Prop. 13.1.1 (4) was definitely more important than for $\tilde{A}$, see Fig. 13.5. The difference was due to the choice of vector b, the denominators of which had to be considered as well. This motivates our interest in methods other than matrix preconditioning, which we will consider in the next section.

`scfxm1-2r-16`, `scfxm1-2r-27`, `scfxm1-2r-32`, `scfxm1-2r-4`, `scfxm1-2r-64`, `scfxm1-2r-8`, `scfxm1-2r-96`, `scfxm2`, `scfxm3`, `scorpion`, `scrs8-2c-64`, `scrs8-2r-128`, `scrs8-2r-256`, `scrs8-2r-32`, `scrs8-2r-512`, `scrs8-2r-64`, *`self`*, `small000`, `south31`, *`stair`*, `stocfor2`, `stocfor3`. For examples in itallic, additionally, $\|\tilde{A}\| = \|A'\|$.

[7] List of files: `gosh`, `gran`, `l30.pre`, `l30`, `mkc`, `mkc1`, `nemsemm2`, `stat96v4`, `watson_1.pre`, `watson_2`, `woodw`.

[8] `25fv47`, `arki001`, `bandm`, `baxter.pre`, `bcsstk19.mtx`, `bcsstk20.mtx`, `bnl1`, `boeing1`, `capri`, `ch`, `co5`, `co9`, `cq5`, `cq9`, `cycle`, `d2q06c`, `de063155`, `de063157`, `de080285`, `dsbmip`, `ge`, `gen1`, `gen4.pre`, `gen4`, `gesa2_o`, `gesa3`, `gesa3_o`, `grow15`, `grow22`, `grow7`, `jendrec1`, `l9`, `maros`, `mcfe.mtx`, `mod2.pre`, `mod2`, `model10`, `model2`, `model3`, `model4`, `model5`, `model6`, `model7`, `momentum1`, `momentum3`, `nemspmm2`, `nemswrld`, `nesm`, `newman`, `newman2`, `newman3`, `nl`, `orna1`, `perold`, `pilot.ja`, `pilot`, `pilot.we`, `pilot4`, `pilot4i`, `pilot87.pre`, `pilot87`, `pilotnov`, `pldd000b`, `progas`, `scrs8`, `scsd8`, `slptsk`, `stat96v1`, `stat96v5`, `ulevimin`, `world`.

Matrix	n	$\frac{\Omega}{n^2}$ [%]	$\frac{\Omega_r}{\Omega}$ [%]	$\frac{\log(D(A))}{\max(\log(D_i))}$	$\log(\lVert A\rVert_r)$	$\log(\lVert\tilde{A}\rVert)$	$\frac{\log(\lVert\tilde{A}\rVert)}{\log(\lVert A\rVert_r)}$	$\log(\lVert A'\rVert)$
gen1	329	10.18	97.33	56.55	56	1576	28.14	89116
gen4.pre	367	6.92	98.05	3.87	30	1061	35.37	4111
gen4	375	6.34	97.90	4.33	30	926	30.87	4009
jendrec1	1779	1.08	100.0	478.67	45	687	15.27	325983
nemspmm2	949	0.72	69.22	3.61	40	603	15.08	2194
nemswrld	2205	0.27	68.48	3.47	38	506	13.32	1740
pilot.ja	567	1.18	78.79	7.9	57	856	15.02	6670
pilot4	289	3.36	88.70	11.24	48	737	15.35	8185
pilot87.pre	1540	1.3	95.19	19.13	44	634	14.41	12101
pilot87	1625	1.19	94.81	17.77	41	634	15.46	11241
slptsk	2315	0.64	100.0	39.59	40	652	16.3	25696

Table 13.1: Parameters for 11 most difficult examples of collection BasisLib+ based on the ratio $\frac{\log(\lVert\tilde{A}\rVert)}{\log(\lVert A\rVert_r)}$.

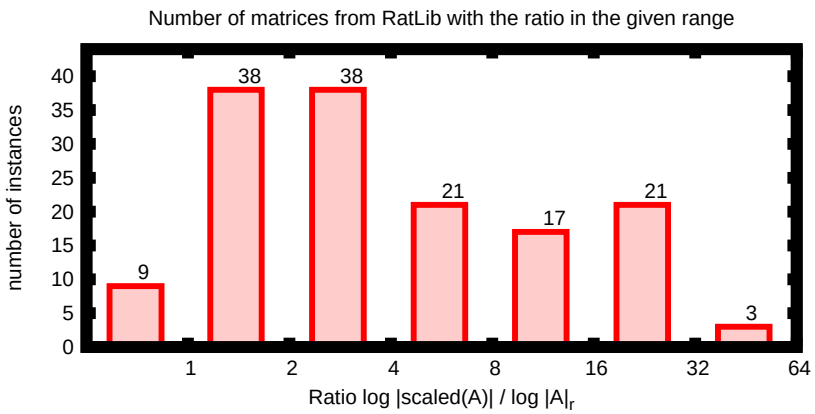

Figure 13.5: Number of instances of scaled integer matrices from BasisLib Collection, for which the growth of norm size (as a ratio of $\log(\lVert A\rVert_r)$) was in the given range. Matrices were used for system solving, the growth depends on rational vector for which solving was performed. Matrices and vectors are distributed in BasisLib Collection.

13.3 Preconditioning of Result - Case Study

Even if we do not choose to precondition the matrix we may still try to deduce some information on the rational solution just by looking at matrix denominators and numerators. If a multiple of denominator is known, the CRA of p-adic scheme can be applied to look for an integer value, which leads to easier termination strategies and often to lower number of iterations.

Assume that $\frac{n}{d}$ is the fraction which we are going to reconstruct and that D is a multiple of d. CRA and p-adic lifting scheme will find the result after at least $\log_p(2nd)$ steps if primes greater than p are used. This can be realized by the use of early termination strategy of Monagan, [90]. However, Wang's [129] termination strategy requires $\log_p(2\max(n^2, d^2))$ steps.

Instead, we may apply the schemes to look for the integer value $\frac{D}{d}n$. CRA and p-adic lifting will require $\log_p(2\frac{D}{d}n)$ steps in order to reconstruct the result. The new number of steps is no more than the number of steps in the previous case provided that $\frac{D}{d} \leq d$. That is, preconditioned strategy can preform better if at least half of bits of the preconditioner is used to approximate d. Additionally, we may skip the rational reconstruction and use a simple termination technique of Lem. 10.6.3.

In the case of system solving, the multiple of denominator cannot be found, but we may compute a preconditioner D', such that $\gcd(D', d)$ is large. Most of the fraction $\frac{D'}{d}$ is then reduced i.e. $\frac{D'}{d} = \frac{D'/\gcd(D',d)}{d/\gcd(D',d)}$ Then, CRA and p-adic lifting schemes will require $\log_p(2\frac{d}{\gcd(D',d)}\frac{D'}{\gcd(D',d)}n$ steps, which is not worse than in the non-preconditioned variant, provided that

$$\frac{d}{\gcd(D', d)}\frac{D'}{\gcd(D', d)} \leq d \Leftrightarrow D' \leq (\gcd(D', d))^2.$$

Analogously as in the case of integer preconditioning, preconditioned strategy is superior if at least half of bits of D' is used to approximate d. Notice, that only the termination strategy of Monagan [90] can be used to early terminate the CRA and p-adic schemes in this case, as Wang's termination strategy may in fact take longer to find the result.

13.3.1 Approximating Denominators

The following proposition gives the approximation of denominators in the case of determinant and characteristic polynomial computation.

Proposition 13.3.1 (Preconditioning of Denominators) Let $A = [\frac{a_{ij}}{b_{ij}}]_{i,j=1\ldots m}$ be a rational $m \times m$ matrix. We define the row (resp. column) common denominators $D_i = \mathrm{lcm}_j(b_{ij})$ (resp. $E_j = \mathrm{lcm}_i(b_{ij})$) in the usual way and we set

$$D = \gcd(\prod_{i=1}^{m} D_i, \prod_{i=1}^{m} E_i). \tag{13.5}$$

Then

$$\begin{aligned} D \cdot \det(A) &\in \mathbb{Z}, \\ D \cdot P_A &\in \mathbb{Z}, \\ D \cdot p_A &\in \mathcal{A}, \end{aligned}$$

where $\det(A)$ is the determinant of A, P_A is the characteristic polynomial of A and p_A is the minimal polynomial of A.

PROOF For the determinant, the result follows immediately from the Gauss formula. In the case of the characteristic polynomial $P_A = \sum_{i=0}^{m} c_i x^i$ we may also use the formulae for the coefficients, which says that for $i = 0, \ldots, m$, c_{m-i} is the sum of all principle $i \times i$

minors. D is the common multiple of the denominators of all principle minors, hence, DP_A is an integer polynomial. The case of minimal polynomial follows from Thm. 13.3.3 that we are going to present next.

For the minimal and characteristic polynomial, the result can further be improved.

Theorem 13.3.2 (Preconditioning for P_A and p_A) *Let A be a rational matrix. Let D be defined as in Eq. (13.5) and $D(A)$ be the common denominator of entries of A. Let $P_A = \sum_{i=0}^{m} c_i x^i$ be the characteristic polynomial of A and $p_A = \sum_{i=0}^{l} c'_i x^i$ be its minimal polynomial. Then*

$$\gcd(D(A)^{m-i}, D)c_i \in \mathbb{Z}, \quad \textit{for} \quad i = 0, \ldots, m; \tag{13.6}$$

$$\gcd(D(A)^{l-i}, D)c'_i \in \mathbb{Z}, \quad \textit{for} \quad i = 0, \ldots, l. \tag{13.7}$$

PROOF Let us set $d = D(A)$. Prop. 13.1.2 implies that $d^{m-i}c_i \in \mathbb{Z}$ and $d^{l-i}c'_i \in \mathbb{Z}$. From Prop. 13.3.1 $Dc_i \in \mathbb{Z}$ and $Dc'_i \in \mathbb{Z}$ as well. Thus, Eq. (13.6) follow.

The result on minimal polynomial is slightly more technical and is subject to Thm. 13.3.3, which we present here. In particular, the same claim might not necessarily be true for any factor p of rational polynomial P, see Ex. 13.3.9. We thank to J.-G. Dumas for pointing the proof of Thm. 13.3.3 to us. We have the following theorem.

Theorem 13.3.3 *Let P_A and p_A be respectively the minimal and characteristic polynomial of a rational matrix A. Let $d \in \mathbb{Z}$ be such that $dP_A \in \mathbb{Z}[X]$. Then $dp_A \in \mathbb{Z}[X]$ as well.*

Before we give the proof to Thm. 13.3.3, we have to introduce the following definitions and lemmas.

Definition 13.3.4 (Primitive Polynomial, Content and Primitive Part) 1. A polynomial in $\mathbb{Z}[X]$ is primitive iff the gcd of its coefficients is equal to 1.

2. The *content*, $\mathcal{C}ont(P) \in \mathbb{Q}$, and *primitive part*, $\mathcal{PP}(P) \in \mathbb{Z}[X]$ of $P \in \mathbb{Q}[X]$ are the unique positive rational and integer polynomial such that $P = \mathcal{C}ont(P)\mathcal{PP}(P)$ and $\mathcal{PP}(P)$ is primitive.

Lemma 13.3.5 (Gauß'lemma) *The product of two primitive polynomials is primitive.*

PROOF See e.g. [89, III.4.2]

Lemma 13.3.6 (Content and Primitive Part) *Content and primitive part are well defined.*

PROOF Suppose that pairs (d_1, P_1) and (d_2, P_2) are content and primitive part of a polynomial $P \in \mathbb{Q}[X]$. We have $d_1P_1 = d_2P_2$, which implies that $\frac{d_1}{d_2}P_1 = P_2 \in \mathbb{Z}[X]$. This means that the denominator $D(\frac{d_1}{d_2})$ must divide every coefficient of P_2 and therefore their gcd. Since P_2 is primitive, the gcd is one, and this proves that $D(\frac{d_1}{d_2}) = 1$. Thus, $\frac{d_1}{d_2} \in \mathbb{Z}$. By the same reasoning, $\frac{d_2}{d_1} \in \mathbb{Z}$ as well. Therefore $\frac{d_1}{d_2} = \pm 1$. As d_1, d_2 are both positive, we may conclude that $d_1 = d_2$ and, consequently, $P_1 = P_2$.

Lemma 13.3.7 (Content of Unitary Polynomial) *Let $P = X^m + \sum_{i=0}^{m-1} q_i X^i$ be a unitary polynomial with rational coefficients $q_i = \frac{a_i}{b_i}$, $a_i, b_i \in \mathbb{Z}$, $b_i > 0$, $\gcd(a_i, b_i) = 1$ for $0 = 1, \ldots, m-1$. Let us set $b = \mathrm{lcm}_i(b_i)$. Then $\mathcal{C}ont(P) = \frac{1}{b}$.*

PROOF $b = \mathrm{lcm}_i(b_i)$ is the smallest positive integer such that $bP \in \mathbb{Z}$. Thus, $\mathcal{C}ont(P) = \frac{a}{b}$ for certain $a \in \mathbb{Z}, a > 0$. We have $\mathcal{PP}(P) = \frac{1}{\mathcal{C}ont(P)} P$, which in the case when P is unitary means that $\mathcal{PP}(P) = \frac{b}{a} X^m + \ldots$. The condition $\mathcal{PP}(P) \in \mathbb{Z}[X]$ implies that $a = 1$ and thus, $\mathcal{C}ont(P) = \frac{1}{b}$.

Lemma 13.3.8 *Let P be a polynomial with rational coefficients and $d \in \mathbb{Z}$ be such that $dP \in \mathbb{Z}[X]$. Then $d\mathcal{C}ont(P) \in \mathbb{Z}$ as well.*

PROOF Let us introduce the following notation. For polynomial $F \in \mathbb{Z}[X]$, $F = \sum_{i=0}^{m} f_i X^i$, define $\gcd(X) = \gcd_i f_i$.

Define $h = \gcd(dP)$. Then $dP = hP'$, where $P' \in \mathbb{Z}[X]$ and $\gcd(P') = 1$. Thus, by uniqueness, $P' = \mathcal{PP}(P)$. Additionally, uniqueness implies that $\mathcal{C}ont(P) = \frac{h}{d}$. Therefore $d\mathcal{C}ont(P) = h \in \mathbb{Z}$.

Finally, we are ready to prove Thm. 13.3.3.

PROOF (Thm. 13.3.3) From linear algebra we know that the characteristic and minimal polynomials P_A and p_A of A have the same irreducible unitary factors P_i, $i = 1, \ldots, s$. Namely, there exist powers $e_i, f_i \in \mathbb{N}, e_i \geq f_i$ such that

$$P_A(X) = \prod (P_i(X))^{e_i},$$
$$p_A(X) = \prod (P_i(X))^{f_i}.$$

By lemma 13.3.7, we can find $b_i \in \mathbb{N}$ such that $\frac{1}{b_i} = \mathcal{C}ont(P_i)$. By lemma 13.3.5, primitive part of a product of polynomials is the product of primitive parts of the composites. Thus, we have

$$P_A = \prod P_i^{e_i} = \prod (\frac{1}{b_i} \mathcal{PP}(P_i))^{e_i} = \prod (\frac{1}{b_i^{e_i}} \prod \mathcal{PP}(P_i)^{e_i},$$

$$p_A = \prod P_i^{f_i} = \prod (\frac{1}{b_i} \mathcal{PP}(P_i))^{f_i} = \prod (\frac{1}{b_i^{f_i}}) \prod \mathcal{PP}(P_i)^{f_i},$$

with $1 \leq f_i \leq e_i$.

By Lem. 13.3.8, $dP_A \in \mathbb{Z}[X]$ implies that $d\mathcal{C}ont(P_A) \in \mathbb{Z}$, that is, $\prod b_i^{e_i} \mid d$. But as $e_i \geq f_i$ for all i, we have that $\prod b_i^{f_i} | \prod b_i^{e_i} | d$. This proves that $dp_A \in \mathbb{Z}[X]$.

In fact, in Thm. 13.3.3 it is essential that both the characteristic and minimal polynomials are unitary. Indeed, for the general case of rational polynomials P, p, $p \mid P$, counterexamples can be given if P or p are not unitary.

Example 13.3.9 1. Let $P = X^2 - \frac{5}{6}X + \frac{1}{6}$ and $p = \frac{1}{2}X - \frac{1}{4}$. Then $p \mid P$ and $6P \in \mathbb{Z}[X]$ but $6p \notin \mathbb{Z}[X]$.

2. Let $P' = 2X^2 - \frac{5}{3}X + \frac{1}{3}$ and $p' = X - \frac{1}{2}$. Then $p' \mid P'$ and $3P' \in \mathbb{Z}[X]$ but $3p' \notin \mathbb{Z}[X]$.

Indeed, we have $P = (x - \frac{1}{2})(x - \frac{1}{3})$, $p = \frac{1}{2}(x - \frac{1}{2})$, so that $p \mid P$ but $6p \notin \mathbb{Z}[X]$. Also, we have $P' = 2(x - \frac{1}{2})(x - \frac{1}{3})$, $p' \mid P'$ but $3p' \notin \mathbb{Z}[X]$.

System Solving

To finish this section, let us consider the problem of solving a linear system of rational equations. In the case of solving $Ax = b$, the denominator of the solution vector x cannot be approximated in advance, i.e. no multiple can be given. Yet, some heuristics might be derived nevertheless, which could help the computation.

We recall that $x_i = \frac{\det([A_i \quad b])}{\det(A)}$, where $[A_i \quad b]$ is equal to matrix A with the ith column replaced by b. Let $N(-)$ and $D(-)$ denote the numerator and denominator of a normalized fraction. Then x is equal to

$$x_i = \frac{D(\det(A))}{N(\det(A))} \frac{N(\det([A_i \quad b]))}{D(\det([A_i \quad b]))}. \tag{13.8}$$

From Prop. 13.3.1 we may deduce that $D(\det(A))$ divides D and that $D(\det([A_i \quad b])$ divides $\mathrm{lcm}(D, D(b))$. Recall that D is given by Eq. 13.5 and $D(b)$ is the common denominator for vector b. If the approximations are 'close', the fraction $\frac{D(\det(A))}{D(\det([A_i \quad b]))}$ is about $\frac{1}{\gcd(D,D(b))}$. Notice that we do not consider this as an approximation in the numerical sense, but we want to say that only a small number of prime factors is missing in the denominator and numerator of $\frac{1}{\gcd(D,D(b))}$ compared to the normalized fraction $\frac{D(\det(A))}{D(\det([A_i \quad b]))}$. According the notion of *bitsize*[9] of [118, 117] this means that the bitsize of

$$\frac{D(\det(A))}{D(\det([A_i \quad b]))} \cdot \gcd(D, D(b))$$

is small.

In general, no division relation has to exist between the two denominators. Still, assuming that most of the fraction $\frac{D(\det(A))}{D(\det([A_i \quad b]))}$ is reduced, x_i is approximated by:

$$x_i \approx \frac{N(\det([A_i \quad b]))}{N(\det(A)) \gcd(D, D(b))}$$

[9] In [117, 118], $bitsize(\frac{p}{q}) = \log(p) + \log(q)$.

in the sense of small *bitsize* of the defect. This would mean that the number of iterations needed to reconstruct x in CRA or p-adic schemes is better than the worst case estimations.

Indeed, in the case when b is a random vector, we may justify that the denominators of $x = A^{-1}b$ can be used to well approximate the denominators of solutions $A^{-1}c$, where c is another integer vector. Indeed, as in Prop. 13.1.1 let us define $A' = D(A)A, b' = D(b)b$. Then, instead of $Ax = b$, we may equivalently solve $A'x' = b'$ and return $x = \frac{D(A)}{D(b)}x'$.

If b is a random vector, then $b' = D(b)b$ is a random integer vector and the common denominator $\tilde{s}_n(A') = D(x_2)$ is a factor of $s_n(A')$. By Thm. 6.3.2, it is 'close' to $s_n(A')$ and both values are probabilistically equal. Precisely, Thm. 6.3.2 requires that at least 2 random vectors are used, but experiments in Sec. 8.8 suggest that one solving gives sufficient approximation. Thus, denominator of $D(x_i)$ of the ith coefficient of x is a multiple of $\frac{D(b)}{\gcd(D(A),D(b))}\frac{\tilde{s}_n(A')}{\gcd(D(A),\tilde{s}_n(A'))}$, which is a 'close' approximation of $\frac{D(b)}{\gcd(D(A),D(b))}\frac{s_n(A')}{\gcd(D(A),s_n(A'))}$.

For given b, A, $d = \frac{D(b)}{\gcd(D(A),D(b)}$ can effectively by computed. It is a product of primes, which appear in $D(b)$ but do not appear in $D(A)$. Thus, given a solution $x = A^{-1}b$, we can derived the approximation of the denominator of $x'' = A^{-1}c$ for another systems as

$$D(x'') \approx \frac{D(c)}{\gcd(D(A), D(c)}\frac{D(x)}{\gcd(D(x), d)}. \tag{13.9}$$

Notice, that for integer vector b, the exact denominator $\frac{\tilde{s}_n}{\gcd(D(A),\tilde{s}_n)}$, which approximates $\frac{s_n(A')}{\gcd(D(A),s_n(A'))}$ can be reconstructed from $D(x)$. The error in this case is solely due to the choice of b and results from Thm. 6.3.2.

By Prop. 13.1.1, the preconditioned matrix $\tilde{A}_1 = \mathrm{diag}(D_i)A$ can be used to solve Eq. $\tilde{A}_1 x = \mathrm{diag}(D_i)b$, which is an integer system as long as $\mathrm{diag}(D_i)b$ is integer. Recall that $\tilde{A}_1$ is given by Eq. (9.2) and D_i are common row denominators. Thus, we will heuristically assume that $s_n(\tilde{A}_1)$ approximates the denominator $D(x)$. This leads to heuristic approximation of $D(x'')$ as

$$D(x'') \approx \frac{D(c)}{\gcd(D(A), D(c)}s_n(\tilde{A}_1), \tag{13.10}$$

where $s_n(\tilde{A}_1)$ is approximated by Alg. 6.3.1 with one integer system solving.

The approach presented for approximating the denominator of a solution to rational system of equations by Eq. (13.10) is based on several heuristic assumptions. This may cause the computed preconditioner to have no division relation with the actual denominator. That is, it is *a priori* neither its factor or a multiple of the denominator. In Sec. 13.4 we will evaluate the quality of preconditioning on numerous examples, thus proving that it can be used in practice.

Remark 13.3.10 In this thesis, we focus on the influence of approximating denominators in the case of CRA and p-adic lifting schemes. It seems however possible, that preconditioning can speed up mixed numeric-symbolic methods as well. This should be the case for the adaptive integer solver of [128] and its rational variant, described in [117]. The observation is based on the fact, that preconditioning reduces the precision to which the result has to be approximated. Still, the system has to be numerically stable in order to apply algorithms of [128, 117].

13.3.2 Bounds on Numerators

The difficulty of finding rational solutions lies in the numerator computation. In order to certify the output of CRA or p-adic schemes, bounds on the numerators and denominators are necessary. In general, if N, D are bounds for the numerator and denominator of the solution, certified scheme requires $O(\log(ND)$ iterations. Bounds on the numerators can be computed by combining the bounds for denominators given in Sec. 13.3.1 with the classic results from linear algebra.

Let $N(-)$ and $D(-)$ denote the numerator and denominator of a normalized fraction. We have the following proposition.

Proposition 13.3.11 (Bounds on Numerators) Let A be a $m \times m$ rational matrix and let $H = m^{\frac{m}{2}} \|A\|^m$ be its Hadamard bound see Prop. 6.1.2.

Let $D(A)$ be the common denominator of entries of A and D be defined by Eq. (13.5).

1. We have: $|N(\det(A))| \leq H|D(\det(A))| \leq HD$.
2. Additionally, let b be a rational vector of size m, and $D(b)$ denote the common denominator of b. Let H' be the Hadamard bound for $[A_i \quad b]$. Let $x = \frac{N(x)}{D(x)}, D(x) \in \mathbb{N}, N(x) \in \mathbb{Z}^m$, be the solution to the rational equation $Ax = b$. We have

$$D(x) \leq HD^2 D(b), \tag{13.11}$$
$$\|N(x)\| \leq H'D^2 D(b), \tag{13.12}$$
$$\tag{13.13}$$

   where $\|N(x)\|$ denote the maximum norm for vector $N(x)$.
3. Let $P_A = \sum_{j=0}^{m} c_j X^j$ be the characteristic polynomial of A. Then

$$\log(|N(c_j)|) \leq H_P + \log(\gcd(D(A)^{m-j}, D)), \tag{13.14}$$

   where $H_P = \frac{m}{2}(\log(m) + log(\|A\|) + 0.21163175)$.
4. Let $p_A = \sum_{j=0}^{t} c'_j X^j$ be the minimal polynomial of A and t be its degree. Then

$$\log(|N(c'_j)|) \leq H_p + \log(\gcd(D(A)^{t-j}, D)), \tag{13.15}$$

   where $H_p = \frac{t}{2}\max(\log(t) + log(\beta), 2\log(\beta))$ and β is the spectral radius of A.

PROOF 1. By Prop. 6.1.2 $\det(A) \leq H$ and by Prop. 13.3.1 $D(\det(A)) \leq D$. The result follows.

2. Eq. (13.8) gives the formula for coefficient x_i. It implies that the common denominator $D(x)$ is a divisor of $N(\det(A))D(\det([A_i \quad b]))$ and the numerators are less than $D(\det(A))N(\det([A_i \quad b]))$. We have

$$\begin{aligned} D(\det(A)) &\leq D \\ N(\det(A)) &\leq HD \\ D(\det([A_i \quad b])) &\leq DD(b) \\ N(\det([A_i \quad b])) &\leq DD(b)H'. \end{aligned}$$

Thus, $D(x) \leq HD^2D(b)$ and $N(x) \leq H'D^2D(b)$.

3. In [109, Lem. 9.1] we are given the bound H_P for the coefficients c_i:

$$\log(|c_j|) \leq H_P = \frac{m}{2}(\log(m) + log(\|A\|) + 0.21163175).$$

Thm. 13.3.2 gives the bound $\gcd(D(A)^{m-j}, D)$ on the denominator of the jth coefficient of P_A. Thus, the bound on the numerator follows.

4. For the minimal polynomial p_A we can use the ovals of Cassini, see [39], and obtain

$$\log(|c'_j|) \leq H_p = \frac{t}{2}\max(\log(t) + log(\beta), 2\log(\beta)), \tag{13.16}$$

where β is the spectral radius of A. By Thm. 13.3.2, the bound on the denominator is given by $\gcd(D(A)^{t-j}, D)$. Thus, the bound on the numerator follows.

The quality of approximating the determinant by the Hadamard's bound has been experimentally studied in [2, 3] in the case of integer matrices. In the case of determinant and minimal/characteristic polynomial computation in the rational case, the precision can be even worse, as the bounds on denominators are obtained heuristically. An eventual miss-approximation carry on to the estimation of the numerators and is repeated twice, when the number of iterations of CRA of p-adic scheme is estimated. In Sec. 13.4 we will attempt to evaluate the approximation experimentally.

In the case of linear system solving, both bounds can be over-estimated by a factor of D^2 as we actually expect that a great part of the fraction could get reduced. Yet, in the worst case, this cannot be assumed. In [118], the authors confirmed experimentally that the bound on the number of iterations of p-adic lifting is indeed significantly over-estimated in the case for rational matrices.

All in all, this implies that early termination is the preferred tool to solve the problems. We refer to Sec. 10.6 and Sec. 11.3 for early termination strategies available. See also [16, 117] for the case of system solving.

13.4 Quality of Preconditioners

In the case when early termination techniques are used in CRA and p-adic solving, the termination of the algorithm is independent of the bound H, H', H_P and H_p defined in Prop. 13.3.11. Yet the choice of matrix/result preconditioning will directly affect early termination, as it changes the size of numbers that are being reconstructed.

In this section, we are going to experimentally evaluate the quality of the preconditioners which have been introduced in Sec. 13.1, 13.4. That is, we need to evaluate the following values.

1. In the case of determinant computation, the ratio $\log(D(\det(A))/\log(D)$, where D is defined in Eq. (13.5) has to be evaluated. In our setting, we will start by evaluating the ratio $\log(D(\det(A))/\log(D)$, where $D = \prod_i(D_i)$ and D_i is the common denominator of the ith row. This gives the upper bound on the required value. Then, we will see if it is possible to improve D.

2. In the case of characteristic polynomial computation, the same ratio $\log(D(\det(A))$ to $\log(D)$ can be used, which describes the approximation of the first and presumably largest coefficient c_0; we assume that the reconstruction of c_0 dominates the computation. Thus, same ratio will be used to characterizes the approximation in the case of minimal polynomial.

3. In the case of rational system solving, we are going to approximate the largest invariant factor $s_n(\mathrm{diag}(D_i)A) \approx D(z)$ first, by solving an integer system of equations $\mathrm{diag}(D_i)Az = b$ with random integer vector b. We will compare $D(z)$ to $N(\det(A))$. Then, we are going to solve a random system $Ax = c$ with rational vector c and check if the approximation $D(x) \approx \frac{D(c)}{\gcd(D(A),D(c))}D(z)$ (see Eq. (13.9)) is justified. Recall that, $D(A)$ (resp. $D(c), D(z)$) denote the common denominator of entries of A (resp. c, z).

Remark 13.4.1 Notice that $\frac{D(c)}{\gcd(D(A),D(c))}D(z)$ might be 'close' to $D(x)$ even in the case when $s_n(\mathrm{diag}(D_i)A) \approx D(z)$ does not approximate $N(\det(A))$ well. This is because $\det(\mathrm{diag}(D_i)A) = \frac{D}{D(\det(A))}N(\det(A))$ and consequently $s_n(\mathrm{diag}(D_i)A)$, which divides $\det(\mathrm{diag}(D_i)A)$, contains factors of $N(\det(A))$ and $\frac{D}{D(\det(A))}$. Thus, the quality of approximation depends on the quality of approximating $D(\det(A))$ and the structure of $\mathrm{diag}(D_i)A$. Experiments are necessary to evaluate the dominant behavior in this case.

For normalized fraction $\frac{p}{q}$, denote by $\mathrm{bs}(q)$ the bitsize of quotient q,

$$\mathrm{bs}(q) = \lfloor \log(p) \rfloor + \lfloor \log(q) \rfloor + 1. \tag{13.17}$$

Thus, for $p \in \mathbb{Z}$, $\mathrm{bs}(p) = \lfloor \log(p) \rfloor + 1$. Recall that $N(-), D(-)$ denote the numerator and denominator of a fraction. For given matrix A, in the case of solving a random system $Ax = c$, c was a random rational vector given according to distribution *Rat* with $\|c\|_r = 100$.

We recall that $D(A)$ is the common denominator of entries of matrix A, D_i is the common denominator for the ith row, $A' = D(A)A$ and $\tilde{A} = \mathrm{diag}(D_i)A$. For integer matrix X, let $\tilde{s}_n(X)$ denote the approximation of $s_n(X)$ by Alg. 6.3.1, obtained by solving one random system of equations. We will use those notations in the remaining part of this section.

We are going to evaluate the following values:

- $D_{err} = \mathrm{bs}(\frac{D(\det(A))}{D}) = \mathrm{bs}(\frac{D}{D(\det(A))}) = \lfloor \log(\frac{D}{D(\det(A))}) \rfloor + 1$; $D_{app} = D = \prod_i D_i$ is taken to approximate $D(\det(A))$. See Prop. 13.1.1 and 13.3.1.
- $N_{err} = \mathrm{bs}(\frac{N(\det(A))}{\tilde{s}_n(\tilde{A})})$; $N_{app} = \tilde{s}_n(\tilde{A})$ is taken to approximate $N(\det(A))$.
- $Dx_{err} = \mathrm{bs}(\frac{D(x)\gcd(D(c),D(A))}{D(c)N_{app}}$; $Dx_{app} = \frac{D(c)}{\gcd(D(c),D(A))}N_{app}$ is taken. See Eq. 13.10 and the related discussion.

13.4.1 Case of Random Matrices

We start with the analysis of the quality of preconditioners for random matrices. We will consider random matrices introduces in Sec. 13.2.1, given according to distributions *Rat*, *RatUni* or by random decimal fractions. See Sec. 13.4 for notations and definition of approximations we are going to evaluate. The error Dx_{err} is evaluated by solving a system $Ax = c$ for random vector c, $\|c\|_r < 100$, where c was given according to *Rat* distribution.)

1. **Matrices of decimal fractions**

   We start with the case of random matrices of decimal fractions. We have evaluated the approximations for 5 random matrices of size $100, 200, \ldots, 1000$ in the case of fraction having 5 and 16 decimal places. Thus, the total set consisted of 100 matrices.

   Fig. 13.6 presents the average errors D_{err}, N_{err} and Dx_{err}. The approximations are very close to the actual values, as the bitsize of the error is never bigger than 9,8 and 8 respectively for all of 100 examples tested. The error is unnoticeable, compared to the size of $N(\det(A))$ and $D(\det(A))$.

   No correlation with matrix size m or matrix norm $\|A\|_r$ has been found, as the average error is constantly small for all matrices tested. Good results of experiments agree with the theoretical evaluation. Indeed, as we remarked in Sec. 13.2.1, A' is equivalent to a random integer matrix with entries chosen uniformly in the range $\{0, \ldots, 10^k - 1\}$. In our experiments $k = 6, 17$ and consequently, $D(A) = 10^6, 10^{17}$ was taken. With high probability, $D_i = D(A)$ for all rows. Thus $D = D(A)^m$ with high probability.

   By Prop. 13.1.1, $\det(A) = \frac{\det(A')}{D(A)^m}$. Thus, the error $\frac{D(A)^m}{D(\det(A))}$ occurs in the case, when the determinant of A' is divisible by 2 or 5. By Thm. 5.4.5, this leads to an error of small size. Indeed, by Thm. 5.4.5, the probability $\mathcal{P}(p^l \mid \det(A'))$,$l > 0$ is less than or equal to $\leq \frac{3}{p^l}$. Thus,

$$\log\left(\frac{D(A)^m}{D(\det(A))}\right) \leq \sum_{l=1}^{\infty}(\frac{3}{2^l} + \frac{3}{5^l}) \leq 4,$$

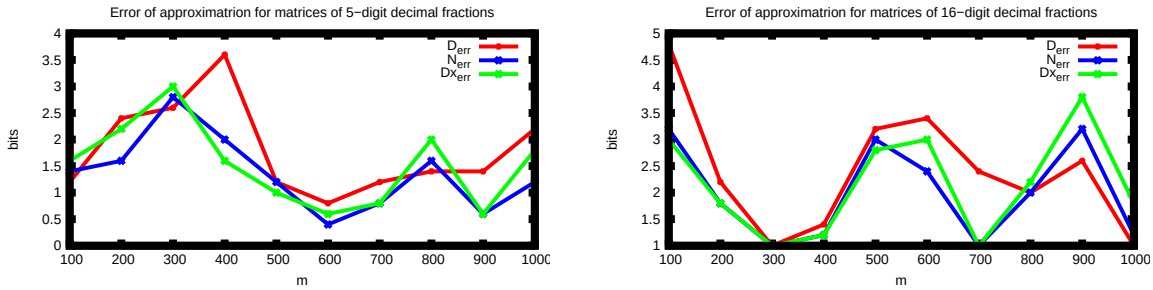

Figure 13.6: Average bitsize of errors D_{err}, N_{err} and Dx_{err}. For each matrix size $m = 100, \ldots, 1000$, average result for 5 random $m \times m$ matrices with entries being decimal fraction with 5 (left) and 16 (right) decimal places is given. N_{app} is equal to $\tilde{s}_n(\tilde{A})$.

and the expected error $\frac{D}{D(\det(A))}$ is at most this big.

In the case when $D = D(A)^m$, the numerator of $N(\det(A))$ is approximated by $\tilde{s}_n(A')$, which is the output of Alg. 6.3.1 for A'. For random matrix A', experiments in Sec. 8.8.1 suggest, that this is close to $\det(A')$ by a few bits, even if only one random system is solved in Alg. 6.3.1. Experiments have confirmed that $\tilde{s}_n(A')$ can have supplementary as well as missing factors compared to $N(\det(A))$. Supplementary factors can be present if $\frac{D(A)^m}{D(\det(A))} > 1$ and missing factors can occur if $\frac{\det(A')}{\tilde{s}_n} > 1$. Both cases were observed during the experiments, and the under-estimation occurred more often.

In the case of solving a system $Ax = c$ of equations, experimental results have confirmed that the heuristics from Sec. 13.3.1 are correct in the case of random matrices of decimal fractions.

2. **Matrices with random entries**

In the case of random matrices with different denominators, we have analyzed distributions *Rat* and *RatUni*, defined in Sec. 13.2.1. We have evaluated the approximations for 5 random matrices of size $100, 150, \ldots, 600$ in the case $\|A\|_r = 100$, 2 random matrices for sizes $50, 100, \ldots, 250$ in the case $\|A\|_r = 10000$ and 1 random matrix for sizes $20, 40, \ldots, 100$ in the case $\|A\|_r = MAX\_INT$. This restriction is due to long running times of the determinant computation, which prevented us to obtain more results in the case when $\|A\|_r$ was bigger. The sets of random matrices consisted of 110 files in the case of $\|A\|_r = 100$, and 20 matrices in the case $\|A\|_r = 10000$ and 10 matrices in the case $\|A\|_r = MAX\_INT$.

Fig. 13.7,13.8 and 13.9 present the average errors of approximations D_{err} and N_{err} of $D(\det(A))$ and $N(\det(A))$ relative to the size of $D(\det(A))$ and $N(\det(A))$ resp. (top) and the average error of approximation as a function of matrix size m (bottom). No significant difference between the two distributions has been found and both absolute and relative errors have comparable values.

