Symbolic Computation Techniques for Solving Large Expression Problems from Mathematics and Engineering

(Spine title: Symbolic Techniques for Reducing Expression Swell)

(Thesis format: Integrated-Article)

by

Wenqin Zhou

Graduate Program
in
Applied Mathematics

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

Faculty of Graduate Studies
The University of Western Ontario
London, Ontario, Canada

© Wenqin Zhou 2007

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-30874-5
Our file Notre référence
ISBN: 978-0-494-30874-5

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

THE UNIVERSITY OF WESTERN ONTARIO
FACULTY OF GRADUATE STUDIES

CERTIFICATE OF EXAMINATION

Supervisor	Examiners
Dr. David J. Jeffrey	Dr. Robert M. Corless
Co-Supervisor	Dr. Marc Moreno Maza
Dr. Gregory J. Reid	Dr. Jerzy M. Floryan
	Dr. Ilias Kotsireas

The thesis by

WENQIN ZHOU

entitled:

SYMBOLIC COMPUTATION TECHNIQUES FOR SOLVING LARGE EXPRESSION PROBLEMS FROM MATHEMATICS AND ENGINEERING

is accepted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Date April 19, 2007

Dr. Ian McLeod

Chair of the Thesis Examination Board

ABSTRACT

This thesis studies the use of computer algebra methods to solve some large-expression problems from mathematics and engineering. We give several strategies for solving problems from symbolic linear algebra and dynamic systems.

First, we describe new forms for fraction-free LU factoring and QR factoring. These new forms keep both the computation and the output results in the same domain as the input domain and thereby increase the computational efficiency in applications by delaying the appearance of quotients of the input data. To compute the new forms, we use a fraction-free variant of Gaussian elimination to control the growth of matrix entries. We give a complexity analysis for standard domains.

Secondly, we propose a general method, *hierarchical representation and signature computing for zero testing,* to deal with problems with intermediate or inherent expression swell. For instance, when we use Gaussian elimination to solve large symbolic linear equations, the resulting large expressions can be handled using our general method. We implement a version of LU factoring using hierarchical representation with signature computing for zero testing. The LU factoring is the standard one, rather than the fraction-free one described above. We prove that the improved algorithm is much faster than the classical LU factoring algorithm using Gaussian elimination and give associated time complexity analysis and experimental results.

Besides large expression problems from linear algebra, we also explore large expression problems from engineering, especially those arising from analyzing and solving multibody dynamic systems and limit cycle computations. We define a new concept, *implicit reduced involutive form,* to cope with large expression problems

resulting from symbolically pre-processing systems of differential algebraic equations (DAE). We also show how symbolic pre-processing can be combined with numerical integration to solve a problem from limit cycle computations which could not be directly solved because of large expression swell.

The techniques we develop in this thesis are quite general and can be easily applied to other similar areas, such as computing determinants and solving more general DAE models.

Keywords: Large Expression Management, Differential Algebraic Equations, RIF-SIMP, LARGEEXPRESSIONS, Multibody Dynamic System, Differentiation and Elimination, Hierarchical Representations, Signature, LU Symbolic Decomposition, Time Complexity, Limit Cycle, Computer Algebra.

DEDICATION

To my father 周志金，my mother 朱玲娅， and my boyfriend Éric Schost.

THE CO-AUTHORSHIP STATEMENT

Chapter 2 is joint work with Dr. Jeffrey.

Chapter 3 is joint work with Dr. Carette, Dr. Jeffrey and Dr. Monagan, which was published in LNAI 4120, 254-268, 2006.

Chapter 4 is joint work with Dr. Jeffrey, Dr. Reid, Dr. Schmitke and Dr. McPhee, which was published in LNCS 3519, 31-43, 2004.

Chapter 5 is joint work with Dr. Jeffrey and Dr. Reid, which has been accepted for publication in Symbolic-Numeric Computation, 335-347, Birkhäuser Verlag Basel/Switzerland, 2007.

Chapter 6 is joint work with Dr. Yu and Dr. Schost.

ACKNOWLEDGEMENTS

First of all, I wish to thank my two PhD supervisors Dr. Jeffrey and Dr. Reid for their guidance, encouragement and support over the years.

I would also like to thank ORCCA lab for providing me the best research environment. Special thanks go to Dr. Moreno Maza, Dr. Watt and Dr. Labahn. I would like to thank all the professors and staff in the Department of Applied Mathematics for their help, especially Dr. Corless, Dr. Yu, Dr. Essex, Dr. Davison, Ms. Malone, Ms. Kager and Ms. Mckenzie.

I am grateful to Dr. Kotsireas for agreeing to be the external examiner of this thesis and Dr. Floryan for agreeing to be the university examiner of this thesis.

There are lots of friends I would like to thank, especially Qing, Zhen, Hui, Tianfu, Shudan, Wenyuan, Shuo, and Robin. I thank my parents for their endless love and support, my sister for her support which made it possible for me to devote myself on study and my boyfriend for his love and caring.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Background

In computer algebra, expression swell problems are very common and are often impossible to avoid. Expression swell can take several forms, and is called by different terms depending upon where in the calculation it appears. If a calculation has a relatively compact output, but during the calculation large expressions are generated, then the problem is called intermediate expression swell. However, if the outputs obtained at the end of a computation remain impractically large then we have what is called inherent expression swell.

This thesis studies a number of problems in which inherent expression swell is a problem. The challenges are in the first place to complete a calculation, and then to find ways of continuing to work with the large expressions that have been generated. We will illustrate our techniques for solving large expression problems by tackling large expression swell problems from linear algebra, by alleviating intermediate expression swell from computations of the resolution of systems of differential equations, and by solving large systems arising from computing limit cycles.

1.2 Literature Review

In this section, we shall review the related work on large expression management and its applications.

1.2.1 Large Expression Management and Signature computation

One type of large expression problem arises because of the desire to avoid fractions during symbolic computation. For example, when Gaussian elimination is applied to a matrix, it is usually presented in the following way.

- For each column in turn, select a nonzero pivot and then reduce the pivot to 1 by division.
- Use this new row to reduce elements below the pivot to zero.

The division immediately introduces fractions or rational functions into any calculation. To avoid the division and reduce terms to zero, cross-multiplying is applied. This avoids fractions but leads to polynomial growth in the size of terms. A similar effect applies to the Gram-Schmidt process.

In order to reduce the inherent expression swell in these processes, fraction-free algorithms with exact division were developed. Bareiss [2] developed a fraction-free version of Gaussian elimination (which he credited to Jordan), while Erlingsson, Kaltofen & Musser [10] developed and implemented an exact division method for their version of Gram-Schmidt process, which they credit to [26]. More recently, Beckermann, Cheng and Labahn published their 2006 paper [3] which describes an algorithm for performing fraction-free row reduction of a matrix of Ore polynomials.

A known factor is predicted and removed efficiently. They applied the same method to matrices of skew polynomials [4].

It is of interest to combine these ideas with Turing's description of Gaussian elimination as a matrix factoring [47], which is well established in linear algebra, particularly numerical linear algebra [46]. Even though the factoring takes a number of different forms (Cholesky, Doolittle-LU, Crout-LU, Turing), only relatively recently has there been an attempt to define a fraction-free form [8, 32]. Some of the so-called fraction-free methods are only partially fraction-free. Here we consider defining a completely fraction-free form. Although the growth of entries is reduced by the fraction-free algorithms referenced above, the growth can be still too large for larger matrices. A more general approach is to use different data representations with an associated probabilistic zero testing. Various tools to implement these ideas have been developed independently, and yet they show similarities in design and intent. For example, in MAPLE the following independent commands address similar aspects of large expressions. The MAPLE command `freeze` replaces an expression by a name, and `thaw` reverses this. The MAPLE command `Veil` in the LARGEEXPRESSIONS package hides a sequence of expressions behind user-defined labels. The `Unveil` command can be used to output the expressions as a computation sequence. On the other hand, the MAPLE command `optimize` in the CODEGEN package attempts to identify common subexpressions for optimization. The Maple command `CompSeq` is a placeholder for representing a computation sequence. All these commands are described in more detail in the MAPLE online help system.

More generally, the following topics have been pursued independently by a number of researchers: computation sequences, straight line programs or directed acyclic graphs, hierarchical evaluation. The common subexpression identification in both

the MAPLE `prettyprinter` command and the `optimize` command search a large expression for common subexpressions [28]. Giusti et al. compute the determinant of polynomial matrices using the MAPLE DAG structure and the Berkowitz algorithm [13]. Freeman, Imirzian, Kaltofen, Lakshman have their DAGWOOD software to compute with straight-line programs [11]. An example of computation sequences was given by Zippel in 1993 [56]. The term *Large Expression Management* (LEM) was first introduced by Corless, Jeffrey, Monagan, Pratibha in 1997 [9]. Their paper was the precursor of the LARGEEXPRESSIONS package in MAPLE. Applications of these ideas are given in [1, 50]. In this thesis, we develop new algorithms to control expression growth during the generation phase based on ideas originating in [9].

An important problem in the manipulation of large symbolic expressions is the problem of testing when an expression is zero. A direct approach to this problem is to reduce an expression to a *normal form* [12]. A normal form is one that is guaranteed to be zero if the original expression is zero. For example, if the expression to be tested is a polynomial, one would expand it and then collect like terms together. Now consider the expression

$$(2781 + 8565x - 4704x^2)^{23}(1407 + 1300x - 1067x^2)^{19}$$
$$-\alpha(1809 + 9051x + 9312x^2)^{19}(2163 - 2162x + 539x^2)^{19}(27 + 96x)^4(103 - 49x)^4.$$

This expression is zero for the case $\alpha = 1$, but if we expand it with α only as a symbol, we will not be able to get any shorter expression. Instead we will have an expression which takes up a great deal of space in memory. Similar things might happen if we expand an expression when it is represented in computation sequence or straight line program. Therefore we conclude that normal forms will run the risk of filling memory in attempting to recognize zero.

In this thesis, we use signature computations for zero testing in a similar way to the probabilistic identity testing approach of Zippel-Schwartz [57, 42], and to the approach of `testeq` in MAPLE by Gonnet *et al.* [14, 15, 29, 30]. The results of Zippel-Schwartz for polynomials were extended to other functions in [14, 15, 31]. Other important references on this topic are [27, 16, 23, 24]. Applications of this approach include determining whether a matrix (of polynomials) is singular, and determining the rank of a polynomial matrix [25].

1.2.2 Reduced Involutive Form and its Applications

We now briefly describe the *differentiation and elimination* algorithm developed by Reid, Rust and Wittkopf. The related theoretical work is described in detail by Reid *et al.* [34, 33, 35], Rust *et al.* [39, 40] and Wittkopf *et al.* [48, 49]. The main steps of this algorithm are described in Figure 1.2.1.

For an input system of differential algebraic equations (DAE), we first define a ranking for the whole system. The ranking of functions and derivatives, which is like the ordering of unknowns (columns) in Gaussian elimination, is fundamental to the process of differential elimination. For more details about the rankings of derivatives for elimination algorithms, please see [40, 39]. Secondly we classify the whole system into two parts based on their highest derivatives with respect to the ranking: its leading linear part $L = 0$ and its leading non-linear part $N = 0$. Thirdly, we solve the leading linear part for its highest derivatives. This is equivalent to solving a linear system with an associated case splitting analysis based on whether the pivots are zero or nonzero. We differentiate the equations of the leading non-linear part and reduce them with respect to the all leading linear equations and the leading non-linear equations to find any new constraints for the system. This process can be viewed as

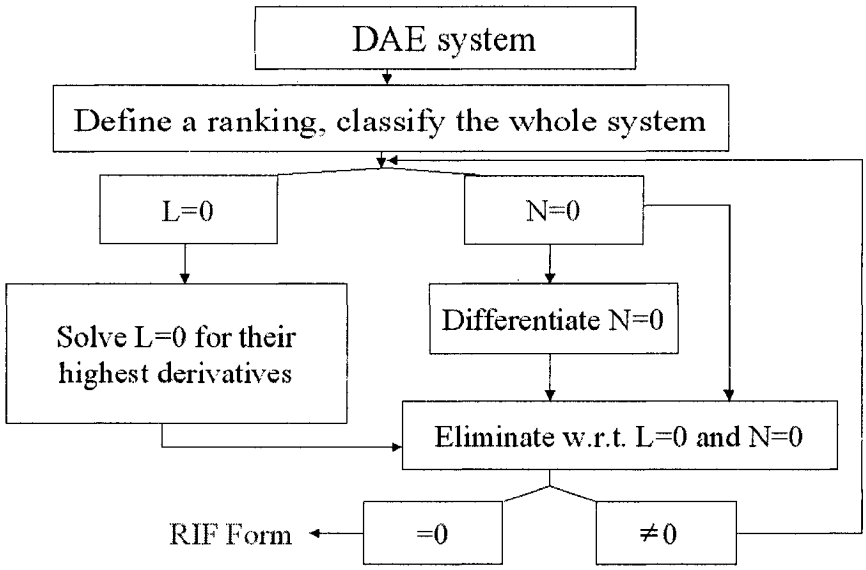

Figure 1.2.1: Differentiation and Elimination Algorithm. The system consists of equations that are leading linear ($L = 0$), and equations that are nonlinear ($N = 0$).

a differential analog of S-polynomials from Gröbner basis computations [49]. Case splitting based on the coefficients (pivots) of the leading derivatives yields a binary tree of cases. In the leading linear part, the leading coefficients are usually nonlinear. For example, $u_x(u_x-1)u_{xx}-u^2=0$ has leading derivative u_{xx} and leading coefficient $u_x(u_x-1)$. Thus we must consider the cases $u_x(u_x-1)=0$ and $u_x(u_x-1)\neq 0$ separately. Fourthly we check the output of the elimination. If we find non-zero constraints, we include the new equations into the system and go back to the second step. Otherwise, it means we have a reduced involutive form for the input system.

This algorithm is implemented in the MAPLE package RIFSIMP [48]. Similar implementations are available in other computer algebra systems, such as the MAPLE package DIFFALG [5, 17, 18, 6, 19, 7, 20, 21, 22]. See [35] and references therein for the proofs of the following theorems.

Theorem 1.2.1. ***[Output RIF form]*** *For a given input analytic DAE system and a given ranking, the output of the* RIFSIMP *algorithm consists of a finite number of cases where each case is in RIF form. That is for each case there exist unique lists of all derivatives (including dependent variables) v and w, where w is lower in ranking than v, such that the output of the algorithm of Figure 1.2.1 has the structure*

$$v = f(t, w)$$

subject to a list of constraint equations and inequations

$$g(t, w) = 0, \quad h(t, w) \neq 0$$

Theorem 1.2.2. ***[Existence and Uniqueness]*** *Given a set of initial conditions (t^0, w^0), such that*

$$g(t^0, w^0) = 0 \ , \quad h(t^0, w^0) \neq 0 \ ,$$

there is a local analytic solution to the input DAE system with these initial conditions.

A general multibody system is a system consisting of one or more bodies that are interconnected by joints. It is described by a DAE system with a high *differential index*. Multibody systems can be robots, biomechanical systems, cars, trains, satellites and so on [41]. There are several software products for modelling multibody systems. These commercial programs use numeric data structures and absolute coordinates for multibody system modelling [37]. Examples include ADAMS by MSC and Working Model Software, DADS by CADSI, SimMechanics by MathWorks, MECANO by Samtech, etc. All these software products use numerical techniques. Such numeric modelling has some disadvantages, as discussed in [45, 37].

In contrast, DYNAFLEX (`http://real.uwaterloo.ca/~dynaflex/`) is a symbolic Maple package for modelling general multibody dynamic systems. Modelling a

mechanism in a purely symbolic way has the advantage that symbolic equations can be visualized, manipulated and communicated. Secondly, using symbolic modelling we can sometimes gain physical insight and find analytical solutions. Symbolic models can also facilitate subsequent real-time simulation of the system. It is suitable for applications in industry, research and education [38].

The general form of the equations that govern multibody dynamic systems is

$$M(t,q,\dot{q})\ddot{q} + \Phi_q^T \lambda = F(t,q,\dot{q}) \ , \tag{1.2.1}$$

$$\Phi(t,q) = 0 \ . \tag{1.2.2}$$

Here, q is a vector of generalized co-ordinates, $M(t,q,\dot{q})$ is the mass matrix, Φ is a vector of the constraint equations, λ is a vector of Lagrange multipliers and F is a force vector [43, 44]:

$$\Phi = \begin{bmatrix} \Phi^1 \\ \vdots \\ \Phi^m \end{bmatrix}, \qquad \Phi_q = \begin{bmatrix} \Phi^1_{q_1} & \cdots & \Phi^1_{q_n} \\ \vdots & \vdots & \vdots \\ \Phi^m_{q_1} & \cdots & \Phi^m_{q_n} \end{bmatrix}, \qquad \lambda = \begin{bmatrix} \lambda^1 \\ \vdots \\ \lambda^m \end{bmatrix} .$$

There are two main kinds of system. The first one is the open-loop system, which has no algebraic constraints and whose equations are only ordinary differential equations of form $M(t,q,\dot{q})\ddot{q} = F(t,q,\dot{q})$. The second one is the closed-loop system which has algebraic constraints and whose equations are differential algebraic equations. We consider the problem of analyzing both open and closed loop systems.

In this thesis, we apply differentiation and elimination algorithm to reduce the index of the general multibody DAE system and obtain its involutive form. Then the system can be either numerically solved or in some special cases, symbolically solved [51].

1.3 Contributions of this Thesis

This thesis mainly demonstrates techniques and strategies for dealing with some typical large expression swell problems by solving several different mathematical and engineering problems involving large expressions. We develop algorithms for the symbolic resolution of linear systems and for the formal treatment of systems of Differential Algebraic Equations (DAE's). We also develop symbolic techniques for coping with large expressions from solving DAE's, linear systems and large polynomials. We put them into practice by carrying out numerical simulations.

1. We present new algorithms for symbolically solving linear equations.

   In order to solve linear equations in one domain and alleviate expression swell problems from symbolic computations, in Chapter 2 we define a *completely fraction-free LU factoring* [54], together with a fraction-free backward substitution and a fraction-free forward substitution. We also apply our completely fraction-free LU factoring to obtain a new fraction-free QR factoring.

   In Chapter 3 we propose the use of different data representations, namely *hierarchical representations*, and a modular method, *signature computation*, for a probabilistic zero testing to solve large expression problems resulting from matrix inversion. We also implemented a new efficient LU factoring algorithm which involves hierarchical representations and signature computation for zero testing. In view of the complexity analysis and benchmarks, the new algorithm demonstrates a promising methodology [55].

2. We develop a new concept for improving the existing algorithms for RIF-reduction.

In Chapter 4, we analyze multibody dynamic systems with the differentiation and elimination algorithm. For an open-loop system, we begin with the MAPLE package RIFSIMP, then numerically or symbolically solve the system, including its generic case and special cases from case splitting. Using a special property of the DAE system from DYNAFLEX, we develop a new concept, *implicit-RIF form*, to analyze this dynamic system [53, 51]. The implicit-RIF form avoids the expression swell coming from matrix inversion. It also contains all the hidden constraints in the DAE system which are essential to have consistent initial conditions for numerical simulations.

3. We apply computer algebra to solve multibody dynamic systems and limit cycle computations.

   In Chapter 5, we apply the symbolic LU factoring algorithm from Chapter 3, and our general solution in implicit-RIF form which is developed in Chapter 4, to get an explicit RIF form for a multibody dynamic system. We also numerically simulate the system [52].

   In Chapter 6, we solve a large polynomial system arising from computing limit cycles from planar vector fields. These systems cannot be directly solved with purely numerically methods. We use Gröbner basis computation and make a primary decomposition of the system. Finally we use numerical methods to complete our solutions.

All the chapters can be connected as in Figure 1.3.2. The details are given below.

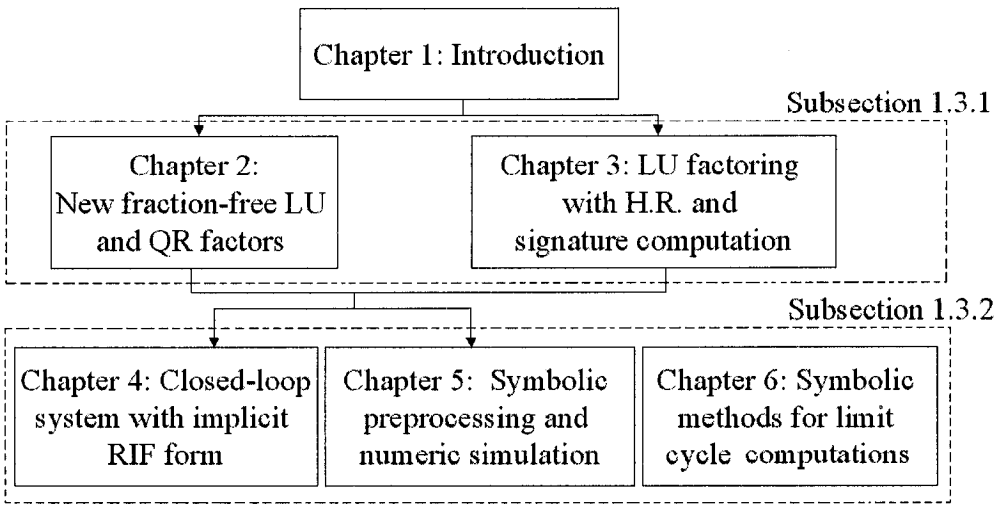

Figure 1.3.2: Outline of the Thesis

1.3.1 Linear Algebra with Large Expressions

In Chapter 2, we give a new completely fraction-free LU factoring form, which keeps the computation in one domain.

Theorem 1.3.1. *Let $\mathbb{I}$ be an integral domain and let A be a full-rank matrix in $\mathbb{I}^{n\times m}$ with $n \leq m$, then A may be written*

$$PA = LD^{-1}U,$$

where

$$L = \begin{bmatrix} p_1 & & & & \\ L_{2,1} & p_2 & & & \\ \vdots & \vdots & \ddots & & \\ L_{n-1,1} & L_{n-1,2} & \cdots & p_{n-1} & \\ L_{n,1} & L_{n,2} & \cdots & L_{n,n-1} & 1 \end{bmatrix},$$

$$D = \begin{bmatrix} p_1 & & & & \\ & p_1 p_2 & & & \\ & & \ddots & & \\ & & & p_{n-2}p_{n-1} & \\ & & & & p_{n-1} \end{bmatrix},$$

$$U = \begin{bmatrix} p_1 & U_{1,2} & \cdots & U_{1,n-1} & U_{1,n} & \cdots & U_{1,m} \\ & p_2 & \cdots & U_{2,n-1} & U_{2,n} & \cdots & U_{2,m} \\ & & \ddots & \vdots & \vdots & \vdots & \vdots \\ & & & p_{n-1} & U_{n-1,n} & \cdots & U_{n-1,m} \\ & & & & p_n & \cdots & U_{n,m} \end{bmatrix},$$

P is a permutation matrix, L and U are triangular matrices as shown, and the pivots p_is that arise are in $\mathbb{I}$. The pivot p_n is also the determinant of the matrix A.

The new formula is much more compact than the one given by Corless and Jeffrey in 1997 [8]. Because it performs no gcd computation during factorization, this fraction-free factorization is better than the existing partially fraction-free LU factoring offered by MAPLE.

Theorem 1.3.2. *For a matrix $A = [a_{i,j}]_{n\times n}$ with entries in $\mathbb{K}[x]$, if every $a_{i,j}$ has*

degree less than d, the time complexity of completely fraction-free LU factoring for A is bounded by $O(n^3M(nd))$ *operations in* $\mathbb{K}$.

For a matrix $A = [a_{i,j}]_{n\times n}$ *with entries in* $\mathbb{Z}$, *if every* $a_{i,j}$ *has its length bounded by* ℓ, *the time complexity of completely fraction-free LU factoring for A is bounded by* $O(n^3M(n\log n + n\ell))$ *word operations.*

For a matrix $A = [a_{i,j}]_{n\times n}$ *with univariate polynomial entries in* $\mathbb{Z}[x]$, *if every* $a_{i,j}$ *has its degree less than d and has its length bounded by* ℓ, *the time complexity of completely fraction-free LU factoring for A is bounded by* $O(n^3M(n^2d\ell + nd^2))$ *word operations.*

We compare this time complexity with the time complexity of the existing partially fraction-free LU factoring algorithm. We also give benchmarks to demonstrate the efficiency of different algorithms, to show the relations between the running time and the size of the matrix and the relations between the running time and the length of the entries of the matrix. The details are given in Chapter 2, Section 2.3 and Section 2.4.

In the same chapter, in Section 2.5, we apply our new completely fraction-free LU factoring, together with fraction-free backward substitution and fraction-free forward substitution, to solve linear systems in a single domain. We also apply this completely fraction-free LU factoring to obtain a new fraction-free QR factoring. However, the entries of output matrices can still be very large. Chapter 3 states another new approach for solving a linear system, which is to use hierarchical representations with signature computations to do LU factoring. We give our definitions of hierarchical representation and signatures.

Definition 1.3.3. [Exponential Polynomial] An exponential polynomial is an element of $\mathbb{Z}[X_1,\ldots,X_n,a_1^{X_1},\ldots,a_n^{X_n}]$, where $a_1,\ldots,a_n$ are positive real numbers.

Definition 1.3.4. [Hierarchical Representation] A hierarchical representation (HR) over a domain $\mathbb{K}$ and a set of independent variables $\{x_1, ..., x_m\}$ is an ordered list $[S_1, S_2, ..., S_l]$ of symbols, together with an associated list $[D_1, D_2, ..., D_l]$ of definitions of the symbols. For each S_i with $i \geq 1$, the definition D_i has the form $S_i = f_i(\sigma_{i,1}, \sigma_{i,2}, ..., \sigma_{i,k_i})$ where $f_i \in \mathbb{K}[\sigma_{i,1}, ..., \sigma_{i,k_i}]$, and for each $\sigma_{i,j}$ is either a symbol in $[S_1, S_2, ..., S_{i-1}]$ or an exponential polynomial in the independent variables.

In order to test zero for an expression in hierarchical representations, we use the concept of signature from [14] and extend it to exponential polynomials. The restriction to exponential polynomials is not essential. It is chosen for definiteness, and because these functions are the ones actually appearing in our applications. Further, they are a class of functions for which we can define signatures.

Definition 1.3.5. [Signature of an Exponential Polynomial] Let P be in $\mathbb{Z}[X_1, \ldots, X_n, a_1^{X_1}, \ldots, a_n^{X_n}]$, where $X_1, \ldots, X_n$ are independent variables and $a_1, \ldots, a_n$ are positive real numbers. Let p be a prime, $x_1, \ldots, x_n$ be in $\mathbb{Z}/p\mathbb{Z}$, $y_1, \ldots, y_n$ be in $\mathbb{Z}/(p-1)\mathbb{Z}$ and t be a primitive root of p. Then the signature of P is defined below:

$$s(P) = P(x_1, \ldots, x_n, t^{y_1}, \ldots, t^{y_n}) \mod p.$$

Definition 1.3.6. [Signature of a Hierarchical Representation] Let H be a hierarchical representation having independent variables $X_1, \ldots, X_n$, given by lists $[S_1, \ldots, S_\ell]$ and $[D_1, \ldots, D_\ell]$ with the form $S_i = f_i(\sigma_{i,1}, \sigma_{i,2}, ..., \sigma_{i,k_i})$ with $f_i \in \mathbb{Z}[\sigma_{i,1}, ..., \sigma_{i,k_i}]$, and for each $\sigma_{i,j}$ is either a symbol in $[S_1, S_2, ..., S_{i-1}]$ or an exponential polynomial in the independent variables. Let p be a prime, $x_1, \ldots, x_n$ be in $\mathbb{Z}/p\mathbb{Z}$, $y_1, \ldots, y_n$ be in $\mathbb{Z}/(p-1)\mathbb{Z}$ and t be a primitive root of p.

The signature of S_i is defined inductively as follows:

- If $\sigma_{i,j} \in \mathbb{Z}[X_1, \ldots, X_n, a_1^{X_1}, \ldots, a_n^{X_n}]$, then $\delta_{i,j} = s(\sigma_{i,j})$.

- If $\sigma_{i,j}$ is a symbol in $[S_1, S_2, ..., S_{i-1}]$, then we necessarily have $i > 1$. Let $h_{i,j} \in [1, \cdots, i-1]$ be such that $\sigma_{i,j} = S_{h_{i,j}}$, then $\delta_{i,j} = s(\sigma_{i,j}) = s(S_{h_{i,j}})$ which is known by induction assumption.

- We define $s(S_i) = f_i(\delta_{i,1}, \cdots, \delta_{i,k_i}) \mod p$.

The signature of H is defined as $s(H) = [s(S_1), \cdots, s(S_\ell)]$.

This is a modular method and we can use it to test zero with a user-controllable probability. In Chapter 3, Section 3.3 gives a detailed analysis of the probability when an expression with signature is zero while the expression is not zero. Because the signature of an exponential polynomial is equal to the signature of its hierarchical representations, we give a new high-level LU factoring tool using hierarchical representations with signature computation for zero testing. In our new package, we include three kinds of strategies for the user: a pivoting strategy, a veiling strategy and a zero-recognition strategy. The main advantages of our new LU factoring technique are:

- It is fast. Veiling strategies improve the intermediate computation speed and the whole computation speed;

- It is flexible and can be adapted to different applications.

- Hierarchical representations can often lead to a more compact and elegant output.

- Our code can solve a wider class of problems and often reduces intermediate expression swell.

By comparing the time complexity and doing benchmarks, we prove that our new LU factoring algorithm is much faster and better than both the classical LU

factoring by Gaussian elimination and a revised MAPLE LU factoring. Because it can control the size of entries in the matrices L and U and the zero-test is fast, the new LU factoring is also faster than the fraction-free LU factoring given in Chapter 2.

Let us assume we have the following veiling strategy: we veil any expression with an integer coefficient of length larger than c_1, or whose degree in x_i is larger than c_2, where c_1, c_2 are positive constants. In order to prevent the cost from growing exponentially with the number of variables, it is best to choose $c_2 = 1$. Then we have the following time complexity for the LU factoring.

Theorem 1.3.7. *Let $A = [a_{i,j}]$ be an n by n matrix where $a_{i,j} \in \mathbb{Z}[x_1, ..., x_m]$ and let p be the prime used for signature arithmetic. Let $d > \max_{i,j,r} \deg(a_{i,j}, x_r)$ and l bound the length of the largest integer coefficient of the entries of the matrix. Let $T > ld^m$. So T bounds the size of matrix entries.*

The time complexity of LU factoring with the above veiling strategy and modulo p signature computation is $O((Tn^2 + n^3)(\log p)^2)$.

Theorem 1.3.7 shows that the time complexity of symbolic LU factoring improves from a complexity of at least $\Omega(n^{2m+5})$ to $O((Tn^2 + n^3)(\log p)^2)$. Details are given in Chapter 3, Section 3.6 and Section 3.7.

1.3.2 Computer Algebra for Large Expressions from Engineering

We apply the differentiation and elimination algorithm to the multibody dynamic system. In Chapter 4, we discuss the drawbacks of applying RIFSIMP to analyze and solve the DAE system. Typically, when a system gets larger and the size of its mass

matrix gets bigger, solving the leading linear system or inverting the mass matrix to get a symbolic canonical form is harder and sometimes even impossible. We propose a new idea, *implicit-RIF form*, for alleviating this problem.

Definition 1.3.8. [Implicit Reduced Involutive Form] Given a differential algebraic equation system (DAEs), let $\prec$ be a ranking of this DAEs. If there exist derivatives $r_1, ..., r_k$ such that L is leading linear in $r_1, ..., r_k$ with respect to $\prec$ (i.e. $L = A[r_1, \cdots, r_k]^T - b = 0$), N is leading nonlinear and

$$[r_1, \cdots, r_k]^T = A^{-1}b\ ,\quad N = 0\ ,\quad \det(A) \neq 0 \tag{1.3.1}$$

is in reduced involutive form. Then the system $L = 0, N = 0$ is said to be in implicit reduced involutive form (implicit-RIF form).

This form is of interest, since computing A^{-1} can be very expensive. We avoid the computation of a matrix inversion. With this new form, we would not have the large expression problems from the matrix inversion, while we still have all the hidden constraints from the differential algebraic equation system. These constraints are very important since they enable the determination of consistent initial conditions for the system's numeric integration.

For our general multibody dynamic system, we prove that to convert a dynamic system of general form (1.2.1,1.2.2) with non-trivial constraints to implicit-RIF form, one would have to at least differentiate the constraints twice.

Theorem 1.3.9. *Consider the ranking $\prec$ defined by $q \prec \dot{q} \prec \lambda \prec \ddot{q} \prec \dot{\lambda} \prec \dddot{q} \cdots$ where the dependent variables q, λ are ordered lexicographically $q_1 \prec q_2 \prec ...$ and $\lambda_1 \prec \lambda_2 \prec ....$ The systems (1.3.2, 1.3.3, 1.3.4, 1.3.5)*

$$M\ddot{q} + \Phi_q^T \lambda = F(t, q, \dot{q}) \tag{1.3.2}$$

$$D_t^2 \Phi = \Phi_q \ddot{q} + H\dot{q} + 2\Phi_{tq}\dot{q} + \Phi_{tt} = 0 \tag{1.3.3}$$

$$D_t \Phi = \Phi_q \dot{q} + \Phi_t = 0 \tag{1.3.4}$$

$$\Phi(t, q) = 0 \tag{1.3.5}$$

where $\Phi_{tq} = \frac{\partial \Phi_q}{\partial t}$, $\Phi_{tt} = \frac{\partial^2 \Phi}{\partial t^2}$ *and*

$$H = \begin{bmatrix} \sum_i \Phi^1_{q_1 q_i} \dot{q}_i & \cdots & \sum_i \Phi^1_{q_n q_i} \dot{q}_i \\ \vdots & \vdots & \vdots \\ \sum_i \Phi^m_{q_1 q_i} \dot{q}_i & \cdots & \sum_i \Phi^m_{q_n q_i} \dot{q}_i \end{bmatrix}$$

are in implicit-RIF form with A, b, $[r_1, ..., r_k]^T$ *in Definition 1.3.8 given by:*

$$A = \begin{bmatrix} M & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix}, \quad b = \begin{bmatrix} F(t, q, \dot{q}) \\ -H\dot{q} - 2\Phi_{tq}\dot{q} - \Phi_{tt} \end{bmatrix}, \quad [r_1, ..., r_k]^T = \begin{pmatrix} \ddot{q} \\ \lambda \end{pmatrix} \tag{1.3.6}$$

and $N = \{\Phi = 0,\ \Phi_q \dot{q} + \Phi_t = 0\}$, *under the assumption that* $\det(A) \neq 0$.

A proof of this theorem is given in Chapter 4, Section 4.4. In addition the applications of the theorem to open-loop systems and closed-loop systems are discussed.

Hybrid symbolic-numeric methods exploit the strengths of symbolic and numeric methods and avoid some of the shortcomings of purely symbolic or purely numeric methods. In Chapter 5 and Chapter 6, we develop hybrid methods to treat a class of dynamic systems efficiently. In Chapter 5, we symbolically preprocess dynamic models to find hidden constraints for DAE systems, according to Theorem 1.3.9. After having implicit-RIF form, it is easier for us to find consistent initial conditions for numerical integration and real-time simulation of such systems. However, often an explicit RIF form is a better form for numeric integration than an implicit-RIF

form. So we invert the symbolic matrix A from Theorem 1.3.9 with the algorithms for solving linear systems we develop in Chapter 3. In Section 5.6, we introduce numerical analytical geometry to compute consistent initial conditions for numerical integration. In Chapter 6, we have large polynomial systems from limit cycle computations of planar vector fields provided by Dr. Yu. The systems are so large that it is impossible for us to use numerical methods to solve them directly. We start by symbolically simplifying the system, and then compute a Gröbner basis of this system. In the end, we use numerical methods to complete our computational study.

Bibliography

[1] Bates, D. J. Theory and Applications in Numerical Algebraic Geometry. PhD Dissertation, *University of Notre Dame*, 2006.

[2] Bareiss, E. H. Sylvester's Identity and Multistep Integer-preserving Gaussian Elimination. *Mathematics of Computation*, 22(103), 565-578, 1968.

[3] Beckermann, B., Cheng, H. and Labahn, G. Fraction-free Row Reduction of Matrices of Ore Polynomials. *Journal of Symbolic Computation*, 41(5), 513-543, 2006.

[4] Beckermann, B., Cheng, H. and Labahn, G. Fraction-free Row Reduction of Matrices of Skew Polynomials. *ISSAC'02*, 8-15, 2002.

[5] Boulier, F., Lazard, D., Ollivier, F. and Petitot, M. Representation for the Radical of a Finitely Generated Differential Ideal. *ISSAC'95*, ACM Press, 158-166, 1995.

[6] Boulier, F., Lazard, D., Ollivier, F. and Petitot, M. Computing Representations for Radicals of Finitely Generated Differential Ideals. *Technical Report IT-306*, LIFL, 1997.

[7] Boulier, F., Lemaire, F. and Moreno-Maza, M. PARDI! *ISSAC'01*, ACM Press, 38-47, 2001.

[8] Corless, R. M. and Jeffrey, D. J. The Turing Factorization of a Rectangular Matrix. *SIGSAM Bull.*, ACM Press, 31(3), 20-30, 1997.

[9] Corless, R. M., Jeffrey, D. J., Monagan, M. B. and Pratibha. Two Perturbation Calculation in Fluid Mechanics Using Large-Expression Management. *J. Symbolic Computation*, 11, 1-17, 1996.

[10] Erlingsson, Ú., Kaltofen, E. and Musser, D. Generic Gram—Schmidt Orthogonalization by Exact Division. *ISSAC'96*, ACM, 275-282, 1996.

[11] Freeman, T. S., Imirzian, G., Kaltofen, E. and Yagati, L. DAGWOOD: A System for Manipulating Polynomials Given by Straight-line Programs. *ACM Trans. Math. Software*, 14(3), 218-240, 1988.

[12] Geddes, K. O., Czapor, S. R. and Labahn, G. Algorithms for Computer Algebra. *Kluwer*, Boston, MA, 1992.

[13] Giusti M., Hägele K., Lecerf G., Marchand J. and Salvy B. The Projective Noether Maple Package: Computing the Dimension of a Projective Variety. *J. Symbolic Computation*, 30(3), 291-307, 2000.

[14] Gonnet, G. H. Determining Equivalence of Expressions in Random Polynomial Time, Extended Abstract. *Proc. of ACM on Theory of Computing*, 334-341, 1984.

[15] Gonnet, G. H. New Results for Random Determination of Equivalence of Expressions. *Proc. of ACM on Symbolic and Algebraic Comp.*, 127-131, 1986.

[16] Heintz, J. and Schnorr, C. P. Testing Polynomials Which are Easy to Compute (Extended Abstract). *Proceedings of ACM on Theory of computing*, 262-272, 1980.

[17] Hubert, E. Essential Components of Algebraic Differential Equations. *Journal of Symbolic Computations*, 28(4-5), 657-680, 1999.

[18] Hubert, E. Improvements to a Triangulation-decomposition Algorithm for Ordinary Differential Systems in Higher Degree Cases. *ISSAC'04*, ACM Press, 2004.

[19] Hubert, E. Factorization Free Decomposition Algorithms in Differential Algebra. *Journal of Symbolic Computations*, 29(4-5), 641-662, 2000.

[20] Hubert, E. Notes on Triangular Sets and Triangulation-decomposition Algorithms II: Differential Systems. Chapter of Symbolic and Numerical Scientific Computations Edited by U. Langer and F. Winkler. LNCS 2630, *Springer-Verlag Heidelberg*, 2003.

[21] Hubert, E. Notes on Triangular Sets and Triangulation-decomposition Algorithms I: Polynomial Systems. Chapter of Symbolic and Numerical Scientific Computations Edited by U. Langer and F. Winkler. LNCS 2630, *Springer-Verlag Heidelberg*, 2003.

[22] Hubert, E. Differential Algebra for Derivations with Nontrivial Commutation Rules. *Journal of Pure and Applied Algebra*, 200(1-2), 163-190, 2005.

[23] Ibarra, O. H. and Leininger, B. S. On the Simplification and Equivalence Problems for Straight-Line Programs. *J. ACM*, 30(3), 641-656, 1983.

[24] Ibarra, O. H. and Moran, S. Probabilistic Algorithms for Deciding Equivalence of Straight-Line Programs. *J. ACM*, 30(1), 217-228, 1983.

[25] Ibarra, O. H., Moran, S. and Rosier L. E. Probabilistic Algorithms and Straight-Line Programs for Some Rank Decision Problems. *Infor. Proc. Lett.*, 12(5), 227-232, 1981.

[26] Lenstra, A. K., Lenstra, H. W. and Lovász, L. Factoring Polynomials with Rational Coefficients. *Math. Ann.* 261, 515-534, 1982.

[27] Martin, W. A. Determining the Equivalence of Algebraic Expressions by Hash Coding. *Proceedings of ACM on Symbolic and Algebraic Manipulation*, 305-310, 1971.

[28] Monagan, M. B. and Monagan, G. A Toolbox for Program Manipulation and Efficient Code Generation with an Application to a Problem in Computer Vision. *ISSAC'97*, 257-264, 1997.

[29] Monagan, M. B. Signatures + Abstract Types = Computer Algebra − Intermediate Expression Swell. PhD Thesis, *University of Waterloo*, 1990.

[30] Monagan, M. B. Gauss: a Parameterized Domain of Computation System with Support for Signature Functions. *DISCO*, Springer-Verlag LNCS 722, 81-94, 1993.

[31] Monagan, M. B. and Gonnet, G. H. Signature Functions for Algebraic Numbers. *ISSAC'94*, 291-296, 1994.

[32] Nakos, G. C., Turner, P. R. and Williams, R. M. Fraction-free Algorithms for Linear and Polynomial Equations. *SIGSAM Bull.*, ACM Press, 31(3), 11-19, 1997.

[33] Reid, G. J. Algorithms for Reducing a System of PDEs to Standard Form, Determining the Dimension of its Solution Space and Calculating its Taylor Series Solution. *European J. of Appl. Math.*, 2, 293-318, 1991.

[34] Reid, G. J., Wittkopf, A. D. and Lin, P. Differential Elimination-completion Algorithms for Differential Algebraic Equations and Partial Differential Algebraic Equations. *Studies in Applied Mathematics*, 106(1), 1-45, 2001.

[35] Reid, G. J., Wittkopf, A. D., and Boulton, A. Reduction of Systems of Nonlinear Partial Differential Equations to Simplified Involutive Forms. *Eur. J. Appl. Math.*, 7, 604-635, 1996.

[36] Reid, G. J., Lin, P. and Wittkopf, A. D. Differential-Elimination Completion Algorithms for DAE and PDAE. *Studies in Applied Mathematics*, 106, 1-45, 2001.

[37] Rideau, P. Computer Algebra and Mechanics, The James Software, Computer Algebra in Industry I. *John Wiley & Sons Ltd*, 1993.

[38] Rudolf, C. Road Vehicle Modeling Using Symbolic Multibody System Dynamics. Diploma Thesis, *University of Waterloo in cooperation with University of Karlsruhe*, 2003.

[39] Rust, C. and Reid, G. J. Rankings of Partial Derivatives. *ISSAC'97*, ACM Press, 9-16, 1997.

[40] Rust, C. Rankings of Derivatives for Elimination Algorithms and Formal Solvability of Analytic Partial Differential Equations. PhD Thesis, *Univ. Chicago*, 1998.

[41] Schiehlen, W. Multibody Systems Handbook. *Springer-Verlag: Berlin*, 1990.

[42] Schwartz J. T. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, 27(4), 701-717, 1980.

[43] Shi, P. and McPhee, J. J. Dynamics of Flexible Multibody Systems Using Virtual Work and Linear Graph Theory. *Multibody System Dynamics*, 4(4), 355-381, 2000.

[44] Shi, P., McPhee, J. J. and Heppler, G. A Deformation Field for Euler-Bernouli Beams with Application to Flexible Multibody Dynamics. *Multibody System Dynamics*, 5(1), 79-104, 2001.

[45] Shi, P., and McPhee, J. J. Symbolic Programming of a Graph-Theoretic Approach to Flexible Multibody Dynamics. *Mechanics of Structures and Machines*, 30(1), 123-154, 2002.

[46] Trefethen, L. N. and Bau III, D. Numerical Linear Algebra. *SIAM*, 1997.

[47] Turing, A. M. Rounding-off Errors in Matrix Processes. *Quart. J. Mech. Appl. Math.*, 1, 287-308, 1948.

[48] Wittkopf, A. D. and Reid, G. J. The Reduced Involutive Form Package. *Maple Software Package.* First distributed as part of Maple 7, 2001.

[49] Wittkopf, A. D. Algorthims and Implementation for Differential Elimination. PhD Thesis. *Simon Fraser University, Burnaby*, 2004.

[50] Yanami, H. and Anai, H. SyNRAC: A New Maple Package for Supporting Design and Analysis in Engineering. *Journal of Japan Society for Symbolic and Algebraic Computation*, 10(1), 34 - 40, 2003.

[51] Zhou, W., Jeffrey, D. J. and Reid, G. J. An Algebraic Method for Analyzing Open-Loop Dynamic Systems. *ICCS*, LNCS 3516, 586-593, 2005.

[52] Zhou, W., Jeffrey, D. J. and Reid, G. J. Symbolic Computation Sequences and Numerical Analytic Geometry Applied to Multibody Dynamical Systems. Symbolic-Numeric Computation, *Birkhäuser Verlag Basel/Switzerland*, 335-347, 2007.

[53] Zhou, W., Jeffrey, D. J., Reid, G. J., Schmitke C. and McPhee, J. J. Implicit Reduced Involutive Forms and Their Application to Engineering Multibody Systems. *IWMM*, LNCS 3519, 31-43, 2004.

[54] Zhou, W., Jeffrey, D. J. and Corless, R. M. Fraction-free Forms of LU Factoring. *Proceedings of Transgressive Computing*, 443-446, 2006.

[55] Zhou, W., Carette, J., Jeffrey, D. J. and Monagan, M. B. Hierarchical Representations with Signatures for Large Expression Management. *AISC*, LNAI 4120, 254-268, 2006.

[56] Zippel, R. Effective Polynomial Computation. *Kluwer*, 1993.

[57] Zippel, R. Probabilistic Algorithms for Sparse Polynomials. *Proc. of Eurosam*, Springer-Verlag LNCS 72, 216-226, 1979.

Chapter 2

Fraction-free Matrix Factors: New Forms for LU and QR Factors

Gaussian elimination and LU factoring have been greatly studied from the algorithmic point of view, but much less from the point view of the best output format. In this paper, we give new output formats for fraction-free LU factoring and for QR factoring. The formats and the algorithms used to obtain them are valid for any matrix system in which the entries are taken from an integral domain, not just for integer matrix systems.

After discussing the new output format of LU factoring, we give some complexity analysis for the fraction-free algorithm and fraction-free output. Our new output format contains smaller entries than previously suggested forms, and it avoids the gcd computations required by some other partially fraction-free computations. As applications of our fraction-free algorithm and format, we demonstrate how to construct a fraction-free QR factorization and how to solve linear systems within a single domain.

2.1 Introduction

Various applications including robot control and threat analysis have resulted in developing efficient algorithms for the solution of systems of polynomial equations. This involves significant linear algebra subproblems which are not standard *numerical* linear algebra problems. The "arithmetic" that is needed is usually algebraic in nature and must be handled exactly [18]. In the standard *numerical* linear algebra approach, the cost of each operation during the standard Gaussian elimination is regarded as being the same, and the cost of numerical Gaussian elimination is obtained by counting the number of operations. For a $n \times n$ matrix with floating point numbers as entries, the cost of standard Gaussian elimination is $O(n^3)$.

However, in *exact* linear algebra problems, the cost of each operation during the standard Gaussian elimination may vary because of the growth of each entry during Gaussian elimination [29]. For instance, in a polynomial ring, the problem of range growth manifests itself both in increased polynomial degrees and in the size of the coefficients. It is easy to modify the standard Gaussian elimination to avoid all division operations but this leads to a very rapid growth [22, 13, 14, 28].

One way to reduce this intermediate expression swell problem is the use of fraction-free algorithms. Perhaps the first description of such an algorithm was due to Bareiss [1]. Afterwards, efficient methods for alleviating the expression swell problems from Gaussian elimination were given in [2, 22, 8, 10, 17, 24, 25].

Shifting from Gaussian elimination to LU factoring, Turing's description of Gaussian elimination as a matrix factoring is well established in linear algebra [27]. The factoring takes different forms under the names Cholesky, Doolittle-LU, Crout-LU or Turing factoring. Only relatively recently has there been an attempt to combine LU factoring with fraction-free Gaussian elimination [18, 3]. One approach was given

by Theorem 2.2.1 in [3]. In this paper, we use the standard fraction-free Gaussian elimination algorithm given by Bareiss [1] and give a new fraction-free LU factoring and a new fraction-free QR factoring.

Something called fraction-free LU decomposition already exists in MAPLE. Here is an example of its use.

Example 2.1.1. Let A be a 3×3 matrix with polynomials as entries,

$$A := \begin{bmatrix} x & 1 & 3 \\ 3 & 4 & 7 \\ 8 & 1 & 9 \end{bmatrix}.$$

If we use MAPLE to compute the fraction-free LU factoring, we have

```
> LUDecomposition(A, method = FractionFree);
```

$$\begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} \cdot A = \begin{bmatrix} 1 & & \\ 3/x & 1/x & \\ 8/x & \frac{x-8}{x(4x-3)} & \frac{1}{4x-3} \end{bmatrix} \cdot \begin{bmatrix} x & 1 & 3 \\ & 4x-3 & 7x-9 \\ & & 29x-58 \end{bmatrix} \tag{2.1.1}$$

The upper triangular matrix has polynomial entries. However, there are still fractions in the lower triangular matrix. In this paper, we call this factoring a partially fraction-free LU factoring, and give the corresponding algorithm in Section 2.3.

Section 2.2 presents a format for LU factoring that is completely fraction-free. In Section 2.3, we give a completely fraction-free LU factoring algorithm and its time complexity, compared with the time complexity of a partially fraction-free LU factoring. Benchmarks follow in Section 2.4 and illustrate the complexity results from Section 2.3. The last part of the paper, Section 2.5, introduces the application of the completely fraction-free LU factoring to obtain a similar structure for a fraction-free

QR factoring. In addition, it introduces the fraction-free forward and backward substitutions to keep the whole computation in one domain for solving a linear system. Section 2.6 gives our conclusion and discussion.

2.2 Fraction-free LU Factoring

In 1968, Bareiss [1] pointed out that his integer-preserving Gaussian elimination (or fraction-free Gaussian elimination) could reduce the magnitudes of the entries in the transformed matrices and increase considerably the computational efficiency in comparison with the corresponding standard Gaussian elimination. We also know that the conventional LU decomposition is used for solving several linear systems with the same coefficient matrix without the need to recompute the full Gaussian elimination. Here we combine these two ideas and give a new fraction-free LU factoring.

In 1997, Nakos, Turner and Williams [18] gave an incompletely fraction-free LU factorization. In the same year, Corless and Jeffrey [3] gave one kind of fraction-free LU factoring, which is theorem 2.2.1.

Theorem 2.2.1. *Any rectangular matrix $A \in \mathbb{Z}^{n\times m}$ may be written*

$$F_1 P A = L F_2 U \ , \tag{2.2.1}$$

where $F_1 = diag(1, p_1, p_1p_2, \ldots, p_1p_2\cdots p_{n-1})$, P is a permutation matrix, $L \in \mathbb{Z}^{n\times n}$ is a unit lower triangular, $F_2 = diag(1, 1, p_1, p_1p_2, \ldots, p_1p_2\cdots p_{n-2})$, and $U \in \mathbb{Z}^{n\times m}$ are upper triangular matrices. The $p_i \in \mathbb{Z}$ are the pivots that arise.

This factoring is modeled on other fraction-free definitions, such as pseudo-division, and the idea is to inflate the given object or matrix so that subsequent divisions are guaranteed to be exact. However, although this model is satisfactory

for pseudo-division, the above matrix factoring has two unsatisfactory features: first, two inflating matrices are required, and secondly, the matrices are clumsy, containing entries that increase rapidly in size. If the model of pseudo-division is abandoned, a tidier factoring is possible. This is the first contribution of this paper.

Theorem 2.2.2. *Let $\mathbb{I}$ be an integral domain and A be a full-rank matrix in $\mathbb{I}^{n\times m}$ with $n \leq m$. Then, A may be written*

$$PA = LD^{-1}U,$$

where

$$L = \begin{bmatrix} p_1 & & & & \\ L_{2,1} & p_2 & & & \\ \vdots & \vdots & \ddots & & \\ L_{n-1,1} & L_{n-1,2} & \cdots & p_{n-1} & \\ L_{n,1} & L_{n,2} & \cdots & L_{n,n-1} & 1 \end{bmatrix},$$

$$D = \begin{bmatrix} p_1 & & & & \\ & p_1p_2 & & & \\ & & \ddots & & \\ & & & p_{n-2}p_{n-1} & \\ & & & & p_{n-1} \end{bmatrix},$$

$$U = \begin{bmatrix} p_1 & U_{1,2} & \cdots & U_{1,n-1} & U_{1,n} & \cdots & U_{1,m} \\ & p_2 & \cdots & U_{2,n-1} & U_{2,n} & \cdots & U_{2,m} \\ & & \ddots & \vdots & \vdots & \vdots & \vdots \\ & & & p_{n-1} & U_{n-1,n} & \cdots & U_{n-1,m} \\ & & & & p_n & \cdots & U_{n,m} \end{bmatrix},$$

P is a permutation matrix, L and U are triangular as shown, and the pivots p_i that arise are in $\mathbb{I}$. The pivot p_n is also the determinant of the matrix A.

Proof. For a full-rank matrix, there always exists a permutation matrix such that the pivots are non-zero during Gaussian elimination. Let P be such a permutation matrix for the full-rank matrix A. We will give details in the algorithm for finding this permutation matrix. Classical Gaussian elimination shows that PA admits the

following factorization.

$$PA = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} & \cdots & a_{2,m} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} & \cdots & a_{3,m} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n} & \cdots & a_{n,m} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & & & & \\ \frac{a_{2,1}}{a_{1,1}} & 1 & & & \\ \frac{a_{3,1}}{a_{1,1}} & \frac{a^{(1)}_{3,2}}{a^{(1)}_{2,2}} & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ \frac{a_{n,1}}{a_{1,1}} & \frac{a^{(1)}_{n,2}}{a^{(1)}_{2,2}} & \frac{a^{(2)}_{n,3}}{a^{(2)}_{3,3}} & \cdots & 1 \end{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} & \cdots & a_{1,m} \\ & a^{(1)}_{2,2} & a^{(1)}_{2,3} & \cdots & a^{(1)}_{2,n} & \cdots & a^{(1)}_{2,m} \\ & & a^{(2)}_{3,3} & \cdots & a^{(2)}_{3,n} & \cdots & a^{(2)}_{3,m} \\ & & & \ddots & \vdots & \cdots & \vdots \\ & & & & a^{(n-1)}_{n,n} & \cdots & a^{(n-1)}_{n,m} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & & & & \\ \frac{a_{2,1}}{a_{1,1}} & 1 & & & \\ \frac{a_{3,1}}{a_{1,1}} & \frac{a^{(1)}_{3,2}}{a^{(1)}_{2,2}} & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ \frac{a_{n,1}}{a_{1,1}} & \frac{a^{(1)}_{n,2}}{a^{(1)}_{2,2}} & \frac{a^{(2)}_{n,3}}{a^{(2)}_{3,3}} & \cdots & 1 \end{bmatrix} \begin{bmatrix} a_{1,1} & & & & \\ & a^{(1)}_{2,2} & & & \\ & & a^{(2)}_{3,3} & & \\ & & & \ddots & \\ & & & & a^{(n-1)}_{n,n} \end{bmatrix}$$

$$\begin{bmatrix} 1 & \frac{a_{1,2}}{a_{1,1}} & \frac{a_{1,3}}{a_{1,1}} & \cdots & \frac{a_{1,n}}{a_{1,1}} & \cdots & \frac{a_{1,m}}{a_{1,1}} \\ & 1 & \frac{a^{(1)}_{2,3}}{a^{(1)}_{2,2}} & \cdots & \frac{a^{(1)}_{2,n}}{a^{(1)}_{2,2}} & \cdots & \frac{a^{(1)}_{2,m}}{a^{(1)}_{2,2}} \\ & & 1 & \cdots & \frac{a^{(2)}_{3,n}}{a^{(2)}_{3,3}} & \cdots & \frac{a^{(2)}_{3,m}}{a^{(2)}_{3,3}} \\ & & & \ddots & \vdots & \cdots & \vdots \\ & & & & 1 & \cdots & \frac{a^{(n-1)}_{n,m}}{a^{(n-1)}_{n,n}} \end{bmatrix}$$

$$= \ell \cdot d \cdot u.$$

The coefficients $a_{i,j}^{(k)}$ are related by the following relations:

$$
\begin{aligned}
& i = 1, \qquad a_{i,j}^{(0)} = a_{i,j}; \\
& 2 \le i \le j \le m, \quad a_{i,j}^{(i-1)} = \begin{vmatrix} a_{i-1,i-1}^{(i-2)} & a_{i-1,j}^{(i-2)} \\ \frac{a_{i,i-1}^{(i-2)}}{a_{i-1,i-1}^{(i-2)}} & \frac{a_{i,j}^{(i-2)}}{a_{i-1,i-1}^{(i-2)}} \end{vmatrix}; \\
& n \ge i \ge j \ge 2, \quad a_{i,j}^{(j-1)} = \begin{vmatrix} a_{j-1,j-1}^{(j-2)} & a_{j-1,j}^{(j-2)} \\ \frac{a_{i,j-1}^{(j-2)}}{a_{j-1,j-1}^{(j-2)}} & \frac{a_{i,j}^{(j-2)}}{a_{j-1,j-1}^{(j-2)}} \end{vmatrix}.
\end{aligned}
$$

Let us define $A_{i,j}^{(k)}$ by $A_{i,j}^{(k)} = \begin{vmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} & a_{1,j} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k} & a_{2,j} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{k,1} & a_{k,2} & \cdots & a_{k,k} & a_{k,j} \\ a_{i,1} & a_{i,2} & \cdots & a_{i,k} & a_{i,j} \end{vmatrix}$.

We have the relations $A_{i,j}^{(k)} = \frac{1}{A_{k-1,k-1}^{(k-2)}} \begin{vmatrix} A_{k,k}^{(k-1)} & A_{k,j}^{(k-1)} \\ A_{i,k}^{(k-1)} & A_{i,j}^{(k-1)} \end{vmatrix}$, which was proved in [1].

From the definitions, we have

$$
\begin{aligned}
m \ge j \ge i \ge 1, \quad a_{i,j}^{(i-1)} &= \frac{1}{A_{i-1,i-1}^{(i-2)}} A_{i,j}^{(i-1)}, \\
n \ge j \ge i \ge 1, \quad a_{j,i}^{(i-1)} &= \frac{1}{A_{i-1,i-1}^{(i-2)}} A_{j,i}^{(i-1)}.
\end{aligned}
$$

So

$$
\begin{aligned}
\ell_{i,k} &= \frac{1}{a_{k,k}^{(k-1)}} \cdot a_{i,k}^{(k-1)} = \frac{A_{k-1,k-1}^{(k-2)}}{A_{k,k}^{(k-1)}} \cdot \frac{A_{i,k}^{(k-1)}}{A_{k-1,k-1}^{(k-2)}} = \frac{A_{i,k}^{(k-1)}}{A_{k,k}^{(k-1)}}; \qquad \ell_{k,k} = 1; \\
u_{k,j} &= \frac{1}{a_{k,k}^{(k-1)}} \cdot a_{k,j}^{(k-1)} = \frac{A_{k-1,k-1}^{(k-2)}}{A_{k,k}^{(k-1)}} \cdot \frac{A_{k,j}^{(k-1)}}{A_{k-1,k-1}^{(k-2)}} = \frac{A_{k,j}^{(k-1)}}{A_{k,k}^{(k-1)}}; \qquad u_{k,k} = 1; \\
d_{k,k} &= a_{k,k}^{(k-1)} = \frac{A_{k,k}^{(k-1)}}{A_{k-1,k-1}^{(k-2)}}
\end{aligned}
$$

Then the fraction-free LU form $PA = LD^{-1}U$ can be written as:

$$\begin{aligned} L_{i,k} &= A_{i,k}^{(k-1)}, \quad n \geq i \geq k \geq 1, \\ U_{k,j} &= A_{k,j}^{(k-1)}, \quad m \geq j \geq k \geq 1, \\ D_{k,k} &= \frac{A_{k-1,k-1}^{(k-2)}}{A_{k,k}^{(k-1)}} \cdot A_{k,k}^{(k-1)} \cdot A_{k,k}^{(k-1)} = A_{k-1,k-1}^{(k-2)} A_{k,k}^{(k-1)}, \quad n \geq k \geq 1, \end{aligned}$$

which is also equivalent to $PA = LD^{-1}U$, where

$$L = \begin{bmatrix} A_{1,1}^{(0)} & & & & \\ A_{2,1}^{(0)} & A_{2,2}^{(1)} & & & \\ \vdots & \vdots & \ddots & & \\ A_{n-1,1}^{(0)} & A_{n-1,2}^{(1)} & \cdots & A_{n-1,n-1}^{(n-2)} & \\ A_{n,1}^{(0)} & A_{n,2}^{(1)} & \cdots & A_{n,n-1}^{(n-2)} & 1 \end{bmatrix},$$

$$D = \begin{bmatrix} A_{1,1}^{(0)} & & & & & \\ & A_{1,1}^{(0)} A_{2,2}^{(1)} & & & & \\ & & \ddots & & & \\ & & & A_{n-2,n-2}^{(n-3)} A_{n-1,n-1}^{(n-2)} & & \\ & & & & A_{n-1,n-1}^{(n-2)} \end{bmatrix},$$

$$U = \begin{bmatrix} A_{1,1}^{(0)} & A_{1,2}^{(0)} & \cdots & A_{1,n-1}^{(0)} & A_{1,n}^{(0)} & \cdots & A_{1,m}^{(0)} \\ & A_{2,2}^{(1)} & \cdots & A_{2,n-1}^{(1)} & A_{2,n}^{(1)} & \cdots & A_{2,m}^{(1)} \\ & & \ddots & \vdots & \vdots & \vdots & \vdots \\ & & & A_{n-1,n-1}^{(n-2)} & A_{n-1,n}^{(n-2)} & \cdots & A_{n-1,m}^{(n-2)} \\ & & & & A_{n,n}^{(n-1)} & \cdots & A_{n,m}^{(n-1)} \end{bmatrix}.$$

□

Example 2.2.3. Let us compute the fraction-free LU factoring of the same matrix A as in Example 2.1.1 (here, we take P to be the identity matrix).

$$A := \begin{bmatrix} x & 1 & 3 \\ 3 & 4 & 7 \\ 8 & 1 & 9 \end{bmatrix}; \quad P = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix}.$$

In Example 2.1.1, we found that there were still fractions in the fraction-free LU factoring from MAPLE computation. When using our new LU factoring, there is no fraction appearing in the upper or lower triangular matrix.

$$P \cdot A = \begin{bmatrix} x & & \\ 3 & 4x-3 & \\ 8 & x-8 & 1 \end{bmatrix} \cdot \begin{bmatrix} x & & \\ & x(4x-3) & \\ & & 4x-3 \end{bmatrix}^{-1} \cdot \begin{bmatrix} x & 1 & 3 \\ & 4x-3 & 7x-9 \\ & & 29x-58 \end{bmatrix}$$

This new output is better than the old one for the following two reasons: first, this new LU factoring form keeps the computation in one domain; second, the division used in the new factoring is an exact division, while in Example 2.1.1 fraction-free LU factoring the division needs gcd computations for the lower triangular matrix as in Equation 2.1.1. We give a more general comparison of these two forms on their time complexity in Theorems 2.3.9 of Section 2.3.