N_{app} is always a factor of $N(\det(A))$ for the examples tested. The relative error $Dx_{err}/\operatorname{bs}(D(x))$ is approximately equal to the relative error $N_{err}/\operatorname{bs}(N(\det(A)))$. $D(x)$

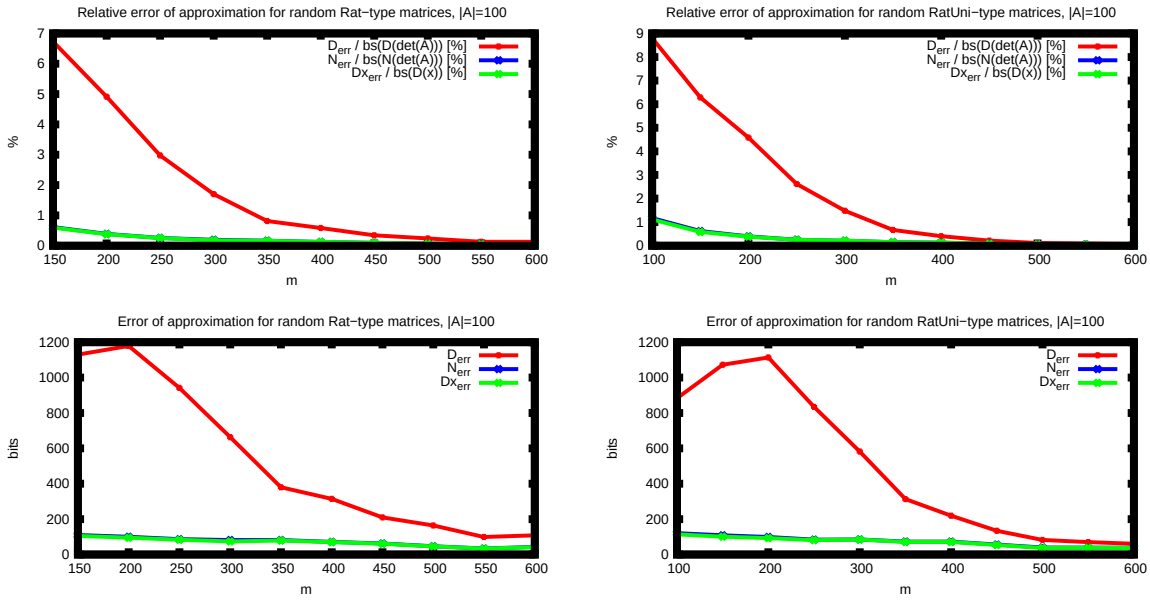

Figure 13.7: Top row: average bitsize of the error $D_{err}, N_{err}, Dx_{err}$; Bottom row: average bitsize of the error relative to the size of $D(\det(A)), N(\det(A)), D(x)$. A is a random matrix given according to *Rat* (left) or *RatUni* (left) distribution. $\|A\|_r$=100.

is often a factor of the approximation, except for 6 out of 110 cases when $\|A\|_r = 100$ and 1 case when $\|A\|_r = MAX\_INT$.

a) Case $\|A\|_r = 100$

Fig. 13.7 presents the evaluation of errors for $\|A\|_r = 100$ and distributions *Rat* and *RatUni*. The results for random matrices are substantially different from the case of random matrices of decimal fractions. First of all, the error is a function of matrix size, reaching its maximum at $m = 200$. D_{err} is over 11 times greater than N_{err} in this case (resp. 11.9 and 11.3 times for *Rat* and *RatUni* distribution). With growing m, the ratio decreases, reaching 2.5 and 1.5 resp for $m = 600$. The absolute error decreases as well, reaching the average of 60 and 38 bits (in the case of D_{err} and N_{err} resp.)

The relative error decreases with m from 8.72 % (resp. 8.8 %) for $m = 100$ to 0.13% (resp. 0.07%) for $m = 600$ in the case of *Rat* (resp. *RatUni*) distributions. This means that asymptotically, the error is insignificant, as it can be recovered by small number of CRA iterations.

Preconditioning by a diagonal matrix has implications on the structure of Smith form of $\tilde{A}$. It can be notice, that size of the product $\mu_{m-1}(\tilde{A})$ of $(m-1)$ smallest invariant factors of $\tilde{A}$ is equal the difference $D_{err} - N_{err}$. This is small compared to $\det(\tilde{A}) = \frac{D_{app}N(\det(A))}{D(\det(A))}$ but could be enough to enforce bonus computation in algorithm Alg. 8.4.1.

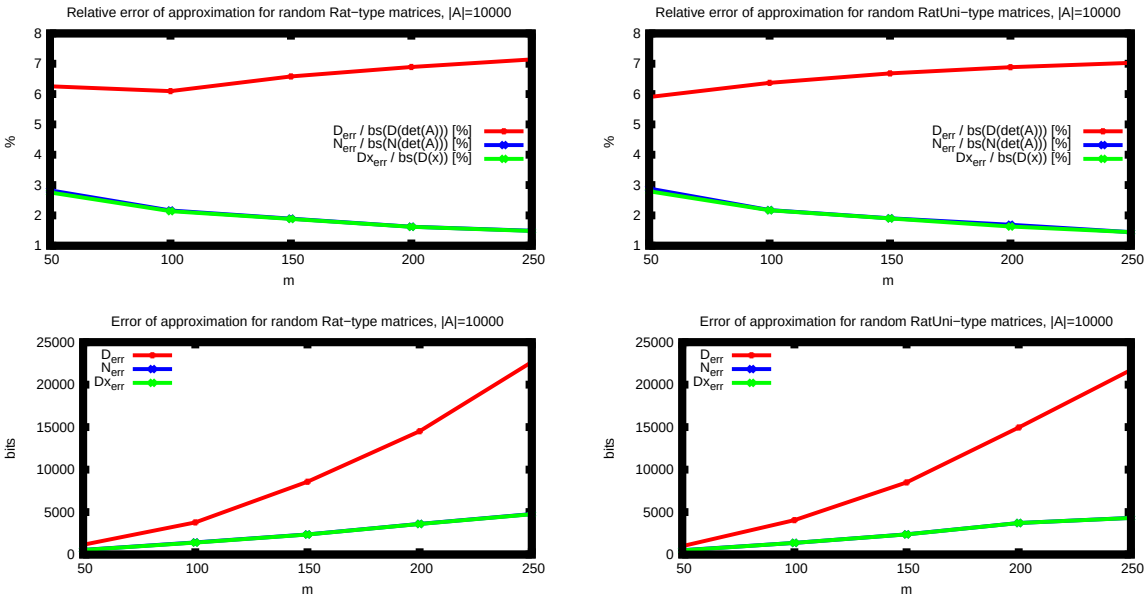

Figure 13.8: Top row: average bitsize of the error $D_{err}, N_{err}, Dx_{err}$; Bottom row: bitsize of the error relative to the size of $D(\det(A)), N(\det(A)), D(x)$. A is a random matrix given according to *Rat* (left) or *RatUni* (left) distribution. $\|A\|_r = 10000$

The structure of $\tilde{A}$ might be influenced by the existence of columns for which the gcd is nontrivial. This might have be caused by choice of numerators of A or by preconditioning. Thus, we propose to correct $\tilde{A}$ in the following way.
For $j = 1, \ldots, m$, let F_j denote the gcd of the jth column of $\tilde{A}$. Let $F = \mathrm{diag}_j(F_j)$. Then we can set $\tilde{\tilde{A}} = \tilde{A}F^{-1}$. The fraction $\frac{D_{app}}{\det(F)}$ is a preconditioner of $\det(A)$ i.e. $\frac{D_{app}}{\det(F)}\det(A) \in \mathbb{Z}$. $\tilde{D}_{app} = \frac{D_{app}}{\gcd(D_{app},\det(F))}$ is the new determinant approximation. Heuristically, we can set $\tilde{N}_{app} = \tilde{s}_n(\tilde{\tilde{A}})\frac{\det(F)}{\gcd(D_{app},\det(F))}$ and $\tilde{Dx}_{app} = \frac{D(b)}{\gcd(D(A),D(b))}N'_{app}$. The exact algorithm for correcting $\tilde{A}$ is given in Alg. 14.1.3. See also Prop. 14.1.2 for details.
We have rerun the experiments for the corrected matrix $\tilde{\tilde{A}}$ and concluded, that the approximations are better by 5 to 47 % in the case of D_{err} (better for bigger m) and 0 to 15 % in the case of N_{err}. Thus, we are going to adopt the correction as the default method and refer to $\tilde{\tilde{A}}$ as $\tilde{A}$ for the rest of this section. We will also skip additional $\tilde{}$ in the notions of errors and approximations.

b) Case $\|A\|_r = 10000$
Fig. 13.8 presents the evaluation of the errors for $\|A\|_r = 10000$ and distributions *Rat* and *RatUni*. The size of errors is increasing with m. D_{err} increases from 1150 (resp. 985) to 22666 (resp. 21728) in the case of *Rat* (resp. *RatUni*) distribution. N_{err} increases from 545 (resp. 512) to 4769 (resp. 4309). Dx_{err} increases from 531 (resp. 489) to 4744 (resp. 4306). The growth seems quadratic for D_{err} and approximately $O(m\log(m))$ for N_{err} and Dx_{err}. D_{err} is 2 to 5 times greater than N_{err} and Dx_{err}, the ratio is increasing with m.

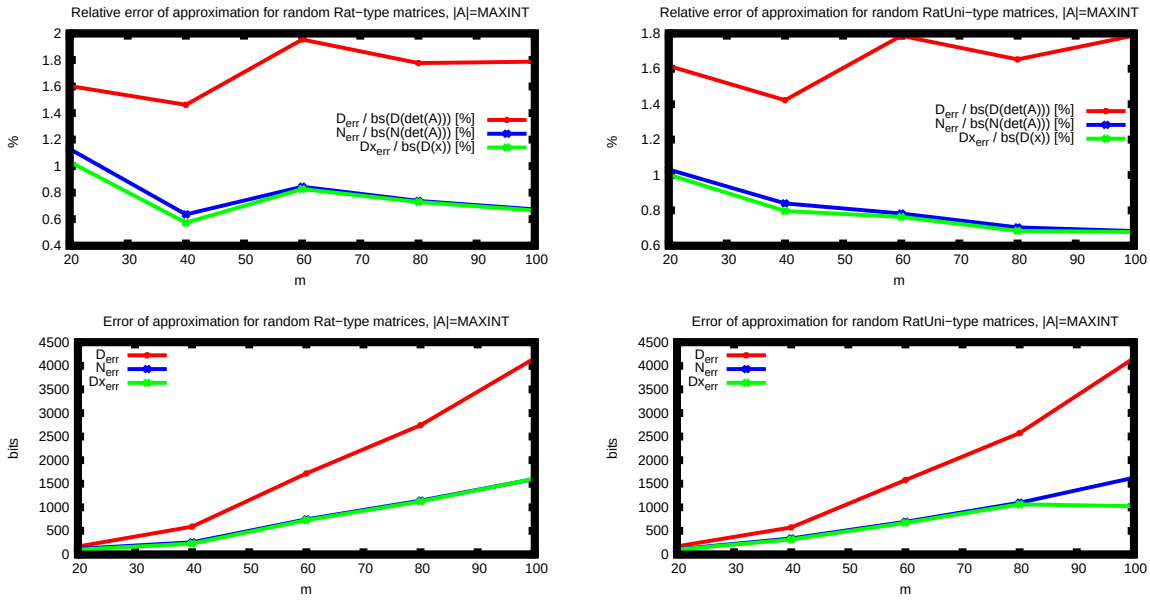

Figure 13.9: Top row: average bitsize of the error $D_{err}, N_{err}, Dx_{err}$; Bottom row: bitsize of the error relative to the size of $D(\det(A)), N(\det(A)), D(x)$. A is a random matrix given according to *Rat* (left) or *RatUni* (left) distribution. $\|A\|_r = MAX\_INT$

The relative error is increasing in the case D_{err} and decreasing in the case of N_{err} and Dx_{err}. The ratio $\frac{D_{err}}{\mathrm{bs}(D(\det(A)))}$ is in the interval $[6.25, 7.15]$ (resp. [5.9,7.03]) for the *Rat* (resp. *RatUni*) distribution. The ratio $\frac{N_{err}}{\mathrm{bs}(N(\det(A)))}$ and $\frac{Dx_{err}}{\mathrm{bs}(D(x))}$ is in the interval $[1.48, 2.83]$ (resp. [1.44,2.89]). The relative error is small but must be considered relatively important in the case of D_{err}.

c) Case $\|A\|_r = MAX\_INT$

Fig. 13.9 presents the evaluation of the errors for $\|A\|_r = MAX\_INT$ and distributions *Rat* and *RatUni*. The size of errors is increasing with m. D_{err} increases from 170 (resp. 173) to 4161 (resp. 4162) in the case of *Rat* (resp. *RatUni*) distributions. N_{err} increases from 121 (resp. 111) to 1602 (resp. 1629) in the case of *Rat* (resp. *RatUni*) distributions. Dx_{err} increases from 109 (resp. 106) to 1595 (resp. 1621) in the case of *Rat* (resp. *RatUni*) distributions. As before, growth seems quadratic for D_{err} but in this case it is also better approximated $O(m\sqrt{m})$ for N_{err} and Dx_{err}. D_{err} is 1.5 to 2.5 times greater than N_{err} and Dx_{err}, the ratio is again increasing with m.

The relative errors seems relatively stable in all cases of D_{err}, N_{err} and Dx_{err}. The ratio $\frac{D_{err}}{\mathrm{bs}(D(\det(A)))}$ is in the interval $[1.46, 1.96]$ (resp. [1.42,1.79]) for the *Rat* (resp. *RatUni*) distribution. The ratio $\frac{N_{err}}{\mathrm{bs}(N(\det(A)))}$ and $\frac{Dx_{err}}{\mathrm{bs}(D(x))}$ is in the interval $[0.63, 1.13]$ (resp. [0.67,1.03]). This means that the errors are relatively insignificant in the case when $\|A\| = MAX\_INT$.

Results for random matrices indicate, that the quality of preconditioning should be good in the generic case of dense matrices. In general, approximating the denominator of $\det(A)$

is the challenging problem. The worst relative error that we have obtained in this case was less than 10% for all experiments considered. Therefore, we would like to classify the quality of denominator as :

1. applicable, if $D_{err} < \mathrm{bs}(D(\det(A)))$; this results with the preconditioned CRA scheme being superior to rational CRA;
2. good, if $D_{err} <\leq 0.1\,\mathrm{bs}(D(\det(A)))$; this corresponds to the 10% bound obtained for random rational matrices in the best case; the error of preconditioning can often be considered negligible in this case;

Additionally, notice that worst results have been obtained when matrix size m is correlated with the norm $\|A\|_r$ of the matrix.

Interestingly, the approximation of $N(\det(A))$ is often better than approximation of $D(\det(A))$. This partially answers the problem posed in Remark. 13.4.1. Namely, the Smith form of $\tilde{A}$ seems to have a non-trivial structure, which means that only a small factor of $\frac{D}{D(\det(A))}$ is included in N_{app}. This might however cause unexpected behavior if Alg. 8.4.1 is considered to compute the determinant of $\tilde{A}$.

13.4.2 Examples of Rational Matrices - Experimental Evaluation

We will consider examples of matrices from Sec. 13.2.2. We have completed the experiments for Hilbert matrices of size 50 to 1000, Lehmer matrices of size 50 to 400, and most of the matrices of BasisLib+ collection. Unfortunately, the computation was not finished for bigger matrix sizes in the case of Lehmer matrices and for another 10 biggest matrices of BasisLib+ collection. This includes matrices jendrec1,nemswrld,slptsk identified as difficult in Sec. 13.2.2. Consequently, BasisLib+ is represented by 147 matrices in this section.

1. Case of Hilbert matrices

   For Hilbert matrices, $N(\det(A)) = 1$ and a the formula for $D(\det(A))$ is known, yielding $\mathrm{bs}(D(\det(A))) = O(n^2)$, see e.g. [17]. Thus, we will not consider the relative error of $N(\det(A))$ approximation. We will also skip the analysis of Dx_{err}, as it depends mostly on denominators of c and not on matrix H_n.

   Fig. 13.10 presents the evaluation of errors for Hilbert matrices H_n as a function of matrix size n. The relative error $D_{err}/\,\mathrm{bs}(D(\det(H_n)))$ stabilizes around the value 8.2 with growing n. The size of error D_{err} grows like $O(n^2)$ and the size of error N_{err} like $O(n)$. Thus, the ratio of errors D_{err}/N_{err} is $O(n)$ and approximately $0.11n$ by our estimations. This leads to a very interesting conclusion on the structure of the Smith form of $\tilde{H}_n$. As $N_{err} = \tilde{s}_n(\tilde{A})$ in this case and $D_{err} = \mathrm{bs}(\det(\tilde{A}))$, this implies that $\tilde{H}_n$ has at least $0.11n$ non-trivial invariant factors.

2. Case of Lehmer matrices

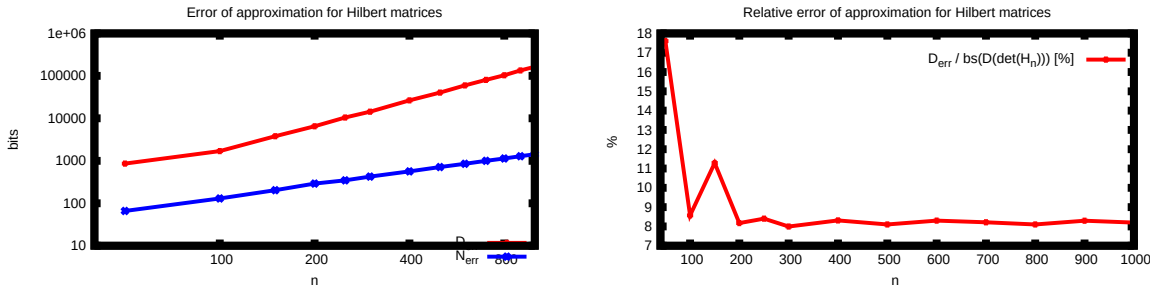

Figure 13.10: Left: bitsize of the error D_{err}, N_{err} for Hilbert matrices as a function of matrix size n; Right: bitsize of the error D_{err} relative to the size of $D(\det(H_n))$, scaling is logarithmic.

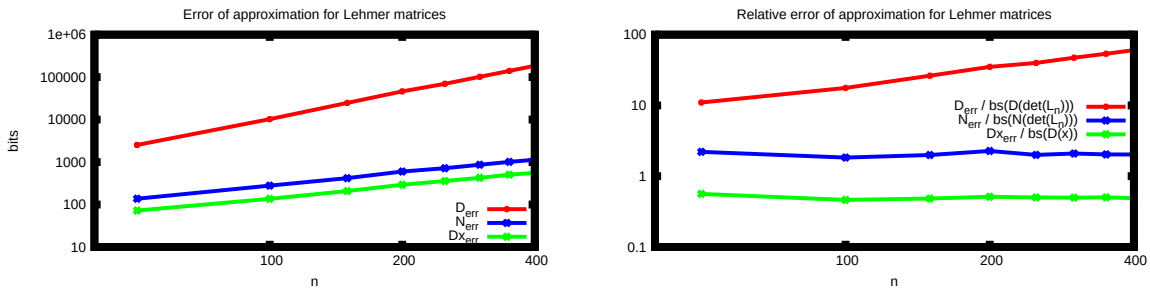

Figure 13.11: Left: bitsize of the error $D_{err}, N_{err}, Dx_{err}$ for Lehmer matrices as a function of matrix size n; Right: bitsize of the error relative to the size of $D(\det(L_n)), N(\det(L_n)), D(x)$.

In the case of Lehmer matrices, the formula for the inverse matrix L_n^{-1} can be given, and it is known to be a symmetric tridiagonal matrix, see e.g. [87, Ex. 7] and the references therein.

Fig. 13.11 presents the evaluation of the errors for Lehmer matrices L_n as a function of matrix size n. The relative error $D_{err}/\operatorname{bs}(D(\det(L_n))))$ grows linearly with n, and the ratio reaches high values from 11 to 60 in the tested range of n. The relative error $N_{err}/\operatorname{bs}(N(\det(L_n)))$ stabilizes around the value of 2, which is still over acceptable point. However, the relative error $Dx_{err}/\operatorname{bs}(D(x))$ stabilizes at an acceptable value of 0.5.

This is the only optimistic point in this experiment, as Lehmer matrices proves not only to be numerically unstable, but additionally do not have preconditioners of good quality for exact computation. Both Dx_{app} and N_{app} were multiples of the actual results.

As in the case of Hilbert matrices, the ratio $(\operatorname{bs}(N(\det(A))) + D_{err})/N_{err}$ is linear in n and equal to $0.27n$ by our estimations. This implies that $\tilde{L}_n$ has at least $0.27n$ non-trivial invariant factors.

3. Case of collection BasisLib+

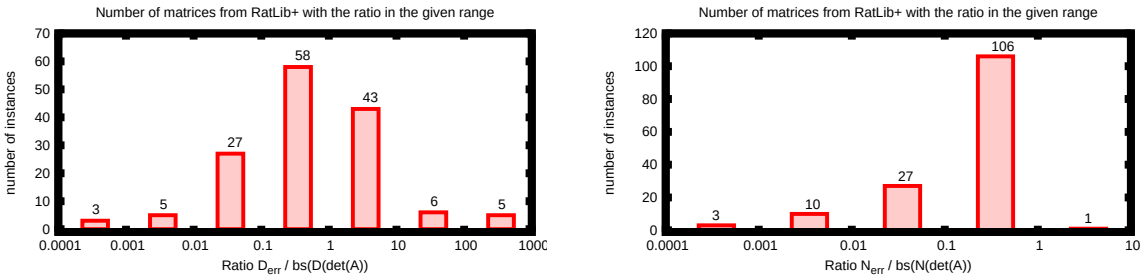

Figure 13.12: Number of instances of matrices from BasisLib+ Collection, for which the relative error of approximation was in the given range. Left: ratio $\frac{D_{err}}{\mathrm{bs}(D(\det(A)))}$. Right: ratio $\frac{N_{err}}{\mathrm{bs}(N(\det(A)))}$.

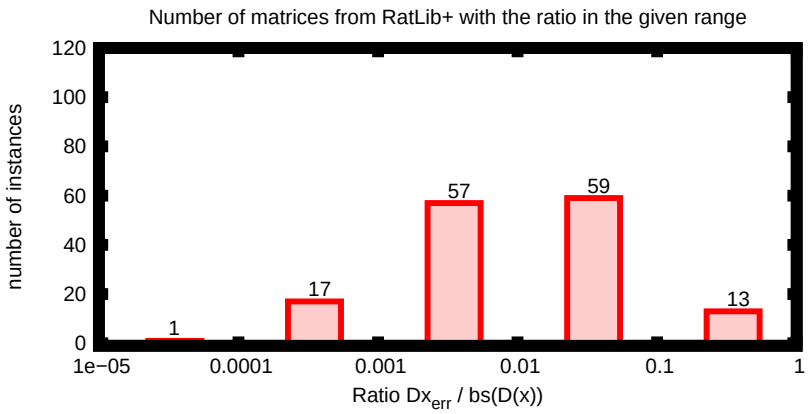

Figure 13.13: Number of instances of matrices from BasisLib+ Collection, for which the relative error $\frac{Dx_{err}}{\mathrm{bs}(D(x)}$ of was in the given range.

As can be seen in Fig. 13.12, the relative error of approximation $D_{err}/\,\mathrm{bs}(D(\det(A))))$ vary greatly on the matrices of BasisLib+ collection. The approximation is good (ratio < 0.1) for 24% of the collection and acceptable (ratio < 1) for 63%. For 29% the ratio was still less than 10, whereas for another 6 examples it was between 10 and 100, and for 5 cases greater than 100. This gives a total of 87% of matrices, for which preconditioning could induce higher number of iterations of preconditioned integer CRA compared to the rational variant. Correction of $\tilde{A}$ was necessary to obtain the results. In the design of rational algorithms, we will have to take measures to identify this cases and act accordingly.

Contrary to that, the quality of approximation of $N(\det(A))$ was acceptable (with the exception of one result of ratio 1.11) and good for 27 % of the collection. The quality of $D(x)$ approximation was very good, as Dx_{err} was less than 10% for all instances, see Fig. 13.13. In fact, the error D_{err} was at most 32 bits for 83% of matrices.

A	m	bs(N)	bs(D)	bs($D(x)$)	N_{err}	D_{err}	Dx_{err}
rd-rplusc-21	148	332	29	167	170 (168,3)	454	10 (4,7)
scrs8-2c-64	168	100	9	13	92 (91,2)	145	2 (2,1)
gen4	375	11281	10972	11201	4163 (137,4027)	192454	3971 (1,3971)
gen4.pre	367	11026	10736	10980	4234 (109,4126)	203266	4063 (1,4063)
route	339	180	1	78	104 (104,1)	20	7 (3,5)
msc98-ip	2897	1103	1	191	912 (912,1)	22	7 (1,7)
scrs8-2r-32	128	75	1	12	71 (70,2)	107	3 (3,1)
scrs8-2r-128	192	112	1	9	108 (107,2)	160	1 (1,1)
scrs8-2r-64	256	149	1	12	145 (144,2)	213	2 (2,1)
scrs8-2r-256	416	242	1	6	238 (237,2)	346	1 (1,1)
scrs8-2r-512	992	576	1	10	573 (572,2)	824	1 (1,1)

Table 13.2: Errors and relative errors for 11 most difficult examples of collection BasisLib+ based on the ratio $\frac{D_{err}}{\mathrm{bs}(D(\det(A)))}$. For brevity, N stands for $N(\det(A))$ and D for $D(\det(A))$. In the case of bs($\frac{p}{q}$), the sizes $\lceil \log(p) \rceil \lceil \log(q) \rceil$ are given in parentheses.

Let us now analyze the 11 worse cases of denominator approximation, for which the ratio $D_{err}/\,\mathrm{bs}(D(\det(A))))$ was greater than 10. Tab. 13.2 gives the errors and relative errors for matrices in this case.

For 7 out of 11 examples, big ratio is due to the fact that the determinant of the rational matrix is integer. This proves that this exceptional event can indeed be the case for rational problems that we might be given. The size of matrices varies from 128 to 2897 those examples. Surely, the ratio $D_{err}/\,\mathrm{bs}(D(\det(A))) = D_{err}$ is very large in this case.

Given the size of the numerator, the number of steps of the integer CRA compared to the rational variant is bigger by approximately 11% in the case of route, only 2% in the case of msc98-ip, but almost 2.5 times larger for scrs8-2r series of matrices. In the next 2 cases, the denominator is small, which leads to similar conclusions.

Notice, that we have also failed to approximate the numerator $N(\det(A))$ correctly, ending up with a bad underestimation. However, this has almost no influence to the case of approximating $D(x)$ in the case of system solving, as the approximation Dx_{app} was correct up to a few bits. Based on the data, we could also conclude, that the corresponding matrices $\tilde{A}$ have non-trivial Smith form, with over $0.2m$ non-trivial invariant factors in the case of scrs8-2r series. The number of invariant factors is estimated as $\frac{\mathrm{bs}(\det(\tilde{A}))}{N_{app}}$, which can be computed using data from Tab. 13.2 as $\mathrm{bs}(N(\det(A))) - N_{err}(1) + N_{err}(2)$, where $N_{err}(1)$ and $N_{err}(2))$ are given in parentheses next to N_{err}.

The two remaining cases of gen4 and gen4.pre are actually instances of the same computational problem. The results are substantially different in his case, as the size of numerator and denominator is large and comparable. The numerator is relatively well approximated by $\tilde{s}_n(\tilde{A})$, with the relative error of resp. 0.38 and 0.37, and is in fact mostly overestimated. The error Dx_{err} is comparable to N_{err}. Notice, that matrices gen4 and gen4.pre were identified in Tab. 13.1 as difficult when the growth in norm due to preconditioning is considered. Indeed, both matrices consist

of fractions with very different denominators. Also, we have estimated the number of non-trivial invariant factors of $\tilde{A}$ as over 13 an 14 respectively.

The emphasis of this section is being put on approximating the denominators for determinant, minimal/characteristic polynomial computation and system solving . Whereas the quality of approximation is generally satisfactory, for certain examples it is completely misguiding. Therefore in Ch. 14 our goal would be to construct an adaptive algorithm that will be able to identify such cases and act accordingly.

14

Adaptive Rational Algorithms

The algorithms for computation with rational matrices are based on the ideas described in Ch. 10 and 13. We may distinguish three main strategies.

1. First, rational matrix A is given and CR Algorithm 10.1.1 is run to reconstruct rational result $X(A)$ using early terminated strategy for the rational case. Also, any strategy which computes preconditioner D but cannot guarantee that $D \cdot X(A) \in \mathbb{Z}$ is classified here. This is the pure rational variant;

2. Second, rational matrix A is given and we define vector of preconditioners D such that $D \cdot X(A) \in \mathbb{Z}$. Alg. 10.1.1 is run with preconditioner D to reconstruct integer value $D \cdot X(A)$. No rational reconstruction is needed in In Alg. 10.1.1. This is the case of **result** preconditioning;

3. Finally, instead on A, computation is performed for preconditioned matrix $B = \mathrm{diag}(D_i)A$ or $B = A\,\mathrm{diag}(E_i)$ or $B = D(A)A$, see Sec. 13.1. Any integer algorithm can be run to compute $X(B)$, which in particular permits p-adic lifting, which does not carry on to the integer case. Then, $X(A)$ is derived from $X(B)$ by using Prop. 13.1.1, 13.1.2 etc. This is the case of **matrix** preconditioning;

Strategies might be run in parallel in a form of an adaptive algorithm. The solution is obtain as soon as one of the strategies terminates. The choice of available and applicable strategies depends on the problem. In the following sections we will discuss the case of system solving, determinant and characteristic/minimal polynomial computation.

14.1 Adaptive Rational Algorithms - Case Study

The algorithms that we are going to present are variants of Alg. 10.1.1. As such, they requires functions $image(A_p, A, p)$ and $iteration(y, A_p)$ to be defined, where *image* computes matrix $A_p = A \mod p$ over $\mathbb{Z}_p$, and *iteration* computes $X(A_p)$, see Sec. 10.2 and 10.3.

In Sec. 9.1, representations $\left[\frac{a_{ij}}{b_{ij}}\right], (D(A), A'), (\mathrm{diag}(D_i), \tilde{A}_1)$ or $(\mathrm{diag}(E_i), \tilde{A}_2)$ are proposed for rational matrix A. Recall that $D(A)$ is the common denominator of entries of

A and D_i and E_j are common denominators of entries in the ith row and the jth column respectively. Moreover $D = \gcd(\prod D_i, \prod E_i))$ is defined in Eq. (13.5).

For integer or rational matrix X, we will distinguish between two cases of $image(X)$ representation. We will set $image(X) = RatIm(Y)$ if X is represented as a rational matrix Y and $image(X) = IntIm(Z)$ if X is represented by an integer matrix Z. Through this chapter, we consider the representations of rational matrix A as $\left[\frac{a_{ij}}{b_{ij}}\right]$, $(D(A), A'), (\mathrm{diag}(D_i), \tilde{A}_1)$ or $(\mathrm{diag}(E_i), \tilde{A}_2)$. Reciprocally, matrix $\tilde{A}_1$ can be represented e.g. by $(\mathrm{diag}(D_i^{-1}), A)$ or $(\mathrm{diag}(\frac{D_i}{D(A)}), A')$ if the situation requires. Thus, rational matrix can be represented by its integer preconditioned variant, and reciprocally the integer matrix can be represented by its rational counterpart. Representations of $A', \tilde{A}_2$ can be considered accordingly.

Best imaging scheme can be determined by explicit image computation for random prime, or, by a simplified method, which will compute the image of one or several entries. We will assume that function $ChooseImage(X)$ returns the best representation. Unless $ChooseImage(X)$ is explicitly called, the default integer/rational representation is used;

Integer reconstruction can generally be defined incrementally by Eq. (10.4.1), although delayed reconstruction, see Sec. 10.4.2 or adaptive strategy of Sec. 10.7 might be considered as well. We use a boolean function $UseRatrec$ to denote the scheduling of rational reconstruction in purely rational variant. Scheduling can be incremental, quadratic, geometric or adaptive, see Sec. 10.5 and 10.7.

Finally, two termination strategies should be available, namely $IntET$ and $RatET$ which correspond to early termination condition in the cases of rational and integer reconstruction. $IntET$ and $RatET$ ensure that the result is correct with a probability at least $1 - \epsilon$, where $\epsilon > 0$ is given on input, whenever one of strategies returns *true*. A size P of primes and function *gen_prime* should fulfill the requirements of Lem. 10.6.1, 10.6.3 and Cor. 10.6.6. In the case of the reconstruction of a vector, see Sec. 10.6.3.

14.1.1 System Solving

In the case of system solving, we will provide an additional argument c which is a rational vector of size m given at the right-hand side of the equation $Ax = c$. In addition to notations introduced earlier in the section, let us recall the notations of Prop. 13.1.1. Recall that $D(c)$ is the common denominator entries of c and $D_i(c)$, the denominator of the ith entry. Additionally, we set $D' = \mathrm{diag}(D_i(c))$.

Only purely rational variant or matrix preconditioning can be used, as no multiple of denominator for resulting vector $A^{-1}c$ is known beforehand. Heuristic preconditioner defined in Eq. (13.10) can be used to enhance early termination. Matrix preconditioning variant is based on Prop. 13.1.1. The bounds D and N for the denominators and numerators of $A^{-1}x$ are given in Prop. 13.3.11. The bounds and preconditioners depend on c.

If one entry of vector x is found, it might be used as preconditioner for other entries, which should reduce the time of vector reconstruction. This strategy is often adapted in linear

algebra softwares, see e.g. [16]and [118, Sec. 3.2.3] for description. As remarked in Sec. 13.3, early termination seems necessary in order to effectively solve a rational system of equations.

In the case of integer system of equations, rational CRA was historically the first algorithm, see [12]. The algorithm carries on to the rational case without a change. By using an early termination strategy of Wang [129, 130] or Monagan [90], one can obtain the output-dependent complexity $O^{\sim}(K)$ i.e. linear in K, where K is the number of iterations needed to obtain the residue which is sufficiently large to reconstruct the result and early terminate. Let $A^{-1}c$ be the solution of the system and $D(A^{-1}c)$ and $N(A^{-1}c)$ its maximal denominator and numerator respectively. Then K is equal to

a) $K = O(\max(\log(D(A^{-1}c)), \log(N(A^{-1}c))))$ for the Wang's strategy,

b) $K = O(\log(D(A^{-1}c)) + \log(N(A^{-1}c)))$ for the Monagan's strategy.

See Sec. 11.6.2 for the experimental comparison of Wang's and Monagan's early termination strategies. The CR algorithm can be applied to solving the original system as well as the preconditioned integer systems.

Other algorithms for integer system solving, based on p-adic lifting were described in Sec. 6.3.5. See also [27, 122, 43]. The complexity of an early terminated strategy for Dixon p-adic lifting is $O\left(m^{\omega}\right) + O\left(m^2 K \log(m\|A\|)\right) + O^{\sim}(K)$. K is the number of steps and is the same as in the case of rational CRA.

The terms stand for the cost of inverse mod p computation by LU (applied to a dense $m \times m$ matrix), the cost of K lifting steps (which is dominated by integer matrix-vector product Av, where $\|v\| \leq p$), and the cost of rational reconstruction. The latter can be made $O^{\sim}(K)$ if fast rational reconstruction of [131] (see Alg. 11.4.3,11.5.1) is applied, and rational reconstruction is well scheduled, see Sec. 10.5. Ideas of Sec. 10.7 apply as well. Using the classic Euclidean algorithm and geometric scheduling, the cost becomes $O(K^2)$. See also [117, Thm. 11] for more details. p-adic based algorithms can be applied only to integer systems.

In general, scheduling is necessary in order to obtain good complexity and good running times of rational CRA and p-adic schemes.

Remark 14.1.1 (Adaptive Algorithm of Wan [128]) In his work [127, 128], Wan has proposed a different approach for solving well conditioned integer systems, by applying a combination of numerical and symbolic methods. This approach can be carried on to the rational case, by matrix preconditioning, see [118, Alg.2],[117, Alg. 4]. It is not our intention to go into details of this algorithm, we prefer to focus on rational methods that can be applied in the case of ill-condition matrices. We would like, however, to emphasize some similarities between the algorithm of [128] and p-adic lifting, that are essential from our point of view.

The algorithm of [128] starts with a numeric computation of the inverse of a matrix. Then, K' iterations are performed, which consist of matrix-vector a product Av, where v has word

size entries (cost $O(m^2 \log(m\|A\|))$). Additionally, rational reconstruction is performed at the end. K' is determined in the adaptive way, and algorithm runs until required precision is obtained. The precision is related to the maximal size of the denominator and thus to the bounds of Prop. 13.3.11. Experiments of [118, 128] suggest, that the method is generally faster but comparable to early terminated p-adic lifting at the same time. In [128, Sec. 3.1], Wan points out, that this is due to the fact, that more bits of the solutions are recovered by numerical solver for well conditioned matrix, than for a p-adic solver which uses BLAS routines.

Thus, although we do not explicitly consider the solver of [128, 117, 118] in the rest of this chapter, we feel free to remark, that our discussion can be carried on to the case. In particular, the running time of both Wan's algorithm and p-adic lifting is proportional to the cost of one matrix-vector product. Consequently, conditions that we design later on in Alg. 14.1.1 and 14.1.4 can be adjusted to the of Wan's algorithm by eventually normalizing the time of one iteration by the number of bits recovered.

In [128], the author remarked that the algorithms does not have to be completed in order to obtain the result with certain precision. Bearing this in mind, it seems plausible, that a scheduled rational reconstruction can be incorporated in the main loop of the algorithm, which could lead to early termination. In this case, preconditioning by Dx_{app} could be applicable, as intuitively, it reduces the precision to which the denominator has to be approximated. The probability of success of early termination and the performance of Wan's algorithm in this case remains to be tested. The number of steps of such approach should be asymptotically equal to K, for the case of Wang's and Monagan's termination strategies.

Choice of Solver

Denote by p the upper bound on primes used in CRA and p-adic lifting. Assume that in both cases, primes of the same size can be used.

By Prop. 13.1.1, the following systems can be solved:

- $Ax = c$; by rational CRA,
- $A'x' = D(c)c$; by rational CRA or p-adic solving,
- $\widetilde{(D'A)}_1 x = D_1 D'c$, where $\widetilde{(D'A)}_1 = D_1 D'A$; by rational CRA or p-adic solving,
- $\widetilde{(D'A)}_2 y = D'c$, where $\widetilde{(D'A)}_2 = D'AD_2$, $x = D_2 y$; by rational CRA or p-adic solving;

In the case of CRA and p-adic solving the same scheduling of rational reconstruction can be envisaged. In the case of computation of x, the same number of iterations (i.e. LU modular routines or lifting steps) is expected. Consequently, the performance of rational CRA and p-adic solving depends on the cost of one iteration. As the bitsize of y might be bigger or smaller than the bitsize of x, the number of iterations for the last solver can

be different. In the most general case, $\mathrm{bs}(y) = \mathrm{bs}(x) \pm \log(\|D_2\|)$, see Eq. 13.17 for the definition of bitsize.

For integer matrix X, the complexity of one p-adic lifting step is $O(m^2 \log(m\|X\|))$ bit operations. For one iteration of CRA, the complexity is $O(m^2 \log(\|X\|) + m^\omega)$. Thus, p-adic lifting is always better in the asymptotic case. For a particular matrix instance, this can also be confirmed experimentally, by explicitly comparing the times of computing one matrix-vector product Xv (where v is a random integer vector and $\|v\| \leq p$) and computing one CRA iteration.

The choice between integer matrices $A', \widetilde{(D'A)}_1$ and $\widetilde{(D'A)}_2$ can easily be made, as the time of one iteration increases with matrix norm. Thus, we may consider only the matrix with the smallest norm. Notice, that we might require to modify this approach if matrix A' is structured and consequently easier to manipulate. We will further assume that integer matrix $\overline{A}$ is chosen based on the norms and structure comparison. Thus, we are left with the choice between rational CRA for rational matrix A and p-adic lifting for $\overline{A}$.

The norms of A and $\overline{A}$ can differ greatly for particular matrix instances, see Sec. 13.2. A bunch of algorithms, including hybrid solutions can be used in the case rational and integer arithmetics, see e.g. [125]. Even more algorithm can be used in the case of CRA and p-adic iterations, which includes running sparse procedures in both or just one case. This makes the asymptotic complexity analysis quite complicated. Our algorithm can made a choice depending on the actual implementation, by explicit comparing the timings and choosing the fastest solution. Independently of the choice, for the asymptotic analysis we may assume, that the algorithm with best asymptotic complexity was chosen.

We propose that a procedure $ChooseSolver(A, \overline{A})$, see Alg. 14.1.1, is run in order to determine the faster solver. More generally $ChooseSolver(A, A', \tilde{A}_1, \tilde{A}_2)$ can be considered, which additionally determines $\overline{A}$ by evaluation of norms.