2.3 Completely Fraction-free Algorithm and Time Complexity Analysis

Here we give an algorithm for computing a completely fraction-free LU factoring. We abbreviate this to CFFLU. This is generic code; an actual MAPLE implementation would make additional optimizations with respect to different input domains.

Algorithm 2.3.1. Completely Fraction-free LU factoring (CFFLU)

Input: A $n \times m$ matrix A, with $m \geq n$.

Output: Four matrices P, L, D, U, where P is a $n \times n$ permutation matrix, L is a $n \times n$ lower triangular matrix, D is a $n \times n$ diagonal matrix, U is a $n \times m$ upper triangular matrix and $PA = LD^{-1}U$.

```
U := Copy(A);
(n,m) := Dimension(U):
oldpivot := 1;
L:=IdentityMatrix(n,n,'compact'=false);
DD:=ZeroVector(n,'compact'=false);
P := IdentityMatrix(n, n, 'compact'=false);
for k from 1 to n-1 do
    if U[k,k] = 0 then
        kpivot := k+1;
        Notfound := true;
        while kpivot < (n+1) and Notfound do
            if U[kpivot, k] <> 0 then
                Notfound := false;
            else
                kpivot := kpivot +1;
            end if;
        end do:
        if kpivot = n+1 then
            error "Matrix is rank deficient";
        else
```

```
            swap := U[k, k..n];
            U[k,k..n] := U[kpivot, k..n];
            U[kpivot, k..n] := swap;
            swap := P[k, k..n];
            P[k, k..n] := P[kpivot, k..n];
            P[kpivot, k..n] := swap;
         end if:
      end if:
      L[k,k]:=U[k,k];
      DD[k] := oldpivot * U[k, k];
      Ukk := U[k,k];
      for i from k+1 to n do
         L[i,k] := U[i,k];
         Uik := U[i,k];
         for j from k+1 to m do
            U[i,j]:=normal((Ukk*U[i,j]-U[k,j]*Uik)/oldpivot);
         end do;
         U[i,k] := 0;
      end do;
      oldpivot:= U[k,k];
end do;
DD[n]:= oldpivot;
```

For comparison, we also recall a partially fraction-free LU factoring (PFFLU).

Algorithm 2.3.2. Partially Fraction-free LU factoring (PFFLU)

Input: A $n \times m$ matrix A.

Output: Three matrices P, L and U, where P is a $n \times n$ permutation matrix, L is a $n \times n$ lower triangular matrix, U is a $n \times m$ fraction-free upper triangular matrix and $PA = LU$.

```
U := Copy(A);
(n,m) := Dimension(U):
oldpivot := 1;
L:=IdentityMatrix(n,n,'compact'=false);
P := IdentityMatrix(n, n, 'compact'=false);
for k from 1 to n-1 do
   if U[k,k] = 0 then
        kpivot := k+1;
        Notfound := true;
        while kpivot < (n+1) and Notfound do
            if U[kpivot, k] <> 0 then
                Notfound := false;
            else
                kpivot := kpivot +1;
            end if;
        end do:
        if kpivot = n+1 then
            error "Matrix is rank deficient";
        else
            swap := U[k, k..n];
            U[k,k..n] := U[kpivot, k..n];
            U[kpivot, k..n] := swap;
```

```
            swap := P[k, k..n];
            P[k, k..n] := P[kpivot, k..n];
            P[kpivot, k..n] := swap;
        end if:
    end if:
    L[k,k]:=1/oldpivot;
    Ukk := U[k,k];
    for i from k+1 to n do
        L[i,k] := normal(U[i,k]/(oldpivot * U[k, k]));
        Uik := U[i,k];
        for j from k+1 to m do
           U[i,j]:=normal((Ukk*U[i,j]-U[k,j]*Uik)/oldpivot);
        end do;
        U[i,k] := 0;
    end do;
    oldpivot:= U[k,k];
end do;
L[n,n]:= 1/oldpivot;
```

The main difference between Algorithm 2.3.1 and Algorithm 2.3.2 is that Algorithm 2.3.2 uses non-exact divisions when computing the L matrix. The reason we give these two algorithms is that we want to show the advantage of a fraction-free output format.

Theorem 2.3.3. *Let A be a $n \times m$ matrix of full rank with entries in a domain I and $n \leq m$. On input A, Algorithm 2.3.1 outputs four matrices P, L, D, U with entries in I such that $PA = LD^{-1}U$, P is a $n \times n$ permutation matrix, L is a $n \times n$*

lower triangular matrix, D is a $n \times n$ diagonal matrix, U is a $n \times m$ upper triangular matrix. Furthermore, all divisions are exact.

Proof. In Algorithm 2.3.1, each pass through the main loop starts by finding a non-zero pivot, and reorders the row accordingly. For the sake of proof, we can suppose that the rows have been permuted from the start, so that no permutation is necessary.

Then we prove by induction that at the end of step k, for $k = 1, \ldots, n-1$, we have

- $D[1] = A_{1,1}^{(0)}$ and $D[i] = A_{i-1,i-1}^{(i-2)} A_{i,i}^{(i-1)}$ for $i = 2, \ldots, k$,
- $L[i,j] = A_{i,j}^{(j-1)}$ for $j = 1, \ldots, k$ and $i = 1, \ldots, n$,
- $U[i,j] = A_{i,j}^{(i-1)}$ for $i = 1, \ldots, k$ and $j = i, \ldots, m$,
- $U[i,j] = A_{i,j}^{(k)}$ for $i = k+1, \ldots, n$ and $j = k+1, \ldots, m$,
- all other entries are 0,

These equalities are easily checked for $k = 1$. Suppose that this holds at step k, and let us prove it at step $k+1$. Then:

- for $i = k+1, \ldots, n$, $L[i, k+1]$ gets the value $U[i, k+1] = A_{i,k+1}^{(k)}$,
- $D[k+1]$ gets the value $A_{k,k}^{(k-1)} A_{k+1,k+1}^{(k)}$,
- for $i, j = k+2, \ldots, m$, $U[i,j]$ gets the value

$$\frac{A_{k,k}^{(k-1)} A_{i,j}^{(k-1)} - A_{k,j}^{(k-1)} A_{i,k}^{(k-1)}}{A_{k-1,k-1}^{(k-2)}} = A_{i,j}^{(k)}$$

- $U[i, k+1]$ gets the value 0 for $i = k+2, \ldots, n$.

This proves our statement by induction. In particular, as claimed, all divisions are exact.

□

In the following part of this section, we discuss the advantages of the completely fraction-free LU factoring and give the time complexity analysis for the fraction-free algorithms and fraction-free outputs in matrices with univariate polynomial entries and with integer entries. First, let us introduce two definitions from the book [6] about a length of an integer and a multiplication time.

Definition 2.3.4. For a nonzero integer $a \in \mathbb{Z}$, we define the length $\lambda(a)$ of a as

$$\lambda(a) = \lfloor \log |a| + 1 \rfloor,$$

where $\lfloor . \rfloor$ denotes rounding down to the nearest integer.

Definition 2.3.5. Let $\mathbb{I}$ be a ring (commutative, with 1). We call a function $M : \mathbb{N}_{>0} \to \mathbb{R}_{>0}$ a multiplication time for $\mathbb{I}[x]$ if polynomials in $\mathbb{I}[x]$ of degree less than n can be multiplied using at most $M(n)$ operations in $\mathbb{I}$. Similarly, a function M as above is called a multiplication time for $\mathbb{Z}$ if two integers of length n can be multiplied using at most $M(n)$ word operations.

In principle, any multiplication algorithm leads to a multiplication time. Table 2.3.1 summarizes the multiplication times for some general algorithms.

For two integers a and b with length less than ℓ, the cost of their division is $O(M(\ell))$ word operations, and the cost of a gcd computation is $O(M(\ell) \log \ell)$ word operations. For two polynomials $a, b \in \mathbb{K}[x]$, where $\mathbb{K}$ is an arbitrary field, of degrees less than d, the cost of division is $O(M(d))$, and the cost of gcd computation is $O(M(d) \log d)$. For two univariate polynomials $a, b \in \mathbb{Z}[x]$ of degree less than d

Algorithm	$M(n)$
classical	$O(n^2)$
Karatsuba(Karatsuba & Ofman 1962)	$O(n^{1.59})$
FFT multiplication (provided that R supports the FFT)	$O(n \log n)$
Schönhage & Strassen (1971), Schönhage (1977) Cantor & Kaltofen (1991); FFT based	$O(n \log n \log\log n)$

Table 2.3.1: Various polynomial multiplication algorithms and their running times.

and coefficient's length less than ℓ, if a divides b and if the quotient has coefficients of length less than ℓ', the cost of their division is $O(M(d(\max(\ell, \ell') + \log d)))$, if the division is non-exact, i.e. we need to compute the gcd of a and b, the cost of computing $\gcd(a, b)$ is $O(M(d)\log d \cdot (d+\ell) \cdot M(\log(d(\log(d)+\ell))) \cdot \log\log(d(\log(d)+\ell)) + dM((d+\ell)\log(d(\log(d)+\ell))) \cdot (\log d + \log \ell))$ [6].

Lemma 2.3.6. *For a $k \times k$ matrix $A = [a_{i,j}]$, with $a_{i,j} \in \mathbb{Z}[x_1, \ldots, x_m]$ with degree and length bounded by d and ℓ respectively, the degree and length of $\det(A)$ are bounded by kd and $k(\ell + \log k + d\log(m+1))$ respectively.*

Proof. This lemma with its proof can be found in the paper Krick, Pardo and Sombra [15]. □

Based on the completely fraction-free LU factoring algorithm 2.3.1, at each k^{th} step of fraction-free Gaussian elimination, we have to compute

$$U_{k,j} = A_{k,j}^{(k-1)},$$

where

$$A_{k,j}^{(k-1)} = \det \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k-1} & a_{1,j} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k-1} & a_{2,j} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{k-1,1} & a_{k-1,2} & \cdots & a_{k-1,k-1} & a_{k-1,j} \\ a_{k,1} & a_{k,2} & \cdots & a_{k,k-1} & a_{k,j} \end{pmatrix}_{k \times k} . \tag{2.3.1}$$

Lemma 2.3.7. *If every entry $a_{i,j}$ of the matrix $A = [a_{i,j}]_{n \times n}$ is a univariate polynomial over a field $\mathbb{K}$ with degree less than d, fraction-free Gaussian elimination keeps the degree of polynomial growth as $\deg(A_{i,j}^{(k)}) \leq kd$.*

If every entry $a_{i,j}$ of the matrix $A = [a_{i,j}]_{n \times n}$ is in $\mathbb{Z}$ and has its length bounded by ℓ, the fraction-free Gaussian elimination keeps the length of each entry grow as $\lambda(A_{i,j}^{(k)}) \leq k(\ell + \log k)$.

If every entry of the matrix $A = [a_{i,j}]_{n \times n}$ is a univariate polynomial over $\mathbb{Z}[x]$ with degree less than d and its coefficient's length bounded by ℓ, fraction-free Gaussian elimination keeps the degree of each entry grow as $\deg(A_{i,j}^{(k)}) \leq kd$ and the length of each entry grow as $\lambda(A_{i,j}^{(k)}) \leq k(\ell + \log k + d \log 2)$.

Proof. If $a_{i,j} \in \mathbb{K}[x]$ of degree less than d, from Lemma 2.3.6, we have the growth of the degree $A_{i,j}^{(k)}$ less than kd. If $a_{i,j} \in \mathbb{Z}$ of length bounded by ℓ, from Equation 2.3.1 and Lemma 2.3.6 with $m = 0$, we have the growth of the length $A_{i,j}^{(k)}$ less than $k(\ell + \log k)$. If $a_{i,j} \in \mathbb{Z}[x]$ of degree bounded by d and the coefficient's length bounded by ℓ, from Equation 2.3.1 and Lemma 2.3.6 with $m = 1$, we have the growth of the degree of each entry less than kd and the growth of the length of each coefficient less than $k(\ell + \log k + d \log 2)$. □

In the following part of this section, we want to demonstrate that the difference between fraction-free LU factoring and our completely fraction-free LU factoring is

the divisions used in computing their lower triangular matrices L. We discuss here only three cases. Case 1, we will analyze the cost of two algorithms with $A \in \mathbb{K}[x]$, where $\mathbb{K}$ is a field, i.e. we only consider the growth of degree during the factoring. Case 2, we will analyze the cost of two algorithms with $A \in \mathbb{Z}$, i.e. we only consider the growth of length during the factoring. Case 3, we will analyze the cost of two algorithms with $A \in \mathbb{Z}[x]$. For more cases, such as $A \in \mathbb{Z}[x_1, ..., x_m]$, the basic idea will be the same as these three basic cases we analysis.

Theorem 2.3.8. *For a matrix $A = [a_{i,j}]_{n\times n}$ with entries in $\mathbb{K}[x]$, if every $a_{i,j}$ has degree less than d, the time complexity of completely fraction-free LU factoring for A is bounded by $O(n^3 M(nd))$ operations in $\mathbb{K}$.*

For a matrix $A = [a_{i,j}]_{n\times n}$ with entries in $\mathbb{Z}$, if every $a_{i,j}$ has its length bounded by ℓ, the time complexity of completely fraction-free LU factoring for A is bounded by $O(n^3 M(n \log n + n\ell))$ word operations.

For a matrix $A = [a_{i,j}]_{n\times n}$ with univariate polynomial entries in $\mathbb{Z}[x]$, if every $a_{i,j}$ has its degree less than d and has its length bounded by ℓ, the time complexity of completely fraction-free LU factoring for A is bounded by $O(n^3 M(n^2 d\ell + nd^2))$ word operations.

Proof. Let case 1 be the case $a_{i,j} \in \mathbb{K}[x]$ with $d = \max_{i,j} \deg(a_{i,j}) + 1$, case 2 be the case $a_{i,j} \in \mathbb{Z}$ with $\ell = \max_{i,j} \lambda(a_{i,j})$ and case 3 be the case $a_{i,j} \in \mathbb{Z}[x]$ with $d = \max_{i,j} \deg(a_{i,j}) + 1$ and $\ell = \max_{i,j} \lambda(a_{i,j})$. From Lemma 2.3.6, at each step k, $\deg(A_{i,j}^{(k)}) \leq kd$ in case 1 and $\lambda(A_{i,j}^{(k)}) \leq k(\ell + \log k)$ in case 2 with $m = 0$, and $\lambda(A_{i,j}^{(k)}) \leq k(\ell + \log k + d \log 2)$ and $\deg(A_{i,j}^{(k)}) \leq kd$ in case 3 with $m = 1$.

When we do completely fraction-free LU factoring, at the kth step, we have $(n-k)^2$ entries to compute. For each new entry from step $k-1$ to step k, we need to do at most two multiplication, one subtraction and one division. The cost will

be bounded by $O(M(kd))$ for case 1, bounded by $O(M(k(\ell + \log k)))$ for case 2 and bounded by $O(M(kd(((2k)\ell + \log(2k) + d\log 2) + \log(kd))))$ for case 3.

Let c_1, c_2 and c_3 be constants. We have

$$O(M(kd)) = c_1 \times M(kd),$$

$$O(M(k(\ell + \log k))) = c_2 \times M(k(\ell + \log k))$$

and

$$O(M(kd(k\ell + \log k + d\log 2) + \log(kd)))) = c_3 \times (k^2 d\ell + kd\log k + kd^2 \log 2 + kd\log(kd))$$

.

For case 1, the total cost for the completely fraction-free LU factoring will be bounded by

$$\sum_{k=1}^{n-1}(n-k)^2 \times O(M(kd)) = \sum_{k=1}^{n-1}(n-k)^2 \times c_1 \times M(kd) = O(n^3 M(nd)).$$

For case 2, the total cost for the completely fraction-free LU factoring will be bounded by

$$\begin{aligned} & \sum_{k=1}^{n-1}(n-k)^2 \times O(M(k(\ell + \log k))) \\ = & \sum_{k=1}^{n-1}(n-k)^2 \times c_2 \times M(k(\ell + \log k)) \\ = & O(n^3 M(n\log n + n\ell)) \end{aligned}$$

For case 3, the total cost for the completely fraction-free LU factoring will be bounded by

$$\begin{aligned} & \sum_{k=1}^{n-1}(n-k)^2 \times O(M(kd((2k\ell + \log(2k) + d\log 2) + \log(kd)))) \\ = & \sum_{k=1}^{n-1}(n-k)^2 \times c \times M(2k^2 d\ell + kd\log(2k) + kd^2\log 2 + kd\log(kd)) \\ = & O(n^3 M(n^2 d\ell + nd^2)). \end{aligned}$$

□

The extra gcd computation in Algorithm 2.3.2 when computing $L[i,k] := U[i,k]$ $/(oldpivot \cdot U[k,k])$ makes the fraction-free LU factoring more expensive. In the following theorem, we will give the time complexity for the partially fraction-free LU factoring and compare it with our completely fraction-free LU factoring.

Theorem 2.3.9. *For a matrix $A = [a_{i,j}]_{n\times n}$, if every $a_{i,j} \in \mathbb{K}[x]$ has its degree less than d, the time complexity of the partially fraction-free LU factoring for A is bounded by $O((n^2\log(nd) + n^3)M(nd))$.*

For a matrix $A = [a_{i,j}]_{n\times n}$, if every $a_{i,j} \in \mathbb{Z}$ has its length bounded by ℓ, the time complexity of partially fraction-free LU factoring for A is bounded by $O((n^2\log(n\log n+ n\ell) + n^3)M(n\log n + n\ell))$.

For a matrix $A = [a_{i,j}]_{n\times n}$, if every $a_{i,j} \in \mathbb{Z}[x]$ has its degree less than d and its length of coefficient bounded by ℓ, the time complexity of the partially fraction-free LU factoring for A is bounded by $O(n^3M(n^2d\ell + nd^2) + n^2\delta M)$, where $\delta M = M(nd)\log(nd)\cdot(n(\ell+\log n+d))\cdot M(\log(nd(\ell+d)))\cdot\log\log(nd(\ell+d)) + ndM(n(\ell+\log n + d)\log(nd(\ell+d)))\cdot\log(n(\ell+d))$.

Proof. As above, case 1 is $a_{i,j} \in \mathbb{K}[x]$ with $d = \max_{i,j}\deg(a_{i,j}) + 1$, case 2 is $a_{i,j} \in \mathbb{Z}$ with $\ell = \max_{i,j}\lambda(a_{i,j})$ and case 3 is $a_{i,j} \in \mathbb{Z}[x]$ with $d = \max_{i,j}\deg(a_{i,j}) + 1$ and $\ell = \max_{i,j}\lambda(a_{i,j})$. From Lemma 2.3.6, at each step k, $\deg(A_{i,j}^{(k)}) \le kd$ in case 1, $\lambda(A_{i,j}^{(k)}) \le k(\ell+\log k)$ in case 2 with $m = 0$, and $\lambda(A_{i,j}^{(k)}) \le k(\ell+\log k + d\log 2)$ and $\deg(A_{i,j}^{(k)}) \le kd$ in case 3 with $m = 1$.

When we do fraction-free LU factoring, at the kth step, we need to do the same computation for U matrix as in Algorithm 2.3.1. In addition, we have $n-k$ entries, which are the entries of kth column of L matrix, containing gcd computation for its division in order to get the common factors out of division.

For case 1, the total cost for the partially fraction-free LU factoring will be bounded by

$$\begin{aligned}
& \sum_{k=1}^{n-1}((n-k)^2 \times O(M(kd)) + (n-k) \times O(M(kd)\log(kd))) \\
= & \sum_{k=1}^{n-1}((n-k)^2 \times c_1 \times M(kd) + (n-k) \times c_2 \times M(kd)\log(kd) \\
= & \ O((n^2\log(nd) + n^3)M(nd)),
\end{aligned}$$

where c_1 and c_2 are constants.

For case 2, the total cost for the partially fraction-free LU factoring will be bounded by

$$\begin{aligned}
& \textstyle\sum_{k=1}^{n-1}((n-k)^2 \times O(M(k(\ell+\log k))) + (n-k) \times O(M(k(\ell+\log k))\log(k(\ell+\log k)))) \\
& = \textstyle\sum_{k=1}^{n-1}((n-k)^2 \times c_3 \times M(k(\ell+\log k)) + (n-k) \times c_4 \times M(k(\ell+\log k))\log(k(\ell+\log k)) \\
& = O((n^2\log(n\log n + n\ell) + n^3)M(n\log n + n\ell)),
\end{aligned}$$

where c_3 and c_4 are constants.

For case 3, the total cost for the partially fraction-free LU factoring will be bounded by

$$\begin{aligned}
& \textstyle\sum_{k=1}^{n-1}((n-k)^2 \times O(M(kd(2k\ell+\log(2k)+d\log 2+\log(kd)))) + (n-k) \times O(M(kd)\log(kd)\cdot \\
& (kd+(k(\ell+\log k+d\log 2)))\cdot M(\log(kd(\log(kd)+(k(\ell+\log k+d\log 2)))))\cdot\log\log(kd(\log(kd)+ \\
& (k(\ell+\log k+d\log 2)))) + kdM((kd+(k(\ell+\log k+d\log 2)))\log(kd(\log(kd)+(k(\ell+ \\
& \log k+d\log 2)))))\cdot(\log(kd)+\log(k(\ell+\log k+d\log 2)))) \\
& = \textstyle\sum_{k=1}^{n-1}((n-k)^2 \times O(M(kd(k\ell+d))) + (n-k) \times O(M(kd)\log(kd)\cdot(k(\ell+\log k+d))\cdot \\
& M(\log(kd(\ell+d)))\cdot\log\log(kd(\ell+d)) + kdM(k(\ell+\log k+d)\log(kd(\ell+d)))\cdot\log(k(\ell+d)))) \\
& = \textstyle\sum_{k=1}^{n-1}((n-k)^2 \times c_5 \times M(kd(k\ell+d)) + (n-k) \times c_6 \times (M(kd)\log(kd)\cdot(k(\ell+\log k+d))\cdot \\
& M(\log(kd(\ell+d)))\cdot\log\log(kd(\ell+d)) + kdM(k(\ell+\log k+d)\log(kd(\ell+d)))\cdot\log(k(\ell+d)))) \\
& = O(n^3M(n^2d\ell + nd^2) + n^2\delta M),
\end{aligned}$$

where c_5 and c_6 are constants, $\delta M = M(nd)\log(nd)\cdot(n(\ell+\log n+d))\cdot M(\log(nd(\ell+d)))\cdot\log\log(nd(\ell+d))+ndM(n(\ell+\log n+d)\log(nd(\ell+d)))\cdot\log(n(\ell+d))$.

□

Compare Theorem 2.3.8 with Theorem 2.3.9, we can clearly see that partially fraction-free LU factoring costs a little more than our completely fraction-free LU factoring because of the different output formats, which is also shown in the following benchmarks.

2.4 Benchmarks

Based on Algorithm 2.3.1 and Algorithm 2.3.2, we benchmark matrices with integer entries and univariate polynomial entries. We do not give a benchmark for matrices with multivariate polynomial entries because the running time is too long for a matrix whose size is larger than 10. We also give benchmarks for the MAPLE command for fraction-free LU. Because the implementation details are not published for the Maple command we use, we will only use it for comparison.

As we know, the main difference in Algorithm 2.3.1 and Algorithm 2.3.2 is in their use of divisions. In our codes, we use `divide` command for exact division in univariate polynomial case and `iquo` command for integer case instead of `normal` command. All results are obtained using the TTY version of MAPLE 10, running on an 2.8Ghz Intel P4 with 1024Megs of memory running Linux, and with time limit set to 2000 seconds for each fraction-free factoring operation, and garbage collection "frequency" (`gcfreq`) set to $2*10^7$ bytes. The detailed codes are in Appendix F.

In the following tables, we label the time used by Algorithm 2.3.1 as CFFLU, the time used by Algorithm 2.3.2 as PFFLU and the time used by the Maple command

`LUDecomposition(A, method = FractionFree)` as MapleLU. We also label the size of a matrix as n and the length of an integer entry of a matrix as ℓ.

For the matrices with integer entries, we have the following two tables. Table 2.4.2 gives the times of three algorithms on different sizes of matrices which are drawn in Figure 2.4.1. Table 2.4.3 gives the relations between the logarithm of the time of completely fraction-free LU factoring with the logarithm of the size of matrix. If we use the MAPLE command `Fit`$(a+bt, x, y, t)$ to fit Table 2.4.3, we find a slope equal to 4.335. This tells us that the relation between the time used by completely fraction-free LU factoring and the size of the matrix is $t = O(n^{4.335})$, i.e. by Theorem 2.3.8, we have $M(n \log n + 100n) = O(n^{1.335})$ in our implementation. We also could use block operations instead of explicit loops in our codes which is an efficient method in Matlab. But this is 10 times slower in MAPLE because the block notation in MAPLE is syntactic sugar for humans not for efficiency. We include our code using block operations in Appendix F.

For fixed-size matrices with integer entries, we have the following two tables. Table 2.4.4 gives the times of three algorithms on matrices with different length of integer entries which are drawn in Figure 2.4.3. Table 2.4.5 gives the relations between the logarithm of the time of completely fraction-free LU factoring with the logarithm of the length of integer entry. If we use the MAPLE command `Fit`$(a + bt, x, y, t)$ to fit Table 2.4.5, we find a slope equal to 1.265. This tells us that the relation between the time used by completely fraction-free LU factoring and the length of integer entry is $t = O(\ell^{1.265})$, i.e. by Theorem 2.3.8, we have $M(5 \log 5 + 5\ell) = O(\ell^{1.265})$ in our implementation.

For the matrices with univariate polynomial entries, we have the following two tables. Table 2.4.6 gives Figure 2.4.5. We can see that completely fraction-free LU

factoring is a little bit faster than the fraction-free LU factoring given in Algorithm 2.3.2. It has a similar speed of the Maple command. Table 2.4.7 gives the logarithms of the times used by completely fraction-free LU factoring and the sizes of matrices. If we use the MAPLE command `Fit`$(a + bt, x, y, t)$ to fit Table 2.4.7, we can have a slope equal to 5.187, i.e. by Theorem 2.3.8, we have $M(4n^2 + 4^2n) = O(n^{2.187})$ in our implementation.

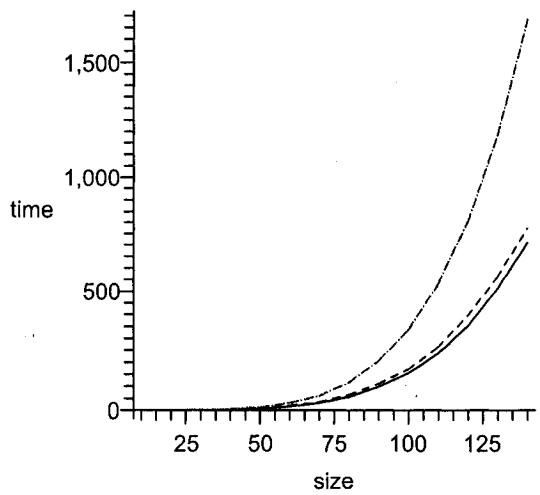

Figure 2.4.1: Times for random integer matrices. The line style of CFFLU is SOLID, the line style of PFFLU is DASH, and the line style of MapleLU is DASHDOT.

n	CFFLU	PFFLU	MapleLU
10	1.00E-02	1.40E-02	2.80E-02
20	0.133	0.188	0.263
30	0.735	0.959	1.375
40	2.617	4.127	4.831
50	6.94	8.622	13.182
60	15.672	19.574	31.342
70	32.003	35.938	61.974
80	56.598	65.553	117.111
90	97.881	110.931	208.665
100	155.757	171.989	335.407
110	237.916	264.304	531.193
120	351.737	396.445	803.898
130	512.428	562.352	1184.575
140	709.42	775.139	1684.698
150	975.429	1060.02	>2000
160	1298.419	1410.204	>2000
170	1714.003	1868.879	>2000

Table 2.4.2: Timings for fraction-free LU factoring of random integer matrices generated by `RandomMatrix(n,n, generator=` $-10^{100}..10^{100}$)

$\ln(n)$	ln(CFFLU)
2.302585093	-4.605170186
2.995732274	-2.017406151
3.401197382	-0.30788478
3.688879454	0.962028624
3.912023005	1.937301775
4.094344562	2.751875681
4.248495242	3.465829648
4.382026635	4.035973649
4.49980967	4.583752455
4.605170186	5.0482971
4.700480366	5.47191767
4.787491743	5.862883737
4.86753445	6.239160213
4.941642423	6.564447735
5.010635294	6.882877374
5.075173815	7.168902649
5.135798437	7.446586849

Table 2.4.3: Logarithms of timings for completely fraction-free LU factoring of random integer matrices from Table 2.4.2

ℓ	CFFLU	PFFLU	MapleLU
400	4.00E-03	6.00E-03	9.00E-03
800	7.00E-03	1.30E-02	1.80E-02
1200	1.00E-02	2.30E-02	3.10E-02
1600	1.40E-02	3.30E-02	4.80E-02
2000	2.20E-02	5.20E-02	6.90E-02
2400	2.80E-02	6.90E-02	9.30E-02
2800	3.60E-02	9.10E-02	0.121
3200	4.40E-02	0.114	0.155
3600	4.90E-02	0.137	0.19
4000	6.30E-02	0.17	0.231
4400	7.10E-02	0.195	0.272
4800	7.90E-02	0.235	0.334
5200	7.50E-02	0.249	0.37
5600	9.40E-02	0.29	0.425
6000	9.90E-02	0.324	0.485

Table 2.4.4: Timings for completely fraction-free LU factoring of random integer matrices with length ℓ, `A := LinearAlgebra[RandomMatrix](5,5, generator =` $-10^{\ell}..10^{\ell}$`)`

$\ln(\ell)$	ln(CFFLU)
5.99	-5.52
6.68	-4.96
7.09	-4.61
7.37	-4.27
7.60	-3.82
7.78	-3.58
7.94	-3.32
8.07	-3.12
8.19	-3.02
8.29	-2.76
8.39	-2.65
8.48	-2.54
8.56	-2.59
8.63	-2.36
8.70	-2.31

Table 2.4.5: Logarithms of timings for completely fraction-free LU factoring of random integer matrices from Table 2.4.4

n	CFFLU	PFFLU	MapleLU
5	8.00E-03	1.20E-02	2.50E-02
10	0.242	0.335	0.305
15	2.322	2.77	3.098
20	10.028	11.873	10.598
25	31.971	35.124	33.316
30	86.361	92.146	86.01
35	179.937	189.297	181.42
40	354.22	373.181	354.728
45	664.608	686.26	667.406
50	1178.374	1210.766	1179.499

Table 2.4.6: Timings for fraction-free LU factoring of random matrices with univariate entries, generated by `RandomMatrix(n,n, generator = (() -> randpoly([x], degree=4)))`.

$\ln(n)$	ln(CFFLU)
1.61	-4.83
2.30	-1.42
2.71	0.84
3.00	2.31
3.22	3.46
3.40	4.46
3.56	5.19
3.69	5.87
3.81	6.50
3.91	7.07

Table 2.4.7: Logarithms of timings for completely fraction-free LU factoring of random matrices from Table 2.4.6

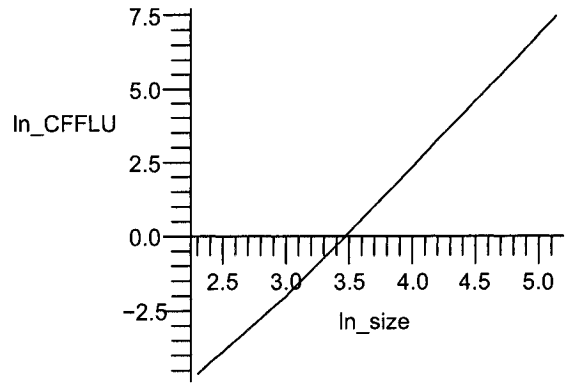

Figure 2.4.2: Logarithms of completely fraction-free LU time and integer matrix size, slope = 4.335.

2.5 Application of Completely Fraction-free LU Factoring

In this section, we will show the applications of our completely fraction-free LU factoring. Our first application of completely fraction-free LU factoring is to solve a symbolic linear system of equations in a single domain. We will introduce fraction-free forward and backward substitutions from [18]. Our second application is to get a new completely faction free QR factoring from our completely fraction-free LU factoring, with the relation between LU factoring and QR factoring from [20].

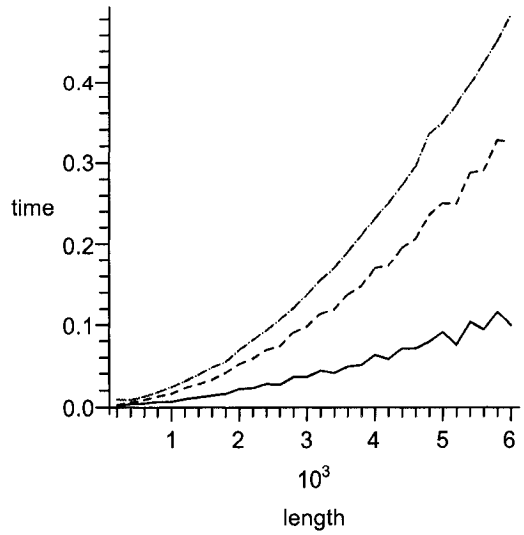

Figure 2.4.3: Time of random integer matrices. The line style of CFFLU is SOLID, the line style of PFFLU is DASH, and the line style of MapleLU is DASHDOT.

2.5.1 Fraction-free Forward and Backward Substitutions

In order to solve a linear system of equations in some domain, we need not only fraction-free LU factoring of coefficient matrix but also fraction-free forward substitution (FFFS) and fraction-free backward substitution (FFBS) algorithms.

Let A be a $n \times m$ matrix. The four matrices P, L, D, U, are from fraction-free LU factoring of the matrix A by Algorithm 2.3.1, where $PA = LD^{-1}U$, P is a $n \times n$ permutation matrix, L is a $n \times n$ lower triangular matrix, D is a $n \times n$ diagonal matrix, U is a $n \times m$ upper triangular matrix. In the following, we use these notations.

Definition 2.5.1. We define that our fraction-free forward substitution (FFFS) is equivalent to solving an equation $LD^{-1}Y = Pb$ for a vector Y.

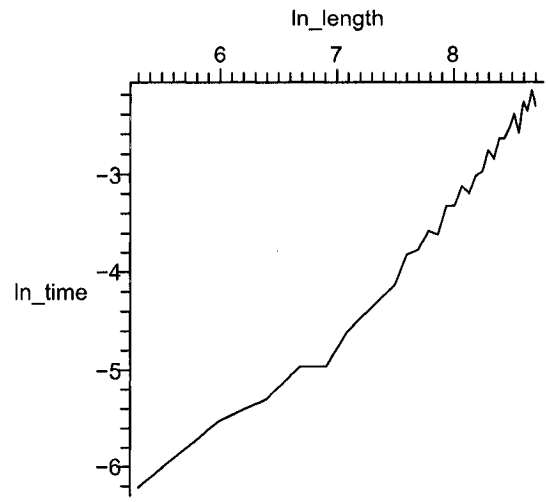

Figure 2.4.4: Logarithms of completely fraction-free LU time and integer length, slope = 1.265.

Theorem 2.5.2. *Y is a fraction-free vector in Definition 2.5.1.*

Proof. From theorem 2.2.2, if $PA = LD^{-1}U$, then for $i = 1...n$,

$$L_{i,i} = A_{i,i}^{(i-1)},$$

and

$$D_{i,i} = A_{i-1,i-1}^{(i-2)} A_{i,i}^{(i-1)}.$$

For

$$LD^{-1}Y = Pb,$$

i.e.

$$Y_i = \frac{D_{i,i}}{L_{i,i}} \left[b_{P_i} - \sum_{k=1}^{i-1} L_{i,k} Y_k \right],$$

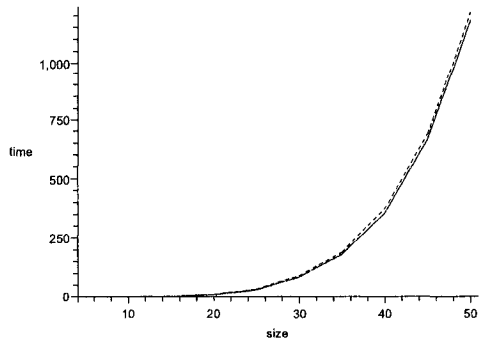

Figure 2.4.5: Time of random matrices with univariate entries. The line style of CFFLU is SOLID, the line style of PFFLU is DASH, and the line style of MapleLU is DASHDOT.

where $b_{P_i} = \sum_{j=1}^{n} P_{i,j} b_j$, it is obvious that $\frac{D_{i,i}}{L_{i,i}}$ is an exact division. So the solution Y_i is fraction-free for any $i = 1...n$. □

Definition 2.5.3. We define that our fraction-free backward substitution (FFBS) is equivalent to solving an equation $UX = U_{n,n}Y$ for a vector X.

Theorem 2.5.4. *X is a fraction-free vector in Definition 2.5.3.*

Proof. As we have proved in theorem 2.2.2, $U_{n,n}$ is the determinant of matrix A. Because

$$Ax = LD^{-1}Ux = Pb,$$

$$LD^{-1}Y = Pb,$$

and

$$UX = U_{n,n}Y,$$

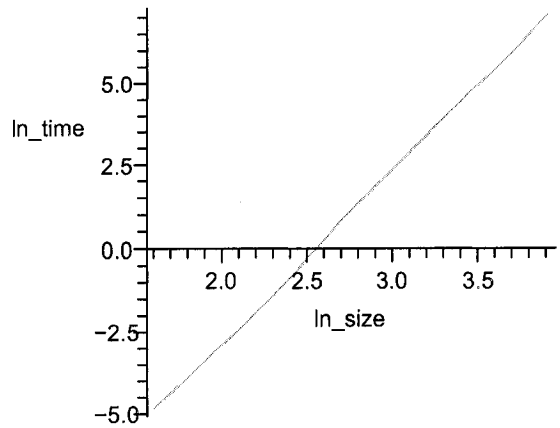

Figure 2.4.6: Logarithms of completely fraction-free LU time and random univariate polynomial matrix size, slope = 5.187

we see that X is a scaled solution and the true solution is the scaled solution divided by the determinant, i.e. $X = U_{n,n}x$, or $x = \frac{X}{\det(A)}$.

From Cramer's Rule, we know that the solution vector of a set of linear equations $Ax = b$ has a unique common denominator $\det(A)$. So we can say $X = \det(A)x$ is a fraction-free vector. □

If we have a set of linear systems with the same n by n matrix A, i.e. $Ax = b_1, Ax = b_2, Ax = b_3, ...$, after completely fraction-free LU decomposition of matrix $PA = LD^{-1}U$, the number of operations for solving a second linear system can be reduced from $O(n^3)$ to $O(n^2)$ for a floating point number entry matrix. Furthermore, for a symbolic entry matrix, such as integer matrix with a length of any entry

bounded by ℓ, the time complexity can be reduced from $O(n^3 M(n \log n + n\ell))$ to $O(n^2 M(n \log n + n\ell))$.

Here we give our fraction-free forward and backward substitution algorithms. Let A be a $n \times n$ matrix, L be a $n \times n$ lower triangular matrix, D be a $n \times n$ diagonal matrix, U be a upper triangular matrix, P be a $n \times n$ permutation matrix, and b be a $n \times 1$ vector, which satisfy the equations $Ax = b$ and $PA = LD^{-1}U$.

Algorithm 2.5.5. Fraction-free forward substitution

Input: Matrices L, D, P and a vector b

Output: A $n \times 1$ vector Y, such that $LD^{-1}Y = Pb$

For i from 1 to n do

$b_{P_i} = \sum_{j=1}^{n} P_{i,j} b_j$,

$Y_i = \frac{D_{i,i}}{L_{i,i}} [b_{P_i} - \sum_{k=1}^{i-1} L_{i,k} Y_k]$,

end loops;

Algorithm 2.5.6. Fraction-free backward substitution

Input: A matrix U, a vector Y from $LD^{-1}Y = Pb$.

Output: A scaled solution X

For i from n by -1 to 1 do

$X_i = \frac{1}{U_{i,i}} [U_{n,n} Y_i - \sum_{k=i+1}^{n} U_{i,k} X_k]$,

end loops;

2.5.2 Fraction-free QR Factoring

In a numerical context, the QR factoring of a matrix is well-known. We recall that $A = QR$ with Q being an orthonormal matrix having the property $Q^T Q = QQ^T = I$.

Pursell and Trimble [20] have pointed out that their algorithm for computing QR factoring using LU factoring is simple and has some surprisingly happy side effects

although it is not numerically preferable to existing algorithms. We use their idea to obtain fraction-free QR factoring using our completely fraction-free LU factoring. In the theorem 2.5.7, we prove the existence of fraction-free QR factoring and give an algorithm on how to use completely fraction-free LU factoring to get fraction-free QR factoring of a given matrix, i.e. how to orthogonalize a given set of vectors which are the columns of the matrix.

Theorem 2.5.7. *Let A be a $n \times m$ matrix whose columns are linearly independent vectors in $\mathbb{I}^n$, where $\mathbb{I}$ is an integral domain. There exist an $m \times m$ lower triangular matrix L, an $m \times m$ diagonal matrix D, an $m \times m$ upper triangular matrix U and an $n \times m$ left orthogonal matrix Θ, such that*

$$[A^T A | A^T] = LD^{-1}[U|\Theta^T]. \tag{2.5.1}$$

Then the fraction-free QR factoring of A is

$$A = \Theta D^{-1} R, \tag{2.5.2}$$

where $R = L^T$.

Proof. The linear independence of the columns of A implies that $A^T A$ is a full rank matrix. Hence, there are $m \times m$ fraction-free matrices, L, D and U using our new fraction-free LU factoring. That is

$$A^T A = LD^{-1}U \tag{2.5.3}$$

Applying the same row operations to the augmented matrix $[A^T A|A^T]$, we have a fraction-free $m \times n$ matrix. Let us define this fraction-free $m \times n$ matrix as Θ^T, which is

$$\Theta^T = DL^{-1}A^T, \tag{2.5.4}$$

then

$$\Theta^T\Theta = (DL^{-1}A^T \cdot A(DL^{-1})^T) = DL^{-1}(LD^{-1}U)(DL^{-1})^T = U(DL^{-1})^T.$$

Because $\Theta^T\Theta$ is symmetric and both U and $(DL^{-1})^T$ are upper triangular matrices, $\Theta^T\Theta$ must be a diagonal matrix. So the columns of Θ are left orthogonal.

Based on the equation 2.5.4, we have $A^T = LD^{-1}\Theta^T$. i.e. $A = \Theta(LD^{-1})^T = \Theta(D^T)^{-1}L^T = \Theta D^{-1}L^T$, set $R = L^T$, then R is a fraction-free upper triangular matrix. The fraction-free QR factoring for the matrix A is $A = \Theta D^{-1}R$.

□

In the following algorithm 2.5.8, we assume the existence of a function CFFLU implementing Algorithm 2.3.1.

Algorithm 2.5.8. Fraction-free QR factoring

Input: A $n \times m$ matrix A

Output: Three matrices Θ, D, R, where Θ is an $n \times m$ left orthogonal matrix, D is an $m \times m$ diagonal matrix, R is an $m \times m$ upper triangular matrix and $A = \Theta D^{-1}R$

1. compute $B := [A^TA|A^T]$
2. $L, D, U :=$ `CFFLU`(B)
3. for $i = 1...m, j = 1...n \quad \Theta_{j,i} := U_{i,m+j}, \quad$ end loop
4. for $i = 1...m, j = 1...m \quad R_{j,i} := L_{i,j}, \quad$ end loop □

Example 2.5.9. We revisit the example in Pursell and Trimble [20].

$$a_1 := \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}, \quad a_2 := \begin{bmatrix} -2 \\ 3 \\ 0 \\ 1 \end{bmatrix}, \quad a_3 := \begin{bmatrix} 1 \\ 1 \\ 1 \\ 5 \end{bmatrix}.$$

Let A be the 4×3 matrix containing a_k as its k^{th} column. Then

$$A^T A = \begin{bmatrix} 2 & 4 & 6 \\ 4 & 14 & 6 \\ 6 & 6 & 28 \end{bmatrix}$$

Applying the fraction-free row operation matrices L and D to the augmented matrix $[A^T A | A^T]$, we have the fraction-free matrix U, where

$$L = \begin{bmatrix} 2 & & \\ 4 & 12 & \\ 6 & -12 & 1 \end{bmatrix}, \quad D = \begin{bmatrix} 2 & & \\ & 24 & \\ & & 12 \end{bmatrix}, \quad U = \begin{bmatrix} 2 & 4 & 6 & 0 & 1 & 0 & 1 \\ & 12 & -12 & -4 & 2 & 0 & -2 \\ & & 48 & -12 & -12 & 12 & 12 \end{bmatrix}$$

Notice that the matrix U differs from that given in Pursell and Trimble [20], which is as follows:

$$\left[\begin{array}{ccc|cccc} 2 & 4 & 6 & 0 & 1 & 0 & 1 \\ & 6 & -6 & -2 & 1 & 0 & -1 \\ & & 4 & -1 & -1 & 1 & 1 \end{array}\right].$$

This is because they implicitly divided each row by the GCD of the row. They also observed for this example that cross-multiplication is not needed during Gaussian elimination and the squares of the each row on the right side is equal to the diagonal of the left side of the matrix. However, these observations are specific to their particular numerical example. Any change to these initial vectors will invalidate their observation, as example 2.5.10 will show.

Now from our fraction-free QR Theorem 2.5.7, we have

$$R = L^T = \begin{bmatrix} 2 & 4 & 6 \\ & 12 & -12 \\ & & 1 \end{bmatrix}, \Theta^T = \begin{bmatrix} 0 & 1 & 0 & 1 \\ -4 & 2 & 0 & -2 \\ -12 & -12 & 12 & 12 \end{bmatrix} \text{ or } \Theta = \begin{bmatrix} 0 & -4 & -12 \\ 1 & 2 & -12 \\ 0 & 0 & 12 \\ 1 & -2 & 12 \end{bmatrix}$$

So the fraction-free QR factoring of matrix A is as follows.

$$\Theta D^{-1}R = \begin{bmatrix} 0 & -4 & -12 \\ 1 & 2 & -12 \\ 0 & 0 & 12 \\ 1 & -2 & 12 \end{bmatrix} \begin{bmatrix} 2 & & \\ & 24 & \\ & & 12 \end{bmatrix}^{-1} \begin{bmatrix} 2 & 4 & 6 \\ & 12 & -12 \\ & & 1 \end{bmatrix} = \begin{bmatrix} 0 & -2 & 1 \\ 1 & 3 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 5 \end{bmatrix} = A$$

□

Example 2.5.10. We slightly change the vector a_1 and keep the other vectors the same as in example 2.5.9.

$$a_1 := \begin{bmatrix} 0 \\ 2 \\ 0 \\ 1 \end{bmatrix}, \quad a_2 := \begin{bmatrix} -2 \\ 3 \\ 0 \\ 1 \end{bmatrix}, \quad a_3 := \begin{bmatrix} 1 \\ 1 \\ 1 \\ 5 \end{bmatrix}.$$

Let C be the 4×3 matrix containing a_k as its k^{th} column. Then

$$C^TC = \begin{bmatrix} 5 & 7 & 7 \\ 7 & 14 & 6 \\ 7 & 6 & 28 \end{bmatrix}$$

Applying the fraction-free row operation matrices L and D to the augmented matrix $[C^TC|C^T]$, we have the fraction-free matrix U, where

$$L = \begin{bmatrix} 5 & & \\ 7 & 21 & \\ 7 & -19 & 1 \end{bmatrix}, \quad D = \begin{bmatrix} 5 & & \\ & 105 & \\ & & 21 \end{bmatrix}, \quad U = \begin{bmatrix} 5 & 7 & 7 & 0 & 2 & 0 & 1 \\ & 21 & -19 & -10 & 1 & 0 & -2 \\ & & 310 & -17 & -34 & 21 & 68 \end{bmatrix}$$

We can verify that the square of the second row on the right side of matrix U is not equal to the diagonal of the left side of the matrix U. It means the observation of Pursell and Trimble is not valid in this example.

The fraction-free QR factoring of matrix C is as follows.

$$\Theta D^{-1}R = \begin{bmatrix} 0 & -10 & -17 \\ 2 & 1 & -34 \\ 0 & 0 & 21 \\ 1 & -2 & 68 \end{bmatrix} \begin{bmatrix} 5 & & \\ & 105 & \\ & & 21 \end{bmatrix}^{-1} \begin{bmatrix} 5 & 7 & 7 \\ & 21 & -19 \\ & & 1 \end{bmatrix} = \begin{bmatrix} 0 & -2 & 1 \\ 2 & 3 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 5 \end{bmatrix} = C$$

□

2.6 Discussions and Conclusions

The main contributions of this paper are firstly the definition of a new fraction-free format for matrix LU factoring, with a proof of the existence of the format; then from the proof, showing one general way for computing the new format and giving the complexity analysis on the two main algorithms.

One of the difficulties constantly faced by computer algebra systems is the perception of new users that numerical techniques carry over unchanged into symbolic contexts. This can force inefficiencies on the system. For example, if a system tries to satisfy user preconceptions that Gram—Schmidt factoring should still lead to QR factors with Q orthonormal, then the system must struggle with square-roots and fractions. By defining alternative forms for the factors, the system can take advantage of alternative schemes of computation.

In this paper, the application of the completely fraction-free LU factoring format gives a new fraction-free QR factoring format which avoid the square-roots and fractions problems for QR factoring. On the other hand, together with the fraction-free forward and backward substitutions, fraction-free LU factoring gives one way to solve linear systems in their own input domains.

Fraction-free Gaussian elimination alleviates the intermediate expression swell from Gaussian elimination. As well, there are lots of other methods to control the expression swell problems, such as hierarchical representations, p-adic algorithms, Chinese Remaindering algorithms. All these methods are well adapt to different application backgrounds.

In the next chapter, we will discuss how to use hierarchical representation to avoid the large expression problem during LU factoring or Gaussian elimination.

Bibliography

[1] Bareiss, E. H. Sylvester's Identity and Multistep Integer-preserving Gaussian Elimination. *Mathematics of Computation*, 22(103), 565-578, 1968.

[2] Bareiss, E. H. Computational Solutions of Matrix Problems over an Integral Domain. *J. Inst. Maths Applics.*, 10, 68-104, 1972.

[3] Corless, R. M. and Jeffrey, D. J. The Turing Factorization of a Rectangular Matrix. *SIGSAM Bull.*, ACM Press, 31(3), 20-30, 1997.

[4] Demmel, J. W. Applied Numerical Linear Algebra. *Society for Industrial and Applied Mathematics*, 1997.

[5] Erlingsson, Ú., Kaltofen, E. and Musser, D. Generic Gram—Schmidt Orthogonalization by Exact Division. *ISSAC'96*, ACM, 275-282, 1996.

[6] von zur Gathen, J. and Gerhard, J. Modern Computer Algebra. *Cambridge University Press*, 1999.

[7] Geddes, K. O., Labahn, G. and Czapor, S. Algorithms for Computer Algebra. *Kluwer*, 1992.

[8] Gentleman, W. M. and Johnson, S. C. Analysis of Algorithms, A Case Study: Determintants of Polynomials. *Proc 5th Annual ACM Symp on Theory of Computing* (Austin, Tex.), 135-142, 1973.

[9] Golub, G and Van Loan, C. Matrix Computations, 2nd edition. *Johns Hopkins Series in the Mathematical Sciences*, 1989.

[10] Griss, M. L. An Efficient Sparse Minor Expansion Algorithm. *ACM* (Houston, Tex.), 429-434, 1976.

[11] Higham, N. J. Accuracy and Stability of Numerical Algorithms. *Society for Industrial & Applied Mathematics*, 1996.

[12] Higham, N. J. Accuracy and Stability of Numerical Algorithms. *SIAM*, 2002.

[13] Kirsch, B. J. and Turner, P. R. Modified Gaussian Elimination for Adaptive Beamforming Using Complex RNS Arithmetic. *NAWC-AD Tech Rep.*, NAWCADWAR 94112-50, 1994.

[14] Kirsch, B. J. and Turner, P. R. Adaptive Beamforming Using RNS Arithmetic. *Pro ARTH 11*, IEEE Computer Society, Washington DC, 36-43, 1993.

[15] Krick, T., Pardo, L. M. and Sombra, M. Sharp Estimates for the Arithmetic Nullstellensatz. *Duke Mathematical Journal*, 109(3), 521-598, 2001.

[16] Lenstra, A. K., Lenstra, H. W. and Lovász, L. Factoring Polynomials with Rational Coefficients. *Math. Ann.*, 261, 1982.

[17] McClellan, M. T. The Exact Solution of Systems of Linear Equations with Polynomial Coefficients. *J ACM*, 20(4), 563-588, 1973.

[18] Nakos, G. C., Turner, P. R. and Williams, R. M. Fraction-free Algorithms for Linear and Polynomial Equations. *SIGSAM Bull.*, ACM Press, 31(3), 11-19, 1997.

[19] Noble, B. and Daniel, J. W. Applied Linear Algebra, 3^{rd} edition. *Prentice-Hall, Englewood Cliffs, N.J.*, 1988.

[20] Pursell, L. and Trimble, S. Y. Gram—Schmidt Orthogonalization by Gaussian Elimination. *American Math. Monthly*, 98(6), 544-549, 1991.

[21] Rice, J. R. Experiments on Gram-Schmidt Orthogonalization. *Math. Comp.*, 20, 325-328, 1996.

[22] Sasaki, T. and Nurao, H. Efficient Gaussian Elimination Method for Symbolic Determinants and Linear Systems. *ACM Transactions on Mathematical Software*, 8(3), 277-289, 1982.

[23] Schönhage, A. and Strassen, V. Schnelle Multiplikation großer Zahlen. *Computing*, 7, 281-292, 1971.

[24] Smit, J. The Efficient Calculation of Symoblic Determinants. *ISSAC'76*, ACM, 105-113, 1976.

[25] Smit, J. A Cancellation Free Algorithm, with Factoring Capabilities, for the Efficient Solution of Large Sparse Sets of Equations. *ISSAC'81*, ACM, 146-154, 1981.

[26] Trefethen, L. N. and Bau, III D. Numerical Linear Algebra. *SIAM*, 1997.

[27] Turing, A. M. Rounding-off Errors in Matrix Processes. *Quart. J. Mech. Appl. Math.*, 1, 287-308, 1948.

[28] Turner, P. R. Gauss Elimination: Workhorse of Linear Algebra. *NAWC-AD Tech Rep*, NAWCAD-PAX 96-194-TR, 1996.

[29] Zhou, W., Carette, J., Jeffrey, D. J. and Monagan, M. B. Hierarchical Representations with Signatures for Large Expression Management. *AISC*, Springer-Verlag LNAI 4120, 254-268, 2006.

Chapter 3

Hierarchical Representations with Signatures for Large Expression Management

We describe a method for managing large expressions in symbolic computations which combines a hierarchical representation with signature calculations. As a case study, the problem of factoring matrices with non-polynomial entries is studied. Gaussian Elimination is used. Results on the complexity of the approach together with benchmark calculations are given.

3.1 Introduction

One of the attractions of MAPLE is that it allows users to tackle larger problems than they can do by hand. However, when users undertake large-scale calculations, they often find that expression swell can limit the size of the problems they can solve [29]. Typically, users might meet two types of expression swell: one type we can call *inherent* expression swell, and the other *intermediate* expression swell.

A number of strategies have been proposed for coping with the large expressions generated during symbolic computation. We list a number of them here, but lack of

space precludes an extensive discussion.

- *Avoid the calculation.* This strategy delays computation of a quantity whose symbolic expression is large until numerical data is given. For example, if the determinant of a matrix is needed in a computation, one uses an inert function until the point at which the elements of the matrix can be evaluated numerically, and then jumps to a numerical evaluation.

- *Use signatures.* See, for example, [21]. Signatures are one of the ideas used in this paper.

- *Use black-box calculations.* This is a strength of the Linbox project [12].

- *Approximate representations.* This is the growing area of symbolic-numeric computation.

- *Use hierarchical representations.* These are studied in this paper, and the term will be abbreviated to HR.

Each of the above methods is successful for a different class of problems. This paper addresses a class of problems in which large expressions are built up from identifiable sub-expressions, and which as a result are suitable applications for hierarchical representations (HR). Hierarchical representations *per se* are not new in computer algebra. Similar ideas have appeared in the literature under a variety of names. Examples are as follows:

- *Maple Directed Acyclic Graphs.* Expressions in MAPLE are represented as DAGs with sub-expressions being reused and hence stored only once. For example, [8] uses this data structure to help compute the determinant of polynomial matrices.

- *Straight-line programs.* The DAGWOOD [6] system computes with straight-line programs.

- *Common subexpression identification.* The MAPLE command `codegen[optimize]` searches a large expression for common subexpressions. (Also available as an option to commands in `CodeGeneration`) [24]

- *Computation sequences and* MAPLE*'s* ***`CompSeq`****.* An early example is given by Zippel in 1993 [31]. The function `CompSeq` in MAPLE is a placeholder for representing a computation sequence.

- *Large Expression Management (LEM).* This term was introduced in [5], and is the name of a MAPLE package.

The goal of this work is the combination of HR with signatures. We do this by modifying the LARGEEXPRESSIONS package in MAPLE and then applying it to a case study. The case study comes from DYNAFLEX [13], a system which computes the equations of motion for a mechanical device created from rigid or flexible bodies. It uses MAPLE for its computations and requires the factoring of matrices whose elements are multivariate polynomials or non-polynomial functions. In this paper, therefore, we consider the factoring of matrices with elements that are multivariate polynomials and exponential polynomials. We could have considered any application where the algorithm at hand only requires zero-recognition on the elements (as well as basic "arithmetic" operations); if obtaining other information, like degree or structural "shape" is absolutely necessary, this would need new ideas on top of the ones we present here.

3.2 Hierarchical Representation

The first point to establish is the need for a modified HR implementation. We begin by giving our definition of HR for this paper, with the purpose of distinguishing our implementation from similar definitions, such as straight-line programs.

Definition 3.2.1. [Exponential Polynomial] An exponential polynomial is an element of $\mathbb{Z}[X_1, \ldots, X_n, a_1^{X_1}, \ldots, a_n^{X_n}]$, where $a_1, \ldots, a_n$ are positive real numbers.

Definition 3.2.2. [Hierarchical Representation] A hierarchical representation (HR) over a domain $\mathbb{K}$ and a set of independent variables $\{x_1, ..., x_m\}$ is an ordered list $[S_1, S_2, ..., S_\ell]$ of symbols, together with an associated list $[D_1, D_2, ..., D_\ell]$ of definitions of the symbols. For each S_i with $i \geq 1$, the definition D_i has the form $S_i = f_i(\sigma_{i,1}, \sigma_{i,2}, ..., \sigma_{i,k_i})$ where $f_i \in \mathbb{K}[\sigma_{i,1}, ..., \sigma_{i,k_i}]$, and for each $\sigma_{i,j}$ is either a symbol in $[S_1, S_2, ..., S_{i-1}]$ or an exponential polynomial in the independent variables.

Hierarchical representation is a more general idea than the (algebraic) straight-line program defined in [17] and used in [18, 6, 15, 19]. A given expression can have different HR, i.e. different lists of definitions $[D_1, D_2, ...]$. The strategy used to assign the symbols during the generation of expressions will be something that can be varied by the implementation. The reason for requiring an ordered list is to exclude implicit definitions. Details on how to build HR are in the section 3.4.

Remark 3.2.3. An important part of the creation of HRs is the order in which assignments happen. For instance in order to use `codegen[optimize]`, an expression must be completely generated first. Clearly, some expressions will be too large to be generated explicitly, in which case `codegen[optimize]` would have nothing to work with.

Remark 3.2.4. There are many types of computational procedures which naturally generate HR. One example is Gaussian elimination, which we study here. Another is the calculation described in [5]. Other computations that are known to generate large expressions, for example Gröbner basis calculations, do not have a obvious hierarchy, although [8] hints at one.

Remark 3.2.5. One can understand HRs as a compromise between *full computations* and *no computations*. Enough of the computation is performed to give a correct result, but not so much that a closed-form can be output. It is a compromise between immediately returning a placeholder and never returning a giant result.

The key issue is control over expression simplification; this includes the identification of a zero expression. In an ordinary computer algebra system, the usual way this proceeds is by normalizing expressions, a step which frequently destroys the HR and causes the appearance of additional expression swell. For example, most systems will normalize[1] the expression

$$(2781 + 8565x - 4704x^2)^{23}(1407 + 1300x - 1067x^2)^{19} - \alpha \cdot$$
$$(1809 + 9051x + 9312x^2)^{19}(2163 - 2162x + 539x^2)^{19}(27 + 96x)^4(103 - 49x)^4$$

by expanding it. The same strategy would be used by the system whether $\alpha = +1$ or $\alpha = -1$. However, in one case the result is zero, while in the other it is just a large expression, which now fills memory.

Consequently, the main purpose of creating user-controlled HR is to control normalization and to integrate different (often more efficient) zero-testing strategies into a computation in a convenient way. As well as creating a HR, one must give equal importance to the prevention of its destruction.

[1] normalization is often confused with simplification, but [3] argues otherwise.

The original LARGEEXPRESSIONS package in MAPLE was created as a result of the investigations in [5]. The authors had external mathematical reasons for knowing that their expressions were nonzero, and hence no provision was made for more efficient testing. In the current implementation, we intend to apply the resulting code more widely, with the consequent need to test efficiently for zero. This we do by incorporating signature testing.

The basic action is the creation of a label for a sub-expression. The command for this was given the name `Veil` in the original LARGEEXPRESSIONS package, and so this will be used as the general verb here. Once an expression has been veiled, the system treats it as an inert object unless the user or the program issues an unveiling command, which reveals the expression associated with the label.

3.3 Signatures

The idea of using signatures is similar to the probabilistic identity testing of the Zippel-Schwartz theorems [30, 27], and to the basis of `testeq` in MAPLE by Gonnet [9, 10], also studied in [21, 22]. The original polynomial results of Zippel-Schwartz were extended to other functions in [9, 10, 23].

Since we need to apply our method to matrices containing exponential polynomials, we first define a signature function that is appropriate for this class of functions.

Definition 3.3.1. [Signature of an Exponential Polynomial] Let P be in $\mathbb{Z}[X_1, \ldots, X_n, a_1^{X_1}, \ldots, a_n^{X_n}]$, where $X_1, \ldots, X_n$ are independent variables and $a_1, \ldots, a_n$ are positive real numbers. Let p be a prime, $x_1, \ldots, x_n$ be in $\mathbb{Z}/p\mathbb{Z}$, $y_1, \ldots, y_n$ be in $\mathbb{Z}/(p-1)\mathbb{Z}$ and t be a primitive root of p. Then the signature of P is defined below:

$$s(P) = P(x_1, \ldots, x_n, t^{y_1}, \ldots, t^{y_n}) \mod p.$$

Definition 3.3.2. [Signature of a Hierarchical Representation] Let H be a hierarchical representation having independent variables $X_1, \ldots, X_n$, given by lists $[S_1, \ldots, S_\ell]$ and $[D_1, \ldots, D_\ell]$ with the form $S_i = f_i(\sigma_{i,1}, \sigma_{i,2}, ..., \sigma_{i,k_i})$ with $f_i \in \mathbb{Z}[\sigma_{i,1}, ..., \sigma_{i,k_i}]$, and for each $\sigma_{i,j}$ is either a symbol in $[S_1, S_2, ..., S_{i-1}]$ or an exponential polynomial in the independent variables. Let p be a prime, $x_1, \ldots, x_n$ be in $\mathbb{Z}/p\mathbb{Z}$, $y_1, \ldots, y_n$ be in $\mathbb{Z}/(p-1)\mathbb{Z}$ and t be a primitive root of p.