Algorithm 14.1.1 ChooseSolver Procedure

Require: : $A \in Q^{m\times m}, \overline{A} \in \mathbb{Z}^{m\times m}$, p prime;
Ensure: : Matrix $\widehat{A}$ and method $M \in \{\text{CRA}, p\text{-adic}\}$, whichever is faster;
1: Run $ChooseImage(A)$ and find the best representation for A;
2: Compute $A \mod p$; Compute $A^{-1} \bmod p$; Store timing in T_A;
3: Generate a random vector $v \in \mathbb{Z}_p$;
4: Compute $\overline{A}v \in \mathbb{Z}$; Store timing in $T_{\overline{A}}$;
5: **if** $T_A > T_{\overline{A}}$ **then** Return ($\overline{A}$, p-adic);
6: **else** Return (A, CRA);

In the case when rational CRA have been chosen, $A^{-1} \mod p$ computed by $ChooseSolver$ can be used to calculate the first residue. In the case when p-adic lifting has been chosen, $\overline{A}^{-1} \mod p$ can be determined by multiplying $A^{-1} \mod p$ by diagonal matrices mod p. Thus, $ChooseSolver$ can be computed at the initialization phase at a small additional cost of one random $\overline{A}v$ product computation. Surely, the computation of $\overline{A}v$ can be interrupted if the time limit T_A is reached.

CRA and p-adic lifting are methods to obtain the solution modulo a large modulus M. Once this is done, rational reconstruction might be used to recover $x = D_2 y$ or $y = D_2^{-1} x$. Thus, we terminate in a number of steps proportional to $\min(\mathrm{bs}(x), \mathrm{bs}(y))$ by scheduling rational reconstruction on x and y at the same time.

Preconditioning

The approximation Dx_{app} of denominator $D(x)$ can be found by solving an integer system of equations $\tilde{A}_1 z = b$, see Eq. (13.10). The results of experiments presented in Sec. 13.2.2 suggest, that the norm of $\|\tilde{A}_1\|$ can be considerably smaller than the norm of $\|\widetilde{(D'A)}_1\|$ or $\|A'\|$, see Fig. 13.5 for a particular choice of vector b. Experiments in Sec. 13.4 suggest that the quality of approximation $Dx_{app} = \frac{D(c)}{\gcd(D(c),D(A)}D(z)$, see Eq. (13.9) is very good, and $Dx_{app} \cdot x$ is often a vector of fractions with small denominators.

The approximation Dx_{app} is provided by Eq. (13.9) after one solving has been done for A by any method. If this is not the case, we might choose to solve a random integer system of equations $\tilde{A}_1 z = b$ first, before solving Eq. $Ax = c$ with preconditioning by a method determined in $ChooseSolver$.

The part $\frac{D(c)}{\gcd(D(c),D(A)}$ of preconditioner can be computed at low cost. Computation of the remaining part $D(z)$ is costly, as it requires another system solving. Let us now evaluate the case, in which it is worth trying. We will heuristically assume that the size of numerator $N(z)$ is approximately the same as the size of denominator $D(z)$ in the case of random system of equations $\tilde{A}_1 z = b$. Secondly, we will assume that $D(z)$ approximates a factor of $D(x)$, i.e. $\frac{D(x)}{D(z)}$ has a small denominator. This is fulfilled in particular if the approximation is good.

Let $T_{\tilde{A}}$ be the time of computing $\tilde{A}v$ for a vector $v, \|v\| \leq p$. Then, the computation od $D(z)$ takes at least $2\alpha \log(D(z))T_{\tilde{A}}$ by p-adic lifting, where $\alpha \approx \frac{1}{\log(p)}$ depends on the prime p used. We can ignore the time of inverse computation, as the inverse can later be used while solving $Ax = c$. We can ignore the time of rational reconstruction, as it is the same for the same number of iterations, performed while solving $Ax = c$. We may assume that computing z takes at most is $4\alpha \log(D(z))T_{\tilde{A}}$, if geometric scheduling is used for reconstruction. Therefore, we can compute $D(z)$ in about $4\alpha \log(D(z))T_{\tilde{A}}$.

Let $T_{\widehat{A}}$ be the time of one iteration of solving $Ax = b$ by the method determined in $ChooseSolver$. Computing $D(z)$ bits of solution x would take $\alpha \log(D(z))T_{\widehat{A}}$ time. Assuming that $D(z)$ 'almost' divides $D(x)$, this means that we have gained in time as soon as

$$4\alpha \log(D(z))T_{\tilde{A}} < \alpha \log(D(z))T_{\widehat{A}} \Leftrightarrow T_{\tilde{A}} \leq \frac{T_{\widehat{A}}}{4}. \tag{14.1}$$

The bound for the ratio $\frac{T_{\tilde{A}}}{T_{\widehat{A}}}$ can be adjusted to another value between 0.25 and 0.5, if we assume that early termination occurs faster for random vector b and random prime p in geometric scheduling. Procedure $ChoosePrec(\widehat{A}, \tilde{A})$ evaluates Eq. (14.1) and returns true if preconditioner should be computed.

Algorithm

The adaptive rational algorithm for system solving is defined in Alg. 14.1.2. We keep the notions introduced earlier in this section.

Algorithm 14.1.2 System solving

Require: $m \times m$ matrix A.
Require: c - a $m \times m$ rational vector,
Require: $ChooseImage$ to choose best representation for imaging,
Require: $ChooseSolver$ to choose best solver, see Alg. 14.1.1
Require: $ChoosePrec$ to determine if preconditioner should be computed,
Ensure: x - solution to $Ax = b$.

```
Compute D' = diag D_i(c), A', (D'A)_1, (D'A)_2;
Compute D_2, D(A), D(c);                                   # See Prop. 13.1.1
(Â, M) = ChooseSolver(A, A', (D'A)_1, (D'A)_2);
Dx_app = D(c) / gcd(D(A), D(c));
if ChoosePrec(Â, Ã) then
    Generate random vector b ∈ Z^m;
    Solve Ãz = b;
    Dx_app = Dx_app · D(z);
if M = p-adic then
    Run p-adic solver for Â, early terminate for Dx_app x or x or y = D_2^{-1} x or Dx_app y;
else
    ChooseImage(Â);
    Run rational CRA for Â, early terminate for Dx_app x or x or y = D_2^{-1} x or Dx_app y;
Return x;
```

14.1.2 Determinant

All three strategies can be used in the case of determinant computation. For result preconditioning strategy, D, given by Eq. (13.5) provides a multiple of the denominator $D(\det(A))$, see Prop. 13.3.1. In the case of matrix preconditioning, see Prop. 13.1.1, we can use any of matrices $A', \tilde{A}_1, \tilde{A}_2$ and recover $\det(A)$ by Eq. (13.1). Notice, that $\det(A')$ is the biggest of all integer matrices, therefore A' can only be envisaged as representation for imaging in the case of structured matrices.

The evaluation of quality of preconditioning in Sec. 13.4 leads to the conclusion, that the error of approximation often do not exceed the actual denominator i.e. $\mathrm{bs}(\frac{D}{D(\det(A))}) < \mathrm{bs}(D(\det(A)))$. This is definitely true for the generic case of random matrices, see Fig. 13.6,13.7,13.8, 13.9. This has also been the case for most of other interesting examples we have encountered, see 13.10,13.12. However, preconditioning might lead to overestimations, see Fig. 13.11 and Tab. 13.2. In this case, purely rational variant could provide solution quicker than preconditioned algorithms.

Correction of $\tilde{A}$

Matrices $\tilde{A}_1, \tilde{A}_2$ can be corrected by finding column and row gcds, as remarked in Sec. 13.4.1. This leads to the procedure $Correction(\tilde{A})$ given in Alg. 14.1.3. Alg. 14.1.3 changes matrix $\tilde{A}_{in}$ *in situ* and returns corrected matrix $\tilde{A}_{out}$ and integer $d \in \mathbb{N}$ such that:

- $\det(\tilde{A})_{out} = \frac{\det(\tilde{A})_{in}}{d}$;
- the gcds of every column and row of $\tilde{A}$ is 1;

Algorithm 14.1.3 Correction Procedure

Require: : $\tilde{A} \in \mathbb{Z}^{m \times m}$;
Ensure: : Matrix $\tilde{A}$, s.t. gcd of every row and column is 1;
Ensure: : d, s.t. $\det(\tilde{A})_{out} = \frac{\det(\tilde{A})_{in}}{d}$;

```
modified = true; d = 1;
while modified do
    modified = false;
    for j = 1 to m do
        g = gcd(Ã[1..m][j]);
        if g > 1 then
            Ã[1..m][j] = Ã[1..m][j]/g;
            d = d · g; modified = true;
    end for
    for i = 1 to m do
        g = gcd(Ã[i][1..m]);
        if g > 1 then
            Ã[i][1..m] = Ã[i][1..m]/g;
            d = d · g; modified = true;
    end for
end while
```

Proposition 14.1.2 (Correction of $\tilde{A}$) Let A be a $m \times m$ rational matrix and let $\tilde{A}_1 = \operatorname{diag}(D_i)A$ and $\tilde{A}_2 = A \operatorname{diag}(E_i)$, where D_i (resp. E_i) are common row (resp. column) denominators of A. Define $D = \gcd(\prod D_i, \prod E_i)$, see Eq. (13.5). Suppose that $\tilde{A}_1, \tilde{A}_2$ are modified by Alg. 14.1.3 and let d_1, d_2 be the values outputted by Alg. 14.1.3 respectively for $\tilde{A}_1, \tilde{A}_2$. Define

$$\begin{aligned} D' &= \gcd\left(\frac{\prod D_i}{\gcd(\prod D_i, d_1)}, \frac{\prod E_i}{\gcd(\prod E_i, d_2)}\right) \\ N' &= \operatorname{lcm}\left(\frac{d_1}{\gcd(\prod D_i, d_1)}, \frac{d_2}{\gcd(\prod E_i, d_2)}\right). \end{aligned} \tag{14.2}$$

Then $D(\det(A)) \mid D' \mid D$, and $N' \mid N(\det(A))$.

PROOF Without loss of generality, it suffices to show that $D(\det(A)) \mid \frac{\prod D_i}{\gcd(\prod D_i, d_1)}$ and $\frac{d_1}{\gcd(\prod D_i, d_1)} \mid N(\det(A))$ for $\tilde{A}_1, d_1$.

Let $\tilde{A}_{1,in}$ denote matrix $\tilde{A}_1$ at the input to Alg. 14.1.3 and $\tilde{A}_{1,out}$, the matrix at the output. Then $\det(\tilde{A}_{1,in}) = d \det(\tilde{A}_{1,out})$, by m-linearity of the determinant. By Prop. 13.1.1, we have

$$\det(A) = \frac{\det(\tilde{A}_{1,in})}{\prod D_i} = \frac{\det(\tilde{A}_{1,out}) d_1}{\prod D_i}.$$

Let $g_1 = \gcd(d_1, \prod D_i)$. Then

$$\frac{\det(\tilde{A}_{1,out}) d_1}{\prod D_i} = \frac{\det(\tilde{A}_{1,out}) \frac{d_1}{g_1}}{\frac{\prod D_i}{g_1}} = \frac{\alpha x \frac{d_1}{g_1}}{\alpha D(\det(A))},$$

for certain $x \in \mathbb{Z}$. Thus, $D(\det(A))$ divides $\frac{\prod D_i}{g_1}$ and $\frac{d_1}{g_1}$ divides $N(\det(A))$. The case of $\tilde{A}_2, d_2$ is analogous. Hence, D', N' can be defined as the gcd and lcm respectively.

Choice of Integer Algorithm

Preconditioned rational matrices $\tilde{A}_1, \tilde{A}_2$ provide good benchmark cases for testing Alg. 8.4.1. As remarked in conclusions to Sec. 13.4.1 and in Sec. 13.4.2, the Smith form of preconditioned matrix $\tilde{A}$ could be non-trivial. Moreover, $\log(\|\tilde{A}\|)$ can be large, see 13.2, which means that ideas of Sec. 8.9 gain on importance, with a special emphasis to *a priori* estimation of $s_m(\tilde{A}_i), i = 1, 2$ and early termination for p-adic lifting in Alg. 6.3.1.

Therefore, in Alg. 14.1.5 we propose to use the evaluation similar as in *ChooseSolver* (see Alg. 14.1.1) and *ChoosePrec*, in order to decide, whether invariant factors should be computed. Recall that invariant factors are computed by solving equation $\tilde{A}x = b$ for random vector b, see Alg. 6.3.1, Alg. 7.3.1. For integer matrix $\tilde{A}$, p-adic solving is asymptotically preferred. Then, $s_m(\tilde{A})$ is approximated by $D(x)$. We will assume for the analysis that $D(x)$ is close to $s_m(\tilde{A})$.

Analogously as in the case of *ChoosePrec* computation, recovering $\log(s_m(\tilde{A}))$ bits of $\det(\tilde{A})$ by CRA requires $\alpha \log(s_m(\tilde{A}))$ steps, where $\alpha \approx \frac{1}{\log(p)}$ depends on the bound p on primes used in CRA. In order to solve the system, we have to compute $N(x)$, and terminate a scheduled rational reconstruction. Assuming that $\log(N(x)) \approx \log(D(x))$, and that geometric scheduling is used for rational reconstruction, this means that up to 4 times more iterations of p-adic lifting are necessary in order to compute $s_m(\tilde{A})$. We ignored the time of rational reconstruction, which could lead to the even worst ratio in terms of time. Thus, we propose procedure *ChooseLIF* in Alg. 14.1.4 which returns true, if it evaluates that the computation of $s_m(\tilde{A})$ by p-adic lifting is cheaper.

The result $\det(\tilde{A}) \mod p$ can be added to the CR loop of Alg. 8.4.1. Thus, the only additional cost of *ChooseLIF* procedure is that of random $\tilde{A}v$ product computation. Surely, the computation of $\overline{A}v$ can be interrupted if the time limit $4T_{CRA}$ is reached.

Algorithm 14.1.4 ChooseLIF Procedure

Require: : $\tilde{A} \in \mathbb{Z}^{m\times m}$, p prime;
Ensure: : **true**, if $s_m(\tilde{A})$ should be computed; **false** otherwise;
1: Run $ChooseImage(\tilde{A})$ and find the best representation for $\tilde{A}$;
2: Compute $\tilde{A} \mod p$; Compute $\det(\tilde{A}) \bmod p$; Store timing in T_{CRA};
3: Generate a random vector $v \in \mathbb{Z}_p$;
4: Compute $\tilde{A}v \in \mathbb{Z}$; Store timing in T_{LIF};
5: **if** $4T_{CRA} > T_{LIF}$ **then** Return **true**;
6: **else** Return **false**;

Algorithm

The adaptive rational algorithm for determinant computation is defined in Alg. 14.1.5. We assume that A is a $m \times m$ matrix.

Define matrices $\tilde{A}_1, \tilde{A}_2$ as output to procedure $Correction$ on $\mathrm{diag}(D_i)A$, $A\,\mathrm{diag}(E_i)$ resp., c.f. Alg. 14.1.3. Let d_1, d_2 be the values returned by the algorithm in this case and define $D'_1 = \frac{\prod D_i}{\gcd(\prod D_i, d_1)}$ and $D'_2 = \frac{\prod E_i}{\gcd(\prod E_i, d_2)}$, $N'_1 = \frac{d_1}{\gcd(\prod D_i, d_1)}, N'_2 = \frac{d_2}{\gcd(\prod E_i, d_2)}$, see Prop. 14.1.2. By Prop. 14.1.2, we have that $\frac{D'}{N'}\det(A) \in \mathbb{Z}$, where $D' = \gcd(D'_1, D'_2)$ and $N' = \mathrm{lcm}(N'_1, N'_2)$.

The following determinant computations can equivalently be performed:

- $\det(A)$; by rational CRA,
- $\det(A)$; by preconditioned integer CRA with multiplicative preconditioner $\frac{D'}{N'}$, where D', N' are defined in Eq. (14.2) (or, $D' = D, N' = 1$ if $Correction$ is not run).
- $\det(\tilde{A}_i)$, $i = 1, 2$; by integer determinant algorithm; $\det(A) = \frac{N'_i}{D'_i}\det(\tilde{A}_i)$ is returned;

See Sec. 4.1 and 8.10.1 for references on integer determinant algorithms. See also [2, 122, 44, 79] among others. We will consider the use of out adaptive algorithm Alg. 8.4.1 (see also Alg. 8.3.1, 8.3.2) for $\tilde{A} = \tilde{A}_1, \tilde{A}_2$. Alg .8.4.1 combines the ideas of algorithms [2, 122, 44, 79], see Part. II.

Alg. 14.1.5 intertwines the above-mentioned variants of computation. We refer by RatDet - to the part which corresponds to running rational CRA. By PrecDet we refer to running preconditioned integer CRA for A. In the case when integer algorithm is run for $\tilde{A}_i$, we will refer to the case when $s_m(\tilde{A})$ is computed as PrecBonus. In the opposite case, integer CRA could be run on matrix $\tilde{A}_i$. However, it is known beforehand, that preconditioned integer CRA is looking for smaller value, thus possibly terminating in smaller number of iterations. The choice of optimal imaging scheme for A (c.f procedure $ChooseImage$) could lead to the actual computation being performed on $\tilde{A}_i$ anyway.

Alg. 14.1.5 is a variant of CRA, with dual integer or rational (case RatDet) termination condition, with or without preconditioning. Variable preconditioner is used in the case of PrecBonus strategy.

Algorithm 14.1.5 Determinant

Ensure: x - the determinant of A.

Compute $\mathrm{diag}(D_i)$, $\mathrm{diag}(E_i)$; $P_1 = \det(\mathrm{diag}(D_i)), P_2 = \det(\mathrm{diag}(E_i))$;
$\tilde{A}_1 = \mathrm{diag}(D_i)A$, $\tilde{A}_2 = A\,\mathrm{diag}(E_i)$;
for $i = 1, 2$ **do**
  $d_i = Correction(\tilde{A}_i)$; Compute $\|\tilde{A}_i\|$;
  Compute $D'_i = \frac{P_i}{\gcd(P_i,d_i)}$, $N'_i = \frac{d_i}{\gcd(P_i,d_i)}$;
end for
if $\|\tilde{A}_1\| < \|\tilde{A}_2\|$ **then** $i = 1$ **else** $i = 2$;
$D = \gcd(D'_1, D'_2)$;
$ChooseImage(A)$;
$\pi = 1$, $k = m$;
repeat
  if $ChooseLIF(\tilde{A}_i)$ **then**
{PrecBonus}
    Run Alg. 8.3.2 to compute $\tilde{s}_k \approx s_k(\tilde{A}_i)$; $k = k - 1$;
    $N'_i = \tilde{s}_k \cdot N'_i$;
  $N = \mathrm{lcm}(N'_1, N'_2)$;
{PrecDet}
  Run some steps of Alg. 10.1.1 on A with preconditioner $\frac{D}{N}$ to compute $x = \det(A)$;
{RatDet}
  if $UseRatrec$ **then** Try to reconstruct x by rational reconstruction;
  if $IntET$ or $RatET$ **then** **Return** x;
until $IntET$ or $RatET$

The idea of using $\pi \approx \prod s_k(\tilde{A}_i)$ in the determinant computation is explained in Ch. 8, see Sec. 8.5. π is approximated in the way, that ensures that $\pi \mid \det(\tilde{A}_i)$. Thus, it ensures that $\frac{D}{N}\det(A) \in \mathbb{Z}$, where D, N are defined in the algorithm, see Prop. 14.1.2.

Technical details presented in Ch. 8 include determining the number of CRA steps that should be performed at each iteration of the **while** loop and adding a worst-case switch to another integer algorithm of better asymptotic complexity. See Alg. 8.4.1 for details. For sake of simplicity, we omit these details in Alg. 14.1.5.

In Sec. 14.2 we analyze the complexities of strategic choices RatDet, PrecDet and PrecBonus in detail, depending on matrix type. We analyze two main cases: matrices where the common denominator of all entries, the common denominator of the rows (columns) and the norm of A are of the same order i.e. $D(A) = O(D_i) = O(\|A\|)$ and matrices with entries given as fractions with different denominators. We will also evaluate the strategic choices RatDet, PrecDet and PrecBonus experimentally in Sec. 14.3.

Alg. 14.1.5 can also be run for integer matrix A. In this case, at the input of the algorithm, $A = \tilde{A}_1 = \tilde{A}_2$ and Alg. 14.1.5 presents a variant of Alg. 8.4.1, for which CRA is run in the worst case. Alg. 14.1.5 improves Alg. 8.4.1 by introducing the following modifications.

- *Correction*(A) is performed. It has been conjectured in Sec. 8.10.1, that such procedure might improve the expected number of non-trivial invariant factors of a sparse matrix; experiments in Sec. 14.3 have confirmed better running times.

- It introduces early terminated p-adic lifting to the algorithm. By checking for termination of RatDet, the algorithm is robust to errors of early termination of p-adic lifting, but runs with the worst case complexity in this case;

- Combined with early terminated p-adic lifting, it allows for *a prior* evaluation of $s_m(A)$, which is postulated in Sec. 8.9;

14.1.3 Minimal and Characteristic Polynomial

Let $p(A) = \sum_{i=0}^{d} c_i x^i$ be the characteristic/minimal polynomial of A that we are going to compute. All three strategies are available. Yet, only A' can be used as preconditioned matrix, see Prop. 13.1.2. Then the coefficients c'_i of $p(A')$ are equal to $D(A)^{d-i}c_i$. Only CR Algorithm is available in the integer case, see [109].

Theorem 13.3.2 gives a vector preconditioner defined by Eq. (13.6) as $\mathbf{D} = [D_{app}(i)]$, where

$$D_{app}(i) = \gcd(D(A)^{d-i}, D) \quad \text{for} \quad i = 0, \ldots, d, \tag{14.3}$$

and D is given in Eq. (13.5). It is easy to see, that D can be replaced by D', defined in Eq. (14.2). The coefficients of $\mathbf{D} \cdot [c_i]_{i=1..d}$ that we are going to reconstruct are never larger that the coefficients of $p(A')$, see Prop. 13.1.1. This implies that matrix preconditioning is impractical in the case of minimal/characteristic polynomial computation and A' can only by used for imagining purposes. Both matrices $\tilde{A}_1, \tilde{A}_2$ can be used for imaging as well, but the original matrix $A \mod p$ has to be reconstructed from $\tilde{A}_i \mod p$.

The algorithm for computation of minimal and characteristic polynomial is equivalent to Alg. 10.1.1 for given set of vector preconditioners $\mathcal{D} = \{\mathbf{D}, [1]_{i=0..d}\}$, using integer/rational termination strategy respectively. We expect that the quality of preconditioner $\mathbf{D}$ will be good for most of the entries, see Sec. 13.4, for the results on quality of $D_{app}(0) = D$ approximation. We will validate this experimentally in Sec. 14.3.3. Preconditioner $[1]_{i=0..d}$ will work well in the very unlikely opposite case, when all coefficients of $p(A)$ are integer.

Notice, that minimal polynomial computation is an example of vector CRA, where the entries of the reconstructed vector $[c_i]$ are possibly different in bitsize. Preconditioners D_i scale accordingly. This means that reconstruction the entries of the vector one by can be better than reconstructing a random linear combination $r \cdot [c_i]$, see Sec. 10.6.3.

If we start by reconstructing c_{m-1} (the trace), which is presumably the smallest and would be reconstructed first, its denominator $D(c_{m-1})$ can be used to approximate $D(c_{m-2})$. Inductively, $D(c_{i-1})$ can be used to approximate $D(c_i)$. This is based on the heuristic assumption, that the denominators of $p(A)$ are correlated, even if the preconditioner $\mathbf{D}$ is erroneous. Notice, that once $c_{m-1}, \ldots, c_{i-1}$ are reconstructed, the quality of preconditioning can be quite well evaluated. In this case, three separate rational reconstruction can

be carried out, for $D_i c_i, D(c_{i-1})c_i$ and c_i. We will evaluate this approach experimentally in Sec. 14.3.3.

Reciprocally, we may start the reconstruction by the biggest entry, which is c_0 (the determinant). In this case, we may heuristically assume, that once c_0 is reconstructed, the modulus is large enough for the rest of coefficients as well.

14.1.4 Correctness of Rational Algorithms

Proposition 14.1.3 (Correctness of Rational Algorithms) Suppose that terminations strategies $IntET$ and $RatET$ are defined for the Chinese Remaindering Algorithm and p-adic lifting, which ensure $1-\epsilon$ probability of correctness of the result. Then, Alg. 14.1.2, Alg. 14.1.5 and the introspective minimal/characteristic polynomial algorithm in 14.1.3 terminate and return the result, which is correct with probability $1-\epsilon$.

PROOF Termination of CRA is defined independently of preconditioning, based on reoccurrence of residue, see Lem. 10.6.3. Lem. 10.6.1 ensures that integer CRA finally terminates. Wang's [129, 130] and Monagan [90] strategies ensure that rational reconstruction terminates in the case of CRA and p-adic lifting. During the course of Alg. 14.1.2, Alg. 14.1.5 and the algorithm of Sec. 14.1.3 at least one of the results thatare reconstructed by CRA or p-adic is not affected by variable preconditioning, thus ensuring termination.

Correctness of CRA is defined in Lem. 10.6.1 by reoccurrence of the residue. Several CRA steps might also be used certify the result of early terminated p-adic lifting and rational CRA with $1-\epsilon$ probability of correctness, see 10.6.2. Reoccurrence of residue ensures that it is correctly computed over $\mathbb{Z}$ with $1-\epsilon$ probability. The actual result is recovered from the residue by inverting the preconditioning, and thus has the same $1-\epsilon$ probability of correctness.

14.2 Complexity Analysis of Alg. 14.1.5

In this section we study the complexity of strategies RatDet, PrecDet and PrecBonus - of Alg. 14.1.5. Without loss of generality, we assume that N_i', computed in line 5 of Alg. 14.1.5 are qual to 1. Then, we divide Alg. 14.1.5 into steps that are necessary for each strategy and assume that termination happens for the corresponding value $\det(A)$ by $RatET$ or $D\det(A)$ and $\frac{D}{\pi}\det(A)$, $\pi = \prod \tilde{s}_k$ by $IntET$ respectively.

In subsection 14.2.1 we present the analysis of the general case, where we assume that the entries of the matrix are fractions with numerators and denominators bounded by $\|A\|_r$. Then, in subsection 14.2.2, we will focus on two special cases i.e. matrices of decimal fractions, Hilbert and Lehmer matrices.

Matrices of decimal fractions represent the propitious case, in which the growth in norm and the quality of preconditioning is good. Hilbert matrices represent the case, in which

the growth in norm is significant but the quality of preconditioning is good. Lehmer matrices represent the most difficult case, when the growth in norm is significant, and the quality of preconditioning is poor in both the case of denominator and numerator approximations. See Sec. 13.2 and 13.4 for experimental evaluation of preconditioning.

Introspective choices in Alg. 14.1.5 ensure that the asymptotic complexity of Alg. 14.1.5 will be equal the minimal complexity of the components. The additional cost is negligible, which we explain in Thm. 14.2.2.

The complexity of the strategies RatDet, PrecDet and PrecBonus of Alg. 14.1.5 depends on the number of iterations required by the CRA loop. Then, depending on the strategy, we have to include the cost of computing modular image of the matrix, the cost of the rational reconstruction or the cost of p-adic lifting. We use the early termination condition, and thus, the number of steps required for the computation of $\det(A)$ depends on the values:

- m - the size of the matrix;
- n, d - the real values of the numerator and denominator of $\det(A)$;
- D - approximation of d, given in Eq. (13.5) (or, in Eq. (14.2) in the case when *Correction* is run), and $s_m(\tilde{A})$, the approximation of n.

The cost of imaging depends on the number of non-zero and rational elements, and the maximum norm of matrices i.e. $\|A\|_r = \max(\|a_{ij}\|, b_{ij})$ and $\|\tilde{A}\| = \|\operatorname{diag}(D_i)A\|$, see Lem. 10.2.1.

We will give the complexities in the case of dense matrices, assuming that fast matrix multiplication in $O(m^\omega)$ bit operations is implemented. Analogous results can be obtained for sparse variants of algorithms.

14.2.1 General case

Lemma 14.2.1 (Cost of Strategies of Alg. 14.1.5) *Let $A = \left[\frac{a_{ij}}{b_{ij}}\right]$ be a $m \times m$ dense rational matrix with numerators and denominators bounded by $\|A\|_r$ in absolute value. Let $D = \prod D_i$, where $D_i = \gcd(b_{ij})$. Assume that $\det(A) = \frac{n}{d}$ and matrix $\tilde{A} = \operatorname{diag}(D_i)A$ is chosen in line 7 of Alg. 14.1.5. Suppose that random primes p_i less that p are used in the algorithm and that $EEA(p)$ denote the upper bound on the complexity of inverse computation in $\mathbb{Z}_{p_i}$. The complexities of strategies RatDet, PrecDet and PrecBonus for computing $\det(A)$ are*

1. $O^{\sim}\left(k(m^2 \min(\log(\|\tilde{A}\|), EEA(p)\log(\|A\|_r)) + m^\omega)\right)$ *for RatDet, where $k = O(\log(n) + \log(d))$ in the* **worst case***;*
2. $O^{\sim}\left(\log(\frac{Dn}{d})(m^2 \min(\log(\|\tilde{A}\|), EEA(p)\log(\|A\|_r)) + m^\omega)\right)$ *for PrecDet in the* **worst case***;*

3. $O^{\sim}\left(k'm^2\log(\|\tilde{A}\|)\right)+O^{\sim}\left(\log(\frac{Dn}{ds})(m^2\min(\log(\|\tilde{A}\|),EEA(p)\log(\|A\|_r))+m^{\omega})\right)$, *for PrecBonus, where* $s=s_m(\tilde{A})$ *and* $k'=O(\log(s))+O^{\sim}(\log(m\log(\|\tilde{A}\|)))$ *is the maximal number of iterations needed to compute* s *by early terminated* p*-adic lifting in Alg. 6.3.1. This is the* **expected** *complexity*[1].

PROOF First, notice that the complexity of $RatIm$ is $O^{\sim}(\log(p)m^2\log(\|A\|_r)EEA(p))$ by Lem. 10.2.1. Analogously, the complexity of $IntIm$ is $O^{\sim}(\log(p)m^2\|\tilde{A}\|)$ for $\tilde{A}$. Procedure $ChooseImage$ determines the best imaging scheme for matrices $A,\tilde{A}$, and thus the complexity of imaging is asymptotically $O\left(\log(p)m^2(\min(\log(\|\tilde{A}\|),EEA(p)\log(\|A\|_r)))\right)$. The time of modular determinant computation is $O(\log(p)m^{\omega})$, using modular LU decomposition.

In the case of RatDet, early termination is possible after $O(\log_p(n)+\log_p(d))$ steps using Wang's [129] or Monagan [90] strategies, see Sec. 11.3. By using ideal scheduling strategy of 10.7, the number of steps of the CRA loop is $O^{\sim}(k)$ and the cost of final rational reconstruction is $O^{\sim}(k)$ as well. Analogously, the number of steps in PrecDet is at most $O_p^{\sim}(\log(\frac{Dn}{d}))$. Running procedure $Correction$ in line 4 of Alg. 14.1.5 can improve the number of steps slightly. Prop. 13.1.1,13.3.1, 14.1.2 ensure that we are always reconstructing an integer value less than $\frac{Dn}{d}$.

The complexity of PrecBonus has to be analyzed in the average case, assuming that the expected value $\frac{s_m(\tilde{A})}{\tilde{s}_m}$ is $O(1)$, where $\tilde{s}_m$ is the output of Alg. 6.3.1. p-adic lifting is run to solve Eq. $\tilde{A}x=b$ for random vector b s.t. $\|b\|=O(m\log(m\|\tilde{A}\|))$, see Thm. 6.3.2. Thus, the bitsize of x is $O(2\log(s)+\log(m\log(m\|\tilde{A}\|)))$ in the expected case, which gives the required asymptotic estimation of k'. Prop. 14.1.2 ensures that we are reconstructing an integer value less than $\frac{Dn}{d\tilde{s}_m}$, which is $\frac{Dn}{ds}$ in the expected case, assuming that 2 random systems of equations are run and the results of early terminated p-adic lifting are correct.

The design of Alg. 8.4.1 ensures that the worst case complexity of PrecBonus is at most $O^{\sim}(k'm^2\log(\|\tilde{A}\|))+O^{\sim}(\log(\frac{D}{d}\frac{n}{s})(m^2\log(\min(\|A\|_r,\|\tilde{A}\|))+m^{\omega}))$, as the cost of all solvings except the first one is dominated by CRA iterations.

Special care should be taken if we consider the use of Alg. PrecBonus. As $\log(\|\tilde{A}\|)$ can potentially be $O(m\log(\|A\|_r))$ and, by taking pessimistically $k'=O(m\log(m\|\tilde{A}\|))$, the worst case complexity is about $O^{\sim}(\log(m^5))$, which is worse than for the CRA computation. If this is the case, it should be detected by procedure $ChooseLIF$.

The gain of computing $s_m(\tilde{A})$ can be significant, as it is the case of Alg. 8.4.1. Experiments in Sec. 13.4 confirm, that $s_m(\tilde{A})$ approximates well the numerator n. Worse results have been obtained for Lehmer matrices, but the relative error was $O(1)$ nonetheless.

We summarize the result for Alg. 14.1.5 in the following theorem.

Theorem 14.2.2 (Cost of Alg. 14.1.5) *Let* $A=\left[\frac{a_{ij}}{b_{ij}}\right]$ *be a* $m\times m$ *dense rational matrix with numerators and denominators bounded by* $\|A\|_r$ *in absolute value. Suppose that*

[1]See Def. 2.3.6; additionally we assume that the results of p-adic lifting are correct; we do not detect errors in PrecBonus strategy.

random primes p_i less that p are used in the algorithm and that $EEA(p)$ denote the upper bound on the complexity of inverse computation in $\mathbb{Z}_{p_i}$. Assume that $\det(A) = \frac{n}{d}$ *and the value of D is computed in line 8 of Alg. 14.1.5. Furthermore, assume that $\pi = s_m(\tilde{A}_i)$ is computed in line 14 and is such that $O(\log(\pi)) \in O(\log(n)+\log(d))$. Then, the complexity of Alg. 14.1.5 is $O^{\sim}(Init + kIter)$ where k is equal to* $\min((\log(n) + \log(d)), \log(\frac{Dn}{d}))$. *The cost of $Init$ and $Iter$ are*

$$Init = m^2 \max(\log(\|\tilde{A}\|), EEA(p)\log(\|A\|_r)),$$
$$Iter = m^2 \min(\log(\|\tilde{A}\|), EEA(p)\log(\|A\|_r)) + m^{\omega}.$$

PROOF Construction of $\tilde{A}_i, i = 1, 2$ is dominated by m computations of lcm of m numbers. The size of lcm can reach $m\log(\|A\|_r)$ in the worst case. Thus, the cost is $O^{\sim}(m^2\log(\|A\|_r))$ if fast gcd algorithm is used. The cost of *Correction* in line 4 of Alg. 14.1.5 is dominated by m^2 computations of gcd of numbers at most $O(\log(\|\tilde{A}\|)$ in size and is $O^{\sim}(m^2\log(\|\tilde{A}\|))$. Notice, that in terms of $O^{\sim}$ notation, it is equivalent to the cost of one modular imaging on A and $\tilde{A}$ resp. The cost of *ChooseImage* is $O^{\sim}(\log(p)m^2\max(\log(\|\tilde{A}\|), EEA(p)\log(\|A\|_r)))$ as images of A and $\tilde{A}$ have to be evaluated, see Lem. 10.2.1. This corresponds to the *Init* part of the cost.

The time of running function *ChooseLIF*, see Alg. 14.1.4, is bounded by the cost of five modular CRA iterations, as the computation of $\tilde{A}_i v$ can be interrupted if this time limit is exceeded, and negative answer is given in this case.

In the case of strategies PrecDet and RatDet, Alg. 14.1.5 stops, if an integer value less than $\frac{Dn}{d}$ or a rational value $\frac{n}{d}$ is reconstructed, whichever occurs first. That is, Alg. 14.1.5 runs at most $\min((\log_p(n) + \log_p(d)), \log_p(\frac{Dn}{d})$ steps of complexity $\log(p)Iter$ in the case when strategy PrecBonus is not used, see Lem. 14.2.1. We now need to show, that in the case when $O(\log(\pi)) \in O(\log(n) + \log(d))$, the asymptotic number of steps is the same.

We have remarked in Lem. 14.2.1, that only the first solving for $\pi = s_m(\tilde{A}_i)$ is problematic, as time limits are imposed for the consecutive solvings. The use of PrecBonus strategy introduces the following difficulties.

- the result of p-adic lifting is incorrect; in this case, certification conditions for Wang's and Monagan's strategies ensure, that at most $O(k')$ p-adic steps, where $k' = O(\log_p(\pi)) + O^{\sim}(\log_p(m\log(\|\tilde{A}\|)))$, have been performed; later in the algorithm, this case can be identified by a few modular CRA iterations; in this case, π is set back to 1;

- if the result is correct; it has been computed faster than by $O(\log_p(\pi))$ steps of CRA, thanks to the choice we have made by *ChooseLIF*; hence, the time of PrecBonus is less than the time of PrecDet;

- still, during the computation of π, CRA is paused; thus, the residue used for rational reconstruction in RatDet is computed only once π is evaluated; this means that Alg. 14.1.5 has already performed $O(\log(\pi))Iter$ bit operations; If $\log_p(n) + \log_p(d) \ll \log_p(\pi))$, this means that we may fail to terminate by RatDet in $O(\log_p(n)+\log_p(d))$ iterations of CRA.

By requiring that $\log(\pi) = O(\log(d) + \log(n))$, we ensure this cannot happen and that the asymptotic cost is indeed that of $O(k)$ iterations of CRA, where $k = \min((\log_p(n) + \log_p(d)), \log_p(\frac{Dn}{d}))$.

Remark 14.2.3 Experiments in Sec. 13.4 suggest, that the error $N_{err} = \mathrm{bs}(\frac{n}{\pi})$ is usually small, so that rational reconstruction of $\frac{n}{\pi d}$ might occur faster than the reconstruction of $\frac{n}{d}$. We may thus attempt to reconstruct both values in RatDet.

14.2.2 Complexity in Special Cases

By the precedent remarks it is visible, that the analysis of the strategies should be divided into two main cases, following the evaluation of norms of A and $\tilde{A}$ in Sec. 13.2. One would consist of matrices, whose entries are given by decimal fraction, or more generally, where the common denominator of all entries, the common denominator of the rows and the norm of A are of the same order i.e. $D(A) = O(D_i) = O(\|A\|_r)$. In the other case, matrix entries are given as fractions with different denominators. We will study the complexity of the algorithms on the example of Hilbert and Lehmer matrices.

In the case of matrices of decimal fractions, let us further assume that $\|A\|_r$ is $O(1)$. This would be the case of numerous ill-conditioned matrices emerging from different applications in science and engineering.

Corollary 14.2.4 (Case $\|A\| = \|\tilde{A}\| = O(1)$) The worst case average complexity of Alg. 14.1.5 in the case when $\|A\|_r = O(\|\tilde{A}\|) = O(1)$ is $O^{\sim}(\log(s)m^2) + O^{\sim}(km^{\omega})$, where $k = \min((\log(n) + \log(d)), \log(\frac{Dn}{ds})$ and $s = s_m(\tilde{A})$. Moreover, in the case of matrices of decimal fractions, Alg. 14.1.5 is likely to terminate by PrecBonus and has the same expected complexity as Alg. 8.4.1, see Sec. 8.7.

PROOF The theorem is a straightforward consequence of Lem. 14.2.1 if we notice, that one iteration of p-adic solver has better complexity than one iteration of CRA. Thus, s will be computed in the asymptotic case. Notice, that $IntIm$ will rather be chosen.

The equivalence with the case of random matrices has been explained on p. 201. The implications of the equivalence to the quality of preconditioning have been explained on p. 215. Putting it together, we may conclude, that the algorithm is likely to terminate by applying strategy PrecBonus, and thus is equivalent to Alg. 8.4.1 for A'.

The other group consists of matrices with rational entries given by fractions with very different denominators. As a model case we can consider Hilbert matrices. Hilbert matrices are the matrices of the form $H_m = [\frac{1}{i+j-1}]_{i,j=1..m}$. They are benchmarks examples for many numerical methods. The formula for the determinant of a Hilbert matrix is well known and is given by the equation

$$\frac{1}{\det(H_m)} = \Pi_{k=1}^{m-1}(2k+1)\binom{2k}{k}^2. \tag{14.4}$$

Corollary 14.2.5 (Case of Hilbert Matrices) The complexity of Alg. 14.1.5 in the case of Hilbert matrices is $O(m^{2+\omega})$. In the asymptotic case, termination is achieved by PrecDet.

PROOF For Hilbert matrices, Eq. (14.4) implies that $\log(d) = O(m^2)$ and $n = 1$. The size of entries of H_m is $\log(\|H_m\|_r) = O(\log(m))$ and $\log(\|\tilde{H}_m\|) = O(m)$, see Fig. 13.2. The value of $\log(\frac{D}{d})$ is $0.08\log(d)$, see Fig. 13.10. Thus, in Thm. 14.2.2, $k = O(m^2)$ and is smaller for preconditioned strategies PrecDet or PrecBonus. Complexity of *Init* is $O(m^3)$. Complexity of *Iter* is $O(m^2 EEA(p) + m^\omega) = O(m^\omega)$ if *image* = *RatIm* is chosen. Complexity of one lifting step is $O(m^3)$ compared to $O(m^\omega)$ in the case of modular iteration, thus, strategy PrecBonus is not chosen in the asymptotic case. This is however not a problem, as $\log(s_m(\tilde{H}_m) = O(m)$, as we have experimentally showed on p. 220, that the number of invariant factors of H_m is linear in m. Thus, we do not expect strategy PrecBonus to be asymptotically faster than PrecDet.

Corollary 14.2.6 (Case of Lehmer Matrices) The complexity of Alg. 14.1.5 in the case of Lehmer matrices is $O(m^{1+\omega}\log(m))$. In the asymptotic case, termination is achieved by RatDet.

PROOF For Lehmer matrix L_m and $\det(L_m) = \frac{n}{d}$, our experiments have shown that d is asymptotically $O(m\log(m))$ and n is $O(m)$. The size of entries is $\log(\|L_m\|) = O(\log(m))$ and $\log(\|\tilde{L}_m\|) = O(m)$, see Fig. 13.2. The value of $\log(\frac{D}{d})/\log(d)$ is $O(m)$, which means that $\log(\frac{D}{d}) = O(m^2\log(m))$, see Fig. 13.11. Thus, termination happens by RatDet in $k = O(m\log(m))$ steps. Complexity of *Init* is $O(m^3)$. Complexity of *Iter* is $O(m^2 EEA(p) + m^\omega) = O(m^\omega)$ if *image* = *RatIm* is chosen. The complexity of one lifting step is $O(m^3)$ compared to $O(m^\omega)$ in the case of modular iteration, thus, strategy PrecBonus is not chosen in the asymptotic case. For smaller m, if PrecBonus is run, this may result with a slowdown of the algorithm, compared to the optimal RatDet variant. Still, by Thm. 14.2.2 we obtain the required complexity.

14.3 Experiments

In this section we will present the results of timing comparison for various problems discussed in Ch. 14. We will perform experiments on sets of random matrices considered in Sec. 13.2 and Sec. 13.4. That is, we will consider random matrices of decimal fractions and random matrices given according to distributions *Rat* and *RatUni* as well as particular examples of matrices, namely Hilbert matrices, Lehmer matrices and BasisLib+ Collection.

We start by the evaluation of imaging schemes in Sec. 14.3.1 i.e. we analyze the output of *ChooseImage* procedure. This provides information complementary to Sec. 13.2, where the growth in norm has been evaluated.

Then in Sec. 14.3.2, we focus on the computation of the determinant of rational matrices. We start by running Alg. 8.4.1 on matrix $\tilde{A}_1 = \mathrm{diag}(D_i)A$, where D_i is the common

denominator of the ith row. This corresponds to strategy PrecBonus and provides information supplementary to [118, 117], where system solving for similar classes of matrices (in particular, for Hilbert, Lehmer and BasisLib+) has been considered.

Based on the results of Sec. 13.4, we have identified a subclass of problems, for which strategy PrecBonus is not optimal, as the error introduced by preconditioning overcomes the actual result, see Sec. 13.4 for discussion. For those matrices, we run rational CRA, see Sec. 12.1.1 and compare the timings with previous results. This corresponds to strategy RatDet.

Finally, we run the full introspective algorithm of Alg. 14.1.5, which implements *ChooseImage*, *Correction* (c.f. Alg. 14.1.3) and *ChooseLIF* (c.f. Alg. 14.1.4). We analyze the choices made by the algorithm and compare the timings with strategies PrecBonus and RatDet.

We finish the section by the evaluation of the introspective algorithm for the rational characteristic polynomial in Sec. 14.3.3. It implements *ChooseImage* procedure and preconditioning defined by Eq. (13.6). Integer precondition CRA is run for preconditioned polynomial coupled with geometrically scheduled rational reconstruction. In the case of rational reconstruction, coefficients are reconstructed one by one, starting with c_{m-1}. After successful reconstruction of c_i, its denominator $D(c_i)$ is used to precondition c_{i-1}, see Sec. 14.1.3. We compare both preconditioning strategies and point out differences with the determinant computation.

Algorithms evaluated in this section are implemented in the LinBox, in files "linbox/-algorithms/charpoly-rational.h" and "linbox/algorithms/det-rational.h". We used dense matrix implementation and apply BLAS routines available in LinBox. For some examples, this resulted with memory trashing, thus we consider using the sparse variant of the algorithm in future. For some examples of larger size we were obliged to interrupt the computation after several hours of running. Unless otherwise stated, the experiments presented in this section were performed Intel(R) Core(TM)2 Duo 2.66GHz CPU with 4Gb memory, running Linux.

14.3.1 Homomorphic Imaging

In this section we will compare rational imaging *RatIm* and *IntIm*. That is, for rational $m \times m$ matrix $A = [\frac{a_{ij}}{b_{ij}}]_{i,j=1..m}$ we will consider the cost of taking modular images $\frac{a_{ij}}{b_{ij}} \mod p = (a_{ij} \mod p) \cdot (b_{ij} \mod p)^{-1}$, $i, j = 1..m$. This corresponds to *RatIm* imaging. Then we will consider preconditioned matrix $[\tilde{a}_{ij}]_{i,j=1..m} = \tilde{A} = (\operatorname{diag} D_i)A$, where D_i is the common denominators of the ith row, and consider the cost of taking modular images $(\tilde{a}_{ij} \mod p)$, $i, j = 1..m$, for integer entries $\tilde{a}_{ij}$. This corresponds to *IntIm*.

The average timing of modular imaging has been evaluated based on the average of 100 image computations for different primes of the same bitsize. The size of primes has been chosen depending on matrix dimension in such a way, that it enables BLAS routines to be run for the matrix, see [35, 109, 36].

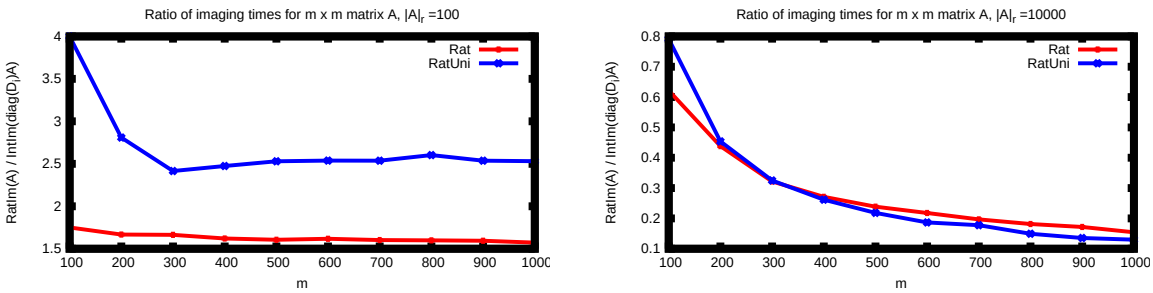

Figure 14.1: Ratio of imaging times $RatIm/IntIm$ for random *Rat* and *RatUni*-type rational matrices. Left: $\|A\|$=100; Right: $\|A\| = 10000$.

1. Case of random matrices

   Average time for 5 matrices has been computed, matrix sizes are $100, 200, \ldots, 1000$.

   a) In the case of random matrices of decimal fractions, the ratio $RatIm/IntIm$ of rational to integer imaging is constant.It is equal to 4.7 on average for matrices with entries given by fractions of 5 decimal places and 10.09 in the case of 16 decimal places precision. This is the cost of redundant division, as the denominator is inverted for each entry separately, despite the fact that it is equal $10^k, k = 6, 17$ for most entries.

   b) In the case of random matrices, $\|A\| = 100$, the ratio $RatIm/IntIm$ of rational to integer imaging decreases slightly with m and eventually stabilizes, see Fig. 14.1 (left). This corresponds to the slight growth of $\log(\|\tilde{A}\|)$ with m, which can be observed in Fig. 13.1. Notice however, that the ratio of timings stabilizes for bigger dimensions than the norm ratio, which is natural, as the norm depends on one maximal entry, and imaging on all of them. The ratio is 1.75 to 1.57 for distribution *Rat* and 4 to 2.54 for distribution *RatUni*.

      The ratio of imaging times for *Rat* and *RatUni* in the case of the same matrix size is 1.43 (1.31 to 1.5) on average for $RatIm$ and 2.35 (2.04 to 3.00) for $IntIm$, i.e. *Rat* gives worse results. Notice, that the norm ratio is 1.05 on average and thus the norms are almost the same. This is in fact the most significant difference between the distributions we have been able to identify.

   c) In the case of random matrices, $\|A\| = 10000$, the ratio of timings is definitely decreasing, see Fig. 14.1 (right), from 0.62 to 0.15 (resp 0.79 to 0.13) in the case of *Rat* to *RatUni* distributions.

      Smaller ratio has been obtained in the case of *Rat* distribution for smaller m, and for *RatUni* distribution for bigger m. This can be explained by the fact, that bigger denominator occur with bigger probability in *RatUni* distribution, which means that the entries of $\tilde{A}$ are on average bigger for *RatUni* distribution, despite the fact that the maximum norm $\|\tilde{A}\|$ is smaller by 5% in this case.

      Thus, $RatIm$ can result in even bigger gain than in the case of *RatUni* distribution. Notice, that the norm ratio $\log(\|\tilde{A}\|)/\log(\|A\|_r)$ is smaller for *RatUni* distribution and comparison of norms does not lead to such conclusion. The norm ratio is

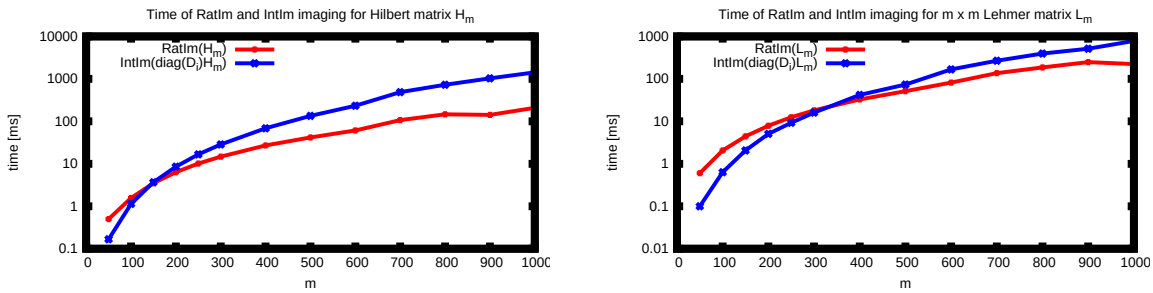

Figure 14.2: Times of $RatIm$ and $IntIm$ for Hilbert (left) and Lehmer (right) matrices. Times in miliseconds are given. Scaling is logarithmic.

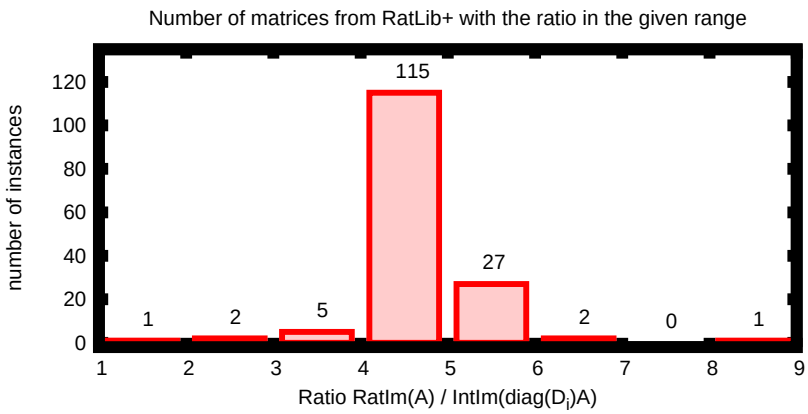

Figure 14.3: Distribution of the ratio of imaging times $RatIm/IntIm$ for BasisLib+ collection. Intger imaging is always better.

eventually bigger for $RatUni$ for $\|A\| = MAX\_INT$. This also means that the comparison of norms is not enough to determine the best imaging scheme.

We have run some examples in the case of random matrices, $\|A\| = MAX_INT$. The ratio $RatIm/IntIm$ is even smaller, ranging from 0.29 in the case of $m = 100$ to 0.08 for $m = 400$.

2. Case of Hilbert and Lehmer matrices

   Fig. 14.2 presents the ratio $RatIm/IntIm$ in the case of Hilbert (left) and Lehmer (right) matrices. At first, integer imaging is better, but with growing matrix size m the ratio decreases and rational imaging starts to win. The threshold occurs at about $m = 150$ for Hilbert matrices and $m = 300$ for Lehmer matrices.

3. Case of Collection BasisLib+

   We have computed imaging times for 153 matrices. For all cases, rational imaging is slower than integer imaging. The ratio $RatIm/IntIm$ is 4.74 on average, which agrees well with the average found for random matrices of decimal fractions with 5 decimal places. In fact, 39 % of matrices in BasisLib+ are represented by decimal fractions.

The distribution of ratio is presented in Fig. 14.3. For the 'easiest' example of `car4`, the ratio $RatIm/IntIm$ is equal to 8.14. The ratio $\log(\|\tilde{A}\|)/\log(\|A\|_r)$ is 0.8 for this example. The difficult examples, for which the ratio is smallest, include `gen1 (1.37),gen4 (2.34),gen4 (2.47)`. The ratio is given in the parentheses. Those examples have also been identified in Tab. 13.1 as hardest. Matrices have rational entries of relatively big bitsize, and the denominators come from a wide range of numbers. The ratio of other examples `pilot4,jendrec1` from Tab. 13.1, is also below 4.

14.3.2 Determinant Computation

In this section, we will present the experimental evaluation of the introspective rational determinant algorithm, which is given in Alg. 14.1.5. We have compared the new implementation of the rational algorithm, which can be found in "linbox/algorithms/det-rational.h", with other existing implementations in LinBox i.e. with our implementation of Alg. 8.4.1, see Sec. 8.8, and rational determinant computation by rational CRA, see Sec. 12.1.1. This corresponds to strategies PrecBonus and RatDet of Alg. 14.1.5 respectively. Rational CRA has been run in the cases, when we have detected that the quality of preconditioning is not satisfactory.

For Alg. 8.4.1, rational matrix is represented by $\tilde{A} = \mathrm{diag}(D_i)A$, where D_i, $i = 1..m$, is the common denominators of the ith row. Integer imaging $IntIm$ and integer early termination condition $IntET$ is used through the algorithm. We will refer to this algorithm as PrecBonus . In the case of rational CRA, rational imaging $RatIm$ and termination condition $RatET$ for rational reconstruction is used. We will refer to this algorithm as RatDet.

In the implementation of Alg. 14.1.5, which we denote by IntroRatDet, we have implemented the $Correction$ procedure by Alg. 14.1.3, the choice of imaging scheme by $ChooseImage$, and the choice between strategies PrecBonus and PrecDet, see Alg. 14.1.4. Decisions are made based in timings, which results in an introspective algorithm.

The choice between PrecDet and PrecBonus is made base on decision, whether the largest invariant factor of $\tilde{A}$ should be computed by p-adic lifting. We assume that primes of the same size are used in the case of p-adic lifting and CRA, and the size is such that it allows for BLAS routines to be run, see [35, 109, 36].

This is however not always the case as taking bigger primes might be necessary in the asymptotic case, when the result is large, in order early terminated with the required probability, see e.g. [75] and Sec. 10.6.4. In our experiments, we did encounter cases, when the number of primes of the size we have chosen for CRA was not enough to complete the computation. We terminated the computation in this case and returned an error.

For the complexity of p-adic lifting with respect to p, see e.g. [97]. In some versions of p-adic lifting, see e.g. [95], primes of bigger size are used in order to obtain better asymptotic complexity bounds. By using bigger primes, we reduce the number of iterations

but increase the cost of one iteration. Thus, normalized times $\overline{T}_{CRA}$ and $\overline{T}_{LIF}$ have to be computed in Alg. 14.1.4. The normalized time is given by formula

$$\overline{T} = \frac{T_p}{\log(p)},$$

where T_p is the time of one iteration for prime p, for both p-adic and CRA schemes.

1. Case of random matrices

   Random matrices of decimal fractions and rational fractions are considered, see Sec. 13.2.1 and Sec. 13.4.1. Matrices of random rational fractions are given according to *Rat* and *RatUni* distributions, see 13.2.1 for definition.

   The quality of preconditioning for all types of random matrices is generally good. Therefore, RatDet is not considered in this case, as we are able to conclude beforehand that other strategies are better. The challenge for the introspective algorithm in this case is to determine the best imaging scheme by $ChooseImage$ and to choose between PrecDet and PrecBonus, see $ChooseLIF$. We can also evaluate the gain of applying $Correction$ by Alg. 14.1.3.

   a) In the case of random matrices of decimal fractions, the quality of preconditioning is especially good, see Fig. 13.6, where the average number of additional bits induced by the preconditioning is presented. The approximation of numerator by $s_m(\tilde{A})$ is good as well. Thus, we have concluded that strategy PrecBonus give best results, see Cor. 14.2.4. For comparison, we have also run IntroRatDet in this case. Fig. 14.4 presents the average timing for both algorithms.

      In the case of PrecBonus strategy, $Correction$ has not been applied to $\tilde{A}$. Thus, introspective algorithm is quicker for matrices with 16 decimal places and has comparable running times for bigger matrices of 5 decimal places. This means that even in this simple case of matrices, applying $Correction$ can be beneficial. Integer imaging $IntIm$ has been correctly chosen by IntroRatDet. Termination occurred by $IntET$ as expected.

   b) In the case of random matrices of rational fraction, where $\|A\| = 100$, the quality of preconditioning is acceptable, see Fig. 13.7 and integer imaging $IntIm$ is better, see Fig. 14.1 (left). The quality of approximation of the numerator is good as well, which implies that strategy PrecBonus should give best results. We compare the timings obtained by strategy PrecBonus with the times of the introspective algorithm IntroRatDet.

      Fig. 14.5 presents the average timing for both algorithms in the case of *Rat* and *RatUni* distributions. Better timings have been obtained for *RatUni* distribution. This is consistent with the findings for imaging times, see Sec. 14.3.1 and for preconditioning, see Sec. 13.4. Adaptive algorithm is faster by up to 5% for bigger matrices, which corresponds to the gain of $Correction$ procedure. Integer imaging $IntIm$ has been correctly chosen by the introspective algorithm. Termination occurred by $IntET$ as expected.

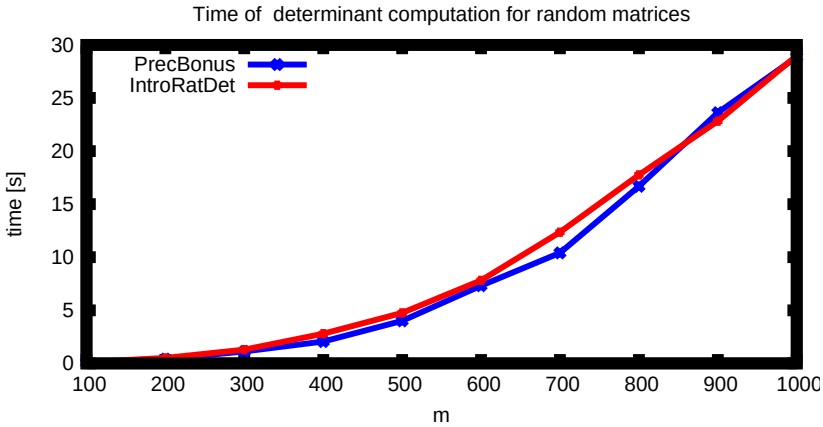

Figure 14.4: Times of rational determinant computation for random matrices of decimal fractions with 5 and 16 decimal places. Average times [in sec.] for 5 random matrices, for strategy PrecBonus and IntroRatDet are given.

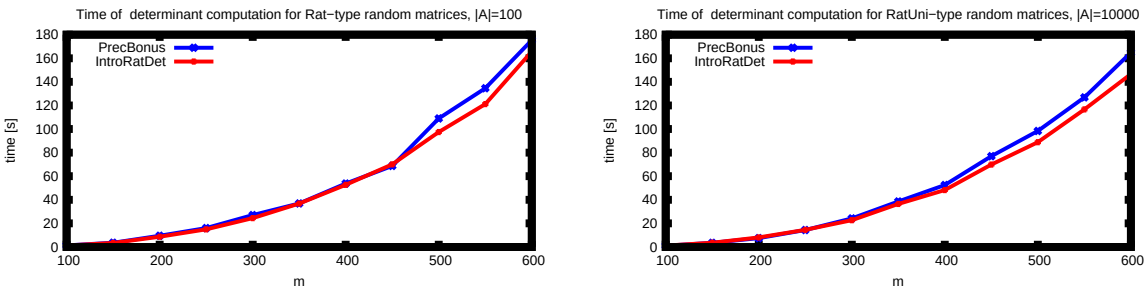

Figure 14.5: Times of rational determinant computation for random matrices of rational fractions for $\|A\| = 100$. Matrices are given according to *Rat* and *RatUni* distribution. Average times [in sec.] for at least 5 random matrices have been computed. Strategy PrecBonus and IntroRatDet are compared.

c) In the case of random matrices of rational fractions, where $\|A\| = 10000$ and $\|A\| = MAX\_INT$, rational imaging $RatIm$ performs better in experiments, see Fig, 14.1 (right). The quality of approximating the denominator and numerator of $\det(A)$ is good. We have obtained results by PrecBonus for matrices of dimension $50, 100, \ldots, 300$ for $\|A\| = 10000$ and dimensions $20, 40$ for $\|A\| = MAX\_INT$. $IntIm$ is eventually chosen by our implementation in the case $m = 50, \|A\| = 10000$ and $RatIm$ is always chosen for bigger matrix sizes.

   In the case $\|A\| = MAX\_INT$, $m \geq 60$, our implementation makes a choice not to perform the p-adic lifting. However, this prevented us form obtaining the results, as the CRA scheme has been interrupted when too many coprime primes have been found. We have used the primes of size 23-24 bits.

2. Case of Hilbert matrices

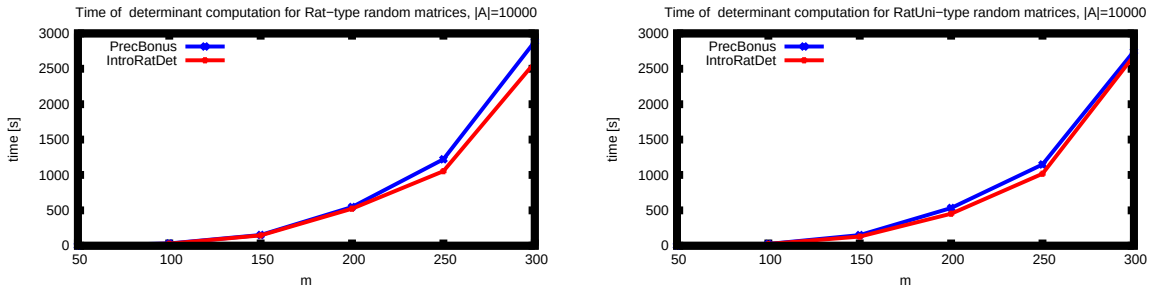

Figure 14.6: Times of rational determinant computation for random matrices of rational fractions for $\|A\| = 10000$. Matrices are given according to *Rat* and *RatUni* distribution. Average times [in sec.] for at least 2 random matrices have been computed. Strategy PrecBonus and IntroRatDet are compared.

In the case of Hilbert matrices, the quality of preconditioning is satisfactory. Fast growth in norm with matrix dimension leads to conclusion, that $RatIm$ is better starting from dimension of about 150, see Fig. 14.2 (left). Also, in the asymptotic case, PrecBonus has worse complexity, see Cor. 14.2.5 for details. The goal of IntroRatDet is to correctly identify the best imaging scheme for given matrix dimension and to judge whether strategy PrecBonus should be applied.

Both $RatIm$ and $IntIm$ has been applied for matrix sizes $m = 50, 100, 150$, which means that the running time for modular determinant computation using both imaging schemes is comparable in this range. From $m = 200$ onwards $RatIm$ is chosen. Termination occurred by $IntET$ condition. PrecBonus is run up to size $m = 600$, which means that PrecDet is chosen as the best strategy for $m = 700, 800, 900, 1000$. Fig. 14.7 presents the running times of Alg. IntroRatDet for $m = 50, \ldots, 1000$. The time for PrecBonus is given for $m = 50, \ldots, 500$, and the running time is approximated from partial results for bigger matrix sizes. We estimate that by using $RatIm$ we are over two times faster. Also, we estimate that by skipping p-adic lifting for bigger matrix sizes, we can gain additional 8% of time for $m = 1000$, when compared to PrecDet with $RatIm$.

3. Case Lehmer matrices

In the case of Lehmer matrices, preconditioning completely fails, see Fig. 13.11 for both the denominator and numerator of $\det(A)$. Thanks to strategy RatDet we have been able to compute the results for matrix sizes up to 1000. Strategy PrecDet and PrecBonus have both worse complexities. The complexity of determinant computation in this case is given by Cor. 14.2.6. The goal of IntroRatDet is be to correctly identify the best imaging scheme, which is $RatIm$ from dimension 300 onwards, see Fig. 14.2. Also, in the asymptotic case, strategy PrecBonus has worse complexity and should not be applied. However it is not clear beforehand if this is the case for the range of dimensions being tested. We have also conjectured in Cor. 14.2.6, that computation of $s_m(L_m)$ by PrecBonus does not pay off, and we may obtain worse running times in this case, compared to pure rational strategy RatDet.

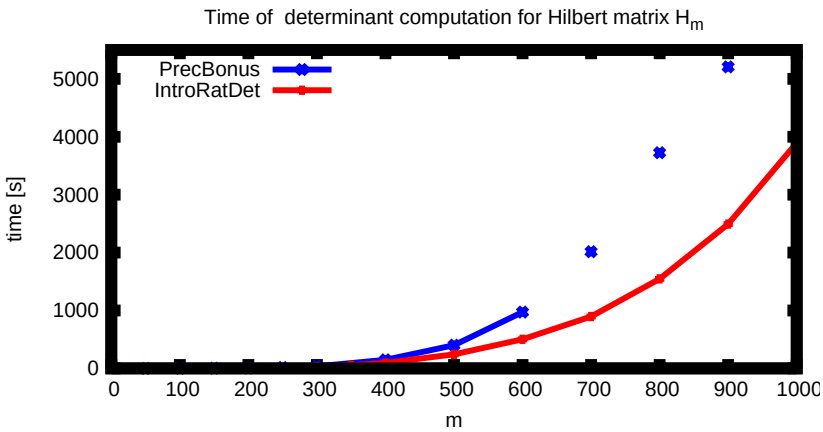

Figure 14.7: Times of rational determinant computation for Hilbert matrices. Times [in sec.] of strategy PrecBonus for $m = 50, \ldots, 500$ are given, and approximated for $m > 500$. Average time algorithm IntroRatDet is given.

Alg. IntroRatDet used integer imaging $IntIm$ for matrix sizes $m < 400$ and rational imaging $RatIm$ for $m \geq 400$. It made the choice to compute $s_m(L_m)$ for all matrices in the range $m = 50..900$. In multiple runs for $m = 1000$, we encounter the algorithm to skip running the p-adic solver only once. Thus, the average and minimal running time are in fact slower than in the case of strategy RatDet, as the results of computations are not used later on in the algorithm. The difference grows in time as the running time of p-adic solver gets worse.

Still, both timings are much better than in the case of strategy PrecBonus. Fig. 14.8 presents the minimal timings for both RatDet and IntroRatDet. Minimal timings represent the case when rational reconstruction is successful at the same optimal step l for both RatDet and IntroRatDet. Thus, it shows the additional cost of IntroRatDet - compared to RatDet in the optimal case. Average timings favor RatDet even more. In IntroRatDet, scheduling in p-adic lifting affects the average timing. Minimal of 10 timings has been chosen.

Several steps can be observed in Fig. 14.8. This is due to geometric scheduling of reconstruction. Steps correspond to cases, when rational reconstruction could not terminated at step 2^l and thus, 2^{l+1} steps are run for the first time. Additionally, a bigger jump for $m = 600$ is caused by the decrease in size of primes used, from 24 to 23 bits. The decrease is required for optimization of BLAS routines, see [35, 109, 36]. In the case of IntroRatDet, geometric scheduling affected both CRA and p-adic lifting, which results with a non-direct correspondence of the steps.

4. Case of BasisLib+ collection

We have computed the determinant for a total of 147 files. Fig. 14.9 presents the distribution of the running times [in sec.] for strategy PrecBonus. 5 matrices can be classify as trivial, as the running times are below 0.1s in this case. 13 matrices are particulary difficult, as the running time is over 1000 seconds. The parameters for

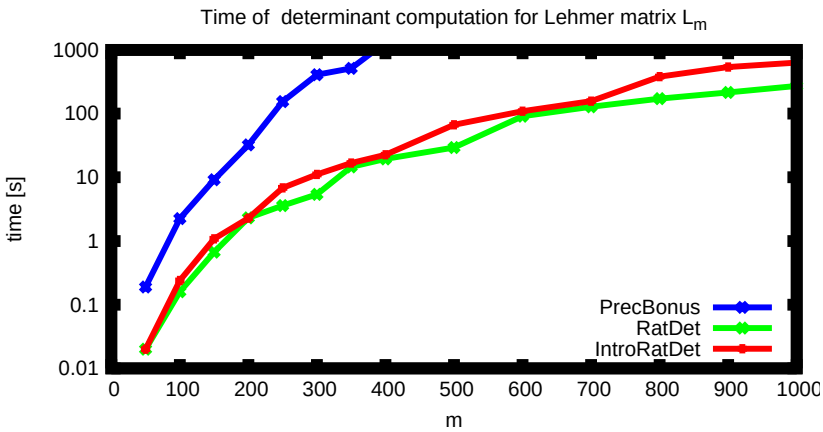

Figure 14.8: Times of rational determinant computation for Hilbert matrices. Time [in sec.] of strategy PrecBonus for $m = 50, \ldots, 500$ is given. Optimal times for RatDet and IntroRatDet are given.

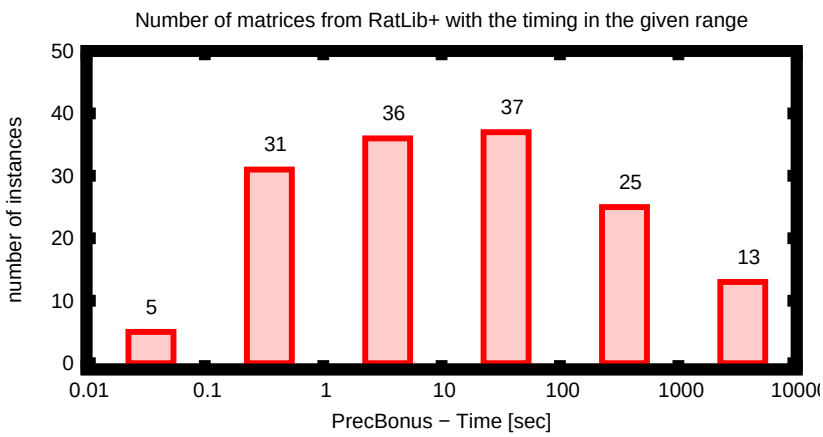

Figure 14.9: Distribution of timings [in sec.] for the rational determinant computation by strategy PrecBonus for BasisLib+ Collection.

those matrices are given in Tab. 14.1. We recall that the computation has not been completed for 10 matrices of the original set BasisLib.

We have identified 61 matrices as suitable for RatDet strategy, based on the condition that the error of denominator approximation D_{err} is greater than $\log(D \det(A))$. We have obtained the results for 60 matrices by RatDet and run out of primes in the case of `bcsstk20`. Fig. 14.10 presents the distribution of the running times [in sec.] (left) for RatDet strategy and the distribution of the ratio of times T(RatDet) / T(PrecBonus) (right). The ratio is concentrated in the interval [0.5,1.5], which corresponds to the relative error $D_{err}/\log(D(\det(A))$ less than 2. Given such values of the ratio, the actual difference between the algorithms depends mostly on quick termination of rational reconstruction (in both p-adic solver and rational CRA). Thus, both algorithms can win, which is confirmed by the distribution of results.

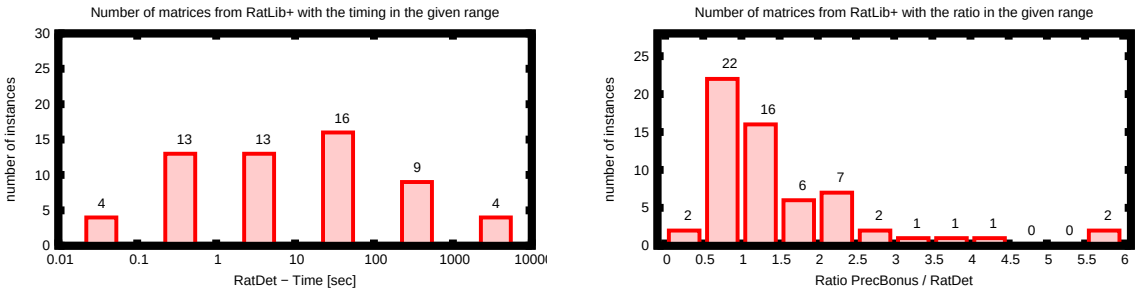

Figure 14.10: Left: distribution of timings [in sec.] for the rational determinant computation by strategy RatDet for BasisLib+ Collection. Right: the distribution of the ratio of timings T(RatDet) / T(PrecBonus); if the ratio is > 1, strategy RatDet is better.

We have obtained better timings for RatDet for 54% of instances, but only for 3 problems where the ratio $D_{err}/\log(D(\det(A)))$ is less than 2. The average speed up is 2.06 in this case. We have obtained worse timings for 46% instances, but only for 3 examples for which the ratio $D_{err}/\log(D(\det(A)))$ is more than 2. The average slowdown is 0.6 in this case. This might be due to the following issues of RatDet

- rational imaging $RatIm$ is used, which slower than integer imaging $IntIm$ for the BasisLib+ Collection, see Fig. 14.3; this effect should however be negligible, as determinant computation dominates the cost of one iteration.
- geometric scheduling is used for the rational reconstruction; suppose that K is the optimal iteration, on which the rational determinant can be reconstructed for the first time; then, we will attempt to reconstruct the result in iteration $2^{\lceil \log(K) \rceil}$; this means that $2 > D_{err}/\log(D(\det(A))) > 1$ might be not sufficient to terminate by rational reconstruction in smaller number of iterations than in the preconditioned variant; moreover, the success of reconstruction is subject to the method used; both Wang [129] and Monagan [90] strategies might fail on iteration $2^{\lceil \log(K) \rceil}$ with some probability.
- the size of residue which needs to be computed depends on the sizes of denominator and numerator; thus, if the size of the numerator is much larger than the size of the denominator and the error, the gain of RatDet is lost in the scheduling; additionally, if the numerator is reasonably well approximated by $\tilde{A}$, it can be computed faster by p-adic lifting; for example, the gain for matrix `scrs8-2r-512`, which have $D(\det(A)) = 1$ and the error is 824 bits, the gain is only 2.72.

The latter issue seems most significant in our case. Indeed, for two examples on which $D_{err}/\log(D(\det(A))) > 2$ but RatDet performed slower and file `bcsstk20`, for which RatDet has failed, the ratio $\log(N(\det(A)))/\log(D(\det(A)))$ is 7,11 and 17 respectively. $s_m(\tilde{A})$ is a big factor of the numerator and can computed faster by p-adic lifting in PrecBonus.

We have run the introspective rational algorithm IntroRatDet to see how it copes with the challenges. For a few smaller matrices, the algorithm could choose both $RatIm$ and $IntIm$, indecisively, this happened for 7 of 147 files. This is due to small cost

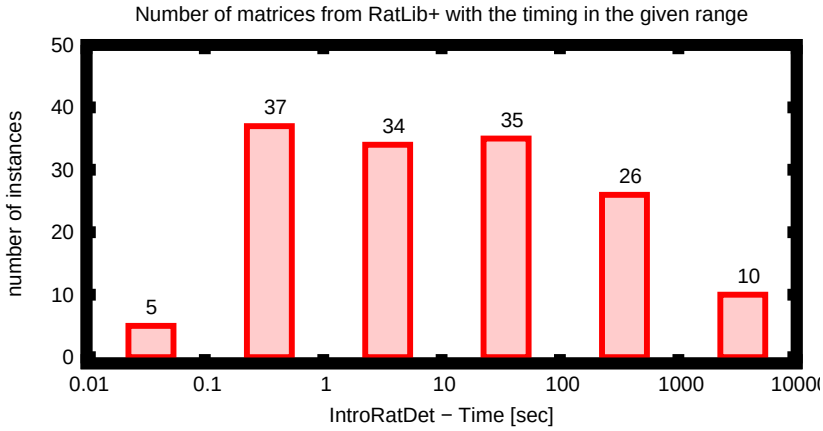

Figure 14.11: Distribution of timings [in sec.] for the rational determinant computation by strategy IntroRatDet for BasisLib+ Collection.