The signature of S_i is defined inductively as follows:

- If $\sigma_{i,j} \in \mathbb{Z}[X_1, \ldots, X_n, a_1^{X_1}, \ldots, a_n^{X_n}]$, then $\delta_{i,j} = s(\sigma_{i,j})$.
- If $\sigma_{i,j}$ is a symbol in $[S_1, S_2, ..., S_{i-1}]$, then we necessarily have $i > 1$. Let $h_{i,j} \in [1, \cdots, i-1]$ be such that $\sigma_{i,j} = S_{h_{i,j}}$, then $\delta_{i,j} = s(\sigma_{i,j}) = s(S_{h_{i,j}})$ which is known by induction assumption.
- We define $s(S_i) = f_i(\delta_{i,1}, \cdots, \delta_{i,k_i}) \mod p$.

The signature of H is defined as $s(H) = [s(S_1), \cdots, s(S_\ell)]$.

Note that, unlike [9], we explicitly do not treat towers of exponentials, but only simple exponentials, which is frequently sufficient in applications.

Proposition 3.3.3. *For all non-zero $y \in \mathbb{Z}/p\mathbb{Z}$, there exists a unique $x \in \mathbb{Z}/(p-1)\mathbb{Z}$, s.t. $s(a^x) = y$.*

Proof. By the definition of a signature $s(a^x)$, $r = s(a)$ is a primitive root modulo p. By the definition of a primitive root of a prime [28], the multiplicative order of r modulo p is equal to $p-1$. So the powers r^i, $i = 1..p-1$ range over all elements of $\mathbb{Z}/p\mathbb{Z} - \{0\}$. □

For the following theorems, we suppose that all random choices are made under the uniform probability distribution.

Theorem 3.3.4. *(Zippel-Schwartz theorem) Given* $F \in \mathbb{Z}[x_1, ..., x_n]$, $F \mod p \neq 0$, *and* $\deg(F) \leq d$, *the probability* $Pr\{s(F) = 0 | F \neq 0\} \leq \frac{d}{p}$.

A proof can be found in [30, 27].

Theorem 3.3.5. *Let* $F \in \mathbb{Z}[y], y = a^x$, *where a could be the base of the natural logarithms or any (non-zero) number less than p,* $F \mod p \neq 0, \deg(F) = d$, *the probability* $Pr\{s(F) = 0 | F \neq 0\} \leq \frac{d}{p-1}$.

Proof. The polynomial $F \in \mathbb{Z}[y]$ has at most d roots in $\mathbb{Z}/p\mathbb{Z}$. For $z \in \mathbb{Z}/p\mathbb{Z}$ such that $F(z) = 0$, Proposition 3.3.3 gives that there exists unique $u_z \in \mathbb{Z}/(p-1)\mathbb{Z}$, s.t. $s(a^{u_z}) = z$. Thus the number of values x such that $s(F(a^x)) = 0$ is at most d. Because the total number of choices for nonzero y is $p-1$, the probability $\Pr\{s(F) = 0 | F \neq 0\} \leq \frac{d}{p-1}$. □

Theorem 3.3.6. *Let* $F \in \mathbb{Z}[x, y], y = a^x$, *where a could be the base of natural logarithms or any non-zero integer less than p,* $F \mod p \neq 0$, *and* $\deg(F) = d$, *the probability* $Pr\{s(F) = 0 | F \neq 0\} \leq \frac{d}{p-1}$.

Proof. The polynomial $F \in \mathbb{Z}[x, y]$ has at most dp roots in $\mathbb{Z}/p\mathbb{Z}$. For (x_i, y_i) such that $F(x_i, y_i) = 0$, based on Proposition 3.3.3, there exists unique x_u, s.t. $s(a^{x_u}) = y_i$. If $x_u = x_i$, then the solution (x_i, y_i) is the one to make $s(F(x_i, a^{x_i})) = 0$. Therefore the number of roots for x, such that $s(F(x, a^x)) = 0$ is at most dp.

As the total number of (independent) choices for (x, y) is $p(p-1)$, the probability $\Pr\{s(F) = 0 | F \neq 0\} \leq \frac{d.p}{p(p-1)} \doteq \frac{d}{p-1}$. □

Signatures can be used to test if an expression is zero, as `testeq` does. However, `testeq` always starts fresh for each new zero-test. This is a source of inefficiency when the signature is part of a continuing computation, and will be seen in later benchmarks which use `testeq`.

The signature of the expression is computed before veiling an expression in HR. This value then becomes the signature of the veiling symbol. When that symbol itself appears in an expression to be veiled, the signature of the symbol is used in the calculation of the new signature. In particular, it is not necessary to unveil any symbol in order to compute its signature.

Other important references on this topic are [20, 11, 16]. Applications of this basic test is the determination of singularity and rank of a matrix [14] shows two applications of this basic technique: determining whether a matrix (of polynomials) is singular, and determining the rank of a polynomial matrix.

3.4 An Implementation of HR with Signatures

The simplest method for tracking HRs is to maintain an association list between an expression and its (new) label. This is easily implemented via (hash) tables; one table associates a "current" number to a symbol (used as an indexed name to generate fresh labels), and another table which associates to each indexed name to the underlying (unveiled) expression. The indexed names play the role of the ordered list of symbols in definition 3.2.2. The main routine is `Veil[K](A)`. Here K is the symbol and A is the expression to be veiled. This routine stores the expression A in the slot associated to $K[c]$ where c is the "current" number, increments c and returns the new symbol. For interactive use, a wrapper function `subsVeil` can be used.

```
> subsVeil:=(e,A)->algsubs(e=Veil[op(procname)](e),A);
> A:= (x+y)^{10} + e^{x+y} + (x^2+1)^5 - 1:
> B:= subsVeil[K](x^2+1,A):
```

```
> C:= subsVeil[K](x+y, B);
```

$$k_2^{10} + e^{k_2} + k_1^5 - 1$$

Notice that there is no longer a danger of expanding the expression $(x^2+1)^5 - 1$ in a misguided attempt to simplify it. In order to retrieve the original expression, one uses `Unveil`.

```
> Unveil[K](C) ;
```

$$(x+y)^{10} + e^{x+y} + (x^2+1)^5 - 1$$

At present, the expressions corresponding to K are stored in the memory space of the implementation module[2]. After a computation is completed and the intermediate results are no longer needed, the memory occupied by K can be cleared using the command `forgetVeil(K)`.

The signature must be remembered between calls to `Veil`, as commented above. The signature could be attached more directly to K, or kept in a separate array specified by the user. The above implementation seemed to provide the best *information hiding*. Until we see, with more experience of case studies, which method is best, we have for the present implementation used the MAPLE facility `option remember` internally for handling some of the tables, for convenience and efficiency. Thus after a call to the routine `SIG`, the signature of any veiled expression is stored in an internal remember table and not re-computed.

The use of `Veil` to generate HRs together with the calculation of signatures will be called Large Expression Management (LEM). In fact it is just expression management, because the `Veil` tool can be used even on expressions which are not large, for the convenience they give to understanding algebraic structure.

[2]In other words, it is a stateful module, à la Parnas, which is also rather like a singleton class in OO.

3.5 LU Factoring with LEM

A well-known method for solving matrix equations is LU factoring, in which a matrix A is factored such that $PA = LU$, where L and U are triangular matrices and P is a permutation matrix; see [25] for further details. The MAPLE command `LUDecomposition` uses large amounts of memory and is very slow for even moderately sized matrices of polynomials. The large expression trees generated internally are part of the reason for this slowdown, but equally significant is the time taken to check for zero. For example,

```
> M :=Matrix(10,10,symbol=m);
> LinearAlgebra[LUDecomposition](M);
```

This LU factoring will not terminate. If the environment variable `Normalizer` is changed from its default of `normal` to the identity function, i.e. Normalizer:= $x->$ x, then the LU factoring can complete. This is why Large Expression Management requires both HR and signatures for its zero-test.

We modified the standard code for LU decomposition to include veiling and signature calculations. At the same time, we generalized the options for selecting pivots and added an option to specify a veiling strategy. One can see [4] for even more design points, and a general design strategy, for this class of algorithms.

Our LU factoring algorithm in high-level pseudo-code:

```
Get maximum_column, maximum_row for matrix A
  For current_column from 1 to maximum_column
    for current_row from current_column to maximum_row
      Check element for zero.
      Test element for being ''best'' pivot
```

```
      Veil pivot [invoke Veiling strategy]
      move pivot to diagonal, recording interchanges.
      row-reduce matrix A with veiling strategy
      store multipliers in L
    end do:
  end do:
return permutation_matrix, L, reduced matrix A
```

The function has been programmed with the following calling sequence.

```
   LULEM(A, K, p, Pivoting, Veiling, Zerotesting)
Parameters
   A           - square matrix
   K           - unassigned name to use as a label
   p           - prime
   Pivoting    - decide a pivot for a column
   Veiling     - decide to veil an expression or not
   Zerotesting - decide if the expression is zero.
```

3.5.1 Pivoting Strategy

The current MAPLE LUDecomposition function selects one of two pivoting strategies on behalf of the user, based on data type. Thus, at present, we have

```
> LUDecomposition(<<12345,1>|<1,1>>);
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1/12345 & 1 \end{bmatrix}, \begin{bmatrix} 12345 & 1 \\ 0 & 12344/12345 \end{bmatrix}$$

even though

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 12345 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & -12344 \end{bmatrix}$$

is more attractive. If the matrix contains floating-point entries, partial pivoting is used.