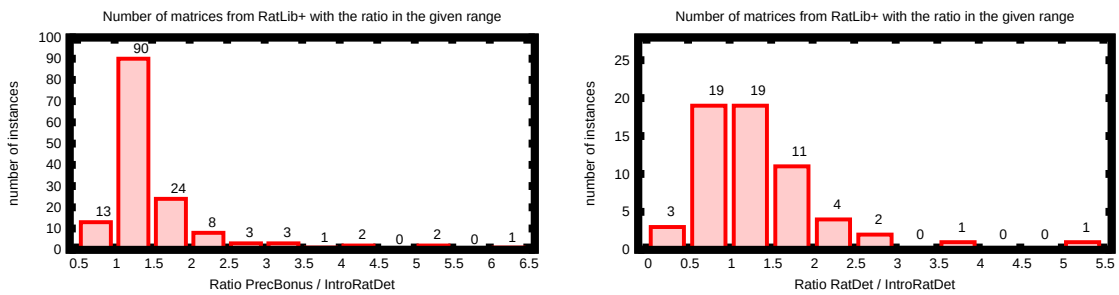

Figure 14.12: Left: distribution of the ratio of timings T(PrecBonus) / T(IntroRatDet) for BasisLib+ Collection. Right: distribution of the ratio of timings T(RatDet) / T(IntroRatDet); if the ratio is > 1, strategy IntroRatDet is better.

of imaging when compared with the modular determinant computation. In general, $IntIm$ is correctly chosen. Algorithm terminated by $RatET$ in 11 cases and by $IntET$ in the remaining ones. In 2 cases, the approximation of $s_m(\tilde{A})$ has been used in the rational reconstruction i.e. $RatET$ occurred for $\frac{\det(A)}{s_m(\tilde{A}}$ and not for $\det A$. In 8 cases, $s_m(\tilde{A})$ has not been computed, as termination occurred by $IntET$ first.

Fig. 14.11 presents the distribution of timings of IntroRatDet. In Fig. 14.12, the distribution of ratios of timings for IntroRatDet and PrecBonus (left), and IntroRatDet and RatDet is shown. IntroRatDet performed better in most cases. The difference between IntroRatDet and PrecBonus can be attributed to applying *Correction* by Alg. 14.1.3 before the actual determinant computation. The most extreme cases of ratio greater than 3 are attributed to not calling the p-adic solver for easiest examples where 8 iterations of CRA yield $IntET$, big gain of *Correction* and using $RatET$. RatDet - performed better in a few cases, in which we have chosen to compute $s_m(\tilde{A})$ instead of performing rational reconstruction. This case has been discussed in Thm. 14.2.2.

A	m	$\|\tilde{A}\|$	$D_{err}/\log(D(\det(A)))$	T_{SOLVE}	T_{DIX}	T_{RAT}	T_{INTRO}
co9	2287	30	1.1	120.68	2321.22	3289.4	1847.66
gen1	329	1576	0.15	2500.29	2524.84	-	2160.05
l30	2492	31	0.01	172.18	2608.89	-	2132.92
model10	1341	429	0.28	594.2	1017.49	-	1500.14
model11	2039	19	2.54	94.15	1873.58	1268.14	1524.01
momentum2	2113	77	1.54	91.69	5692.5	10159.3	5452.43
pilot87	1540	634	0.21	5066.88	5855.59	-	2808.24
pilot87.pre	1625	634	0.18	4670.92	5250.43	-	2557.98
progas	1167	154	0.13	782.94	1026.86	-	790.45
scfxm1-2r-64	2870	14	1.56	5277.49	6044.00	5400.04	4281.32
stat96v4	3139	41	0.05	1363.12	7612.95	-	6835.8
stat96v5	812	193	2.14	128.85	1009.13	1165.12	747.28
stocfor3	1782	16	5.67	82.42	1210.32	843.69	475.78

Table 14.1: Parameters of 13 most challenging problems of BasisLib+ collection in the case of determinant computation. Times for one system solving to approximate $s_m(\tilde{A})$ (T_{SOLVE}), times for strategy PrecBonus (T_{DIX}) , RatDet (T_{RAT}, if applicable) and Alg. IntroRatDet (T_{INTRO})are given. All times in seconds.

Difficult problems in Tab. 14.1 are usually large in dimension m, with the exception of `gen1`,`stat96v5`. For smaller problems, on the contrary, the norm $\|\tilde{A}\|$ is large. The relative error of preconditioning ranges from 0.01 in the case of `l30` to 5.67 in the case of stocfor3. Rational reconstruction slightly improved the timings in 3 cases. In the case of `stockfor3`, Alg. IntroRatDet terminated by $RatET$ as well, performing one reconstruction less than RatDet for this particular run. In the remaining cases, IntroRatDet terminated by $IntET$. Better timings are result of earlier termination in p-adic solver, which have been triggered by applying Alg. *Corrections* 14.1.3. In one case, RatDet is still superior.

14.3.3 Characteristic Polynomial Computation

In this section we evaluate the introspective algorithm for computation of the characteristic polynomial of a rational matrix. Given $m \times m$ matrix A, we denote by $P_A = \sum_{i=0}^{m} c_i x^m$ its characteristic polynomial.

We have implemented the preconditioned CRA algorithm with geometrically scheduled rational reconstruction presented in Sec. 14.1.3. Rational reconstruction is performed at the kth step of CRA, if $k = 2^l$ for some $l = 2, 3, \ldots$. We reconstructed a vector of coefficients $[c_i]$, starting by c_m and reconstructing $D(c_{i-1})c_i$ in the case of ith coefficient. Recall that $D(-)$ denote the denominator of a rational number. We have found the quality of this preconditioning satisfactory in practice. We will comment on this in the description of particular cases.

We remain the reader, that in the case of characteristic polynomial computation, $A \mod p$ might by computed by $RatIm$ in the usual way; for $IntIm$, A is represented as $A = (\mathrm{diag}(D_i)\tilde{A}))$ and $A \mod p = (\mathrm{diag}(D_i \mod p)(\tilde{A} \mod p)$ is computed in the modular arithmetics. This is a slightly different approach than in Sec. 14.3.1, as m modular images and modular inverses of potentially large integer numbers D_i have to be computed as well.

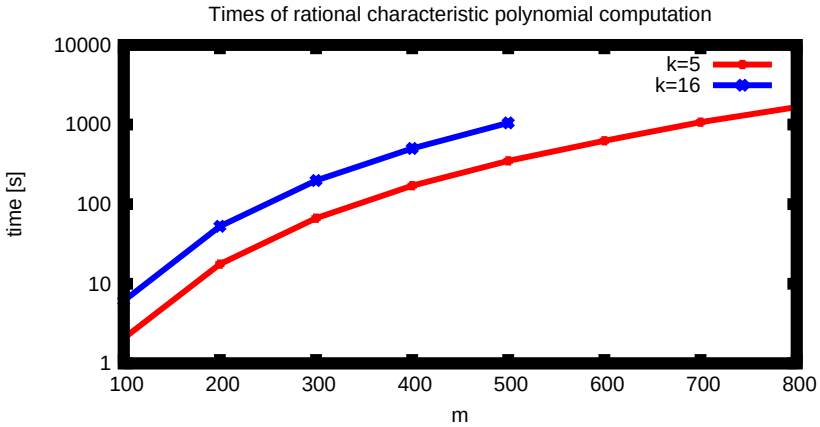

Figure 14.13: Average time [in sec.] of the introspective rational algorithm for characteristic polynomial computation in the case of random matrices of decimal fractions with $k = 5, 16$ decimal places. Scaling is logarithmic.

Then, matrix $A \mod p$ has to be multiplied by a diagonal matrix. Thus, the difference between imaging timings will be less significant than in Sec. 14.3.1. In some cases, rational imaging is chosen even if the results in Sec. 14.3.1 suggest otherwise.

We have compared the timings of computing characteristic polynomial mod p using two different primes for every imaging scheme. In the case, when the computation of characteristic polynomial in modular arithmetics dominates the cost, our check might have produced incorrect answer. The time difference might correspond to fluctuations in timing for characteristic polynomial for different primes. However, this should not be the case on average and, what is more important, this only happens if the time of imaging is negligible anyway.

1. Case of random matrices

   a) In the case of random matrices of decimal fractions $IntIm$ is always chosen by the introspective algorithm. Termination always happened by $IntET$, which proves that the quality of preconditioning in Eq. 13.6 is good in this case. The size of denominators of consecutive coefficients c_i grows linearly with i by an average of 16 and 55 bits in the case of k=5 and 16 decimal places. This agrees very well with the approximation $D(c_{i-1})/D(c_i) = 10^{k+1}$. In the experiment, $D(c_i)$ is always a factor of $D(c_{i-1})$ for $k = 16$, but a few (up to 7) supplementary are present in the case $k = 5$. The average error of approximation for all coefficients is 1.5 bits in both cases. Results are averaged over a few instances of random matrices. Fig. 14.13 presents the average running times for the characteristic polynomial computation.

   b) In the case of random matrices, $\|A\| = 100$, we recall that $IntIm$ is always better in Sec. 14.3.1. In the experiments, $RatIm$ has been chosen for 8 out of 47 instances that we have run. For each matrix size, we have compared the results obtained using $RatIm$ and $IntIm$ and concluded that no it has not resulted in slowdown.

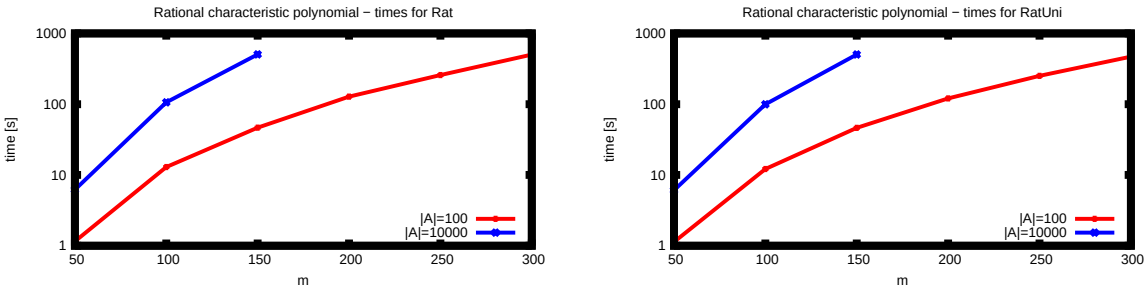

Figure 14.14: Average time [in sec.] of the introspective rational algorithm for characteristic polynomial computation in the case of random matrices of type *Rat* and *RatUni*, with $\|A\| = 100$ and $\|A\| = 10000$ resp. Scaling is logarithmic.

Algorithm terminated by $IntET$ in all except 4 cases, when termination by $RatET$ occurred first. This happened only for matrix dimension $m = 100$, twice for *Rat* and twice for *RatUni*. Similar timings have been obtained in both cases, as the termination happened at approximately the same iteration.

Slightly better timing have been obtained for *RatUni* distribution, which might be result of better imaging times, see Sec. 14.3.1. Fig. 14.14 presents the times of characteristic polynomial computation as a function of matrix dimension m for both distribution and $\|A\| = 100$ and $\|A\| = 10000$. In the case of $\|A\| = 10000$ we have run a limited number of examples for $m = 50to150$. We have observed that both $IntIm$ and $RatIm$ can been chosen and still, comparable timings are observed. Termination occurred by $IntET$ in all cases.

Let us have a closer look at the error of approximation of denominators. In the case when $\|A\| = 100$, we recall from Sec. 13.4 that the relative error of approximating the denominator of $D(\det(A))$ is up to 9%, but decreases quickly with growing m. For $\|A\| = 10000$, the relative error of approximation is between 6% and 7% for the consider range of m, see Fig. 13.8.

Let $D_{app}(i) = \gcd(D(A)^{m-i}, D)$ be given as in Eq. (14.3). Denote by $D_{err}(i)$ the error of approximation of the denominator of c_i i.e. $D_{err} = \log(\frac{D_{app}(i)}{D(c_i)})$. In our experiments, $D_{err}(1)$ is the maximal error for both $\|A\| = 100, 10000$. In the case when $m = 100, \|A\| = 100$, $D_{err}(1)$ is on average 1.17 times bigger than $D_{err}(0) = D_{err}$, where D_{err} is the error of approximation of $D(\det(A))$. The ratio grows to 1.5 for 300×300 matrices. For $\|A\| = 10000$, the ratio $\frac{D_{err}(1)}{D_{err}(0)}$ is 5.15 for 50×50 matrix, but decreases quickly to 2.18 for $m = 150$. For $\|A\| = MAX\_INT$, initial results that we have obtained for $m = 20, 40$ suggest that the situation might be even worst.

In general, this means that taking $D_{app}(i) = \gcd(D(A)^{m-i}, D)$ is not enough to improve the approximation of consecutive denominators $D(c_i)$ and does not eliminate all errors. Surely, in the case when $m \ll \|A\|$, $D(A)$ is close to D, and the situation is even worse.

Thus, we have checked the size of the maximal residue $N(c_i)D_{app}(i)/D(c_i)$ that is to be reconstructed. We have found out, that the sizes of several first residues are com-

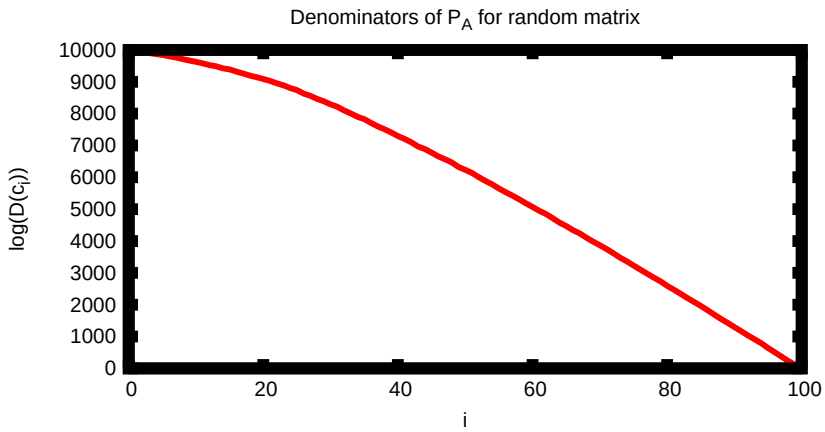

Figure 14.15: Size of consecutive coefficients $\log(D(c_i))$ of the characteristic polynomial P_A, for 100×100 matrix A given according to *Rat* distribution.

parable, and in particular $\log(N(c_1)D_{app}(1)/D(c_1))$ is at most $\log(N(c_0)D_{app}(0)/D(c_0))+1$. Thus, the overall number of steps of preconditioned CRA is not influenced, and the quality of preconditioning is the same as for the case of determinant computation, as we postulated in Sec. 13.4.

Fig. 14.15 illustrates the decrease of $\log(D(c_i))$ as function of i for a particular 100×100 matrix A, given according to *Rat* distribution. The function is decreasing with i, i.e. $D(c_0)$ is biggest. The sizes of several first denominators are comparable.

In our experiments we have also evaluated the quality of approximating $D(c_{i-1})$ by the previous denominator $D(c_i)$. From Fig. 14.15 we may deduce, that we cannot expect $D(c_{i-1})$ to be close to $D(c_i)$, especially for big i. Yet, $D(c_i)$ 'almost' divides $D(c_{i-1})$ i.e. the fraction $\frac{D(c_i)}{D(c_{i-1})}$ has a small denominator. In our experiments, the number of new bits introduced in the denominator is up to 12 for *Rat*, $\|A\| = 100$ and 19 for *RatUni*, $\|A\| = 100$. For $\|A\| = 10000$ and $\|A\| = MAX\_INT$ initial results are slightly worse, as up to 38 bits might have been introduced. The divisibility is actually expected, if we consider that c_i is equal to the sum of $(m-i)$ principal minors up to the sign. Given the Gauss formula for the $(m-i+1)$th principal minor, we can see that all $(m-i)$ principal minors are included in the sum. By summation, several bits might get reduced.

The stabilization of the denominator is also easy to explain. Notice, that computation of coefficient c_i involves only matrix entries $\frac{a_{kj}}{b_{kj}}$ for which $|k-j| = m-i-1$. This means that for $m \times m$ matrix A, a linear number of entries is used to compute several first coefficients. In general c_i is computed using $m + (m-i-1)(m+i)$ coefficients of A.

2. Case of Hilbert matrices

In the case of Hilbert matrices, integer imaging $IntIm$ is chosen only in the case of smallest matrix H_{50}. Then, $RatIm$ is used through the computation. The transition occurs earlier in the case of characteristic polynomial computation, than in the case of homomorphic imaging for $\tilde{H}_m$, see Sec. 14.2. This is due to the fact, that the modular

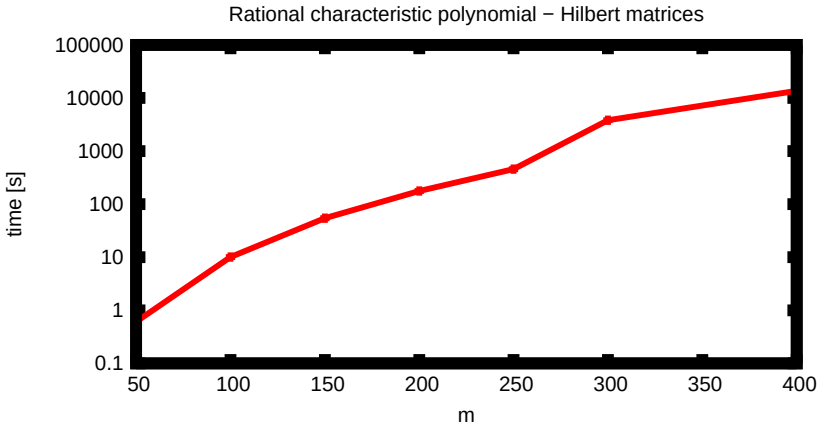

Figure 14.16: Time [in sec.] of the introspective rational algorithm for characteristic polynomial computation for Hilbert matrices. Scaling is logarithmic.

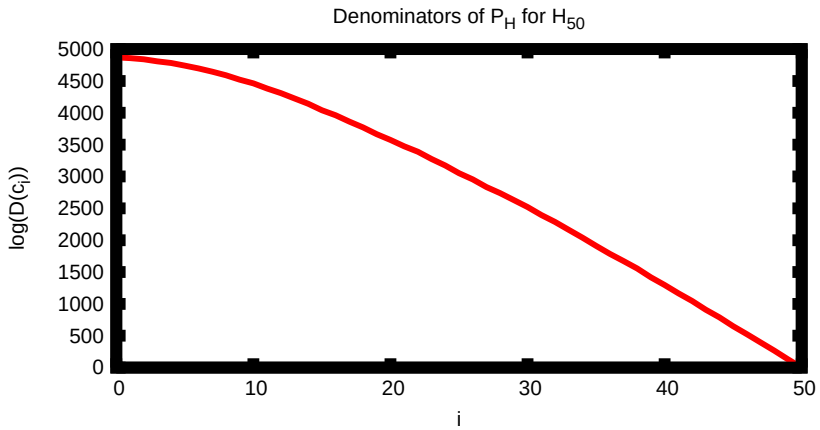

Figure 14.17: Size of consecutive coefficients $\log(D(c_i))$ of the characteristic polynomial P_A, for H_{50}.

image $\tilde{H}_m \mod p$ has to be further left multiplied by $\mathrm{diag}(D_i^{-1} \mod p)$ in order to be suitable for characteristic polynomial computation.

Termination always happened by $IntET$ condition, which is expected, as the quality of determinant approximation is about 8.2% for Hilbert matrices, see p. 1. The error of approximation was always biggest for c_0. We have computed the characteristic polynomial of H_m for matrix dimensions varying from $m = 50$ to 400. Timing is given in Fig. 14.16.

Rational reconstruction is not necessary to complete the computation of the characteristic polynomial of H_m, but we remark that at most 40 supplementary bits are introduced by preconditioning $D(c_i)c_{i-1}$. This makes the error of preconditioning negligible. Yet, this is not enough to force termination by rational reconstruction, as the remaining part of the denominator of $D(c_i)c_{i-1}$ is still significant. The growth of coefficient followed similar pattern as for the case of random matrices, see Fig. 14.17.

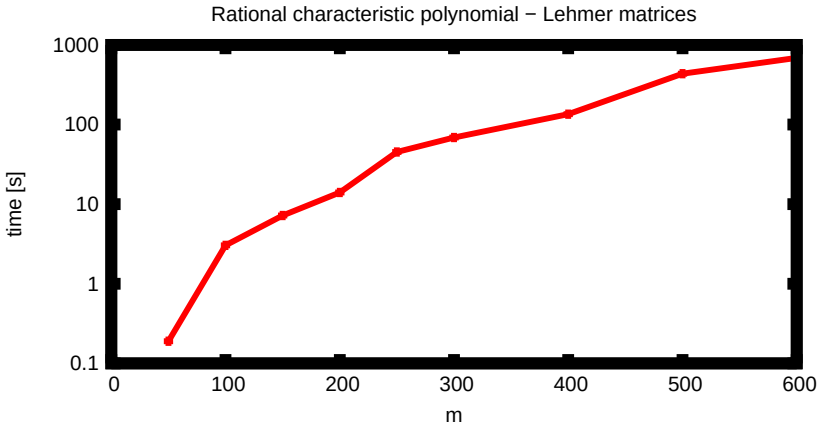

Figure 14.18: Time [in sec.] of the introspective rational algorithm for characteristic polynomial computation for Lehmer matrices. Scaling is logarithmic.

3. Case of Lehmer matrices

   In the case of Lehmer matrices, integer imaging $IntIm$ is chosen last time for m=250; On the other hand, $RatIm$ is chosen first for $m = 200$. This gives defines a transition phase of sizes of m, for which the times of imaging are comparable. The transition occurs earlier in the case of imaging for characteristic polynomial, than in the case of simple homomorphic imaging for $\tilde{L}_m$, see Sec. 14.2.

   Termination always happened by $RatET$ condition, which is expected, as the quality of determinant approximation for Lehmer matrices, see p. 1 is disastrous. We have computed the characteristic polynomial of L_m for matrix dimensions varying from $m =$ 50 to 600. Timing is given in Fig. 14.18.

   It is essential for these matrices to analyze the gain of preconditioning $D(c_i)c_{i-1}$. In Fig. 14.19 gives the size of denominators $D(c_i)$ and the difference in sizes between consecutive coefficients. An interesting pattern occurs, as two consecutive denominators are of the same size. Actually, $D(c_{i-1})$ might be even less than $D(c_i)$ as the difference is often negative. Thus, $D(c_i)c_{i-1}$ is often integer, and we are dealing with overestimation instead of underestimation, which is substantially different from previous cases, and easier to overcome by CRA. A sharp decrease occurs when passing from $D(c_1)$ to $D(c_0)$.

   The preconditioning $D(c_i)c_{i-1}$ is substantially different than previous denominator preconditioners $D_{app}, D_{app}(i)$ or Dx_{app} as the goal is to approximate a divisor of $D(c_{i-1})$ rather than the whole denominator. Therefore we will define the measure of error as $Dc_{err}(i) = \log(D(\frac{c_{i-1}}{c_i}))$. The maximal relative error $\max(Dc_{err}(i)/log(D(c_i)))$ as a function of matrix size m is presented in Fig. 14.20. As implied by Fig. 14.19, maximal error corresponds to c_0. The value of relative error is decreasing with m from 0.29 in the case of $m = 50$ to 0.17 for $m = 600$. That is, the quality of preconditioning is much less than 1, hence satisfactory.

4. Case of Collection BasisLib+

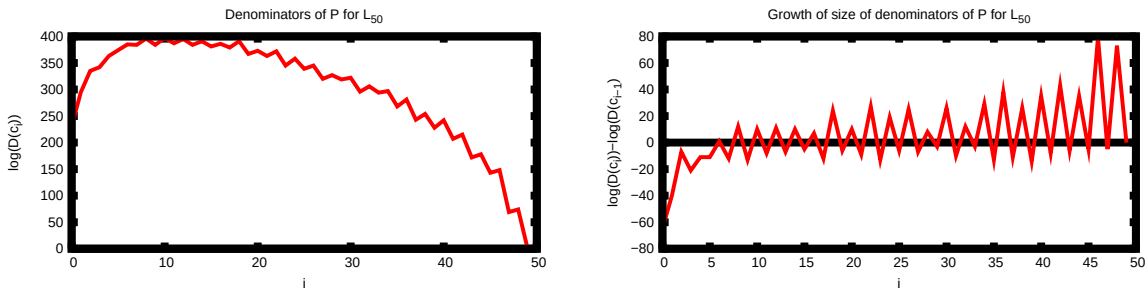

Figure 14.19: Size of consecutive coefficients $\log(D(c_i))$ and the difference $\log(D(c_i)) - \log(D(c_i))$ in the case of the characteristic polynomial P_A of L_{50}.

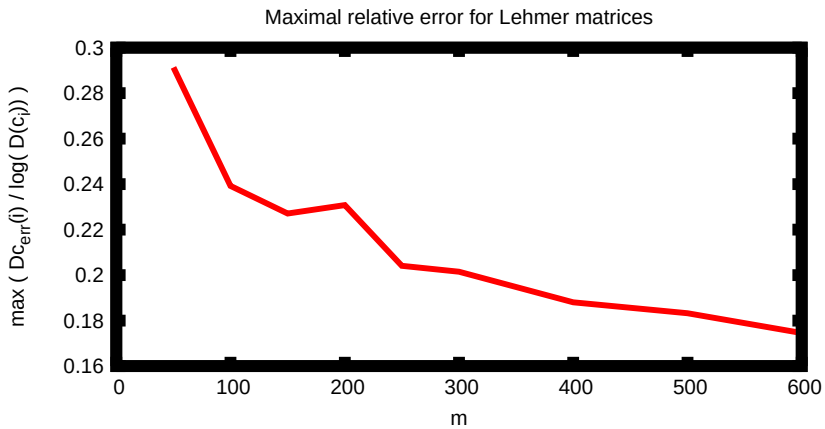

Figure 14.20: Maximal relative error $\max(Dc_{err}(i)/log(D(c_i)))$ of approximating c_{i-1} by c_i for Lehmer matrices.

We have computed the characteristic polynomial for a total of 96 files. $IntIm$ is run for 50 and $RatIm$ for 46 cases. This corresponds to cases, when additional cost of recovering A for its representation $(\mathrm{diag}(D_i), \tilde{A})$ significantly reduces the performance of integer $IntIm$, when compared with Sec. 14.3.1.

Termination happened by $IntET$ for a majority of 61 cases, compared to 34, in which rational reconstruction terminated first. This is significantly more than in the case of determinant computation, where only 11 cases terminated by $RatET$. This should not be due to the growth of error of preconditioning, as the maximal residues reconstructed is on average only 1.15 times bigger than the residue for $D_{app}(0)c_0$, except for two extreme cases of `r05,p05` when the ratio is 9.33. Yet in this cases as well, integer residue reconstructed is smaller than $\max(\mathrm{bs}(c_i))$. Thus, the error of approximation is still acceptable.

In the case when $IntET$ occurred, the average ratio $\max(\log(D_{app}(i)c_i))/\max(\mathrm{bs}(c_i))$ is 0.67, and equal to 1.002 and 1.033 in the only two cases of `scsd8,woodw` when it is greater than 1. This corresponded to 13 and 52 bits respectively, which can be recovered by to up to 3 iterations of CRA using 24-bit primes.

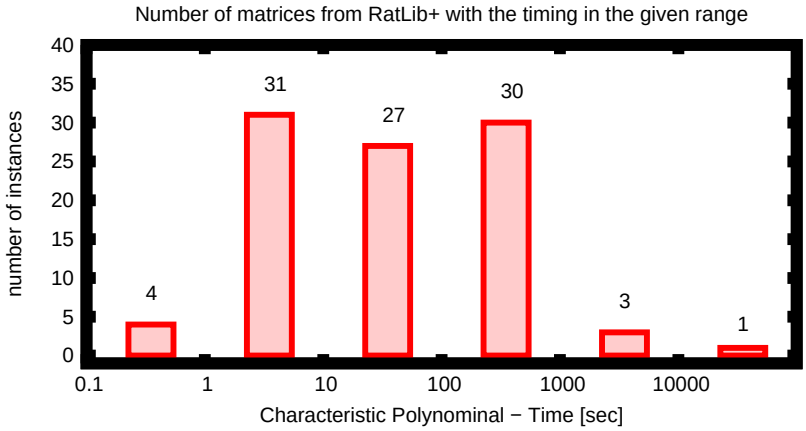

Figure 14.21: Distribution of running times [in seconds] of the characteristic polynomial computation for 96 rational matrices of BasisLib+ collection.

We have also compared $\max(\log(D_{app}(i)c_i))$ to $\max(\mathrm{bs}(D(c_{i-1})c_i))$ in the case of $IntET$ termination, and concluded that the latter is smaller in most cases, and never more than 17 bits higher. On average, the ratio $\max(\mathrm{bs}(D(c_{i-1})c_i))/\max(\log(D_{app}(i)c_i))$ is 0.89 and 0.51 in the minimal case. This means that the quality of approximation $D(c_{i-1})c_i$ is often better than the quality of integer preconditioning we have propose originally. Yet, simpler termination strategy in the case of integer CRA leads to termination by $IntET$. Recall, that geometrically scheduled rational reconstruction, which is used in this implementation, needs in the worst case twice the optimal number of bits in the residue.

Given the quality of preconditioning $D(c_{i-1})c_i$ it is not surprising that $RatET$ is successful in 35% of cases. In the case of $RatET$ termination, we have evaluated the quality of preconditioning by estimating the maximal relative number of bits added to the denominator by preconditioning i.e. $\max(\log(D(D(c_{i-1})c_i))/\log(D(c_i)))$. The ratio is indeed very small and equal to 0.13 on average. In fact, only a few supplementary bits are often present.

To summarize this example, we give the distributions of timings for BasisLib+ collection in Fig. 14.21. The minimal time is 0.27 s. for 106×106 matrix `mkc` ($IntIm$ and $RatET$ are used) and the maximal time is 3.1h for 713×713 matrix `greenbeb` ($RatIm$ and $RatET$ are used). Longer computations have been interrupted. The running time of the algorithm is below 100 seconds for 65% of tested matrices, and more for the remaining 35%.

We have been unable to complete the computation for 52 other matrices, which includes all matrices of Tab. 13.1 and matrix `msc98-ip` from Tab. 13.2. Matrix dimension is the limiting factor, as we have been unable to compute the characteristic polynomial for matrices greater than 1135×1135.

5. Perception Matrices

A	T_{iter} [s]	N_{iter}	T_{all}
perc4249_1	23.67s	21712	5.95d
perc4249_2	23.34s	20614	5.57d
perc4249_3	17.3s	23778	4.76d

Table 14.2: Details on computation of rational polynomial for rational matrix A: time of one modular iteration T_{iter} [in sec.], number of iterations N_{iter}, and overall time T_{all} [in days].

As an application of rational algorithms, we partially solved a problem coming from the domain of image recognition. Methods used in this context, require the estimation of the number of small eigenvalues for certain symmetric matrices. We have been given three examples of 4249×4249 matrices, with entries given as binary fractions with denominator being equal to 2^{106}. Our efforts have focused on computing the exact characteristic polynomial. Some initial experiments have proved the useability of the computed polynomials in (exact) eigenvalues computation. Characteristic polynomial is equal to the minimal polynomial in this case.

Tab. 14.2 gives the number of iterations, the time of one modular iteration [in sec.] and the estimated running time [in days] for the considered matrices. Rational imaging $RatIm$ and integer termination $IntET$ are used. The quality of preconditioning is exemplary good, as the relative error $\log(\frac{D}{D(\det(A))})/\log(D(\det(A)))$ is resp. 0.00275,0.00384 and 0.01947 in the worst case. We think that successful preconditioning has been essential in completing the problem.

Part IV

Smith Form by Reduction of Chain Complexes as Example of Heuristic Algorithm

15

Motivations

One of the applications of the Smith form computation is the computation of homologies and cohomologies of chain complexes, which on its own has many applications in mathematics (see e.g. [46]) and beyond e.g. in graphics ([60]) and medicine ([101]).

The computation of homology follow a general scheme. Based on the initial object, another combinational object is constructed, such that it has the same homology groups. We call it a cellular complex (or simplicial complex, cubical complex, depending on the genesis, for specific constructions, see e.g. [72, 99, 63]). To the cellular complex one bounds an algebraic structure of abelian modules, called a chain complex; and a boundary operator. The boundary operator is a homomorphism between the modules. Apart from some degenerated cases, the modules are free, and the boundary operator can be represented by an exact sequence of matrices.

The matrices that arise in the applications are extremely sparse but at the same time, extremely large. This makes the dense methods [121, 44] irrelevant due to large matrix size. In order to deal with homology computations, effective sparse methods are needed, such as in [38, 39, 37]. In fact, elimination methods based on the original algorithm of Smith and its first modifications [80, 69] seem to perform well and in practice, do not suffer from filling and/or prohibitive growth of coefficients, see e.g. [28].

This may be due to some hidden structure of the object, and thus, of the matrices. We name just a few most common facts.

- non-zero entries are ± 1 in the case of regular complexes,
- there is usually a limited number of entries per row/column, block-like structures are present, which is the result of some geometric restrictions in the case of simplicial and cubical homologies,
- if the resulting homology is relatively simple, most of the computation can be done at a glance.

The object for which the computation of homologies is performed come from different domains. Often, the domain permits a reduction of the object, which leads to another

object, which has the same homology groups but for which the computation "should" be easier. The reduction procedures are often based on the excision theorem, see e.g. [1, Thm. 2.20], [72, Thm. 9.14], which is universal for all theories of homology.

In general, the reduction procedures are heuristic and often greedy. No quantitative results exist which would bound problem sizes before and after reduction. Examples include [94] for cubical homologies and more generally, [73, 93] for chain complexes. See also [47] for an example of reduction from K-theory (motivic cohomology theory) and [34] for its application to the computation of cohomologies. In [37] an overview of other methods is given, which includes the algebraic shifting of [74], minimal free resolution, see e.g. [65], and the computation of a dual complex, see e.g. [99]. See [37] and the references therein.

The computation of a Smith form is the ultimate step in the homology computation. After reductions, matrices of the boundary operator are generated and linear algebra subroutines are called on each object. The development of Smith form algorithms has been a subject of studies on its own now, see Sec. 15.2.2 for references.

It might be however interesting to translate the problem of reductions of chain complexes to the corresponding sequence of matrices and mimic the reduction algorithm by operations on rows and columns of matrices. This lead to the following problem:

Given an exact sequence of matrices (potentially, but not necessarily tied to homology computation), find a way to reduce it, so that the computation of Smith forms for all matrices is faster.

To our knowledge, such question has not been explicitly considered so far in the context of Smith Form computation.

In this chapter we will show how to apply the reductions of [73] to a problem from K-theory, in order to couple it with reductions of [47]. By coupling the reductions, we are able to compute the Smith form of matrices reaching almost $2,000,000 \times 2,000,000$ in size. We show, that by putting the reductions of [73] it the matrix setting, we can reduce the problem to Gaussian elimination, which leads to simplified proofs of correctness, independent of homology theory. The algorithm can also be used for modular rank computation over a field. Additionally, thanks to matrix setting, the idea of coreductions can be introduced. We show that the repeated reductions/coredutions scheme stands behind the successful computation of the Smith forms of large matrices from [34].

15.1 Outline of this Part

The scheme of this part is as follows. We start in Sec. 15.2 by giving a short background on the Smith form and existing algorithms for its computation. We continue by introducing the basic notions from the theory of homology in Sec. 15.3. Then in Ch. 16 we introduce the reductions of [73]. In Sec. 16.2 we present the reductions in the matrix setting and give the main theorem. Then we propose to generalize the reductions of [73] to coreductions and show how to omit some of the assumptions. In Sec. 17.2.3 we present the implementational details and consider the complexity issues. In Sec. 17.3 we present

the computational results and timings. In Sec. 17.3.1 we compare our method to the parallel rank computation presented in [34].

15.2 Background on Smith Form Theory and Algorithms

15.2.1 Theory of the Smith Form

The concept of the Smith normal form was first introduced by H.J. Smith [116] in 1869 for integer matrices. The definition can be extended to the case of Principal Ideals Rings (PIR)[1]. We have the following definition.

Definition 15.2.1 (Diagonal Matrix) Let $k, n \in \mathbb{N}$ and $k, n > 0$. Let $S = [s_{ij}]_{i=1..k, j=1..n}$ be a $k \times n$ matrix over a ring R. We say that S is diagonal if $s_{ij} = 0$ for $i \neq j$.

Definition 15.2.2 (Smith Form) Let R be a PIR. Let $k, n \in \mathbb{N}$ and $k, n > 0$. Let A be a $k \times n$ matrix over R.

Let S be a diagonal $k \times n$ matrix, $S = \mathrm{diag}(s_1, \ldots, s_{\min(k,n)})$, such that

$$s_i \mid s_{i+1} \quad \forall i = 1 \ldots \min(k, n).$$

S is called a (generalized) Smith form of A over R iff there exist $k \times k$ and $n \times n$ matrices U, V, such that $\det(U)$ and $\det(V)$ are invertible over R and

$$A = USV.$$

15.2.2 Historic Background

The first algorithm for the construction of the Smith form was given by Smith [116] in his original paper and consisted of repeated elementary row and columns operations i.e. exchanging rows/columns, multiplying by ± 1 and adding multiples of rows/columns to another one. His approach is presented in many papers, see e.g. [58, Ch. 3 and 8] (the case of PID, generalizable to PIR) or [72, Ch.3] (for integer matrices). For integer matrices, this algorithm may suffer from coefficients swell and may have exponential space and time complexity in the worst case, see [49, 48].

Despite this fact, heuristics for integer elimination has been developed and elimination techniques are applied to solve practical problems. We refer to [48] and the references therein for some examples. See also [64] for strategies of choosing ± 1 pivots. In [28] the authors claim that elimination techniques might be efficient for the computation of homologies of triangulation in design. Also, in [37] an effective heuristic pivoting strategy

[1] Ring R is called **P**rincipal **I**deal **R**ing (PIR) if every ideal of R is generated by exactly one element. A PIR R without zero-divisors is called **P**rincipal **I**deal **D**omain (PID)

for homology matrices is presented, which combined effective implementation leads to an algorithm successful in practice.

It was not until 1979 that the first polynomial time algorithm was given by Kannan and Bachem [80], which was later improved by Chou and Collins [18]. The idea of these algorithms was again to perform elimination by means of elementary row and columns operations but at the same time, to control the coefficient swell. See also [127, Alg. 1] for a nice presentation. The algorithm is based repeated triangulation steps based on (partial) row and column elimination.

In 1989 Iliopoulos presented an algorithm, which computes the Smith Form modulo the determinant in about $O^{\sim}(n^5)$ operations, where n is the size of the matrix, see [69]. As invariant factors are divisors of the determinant, this is equal the output of the algorithm gives the integer Smith form. As remarked by [112] the computation can actually be performed modulo the largest invariant factors (LIF), see Alg. 6.3.1 and [2, 44, 38, 39] for algorithms to compute LIF. For Iliopoulos algorithm, the condition that the computation is performed in modular arithmetics implies that virtually any modular elimination procedure can transform the input matrix into an equivalent diagonal matrix. Then, the computation of a Smith form of a diagonal matrix is a much simpler problem.

The complexity of modular Smith form algorithms was gradually improved by other authors by reducing the problem to matrix multiplication. In 1991 Hafner and McCurly proposed a modular algorithm based on repeated triangulation steps, see [62]. A randomized Las Vegas version of this algorithm was proposed by Giesbrecht in [53, Sec.1]. In the same article (see [53, Sec. 2]) the author proposed a black-box approach, which for dense matrices leads a $O(n^{\omega}m)$ complexity Monte Carlo algorithm. In 1996 Storjohann improved this by a factor of n and achieved an $O^{\sim}(n^{\omega-1}m)$ complexity algorithm by repeated reduction of banded matrix.

For sparse matrices, the black-box approach of [53] (see also [54, 55] for detailed presentation) allows to omit the problem of coefficient swell and matrix filling and leads to $O(m^2\Omega)$ complexity Monte Carlo algorithm.

A mixture of modular elimination and black-box approach is applied in [38, 39, 37, 31] to design a Smith form algorithm. First, the valence is computed by black-box methods, which allows to determine primes, for which local Smith form is non trivial. Then, local Smith forms by elimination mod p^l are computed and integer Smith form is computed as the product. The method was thoroughly experimentally tested for homology matrices, see [38, 39, 37].

A totally different approach, which is due to Eberly, Giesbrecht and Villard is presented in [44]. In Ch. 3 we presented the idea of the algorithm and its modifications. The ideas of [44] coupled with practical improvements of [112] lead to a successful adaptive algorithm, which is optimized for dense matrices. Moreover, the algorithm can easily be adapted to the sparse case, by the use of sparse solver of [43]. Yet it seems that the algorithm is to some extend far from the original motivations of the Smith form computation, where often structured (sparse, banded) matrices are given on input.

15.2.3 Algorithms in Spacial Cases

The computation of the Smith form of a diagonal matrix is a slightly easier problem, which reduces to several extended gcd computation, see [69]. An effective algorithm which uses sorting networks has been proposed by [127, Sec. 5.7.1]. For a square non-singular diagonal integer matrix A, the complexity of the algorithm is $O(\log(|\det(A)|))$, see [127, Thm. 5.20].

Over a field, the computation of the Smith form reduces to rank computation. For a matrix of rank r the Smith form consist of r invariant factors equal to 1 and some remaining zeros.

Over Z_{p^l}, an elimination based algorithm for local Smith form computation, see [38, 39, Alg. LRE] leads to effective Smith form computation in $O(rmn)$ modulo p^l operations, see [38, Thm. 4] for a $n \times m$ matrix of rank r.

15.3 Background on Homology Theory

In this section we present some background on the homology theory and the basic ideas for the computation of homologies. The ideas in this section are based on [72, Ch. 2]. For other presentations see e.g. [99, 63]. Let us start with some definitions.

Definition 15.3.1 (Exact Sequence) Let $n \in \mathbb{N} \cup \{\infty\}$ and let $\{\mathcal{C}_k\}_{k=0..n}$ be a sequence of modules. Let $\partial_k : \mathcal{C}_k \to \mathcal{C}_{k-1}$, $k = 0..n$ be a sequence of morphisms between modules. We say that the sequence $\{\partial_k\}$ is **exact** iff

$$\forall k = 1..n : \quad \partial_k \circ \partial_{k+1} = 0. \tag{15.1}$$

Definition 15.3.2 (Chain Complex) Let $n \in \mathbb{N} \cup \{\infty\}$ and let R be an abelian ring. Let $\{\mathcal{C}_k\}_{k=0..n}$ be modules over R and let $\partial_k : \mathcal{C}_k \to \mathcal{C}_{k-1}$,$k = 1..n$ be an exact sequence of homomorphisms between modules. Then $(\mathcal{C}, \partial) = (\mathcal{C}_k, \partial_{k+1})_{k=0..n}$ is a **chain complex (over R)**.

Given a chain complex, we define the groups of boundaries and cycles and the groups of homology as follows.

Definition 15.3.3 (Boundaries and Cycles) Let $n \in \mathbb{N} \cup \{\infty\}$ and let R be an abelian ring. Let $(\mathcal{C}, \partial) = (\mathcal{C}_k, \partial_{k+1})_{k=0..n}$ be a chain complex (over R).

The group of **cycles** $\mathbf{Z}_k \subset \mathcal{C}_k$ is defined by

$$\begin{aligned} \mathbf{Z}_0 &= \mathcal{C}_0, \\ \mathbf{Z}_k &= \ker(\partial_k), \quad k = 1..n. \end{aligned} \tag{15.2}$$

The group of **boundaries** $\mathbf{B}_k \subset \mathcal{C}_k$ is defined by

$$\mathbf{B}_k = \mathcal{I}(\partial_{k+1}), \quad k = 0..n-1, \tag{15.3}$$
$$\mathbf{B}_n = \{0\} \tag{15.4}$$

We have the following proposition.

Proposition 15.3.4 (Prop. 2.39 of [72]) Let $n \in \mathbb{N} \cup \{\infty\}$ and let R be an abelian ring. Let $(\mathcal{C}, \partial) = (\mathcal{C}_k, \partial_{k+1})_{k=0..n}$ be a chain complex (over R). Let $\{\mathbf{Z}_k\}_{k=0..n}$ be the groups of cycles and $\{\mathbf{B}_k\}_{k=0..n}$ be the groups of boundaries of $(\mathcal{C}, \partial)$. Then

$$\mathbf{B}_k \subset \mathbf{Z}_k, \quad k = 0..n$$

PROOF Let $b \in \mathbf{B}_k$. By Eq. (15.3) this means that there exists $a \in \mathcal{C}_{k+1}$ such that $\partial_{k+1}(a) = b$. Then, by (15.1) $\partial_k b = \partial_k(\partial_{k+1} a) = 0$. Thus, by Eq. (15.2) $b \in \mathbf{Z}_k$.

Now, we may define the groups of homologies for $(\mathcal{C}, \partial)$.

Definition 15.3.5 (Groups of Homology) Let $n \in \mathbb{N} \cup \{\infty\}$ and let R be an abelian ring. Let $(\mathcal{C}, \partial) = (\mathcal{C}_k, \partial_{k+1})_{k=0..n}$ be a chain complex (over R). Let $\{\mathbf{Z}_k\}_{k=0..n}$ be the groups of cycles and $\{\mathbf{B}_k\}_{k=0..n}$ be the groups of boundaries of $(\mathcal{C}, \partial)$.

The groups of **homology** of $(\mathcal{C}, \partial)$ (over R) are define by

$$\mathbf{H}_k = \mathbf{Z}_k / \mathbf{B}_k, \quad k = 0..n. \tag{15.5}$$

PROOF By Prop. 15.3.4, the groups of homology are well defined.

15.3.1 Computation of Homologies

Let $n \in \mathbb{N} \cup \{\infty\}$ and let R be an abelian ring. Let $(\mathcal{C}, \partial) = (\mathcal{C}_k, \partial_{k+1})_{k=0..n}$ be a chain complex (over R). Let $\{\mathbf{Z}_k\}_{k=0..n}$ be the groups of cycles and $\{\mathbf{B}_k\}_{k=0..n}$ be the groups of boundaries of $(\mathcal{C}, \partial)$. To compute the quotient group $\mathbf{H}_k = \mathbf{Z}_k / \mathbf{B}_k$ the groups of cycles $\mathbf{Z}_k$ and boundaries $\mathbf{B}_k$ have to be determined. In the case when $\mathcal{C}_k$ is a free module over a number ring R, the task can be solved algorithmically.

In fact, by universal coefficient homology theorem, see [63, Thm. 3A.3], the computation of homologies over $\mathbb{Z}$ allows to obtain the result for other rings as well. Therefore, the computation of homologies over $\mathbb{Z}$ is of particular interest.

In the case when R is a PID, e.g. R is: integer ring $\mathbb{Z}$,, rational field $\mathbb{Q}$,, finite field $\mathbb{Z}_p$ for a prime p, the bases of $\mathcal{C}_k, \mathbf{Z}_k, \mathbf{B}_k$ are well defined and $\mathbf{H}_k$ can be deduced from comparison of bases.

In the case when R is a finite ring $\mathbb{Z}_q$, for $q \in \mathbb{N}, q > 1$, or, more generally, an abelian PIR, the concept generating sets can be used. See [58, Lem. 8.4.1], [58, Lem. 8.4.4] for theoretical background.

As $\mathcal{C}_k$ is a finitely generated free module, an exact chain of matrices M_k can be constructed to represent ∂_k. In the next subsection we shortly present the notations for matrices generated by boundary operators. Then we show how the Smith form of M_k can be used to compute the generating sets of $\mathbf{Z}_k, \mathbf{B}_k$ and thus, homologies $\mathbf{H}_k$.

Matrix

Matrices can represent homomorphisms between finitely generated modules over abelian rings. This is based on the following theorem.

Theorem 15.3.6 (Lem. 8.4.1 of [58]) *Let R be an abelian ring with 1 and let $m \in \mathbb{N}, m > 0$. Every basis of the finitely generated free module R^m has cardinality m.*

Let us introduce the following notation. See [72, Def. 2.21, Prop. 2.22]

Definition 15.3.7 (Def. 2.21 of [72]) Let R be an abelian ring with 1. Let $k \in \mathbb{N}, k > 0$ and let R^k be a finitely generated module over R. Let $E = \{e_i\}_{i=1..k}$ be the basis of R^k and let $a, b \in R^k$. Assume that $a = \sum_{i=1}^{k} a_i e_i$ and $b = \sum_{i=1}^{k} b_i e_i$, where $a_i, b_i \in R, i = 1..k$. Then the 'scalar' product $< a, b >$ is defined as

$$< a, b >_E =^{df} \sum_{i=1}^{k} a_i b_i.$$

The 'scalar' product defined in Def. 15.3.7 is bilinear. Now, we may define a matrix.

Definition 15.3.8 (Matrix of Homomorphism) Let R be an abelian ring with 1. Let $n, m \in \mathbb{N}, n, m > 0$. Let R^m and R^n be two finitely generated modules and let $U = \{u_j\}_{j=1..m}$ and $V = \{v_i\}_{i=1..n}$ be fixed bases for R^m and R^n respectively. Let $f : R^m \to R^n$ be a homomorphism.

A $n \times m$ (m columns, n rows) matrix $A = [a_{ij}]_{i=1..n, j=1..m}$ over the ring R represents a homomorphism f between two finitely generated free modules R^m and R^n in fixed bases U and V if and only if a_{ij} is equal to the ith coefficient of $f(u_j)$ in basis V i.e.

$$a_{ij} =< f(u_j), v_i >_V .$$

Matrix-vector product represents applying f_A to the vector in the same basis. The sum and the product of matrices is defined in the classic way. Usually, a matrix represents the homomorphism in canonical basis $\{e_i\} = [0 \dots 01_i 0 \dots 0]^T$. In order to represent a homomorphism by a matrix in another bases we may use the following proposition.

Proposition 15.3.9 (Sec. 3.1 of [72]) Let R be an abelian ring with 1. Let $n, m \in \mathbb{N}, n, m > 0$. Let R^m and R^n be two finitely generated modules and let $U = \{u_j\}_{j=1..m}$ and $V = \{v_i\}_{i=1..n}$ be fixed bases for R^m and R^n respectively. Let $f : R^m \to R^n$ be a homomorphism.

Let $A = [a_{ij}]_{i=1..n,j=1..m}$ be a $n \times m$ matrix that represents homomorphism f in canonical bases $\{e_j^{(m)}\}$ and $\{e_i^{(n)}\}$ of R^m and R^n respectively. Let $B = [b_{ij}]_{i=1..n,j=1..m}$ be a $n \times m$ matrix that represents f in bases U and V. Let u_{lj}, v_{ki} be the l and kth coefficient of u_j and v_i respectively in the canonical bases, $l, j = 1..m, k, i = 1..n$. Let us define matrices $\tilde{U} = [u_{lj}]_{l,j=1..m}$ and $\tilde{V} = [v_{ki}]_{k,i=1..n}$. Then

$$A\tilde{U} = \tilde{V}B.$$

Matrix of boundary operator

Let R be a number ring. Let $n \in \mathbb{N} \cup \{\infty\}$ and $(\mathcal{C}, \partial) = (\mathcal{C}_k, \partial_{k+1})_{k=0..n}$ be a chain complex over R. We assume that $\mathcal{C}_k$, $k = 0..n$ are free finitely generated modules over R.

As ∂_k is a homomorphism between finitely generated free modules, by Def. 15.3.8 it can be represented by matrix M_k. Since $\{\partial_k\}$ is an exact sequence, so is $\{M_k\}$. The groups of cycles and boundaries, $\mathbf{Z}_k$ and $\mathbf{B}_k$ are submodules of $\mathcal{C}_k$. To compute the quotient group $\mathbf{H}_k = \mathbf{Z}_k/\mathbf{B}_k$, the corresponding generating set for $\mathbf{Z}_k$ and $\mathbf{B}_k$ can be found. A way to compute the generating set for $\mathbf{B}_k = \mathcal{I}(\partial_{k+1})$ is to compute the Smith form of the corresponding matrix M_{k+1}. The practical problem of generation of matrices of the boundary operator from the (simplicial) complexes is addressed in [37].

The following propositions give the formula for image and kernel of matrix A from its Smith form S.

Proposition 15.3.10 (Image of A) Let R be an abelian PIR with 1 and let $l, n, m \in \mathbb{N}, n, m > 1, l \leq \min(n, m)$. Let A be a $n \times m$ matrix over R. Let $S = \mathrm{diag}(s_1, \ldots, s_l, 0 \ldots, 0)$ be a diagonal matrix, $s_i \in R, i = 1..l$. Let U, V be a $n \times n$ and $m \times m$ invertible matrices over R, such that $A = USV$. Then the image $\mathcal{I}(A)$ of A is equal to

$$\mathcal{I}(A) = \mathrm{span}(s_1 u_1, \ldots, s_l u_l),$$

where $u_i, i = 1..l$ is the lth column of U.

PROOF See e.g. [72, Prop. 3.9]. By Prop. 15.3.9, S represents the same homomorphism as A, in bases U^{-1}, V^{-1}. Hence, the formula for $\mathcal{I}(A)$.

Proposition 15.3.11 (Kernel of A) Let R be an abelian PIR with 1 and let $l, n, m \in \mathbb{N}, n, m > 1, l \leq \min(n, m)$. Let A be a $n \times m$ matrix over R. Let $S = \mathrm{diag}(s_1, \ldots, s_l, 0 \ldots, 0)$ be a diagonal matrix, $s_i \in R, i = 1..l$. Let U, V be a $n \times n$ and $m \times m$ invertible matrices over R, such that $A = USV$. Moreover, let us assume, that s_i, $i = 1..l$ are not divisors of 0. Then the kernel $\ker(A)$ of A is equal to

$$\ker(A) = \mathrm{span}(V^{-1}e_{l+1}, \ldots, V^{-1}e_m) \simeq R^{m-l},$$

where $u_i, i = 1..l$ is the lth column of U.

PROOF See e.g. [72, Prop. 3.9]. By Prop. 15.3.9, S represents the same homomorphism as A, in bases U^{-1}, V^{-1}. Hence, the formula for $\ker(A)$. As V is invertible, $\ker(A)$ is isomorphic to R^{m-l}.

The following corollary gives the formula for homologies over $\mathbb{Z}$.

Corollary 15.3.12 (Homologies over $\mathbb{Z}$) Let $n \in \mathbb{N} \cup \{\infty\}$ and let $(\mathcal{C}, \partial) = (\mathcal{C}_k, \partial_{k+1}, k = 0..n)$ be a chain complex over $\mathbb{Z}$. Let $\{\mathbf{Z}_k\}_{k=0..n}$ be the groups of cycles and $\{\mathbf{B}_k\}_{k=0..n}$ be the groups of boundaries of $(\mathcal{C}, \partial)$. Let n_k denote the dimension of $\mathcal{C}_k$ and let M_k denote the $n_{k-1} \times n_k$ matrix of homomorphism ∂_k, $k = 1..n$ in the canonical bases. Let $l_k \in \mathbb{N}$ and $S_k = \mathrm{diag}(s_1^{(k)}, \ldots, s_{l_k}^{(k)}, 0, \ldots, 0)$, $s_i^{(k)} \neq 0$ for $i = 1..l_k$, be the Smith form of M_k. Then for $k = 1..n-1$, $n_k - l_k - l_{k+1} \geq 0$ and the kth homology group is equal to

$$\mathbf{H}_k \simeq \mathbb{Z}^{n_k - l_k - l_{k+1}} \oplus \mathbb{Z}_{s_1^{(k+1)}} \oplus \cdots \oplus \mathbb{Z}_{s_{l_{k+1}}^{(k+1)}} .$$

Additionally, $n_0 - l_1 \geq 0, n_n - l_n \geq 0$ and

$$\begin{aligned} \mathbf{H}_0 &\simeq \mathbb{Z}^{n_0 - l_1} \oplus \mathbb{Z}_{s_1^{(1)}} \oplus \cdots \oplus \mathbb{Z}_{s_{l_1}^{(1)}} \\ \mathbf{H}_n &\simeq \mathbb{Z}^{n_n - l_n} . \end{aligned}$$

PROOF The computation follows from Prop. 15.3.10,15.3.11.

16

Reduction of a Chain Complex

In [73], Kaczyński, Mrozek, and Ślusarek propose a procedure to reduce chain complexes by *algebraic reductions*. The idea is based on the following theorem.

16.1 Algebraic Reductions of [73]

Theorem 16.1.1 (One-step Reduction of Sec. 2 in [73]) *Let R be an abelian ring and let $n \in \mathbb{N} \cup \{\infty\}$. Let $(\mathcal{C}, \partial) = (\mathcal{C}_k, \partial_k)_{k=0..n}$ be a chain complex over R such that $\mathcal{C}_k$ are finitely generated modules over R, $k = 0..n$. Assume that bases E_k are given for each $\mathcal{C}_k$ and let $< -, \cdot >$ denote the corresponding 'scalar' product (cf. Def. 15.3.7). Let us fix $m \in \{1..n\}$. Assume that $a \in E_{m-1}$ and $b \in E_m$ are two elements such that there exists $\lambda \in R, r \in \mathcal{C}_{m-1}$, such that*

$$\partial_m b = \lambda a + r \tag{16.1}$$

and λ is invertible in R, $< a, r >= 0$. Define for $k = 0..n$

$$\overline{\mathcal{C}}_k = \begin{cases} \mathcal{C}_k & \text{if } k \neq m, m-1 \\ \{v \in \mathcal{C}_k :< a, v >= 0\} & \text{if } k = m-1 \\ \{v \in \mathcal{C}_k :< b, v >= 0\} & \text{if } k = m \end{cases} .$$

For $v \in \overline{\mathcal{C}}_k$ let us define

$$\overline{\partial}_k = \begin{cases} \partial_k(v) & \text{if } k \neq m+1, m \\ \partial_k(v) - \lambda^{-1} < a, \partial_k v > \partial b & \text{if } k = m \\ \partial_k(v) - < \partial_k v, b > b & \text{if } k = m+1 \end{cases} . \tag{16.2}$$

Then the homologies groups of $(\mathcal{C}, \partial)$ are equal to homologies groups of $(\overline{\mathcal{C}}, \overline{\partial})$.

PROOF See Sec. 2 of [73] and [73, Thm. 2] in particular. The proof of the theorem follows from the inclusion theorem.

16.1.1 Geometric Interpretation of Reduction

Thm. 16.1.1 has an intuitive geometric interpretation. Several examples are detailed in [73, Sec. 3]. Here, let us present the concept of free-faces reduction (or exterior face collapsing, or elementary collapse), see also [72, Sec. 2.4].

Definition 16.1.2 (Free Face) Let R be an abelian ring and let $n \in \mathbb{N} \cup \{\infty\}$. Let $(\mathcal{C}, \partial) = (\mathcal{C}_k, \partial_k)_{k=0..n}$ be a chain complex over R such that $\mathcal{C}_k$ are finitely generated modules over R, $k = 0..n$. Assume that bases E_k are given for each $\mathcal{C}_k$. Let us fix $m \in \{1..n\}$ and let $a \in E_{m-1}$. We say that a is a free face, if there exists exactly one $b \in E_m$ (we call it a cell) such that $< a, \partial_m b > \neq 0$.

Remark 16.1.3 Compare with the concept of exterior face in [73, Sec. 4] in the case of simplicial complexes and with [72, Def. 2.60] in the case of cubical sets.

Suppose now that $a \in \mathcal{C}_{m-1}$ is a free face and that $S \in \mathcal{C}_m$ is the corresponding cell such that $< a, \partial_k S >= \lambda \neq 0$. Moreover, let λ be invertible in R. Notice, that theorem 16.1.1 can be applied to the pair (a, S).

Let us illustrate the reduction by Fig. 16.1. It is immediately visible that the U figure on the right is a retract of the square on the left. Hance, homologies of both figures are the same.

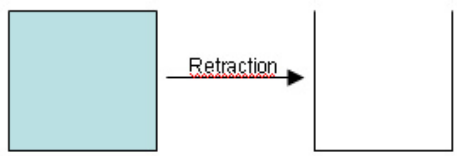

Figure 16.1: Retraction of a square

Proposition 16.1.4 (Thm. 16.1.1 for Free Faces) Let a be a free face and b the corresponding cell. Let us assume that $\lambda =< a, \partial b >$ is invertible. Reduction of (a, b) based on Thm. 16.1.1 does not require any modification of ∂ i.e. $\overline{\partial}$ is a restriction of ∂ to a smaller complex.

PROOF Let us assume that $a \in E_{m-1}$ has dimension $m - 1$. By Eq. (16.2), ∂_m is modified in the case when there exists $v \neq b$ such that $< \partial_m v > \neq 0$. By the definition of a free face this is not possible. On the other hand, ∂_{m+1} is modified if there exist v such that $< \partial v, b > \neq 0$. Assume for a moment that such v exists and $< \partial v, b >= \mu$. Then, $< \partial\partial v, a >=< \mu\partial b, a >= \mu\lambda$, which is non-zero as λ is invertible. But $\partial\partial v = 0$ from the definition of boundary operator. Thus, no such v exists and ∂_{m+1} is not modified.

Remark 16.1.5 In some simple cases, reduction of free faces is often enough to reduce the complex to a small size. On the other hand, examples of complexes for which no free faces exists can be found, e.g. Bing's house.

16.2 Reduction Algorithm for Matrices

In this section, we will discuss matrix representation of algebraic reductions and its application to Smith form and rank computation for an exact sequence of matrices.

Let us consider the notions of Thm. 16.1.1. Let M_k be the matrix of homomorphism ∂_k. Let $e_i^{(k)} \in E_k$ be the elements of the basis. Then we may consider $e_i^{(k)}$ as labels for columns of M_k and $e_j^{(k-1)}$ as labels for rows of M_k. Let us consider Eq. (16.2). Taking into account Eq. (16.1), one may observe that it gives the formula for Gaussian elimination of M_k, which leads to the elimination of the column of M_k labelled by b. Indeed, we have the following theorem.

Definition 16.2.1 (Admissible Entry) Let $m, n \in \mathbb{N}$, let R be a PIR and let $M = [a_{ij}]_{i=1..n, j=1..m}$ be a $n \times m$ matrix over R. We say that an entry a_{ij} is admissible if a_{ij} is not a zero divisor and

$$a_{ij} \mid a_{it} \quad \forall t = 1..m_k \tag{16.3}$$

$$a_{ij} \mid a_{sj} \quad \forall s = 1..n_k. \tag{16.4}$$

Theorem 16.2.2 (One-step Reduction of an Exact Sequence of Matrices) *Let R be a PIR and let $n \in \mathbb{N} \cup \{\infty\}$. Let M_k, $k = 1..N$ be an exact sequence of matrices over R. Let us take $k = 1..N$. Let us assume that $M_k = [a_{st}]$ is a $n_k \times m_k$ matrix and let us take $i \in \{1..n_k\}, j \in \{1..m_k\}$. Suppose that a_{ij} is an admissible entry as in Def. 16.2.1. Then there exist a matrix L such that the jth column of LM_k has only one non-zero entry in the ith row, equal to a_{ij}. Let M'_k be the minor of LM_k obtained by deleting the row and column corresponding to a_{ij}. Then*

$$\begin{aligned} &SF(\mathrm{diag}(a_{ij}, SF(M'_k))) = SF(LM_k) = SF(M_k) \\ &\mathrm{rank}(M'_k) + 1 = \mathrm{rank}(LM_k) = \mathrm{rank}(M_k). \end{aligned} \tag{16.5}$$

Moreover, if $k > 1$, let M'_{k-1} denote the minor of M_{k-1} obtained by deleting the ith column and, if $k < N$, let M'_{k+1} denote the minor of M_{k+1} obtained by deleting the jth row. Then $\ldots M_{k-2} M'_{k-1} M'_k M'_{k+1} M_{k+2} \ldots$ is an exact sequence of matrices. Moreover,

$$\begin{aligned} &[SF(M'_{k-1}), 0] = SF(M_{k-1}) \\ &\mathrm{rank}(M'_{k-1}) = \mathrm{rank}(M_{k-1}) \end{aligned} \tag{16.6}$$

and

$$\begin{aligned} &\begin{bmatrix} SF(M'_{k+1}) \\ 0 \end{bmatrix} = SF(M_{k+1}) \\ &\mathrm{rank}(M'_{k+1}) = \mathrm{rank}(M_{k+1}). \end{aligned} \tag{16.7}$$

PROOF Thanks to Eq. (16.3) there exists matrix L, given as a product row operations i.e.

$$L = \prod_{t \neq i} L_t, \quad L_t = \mathrm{Id} + \begin{array}{c} \begin{array}{ccc} & j & \end{array} \\ \begin{bmatrix} 0 \dots & 0 & \dots 0 \\ & \vdots & \\ 0 & -a_{it}/a_{ij} & 0 \\ & \vdots & \\ 0 \dots & 0 & \dots 0 \end{bmatrix} \end{array} \begin{array}{c} \\ \\ t \\ \\ \\ \end{array}$$

$$L = \begin{array}{c} \begin{array}{ccccccc} & & & j & & & \end{array} \\ \begin{bmatrix} 1 & & & -a_{i1}/a_{ij} & & & \\ & \ddots & & \vdots & & & \\ & & 1 & -a_{i,i-1}/a_{ij} & & & \\ & & & 1 & & & \\ & & & -a_{i,i+1}/a_{ij} & 1 & & \\ & & & \vdots & & \ddots & \\ & & & -a_{in_k}/a_{ij} & & & 1 \end{bmatrix} \end{array}, L^{-1} = \begin{array}{c} \begin{array}{ccccccc} & & & j & & & \end{array} \\ \begin{bmatrix} 1 & & & a_{i1}/a_{ij} & & & \\ & \ddots & & \vdots & & & \\ & & 1 & a_{i,i-1}/a_{ij} & & & \\ & & & 1 & & & \\ & & & a_{i,i+1}/a_{ij} & 1 & & \\ & & & \vdots & & \ddots & \\ & & & a_{in_k}/a_{ij} & & & 1 \end{bmatrix} \end{array}. \tag{16.8}$$

Matrix L is invertible over R and $\det(L) = 1$. Thus, $SF(LM_i) = SF(M_i)$ and $\mathrm{rank}(LM_i) = \mathrm{rank}(M_i)$. Moreover, thanks to Eq. (16.4) there exist a matrix U, given as a product of column operations, such that

$$LM_iU = \begin{bmatrix} A & 0 & A' \\ 0 & a_{ij} & 0 \\ B & 0 & B' \end{bmatrix},$$

$$U = \prod_{s \neq j} U_s, \quad U_s = \mathrm{Id} + \begin{array}{c} \begin{array}{ccc} & s & \end{array} \\ \begin{bmatrix} 0 \dots & 0 & \dots 0 \\ & \vdots & \\ 0 & -a_{sj}/a_{ij} & 0 \\ & \vdots & \\ 0 \dots & 0 & \dots 0 \end{bmatrix} \end{array} \begin{array}{c} \\ \\ i \\ \\ \\ \end{array}$$

According to the definition, matrix $\begin{bmatrix} A & A' \\ B & B' \end{bmatrix}$ is equal to M'_k. Thus, $SF(\mathrm{diag}(a_{ij}, SF(M'_k))) = SF(M_k)$ and $\mathrm{rank}(M'_k) + 1 = \mathrm{rank}(M_k)$.

Let us now consider the case when $k > 1$ and let us consider matrix M'_{k-1} delivered from M_{k-1}. We have that

$$M_{k-1}M_k = 0 \Rightarrow (M_{k-1}L^{-1})(LM_k) = 0.$$

Thus, in particular,

$$M_{k-1}L^{-1}\begin{bmatrix}0 & \dots & a_{ij} & \dots & 0\end{bmatrix}^T = 0,$$

i.e. the ith column of $M_{k-1}L^{-1}$ fulfills

$$a_{ij}(M_{k-1}L^{-1})[1..n_{k-1}, i] = 0.$$

This implies that $(M_{k-1}L^{-1})[1..n_{k-1}, i] = 0$, since a_{ij} is not a zero divisor. By the formula for L^{-1} in Eq. (16.8), the ith column of M_{k-1} is a combination of the remaining columns. Moreover, multiplying by L^{-1} does not change any other column, thus $M_{k-1}L[1..n_{k-1}][1..\hat{i}..m_{k-1}] = M'_{k-1}$ and $M'_{k-1}M'_k = 0$ and Eq. (16.6) holds.

Let us now consider the case when $k < N$. We have

$$M_kM_{k+1} \Rightarrow LM_kM_{k+1} = 0,$$

which after a suitable permutation implies that

$$\begin{bmatrix} M'_k & 0 \\ a & a_{ij} \end{bmatrix} \cdot \begin{bmatrix} M'_{k+1} \\ M_{k+1}[j, 1..m_{k+1}] \end{bmatrix} = 0.$$

Thus, $M'_k \cdot M'_{k+1} = 0$ and $a \cdot M'_{k+1} + a_{ij}M_{k+1}[j, 1..m_{k+1}] = 0$. By Eq. (16.4), a_{ij} divides a and thus the jth row of M_{k+1} is a 0 combination of the remaining rows. Hence, Eq. (16.7) holds.

Remark 16.2.3 The theorem can be applied in the case when a_{ij} is invertible but it is not restricted to this case.

As a corollary, let us now present a column version of the theorem.

Corollary 16.2.4 [One-step Reduction of an Exact Sequence of Matrices II] Let R be a PIR and let $n \in \mathbb{N} \cup \{\infty\}$. Let M_k, $k = 1..N$ be an exact sequence of matrices over R. Let us take $k = 1..N$. Let us assume that $M_k = [a_{st}]$ is a $n_k \times m_k$ matrix and let us take $i \in \{1..n_k\}, j \in \{1..m_k\}$. Suppose that a_{ij} is an admissible entry as in Def. 16.2.1. Then there exist a matrix U such that the ith row of M_kU has only one non-zero entry in the jth column, equal to a_{ij}. Let M'_k be the minor of M_kU obtained by deleting the row and column corresponding to a_{ij}. Then

$$\begin{aligned} &SF(\mathrm{diag}(a_{ij}, SF(M'_k))) = SF(LM_k) = SF(M_k) \\ &\mathrm{rank}(M'_k) + 1 = \mathrm{rank}(LM_k) = \mathrm{rank}(M_k). \end{aligned} \tag{16.9}$$

Moreover, if $k > 1$ let M'_{k-1} denote the minor of M_{k-1} obtained by deleting the ith column and if $k < N$ let M'_{k+1} denote the minor of M_{k+1} obtained by deleting the jth row. Then $\dots M_{k-2}M'_{k-1}M'_kM'_{k+1}M_{k+2}\dots$ is an exact sequence of matrices. Moreover,

$$\begin{aligned} &[SF(M'_{k-1}), 0] = SF(M_{k-1}) \\ &\mathrm{rank}(M'_{k-1})\,\mathrm{rank}(M_{k-1}) \end{aligned} \tag{16.10}$$

and

$$\begin{bmatrix} SF(M'_{k+1}) \\ 0 \end{bmatrix} = SF(M_{k+1})$$
$$\operatorname{rank}(M'_{k+1}) \operatorname{rank}(M_{k+1}). \tag{16.11}$$

PROOF Chain M_k^T, $k = N..1$ fulfills Theorem 16.2.2 so the corollary follows as a natural consequences.

Remark 16.2.5 We refer to the latter case as coreductions.

As a result of reductions and coreductions in Thm. 16.2.2 and Cor. 16.2.4, a new exact sequence of matrices is produces, for which reductions can be applied again. For this, we require that reduction is induced by an admissible entry, which by definition, is not a divisor of zero. In Sec. 16.2.1 we show that this condition is necessary to ensure that Thm. 16.2.2 and Cor. 16.2.4 could be applied again to the resulting sequence. In Thm. 16.2.8 we show that the requirement can be omitted if a stronger condition on the sequence is imposed.

16.2.1 Reductions modulo p^l

To avoid coefficient swell, many authors propose turning to modular algorithms. This is the case of [69, 62, 121]. Also, elimination modulo p^l is an efficient way to compute local Smith form and helps compute the Smith form, see [38, 39]. Let us thus focus on the results we may have in the case where the ring R is equal to $\mathbb{Z}_q$ for a composite q, where λ is a divisor of 0. We have been able to delivered both negative and positive results.

Example 16.2.6 Let us consider matrices

$$A = \begin{bmatrix} 2 & 2 & -2 \\ 6 & 3 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \tag{16.12}$$

over $\mathbb{Z}$. The product AB is equal to 0 over $\mathbb{Z}$. Let us now consider the process of reduction modulo 6 of A. The entry a_{11} fulfills all properties of Def. 16.2.1 apart form the fact that it is a zero divisor. Let us see what happens if we try to perform a reduction according to Thm. 16.2.2 using a_{11}. As $a_{21} = 6 = 0 \mod 6$ the process of reduction consists of deleting the first row and column of A as well as the first row of B. We obtain

$$A' = \begin{bmatrix} 3 & 0 \end{bmatrix}, \quad B' = \begin{bmatrix} 2 \\ 3 \end{bmatrix}.$$

Notice that $A'B' = 0 \mod 6$ but $A'B' \neq 0$ over $\mathbb{Z}$. Moreover, equations (16.5) and (16.7) hold, as long as the Smith form is concerned. Indeed, a quick revision of the proof of Thm.

16.2.2 shows, that for an exact chain of matrices over $\mathbb{Z}$, one can take modular images and perform one step of reduction. Yet, repeating the reduction may lead to a mistake, as we will now show.

Let us continue the reduction of A and delete element $a'_{11} = 3$. After reduction A' becomes void, and the Smith form of A is correctly determined modulo 6 as $SF(A) = SF(\mathrm{diag}(2,3)) = \mathrm{diag}(1,6) = \mathrm{diag}(1,0) \mod 6$. Also, the first row of B' is deleted and we are left with a one-entry matrix $B'' = [3]$. However, the conclusion $SF(B') = \mathrm{diag}(SF(B''),0) = \mathrm{diag}(3,0)$ is false in this case.

Notice that the elimination of A followed the scheme of the algorithm of Iliopoulos [69] modulo $\mathrm{lcm}(d_2(A), d_1(B))$, where d_i is the gcd of all $i \times i$ minors. Repeated reductions with admissible entries being divisors of zero lead to uncorrect results in this case.

Example 16.2.7 Let us consider matrices

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix} \tag{16.13}$$

over $\mathbb{Z}_4$ but $AB = \begin{bmatrix} 4 & 0 \\ 0 & 0 \end{bmatrix}$ over $\mathbb{Z}$. The product $AB = 0 \mod 4$. The entry a_{11} fulfills all properties of Def. 16.2.1 apart form the fact that it is a zero divisor. Let us perform a reduction according to Thm. 16.2.2 using a_{11}. The process of reduction consists of deleting the first row and column of A as well as the first row of B. After reduction we obtain a matrix $A' = [1]$, such that $SF(A) = SF(\mathrm{diag}(2, A'))$, and matrix $B' = [0]$ such that $SF(B) \neq [SF(B'), 0]^T$. Reduction modulo a divisor of zero leads to uncorrect results in this case.

Despite these negative results, the following theorem allows to apply reductions in the case of local Smith form computation.

Theorem 16.2.8 (One-step Reduction Modulo p^l) *Let p be a prime and $l \in \mathbb{Z}, l > 1$. Let $R = \mathbb{Z}_{p^l}$ and let $n \in \mathbb{N} \cup \{\infty\}$. Let M_k, $k = 1..N$ be an exact sequence of matrices over $\mathbb{Z}_{p^l}$.Let us take $k = 1..N$ and let us additionally assume that*

1. *if $k > 1$ then there exist integer matrix Z, matrix X over $\mathbb{Z}_{p^l}$, such that $M_{k-1}M_k = p^l Z$ over $\mathbb{Z}$ and $Z = M_{k-1}X \mod p^l$,*
2. *if $k < N$ then there exist integer matrix W, matrix Y over $\mathbb{Z}_{p^l}$, such that $M_k M_{k+1} = p^l W$ over $\mathbb{Z}$ and $W = Y M_{k+1} \mod p^l$.*

Let us assume that $M_k = [a_{st}]$ is a $n_k \times m_k$ matrix and let us take $i \in \{1..n_k\}, j \in \{1..m_k\}$. Suppose that a_{ij} fulfills Eq. (16.3) and (16.4) in $\mathbb{Z}_{p^l}$[1].

[1]Notice, that a_{ij} can be a zero divisor

Then there exist an integer matrix L such that the jth column of LM_k has only one non-zero entry modulo p^l in the ith row, equal to a_{ij}. Let M'_k be the minor of LM_k obtained by deleting the row and column corresponding to a_{ij}. We have

$$SF(\mathrm{diag}(a_{ij}, SF(M'_k))) = SF(LM_k) = SF(M_k), \tag{16.14}$$

where SF denote the Smith form modulo Z_{p^l}.

Moreover, if $k > 1$ let M'_{k-1} denote the minor of M_{k-1} obtained by deleting the ith column and if $k < N$ let M'_{k+1} denote the minor of M_{k+1} obtained by deleting the jth row. Then $\ldots M'_{k-1}(M'_k \mod p^l)M'_{k+1}\ldots$ is an exact sequence of matrices over $\mathbb{Z}_{p^l}$ such that conditions 1, 2 hold. Moreover,

$$[SF(M'_{k-1}), 0] = SF(M_{k-1}), \tag{16.15}$$

$$\begin{bmatrix} SF(M'_{k+1}) \\ 0 \end{bmatrix} = SF(M_{k+1}). \tag{16.16}$$

PROOF The proof is rather technical, though some ideas of the proof of Thm. 16.2.2 carry on. First, matrices L and U can be constructed in the same way as in the proof of Thm. 16.2.2 as products of elementary operations. Thus, matrices L, U, are integer matrices s.t. $\det(L) = \det(U) = 1$.

Let us now prove that we can delete the row and column in the neighboring matrices and still obtain an exact sequence of matrices such that conditions 1 and 2 hold. Notice, that unless explicitly mentioned otherwise, all matrices and operations are over $\mathbb{Z}$.

First, let us assume that $k > 1$. Without loss of generality we may assume that $i, j = 1$. By condition 1, over $\mathbb{Z}$ we have

$$M_{k-1}L^{-1}LM_kU = p^l ZU. \tag{16.17}$$

Also, by Eq. (16.8), L^{-1} is of a form

$$L^{-1} = \begin{bmatrix} 1 & & & \\ l_{21} & 1 & & \\ \vdots & & \ddots & \\ l_{n_k 1} & & & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \mathbf{l} & \mathrm{Id} \end{bmatrix}.$$

As $\det(L) = 1$, $SF(M_{k-1}L^{-1}) = SF(M_{k-1})$.