```
> LUDecomposition(<<1,12345.>|<1,1>>);
```

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1. & 0. \\ (8.1)10^{-5} & 1. \end{bmatrix}, \begin{bmatrix} 12345. & 1 \\ 0 & 0.99992 \end{bmatrix}$$

Since we wished to experiment with different pivoting strategies, we made it an option. Rather than make up names, such as 'partial pivoting' or 'non-zero pivoting', to describe strategies, we allow the user to supply a function which takes 2 arguments. The function returns true if the second argument is a preferred pivot to the first argument. For example, the preferred pivoting strategy for the example above (choose the smallest pivot) can be specified by the function `(p1,p2)->evalb(abs(p2)<abs(p1))`. In a symbolic and veiling context there are a number of conceivable strategies which one might wish to try. These can be based on operation count, size of expression or number of indeterminants. However, the definition of LU factors only allows pivoting on one column, so no form of full pivoting is offered.

3.5.2 Veiling Strategy

In the same spirit of experimentation, we have used a function to specify a veiling strategy. This function takes one argument and returns true if the expression should be veiled. The current LARGEEXPRESSIONS package, for example, follows a strategy of ignoring integers. Thus an integer, however large, cannot be veiled at present.

Similarly, integer content is extracted from expressions before veiling. Rather than make these decisions in advance, we leave them to the declaration of a veiling-strategy function.

Of particular interest is the 'granularity' of the HR, namely whether one veils every pairwise operation, or whether one waits until an expression of a pre-determined size is allowed to accumulate. In the former case, the HR would look similar to a straight-line program as defined in [17]. For our experiments, we have based our strategies on the MAPLE `length` command, as being a convenient measure of expression complexity.

3.5.3 Zero Test Strategy

We need to do zero tests to find pivots. This can also help us simplify our expressions, if needed. During the LU factoring, we use signatures to perform this test quickly (more precisely, in random polynomial time). It is important to note that for LU factoring, we only need to find a provably non-zero pivot, so that a false positive (an entry which seems to be zero but in fact is not) rarely leads to a problem. And, in that case, we can always resort to a full zero-test.

We use the signatures computed along with the hierarchical representations to do the zero test for the expressions in HR. But a user could choose any MAPLE commands, like `Normalizer`, `testeq`, `simplify` or `evalb`, to do the zero test. Which one is best depends on the application at hand.

3.6 Time Complexity Analysis for LU with Veiling and Signatures

Since our case study compares current LU factoring and LU factoring with expression management, it is important to have some measure of the time complexity of each procedure. We therefore start with the time complexity of conventional Gaussian elimination (see [1, 2, 26] for early work). Although some cases of the following theorems are "well known", there seem to be no convenient published statements of them.

Here we consider the time complexity measure is the number of bit operations, which can be rigorously defined as the number of steps of a Turing or register machine or the number of gates of a Boolean circuit implementing the algorithm. [7] Throughout, we make the simplifying assumption that entries grow linearly, in both degree and in coefficient size. This is actually optimistic, as growth is usually worse than this if we apply the classical Gaussian elimination algorithm.

Theorem 3.6.1. *For an n by n matrix $A = [a_{i,j}]$, with $a_{i,j} \in \mathbb{Z}[x_1, \ldots, x_m]$, the time complexity of LU factoring for A is at least $\Omega(n^{2m+5})$ for naive arithmetic.*

Proof. Let $d_r > \max_{i,j} \deg(a_{i,j}, x_r)$. Then $d_1 d_2 \times \ldots \times d_m$ bounds the number of non-zero terms of the polynomials $a_{i,j}$. Let ℓ bound the length of the largest integer coefficient in the $a_{i,j}$. Suppose the degree of the polynomials $a_{i,j}$ in x_r and the size of their integer coefficients are growing linearly with each step of the LU factorization, i.e., at step k, $\deg(a_{i,j}, x_r) < k\ d_r$ and the largest integer coefficient is bounded by ℓk. When we do LU factoring, at the k'th step, we have $(n-k)^2$ entries to manipulate. For each new entry from step $k-1$ to step k, we need to do at least one multiplication, one subtraction and one division. The cost will be at least $\Omega((k\ d_1\ k\ d_2 \ldots k\ d_m\ \ell\ k)^2)$

for naive arithmetic.

The total cost for the LU factoring will be at least $\sum_{k=1}^{n-1}(n-k)^2 \times \Omega((kd_1 \times kd_2 \times ... \times kd_m \times k\ell)^2) = \Omega(d_1^2 d_2^2 ... d_m^2 \ell^2 n^{2m+5})$ (for naive arithmetic). □

With respect to the time complexity for LU factoring with veiling and signatures, we separate the time complexity analysis for LU factoring into two parts. Lemma 3.6.2 shows the time complexity for LU with veiling but without signature. Lemma 3.6.4 gives the time complexity for LU with signatures. The total cost will be the complexity for LU with veiling and signatures in Theorem 3.6.5.

This first lemma is valid for the following veiling strategy: we veil any expression with an integer coefficient of length larger than c_1, or whose degree in x_i is larger than c_2, where c_1, c_2 are positive constants. The cost for veiling an expression is $O(1)$. Then the length of each coefficient will be less than $c = c_1 c_2^m$ and the degree in x_i will be less than c_2.

Lemma 3.6.2. *For an n by n matrix $A = [a_{i,j}]$ the time complexity of LU factoring with large expression management (and the above veiling strategy) is $O(n^3)$.*

Proof. Let $a_{i,j} \in \mathbb{Z}[x_1, ..., x_m]$, $d_r > \max_{i,j} \deg(a_{i,j}, x_r)$, and the length of the integer coefficients of $a_{i,j}$ be at most ℓ. At each step there are at most two multiplications, one division and one subtraction. The cost of each step will be less than $4 \times O((c_1.c_2^m)^2) + O(1)$ for naive arithmetic. At each step k, one performs arithmetic on $(n-k)^2$ matrix elements, for a total cost of $\sum_{k=0}^{n-1}(n-k)^2 \times O((c_1\ c_2^m)^2) = O(n^3)$. □

Remark 3.6.3. To prevent the cost from growing exponentially with the number of variables, the above computation clearly shows that it is best to choose $c_2 = 1$.

Lemma 3.6.4. *Let $A = [a_{i,j}]$ be an n by n matrix where $a_{i,j} \in \mathbb{Z}[x_1, ..., x_m]$. Let $d > \max_{i,j,r} \deg(a_{i,j}, x_r)$ and let ℓ bound the length of the largest integer coefficient of*

the entries of the matrix. Let $T > \ell d^m$. So T bounds the size of matrix entries. Let p be the prime being used to compute signatures.

Then the time complexity for computing all signatures modulo p in the LU factorization is $O((Tn^2 + n^3)(\log p)^2)$.

Proof. The cost of dividing an integer coefficient of $a_{i,j}$ of length ℓ by p is $O(\ell)$ arithmetic operations modulo p and there are less than d^m terms in $a_{i,j}$. Assuming nested horner form is used, the polynomial $a_{i,j}$ can be evaluated modulo p in less than d^m multiplications and d^m additions modulo p. Thus cT bounds the number of arithmetic operations modulo p needed to compute the signature of each input matrix entry for some positive integer c.

After the initial computation of signatures for the entries of A, we need at most four operations in $\mathbb{Z}/p\mathbb{Z}$ for computing the other entries' signatures at step k of the factorization. This costs $O((\log p)^2)$ for naive integer arithmetic. We compute all the signatures for the entries at each step, to greatly simplify zero-testing. So the total cost for computing signatures for the LU factoring is bounded by

$$\left[cTn^2 + \sum_{k=1}^{n-1} 4(n-k)^2\right] \times O((\log p)^2) = O((Tn^2 + n^3)(\log p)^2).$$

□

Theorem 3.6.5. *Let $A = [a_{i,j}]$ be an n by n matrix where $a_{i,j} \in \mathbb{Z}[x_1, ..., x_m]$ and let p be the prime used for signature arithmetic. Let $d > \max_{i,j,r} \deg(a_{i,j}, x_r)$ and ℓ bound the length of the largest integer coefficient of the entries of the matrix. Let $T > \ell d^m$. So T bounds the size of matrix entries. The time complexity of LU factoring with the above veiling strategy and modulo p signature computation is $O((Tn^2 + n^3)(\log p)^2)$.*

Proof. Immediate from the above two lemmas. □

From Theorem 3.6.1 and Theorem 3.6.5, we can see the more variables and the bigger the size of the matrix, the bigger the difference between the algorithms which are with and without veiling and signatures. These results agree completely with our empirical results.

3.7 Empirical Results

We present some timing results. For the benchmarks described below, we use strategies based on MAPLE's `length` command. As these strategies are heuristics, any reasonable measure of the complexity of an entry is sufficient. The pivoting strategy searches for the element with the largest `length`. The `veiling` strategy depends on the type of matrix. For integer matrices, we veil all integers whose length is greater than 1000, while for polynomial matrices, the treshold is length 30. These constants reflect the underlying constants involved in the arithmetic for such objects.

For all benchmarks, three variations are compared: our own LU factoring algorithm with veiling and signatures, MAPLE's default `LinearAlgebra:-LUDecomposition`, and a version of `LinearAlgebra:-LUDecomposition` where `Normalizer` has been set to be the identity function and `Testzero` has been set to a version of `testeq`. We first had to "patch" MAPLE's implementation of `LUDecomposition` to use `Testzero` instead of an explicit call to `Normalizer(foo) <> 0`, and then had to further "patch" `testeq` to avoid a silly coding mistake that made the code extremely inefficient for large expressions[3]. All tests were first run with a time limit of 300 seconds. Then the first test that timed out at 300 seconds was re-run with a time limit of 1000 seconds, to see if that was sufficient for completion. Further tests

[3]Both of these deficiencies were reported to MAPLESOFT and will hopefully be fixed in later versions of MAPLE

in that column were attempted. Furthermore, the sizes of matrices used varies according to the results, to try and focus attention to the sizes where we could gather some meaningful results in (parts of) the three columns. All results are obtained using the TTY version of MAPLE10, running on an 1.8Ghz Intel P4 with 512Megs of memory running Windows XP SP2, and with garbage collection "frequency" set to 20 million bytes used. All results are for dense matrices. In each table, we report the times in seconds, and for the LEM column, the number in parentheses indicates how many[4] distinct labels (ie total number of veiled expressions) were needed by the computation, as an indication of memory requirements.

The reason for including the MapleFix column is to really separate out the effect of arithmetic and signature-based zero-testing from the effects of Large Expression Management; MapleFix measures the effect of not doing polynomial arithmetic and using signatures for zero-recognition, and is thus expected to be a middle ground between the other two extremes.

Table 3.7.1 shows the result for random matrices over the integers. Only for fairly large matrices (between 90x90 and 100x100) does the cost of arithmetic, due to coefficient growth, become so large that the overhead of veiling becomes worthwhile, as the LEM column shows. Since integer arithmetic is automatic in MAPLE, it is not surprising that the MapleFix column shows times that are the same as the Maple column. Here the veiling strategy really matters: for integers of length 500, veiling introduces so much overhead that for 110x110 matrices, this overhead is still larger than pure arithmetic. For length 2000, no veiling at all occurs.

Table 3.7.2 shows the result for random univariate matrices, where the initial polynomials have degree 5 and small integer coefficients. The effect of LEM here is

[4]and we use a postfix K or M to mean 10^3 and 10^6 as appropriate

Size	LEM	MapleFix	Maple
10	.03 (0)	.07	.04
20	.2 (0)	.2	.2
30	.8 (0)	.7	.7
40	2.3 (0)	2.2	2.2
50	6.1 (148)	5.2	5.2
60	12.5 (902)	10.7	10.5
70	17.8 (2788)	19.4	19.2
80	27.6 (5948)	33.8	32.6
90	42.4 (12779)	54.0	52.8
100	56.4 (22396)	83.8	85.8
110	75.4 (36739)	124.7	123

Table 3.7.1: Timings for LU factoring of random integer matrices generated by `RandomMatrix(n,n, generator=`$-10^{12}..10^{12}$`)`. The entries are explained in the text.

Size	LEM	MapleFix	Maple
5	.12 (26)	.06	.53
10	.06 (237)	.07	1.5
15	.18 (872)	.16	9.3
20	.44 (2182)	.30	39.2
25	.87 (4417)	.56	110.4
30	1.9 (7827)	1.87	269.8
35	3.0 (12K)	332	431
40	4.5 (19K)	>1000	845
45	7.8 (28K)	–	>1000
50	9.1 (39K)	–	–

Table 3.7.2: Timings for LU factoring of random matrices with univariate entries of degree 5, generated by `RandomMatrix(n,n, generator = (() -> randpoly(x)))`. The entries are explained in the text.

Size	LEM	MapleFix	Maple
5	.05 (26)	.06	35.3
10	.09 (237)	.09	> 1000
15	.23 (872)	.20	–
20	.49 (2182)	.39	–
25	.99 (4417)	.75	–
30	1.7 (7827)	3.2	–
35	2.8 (12K)	949	–
40	4.2 (19K)	>1000	–
45	6.0 (28K)	–	–
50	8.8 (39K)	–	–

Table 3.7.3: Timings for LU factoring of random matrices with trivariate entries, low degree, 8 terms, generated by `RandomMatrix(n,n, generator = (() -> randpoly([x, y, z], terms = 8)))`. The entries are explained in the text.

Size	LEM	MapleFix	Maple
5	.047 (22)	.03	1.56
10	.078 (218)	.08	>1000
15	.20 (858)	.14	–
20	.51 (2163)	.30	–
25	.88 (4393)	.58	–
30	1.7 (7798)	3.8	–
35	2.95 (12K)	>1000	–

Table 3.7.4: Timings for LU factoring of fully symbolic matrix: `Matrix(n,n,symbol = m)`. The entries are explained in the text.

Size	LEM	MapleFix	Maple
5	.031 (26)	xx	0.99
10	.094 (237)	xx	117
15	.22 (872)	xx	> 1000
20	.50 (2182)	xx	–
25	.99 (4417)	xx	–
30	1.7 (7827)	xx	–
35	2.8 (12K)	xx	–

Table 3.7.5: Timings for LU factoring of random matrix with entries over $\mathbb{Z}[x, 3^x]$: `RandomMatrix(n,n, generator = (() -> eval( randpoly([x,y], terms=8), y = 3^x )))`. The entries are explained in the text.

immediately apparent. What is not shown is that MapleFix uses very little memory (both allocated and "used"), while the Maple column involves a huge amount of memory "used", at all sizes, so that computation time was swamped by garbage collection time. Another item to notice is that while the times in the Maple column grow steadily, the ones in the MapleFix column are at first consistent with the LEM column, and then experience a massive explosion. Very careful profiling[5] was necessary to unearth the reason for this, and it seems to be somewhat subtle: for both LEM and MapleFix, very small DAGs are created, but for LEM we have full control of these, while for MapleFix, the DAGs are small but the underlying expression tree is enormous. All of Maple's operations on matrix elements first involve the element being *normalized* by the kernel (via the user-inaccessible `simpl` function), and then *evaluated.* While normalization follows the DAG, evaluation in a side-effecting language, such as MAPLE, must folllow the expression tree, and thus is extremely expensive. Along with the fact that no information is kept between calls to `testeq`, this causes the time to explode for MapleFix for 35x35 (and larger) matrices. Since the veiling strategy used for the last 4 tables is the same, it is not very suprising that the number of veilings is essentially the same. The reason that the all-symbolic is a little lower is because we start with entries of degree 1 and coefficient size 1, and thus these entries do not get veiled immediately. However, one can observe a clear cubic growth in the number of veilings, as expected.

Table 3.7.3 shows the result for random trivariate matrices, where the initial polynomials have 8 terms and small integer coefficients. The results here clearly show the effect that multi-variate polynomial arithmetic has on the results. Table 3.7.4

[5]Here we used a combination of procedure-level profiling via `CodeTools[Profiling]` and global profiling via `kernelopts(profile=true)`

shows the results for a matrix with all entries symbolic, further accentuating the results in the trivariate case. Again, MapleFix takes moderate amounts of memory (but a lot of CPU time at larger sizes), while Maple takes huge amounts, causing a lot of swapping and trashing already for 10x10 matrices.

Table 3.7.5 shows results for matrices with entries over $\mathbb{Z}[x, 3^x]$. Overall the behaviour is quite similar to bivariate polynomials, however the **xx** in the MapleFix entry indicate a weakness in MAPLE's `testeq` routine, where valid inputs (according to the theory in [9]) return `FAIL` instead. Our signature implementation can handle such an input domain without difficulty.

While we would have liked to present memory results as well, this was much more problematic, as MAPLE does not really provide adequate facilities to achieve this. One could look at **bytes used**, but this merely reflects the memory asked of the system, the vast majority of which is garbage and immediately reclaimed. This does measure the amount of overall memory *churn*, but does not give an indication of final memory use nor of the true *live set*. **bytes alloc** on the other hand measures the actual amount of system memory allocated. Unfortunately, this number very quickly settles to something a little larger than **gcfreq**, in other words the amount of memory required to trigger another round of garbage collection, for all the tests reported here. This reflects the huge amount of memory used in these computations, but does not reflect the final amount of memory necessary to store the end result. Neither can we rely on MAPLE's **length** command to give an accurate representation of the memory needed for a result because, for some unfathomable reason, **length** returns the expression tree length rather than the DAG length! Thus, for matrices whose results are un-normalized polynomials, we have no easy way to measure their actual size. As a proxy, we can find out the total number of variables introduced by

the veiling process.

Bibliography

[1] Bareiss, E. H. Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination. *Mathematics of Computation*, 22(103), 565-578, 1968.

[2] Bareiss, E. H. Computational Solutions of Matrix Problems Over an Integral Domain. *J. Inst. Maths Applics*, 10, 68-104, 1972.

[3] Carette, J. Understanding Expression Simplification. *ISSAC'04*, 72-79, 2004.

[4] Carette, J. and Kiselyov O. Multi-stage Programming with Functors and Monads: Eliminating Abstraction Overhead from Generic Code. *Generative Programming and Component Engineering*, 256-274, 2005.

[5] Corless, R. M., Jeffrey, D. J., Monagan, M. B. and Pratibha. Two Perturbation Calculation in Fluid Mechanics Using Large-Expression Management. *J. Symbolic Computation*, 11, 1-17, 1996.

[6] Freeman, T. S., Imirzian, G., Kaltofen, E. and Lakshman, Y. N. DAGWOOD: A System for Manipulating Polynomials Given by Straight-line Programs. *ACM Trans. Math. Software*, 14(3), 218-240, 1988.

[7] von zur Gathen, J. and Gerhard, J. Modern Computer Algebra. *Cambridge University Press*, 1999.

[8] Giusti, M., Hägele, K., Lecerf, G., Marchand, J. and Salvy, B. The Projective Noether Maple Package: Computing the Dimension of a Projective Variety. *J. Symbolic Computation*, 30(3), 291-307, 2000.

[9] Gonnet, G. H. Determining Equivalence of Expressions in Random Polynomial Time, Extended Abstract. *Proc. of ACM on Theory of Computing*, 334-341, 1984.

[10] Gonnet, G. H. New results for Random Determination of Equivalence of Expressions. *Proc. of ACM on Symbolic and Algebraic Comp.*, 127-131, 1986.

[11] Heintz, J. and Schnorr, C. P. Testing Polynomials Which are Easy to Compute (Extended Abstract). *Proceedings of ACM on Theory of Computing*, 262-272, 1980.

[12] `http://www.linbox.org`

[13] `http://www.maplesoft.com/products/\-thirdparty/dynaflexpro/`

[14] Ibarra, O. H., Moran, S. and Rosier, L. E. Probabilistic Algorithms and Straight-Line Programs for Some Rank Decision Problems. *Infor. Proc. Lett.*, 12(5), 227-232, 1981.

[15] Ibarra, O. H. and Leininger, B. S. On the Simplification and Equivalence Problems for Straight-Line Programs. *J. ACM*, 30(3), 641-656, 1983.

[16] Ibarra, O. H. and Moran, S. Probabilistic Algorithms for Deciding Equivalence of Straight-Line Programs. *J. ACM*, 30(1), 217-228, 1983.

[17] Kaltofen, E. Computing with Polynomials Given by Straight-line Programs I: Greatest Common Divisors. *Proceedings of ACM on Theory of Computing*, 131-142, 1985.

[18] Kaltofen, E. Greatest Common Divisors of Polynomials Given by Straight-Line Programs. *J. of the Association for Computing Machinery*, 35(1), 231-264, 1988.

[19] Kaltofen, E. On Computing Determinants of Matrices without Divisions. *ISSAC*, 342-349, 1992.

[20] Martin, W. A. Determining the Equivalence of Algebraic Expressions by Hash Coding. *Proceedings of ACM on Symbolic and Algebraic Manipulation*, 305-310, 1971.

[21] Monagan, M. B. Signatures + Abstract Types = Computer Algebra − Intermediate Expression Swell. PhD Thesis, *University of Waterloo*, 1990.

[22] Monagan, M. B. Gauss: a Parameterized Domain of Computation System with Support for Signature Functions. *DISCO*, Springer-Verlag LNCS, 722, 81-94, 1993.

[23] Monagan, M. B. and Gonnet, G. H. Signature Functions for Algebraic Numbers. *ISSAC'94*, 291-296, 1994.

[24] Monagan, M. B. and Monagan, G. A Toolbox for Program Manipulation and Efficient Code Generation with an Application to a Problem in Computer Vision. *ISSAC'97*, 257-264, 1997.

[25] Nicholson, W. K. Linear Algebra with Applications, Fourth Edition. *McGraw-Hill Ryerson*, 2003.

[26] Sasaki, T. and Murao, H. Efficient Gaussian Elimination Method for Symbolic Determinants and Linear Systems (Extended Abstract). *ISSAC'81*, 155-159, 1981.

[27] Schwartz, J. T. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, 27(4), 701-717, 1980.

[28] Shoup, V. A Computational Introduction to Number Theory and Algebra. *Cambridge University Press*, 2005.

[29] Steinberg, S. and Roach P. Symbolic Manipulation and Computational Fluid Dynamics. *Journal of Computational Physics*, 57, 251-284, 1985.

[30] Zippel, R. Probabilistic Algorithms for Sparse Polynomials. *Proc. of Eurosam*, Springer-Verlag LNCS 72, 216-226, 1979.

[31] Zippel, R. Effective Polynomial Computation. *Kluwer*, 1993.

Chapter 4

Implicit Reduced Involutive Forms and Their Application to Engineering Multibody Systems

The RIFSIMP package in Maple transforms a set of differential equations to Reduced Involutive Form. This paper describes the application of RIFSIMP to challenging real-world problems found in engineering design and modelling. RIFSIMP was applied to sets of equations arising in the dynamical studies of multibody systems. The equations were generated by the Maple package DYNAFLEX, which takes as input a graph-like description of a multibody mechanical system and generates a set of differential equations with algebraic constraints. Application of the standard RIFSIMP procedure to such Differential Algebraic Equations can require large amounts of computer memory and time, and can fail to finish its computations on larger problems.

We discuss the origin of these difficulties and propose an Implicit Reduced Involutive Form to assist in alleviating such problems. This form is related to RIFSIMP form by the symbolic inversion of a matrix. For many applications such as numerically integrating the multibody dynamical equations, the extra cost of symbolically inverting the matrix to obtain explicit RIFSIMP form can be impractical while Im-

plicit Reduced Involutive Form is sufficient.

An approach to alleviating expression swell involving a hybrid analytic polynomial computation is discussed. This can avoid the excessive expression swell due to the usual method of transforming the entire input analytic differential system to polynomial form, by only applying this method in intermediate computations when it is required.

4.1 Introduction

A principal goal of multibody dynamics is the automatic generation of the equations of motion for a complex mechanical system, given a description of the system as input [9]. After generation, the set of equations must be analyzed or solved. Commercial programs exist that can generate and integrate such systems of equations. ADAMS, DADS and WORKING MODEL are examples of such products, and they are in widespread use in the automotive, aerospace and robotics industries [12]. These programs have many strengths, in particular they can handle systems containing many bodies (up to 100), but they have drawbacks.

For multibody systems, the general form of the dynamic equations is

$$M(t,q,\dot{q})\ddot{q} + \Phi_q^T \lambda = F(t,q,\dot{q}) \,, \tag{4.1.1}$$

$$\Phi(t,q) = 0 \,. \tag{4.1.2}$$

Here, q is a vector of generalized co-ordinates, $M(t,q,\dot{q})$ is the mass matrix, Φ is a vector of the constraint equations, λ is a vector of Lagrange multipliers and F is a

vector of force [10, 11]:

$$\Phi = \begin{bmatrix} \Phi^1 \\ \vdots \\ \Phi^m \end{bmatrix}, \qquad \Phi_q = \begin{bmatrix} \Phi^1_{q_1} & \cdots & \Phi^1_{q_n} \\ \vdots & \vdots & \vdots \\ \Phi^m_{q_1} & \cdots & \Phi^m_{q_n} \end{bmatrix}, \qquad \lambda = \begin{bmatrix} \lambda^1 \\ \vdots \\ \lambda^m \end{bmatrix}.$$

Because the programs are purely numerical, it is difficult to check or comprehend the basic equations they have generated, and it is difficult to obtain analytic insight into the equations' properties. Also, when used for simulation, the programs are inefficient because the equations are effectively re-assembled at each time step, and the numerical assembly may include many multiplications in which one of the terms is 0 or 1. As a consequence, these programs are not well suited to real-time simulations and virtual reality, and, because of their large size, they cannot be downloaded onto the microprocessors that are typically used in real-time controllers [12, 6].

In contrast to numerically based programs, packages such as DYNAFLEX use symbolic programming to generate the equations of motion in a completely analytical form [13]. This approach offers several advantages [7]. The structure of the equations is easily obtained and manipulated, allowing one to gain a physical insight into the system; the equations can be easily exchanged with other researchers or engineers, something crucial to communication between different design groups; and real-time simulations are facilitated.

However, symbolic packages also have drawbacks. The equations generated by DYNAFLEX are usually too complicated to be solved symbolically. Even numerical solution is often difficult, inefficient or even impossible, because the equations are Differential Algebraic Equations (DAE), which typically are of second order but of high differential index. Also, the number of bodies that these programs can handle is not as large as for the numerically based programs.

Therefore, it is natural to develop methods for symbolically pre-processing the output of programs such as DYNAFLEX so that the output has desirable features such as being in simplified canonical form and including all constraints. Since any consistent initial value must satisfy all constraints, the inclusion of all constraints is a necessary condition for stating an existence and uniqueness theorem for such systems. The statement of such a theorem is another desirable feature for the output of such methods. Such features enable the consistent initialization of numerical solution procedures and can facilitate the identification of analytical solutions. In this paper we discuss how the RIFSIMP package can be used for the symbolic simplification of ODE and PDE systems and return canonical differential forms. It has the following features [16, 17]:

- Computation with polynomial nonlinearities.
- Advanced case splitting capabilities for the discovery of particular solution branches with desired properties.
- A visualization tool for the examination of the binary tree that results from multiple cases.
- Automatic generation of an existence and uniqueness theorem for the system.
- Algorithms for working with formal power series solutions of the system.

Applying RIFSIMP to the equations output by DYNAFLEX has the benefit of symbolically and automatically generating all special cases, through the RIFSIMP case-split options [16, 17]. In a full case analysis, some cases can be very complicated while others can be simple enough to be solved analytically. The canonical form generated by RIFSIMP is of low (0 or 1) differential index [1, 5] which is suitable

for the application of numerical solvers. An important option with RIFSIMP is the possibility of excluding special cases that are known to be not of interest. Thus if we know that a special case, say $m = 0$, is of no physical interest, then we can append the *inequation* $m \neq 0$ to the input system [16, 17].

Application of the RIFSIMP package to multibody systems revealed that it has difficulty handling large systems generated by DYNAFLEX. The symptoms are excessive time and memory requirements. It is a well-known effect in computer algebra that these are linked, in that a computation that overflows physical memory will cause the operating system to swap memory to disk. The swapping, however, essentially brings the system to a halt. There are a number of contributing factors to the growth in time and memory, as will be described below. Therefore, if one wants to handle industrial-scale problems, one must modify the RIFSIMP approach. The modification we introduce here is the possibility of relaxing the requirements on the canonical form.

We remark that these are common difficulties encountered during the application of computer algebra methods to obtain canonical forms (e.g. such as Gröbner Bases for systems of multi-variate polynomials). An underlying idea in this paper is that many application may not require the full potency of canonical simplification (canonicity can be very expensive); and it is important to explore weaker non-canonical forms when they may achieve the objective in the given application.

4.2 Two Examples of Mechanical Systems

Two examples will be used to illustrate the application of computer algebra methods to multi-body systems.

The three dimensional top example considered in the paper, is an example of a small open-loop system. Other examples of open loop systems include robot arms or similar devices with a free end and a fixed end. The slider-crank mechanism considered in the paper, is an example of a small closed-loop system. Another typical example of a closed loop is a piston turning a crank through connecting rods, so that there are constraints on both the crank and the piston. Simple textbook problems in dynamics use *ad hoc* choices of co-ordinate systems to produce simple systems of equations without constraints modelling the problems. But this method is usually not possible with complicated systems, which are automatically generated by packages such as DYNAFLEX and constraint equations can not be eliminated. In addition, the constraints introduce additional variables (essentially Lagrange multipliers) into the equations, representing the forces they exert.

4.2.1 Open Loop: Three-dimensional Top

In this classic problem, the top is an axisymmetric body that can precess about a vertical (Z) axis, nutate about a rotated X axis, and spin about a body-fixed symmetry axis, see figure 4.2.1. The top is assumed to rotate without slipping on the ground; this is modelled by placing a spherical (ball-and-socket) joint at O. DYNAFLEX automatically generates co-ordinates using standard 3-1-3 Euler angles (ζ, η, ξ), which correspond to precession, nutation, and spin, respectively.

The top has a moment of inertia J about the symmetry axis, and A about an axis at O perpendicular to the symmetry axis. Two angles η (the angle of the axis of symmetry to the z axis), and ζ specify the orientation of the axis of symmetry of the top, while ξ specifies how a point on the top is moving relative to its axis of symmetry. There is a coordinate singularity when η is equal to 0 or π, which RIF-

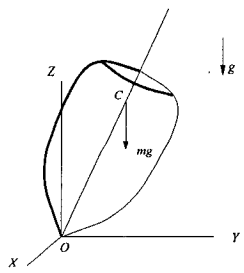

Figure 4.2.1: The three-dimensional top. The centre of mass is at C and OC= ℓ. Gravity acts in the $-Z$ direction.

SIMP will automatically detect as part of its case analysis. Coordinate singularities are ubiquitous in automatically generated mechanical systems, and their automatic detection is an important problem.

The dynamic equations (4.1.1,4.1.2) generated by DYNAFLEX are, after changing from DYNAFLEX-generated symbols to more conventional ones, as follows. For details, we refer to the DYNAFLEX Users Guide [13].

$$M = \begin{bmatrix} (A\sin^2\eta + J\cos^2\eta) & 0 & J\cos\eta \\ 0 & A & 0 \\ J\cos\eta & 0 & J \end{bmatrix}, \quad q = \begin{bmatrix} \zeta \\ \eta \\ \xi \end{bmatrix} \tag{4.2.1}$$

and

$$F = \begin{bmatrix} \sin(\eta)\left[2(J-A)\cos(\eta)\dot{\zeta} + J\dot{\xi}\right]\dot{\eta} \\ \sin(\eta)\left[(A-J)\cos(\eta)\dot{\zeta}^2 - J\dot{\xi}\dot{\zeta} + mgl\right] \\ J\sin(\eta)\dot{\eta}\,\dot{\zeta} \end{bmatrix} \tag{4.2.2}$$

The fact that the top is an open-loop system is reflected in the fact that DYNAFLEX has generated 3 equations for 3 unknowns.

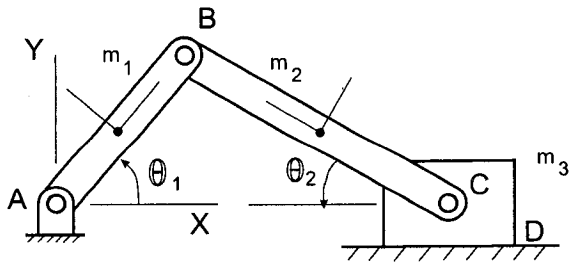

Figure 4.2.2: The two-dimensional slider crank. The arm of length ℓ_1 and mass m_1 rotates and causes the mass m_3 attached to the end of the arm of length ℓ_2 to move left and right. Each arm has mass m_i and moment of inertia J_i, for $i = 1, 2$.

4.2.2 Two-dimensional Slider Crank

The slider crank is a simple example of a closed-loop system with $q^T = (\theta_1, \theta_2)^T$ where $\theta_1 = \theta_1(t)$ and $\theta_2 = \theta_2(t)$ are the angles shown in figure 4.2.2. The system is given by (4.1.1)–(4.1.2) where:

$$M = \begin{bmatrix} \ell_1^2(\frac{m_1}{4} + m_2 + m_3) + J_1 & -\ell_1\ell_2\cos(\theta_1 + \theta_2)(\frac{m_2}{2} + m_3) \\ -\ell_1\ell_2\cos(\theta_1 + \theta_2)(\frac{m_2}{2} + m_3) & \ell_2^2(\frac{m_2}{4} + m_3) + J_2 \end{bmatrix} \tag{4.2.3}$$

$$F = \begin{bmatrix} -\ell_1 g(\frac{m_1}{2} + m_2 + m_3)\cos\theta_1 - \ell_1\ell_2\dot{\theta_2}^2\sin(\theta_1 + \theta_2)(\frac{m_2}{2} + m_3) \\ \ell_2 g(\frac{m_2}{2} + m_3)\cos\theta_2 - \ell_1\ell_2\dot{\theta_1}^2\sin(\theta_1 + \theta_2)(\frac{m_2}{2} + m_3) \end{bmatrix} \tag{4.2.4}$$

and there is a single constraint equation between the angles:

$$\Phi = \ell_1\sin\theta_1 - \ell_2\sin\theta_2 = 0 \tag{4.2.5}$$

Therefore, $\Phi_q^T\lambda$ in (4.1.1) is given by $\Phi_q^T\lambda = \begin{pmatrix} \ell_1\cos\theta_1 \\ -\ell_2\cos\theta_2 \end{pmatrix}\lambda$. Note that in this example, λ is a scalar. In other words, in addition to generating the constraint equation (4.2.5), DYNAFLEX automatically generated the constraint force $\lambda(t)$. For both of these examples, the challenge now is to analyze the equations with computer algebraic methods such as RIFSIMP.

4.3 Simplification Using RifSimp with Case Split

Given that symbolic algorithms such as DYNAFLEX exist for automatically producing the equations modelling multi-body systems, it is natural to exploit the further simplification and transformation of such systems using symbolic methods. In this section we discuss the simplification of such systems using the Maple algorithm `casesplit` which is part of RIFSIMP.

The theory underlying the Reduced Involutive Form given in [8] applies to systems of analytic nonlinear PDE in dependent variables $u_1, u_2, \ldots u_n$, which can be functions of several independent variables. While the general method applies to analytic systems, like most methods in computer algebra, the implemented algorithms apply to systems which are polynomial functions of their unknowns. This is to avoid well-known undecidability issues for classes of functions wider than polynomial or rational functions (e.g. there is no finite algorithm that can decide whether an analytic function is zero or non-zero at a point).

For the present application the only independent variable is time. In general the systems produced by DYNAFLEX are polynomial functions of sines and cosines of the angles $\theta_1(t), \theta_2(t), ..., \theta_n(t)$ between different components (arms and rotors etc).

One common method to convert such systems to rational form is the transformation

$$\cos\theta_j = x_j(t), \;\; \sin\theta_j = y_j(t), \;\; x_j^2 + y_j^2 = 1 \tag{4.3.1}$$

and another is the well-known Weierstrass transformation [4]:

$$\cos\theta_j = (1 - u_j(t)^2)/(1 + u_j(t)^2), \;\; \sin\theta_j = 2u_j(t)/(1 + u_j(t)^2) \tag{4.3.2}$$

The transformation (4.3.2) has the advantage that the number of variables remains the same, and no new constraints are introduced. The transformation (4.3.1) has

the disadvantage that additional constraints are introduced.

We will later discuss an alternative hybrid analytic-polynomial strategy. In that approach, the equations are manipulated in their original analytic form and conversions to polynomials by transformations such as those above are only used at intermediate computations and only for parts of the system which require the full algorithmic power of rational polynomial algebra. After resolving an analytic obstacle in this manner the inverse transformation yields the analytic form and the computation continues until the next algorithmic analytic obstacle is encountered.

RIFSIMP takes as its input a system of differential equations and a ranking of dependent variables and derivatives. Using its default ranking, RIFSIMP orders the dependent variables lexicographically and the derivatives primarily by total derivative order [16, 17]. For example for systems of ODE this default ranking is:

$$u_1 \prec u_2 \prec ... \prec u_n \prec u_1' \prec u_2' \prec ... \prec u_n' \prec u_1'' \prec ... \tag{4.3.3}$$

Each equation is then classified as being either leading linear or nonlinear, meaning linear or nonlinear in its highest derivative with respect to the ranking $\prec$. RIFSIMP solves the leading linear equations for their highest derivatives until it can no longer find any such equations.

While solving explicitly for the highest derivatives, RIFSIMP splits cases based on the pivots (the coefficients of the leading derivatives) with which it divides. This yields a binary tree of cases.

Each leading-nonlinear equation (a so-called constraint) is differentiated and then reduced with respect to the current set of leading linear equations and then with respect to the leading nonlinear equations. A nonzero result means that this equation is a new constraint which should be appended to the system [16, 17].

For the current application if the solutions are 1 dimensional curves then each

case output by the RIFSIMP algorithm has form:

$$v = f(t, w) \ , \tag{4.3.4}$$

$$g(t, w) = 0 \ , \tag{4.3.5}$$

$$h(t, w) \neq 0 \ . \tag{4.3.6}$$

Here v is the list of (highest-order) derivatives; w is a list of all derivatives, including dependent variables, lower in the ranking than v; g is a list of constraint equations; h is a list of inequations. From this form, it is possible to prove an existence and uniqueness theorem.

In particular, in our application, where there is a single independent variable t, the initial condition is $w(t^0) = w^0$ where the initial condition must satisfy the constraint $g(t^0, w^0) = 0$ and any inequations $h(t^0, w^0) \neq 0$ and this leads to a local analytic solution to the original system with this initial condition. Then the Existence and Uniqueness Theorem [8] states that there exists a local analytic solution satisfying the above initial conditions and inequations.

4.3.1 Application to Spinning Top

In order to apply RIFSIMP to equations (4.1.1,4.1.2) with M and F given by (4.2.1) – (4.2.2), we first convert the trigonometric functions to polynomials using the Weierstrass transformation, which is $\cos\eta = (1-u(t)^2)/(1+u(t)^2)$, $\sin\eta = 2u(t)/(1+u(t)^2)$. This yields a rational polynomial differential system. The resulting case tree produced by `casesplit` is surprisingly large, containing 24 cases, see figure 4.3.3.

It is important to understand the reasons for the many cases discovered by RIFSIMP, because for more complicated systems, the case analysis can become overwhelming. We first note that rigid-body mechanics is inherently complicated, and

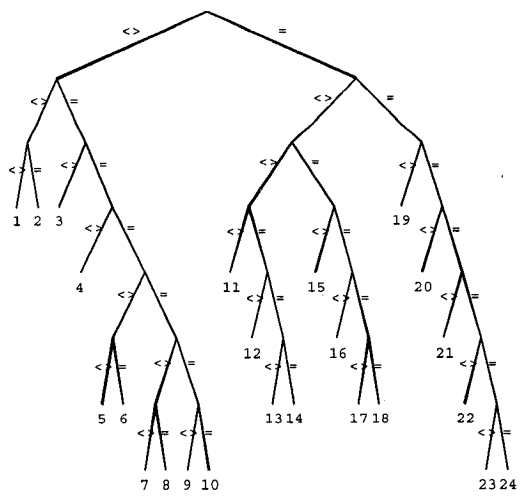

Figure 4.3.3: Complete Case Tree for the 3D Top.

flexible body mechanics more so. The motion is mostly rotational, meaning that linear and angular momentum must be considered (introducing mass and moment-of-inertia parameters), and that the equations contain trigonometric functions. Beyond this, however, we note that one of the advantages of symbolic analysis for the engineer is the use of *symbolic* parameters for the masses, moments of inertia, lengths, etc. This is useful for their design studies, but it is well known in symbolic computing that the introduction of large numbers of symbols causes expression swell, slowdowns in the computation, and the occurrence of many special cases. Finally, for RIFSIMP, there is the problem that the program assumes computation in the complex domain, whereas the engineering application only requires real variables. Thus the special cases identified in the complex plane are not relevant.

RIFSIMP has an input option that allows inequations to be appended to the system. These allow us to record physical facts such as $m \neq 0$, $g \neq 0$, etc. By

recording as much information as possible in the list of inequations, the case tree can be significantly reduced. We find, for example, that the 24 cases in figure 4.3.3 can be reduced to 9. Amongst these special cases, RIFSIMP can identify dynamically degenerate cases. Examples are the cases of the top being oriented exactly vertically or exactly horizontally. In each case, one part of the precessional motion is not present. However, this strategy only delays the arrival of overwhelming expression swell, and we have therefore looked for an alternative to the standard RIFSIMP process.

4.4 Implicit Reduced Involutive Form Method

Two origins of expression swell for RIFSIMP are the following. If there are many equations containing many symbols, then inverting the matrix M to obtain explicit expressions for the $\ddot{q}$ results in division by many pivots that might be zero. After this, the reduction of equations modulo the existing equations is also difficult, because of the size of the equation set and the number of unknowns. From these observations we are led to seek a form weaker than a canonical solved form which is computationally feasible, while retaining as much of the power of RIFSIMP as possible. To this end, we introduce an *implicit* reduced involutive form.

Definition 4.4.1. [Implicit Reduced Involutive Form] Given a differential algebraic equation system (DAEs), let $\prec$ be a ranking of this DAEs. If there exist derivatives $r_1, ..., r_k$ such that L is leading linear in $r_1, ..., r_k$ with respect to $\prec$ (i.e. $L = A[r_1, \cdots, r_k]^T - b = 0$), N is leading nonlinear and

$$[r_1, \cdots, r_k]^T = A^{-1}b\,, \quad N = 0\,, \quad \det(A) \neq 0 \tag{4.4.1}$$

is in reduced involutive form. Then the system $L = 0, N = 0$ is said to be in implicit reduced involutive form (implicit-RIF form).

This form is of interest, since computing A^{-1} on examples can be very expensive. Sometimes implicit rif-form can be obtained very cheaply, just by appropriate differentiation of the constraints.

Example 4.4.2. [Spinning Top] In this case the system has form $M\ddot{q} = F$ and this is in implicit reduced involutive form provided $\det(M) \neq 0$. In general implicit reduced involutive form can be regarded as a cheap way of obtaining some but not all of the cases resulting from a system (e.g. the cases in this example with $\det(M) = 0$ are not covered).

Example 4.4.3. [General Multi-Body Systems] To convert a system of general form (4.1.1,4.1.2) with non-trivial constraints to implicit reduced involutive form one would have to at least differentiate the constraints twice. Carrying this out we obtain

$$M\ddot{q} + \Phi_q^T \lambda = F(t, q, \dot{q}) \tag{4.4.2}$$

$$D_t^2 \Phi = \Phi_q \ddot{q} + H\dot{q} + 2\Phi_{tq}\dot{q} + \Phi_{tt} = 0 \tag{4.4.3}$$

$$D_t \Phi = \Phi_q \dot{q} + \Phi_t = 0 \tag{4.4.4}$$

$$\Phi(t, q) = 0 \tag{4.4.5}$$

where $\Phi_{tq} = \frac{\partial \Phi_q}{\partial t}$, $\Phi_{tt} = \frac{\partial^2 \Phi}{\partial t^2}$ and

$$H = \begin{bmatrix} \sum_i \Phi^1_{q_1 q_i} \dot{q}_i & \cdots & \sum_i \Phi^1_{q_n q_i} \dot{q}_i \\ \vdots & \vdots & \vdots \\ \sum_i \Phi^m_{q_1 q_i} \dot{q}_i & \cdots & \sum_i \Phi^m_{q_n q_i} \dot{q}_i \end{bmatrix}$$

We now show:

Theorem 4.4.4. *Consider the ranking* $\prec$ *defined by* $q \prec \dot{q} \prec \lambda \prec \ddot{q} \prec \dot{\lambda} \prec \dddot{q} \cdots$ *where the dependent variables* q, λ *are ordered lexicographically* $q_1 \prec q_2 \prec \ldots$ *and* $\lambda_1 \prec \lambda_2 \prec \ldots$. *The systems (4.4.2, 4.4.3, 4.4.4, 4.4.5) are in implicit rif-form with* A, b, $[r_1, ..., r_k]^T$ *in Definition 4.4.1 given by:*

$$A = \begin{bmatrix} M & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix}, \quad b = \begin{bmatrix} F(t,q,\dot{q}) \\ -H\dot{q} - 2\Phi_{tq}\dot{q} - \Phi_{tt} \end{bmatrix}, \quad [r_1, ..., r_k]^T = \begin{pmatrix} \ddot{q} \\ \lambda \end{pmatrix} \tag{4.4.6}$$

and $N = \{\Phi = 0,\ \Phi_q\dot{q} + \Phi_t = 0\}$, *under the assumption that* $\det(A) \neq 0$.

Proof. Set $N = \{\Phi = 0, D_t\Phi\} = \{\Phi = 0, \Phi_q\dot{q} + \Phi_t = 0\}$ in Theorem 4.4.4. Differentiating again yields $D_t^2\Phi$ given by (4.4.3).

Rewriting the system (4.4.2,4.4.3) in matrix form yields A, b, $[r_1, ..., r_k]^T$ in (4.4.6).

It remains to verify that $D_t\psi$ when reduced first with respect to L and then with respect to N yields zero for each $\psi \in N$. First $D_t\Phi = \Phi_q\dot{q} + \Phi_t$ is unaltered by reduction with respect to L and then reduces to zero with respect to N (since it is already in N). Next $D_t(\Phi_q\dot{q} + \Phi_t)$ is given in (4.4.3) and reduces to zero on simplification with respect to L (since it is a member of L). Note that we are working with analytic systems. Here as described in Rust [8] reduction to zero means detection as member of the analytic ring of functions (with coefficients again analytic functions) generated by the members of N. In general this is not algorithmic, but for multi-body systems by using one of the transformations to polynomial form, it can be converted into a polynomial ideal membership question which can be answered algorithmically, then transformed back to the analytic form. Here however the detection is trivial and does not require such techniques.

Again, we can note that implicit reduced involutive form easily obtained non-degenerate cases corresponding to $\det(A) \neq 0$. To determine whether a case is

empty or not requires the analysis of whether there are any common solutions satisfying $N = 0$ and $\det(A) \neq 0$. This is a purely algebraic problem, which can be resolved in the worse case by applying one of the transformations to rational polynomial form. In that form one of the standard methods of commutative algebra (e.g. triangular sets) can be applied. Alternatively one can use some of the techniques of the new area of numerical algebraic geometry such as the methods of Sommese, Verschelde and Wampler [14]. That method determines so-called *generic* points on components of the variety determined by $N = 0$. Substitution of these points into A and application of some technique from numerical linear algebra (e.g. the singular value decomposition) can determine if $\det(A) \neq 0$. □

A full analysis would have to consider the more difficult cases with $\det(A) = 0$. Indeed higher index problems (index > 2) yield $\det(A) = 0$ and further differentiations of the constraints need to be carried out to obtain implicit reduced involutive form.

Example 4.4.5. [Slider-Crank] The example of the two-dimensional slider crank described above exactly fits into the class being discussed. Notice that even this simple case generates 5 independent parameters: each arm has a mass and a moment of inertia and the slider has a mass. The computation of this example is much harder to achieve in reduced involutive form.

4.5 Conclusion and Future Work

The mechanical systems generated by programs such as DYNAFLEX mean that they are ideal for testing new algorithms for dealing with DAE. The underlying idea is that such directly physical systems should lead to insights and new techniques for

such DAE.

The implicit forms obtained can be useful in the numerical solution of such systems. For example, the matrix A above yields a system of DAE which can be solved using an implicit numerical method (i.e. along a solution curve, the constant matrix A evaluated at a certain time step is a constant matrix, which is inverted using stable numerical methods). Thus a very expensive symbolic (exact) inversion of a matrix has to be compared to the solution using LU decomposition at each step along the path. In many applications we stress that the repeated solution of these systems along the path, are *much cheaper than symbolically inverting the matrix once and then evaluating the solution along the path.* Finding a balance between the cost of symbolic simplification, verses implicit results is the subject of our ongoing research. This is important for example in being able to carry out real time simulations.

In some respects the method that we eventually are approaching is quite similar to that appearing in the literature (e.g. see Visconti [15]). The purpose of the article is to try to draw rigorous differential elimination approaches closer together with such methods. In addition we suggest the use of analytic systems to assist in efficiency (avoiding a full polynomialization of the problem, since this can increase the complexity of the problem). In our calculations full polynomialization led to many extra equations compared to the analytic approach. The total degree (which is a measure of the complexity of the system) was often dramatically increased by the transformations, and this was reflected in our experience with calculations.

Indeed it is quite surprising that some of the techniques in DAE have not produced analogous strategies in general differential elimination packages for ODE and PDE such as `diffalg` or the RIFSIMP package. Our article is an effort to try to bridge this gap. Indeed an interesting aspect of the article and the work was the in-

teraction between the authors from mechanical engineering (McPhee and Schmitke) and those from computer algebra (Jeffrey, Reid and Zhou). It forced the computer algebraists to examine some of the underlying techniques and assumptions routinely made in computer algebra. For example the conversion of analytic systems to rational polynomial form, is almost automatic and unquestioned as desirable in computer algebra approaches. The restriction to polynomial or rational polynomial functions also arose historically in the largely algebraic earlier era of symbolic computation. But as indicated in this article such a conversion can lead to unnecessarily large expressions.

Our planned work includes other strategies for controlling the generation of large expressions, since there will always be a desire on the part of design engineers to increase the number of bodies that can be modelled. One strategy for large expression management (LEM) has been described in [3]. The key idea is that large expressions are not arbitrary collections of terms, but contain a structure. An analogy can be drawn with the situation in the study of matrices arising in engineering applications: they almost always have a 'structure' to them. For example, they are banded, or otherwise sparse. By recognizing structure, we can solve larger problems. Returning to symbolic manipulation, we can note that simplification routines in computer algebra can cause a loss of structure, usually with the result that larger expressions are generated. A very simple example is the apparent simplification of $(1+x)^9 - 1$, where a computer system will expand the bracket in order to cancel the 1 from the expansion with the 1 outside the bracket. Using the tools developed in [3] and now incorporated into Maple, we can preserve the structure inherent in engineering equations, such as those described here.

Bibliography

[1] Ascher, U. and Petzold, L. Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations. *SIAM*, 1998.

[2] Buchberger, B. and Collins, G. E. Computer Algebra Symbolic and Algebraic Computation. *Springer-Verlag*, 1983.

[3] Corless, R. M., Jeffrey, D. J., Monagan, M. B. and Pratibha. Two Perturbation Calculations in Fluid Mechanics Using Large-Expression Management. *J. Symbolic Computation*, 11, 1-17, 1996.

[4] Cox, D. A., Little, J. B. and O'Shea, D. Ideals, Varieties and Algorithms. *Springer-Verlag*, 1997.

[5] Reid, G. J., Lin, P. and Wittkopf, A. D. Differential-Elimination Completion Algorithms for DAE and PDAE. *Studies in Applied Mathematics*, 106, 1-45, 2001.

[6] Rideau, P. Computer Algegbra and Mechanics, The James Software. Computer Algebra in Industry I. *John Wiley*, 1993.

[7] Rudolf, C. Road Vehicle Modeling Using Symbolic Multibody System Dynamics. Diploma Thesis, *University of Waterloo in cooperation with University of Karlsruhe*, 2003.

[8] Rust, C. J. Rankings of Partial Derivatives for Elimination Algorithms and Formal Solvability of Analytic Partial Differential Equations. PhD Thesis, *University of Chicago*, 1998.

[9] Schiehlen, W. Multibody Systems Handbook. *Springer-Verlag*, 1990.

[10] Shi, P. and McPhee, J. J. Dynamics of Flexible Multibody Systems Using Virtual Work and Linear Graph Theory. *Multibody System Dynamics*, 4(4), 355-381, 2000.

[11] Shi, P., McPhee, J. J. and Heppler, G. A Deformation Field for Euler-Bernouli Beams with Application to Flexible Multibody Dynamics. *Multibody System Dynamics*, 4, 79-104, 2001.

[12] Shi, P. and McPhee, J. J. Symbolic Programming of a Graph-Theoretic Approach to Flexible Multibody Dynamics. *Mechanics of Structures and Machines*, 30(1), 123-154, 2002.

[13] Shi, P. and McPhee, J. J. DynaFlex User's Guide. Systems Design Engineering, *University of Waterloo*, 2002.

[14] Sommese, A. J., Verschelde, J. and Wampler, C. W. Numerical Decomposition of the Solution Sets of Polynomial Systems into Irreducible Components. *SIAM J. Numer. Anal.*, 38(6), 2022-2046, 2001.

[15] Visconti, J. Numerical Solution of Differential Algebraic Equations Global Error Estimation and Symbolic Index Reduction. PhD Thesis, *Laboratoire de Modélisation et Calcul. Grenoble*, 1999.

[16] Wittkopf, A. D. and Reid, G. J. The Reduced Involutive Form Package. *Maple Software Package*, First distributed as part of Maple 7, 2001.

[17] Wittkopf, A. D. Algorithms and Implementations for Differential Elimination. PhD Thesis, *Simon Fraser University, Burnaby*, 2004.

Chapter 5

Symbolic Computation Sequences and Numerical Computation in Multibody Dynamical Systems

The symbolic-numeric computing described here consists of an extensive symbolic pre-processing of systems of differential-algebraic equations (DAE), followed by the numerical integration of the system obtained. The application area is multibody dynamics. We deal symbolically with a DAE system using differentiation and elimination methods to find all the hidden constraints, and produce a system that is leading linear (linear in its leading derivatives). Then we use LU symbolic decomposition with Large Expression Management to solve this leading linear system for its leading derivatives, thereby obtaining an explicit ODE system written in terms of computation sequences obtained from using the MAPLE package `LargeExpressions`. Subsequently the Maple command `dsolve` is applied to this explicit ODE to obtain its numeric solution. Advantages of this strategy in avoiding expression explosion are illustrated and discussed. We briefly discuss a new class of methods involving Numerical Algebraic and Analytic Geometry.

5.1 Introduction

In the study of multibody dynamics, both purely symbolic and purely numeric methods separately suffer drawbacks. The symbolic methods encounter large expressions and the numerical methods are very slow in formulating the system. Examples of the applications of multibody dynamics are robot arms, vehicle suspensions, and automatic barriers. A modern robot arm, such as the Space Shuttle's Remote Manipulator System (RMS, or Canadarm) consists of several rigid bodies (the arm segments and the end effectors) linked by joints. Therefore the systems have at least 7 degrees of freedom, and typically more. In order to simulate the behaviour of systems like these, a number of programs have been developed that automatically generate the equations of motion for the system from its engineering description [14]. Several computer-algebra based packages have been developed, for example DYNAFLEX and SYMOFROS, that will generate the equations of motion in symbolic form. At present, these equations are handed over to purely numerical systems for integration as soon as the symbolic system has generated the equations.

What is described in this paper is a closer coupling of the symbolic analysis of the mechanical system and the final numerical integrations used by a simulation. The coupling takes the form of a symbolic pre-processing of the equations that improves the efficiency of their numerical solution. This increased interaction between the symbolically based part of the computation and the numerical part deserves to be called a "symbolic–numeric" calculation, we feel, even though the calculation is quite different from the numerical polynomial algebra [18] that has so far been the subject covered by the description "symbolic–numeric computation".

The equations describing the dynamics of a multibody system take the general

form

$$M(t,q,\dot{q})\ddot{q} + \Phi_q^T \lambda = F(t,q,\dot{q}) \,, \tag{5.1.1}$$

$$\Phi(t,q) = 0 \,. \tag{5.1.2}$$

Here, q is a vector of generalized co-ordinates, $M(t,q,\dot{q})$ is the mass matrix, Φ is a vector of the constraint equations, λ is a vector of Lagrange multipliers and F is a vector of force [16, 15]:

$$\Phi = \begin{bmatrix} \Phi^1 \\ \vdots \\ \Phi^m \end{bmatrix}, \qquad \Phi_q = \begin{bmatrix} \Phi^1_{q_1} & \cdots & \Phi^1_{q_n} \\ \vdots & \vdots & \vdots \\ \Phi^m_{q_1} & \cdots & \Phi^m_{q_n} \end{bmatrix}, \qquad \lambda = \begin{bmatrix} \lambda^1 \\ \vdots \\ \lambda^m \end{bmatrix}.$$

The Lagrange multipliers λ can be interpreted as the forces that the joints must exert between the elements in order to enforce the geometrical constraints they impose.

These symbolic models are too complicated to be solved symbolically, both because of their nonlinearity and because of the number of equations. However there is still the possibility of symbolically pre-processing them before attempting a numerical solution. It can be noted from the general form of the equation system (5.1.1)-(5.1.2) that it is differential-algebraic (DAE), with the numerical difficulties that that entails. Suitable objectives for the pre-processing include the simplification of the system and the determination of all constraints in order to facilitate the subsequent numerical simulation of the system. More specifically, we show below that we can convert the system to a purely differential one, and moreover separate the constraint forces λ from the simulation variables q. Our primary tool for this is the RIFSIMP package, which uses differentiation and elimination methods to simplify any over-determined polynomially nonlinear PDE or ODE system and to return a canonical differential form [12].

Direct application of the RIFSIMP package to multibody systems reveals that it has difficulty handling the large systems generated by DYNAFLEX. We use Implicit Reduced Involutive Form (IRIF) to assist in alleviating such large expression swell problems [19]. Implicit RIF form is converted to RIF form by symbolically solving the IRIF for its leading linear derivatives. In particular we use symbolic LU decomposition with large expression management to obtain RIF form, expressed in terms of computation sequences.

In this paper, we use the simple example of a two-dimensional slider crank to illustrate our approach to the symbolic-numeric solving of this kind of DAE system. After receiving the DAE model of the slider crank from DYNAFLEX we first use implicit RIFSIMP to compute all the hidden constraints of this higher index DAE, by reducing it to an index one or zero DAE system. The details are given in section 5.2 and section 5.3. Then in section 5.4, we use symbolic LU matrix factoring with large expression management to get the canonical form for the differential equations, which helps the numerical integration. Using these symbolic equations with computation sequences, we illustrate the numerical simulation in the section 5.5. Finally in section 5.6, we discuss some new ideas for solving the constraints to obtain consistent initial values for the integration. It should be emphasized that each of the sub-tasks described here are fully algorithmic and automated computations. Their integration into a single piece of software is in principle straightforward.

5.2 Two-Dimensional Slider Crank

Space limitations prevent us from describing a realistic mechanical system, both because the system description would take up significant space, and also because the

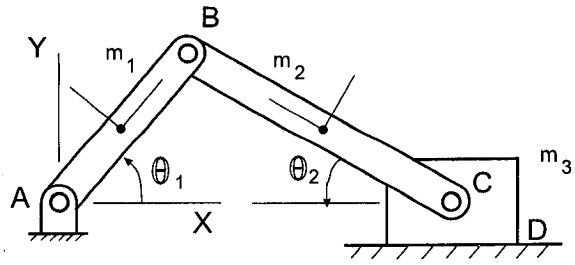

Figure 5.2.1: The two-dimensional slider crank. The arm of length ℓ_1 and mass m_1 rotates while the mass m_3 attached to the end of the arm of length ℓ_2 moves left and right. Each arm has mass m_i and moment of inertia J_i, for $i = 1, 2$.

equations generated by DYNAFLEX would by too lengthy for the reader to follow. The two-dimensional slider crank is a simple example of a closed-loop system with $q^T = (\theta_1, \theta_2)^T$ where $\theta_1 = \theta_1(t)$ and $\theta_2 = \theta_2(t)$ are the angles shown in figure 5.2.1. The system is given by (5.1.1)–(5.1.2) where:

$$M = \begin{bmatrix} \ell_1^2(\frac{1}{4}m_1 + m_2 + m_3) + J_1 & -\ell_1\ell_2\cos(\theta_1+\theta_2)(\frac{1}{2}m_2 + m_3) \\ -\ell_1\ell_2\cos(\theta_1+\theta_2)(\frac{1}{2}m_2 + m_3) & \ell_2^2(\frac{1}{4}m_2 + m_3) + J_2 \end{bmatrix} \tag{5.2.1}$$

$$F = \begin{bmatrix} -\ell_1 g(\frac{1}{2}m_1 + m_2 + m_3)\cos\theta_1 - \ell_1\ell_2\dot{\theta_2}^2\sin(\theta_1+\theta_2)(\frac{1}{2}m_2 + m_3) \\ \ell_2 g(\frac{1}{2}m_2 + m_3)\cos\theta_2 - \ell_1\ell_2\dot{\theta_1}^2\sin(\theta_1+\theta_2)(\frac{1}{2}m_2 + m_3) \end{bmatrix} \tag{5.2.2}$$

and there is a single constraint equation between the angles:

$$\Phi = \ell_1\sin\theta_1 - \ell_2\sin\theta_2 = 0\ . \tag{5.2.3}$$

Therefore, $\Phi_q^T\lambda$ in (5.1.1) is given by $\Phi_q^T\lambda = \begin{pmatrix} \ell_1\cos\theta_1 \\ -\ell_2\cos\theta_2 \end{pmatrix}\lambda$. Note that in this example, λ is a scalar. Thus, in addition to generating the constraint equation (5.2.3), DYNAFLEX automatically generated the constraint force $\lambda(t)$. For this example, the challenge now is to analyze the equations with computer algebraic methods such as RIFSIMP.

5.3 Implicit Reduced Involutive Form

We use implicit RIFSIMP to get all hidden constraints in the multibody dynamical general model (5.1.1)–(5.1.2).

Definition 5.3.1. [Implicit Reduced Involutive Form] Given a differential algebraic equation system (DAEs), let $\prec$ be a ranking of this DAEs. If there exist derivatives $r_1, ..., r_k$ such that L is leading linear in $r_1, ..., r_k$ with respect to $\prec$ (i.e. $L = A[r_1, \cdots, r_k]^T - b = 0$), N is leading nonlinear and

$$[r_1, \cdots, r_k]^T = A^{-1}b\,, \quad N = 0\,, \quad \det(A) \neq 0 \tag{5.3.1}$$

is in reduced involutive form. Then the system $L = 0, N = 0$ is said to be in implicit reduced involutive form (implicit-RIF form).

This form is of interest because computing A^{-1} symbolically in practice can be very expensive. Sometimes implicit RIF-form can be obtained very cheaply, just by appropriate differentiation of the constraints.

To convert a system of general form (5.1.1,5.1.2) with non-trivial constraints to implicit reduced involutive form, one would have to at least differentiate the constraints twice [19]. Carrying this out we obtain

$$M\ddot{q} + \Phi_q^T\lambda = F(t, q, \dot{q}) \tag{5.3.2}$$

$$D_t^2\Phi = \Phi_q\ddot{q} + H\dot{q} + 2\Phi_{tq}\dot{q} + \Phi_{tt} = 0 \tag{5.3.3}$$

$$D_t\Phi = \Phi_q\dot{q} + \Phi_t = 0 \tag{5.3.4}$$

$$\Phi(t, q) = 0 \tag{5.3.5}$$

where $\Phi_{tq} = \frac{\partial \Phi_q}{\partial t}$, $\Phi_{tt} = \frac{\partial^2 \Phi}{\partial t^2}$ and

$$H = \begin{bmatrix} \sum_i \Phi^1_{q_1 q_i} \dot{q}_i & \cdots & \sum_i \Phi^1_{q_n q_i} \dot{q}_i \\ \vdots & \vdots & \vdots \\ \sum_i \Phi^m_{q_1 q_i} \dot{q}_i & \cdots & \sum_i \Phi^m_{q_n q_i} \dot{q}_i \end{bmatrix}$$

We now show:

Theorem 5.3.2. *Consider the ranking $\prec$ defined by $q \prec \dot{q} \prec \lambda \prec \ddot{q} \prec \dot{\lambda} \prec \dddot{q} \cdots$ where the dependent variables q, λ are ordered lexicographically $q_1 \prec q_2 \prec \ldots$ and $\lambda_1 \prec \lambda_2 \prec \ldots$. The system (5.3.2, 5.3.3, 5.3.4, 5.3.5) is in implicit* RIF*-form with A, b, $[r_1, \ldots, r_k]^T$ in the definition above given by:*

$$A = \begin{bmatrix} M & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix}, \quad b = \begin{bmatrix} F(t, q, \dot{q}) \\ -H\dot{q} - 2\Phi_{tq}\dot{q} - \Phi_{tt} \end{bmatrix}, \quad [r_1, \ldots, r_k]^T = \begin{pmatrix} \ddot{q} \\ \lambda \end{pmatrix} \tag{5.3.6}$$

and $N = \{\Phi = 0,\ \Phi_q \dot{q} + \Phi_t = 0\}$, under the assumption that $\det(A) \neq 0$. *[19]*

Applying this to the slider crank, with the ranking $\theta_1 \prec \theta_2 \prec \dot{\theta}_1 \prec \dot{\theta}_2 \prec \lambda \prec \ddot{\theta}_1 \prec \ddot{\theta}_2 \prec \dot{\lambda} \prec \cdots$, we obtain the implicit RIF-form for the system.

$$AX = b \tag{5.3.7}$$

Here

$$A = \begin{bmatrix} \ell_1^2 \left(\frac{1}{4} m_1 + m_2 + m_3\right) + J_1 & -\ell_1 \ell_2 M \cos\theta_3 & \ell_1 \cos\theta_1 \\ -\ell_1 \ell_2 M \cos\theta_3 & {\ell_2}^2 \left(\frac{1}{4} m_2 + m_3\right) + J_2 & -\ell_2 \cos\theta_2 \\ \ell_1 \cos\theta_1 & -\ell_2 \cos\theta_2 & 0 \end{bmatrix};$$

$$b = \begin{bmatrix} -\ell_1 g \left(\frac{1}{2} m_1 + m_2 + m_3\right) \cos\theta_1 - \ell_1 \ell_2 M \dot{\theta_2}^2 \sin\theta_3 \\ \ell_2 M g \cos\theta_2 - \ell_1 \ell_2 M \dot{\theta_1}^2 \sin\theta_3 \\ \ell_1 \sin\theta_1 \dot{\theta_1}^2 - \ell_2 \sin\theta_2 \dot{\theta_2}^2 \end{bmatrix};$$

$$X^T = [r_1, r_2, r_3]^T = [\ddot{\theta}_1, \ddot{\theta}_2, \lambda]^T$$

where for the sake of a compact presentation, we have used $M = \frac{1}{2}m_2 + m_3$ and $\theta_3 = \theta_1 + \theta_2$. The constraints are:

$$\Phi = \ell_1 \sin\theta_1 - \ell_2 \sin\theta_2 = 0, \tag{5.3.8}$$

$$D_t\Phi = \ell_1 \cos\theta_1\dot{\theta}_1 - \ell_2 \cos\theta_2\dot{\theta}_2 = 0. \tag{5.3.9}$$

We note that neither the matrix A, nor vector b include the dependent variable $\lambda(t)$, which was generated by DYNAFLEX. Because of this, the ode system with constraints (5.3.7, 5.3.8, 5.3.9), can be symbolically solved separately for the variables $\theta_1(t)$, $\theta_2(t)$.

5.4 Explicit Reduced Involutive Form with Large Expression Management

Above, we obtained the implicit RIF-form for the slider crank as equations (5.3.7, 5.3.8, 5.3.9). We could in principle symbolically invert the matrix A using Maple, and get the explicit RIF-form, but Maple will give a huge output for A^{-1} and require a lot of memory and computation time. Therefore, we have used symbolic inversion using large expression management (LEM). A full discussion of large expression management will be given elsewhere. Here we give a brief outline.

We have used the MAPLE package `LargeExpressions` which is based on tools first developed for perturbation calculations in fluid mechanics [2]. From the paper [2], we have the following definition for a hierarchy and the main idea for hierarchical representations.

Definition 5.4.1. [Hierarchy] A hierarchy is an ordered list $[S_0, S_1, ...]$ of symbols, together with an associated list $[D_0, D_1, ...]$ of definitions of the symbols. For each $s \in S_i$ with $i \geq 1$, there is a definition $d \in D_i$ of the form $s = f(\sigma_1, \sigma_2, ..., \sigma_k)$ where f is some well-understood function such as an elementary function and each σ_j is a symbol in $[S_0, S_1, ..., S_{i-1}]$ and is thus lower in the hierarchy than s.

A computation sequence c is recursively defined as an expression of the form $c = g(s_1, s_2, ..., s_k)$ containing symbols s_j from a known hierarchy, together with the computation sequences defined by the associated definitions $d_1, d_2, ..., d_k$ of the symbols appearing in c. Obviously a computation sequence defined in terms of symbols in S_0, the set of atoms of the system, is just an expression.

Intuitively, a hierarchy is a framework for constructing computation sequences, and a computation sequence is an expression defined in terms of simpler expressions. We used the package `LargeExpressions` to code our own LU symbolic decomposition routine and also the forward and backward substitutions for solving a linear system. The details will be given elsewhere [20]. After LU decomposition with pivoting and zero-recognition, we get matrices L and U with the pivoting P as follows:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \frac{-{\ell_1}^2\left(\frac{1}{4}m_1+m_2+m_3\right)+J_1}{\ell_1 \ell_2 M \cos\theta_3} & 1 & 0 \\ \frac{-\cos\theta_1}{\ell_2 M \cos\theta_3} & \frac{-4W_3}{W_1} & 1 \end{bmatrix} \qquad P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.4.1}$$

$$U = \begin{bmatrix} -\ell_1 \ell_2 M \cos\theta_3 & {\ell_2}^2\left(\frac{1}{4}m_2 + m_3\right) + J_2 & -\ell_2 \cos\theta_2 \\ 0 & -\frac{1}{8}W_1 & \frac{1}{2}W_2 \\ 0 & 0 & -2W_4 \end{bmatrix} \tag{5.4.2}$$

where $PA = LU$. Now using forward and backward substitution in the usual way, we get X coded with the `LargeExpressions` package.

After that, we get Explicit RIF-form in the computation sequence form:

$$\ddot{\theta}_1 = -W_7 \tag{5.4.3}$$

$$\ddot{\theta}_2 = \frac{8}{W_1}[-\ell_1 g \left(\frac{1}{2}m_1 + m_2 + m_3\right) \cos\theta_1 - \ell_1 \ell_2 \dot{\theta}_2^2 \sin(\theta_1 + \theta_2) \left(\frac{1}{2}m_2 + m_3\right) - \frac{1}{4}W_5 + \frac{1}{4}W_2 W_6] \tag{5.4.4}$$

$$\lambda = \frac{1}{2}W_6 \tag{5.4.5}$$

and with the constraints

$$\Phi = \ell_1 \sin\theta_1 - \ell_2 \sin\theta_2 = 0; \tag{5.4.6}$$

$$D_t\Phi = \ell_1 \cos\theta_1\dot{\theta}_1 - \ell_2 \cos\theta_2\dot{\theta}_2 = 0. \tag{5.4.7}$$

The complete symbolic expressions for the $W[i]$ are given in the Appendix A.

The equations (5.4.3, 5.4.4, 5.4.5, 5.4.6, 5.4.7) are also the explicit RIF form for the 2d slider crank. The symbolic preprocessing helps find all the hidden constraints in the general DAE system (5.1.1,5.1.2). This preprocessing procedure makes it possible for us to integrate only interesting variables without computing all the independent variables. For example, to this two dimensional slider crank, as already mentioned, now we can symbolically separate equation (5.4.5) containing $\lambda(t)$ from the other equations (5.4.3, 5.4.4, 5.4.6, 5.4.7) and we can solve directly for $\theta_1(t)$ and $\theta_2(t)$.

5.5 Numerical Integration using Computation Sequences

We now show how the two second-order equations (5.4.3, 5.4.4) for θ_1 and θ_2 can be integrated. In general it is not possible to unveil all the expressions in the array W. To do so would be to introduce the memory problems that were avoided by using the `Veil` command. Instead we set up computation sequences to pass to `dsolve`. We use MAPLE for the calculations. In particular we only use the MAPLE `LargeExpressions` command `Unveil` to a minimal depth 1, to express the computation sequence of substitution rules SW:

```
> SW := [seq(W[i] = Unveil[W](W[i],1), i = 1 .. LastUsed[W])];
```

Then in order to use `dsolve/numeric`, we make the variable substitutions as follows:

```
> YW := [ subs(θ1(t) = Y[1], θ2(t) = Y[3], θ̇1(t) = Y[2], θ̇2(t) = Y[4], SW)];
```

Now we use the `codegen` package in Maple to make a procedure for the numeric integration.

```
> f := codegen[makeproc]( YW, YP[1] = Y[2], YP[2] = rhs(odesys[1]),
YP[3] = Y[4],YP[4] =  rhs(odesys[2]], parameters = [N, t, Y, YP] );
```

In order to complete the demonstration, we integrate the system for sample numerical values. For the parameters, we choose the following values.

$$l1 := 1; l2 := 2; m1 := 1; m2 := 1; m3 := 2; g := 9.8; J1 := 4.5; J2 := 5.5;$$

For the initial conditions, we choose the following

$$\theta_1(0) = 0, \dot{\theta}_1(0) = 1, \theta_2(0) = 0, \dot{\theta}_2(0) = \frac{1}{2}$$

which are consistent with the constraint equations

$$\Phi = \ell_1 \sin\theta_1 - \ell_2 \sin\theta_2 = 0; \tag{5.5.1}$$

$$D_t\Phi = \ell_1 \cos\theta_1\dot{\theta_1} - \ell_2 \cos\theta_2\dot{\theta_2} = 0. \tag{5.5.2}$$

The Maple numeric dsolve routine can solve the system as follows.