We can introduce the following block notations for matrices

$$M_{k-1} = \begin{bmatrix} \mathbf{c} & M'_{k-1} \end{bmatrix}, M_{k-1}L^{-1} = \begin{bmatrix} \mathbf{c} + M'_{k-1}\mathbf{l} & M'_{k-1} \end{bmatrix}.$$

Let $\alpha = \mathrm{ord}_p(a_{ij}$. We have $0 \le \alpha < l$. Let us also define integer vectors $\mathbf{w}', \mathbf{w}, \mathbf{v}$ and matrix $M''_k =_{p^l} M'_K$, such that

$$LM_k = \begin{bmatrix} a_{11} & p^\alpha \mathbf{w}'^T \\ p^l \mathbf{v} & M'_k \end{bmatrix}, \quad LM_kU = \begin{bmatrix} a_{11} & p^l \mathbf{w}^T \\ p^l \mathbf{v} & M''_k \end{bmatrix}, \tag{16.18}$$

Now, from Eq. (16.17) we can deduce that

$$a_{11}(\mathbf{c} + M'_{k-1}\mathbf{l}) + p^l M'_{k-1}\mathbf{v} = p^l ZU \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}^T = p^l \mathbf{z}, \tag{16.19}$$

where $ZU \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}^T = \mathbf{z}$. Let $=_{p^l}$ denote the equality modulo p^l. By Cond. 1

$$\mathbf{z} =_{p^l} M_{k-1}XU \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}^T =_{p^l} M_{k-1}\mathbf{x} = \begin{bmatrix} \mathbf{c} & M'_{k-1} \end{bmatrix} \begin{bmatrix} x & \mathbf{x}'^T \end{bmatrix}^T,$$

where

$$X = \begin{bmatrix} \mathbf{x} & \tilde{X} \end{bmatrix} = \begin{bmatrix} x & \tilde{\mathbf{x}} \\ \mathbf{x}' & \tilde{\tilde{X}} \end{bmatrix}. \tag{16.20}$$

Eq. (16.19) transforms to

$$\begin{aligned} a_{11}/p^\alpha \mathbf{c} &= -M'_{k-1}(a_{11}/p^\alpha \mathbf{l} + p^{l-\alpha}\mathbf{v}) + p^{l-\alpha}(x\mathbf{c} + M'_{k-1}\mathbf{x}') \\ (a_{11}/p^\alpha - p^{l-\alpha}x)\mathbf{c} &= M'_{k-1}\mathbf{r}, \end{aligned} \tag{16.21}$$

where $r = a_{11}/p^\alpha \mathbf{l} + p^{l-\alpha}\mathbf{v} + p^{l-\alpha}\mathbf{x}'$.

As $\gcd(a_{11}/p^\alpha - p^{l-\alpha}x, p) = 1$, $a_{11}/p^\alpha - p^{l-\alpha}x$ is invertible over $\mathbb{Z}_{p^l}$ and there exist vector $\mathbf{r}'$ over $\mathbb{Z}_{p^l}$ such that $\mathbf{c} =_{p^l} M'_{k-1}\mathbf{r}'$. Thus, $\mathbf{c}$ is a combination of the columns of M'_{k-1}, $SF(M_{k-1}) = [0, SF(M'_{k-1})]$ and Eq. (16.15) holds.

Now, we need to construct matrices Z', X', such that Cond. 1 holds. By Eq. (16.17)

$$p^l(\mathbf{c} + M'_{k-1}\mathbf{l})\mathbf{w}^T + M'_{k-1}M''_k = p^l \tilde{Z},$$

where $\tilde{Z}$ is composed of all but the first columns of ZU. Thus, we may put $Z'' = \tilde{Z} - (\mathbf{c} + M'_{k-1}\mathbf{l})\mathbf{w}^T$. Consider Eq. (16.20). By condition 1, modulo p^l we have,

$$Z'' =_{p^l} M_{k-1}\tilde{X} - (\mathbf{c} + M'_{k-1}\mathbf{l})\mathbf{w}^T =_{p^l} \mathbf{c}\tilde{\mathbf{x}} + M'_{k-1}\tilde{\tilde{X}} - (\mathbf{c} + M'_{k-1}\mathbf{l})\mathbf{w}^T,$$

As $\mathbf{c} =_{p^l} M'_{k-1}\mathbf{r}'$, we have

$$Z'' =_{p^l} M'_{k-1}(\mathbf{r}'\tilde{\mathbf{x}} + \tilde{\tilde{X}} - (\mathbf{r}' + \mathbf{l})\mathbf{w}^T).$$

Now, let us take $(M'_k \mod p^l)$ instead of M''_k and define $Z' = \frac{1}{p^l} M'_{k-1}(M'_k \mod p^l)$. As $M'_k =_{p^l} M''_k$, Z' is a well defined integer matrix, equal to Z'' modulo p^l. Thus, it suffices to take

$$X' = \mathbf{r}'\tilde{\mathbf{x}} + \tilde{\tilde{X}} - (\mathbf{r}' + \mathbf{l})\mathbf{w}^T.$$

Let us now assume that $k < N$. By condition 2, over $\mathbb{Z}$ we have

$$LM_kM_{k+1} = p^l LW. \tag{16.22}$$

Let us define a vector $\mathbf{d}$, such that $M_{k+1} = \begin{bmatrix} \mathbf{d}^T \\ M'_{k+1} \end{bmatrix}$. Now, from Eq. (16.22), (16.18) we can deduce that

$$a_{11}\mathbf{d}^T + p^\alpha \mathbf{w}'^T M'_{k+1} = p^l \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix} LW = p^l W_1,$$

where W_1 is the first row of LW. Thus $a_{11}/p^\alpha \mathbf{d}^T = p^{l-\alpha} W_1 - \mathbf{w}'^T M'_{k+1}$. Let

$$\begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix} Y = \mathbf{y}^T = \begin{bmatrix} y & \mathbf{y}'^T \end{bmatrix}.$$

Modulo p^l we have

$$a_{11}/p^\alpha \mathbf{d}^T =_{p^l} p^{l-\alpha} L\mathbf{y} M_{k+1} - \mathbf{w}'^T M'_{k+1} = a_{11}/p^\alpha \mathbf{d}^T =_{p^l} p^{l-\alpha} y\mathbf{d}^T + L\mathbf{y}'^T M'_{k+1} - \mathbf{w}'^T M'_{k+1},$$
$$\mathbf{d}^T(a_{11}/p^\alpha - p^{l-\alpha} y) = \mathbf{y}'^T M'_{k+1} - \mathbf{w}'^T M'_{k+1}.$$

As $\gcd(a_{11}/p^\alpha - p^{l-\alpha} y, p) = 1$, $a_{11}/p^\alpha - p^{l-\alpha} y$ is invertible over $\mathbb{Z}_{p^l}$, and there exist vector $\mathbf{q}'$ over $\mathbb{Z}_{p^l}$ such that $\mathbf{d}^T =_{p^l} \mathbf{q}'^T M'_{k+1}$. Thus, $\mathbf{d}^T$ is a combination of the rows of M'_{k+1}, $SF(M_{k+1}) = [0, SF(M'_{k+1})]$ and Eq. (16.16) holds.

Now, we need to construct matrices W', Y' such that Cond. 2 holds. By Eq. (16.22) we have

$$p^l \mathbf{v}\mathbf{d}^T + M'_k M'_{k+1} = p^l \tilde{W},$$

where $\tilde{W}$ is composed of all but the first rows of LW. This gives

$$M'_k M'_{k+1} = p^l(\tilde{W} - \mathbf{v}\mathbf{d}^T) = p^l W''.$$

Modulo p^l, W'' is equal to

$$W'' =_{p^l} \tilde{Y} M_{k+1} - \mathbf{v}\mathbf{q}'^T M'_{k+1} =_{p^l} \tilde{\mathbf{y}}\mathbf{d}^T + \tilde{\tilde{Y}} M'_{k+1} =_{p^l} (\tilde{\mathbf{y}}\mathbf{q}'^T + \tilde{\tilde{Y}}) M'_{k+1},$$

where

$$\tilde{Y} = [-\mathbf{l} \quad \mathrm{Id}]Y = [\tilde{\mathbf{y}} \quad \tilde{\tilde{Y}}].$$

It suffices now to put $W' = \frac{1}{p^l}(M'_k \mod p^l)M'_{k+1}$, which is a well defined integer matrix. Moreover $W' =_{p^l} W''$ and thus $Y' = \tilde{\mathbf{y}}\mathbf{q}'^T + \tilde{\tilde{Y}}$ fulfills the requirements.

Remark 16.2.9 1. Conditions 1 and 2 ensure that Thm. 16.2.8 can recursively be applied for the resulting chain $\ldots M'_{k-1}M'_kM'_{k+1}\ldots$ as long as an admissible entry is found.

2. Conditions 1 and 2 are trivially fulfilled for modular images of exact chains of matrices over $\mathbb{Z}$. It suffices to take $Z = X = 0$ and $W = Y = 0$. Thus, Thm. 16.2.8 can repeatedly be applied in order to compute local smith forms at a prime p for exact chains of integer matrices.

3. Let us suppose that $q \in \mathbb{Z}$, q is composite. Suppose that M_k is an exact chain of matrices over $\mathbb{Z}$ and let us take $M_k \mod q$. One reduction as in Thm. 16.2.8 is possible i.e. the Smith forms agree after reduction. However, attempting another reduction may fail, see Ex. 16.2.6.

4. The proof of Thm. 16.2.8 is based on the special form of zero-divisors in $\mathbb{Z}_{p^l}$ compared with $\mathbb{Z}_q$. As a related property we also notice, that for matrices over $\mathbb{Z}_{p^l}$ all minimal generating sets of columns/rows have the same, minimal cardinality equal to the number of non-zero invariant factors modulo p^l. Algorithm LRE of [38, 39] can be adapted to prove this claim. Therefore, by finding a dependency, we may remove the dependent vector from the generating set in order to obtain a generating set of minimal cardinality at the end of the process. In fact, in many aspect, matrices over $\mathbb{Z}_{p^l}$ behave like matrices over a field.

5. Theorems 16.2.2,16.2.8 allow to trace bases of the kernel (resp. image) of M_k, which become LE_k (resp. $E_{k-1}U$), if E_k (resp E_{k-1}) is the initial base. This is an important factor in some applications, see e.g. for computing homologies of maps, see [72, Ch. 7]. Over integers, computational issues regarding coefficient swell arise, which is discussed in [64]. However, this can be seen as a good point of the method, compared to undirect methods such as [44].

17

Smith Form by Reductions - Heuristic Algorithm

17.1 Introduction

Based on reductions and coreductions of Thm. 16.2.2 and Cor. 16.2.4, we may construct an algorithm for the computation of rank and Smith form, for exact sequences of matrices M_k over a PIR R. Whenever an admissible entry is found in matrix M_k, see Def. 16.2.1, the remaining entries in a row and column can be bring to zero by elimination, and then deleted, while the precomputed value of (partial) rank is increased an/or invariant factors updated. The elimination then propagates on the neighboring matrices M_{k-1} and M_{k+1} by deletion of corresponding rows and columns. After this operations, we obtain again an exact chain of matrices and the process can either be repeated for another admissible entry of M_k or, for another matrix of the chain. For a prime p and $l \in \mathbb{Z}, l > 1$, the idea carry on to the modular ring $\mathbb{Z}_{p^l}$ by Thm. 16.2.8, which allows for local Smith form computation.

In this chapter, we will focus on the case when R is equal to $\mathbb{Z}$ and the sequence M_i, $i = 1..N$ is finite. We will propose a heuristic reduction base algorithm for the computation of the Smith form of an exact chain of integer matrices and show by experiments that this approach may be an effective measure against the coefficient swell and filling, allowing for successful computation in the case of very large matrices. Using this approach we were able to conquer the challenge problem of [34] and compute the Smith forms of large matrices coming from the K-theory.

We will start this chapter by presenting the setting for our problem in Sec. 17.1.1. Then we will discuss the choice of an admissible entry for reduction within our heuristics and compare it to the existing elimination-based algorithms for the Smith form computation, such as [31, Sec.5.1-2] and the references therein. This leads to the presentation of our algorithm in Sec. 17.2. Then we present some implementation details in Sec. 17.2.3 and comment on the complexity of our approach. We will then check the performance of our heuristic approach and present experimental results and timings in Sec. 17.3. We will compare the effectiveness of our approach against the parallel rank computation of [34] in Sec. 17.3.1.

17.1.1 Problem Setting

The performance of reduction algorithm depends on the choice of admissible entries at each step of reduction. The choice should ensure that

- the filling and coefficient swell does not hamper elimination and does not lead to memory trashing;
- it is always possible to find the next admissible entry.

Strategies for the choice of an admissible entry can be discussed in two different settings.

1. In the global (or omniscient) setting, all matrices fit into memory and can be processed at the same time.
2. In the local (or linear) setting, only one matrix fit into memory and is processed, additional (static) information on neighboring matrices can eventually be available.

In what follows, we will focus on the local setting, which is a rational choice for very large matrices for which other algorithms fail. This approach can make use of existing sparse matrix structures of LinBox, with some modifications to enhance performance.

Also, admissible entries are independent in a sense that

- new admissible entries in M_k arise as results of elimination on M_k,
- deletion of rows/columns in neighboring matrices cannot turn a non admissible entry into an admissible one,
- finally, any choice of admissible entries in the global setting, can be realized in the local setting choosing no admissible entries for certain matrices.

Therefore we feel that the applicability of global approach is limited in this sense and it is difficult to anticipate a serious gain of the global setting compared to the local one. Also, it seems that launching in parallel several reduction processes in the local settings should be envisaged instead.

17.2 Reduction Algorithm

In the global setting, Alg. 17.2.1 presents the scheme for the reduction procedure. The algorithm gives the general scheme of reconstruction in terms of function *Reduce* and *PartialElimination*. On the input of the algorithm, a finite exact chain $M_k, k = 1..N$ of matrices (over $\mathbb{Z}$ or $\mathbb{Z}_p$ for a prime p). For $i = 1..N$, matrix M_i is written in bases E_i and E_{i-1} and vectors V_i, V_{i-1} associated with M_i contain marks for indices of eliminated rows and columns respectively.

Procedure $Reduce(M_{i+1}, V_i)$ reads matrix M_{i-1} from data file and zeroes rows from M_{i-1} which are marked in V_i. $PartialElimination(M_i, V_{i-1}, V_i, K)$ repeatedly finds admissible entries and perform reductions according to Thm. 16.2.2, maintaining marks, so that V_{i-1} and V_i contain information about the rows and columns that have to be zeroed in the neighboring matrices, see Alg. 17.2.2 for details.

Algorithm 17.2.1 Reduction/coreduction scheme for the Smith form computation

Require: M_k, $k = 1..N$, a finite exact chain of matrices,
Ensure: D_k, $k = 1..N$, a sequence of diagonal matrices,
Ensure: M'_k, $k = 1..N$, an exact chain of matrices, such that

$$SF(M_k) = SF(\mathrm{diag}(D_k, M'_k)), k = 1..N.$$

1: $M'_i = M_i, i = 1..N$;
2: $K = 1$;
3: **repeat**
4: **for** $i = 1$ to $N - 1$ **do**
5: $D_i = PartialElimination(M'_i, V_{i-1}, V_i, K + 1)$;
6: $Reduce(M'_{i+1}, V_i)$;
7: **end for**
8: $D_N = PartialElimination(M'_N, V_{N-1}, V_N, K + 1)$;
9: **for** $i = N$ to 2 **do**
10: $D_i = PartialElimination(M_i'^T, V_i, V_{i-1}, K + 1)$;
11: $Reduce(M_{i-1}'^T, V_i)$;
12: **end for**
13: $D_1 = PartialElimination(M_1'^T, V_0, V_1, K + 1)$;
14: $K = K + 1$;
15: **until** No more admissible entries are found.

Lemma 17.2.1 (Correctness of Alg. 17.2.1) *Suppose $M_k, k = 1..N$, an exact sequence of matrices over a PIR R is given at the entrance of Alg. 17.2.1. Let $1 < i \leq N$. During the course of the algorithm, whenever $PartialElimination(M'_i, V_{i-1}, V_i, K)$ (resp. $(M_{i-1}'^T, V_{i-1}, V_{i-2}, K)$) is about to start (i.e. at line 5,8 (resp. 10,14), the following conditions are fulfilled*

$$M'_{i-1}M'_i = 0, \quad (resp.\ M_i'^T M_{i-1}'^T = 0).$$

As a consequence, at the end of the algorithm, $SF(\mathrm{diag}(D_i, M'_k))$ is equal to the Smith form of the original matrix M_k.

PROOF At the beginning of the algorithm M'_k is an exact sequence and thus the conditions are fulfilled. The condition $M'_{i-1}M'_i = 0$ is first violated when *PartialElimination* on $(M'_{i-1}, V_{i-2}, V_{i-1})$ is performed, as matrix M'_{i-1} is changed. According to Thm. 16.2.2 this can be repaired by a deletion of rows of M'_i, which correspond to eliminated columns.

Provided that V_{i-1} keeps track of columns of M'_{i-1} which have been eliminated, this is immediately done in $Reduce(M'_i, V_{i-1})$ and therefore is fulfilled at the beginning of line 4 or 8.

In the next run of the **for** loop, the call to $PartialElimination(M'_i, V_{i-1}, V_i)$ violates the condition again as M'_i is modified. Again, Thm. 16.2.2 point us to delete columns of M'_{i-1} in order to maintain the conditions. This is not done immediately, but information on the columns is stored in V_{i-1}. Then the call to $PartialElimination(M_i'^T, V_i, V_{i-1})$ changes M_i once again and information of the eliminated rows of M_i are added in V_{i-1}. By Cor. 16.2.4 these columns should be deleted from M_{i-1} as well.

This is exactly done by $Reduce(M_{i-1}'^T, V_{i-1})$, before $PartialElimination$ on $M_{i-1}'^T$ is called. Notice, that the rows of $M_{i-1}'^T$ are exactly the columns of M'_{i-1} and the marks in V_{i-1} are set according to the status of the column set in $PartialElimination(M'_i, V_{i-1}, V_i)$ and $PartialElimination(M_i'^T, V_i, V_{i-1})$. Thus, the call to $Reduce$ ensures that before the launch of $PartialElimination(M_{i-1}'^T, V_{i-1}, V_{i-2}, K)$ we have that $M_i'^T M_{i-1}'^T = 0$ and $M'_{i-1} M'_i = 0$. As the loop continues, the reasoning can be repeated.

The above reasoning justifies that Thm. 16.2.2 and Cor. 16.2.4 can be applies whenever $PartialElimination$ is launched. Thus, condition on the Smith form follows.

17.2.1 Choice of Admissible Entry

At this point, the essential part of the algorithm is the strategy of finding admissible entries in $PartialElimination$. In general, as remarked by many authors, see e.g. [134], finding an elimination scheme that optimize filling and coefficient swell is a NP complete problem. Finding an optimal route for elimination/reduction scheme for the whole sequence seems at least as difficult.

Optimal Elimination Strategy

Suppose now that an optimal elimination strategy for matrix M'_k is known i.e. a sequence $(i_1, i_j), \ldots, (i_K, j_K)$ of indices of admissible entries is given, which results in optimal minimal filling and coefficient swamp. One solution is to perform full elimination on M'_k according to the optimal strategy. Yet, the elimination of M'_k can be suspended and resumed only in the next run of the *repeat* loop of Alg. 17.2.1. Indeed, deleting columns and rows other than i_s and resp. j_s, $s = 1..K$, does not violate the admissibility of entries of M'_k. By preserving a condition that no columns labelled by $i_s, s = 1..K$ are deleted in M'_{k+1}, and no rows labelled by $j_t, t = 1..K$ are deleted in M'_{k-1}, we may then resume optimal elimination strategy for M'_k, possible reducing the cost, as the matrix is smaller. Surely, it is possible that a new choice of admissible entries becomes optimal due to deletion of rows and columns of M'_k. Also, eliminating columns labelled by i_s in M'_{k+1} or eliminating rows labelled by j_s in M'_{k-1} might lead to better strategies.

Heuristic Elimination Strategy

In general it seems that a locally greedy approach, which leads to maximal elimination of M_k might have drawbacks, mostly as no new admissible entries arise from (co)reductions in M_k. Actually, this is exactly the approach proposed by [73] in the original paper, as it uses Thm. 16.2.2 only and not Cor. 16.2.4 for the transposed chain. It seems that a balanced approach, where elimination steps intertwine with (co)reductions is more justified as it allows to localize the computation in M_k as long as appropriate and profit from reductions as well.

In [31, Sec. 5.1-2] a handful of elimination strategies are analyzed or referred to, which can be used here. In particular, Markowitz's minimal degree algorithm can be mentioned here, see [31, Sec. 5.1-2] and the references therein. In [31, Sec. 5.1-2] a modification is proposed and experimentally evaluated, which point out to first reduce the shortest rows, by choosing the entry which lies in the shortest column.

PartialElimination Algorithm

It is easy to enforce a break in the elimination strategy by adding a parameters K_1, K_2 and saying that only rows with less than K_1 non-zero entries and columns with less than K_2 non-zero entries can be eliminated at one pass of the *repeat* loop in Alg. 17.2.1. Also, in the presence of reductions possibility, we propose the following locally greedy modification: if a coefficient swell or filling is encounter, mark the affected row/column as preferred for deletion. This can be taken into account while performing elimination on the neighboring matrices. If reduction is possible, this will result with better filling and slower coefficient growth. In Alg. 17.2.2 we summarize these ideas in a form of an algorithm. In Alg. 17.2.2 we decided to take $K_1 = K$ and $K_2 = \infty$ (or column dimension in practice).

Notice that the elimination of 1-rows comes at small cost as only a row and column are zeroed. This case is equivalent to free face reduction, see Sec. 16.1.1. Moreover, elimination of 2-rows does not lead to the growth of the overall number of non-zero entries. Therefore elimination of 1-2 rows should have higher priority. In fact, 1-2 rows elimination is often sufficient in some cases with geometric e.g. for the computation of homology groups of cubical sets. Indeed, in [73, Sec. 4.3], the authors envisaged a tree structure, which will allow to access shortest rows and columns (notice a correspondence to faces and edges) first. Also, a priority queue can be envisaged.

In Sec. 17.2.3 we discuss a structure of grid to store matrix entries. This modification of sparse matrix implementation allows us to realize Alg. 17.2.2.

17.2.2 Remarks on *Reduce* Procedure

The definition of function *Reduce* is straightforward from its requirements. From the point of view of implementational in order to obtain good complexity of grid creation, it is essential that in the matrix file entries from the same rows come together. Notice

Algorithm 17.2.2 PartialElimination Algorithm

Require: M_i, V_{i-1}, V_i,
Require: $K > 0$, $T \geq 0$ - threshold for stopping condition,
Ensure: D_i, M'_i such that $SF(\text{diag}(D_i, M'_i)) = SF(M_i)$.

```
M'_i = M_i;
prevrank = 0; rank = 0;
repeat
    prevrank = rank;
    for s = 1 or M'_k.rowdim do
        if size(M'_k[s]) ≤ K)        #size is the number of non-zero elements  then
            Find an admissible entry a_st in the preferred or shortest column;
            Eliminate tth column;
            rank = rank + 1;
            Mark V_{i-1}[s] = eliminated, V_i[t] = eliminated;
            If coefficient swell of filling occur, mark column as preferred to reduction; break;
    end for
until rank − prevrank < T
```

that *Reduce* repeatedly gets matrix M'_i and M'^T_i. Thus, *PartialElimination* should write back matrix $M'T_i$ to the file.

For matrices that do not fit into memory, the paper of [37] proposes an elimination approach that allows to treat matrix row by row and perform elimination delayed in time when same pivot is repeated. This seems a good base for the development of function *Reduce* and *PartialElimination* in the case when memory is sacred.

17.2.3 Implementation and Complexity Study

In order to obtain a good complexity of Alg. 17.2.2 we need a matrix structure, that will allow to perform column elimination in $O(size(column))$ time. In standard sparse matrix implementation as a vector of pairs (column index,element) or a pair of vectors of elements and column indices, this is not ensured. Additionally we require that it is possible to read all elements of a row in linear time and that it is easy to perform a transpose. The order of elements in the row is not important. To ensure that, we propose a structure of a horizontally and vertically connected grid, the idea of which is shown in Fig. 17.1. This is enough to efficiently perform *PartialElimination* and 17.2.1.

The structure can be realized by two vector of lists of elements, where element consist of row and column indices and the non-zero value. It is implemented in LinBox library in file "linbox/matrix/grid.h". Additional information such as gcd of row/column or its size (i.e. the number of non-zero elements) can also be stored. Also, it can be wrapped in a priority queue, which would manage indices of shortest rows.

The value of an element can be changed provided that a reference to the element is given. Given a reference, an element, can be removed in $O(1)$ time. An element can be added

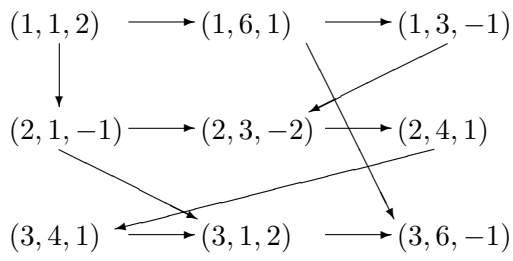

Figure 17.1: Grid structure

to the list in $O(1)$ time, provided that a reference to the element above/below it is given. Let us now analyze the complexity of performing a row elimination.

Lemma 17.2.2 (Cost of Row Elimination) *Let a_{ij} be an admissible element of matrix A in a row consisting of N non-zero elements. Let M be the number of non-zero elements in the jth column and M_{max} the maximal number of non-zero elements in all columns of A. Let grid position of a be given. Then the elimination of row r and column c can be performed in at most $O(NM)$ integer multiplications, and $O(NM_{max})$ additional grid operations.*

PROOF The formula for elimination of j column by the ith entry a_{ij} changes entries a_{st} according to the equation

$$a_{st} = a_{st} - a_{it}\frac{a_{sj}}{a_{ij}} \quad , \|a_{st}\| \leq \|a_{st}\| + \|a_{sj}\|\|a_{sj}\| \in O(\|A\|^2), \tag{17.1}$$

which means that a_{st} is changed in a_{it}, a_{sj} are both non zero. By definition of N and M this happens in $(N-1)(M-1)$ cases. A change an operation on grid such as: change of the value, insertion, deletion, which cost $O(1)$ each. For each $t, a_{it} \neq 0$, this can be done by scanning the tth and jth column at the same time to find the right place in at most $2M_{max}$ comparisons.

Lemma 17.2.3 (Cost of PartialElimination) *Let $n, m, \in \mathbb{N}$ and let A be a matrix $n \times m$ matrix on which PartialElimination Alg. 17.2.2 is performed. Suppose that rows of size at most $K > 0$ are eliminated. Let A' denote the matrix at the exit of the algorithm, and let r denote the increase in precomputed rank i.e. $r = \text{rank}(A) - \text{rank}(A')$. Then the cost of PartialElimination Alg. 17.2.2 is $O(rKn)$ integer multiplications and grid operations.*

PROOF The cost of treating an individual admissible entry is at most $O(Kn)$ by Lem. 17.2.2.

Remark 17.2.4 In the worst case, the growth of coefficient in the case of integer elimination is exponential, as the elimination formula in Eq. (17.1) gives

$$\|a_{st}\| \leq \|a_{st}\| + \|a_{sj}\|\|a_{sj}\| \in O(\|A\|^2),$$

if $\|A\|$ is a bound on entries of matrix A at the beginning of elimination.

We may yet control a_{ij}, a_{sj} by the choice of admissible entries in Alg. 17.2.2, thus limiting the coefficient swell. Similarly, filling i.e. the growth of N, M, can be controlled. Deletion of rows/columns by reductions also helps control this factors. Indeed, in Fig. 17.2 we show that filling and in matrix norm (in Fig. 17.3) were kept at bay in the case of large scale computation of Smith form for matrices from K-Theory.

17.3 Experiments and Results

All computation in this section were done on a SGI Altix 3700 gathering 64 Itanium2 processors with 192Gb memory and running SuSE Linux Enterprise System 10. Further information on this platform are available at `www.math.uwaterloo.ca/mfcf/computing-environments/HPC/pilatus`.

We are grateful to Arne Storjohann and to the Computer Science Computing Facilities of the University of Waterloo for letting us fill up the resources.

17.3.1 Application to K-Theory

Comparison with [34]

In our PASCO 2007 paper [34], we pose a problem of computing the Smith forms for a set of 17 sparse matrices, representing the boundary operator for one of the problems in K-theory. The size of largest matrices in the chain was approaching 2,000,000 with 40 millions non-zero entries, see Tab. 17.1 for parameters.

The challenge was to compute high torsions for primes $p > 7$ or to conclude that these do not exist. Theory predicted that small torsions for primes $p \leq 7$ should be present, due to the construction of the chain as an equivalence class. Indeed, this is the case as seen in Tab. 17.1.

1. **Rank Computation**

   In [34] we dealt with the computation of the rank of matrices over $\mathbb{Z}$ or, equivalently, $\mathbb{Q}$. The computation took several months to complete in a highly parallel environment. In [34, Sec. 2.2] we present the parallel Wiedemann algorithm, which was used to compute the rank of the matrices modulo a FFT prime. Thus, we obtained a lower bound for the rank over $\mathbb{Z}$. Using Eq. (15.1) we can conclude that for matrix $M_k = GL7d_k$ of the sequence

   $$\operatorname{rank}(M_k) \leq \ker(M_{k-1}) = m_k - \operatorname{rank}(M_{k-1}),$$

   where m_k and n_k are the number of resp. rows and columns of matrix M_k. The upper and lower bounds obtained in this way agreed on all matrices except GL7d13.

M_k	Ω_k	n_k	m_k	prec. rank	rank	ker	Smith form
GL7d10	8	60	1	1	1	59	1
GL7d11	1,513	1019	60	59	59	960	1 (59)
GL7d12	37,519	8899	1019	960	960	7939	1 (958), 2 (2)
GL7d13	356,232	47271	8899	7937	7938	39333	1 (7937), 2 (1)
GL7d14	1,831,183	171375	47271	39311	39332	132043	1 (39300),2 (29),4 (3)
GL7d15	6,080,381	460261	171375	131994	132043	28218	1 (131993), 2 (46),12 (4)
GL7d16	14,488,881	955128	460261	328167	328218	626910	1 (328183), 2(33), 4(1) 12 (1)
GL7d17	25,978,098	1548650	955128	626583	626910	921740	1 (626857), 2(52), 4(1)
GL7d18	35,590,540	1955309	1548650	920292	921740	1033569	1 (921637) 2(100) 42 (2) 252 (1)
GL7d19	37,322,725	1911130	1955309	1021546	1033568	877562	1 (1033458) 2(110) * ?
GL7d20	29,893,084	1437547	1911130	876185	877562	559985	1 (877526), 2(33), 6 (3)
GL7d21	18,174,775	822922	1437547	558862	559985	262937	1 (559895), 2(88), 6(2)
GL7d22	8,251,000	349443	822922	260264	262937	86506	1 (262937), 2 (98), 4(3), 12 (1)
GL7d23	2,695,430	105054	349443	86486	86505	18549	1 (86488), 2 (12), 6 (3), 42 (1), 84 (1)
GL7d24	593,892	21074	105054	18549	18549	2525	1 (18544),2 (4),4 (1)
GL7d25	81,671	2798	21074	2522	2525	273	1 (2507), 2 (18)
GL7d26	7,412	305	2798	264	273	32	1 (258), 2 (7), 6 (7), 36 (1)

Table 17.1: Results of the rank and Smith form computation for $GL_7(\mathbb{Z})$ matrix M_k of dimension $n_k \times m_k$ with Ω_k non-zero entries. Precomputed rank by Alg. 17.2.1, rank and nullity (ker) over $\mathbb{Z}$ computed in [34] and the Smith form computed using Thm. 16.2.2 is given.

Then, the rank of GL7d13 can be computed exactly using Alg. 17.2.1. Thus, we have certified all results of the rank computation over $\mathbb{Z}$ ($\mathbb{Q}$). The computation of rank and kernel enabled us to compute cohomologies over $\mathbb{Q}$ of $GL_7(\mathbb{Z})$.

2. **Smith Form Computation**

   In [34], the Smith form was computed for 8 smallest matrices by a modified adaptive algorithm of [112]. For two other matrices, partial results i.e. rank modulo 2,3,5 was obtained, but filling was to much of a factor and memory trashing occurred for the remaining matrices. Thanks to Alg. 17.2.1 we were able to obtain other results, as seen in Tab. 17.1.

   The Smith forms were computed by Thm. 16.2.2 using the value of precomputed rank and torsions obtained by the reductions and coreductions in Alg. 17.2.1. A few factors 2,3,4 up to 12, were indeed found by Alg. 17.2.1 for some of the matrices. The Smith form of the remaining matrix M'_k was computed once again the modified algorithm of [112] described in [34, Sec. 2.6].

   To be specific, column 'prec. rank' of Tab. 17.1 gives the number of rows and columns eliminated by Alg. 17.2.1 in matrix M'_k, and thus, provides the lower bound for its rank. The value is close to the actual value computed in [34]. As a result, at the end of Alg. 17.2.1, matrix M'_k was small and Smith form computation by modified algorithm of [112] was possible. The rank of the output matrix M'_k is the difference between the rank of M_k and the precomputed rank. From Tab. 17.1, it can be seen that matrix GL7d24 was reduced to a zero matrix, and the largest remaining matrix GL7d19 had rank 12022 after reductions.

3. **Comparison of Timings and Performance**

   The full reduction for the $GL_7(\mathbb{Z})$ matrix sequence required 147 passes of the **repeat** loop of Alg. 17.2.1 i.e. at most 294 runs (iterations) of *PartialElimination* by Alg. 17.2.2 in the case of largest matrices GL7d19 and GL7d20. We will use the word 'iteration' to refer to one pass of Alg. 17.2.1 over matrix M'_k or its transpose.

   Table 17.2 presents the timings for Alg. 17.2.1 for $GL_7(\mathbb{Z})$ matrices and compares it to the parallel rank computation in [34]. Column T_{reduce} and T_{elim} gives the overall time of calls to *Reduce* and *PartialElimination* procedures respectively, for matrix M'_k and M'^T_k .

   Column $T_{overall}$ of Tab. 17.2 approximates the running time of Alg. 17.2.1 after which no new admissible entries were found for M'_k, M'^T_k. Contrary to other columns, it reflects the cost of computation on the whole sequence of matrices an NOT on matrix M'_k itself. This is an approximate of true timing based on data logged during computation. Due to large scale of computation, the values in this column include the additional cost of data handling e.g. read and write operation and memory managing. Notice that the overall running time of Alg. 17.2.1 is $\max(T_{overall}$.

   On the contrary, $T_{parallel}$ accounts for the time of parallel rank computation, see [34, Tab. 4]. The number of processors is given in brackets. The value is a sum of the time of sequence generation and σ-basis computation for the parallel rank. For

M_k	T_{reduce}	T_{elim}	$T_{overall}$	$T_{parallel}$	$T_{sequential}$
GL7d12	0.31 s	0.1 s	31.14 m [3]	12.48 s [30]	9.6 s
GL7d13	6.09 s	1.62 s	5.64 h [3]	5.01 m [30]	25.83 m
GL7d14	34.99 m	53.69 m	1.22 d [36]	1.01 h [30]	16.8 h
GL7d15	2.78 h	8.95 h	15.11 d [73]	4.7 h [30]	2.81 d
GL7d16	10.03 h	61.89 h	**29.16 d [142]**	1.39 d [30]	**34.11 d**
GL7d17	34.07 h	12.79 d	**39.84 d [162]**	14.57 d [30]	**420 d**
GL7d18	6.53 d	28.84 d	**206.4 d [260]**	29 d [40]	**840 d**
GL7d19	8.37 d	20.48 d	**240.6 d [294]**	36.56 d [50]	**1050 d**
GL7d20	7.96 d	9.82 d	**240.6 d [294]**	11.41 d [48]	**300 d**
GL7d21	4.51 d	11.75 d	218.6 d [276]	5.55 d [30]	150 d
GL7d22	44.57 h	11.97 d	141.4 d [211]	22.32 h [30]	20.59 d
GL7d23	1.78 h	3.85 h	9.83 d [52]	2.26 h [30]	1.46 d
GL7d24	16.18 s	6.9 s	12.5 h [15]	14.26 m [30]	1.57 h
GL7d25	1.92 s	0.77 s	9.04 h [8]	43.03 s [30]	46.8 s
GL7d26	0.2 s	0.08 s	9.04 h [7]	2.06 s [30]	0.9 s
Σ	31.28 d	98.8 d	**240.6 d [294]** [1]	99.75 d	**2796 d**

Table 17.2: Approximate running times of Alg. 17.2.1: time of *Reduce* and *PartialElimination* on M_k; the overall time of running Alg. 17.2.1 on M_k until no admissible entries were found, with the number of iterations in brackets. For comparison, in two last columns: parallel times for rank computation in [34] and equivalent sequential times are given. Times in seconds [s], minutes [m], hours [h] or days [d]

comparison, $T_{sequential}$ is given, which corresponds to the equivalent sequential time of sequence generation. Notice, that this gives the timiming of only one modular rank computation, whereas $T_{overall}$ gives the time of the Smith form computation.

The last row of the column give the overall time of all calls to *Reduce*, *PartialElimination* and Alg. 17.2.1, compared to the overall parallel and sequential time of the rank computation by the Wiedemann algorithm in [34]. Notice that *PartialElimination* (cpu time) took approximately the same time as the overall parallel computation. *Reduce* (which mainly read matrix entries from files) took one third of this time. The actual running time given in $T_{overall}$ is about twice the cpu time of *Reduce* and *PartialElimination*. The computation was highly sequential, thus it it worth remarking, that the whole (real) time of Alg. 17.2.1 was over 11 times shorter, than the sequential computation of the modular rank by Wiedemann's algorithm would be. Moreover, the timing was still comparable to the parallel rank computation, which is significant, as the computation of integer Smith form is a much complex task than modular rank computation.

Reductions for $GL_7(\mathbb{Z})$ Matrices

In this section we will present technical details of the implementation of Alg.17.2.1 and experiments with $GL_7(\mathbb{Z})$ matrices. We will show that reduction/coreduction scheme enabled us to deal with filling and coefficient swell despite the fact that our implementation did not took advantage of all ideas mentioned in Sec. 16. In what follows, unless otherwise

stated, we will drop the distinction between reductions and coreductions as passes of Alg. 17.2.1 on M'_k and M'^T_k resp. We will refer to the run of *Reduce* procedure as 'reduction' and to the run of *PartialElimination* as 'elimination'. As a consequence, elimination is always performed on rows of a matrix, an columns are changed according to the elimination formula.

Our implementation of Alg. 17.2.1 for the experiment with $GL_7(\mathbb{Z})$ matrices was slightly different than described in Sec. 17.2.1. In particular, the idea of marking preferred rows and columns was not implemented. Also, only 1-2 rows were treated as soon as they appeared, and all the others were discovered in repeated passes on the rows. In fact, reduction scheme for matrices $GL_7(\mathbb{Z})$ took relatively long to compute, see Tab. 17.2. Actually, the experiments with the matrices gave raise to the ideas presented in Sec. 17.2, as we gradually designed and implemented new heuristic strategies in order to improved the performance. Then, some part of the initial computation was re-run to evaluate the ideas.

1. **Rank Precomputation and the Number of Non-zero Elements**

   Figures Fig. 17.2 presents the change of the number of non-zero elements at the end of PartialElimination and the value of the precomputed rank throughout Alg. 17.2.1, for each matrix separately. Comparison of the plots can reveal interesting dependencies between minima and maxima of Ω and the rate on which the precomputed rank grew.

   It can be notice that fast rank precomputation for M'_k (i.e. large number of rows eliminated in Alg. 17.2.2 at a given iteration) generally accounts for a sharp decrease in the value of Ω in M'_k and the neighboring matrices. Also, if no more admissible entries can be found for M'_k (the matrix becomes inactive) i.e. the maximal precomputed rank is reached, then

   - the computation on smaller neighboring matrix is completed, no further reductions or elimination are possible;
   - the elimination of matrix M'_k is completed, Ω_k will decrease sharply due to reductions;
   - no reductions are induced by M'_k on the neighboring matrices, as a result, filling occurs for the bigger one;

   In general, the plot of Ω_k in Fig. 17.2 can be read as follows. Whenever the value of Ω_k increased, elimination was the dominant operation in the iteration. If the value of Ω_k decreased, this means that reduction was the dominant factor. Surely, elimination can increase the value of Ω_k only to a certain level, when the remaining matrix M'_k becomes mostly dense. At this point the number of short rows in *PartialElimination* is limited, and reductions gain on significance. This results with a slow downwards trend, see plots of Ω_k for GL7d17, GL7d22 on Fig. 17.2. The final sharp fall occurs at the moment where the dimension of M'_k becomes smaller than parameter K in *PartialElimination*.

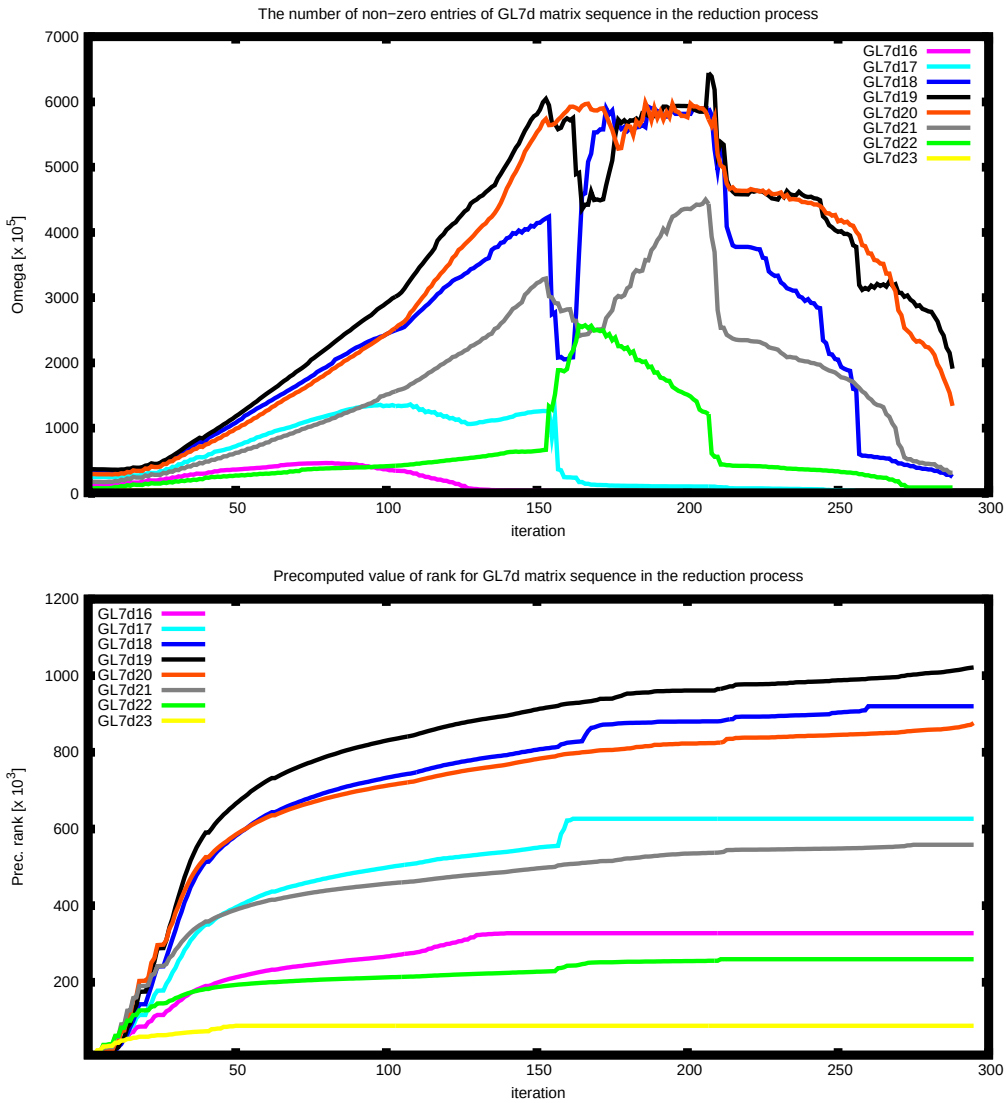

Figure 17.2: In the upper plot, evolution of Ω for $GL_7(\mathbb{Z})$ matrices throughout the reduction process is presented. In the lower plot, precomputation of rank for $GL_7(\mathbb{Z})$ matrices is presented.

2. **Dealing with Filling**

   One of the most important modifications of our method compared to Alg. 17.2.1 was an engineered choice of parameter K i.e. the size of rows treated in $PartialElimination$. For iterations 1-170, the value of K increased uniformly for all matrices by a constant factor. Then we decided to provide K for each matrix M_k separately and set it to much bigger value for border active matrices for the sequence. Assuming that no admissible entries exist for M'_{k-1}, $PartialElimination$ on M'_k would was run with the value of K at least twice as large as for M'_{k+1} (or M'_{k-1}, depending whether we consider reduction or coreduction phase). The value of K was adjusted experimentally in order to ensure that precomputed rank grows steadily for each matrix of the sequence.

   This approach was based on the following heuristics:

   - elimination should be easier on M_k as it is currently the smallest matrix to eliminate;
   - the only way to enhance reductions on M_k is to enhance eliminate M_{k+1}; thus, by inducing more reductions on M_{k+1} we should enhance its elimination, and consequently obtain more reduction on M_k in the following coreduction step;

   This approach indeed allowed us to complete the computation for border matrices efficiently. Yet, it came at the cost of increased filling, as it can be see on Fig. 17.2 for border active matrices GL7d18 and GL722.

   In fact, it was necessary to introduce a bound on the number of non-zero elements in the eliminated columns, which we temporally set to αK, for a heuristically chosen value $\alpha = 100$. As a result, the rate of growth of the precomputed rank became slower, but filling and the time of one row elimination was limited.

   Let us summarize the analysis of Fig. 17.2. In iterations 1-160, Ω_k gradually increased for all matrices, before slowly falling at the end of the computation on matrices GL7d16, GL7d17 (see Tab. 17.2 for exact iteration numbers). At iterations 162, $PartialElimination$ for matrix $GL7d17$ finished, and a sharp fall in the number of non zero elements of GL7d18,GL7d19 followed. Then, with no reductions coming from matrix $GL7d17$, Ω_{18} increased even more sharply as more rows were allowed to be eliminated (we increasing the value of K for $GL7d18$). This forced us to limit the elimination by setting the upper size on columns that can be eliminated. As a result, Ω_k for matrices $GL7d18, GL7d19, GL7d20$ stayed roughly at the same level until iterations 210, when we released the condition on columns. This immediately led to final elimination of $GL7d22$. A sharp decrease in Ω followed for all active matrices $GL7d21, GL7d20, GL7d19, GL7d18$ and the number of non-zero elements gradually continued to decrease.

3. **Dealing with Matrix Norm Growth**

   In the upper plot of Fig. 17.3 we give the average time of one row elimination for every iteration, computed by dividing the time of $PartialElimination$ by the precomputed value of rank. The value was approximated for iterations and matrices

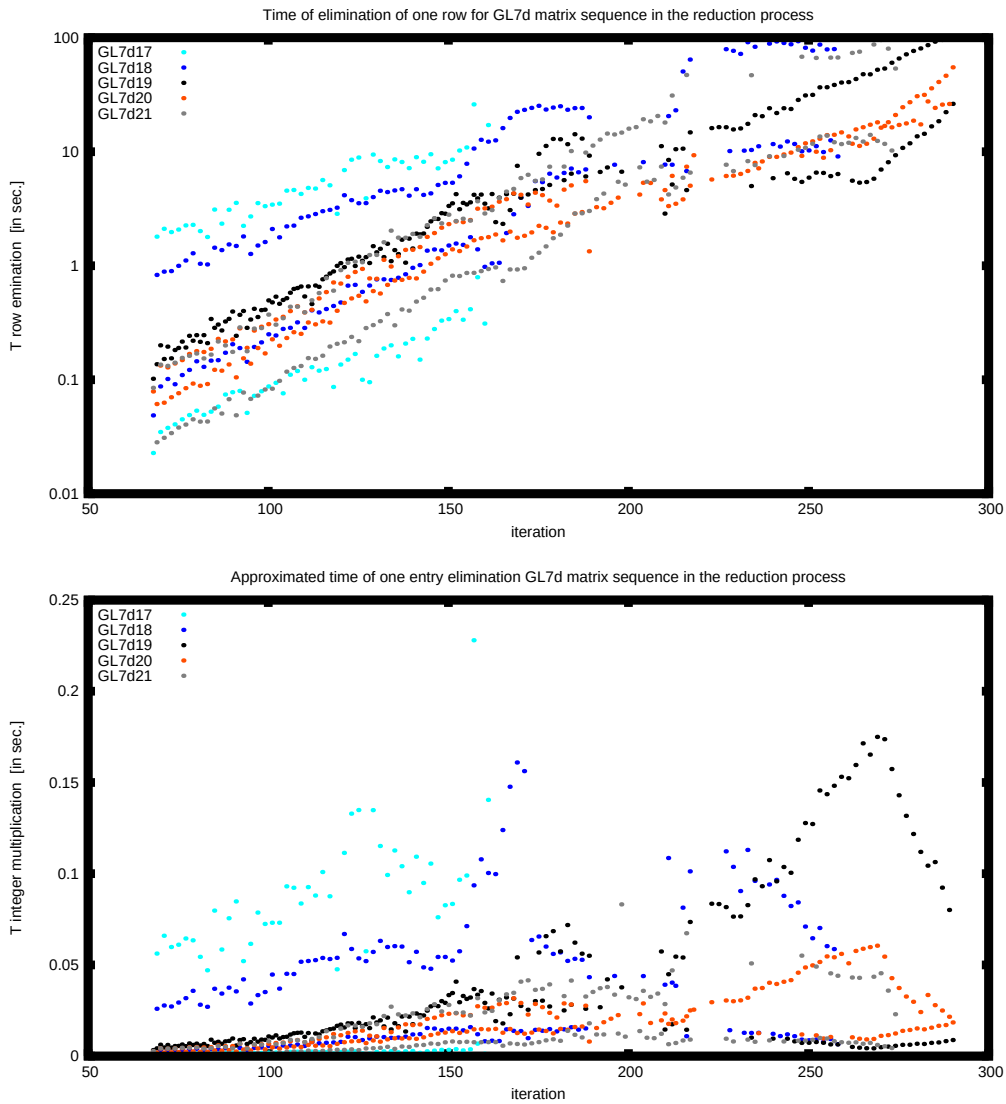

Figure 17.3: In the upper plot, time of one elimination in Alg. 17.2.1 for $GL_7(\mathbb{Z})$ matrices in logarithmic scaling is presented. In the lower plot, the time of elimination of one element is presented, which was approximated from the time of one row elimination. This corresponds to the maximal norm of eliminated rows and columns at each iteration.

for which the precomputed rank was greater than 100. Timing is presented in logarithmic scale. Two branches of each plot correspond to reductions and coreductions on the matrix. The time seems polynomial for one branch of the plot but rather exponential for the other branch, at least in some fragments of the plot. It is however difficult to analyze the plot after iteration 170, where the value of K was different for every matrix and for reductions and coreductions.

Thus in the lower plot of Fig. 17.3 we analyze the average time of the elimination of one entry, which was computed by dividing the time of $PartialElimination$ by the precomputed value for rank and parameter K. Again, the value was approximated for iterations and matrices for which the precomputed rank was greater than 100. As the cost of integer multiplication is $\sim (\log(\|M'_k(i)\|))$, where $\|M'_K(i)\|$ denote the norm of M'_k, this gives a rough approximation of the growth of the norm of matrix entries in Alg. 17.2.1.

Two trends for every matrix are observed: for even and odd iterations i.e. reductions and coreductions on matrix M'_k and M'^T_k respectively. This means that although the norm of a matrix should be approximately the same in both cases, rows and columns with different norms were chosen in the case of the call to $PartialElimination$ on M_k and M_k^T. It seems that the explosion in norm occurs for one branch of computation, whereas for the other the growth is linear. Moreover, after it 210 the growth seems under control and decrease in time can be observed at the end of the computation for both branches.

It is significant that by temporally limiting the size of columns eliminated in $PartialElimination$, we were able to reduce the time of one row elimination and the cost of one entry elimination for the remaining matrices, when the condition was dropped in iteration 210. As the elimination of border active matrices was in general not controlled, it is not surprising to see that the cost of elimination of border matrices was often the biggest in every pass of the *repeat* loop of Alg. 17.2.1.

17.3.2 Application to Cubical Homology

The reduction algorithm of [73] has been implemented in the CAPD[2] library for the computation of homologies of cubical sets. The reductions are performed on the complex itself, which results with better performance, when compared to the applying Alg. 17.2.1 to matrices generated for the (initial) boundary operator. Also, other kinds of reductions are possible in the case of 3D cubical sets, which result in almost instant performance. Still, the timings for algebraic reductions are comparable for both procedures.

In Tab. 17.3 we present the timings of homology computations for matrices of over 8 million cubes. AsLtHom and CrHom are specialized reduction procedures for 3D sets implemented in CAPD, see [94]. ArHom is the implementation of algebraic reductions (for complexes) in the CAPD library. LinBoxHom refers to our implementation of Alg. 17.2.1 in LinBox, which was applied to compute homologies over $\mathbb{Q}$ (by rank computation).

[2] http://capd.wsb-nlu.edu.pl/

	AsLtHom	CrHom	ArHom	LinBoxHom(rank)	LinBoxHom/ArHom
P0098	10.9408	11.051	3936.69	15022.9	3.82

Table 17.3: Times (in sec.) for the computation of homologies for the sets of over 8 million cubes.

M_k	n	m	rank	ker
P0098_1	8,392,997	26,299,032	8,392,997	17,906,035
P0098_2	26,299,032	27,427,163	17,906,035	9,521,128
P0098_3	27,427,163	9,520,072	9,520,071	1

Table 17.4: Matrix sizes, rank and kernel for P0098 matrices.

The time of the generation of matrices was not taken into account in this case. The time of reading matrices from data set was taken into account and is responsible for a big part of the workload.

In Tab. 17.4 the results of rank computation for matrices of homology of the same cubical set are presented. The result 17,906,035 is the biggest rank ever computed by LinBox. In particular, the time of full reduction for the biggest matrix was 9740,47 sec. Only one run of the **repeat** loop of Alg. 17.2.1 was required to reduce matrices and to allow for the computation of rank using sparse elimination methods. Elimination (deletion) of rows of size 1 coupled with reductions was sufficient to complete the computation.

Conclusions

In part I of the thesis, we introduced the notion of output dependent and expected complexities as measures of performance of adaptive algorithms, see Sec. 2.3. Then in Ch. 3 we showed how this concepts can be applied in order to analyze Smith form algorithm for dense matrices of [44] and possibly improve its running times by introducing an output-dependent modification in Alg. 3.4.1. Thanks to the adaptive approach, algorithm of [44] might efficiently be used on other types of matrices e.g. sparse, symmetric, Hankel/Toeplitz type. The evaluation of the expected complexity of Alg. 3.4.1 for these matrix types is an open problem, as it is subject to results on the expected number of non-trivial invariant factors.

The main contribution of Part II of the thesis is the adaptive determinant algorithm showed in Alg. 8.4.1. It is based on the extended bonus concept presented in Alg. 7.3.1. Alg. 8.4.1 is a version of preconditioned Chinese Remaindering loop, which is a generalization of the ideas of [2], see Ch. 6.

In Thm. 8.7.2 we give the expected complexity of the algorithm for random dense matrices which improves over the algorithm of [122] by a $O(\log^{0.5}(n))$ factor. At the same time, Thm. 8.7.3 proves that the worst case complexity of Alg. 8.4.1 is that of the state-of-the-art, see [78, 79] for discussion on determinant algorithms. By Thm. 8.7.1, Alg. 8.4.1 can be even faster on many matrices. Actually, our algorithm follows the computational part of [2] at the beginning of computation and apart from some degenerated case should always be faster than [2] or CRA scheme, see Sec, 8.9. Moreover, it will generally perform better than the determinant of [44], as it computes a smaller number of invariant factors.

We tested the algorithm on dense and sparse matrix representations and remarked that it should also be applicable to structured matrices. The evaluation of expected performance of Alg. 8.4.1 is subject to results on the expected number of non-trivial invariant factors.

While analyzing Alg. 8.4.1 we came across several problems regarding the distribution of almost uniformly distributed matrices and random integer matrices. In Lem. 5.2.2, we generalize the result of [10] on the probability of rank distribution to the case of almost uniformly distributed matrices over a field. In Lem. 5.2.3 the probability that rank is jess than given j is evaluated. Question remains whether better bounds could be found by evaluation of sums from Eq. (5.12).

In Thm. 5.3.10 we give a asymptotically better bound on the expected number of non-trivial invariant factors for a random dense matrix, which is $O(\log^{0.5}(n))$ compared to previous $O(\log(n))$ bound of [44, Thm. 6.2]. In Cor. 5.3.15 we apply the result of [7] to almost uniformly distributed sparse matrices, under some favorable assumption of Eq. (5.39). The computation of expected number of non-trivial invariant factors for integer structured (sparse, symmetric, Hankel/Toeplitz) matrices remains an open problem. In particular, Teoplitz/Hankel $n \times n$ matrices are determined by $O(n)$ of their entries, which leaves us with a low degree of freedom in the choice of random matrix. For very sparse matrices (e.g. banded), the problem can be restricted to the class of matrices, for which all gcd of rows and columns are 1, see Sec. 8.10.1.

In fact, in view of Eq. (5.39) evaluating the distribution of minors of a matrix or their gcd seems and interesting problem. The question is not easy, as can be seen in Thm. 5.4.5,

where the distribution of determinant modulo p^s is analyzed. In Remark 7.5.2 we state another problem, of distribution of minors of $s_n(A)A^{-1}$ for random matrix A, which could then be apply to simplify extended bound computation.

The main contribution of part III is the development of preconditioning methods in Ch. 13, which allow us to use integer algorithms and integer precondition CRA instead of rational CRA. This allows us to carry on the computation of determinant, linear system solution and minimal/characteristic polynomial for rational matrices. In Alg. 14.1.2 and 14.1.5 we combine the ideas in a form of an adaptive rational algorithm. These rational algorithms are based on the general algorithm for the Chinese Remaindering. Thus, in Ch. 10 we present a survey on CRA, which we and by a proposition of introspective reconstruction scheduling in Sec. 10.7.

Additionally, in order to improve the performance of rational reconstruction, we corrected and implemented the algorithm of [131], see Alg. 11.4.3, and show how it can be used in Maximal Quotient RR of [90], see Alg. 11.5.1.

Exact approach should be compared to numerical methods in terms of the quality of the result an its applicability to real life problems. Namely, as ill conditioned matrices are sensible to small entries modification, even a rigorous evaluation of the floating point representation of real data might lead to large errors. On the other hand, continued fractions offer best approximants of real numbers with smallest denominators, in the sense of [70, Ch. 4]. Thus, representing data by continued fractions might be an interesting alternative. Exact computation with continued fractions can be carried on by algorithms developed on Ch. 14.

In part IV of the thesis, we generalized Thm. 16.1.1 of [73] to the case of admissible entries over PIR, see Thm. 16.2.2. Also, in Cor. 16.2.4 we proposed an analogous concept of coreductions, see also [93] for an independent idea. This resulted in a heuristic (co)reduction scheme for Smith form computation in Alg. 17.2.1. In Sec. 17.3 we showed that it can be used to solve extremely large computational problems (see Sec. 17.3.1) and at the same time has comparable timings with software that deals purely with complex chains (see Sec. 17.3.2).

Perspectives for development include a parallelization of the scheme. Indeed, the scheme we propose in Alg. 17.2.1 is very sequential although many parts of the computation are independent. Parallelism can be envisaged for admissible elements coming from different and not neighboring matrices. Synchronization would be required in order to preserve the chain condition at the entry of elimination phase, see Lem. 17.2.1.

Moreover, in Alg. 17.2.2 full row elimination is required, which does not allow us to follow existing polynomial Smith form algorithms such as [80, 18]. Moreover, we get blocked if no admissible entry is found. Thm. 16.2.2 can be restated to capture the idea of elimination being a base change, of which case (co)reduction scheme is a simple case. In global setting, maintaining same bases for neighboring matrices can be envisages, yet it is not sure whether additional computational cost can be outweighed by benefits.

Finally, it would also be interesting to analyze if and how other concepts generalize to matrix representation e.g. reduction of acyclic subspaces of [94], shifting [74], minimal free resolution [65], and the computation of a dual complex to mention a few.

Bibliography

[1]

[2] J. Abbott, M. Bronstein, and T. Mulders. Fast deterministic computation of determinants of dense matrices. In Dooley [30], pages 197–204.

[3] J. Abbott and T. Mulders. How tight is hadamard's bound? *Experimental Mathematics*.

[4] J. Adams, B. D. Saunder, and Z. Wan. Signature of symmetric rational matrices and the unitary dual of lie groups. In Kauers [81].

[5] C. W. Antoine, A. Petitet, and J. J. Dongarra. Automated empirical optimization of software and the atlas project. *Parallel Computing*, 27:2001, 2000.

[6] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[7] J. Blömer, R. Karp, and E. Welzl. The rank of sparse random matrices over finite fields. *Random Struct. Algorithms*, 10(4):407–419, 1997.

[8] A. Bogdanov and L. Trevisan. *Average-Case Complexity*. 2006.

[9] W. Bosma, J. Cannon, and C. Playoust. The magma algebra system i: the user language. *J. Symb. Comput.*, 24(3-4):235–265, 1997.

[10] R. Brent and B. McKay. Determinants and ranks of random matrices over zm. *Discrete Mathematics*, 1987.

[11] R. Brent and B. McKay. On determinants of random symmetric matrices over zm. *Ars Combinatoria*, 1988.

[12] S. Cabay and T. P. L. Lam. Congruence techniques for the exact solution of integer systems of linear equations. *ACM Trans. Math. Softw.*, 3(4):386–397, 1977.

[13] B. W. Char, B. D. Saunders, and B. Youse. Linbox and future high performance computer algebra. In Watt [132], pages 15–23.

[14] L. Chen, W. Eberly, E. Kaltofen, D. B. Saunders, W. J. Turner, and G. Villard. Efficient matrix preconditioners for black box linear algebra. *Linear Algebra and its Applications*, 343–344:119–146, 2001.

[15] L. Chen and M. Monagan. Algorithms for solving linear systems over cyclotomic fields. *J. Symb. Cmpt., submitted*, 2008.

[16] Z. Chen and A. Storjohann. A blas based c library for exact linear algebra on integer matrices. In Kauers [81], pages 92–99.

[17] M.-D. Choi. Tricks or treats with the hilbert matrix. *Amer. Math. Monthly.*

[18] T.-W. Chou and G. Collins. Algorithms for the solution of systems of linear diophantine equations. *Journal of Computation*, 1982.

[19] G. E. Collins and M. J. Encarnación. Efficient rational number reconstruction. *Journal of Symbolic Computation*, 20:287–297, 1994.

[20] C. Cooper. On the distribution of rank of a random matrix over a finite field. *Random Struct. Algorithms*, 17(3-4):197–212, 2000.

[21] C. Cooper. On the rank of random matrices. *Random Struct. Algorithms*, 16(2):209–232, 2000.

[22] D. Coppersmith and S. Winogard. Matrix multiplication via arithmetic progression. In *Proc. 19th Annual ACM Symposium of Theory of Computing*, pages 1–6, 1987.

[23] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms.* MIT Press and McGraw-Hill, 1990.

[24] V. D. Cung, V. Danjean, J.-G. Dumas, T. Gautier, C. Rapine, J.-L. Roch, and D. Trystram. Adaptive and hybrid algorithms: classification and illustration on triangular system solving. In *Proceedings of Transgressive Computing 2006, Granada, España*, 2006.

[25] P. d'Alberto and A. Nicolau. Adaptive strassen's matrix multiplication. In *Proceedings of the 21st annual international conference on Supercomputing*, pages 284–292, 2007.

[26] V. Danjean, R. Gillard, S. Guelton, J.-L. Roch, and T. Roche. Adaptive loops with kaapi on multicore and grid: applications in symmetric cryptography. In Watt [132], pages 15–23.

[27] J. Dixon. Exact solution of linear equations using p-adic expansions. *Numerische Mathematik*, 40(1):137–141, 1982.

[28] B. R. Donald and D. R. Chang. On the complexity of computing the homology type of a triangulation. In *SFCS '91: Proceedings of the 32nd annual symposium on Foundations of computer science*, pages 650–661, Washington, DC, USA, 1991. IEEE Computer Society.

[29] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 16(1):1–17, 1990.

[30] S. Dooley, editor. *ISSAC'99. Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation, Vancouver, Canada.* ACM Press, New York, July 1999.

[31] J.-G. Dumas. *Algorithmes parallèles efficaces pour le calcul formel: algèbre linéaire creuse et extensions algébriques.* PhD thesis, Laboratoire Informatique et Distribution, 2000.

[32] J.-G. Dumas, editor. *ISSAC'2006. Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation, Genova, Italy.* ACM Press, New York, July 2006.

[33] J.-G. Dumas and D. Duval. Towards a diagrammatic modeling of the LinBox C++ linear algebra library. *ArXiv Computer Science e-prints*, Oct. 2005.

[34] J.-G. Dumas, P. Elbaz-Vincent, P. Giorgi, and A. Urbanska. Parallel computation of the rank of large sparse matrices from algebraic k-theory. 2007.

[35] J.-G. Dumas, T. Gautier, and C. Pernet. Finite field linear algebra subroutines. In Mora [91], pages 63–74.

[36] J.-G. Dumas, P. Giorgi, and C. Pernet. Ffpack: finite field linear algebra package. In Gutierrez [61], pages 119–126.

[37] J.-G. Dumas, F. Heckenbach, D. Saunders, and V. Welker. Computing simplicial homology based on efficient smith form algorithms. *Algebra, Geometry and Software Systems*, 2003.

[38] J.-G. Dumas, B. D. Saunders, and G. Villard. Integer smith form via the valence: Experiments with large sparse matrices from homology.

[39] J.-G. Dumas, B. D. Saunders, and G. Villard. On efficient sparse integer matrix Smith normal form computations. *Journal of Symbolic Computations*, 32(1/2):71–99, jul–aug 2001.

[40] J.-G. Dumas and G. Villard. Computing the rank of sparse matrices over finite fields. In Ganzha et al. [50], pages 47–62.

[41] Dumas, Jean Guillaume and Thierry Gautier and Mark Giesbrecht and Pascal Giorgi and Bradford Hovinen and Erich Kaltofen and Saunders, David B. and Will J. Turner and Gilles Villard. LinBox: A Generic Library for Exact Linear Algebra. In *Proceedings of ICMS'2002 : International Congress of Mathematical Software*, pages 17–19, Beijing, China, aug 2002.

[42] A. Duran, D. Saunders, and Z. Wan. Rank of sparse 0,1,-1 matrices. *Proceeding of the SIAM International Conference on Applied Linear Algebra*, 2003.

[43] W. Eberly, M. Giesbrecht, P. Giorgi, A. Storjohann, and G. Villard. Solving sparse rational linear systems. In Dumas [32], pages 63–70.

[44] W. Eberly, M. Giesbrecht, and G. Villard. On computing the determinant and Smith form of an integer matrix. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 675–687. IEEE Computer Society, 2000.

[45] W. Eberly and E. Kaltofen. On randomized lanczos algorithms. In Küchlin [84].

[46] P. Elbaz-Vincent. Perfects lattices, homology of modular groups and algebraic k-theory. *Oberwolfach Reports (OWR)*, 2, 2005. based on joint work with H. Gangl and C. Soulé.

[47] P. Elbaz-Vincent, H. Gangl, and C. Soulé. Perfect forms, cohomology of modular groups and k-theory of integers. in preparation.

[48] X. G. Fang and G. Havas. On the worst-case complexity of integer gaussian elimination. In Küchlin [84].

[49] M. A. Frumkin. Polynomial time algorithms in the theory of linear diophantine equations. *Lecture Notes in Computer Science*, 1977.

[50] V. G. Ganzha, E. W. Mayr, and E. V. Vorozhtsov, editors. *CASC'2002. Proceedings of the Fifth International Workshop on Computer Algebra in Scientific Computing, Yalta, Ukraine*. Technische Universität München, Germany, Sept. 2002.

[51] M. R. Garey, D. S. Johnson, and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.

[52] T. Gautier, X. Besseron, and L. Pigeon. Kaapi: A thread scheduling runtime system for data flow computations on cluster of multi-processors.

[53] M. Giesbrecht. Fast computation of the smith normal form of an integer matrix.

[54] M. Giesbrecht. Probabilistic computation of the smith normal form of a sparse integer matrix. In *ANTS*, pages 173–186, 1996.

[55] M. Giesbrecht. Fast computation of the smith form of a sparse integer matrix. *Computational Complexity*, 10(1):41–69, 2001.

[56] O. Goldreich. Notes on levin's theory of average-case complexity. Technical report, Electronic Colloquium on Computational Complexity, 1997.

[57] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[58] F. Goodman. *Algebra. Abstract and Concrete*. Prentice Hal, 1997.

[59] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete mathematics*. Addison-Wesley, 2 edition, 1994.

[60] M. Grandis. Combinatorial homology in a perspective of image analysis. Technical report, Univ. Genova, 1999.

[61] J. Gutierrez, editor. *ISSAC'2004. Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation, Santander, Spain.* ACM Press, New York, July 2004.

[62] J. L. Hafner and K. S. McCurley. Asymptotically fast triangularization of matrices over rings. *SIAM J. Comput.*, 20(6):1068–1083, 1991.

[63] A. Hatcher. *Algebraic Topology.* Cambridge University Press, 2002.

[64] G. Havas, D. F. Holt, and S. Rees. Recognizing badly presented z-modules. *Linear Algebra and its Applications*, 192:137, 1993.

[65] M. Hochster. CohenŰmacaulay rings, combinatorics, and simplicial complexes. *Lect. Notes in Pure and Appl. Math*, 26:171–223, 1977.

[66] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition).* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[67] J. Hromkovic and I. Zámecniková. *Design and Analysis of Randomized Algorithms: Introduction to Design Paradigms (Texts in Theoretical Computer Science. An EATCS Series).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[68] O. Ibarra, S. Moran, and R.Hui. A generalization of the fast lup matrix decomposition algorithm and applications. *Journal of Algorithms*, 3(1), 1982.

[69] C. Iliopoulos. Worst-case complexity bounds on algorithms for computing the canonical structure of finite abelian groups and the hermite and smith forms of an integer matrix. 1989.

[70] J. G. J. von Gathen. *Modern Computer Algebra.* Cambridge University Press, 1999.

[71] D. S. V. W. Jean-Guillaume Dumas, Frank Heckenbach. Simplicial homology - a proposed share package for gap, 2000.

[72] T. Kaczyński, K. Mischaikow, and M. Mrozek. *Computational Homology.* Springer, 2004.

[73] T. Kaczyński, M. Mrozek, and M. Ślusarek. Homology computation by reduction of chain complexes. *Computers and Mathematics*, 35(4):59–70, 1998.

[74] G. Kalai. Algebraic shifting. In *In Computational Commutative Algebra and Combinatorics*, pages 121–163, 2001.

[75] E. Kaltofen. An output-sensitive variant of the baby steps/giant steps determinant algorithm. In Mora [91].

[76] E. Kaltofen and A. Lobo. On rank properties of Toeplitz matrices over finite fields. In Lakshman [85], pages 241–249.

[77] E. Kaltofen and B. D. Saunders. On Wiedemann's method of solving sparse linear systems. In *Applied Algebra, Algebraic Algorithms and Error–Correcting Codes (AAECC '91)*, volume 539 of *Lecture Notes in Computer Science*, pages 29–38, Oct. 1991.

[78] E. Kaltofen and G. Villard. Computing the sign or the value of the determinant of an integer matrix, a complexity survey. *Journal of Computational and Applied Mathematics*, pages 133–146, 2004.

[79] E. Kaltofen and G. Villard. On the complexity of computing determinants. *Computational Complexity*, pages 91–130, 2005.

[80] R. Kannan and A. Bachem. Polynomial algorithms for computing the smith and hermite normal forms of an integer matrix. *Journal of Computation*, 1979.

[81] M. Kauers, editor. *ISSAC'2005. Proceedings of the 2005 International Symposium on Symbolic and Algebraic Computation, Beijing, China*. ACM Press, New York, July 2005.

[82] S. Khodadad and M. Monagan. Fast rational function reconstruction. In Dumas [32], pages 184–190.

[83] D. E. Knuth. *The art of computer programming, Vol. 1: Fundamental algorithms*. Addison-Wesley, 1981.

[84] W. W. Küchlin, editor. *ISSAC'97. Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation, Maui, Hawaii*. ACM Press, New York, July 1997.

[85] Y. N. Lakshman, editor. *ISSAC'96. Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation, Zurich, Switzerland*. ACM Press, New York, July 1996.

[86] L. A. Levin. Average case complete problems. 1986.

[87] J. W. Lewis. Inversion of tridiagonal matrices. *Numerische Mathematik*, 38(3):333–345, 1982.

[88] D. Lichtblau. Half-gcd and fast rational recovery. In Kauers [81], pages 231–236.

[89] M. Mignotte. *Math matiques pour le calcul formel.*

[90] M. Monagan. Maximal quotient rational reconstruction: An almost optimal algorithm for rational reconstruction. In Gutierrez [61], pages 243–249.

[91] T. Mora, editor. *ISSAC'2002. Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, Lille, France*. ACM Press, New York, July 2002.

[92] R. Motwani and P. Raghavan. *Randomized Algorithms*. Stanford University, California, 1995.

[93] M. Mrozek and B. Batko. Coreduction homology algorithm. *Discrete and Computational Geometry*, 2008.

[94] M. Mrozek, P. Pilarczyk, and N. Zelazna. Homology algorithm based on acyclic subspace. *Computers and Mathematics*, pages 2395Ú–2412, 2008.

[95] T. Mulders. Certified dense linear system solving. *Journal of Symbolic Computation*, 2004.

[96] T. Mulders. Certified sparse linear system solving. *Journal of Symbolic Computation*, 2004.

[97] T. Mulders and A. Storjohann. Diophantine linear system solving. In Dooley [30], pages 181–188.

[98] T. Mulders and A. Storjohann. Rational solutions of singular linear systems. In C. Traverso, editor, *Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation, St Andrews, Scotland*, pages 242–249. ACM Press, New York, Aug. 2000.

[99] J. Munkres. *Elements of Algebraic Topology.* Addison Wesley, Menlo Park, 1984.

[100] D. R. Musser. Introspective sorting and selection algorithms. *Software–Practice and Experience*, 1997.

[101] M. Niethammer, A. Stein, W. Kalles, P. Pilarczyk, K. Mischaikow, and A. Tannenbaum. Analysis of blood vessel topology by cubical homology. pages II: 969–972, 2002.

[102] Z. Olesh and A. Storjohann. The vector rational function reconstruction problems. In I. S. K. amd Eugene V. Zima, editor, *Proceedings of the Waterloo Workshop on Computer Algebra: devoted to the 60th birthday of Sergei Abramov*, pages 137–149. World Scientific, 2007.

[103] V. Pan. Computing the determinant and the characteristic polynomial of a matrix via solving linear systems of equations. *Inform. Process. Lett.*, pages 71–75, 1988.

[104] V. Pan. Can we optimize toeplitz/hankel computations? In Ganzha et al. [50].

[105] V. Y. Pan. Superfast algorithms for singular toeplitz-like matrices. Technical Report TR 2004015, CUNY, 2004.

[106] V. Y. Pan, B. Murphy, R. E. Rosholt, and X. Wang. Nearly optimal toeplitz/hankel computations. Technical Report TR 2002017, CUNY, 2002.

[107] V. Y. Pan, B. J. Murphy, R. E. Rosholt, and X. Wang. Toeplitz and hankel meet hensel and newton modulo a power of two. Technical Report TR 2005008, CUNY, 2005.

[108] C. H. Papadimitriou. *Computational Complexity.* 1994.

[109] C. Pernet. *Algèbre linéaire exacte efficace: le calcul du polynôme caractéristique.* PhD thesis, Universite Joseph Fourier, 2006.

[110] C. Pernet and A. Storjohann. Faster algorithms for the characteristic polynomial. ACM Press, New York, July 2007.

[111] W. A. Rudich Steve, editor. *Computational complexity theory*, volume 10. IAS Park city mathematics series, 2004.

[112] B. D. Saunders and Z. Wan. Smith normal form of dense integer matrices, fast algorithms into practice. In Gutierrez [61], pages 274–281.

[113] B. D. Saunders and Z. Wan. An engineered algorithm for the smith form of an integer matrix. *ACM Transactions on Computational Logic*, 2006.

[114] J. E. Savage. *Models of Computation: Exploring the Power of Computing.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[115] U. Schöning and R. Pruim. *Gems of Theoretical Computer Science.* Springer Verlag, 1998.

[116] H. M. S. Smith. On systems of indeterminate equations and congruences. *Philos. Trans.*

[117] D. Steffy. Exact solutions to linear systems of equations using output sensitive lifting, 2009.

[118] D. Steffy and W. Cook. Solving very sparse rational systems of equations, 2009.

[119] D. Stehlé and P. Zimmermann. A binary recursive gcd algorithm. In *Proceedings of the 6th Algorithmic Number Theory Symposium (ANTS VI)*, volume 3076 of *Lecture Notes in Computer Science*, pages 411–425. Springer-Verlag, 2004.

[120] A. Storjohann. Faster algorithms for integer lattice basis reduction. Technical Report 249, Departement Informatik, ETH Zurich, 1996.

[121] A. Storjohann. Near optimal algorithm for computing smith normal forms of integer matrices. In Lakshman [85].

[122] A. Storjohann. The shifted number system for fast linear algebra on integer matrices. *Journal of Complexity*, 21(4):609–650, 2005.

[123] A. Storjohann, P. Giorgi, and Z. Olesh. Implementation of a las vegas integer matrix determinant algorithm. In *ECCAD'05: East Coast Computer Algebra Day*, 2005.

[124] A. Storjohann and T. Mulders. Fast algorithms for linear algebra modulo n. In *ESA'98: Proceedings of 6th Annual European Symposium on Algorithms*, pages 139–150, 1998.

[125] T. G. Team. Gmp manual.

[126] W. J. Turner. *Black box linear algebra with the linbox library.* PhD thesis, 2002. Chair-Erich Kaltofen.

[127] Z. Wan. *Computing the Smith Forms of Integer Matrices and Solving Related Problems.* PhD thesis, University of Delaware, 2005.

[128] Z. Wan. An algorithm to solve integer linear systems exactly using numerical methods. *Journal of Symbolic Computation*, 41(6):621–632, 2006.

[129] P. Wang. p-adic algorithm for univariate partial fractions. *Proc. of the 4th ACM Symp. on Symb. and Alg. Comp*, pages 212–217, 1981.

[130] P. S. Wang, M. J. T. Guy, and J. H. Davenport. p-adic reconstruction of rational numbers. *SIGSAM Bulletin*, 16(2), 1982.

[131] X. Wang and V. Pan. Acceleration of euclidean algorithm and rational number reconstruction. *SIAM J. of Computing*, 32,2:548–556, 2003.

[132] S. Watt, editor. *PASCO '07: Proceedings of the 2007 international workshop on Parallel symbolic computation, PASCO 2007*, New York, NY, USA, 2007. ACM.

[133] D. H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, 32(1):54–62, Jan. 1986.

[134] M. Yannakakis. Computing the minimum fill-in is np-complete. *SIAM J. Alg. Disc. Meth*, 1981.