```
> ics := array( [0,1,0,1/2] );
> dvars := [θ1(t), θ̇1(t), θ2(t), θ̇2(t)];
> dsol := dsolve(numeric, number=4, procedure =f, start =0,
  initial = ics, procvars = dvars);
```

Since the system is being integrated as an ordinary ODE system rather than a DAE one, we gain in speed but expect to lose accuracy. If we pick up some points for testing the numeric ODE solutions with respect to the algebraic constraints, we see a slow loss of precision. For example, when $t = 1$, we have the solutions *dsol* as $[t = 1.,\ \theta_1(t) = .2629,\ \dot{\theta}_1(t) = -.4375,\ \theta_2(t) = .1303,\ \dot{\theta}_2(t) = -.2130]$, and errors in the constraints (5.5.1, 5.5.2) are $0.14e-7$ and $-0.69e-7$ respectively. When $t = 20$, the errors are $0.42e-5$ and $0.59e-6$ respectively.

Plots of the numerical simulation of the two dimensional slider crank are shown below as figure 5.5.2 and figure 5.5.3. It tells us that the 2d slider crank is drift oscillation with the given initial conditions $\dot{\theta}_1(0) = 1$ and $\dot{\theta}_2(0) = 1/2$. If we increase the initial speeds of θ_1 and θ_2, for example let $\dot{\theta}_1(0) = 2$ and $\dot{\theta}_2(0) = 1$ which are consistent initial conditions, we get figure 5.5.4 and figure 5.5.5 which show us the monotonic mode of the angle θ_1.

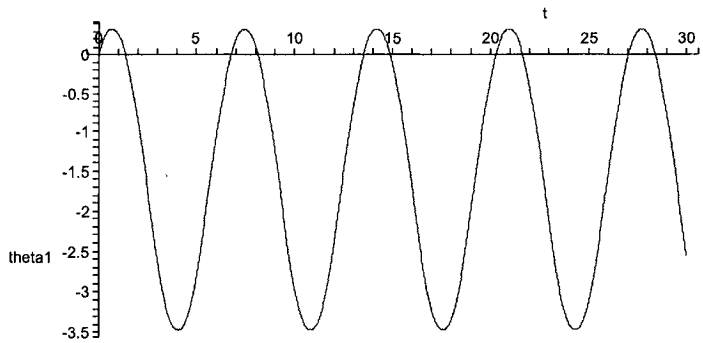

Figure 5.5.2: The angular θ_1 oscillating movement

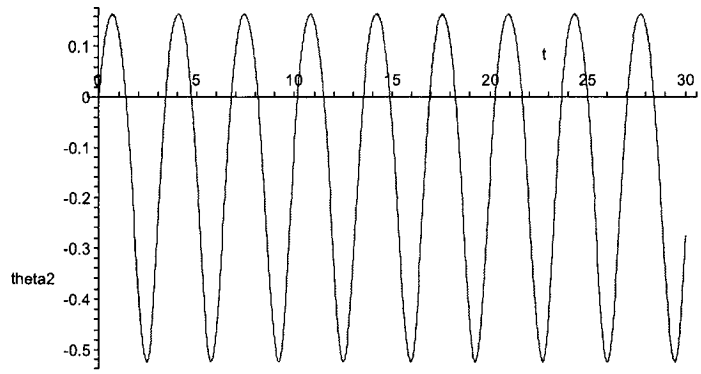

Figure 5.5.3: The angular θ_2 oscillating movement

5.6 Application of Numerical Algebraic and Analytic Geometry

In this section we discuss the use of some new techniques that are being applied to DAE such as those studied here. A significant problem in large systems, such as the target systems for this work, is the finding of consistent initial conditions. Again

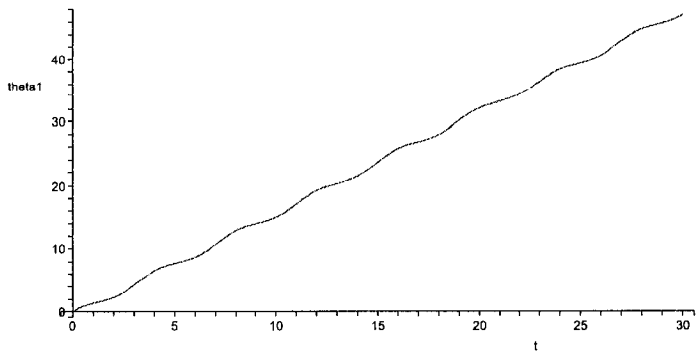

Figure 5.5.4: Monotonic mode for θ_1.

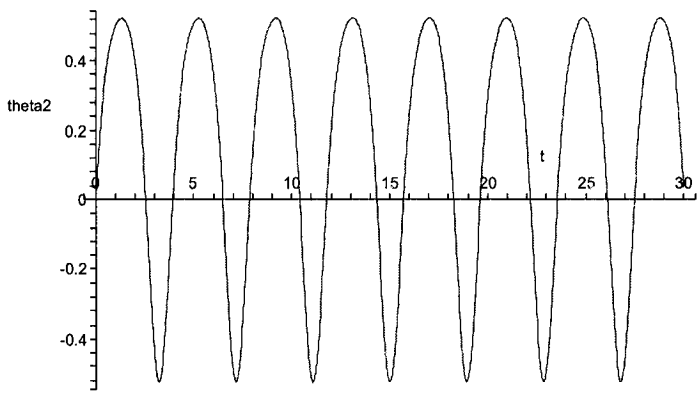

Figure 5.5.5: The angular motion θ_2 corresponding to monotonic mode of θ_1.

there is a need for combined symbolic-numeric methods. The first area that can be harnessed to DAE is that of Numerical Algebraic Geometry [17]. In that approach, components of a polynomial system are characterized by "witness" points, which result from intersecting the component with random linear spaces of complementary dimension. For polynomial DAE, this gives a method of determining points on the constraints, for the consistent initialization of numerical integrators. Also the hybrid

symbolic-numeric completion process described in [13] can be applied to polynomial DAE, using numerical algebraic geometry.

Another interesting possibility for analytic DAE is to extend the methods of Numerical Algebraic Geometry [17] as applied to polynomial PDE [13] to the case of analytic DAE. This requires using the substitution of approximate points on the components of the leading nonlinear systems to test ideal membership in the application of RIF. It is natural to generalize the techniques of algebraic geometry to the analytic case and to characterize irreducible components of analytic functions by points cut out by the intersection of the components with random linear spaces of complementary dimension. The systems are regularized by embedding in appropriate square systems, as in the polynomial case. Newton methods converge to the points on the components, locally. See [9, §5.2.3] for material on the range of non-constant entire functions. Global results like those in the polynomial case, using homotopy continuation, are much harder to obtain. Usually computations must be executed locally on some compact set.

For example, consider the determination of points on the components of an analytic function $f(z, w) = 0$ on some compact set $U \subset \mathbb{C}^2$. By intersecting the component with a random line $\alpha z + \beta w + \gamma = 0$, we obtain a univariate problem whose solutions can be found on $\mathbb{C}$. Powerful tools are already available to automate this process, for example the Maple command `Rootfinding[Analytic]`. This uses the Cauchy Integral Formula to determine the number of zeros in a given subdomain of $\mathbb{C}$. This subdomain is then divided to isolate the roots. We note that α-theoretic methods can lead to guaranteed convergence, in the isolated zero case, provided certain local criteria are met [4, 5]. A forthcoming work will be devoted to such *Numerical Analytic Geometry* techniques.

5.7 Conclusion

This paper uses a simple multibody dynamic system to show how we can combine symbolic methods and numeric methods for simulating mechanical systems described by higher-index DAE systems. An advantage of this combination is not only that it helps the numeric method to find consistent initial conditions, but it also extend the potential for symbolic methods, such as RIFSIMP, to solve more complex symbolic multibody dynamic systems. Using computation sequences to invert a symbolic matrix allows the calculation to be completed in less memory and less time, more details being given in a upcoming paper [20].

We also discussed applying new methods from Numerical Algebraic Geometry and Numerical Analytic Geometry to DAE. Already in her analysis of analytic DAE, Ilie used computation sequences to establish polynomial cost methods. The method upon which she based her complexity analysis is Pryce's structural analysis of DAE [6, 7, 3]. Specifically Pryce's method can be regarded as an efficient way to obtain implicit RIFSIMP forms, in certain cases. Numerical analytic geometry naturally partners such methods.

Bibliography

[1] Corless, R. M. Essential Maple 7. *Springer-Verlag*, 2002.

[2] Corless, R. M., Jeffrey, D. J., Monagan, M. B. and Pratibha. Two Perturbation Calculation in Fluid Mechanics Using Large-Expression Management. *J. Symbolic Computation*, 11, 1-17, 1996.

[3] Corless, R. M. and Ilie, S. Polynomial Cost for Solving IVP for High-Index DAE, *submitted.*

[4] Giusti, M., Lecerf, G., Salvy, B. and Yakoubsohn, J. C. Location and Approximation of Clusters of Zeros of Analytic Functions. *Foundations of Computational Mathematics*, 5, 257-311, 2005.

[5] Giusti, M., Lecerf, G., Salvy, B. and Yakoubsohn, J. C. On Location and Approximation of Clusters of Zeros: Case of Embedding Dimension One. *Foundations of Computational Mathematics*, 7(1), 1-58, 2007.

[6] Ilie, S., Corless, R. M. and Reid, G. J. Numerical Solutions of Index-1 Differential Algebraic Equations can be Computed in Polynomial Time. *Numerical Algorithms*, 41, 161–171, 2006.

[7] Ilie, S. Computational Complexity of Numerical Solutions of Initial Value Problems for Differential Algebraic Equations. PHD Thesis, *University of Western Ontario*, 2005.

[8] Monagan, M. B. Gauss: a Parameterized Domain of Computation System with Support for Signature Functions. *Proceedings of DISCO*, Spring-Verlag LNCS 722, 81-94, 1993.

[9] Nishino, T. Function Theory in Several Complex Variables. *Translations of Mathematical Monographs*, 193, AMS, 2001.

[10] Pryce, J. D. Solving High-index DAEs by Taylor Series. *Numer. Algorithms*, 19, 195-211, 1998.

[11] Pryce, J. D. A Simple Structural Analysis Method for DAEs. *BIT*, 41(2), 364-394, 2001.

[12] Reid, G. J., Wittkopf, A. D. and Boulton, A. Reduction of Systems of Nonlinear Partial Differential Equations to Simplified Involutive Forms. *Eur. J. Appl. Math.*, 7, 604-635, 1996.

[13] Reid, G. J., Verschelde, J., Wittkopf, A. D. and Wu, W. Symbolic-Numeric Completion of Differential Systems by Homotopy Continuation. *ISSAC*, ACM Press, 269-276, 2005.

[14] Schiehlen, W. Multibody Systems Handbook. *Springer-Verlag: Berlin*, 1990.

[15] Shi, P. and McPhee, J. J. Dynamics of Flexible Multibody Systems Using Virtual Work and Linear Graph Theory. *Multibody System Dynamics*, 4(4), 355-381, 2000.

[16] Shi, P. and McPhee, J. J. Symbolic Programming of a Graph-Theoretic Approach to Flexible Multibody Dynamics. *Mechanics of Structures and Machines*, 30(1), 123-154, 2002.

[17] Sommese, A. J. and Wampler, C. W. The Numerical Solution of Systems of Polynomials Arising in Engineering and Science. *World Scientific Press, Singapore*, 2005.

[18] Stetter, H. Numerical Polynomial Algebra. *SIAM*, 2005.

[19] Zhou, W., Jeffrey, D. J., Reid, G. J., Schmitke, C. and McPhee, J. J. Implicit Reduced Involutive Forms and Their Application to Engineering Multibody Systems. *IWMM*, LNCS 3519, 31-43, 2005.

[20] Zhou, W., Carette, J., Jeffrey, D. J. and Monagan, M. B. Hierarchical Representations with Signatures for Large Expression Management. *AISC*, LNAI 4120, 254-268, 2006.

Chapter 6

Handling Large Polynomial Equations in Symbolic Computation of Limit Cycles

In the computation of limit cycles of nonlinear dynamical systems, one is often forced to deal with very large expressions involved in symbolic computation. In particular, when the attention is focused on small (local) limit cycles bifurcating from a degenerate Hopf critical point, one must solve multivariate polynomial equations, which usually gives rise to very large expressions in symbolic computation. In this paper, an example related to Hilbert's 16th problem is presented to show the natural appearance of large expressions in computation and an approach to solve the problem.

6.1 Introduction

Symbolic computation and computer algebra system software such as Maple, Mathematica have been widely used in solving problems arising from mathematics, physics, engineering and other disciplines. One of the applications is to find the limit cycles of nonlinear systems. In particular, when one considers degenerate Hopf bifurcation at a fixed point, leading to multiple limit cycles, one must symbolically compute the

focus values. Then one should solve a system of polynomial equations to determine parameter values such that as many focus values become zero as possible. This usually yields very large expressions in symbolic computation, and one cannot avoid this by purely using numerical computations. To illustrate this, in the following, as an example, we will consider Hilbert's 16th problem.

The well-known Hilbert's 16th problem has remained unsolved since D. Hilbert proposed the 23 mathematical problems at the Second International Congress of Mathematics in 1900 [2]. Recently, a modern version of the second part of Hilbert's 16th problem was formulated by S. Smale, chosen as one of the 18 challenging mathematical problems for the 21st century [13]. To be more specific, consider the following planar system:

$$\frac{dx}{dt} = P_n(x, y), \quad \frac{dy}{dt} = Q_n(x, y), \tag{6.1.1}$$

where $P_n(x, y)$ and $Q_n(x, y)$ represent n^{th}-degree polynomials of x and y. The second part of Hilbert's 16th problem is to find the upper bound $K = H(n) \leq n^q$ on the number of limit cycles that the system can have, where q is a universal constant. In general, this is a very difficult problem, in particular, for determining large (global) limit cycles. A recent survey article [5] (and more references therein) has comprehensively discussed this problem and reported the recent progress.

If the problem is restricted to the neighbourhood of isolated fixed points, the problem is reduced to studying degenerate Hopf bifurcations, which give rise to considering fine focus points. In the past 50 years, many researchers have investigated the local problem and obtained many results (e.g., see [1, 3, 6, 8, 10]). In the last two decades, much progress on finite cyclicity near a fine focus point or a homoclinic loop has been achieved. Roughly speaking, the so-called finite cyclicity means that at most a finite number of limit cycles can exist in some neighbourhood of focus

points or homoclinic loop under small perturbations on the system's parameters.

For a quadratic system, it is well known that the maximum number of small limit cycles is three [1]. However, globally, it is unsolved even for quadratic systems. For cubic order systems, on the other hand, the best results published so far are twelve limit cycles, found from the following system [15, 16, 17, 18, 19]:

$$\begin{aligned}\frac{dx}{dt} &= a\,x + b\,y + a_{30}\,x^3 + a_{21}\,x^2\,y + a_{12}\,x\,y^2 + a_{03}\,y^3,\\ \frac{dy}{dt} &= b\,x + a\,y + b_{30}\,x^3 + b_{21}\,x^2\,y + b_{12}\,x\,y^2 + b_{03}\,y^3,\end{aligned} \tag{6.1.2}$$

where a, b, a_{ij}'s and b_{ij}'s are constant coefficients.

The key step in finding the number of limit cycles of a system in the neighbourhood of a fixed point (which is a linear center) is to compute the focus values of the point. This is equivalent to calculate the normal form of the system associated with Hopf singularity. The basic idea of normal form theory is employing successive coordinate transformations to systematically construct a simplified form of the original system without changing the fundamental dynamics of the system in the vicinity of an equilibrium point. Although it is easy to construct an "abstract" form of the normal form for a given singularity, it is difficult to obtain the explicit expressions of normal form given in terms of the coefficients of the original differential equations. Therefore, computer algebra systems such as Maple, Mathematica, etc. have been introduced in computing normal forms. Recently, many researchers have paid attention to developing efficient computational methods with the aid of computer algebra systems. Among various methods, a perturbation technique [20] has been proved computationally efficient. Usually, if the dimension of a system is higher than the dimension of the center manifold, one needs to apply center manifold theory first in order to reduce the dimension of the system. However, the perturbation technique does not require this step for it combining normal form theory with center manifold

theory to generate a unified approach.

Since in this paper we do not intend to discuss normal form computation, we suppose that the normal of system (6.1.1) has been obtained in the polar coordinates as follows (interested readers can find the details of normal form computation in [20]):

$$\frac{dr}{dt} = r\,(v_0 + v_1\,r^2 + v_2\,r^4 + \cdots + v_k\,r^{2k}), \tag{6.1.3}$$

$$\frac{d\theta}{dt} = \omega + t_1\,r^2 + t_2\,r^4 + \cdots + t_k\,r^{2k}, \tag{6.1.4}$$

up to $(2k+1)$th order term, where both v_k and t_k are explicitly expressed in terms of the original system's coefficients. v_k is called the kth-order focus value of the Hopf-type critical point (the origin). Note that v_0 is the term obtained from linear perturbation.

The basic idea of finding k small limit cycles of system (6.1.1) around the origin is as follows: First, find the conditions such that $v_1 = v_2 = \cdots = v_{k-1} = 0$ (v_0 is automatically satisfied at the critical point), but $v_k \neq 0$, and then perform appropriate small perturbations to prove the existence of k limit cycles. This indicates that the procedure for finding multiple limit cycles involves two steps: Computing the focus values (i.e., computing the normal form) and solving the coupled nonlinear equations: $v_1 = v_2 = \cdots = v_{k-1} = 0$. In the following, we give a sufficient condition for the existence of small limit cycles. (The proof can be found in [17, 18, 19].)

Theorem 6.1.1. *If the focus values v_i in Eq. (6.1.3) satisfy the following conditions:*

$$v_i\,v_{i+1} < 0 \quad \textit{and} \quad |v_i| \ll |v_{i+1}| \ll 1, \quad \textit{for} \quad i = 0,\, 1,\, 2,\, \ldots,\, k-1,$$

the polynomial equation given by $\frac{dr}{dt} = 0$ in Eq. (6.1.3) has k positive real roots of r^2, and thus the original system (6.1.1) has k limit cycles in the vicinity of the origin.

6.2 The Liénard System

In this section, we consider a simplified version of the 16th Hilbert's problem – the Liénard equation. Most of the early history in the theory of limit cycles was stimulated by practical problems displaying periodic behaviour. For example, the differential equation derived by Rayleigh [12] in 1877, related to the oscillation of a violin string, is given by

$$\ddot{x} + \epsilon \left(\tfrac{1}{3}\, \dot{x}^2 - 1\right) \dot{x} + x = 0, \tag{6.2.1}$$

where ϵ is a small perturbation parameter. Following the invention of the triode vacuum tube, which was able to produce stable self-excited oscillations of constant amplitude, van der Pol [14] obtained the following differential equation to describe this phenomenon:

$$\ddot{x} + \epsilon \left(x^2 - 1\right) \dot{x} + x = 0. \tag{6.2.2}$$

Systems (6.2.1) and (6.2.2) can both display periodic behaviour.

Perhaps the most famous class of differential equations, which generalize equation (6.2.2), are those first investigated by Liénard [7] in 1928:

$$\ddot{x} + f(x)\, \dot{x} + g(x) = 0, \tag{6.2.3}$$

where $f(x)$ and $g(x)$ are polynomials. Letting $\dot{x} = \tilde{y}$, one obtains the above system described in phase plane:

$$\begin{aligned} \dot{x} &= \tilde{y}, \\ \dot{\tilde{y}} &= -g(x) - f(x)\, \tilde{y}. \end{aligned} \tag{6.2.4}$$

Further, let $\tilde{y} = y - F(x)$, where $F(x) = \int_0^x f(s)\, ds$, then we have the following

equivalent system:

$$\begin{aligned} \dot{x} &= \tilde{y} = y - F(x), \\ \dot{y} &= \dot{\tilde{y}} + \frac{dF}{dx}\dot{x} = -g(x) - f(x)\,\tilde{y} + f(x)\,\tilde{y} = -g(x). \end{aligned} \tag{6.2.5}$$

Let 'deg' denote the degree of a polynomial, and $\hat{H}(i,j)$ be the maximum number of small-amplitude limit cycles, where i and j are the degrees of f and g, respectively. Then the existing results for the Liénard system (6.2.5) are summarized in Table 1, where the numbers marked by * are new results obtained in this paper. Note that the table is symmetric with $\deg(f) = \deg(g)$, i.e., $\hat{H}(i,j) = \hat{H}(j,i)$ [9]. The results given in Table 1 are actually also applicable to the so called 'generalized' Liénard systems which will be discussed below, since the normal form of generalized Liénard systems is identical to that of system (6.2.5). One should note that the numbers listed in the paper only denote the maximum number of small limit cycles which may exist in the neighbourhood of the origin. That is, for example, $\hat{H}(5,5) \leq 6$. These numbers do not include any global (large) limit cycles or any small limit cycles which may appear in the vicinity of other non-zero Hopf critical points.

Recently, there has been a great deal of progress when considering the bifurcation of small amplitude limit cycles from the origin for generalized Liénard systems of the form

$$\begin{cases} \dot{x} = \phi(y) - F(x), \\ \dot{y} = -g(x), \end{cases} \tag{6.2.6}$$

where $\phi(y)$ is in the polynomial form

$$\phi(y) = y + c_2 y^2 + c_3 y^3 + \cdots \tag{6.2.7}$$

Deg of f																		
	50	↑	↑	38														
	49	24	33	38														
	48	24	32	36														
	⋮	⋮	⋮	⋮														
	13	6	9	10														
	12	6	8	10														
	11	5	7	8														
	10	5	7	8														
	9	4	6	8	9													
	8	4	5	6	9	10*												
	7	3	5	6	8	9*												
	6	3	4	6	7	8*												
	5	2	3	4	6	6	8*	9*	10*									
	4	2	3	4	4	6	7	8	9	9								
	3	1	2	2	4	4	6	6	6	8	8	8	10	10	⋯	36	38	38
	2	1	1	2	3	3	4	5	5	6	7	7	8	9	⋯	32	33	→
	1	0	1	1	2	2	3	3	4	4	5	5	6	6	⋯	24	24	→
		1	2	3	4	5	6	7	8	9	10	11	12	13	⋯	48	49	50
	Deg of g																	

Table 6.2.1: The values of $\hat{H}(i,j)$ for the generalized Liénard systems when f and g are of varying degrees.

Further for definiteness, let

$$\begin{aligned} f(x) &= a_1 + 2\,a_2\,x + 3\,a_3\,x^2 + 4\,a_4\,x^3 + \cdots \\ g(x) &= b_1\,x + b_2\,x^2 + b_3\,x^3 + b_4\,x^4 + \cdots \end{aligned} \tag{6.2.8}$$

then

$$F(x) = a_1\, x + a_2\, x^2 + a_3\, x^3 + \cdots \tag{6.2.9}$$

where $a_1 = 0$, $b_1 = 1$. Introduce the following transformation:

$$u^2 = 2\,G(x) = 2\int_0^x g(s)\; ds, \quad u(0) = 0, \quad u'(0) > 0 \tag{6.2.10}$$

so that $u^2 = x^2 + O(x^3)$ as $x \to 0$. Therefore, this transformation defines an analytic change of coordinates in a neighbourhood of the origin. Let the inverse transformation be given by

$$x = \xi(u) = u + O(u^2) \quad \text{as} \quad u \to 0, \tag{6.2.11}$$

and thus,

$$\frac{g(\xi(u))}{u} \longrightarrow \frac{(g'(0))^{1/2}}{1}. \tag{6.2.12}$$

Finally, in the (u, y) coordinate plane, we obtain the new system:

$$\begin{cases} \dot{u} = u^{-1} g(\xi(u))\; [\,\phi(y) - F^*(u)\,], \\ \dot{y} = -\,g(\xi(u)). \end{cases} \tag{6.2.13}$$

By the condition (6.2.12), the local qualitative behaviour of system (6.2.13) is identical to the following system:

$$\begin{aligned} \dot{u} &= \phi(y) - F^*(u) = \phi(y) - (A_1\, u + A_2\, u^2 + A_3\, u^3 + A_4\, u^4 + A_5\, u^5 + \cdots), \\ \dot{y} &= -\,g(\xi(u)) = -\,u, \end{aligned} \tag{6.2.14}$$

where $A_1 = a_1 = 0$, $A_2 = a_2$, and other A_i's are given explicitly in terms of the original system coefficients a_i and b_i. The coefficients A_i's are listed in Appendix B. Then, with the Maple program developed in [20] for computing the normal form

of Hopf and generalized Hopf bifurcations, the focus values of system (6.2.14) can be found as follows:

$$
\begin{aligned}
v_0 &= -A_1\,, \\
v_1 &= -\tfrac{3}{8}A_3\,, \\
v_2 &= -\tfrac{5}{16}A_5 - \tfrac{5}{24}A_2^2 A_3\,, \\
v_3 &= -\tfrac{35}{128}A_7 - \tfrac{205}{1152}A_4^2 A_5 - (\,\tfrac{1885}{13824}A_4^2 + \tfrac{2}{3}A_2 A_4 + \tfrac{999}{8192}A_3^2\,)\,A_3\,, \\
&\vdots
\end{aligned}
$$

In general, it can be shown that

$$v_0 = v_1 = \cdots = v_{k-1} = 0,\ \ v_k \neq 0 \iff A_1 = A_3 = \cdots = A_{2k-1} = 0,\ \ A_{2k+1} \neq 0.$$

Therefore, in order for system (6.2.14) to have k small limit cycles, we require that $A_1 = A_3 = \cdots = A_{2k-1} = 0$, but $A_{2k+1} \neq 0$. For example, when $A_1 = 0$, but $A_3 \neq 0$, then system (6.2.14) or (6.2.6) has maximum one limit cycle around the origin; when $A_1 = A_3 = 0$, but $A_5 \neq 0$, then system (6.2.14) or (6.2.6) has maximum two limit cycles in the vicinity of the origin; etc. Since the coefficients A_i's are given in terms of a_i's and b_i's, one need to determine the values of a_i's and b_i's to satisfy these necessary conditions.

In the next two sections, we will present the results for $\hat{H}(6,5)$, $\hat{H}(7,5)$ and $\hat{H}(8,5)$. The general idea or procedure is to reduce the variables of the polynomial systems by using either 'eliminate' Maple command, or 'Gröebner' basis computation.

6.3 $\hat{H}(6,5) \leq 8$

To show $\hat{H}(6,5) \leq 8$, we need to solve the following equations:

$$A_3 = A_5 = A_7 = A_9 = A_{11} = A_{13} = A_{15} = 0.$$

Note that $A_1 = 0$ is already satisfied. Also note that when $\hat{H}(6,5) = 8$, $A_{17} \neq 0$. If we want to show $\hat{H}(6,5) \leq 8$, we must prove that it is not possible to have $A_{17} = 0$. Otherwise, $\hat{H}(6,5) \leq 9$. Therefore, we need to solve the above 7 nonlinear polynomial equations. In this case $a_i = 0$ for $i \geq 8$, and $b_j = 0$ for $j \geq 6$.

It is seen from Appendix A that one can use the coefficients a_3, a_5 and a_7 to solve the equations $A_3 = 0$, $A_5 = 0$ and $A_7 = 0$, respectively. Then one can use a_6 to solve $A_9 = 0$. The remaining three equations $A_{11} = A_{13} = A_{15} = 0$ have four coefficients b_2, b_3, b_4 and b_5. Thus, it might be possible to have $A_{11} = A_{13} = A_{15} = A_{17} = 0$ but $A_{19} \neq 0$, which implies that $\hat{H}(6,5) \leq 9$. However, it can be shown that if a set of values of b_2, b_3, b_4 and b_5 yields $A_{11} = A_{13} = A_{15} = A_{17} = 0$, then $A_{19} = 0$, leading to a center for this set of values. This is not what we want. Further it has been noted that these four equations are actually homogeneous. To show this, let

$$b_3 = B_3\, b_2^2, \quad b_4 = B_4\, b_2^3, \quad b_5 = B_5\, b_2^4, \tag{6.3.1}$$

and then substituting the above equations and the solutions of a_3, a_5, a_7 and a_6 into A_{11}, A_{13}, A_{15} and A_{17} results in the following expressions (form Maple output):

```
A11 := 2/2025*b2^7*(2*a4-a2*B3*b2^2)/(-81*B4+135*B3+220)
      *(87750*B3^3-210600*B4*B3^2+162000*B5*B3-14040*B4^2-32805*B5*B4^2-56862*B4^3
        +54675*B5*B3*B4+189540*B4^2*B3+46800*B3*B4-113400*B5*B4-104000*B5
        +41600*B4+19500*B3^2-91125*B5^2):
A13 :=-1/12150*b2^9*(2*a4-a2*B3*b2^2)/(-81*B4+135*B3+220)
      *(775200*B3*B4-11372400*B4^2*B3+13219200*B4*B3^2-14076000*B5*B3-12393000*B3^3*B4
        +11153700*B3^2*B4^2+7290000*B3^2*B5+7030800*B5*B4+157464*B4^4-4100625*B3*B5^2
        +984150*B5*B4^2+2733750*B5^2-969000*B3^2+12240*B4^2-2067200*B4+3261060*B4^3
```

```
        -2907000*B3^3+5168000*B5-6743250*B5*B3*B4+5163750*B3^4-3608550*B3*B4^3
        -1476225*B3*B5*B4^2+2460375*B3^2*B4*B5):
A15 := 17/4374000*b2^11*(2*a4-a2*B3*b2^2)/(-81*B4+135*B3+220)
      *(17950896*B3*B4^4+139840000*B4-349600000*B5+1278139500*B5*B3*B4+98325000*B3^3
        +103831200*B4^2-256151160*B4^3+65550000*B3^2-20092500*B5^2-802332000*B4*B3^2
        +1415880000*B5*B3-238698900*B5*B4^2+857530800*B4^2*B3-262200000*B3*B4-766764000*B5*B4
        +62329500*B3*B5*B4^2-339349500*B3^2*B4*B5-84144825*B4^2*B3^2*B5-39366000*B4*B3*B5^2
        +140241375*B4*B3^3*B5-123661728*B4^4-619447500*B3^4+2133054000*B3^3*B4
        -2191228200*B3^2*B4^2-1580040000*B3^2*B5+304357500*B3*B5^2+919152360*B3*B4^3
        +23619600*B4^2*B5^2-49572000*B5^2*B4+785351700*B3^3*B4^2-233735625*B3^2*B5^2
        -265523670*B3^2*B4^3-872613000*B3^4*B4+323190000*B3^3*B5+65610000*B5^3+363588750*B3^5):
A17 := -19/787320000*(2*a4-a2*B3*b2^2)/(-81*B4+135*B3+220)*b2^13*A17(B3,B4,B5):
```

where $A_{17}(B_3, B_4, B_5)$ is a polynomial of B_3, B_4 and B_5. There are two common factors in these four polynomials, which cannot be used since the solutions of these two factors generate centers. Thus, solutions must come from the remaining factors. However, note that there are only three parameters B_3, B_4, B_5 for solving the four polynomial equations. In generic case, there do not exist non-zero solutions of B_3, B_4 and B_5 for these four equations. Hence, the best possibility is $A_{11} = A_{13} = A_{15} = 0$, but $A_{17} \neq 0$, implying that $\hat{H}(6,5) \leq 8$.

We now want to find all the solutions of B_3, B_4 and B_5 from the three equations $A_{11} = A_{13} = A_{15} = 0$, and verify that none of these solutions satisfy $A_{17} = 0$. To achieve this, eliminating B_5 from the two equations $A_{11} = A_{13} = 0$ yields a solution for B_5 and a resultant equation $f_1 = 0$, given below (from Maple output).

```
B5 := 1/200/(-1280+2835*B3-2268*B4)*(-102400*B4+19683*B4^4-144000*B3^3-51120*B4^2+194400*B4^3
            -48000*B3^2+599400*B4*B3^2+328050*B3^2*B4^2-631800*B4^2*B3+38400*B3*B4
            -364500*B3^3*B4+151875*B3^4-131220*B3*B4^3):
f1 := (-3*B4+5*B3)*(-81*B4+220+135*B3)
     *(-5570560000*B4+14492520000*B3^3-19137945600*B4^2-14921288640*B4^3-2611200000*B3^2
       -45448776000*B4*B3^2+46920772800*B4^2*B3-12675852000*B3^3*B4+37539417600*B3^2*B4^2
       -26879104800*B3*B4^3-39060913500*B3*B4^4-153763596000*B3^3*B4^2+110114575200*B3^2*B4^3
       +106435822500*B3^4*B4+29350656000*B3*B4+6022998000*B4^4-5183190000*B3^4-29278462500*B3^5
       +8303765625*B3^6+5490848412*B4^5+16142520375*B3^2*B4^4-29893556250*B4*B3^5
       +44840334375*B3^4*B4^2-35872267500*B3^3*B4^3-3874204890*B4^5*B3+387420489*B4^6):
```

Next, substituting the above solution B_5 into $A_{15} = 0$ results in an equation $f_2 = 0$, where f_2 is

```
f2 := (-3*B4+5*B3)*(-81*B4+220+135*B3)
     *(3111526817321760000*B3*B4^3+669255752105281800 0*B3^2*B4^4+22764069605376000 00*B4^2*B3
       -606164151214080000*B4^3-47655682048000000*B4-2232630864000000000*B4*B3^2
       +509689921536000000*B3*B4+490549206726000000 0*B3^3*B4-6389236951351200000*B3^2*B4^2
       +245199882240000000*B3^3-282236446310400000*B4^2-23338600960000000*B3^2
       -524890835482752000*B4^4-920671484220000000*B3^4+1155169923885000000*B3^5
       +529096948223256000*B3*B4^4+4790564277789600000*B3^3*B4^2-242992255782456000 0*B3^2*B4^3
       -398442957773400000 0*B3^4*B4-507624670875375000 0*B4*B3^5+114180458812704000 00*B3^4*B4^2
       -121228067762247600 00*B3^3*B4^3+68892674151375000 0*B3^6-573516097576128 00*B4^5
       +180782185403016000*B4^6-1814614995849166800*B4^5*B3+2731742159669856 0*B4^8
       +114833385538864560*B4^7-1707104829882170250 0*B4^4*B3^3+6253416401554287000*B4^5*B3^2
       +172035559709332650 0*B4^6*B3^2-1275888655005608700*B4^6*B3-5883989549549089500*B4^5*B3^3
       +130620780796827600 00*B4^4*B3^4-3105120980466540 00*B4^7*B3-2581737006671156250 0*B3^5*B4^2
       +129363835223778750 00*B3^6*B4+2750555336879448 0000*B3^4*B4^3-2640053239950937500*B3^7
       -187171060159611000 00*B3^5*B4^3+1663915487921043750 0*B3^6*B4^2-8335317443245312500*B3^7*B4
       +179919350550000000 0*B3^8+330103642238992 8*B4^9+279689969977134300*B4^7*B3^2
       -105992274672113287 5*B4^6*B3^3+2681864580509651250*B4^5*B3^4-44770730122968120*B4^8*B3
       -467273784159137812 5*B3^5*B4^4+5561856967645125000*B3^6*B4^3-4324611475688203125*B3^7*B4^2
       +197856854421328125 0*B4*B3^8+17157594341220750*B4^8*B3^2-4448265199575750 00*B4^5*B3^5
       +222413259787875 00*B4^6*B3^4-76255974849870000*B4^7*B3^3+6178146110521875 00*B4^4*B3^6
       -228767924549610 0*B4^9*B3+137260754729766*B4^10-4034789337480468 75*B3^9
       +367746792292968750*B3^8*B4^2-136202515664062500*B3^9*B4-5883948676687500 00*B3^7*B4^3
       +22700419277343750*B3^10):
```

Finally, eliminating B_4 from the two equations $f_1 = 0$ and $f_2 = 0$ (by taking the different factors) gives the solution $B_4 = \frac{B_{4n}(B_3)}{B_{4d}(B_3)}$, and a resultant equation $f_3 = 0$, where f_3 is given as follows:

```
f3 := B3*(63*B3+496)*(189*B3+256)*(178605*B3^2-1357776*B3+241664)
        *(5779045668619503*B3^5+56315474719135758*B3^4+215033754040484148*B3^3
          +258790514446979496*B3^2-81098473487101056*B3-11257720644513280):
```

f_3 has five factors. The solution from the first factor, $B_3 = 0$, leads to a center, while the two solutions from the second and third factors render into the denominator of B_5 zero. The two solutions from the fourth factor yields $A_{11} \neq 0$ and thus are not solutions. The last factor has three real solutions, and all of them satisfy $A_{11} = A_{13} = A_{15} = 0$, but $A_{17} \neq 0$. This indeed implies that $\hat{H}(6,5) \leq 8$. The three solutions are (taken 30 digits):

```
(B3,B4) = (-2.77511371733863431746529008102l,-3.72306981674017854970618394007 7),
          (-0.10602006876159062416178891635 9,-0.00518341903127501255414855517 3),
          ( 0.33822530891806927386942237347 4,-0.24462182522747617755690277725 6).
```

It should be noted that there are actually infinitely many solutions since the parameters a_2, a_4 and b_2 can be chosen arbitrarily. A numerical example is given below:

$$
\begin{aligned}
a_2 &= a_4 = 1, \quad a_3 = 0.4, \\
a_5 &= 0.87794314313041191502038747924 4, \\
a_6 &= -0.44867762000924884776757942155 9, \\
a_7 &= -0.63861353041302154450016205132 3, \\
b_2 &= 0.6, \\
b_3 &= -0.99904093824190835428750442916 8, \\
b_4 &= -0.80418308041587856673653573105 7, \\
b_5 &= 0.11511783358513407149814706926 6.
\end{aligned}
$$

It should be pointed out that the above numerical values are obtained in the final step by solving a univariate polynomial with a numerical approach. With the above parameter values, we execute the Maple program developed in [20] for computing the normal forms of Hopf and generalized Hopf bifurcations to obtain the following

focus values (using 50 decimal digits in Maple):

$$
\begin{aligned}
v_0 &= 0,\\
v_1 &= 0,\\
v_2 &= 0.35 \times 10^{-49},\\
v_3 &= 0.68917508814324821179077156414598587963767012746490 \times 10^{-49},\\
v_4 &= 0.39109531090087755792768878835236843712508336184996 \times 10^{-48},\\
v_5 &= 0.11942168450861858237483295764520783970925881004195 \times 10^{-46},\\
v_6 &= -0.10126644455693454449604395413567346715520529430680 \times 10^{-44},\\
v_7 &= 0.60510290205877429614619932108504336044026006371765 \times 10^{-43},\\
v_8 &= -0.00174295159432804496353703392668507827553570565483,
\end{aligned}
$$

where v_i, $i = 0, 1, \ldots, 7$ should be exactly zero, but are not zero due to the truncation in the above critical parameter values. Since $|v_8| \ll 1$, it implies that there exists at most 8 small limit cycles in the vicinity of the origin. That is,

$$\hat{H}(6,5) \leq 8.$$

6.4 $\hat{H}(7,5) \leq 9$

Similarly, we can prove that $\hat{H}(7,5) \leq 9$. We omit the details but present the Maple algorithm (or Maple pseudo code) and a numerical example below.

Maple algorithm:

Step 1. Get the input A_i, $i = 1, 3, 5, \ldots$

Step 2. Suppose $\hat{H}(i,j) \leq k =$ the number of independent parameters.

Step 2. Solve the set of k polynomial equations: $A_1 = A_3 = A_{2k-1} = 0$. There are two cases.

Case (i) No solution exists, then set $k \to k-1$ and return to *Step* 1.

Case (ii) There exist solutions. Check if the solutions satisfy $A_{2k+1} \neq 0$. If no solution satisfies the condition, then set $k \to k-1$ and return to *Step* 1; if there exists at least one solution satisfying this condition, then $\hat{H}(i,j) = k$, and goto *Step* 3.

Step 3. Output k, the solution or print "solution exists for k".

A numerical example for this case is given as follows.

$$
\begin{aligned}
a_2 &= a_4 = b_2 = 1, \quad a_3 = \frac{2}{3}, \\
a_5 &= 1.11337809234121735418747395 2135, \\
a_6 &= 0.56120203702966493506958090 5877, \\
a_7 &= -0.00006824557123386824832129 2125, \\
a_8 &= -0.14344351585942918489595001 5870, \\
b_3 &= -0.29876426065098819399350725 7185, \\
b_4 &= -1.04782853689860360452049388 1638, \\
b_5 &= -0.60723889805814031800685351 2737,
\end{aligned}
$$

which gives the following focus values:

$$
\begin{aligned}
v_0 &= 0,\\
v_1 &= 0,\\
v_2 &= 0.2 \times 10^{-49},\\
v_3 &= -0.93228830715315051161362526822019321313096375337135 \times 10^{-49},\\
v_4 &= 0.2448 \times 10^{-47},\\
v_5 &= -0.730135 \times 10^{-47},\\
v_6 &= -0.84302773200555906169558965497808153113461923691600 \times 10^{-45},\\
v_7 &= -0.71496600125897833282249990911157592066256725883515 \times 10^{-44},\\
v_8 &= -0.942714920585 \times 10^{-41},\\
v_9 &= -0.00414339916083806333542873186316152173743875455175,
\end{aligned}
$$

which indeed shows $\hat{H}(7,5) \leq 9$.

6.5 $\hat{H}(8,5) \leq 10$

Now we turn to the case $\hat{H}(8,5)$. Let us check if we can have $\hat{H}(8,5) = 11$. Thus, we must solve 10 equations: $A_i = 0$, $i = 3, 5, 7, 9, 11, 13, 15, 17, 19, 21$, but $A_{23} \neq 0$. In this case, $a_i = 0$ for $i \geq 10$, and $b_j = 0$ for $j \geq 6$. Similarly, we can use a_3, a_5, a_7, a_9, a_8 and a_6 to solve the equations $A_i = 0$, $i = 3, 5, 7, 9, 11, 13$. Then the expressions A_{15}, A_{17}, A_{19}, A_{21} and A_{23} are simplified.

Similar to case $\hat{H}(6,5)$, it is seen from the expressions of A_{15}, A_{17}, A_{19}, A_{21} and A_{23} that they have a common factor $-2\,a_4 + a_2\,b_3$ whose root is not a solution. Thus, only 4 coefficients b_2, b_3, b_4 and b_5 are left for solving the 5 equations. We

are then led to study the system

$$\begin{aligned} A_{15}(b_2,b_3,b_4,b_5) &= 0,\\ A_{17}(b_2,b_3,b_4,b_5) &= 0,\\ A_{19}(b_2,b_3,b_4,b_5) &= 0,\\ A_{21}(b_2,b_3,b_4,b_5) &= 0,\\ A_{23}(b_2,b_3,b_4,b_5) &\neq 0, \end{aligned} \tag{6.5.1}$$

which would give the best possibility $\hat{H}(8,5) \leq 11$.

The following remark makes the resolution simpler: this system is a weighted homogeneous system with the respective weights $1,2,3,4$ for b_2, b_3, b_4, b_5. More precisely, we could follow the procedure used in the previous section in considering the case $\hat{H}(6,5)$. Here we present a slightly different approach which leads to the same result. Thus, with the above mentioned weights, the polynomials $A_{15}, \ldots, A_{23}$, which are listed in Appendix C, satisfy the following equalities:

$$\begin{aligned} A_{15}(tb_2,t^2b_3,t^3b_4,t^4b_5) &= t^{21}A_{15}(b_2,b_3,b_4,b_5),\\ A_{17}(tb_2,t^2b_3,t^3b_4,t^4b_5) &= t^{23}A_{17}(b_2,b_3,b_4,b_5),\\ A_{19}(tb_2,t^2b_3,t^3b_4,t^4b_5) &= t^{25}A_{19}(b_2,b_3,b_4,b_5),\\ A_{21}(tb_2,t^2b_3,t^3b_4,t^4b_5) &= t^{27}A_{21}(b_2,b_3,b_4,b_5),\\ A_{23}(tb_2,t^2b_3,t^3b_4,t^4b_5) &= t^{29}A_{23}(b_2,b_3,b_4,b_5). \end{aligned}$$

We can freely assume that $b_2 \neq 0$. Otherwise, all the parameters equal zero, leading to a center. Then if (b_2,b_3,b_4,b_5) is a solution of equations given in (6.5.1) with $b_2 \neq 0$, the vector

$$\left(\frac{b_3}{b_2^2}, \frac{b_4}{b_2^3}, \frac{b_5}{b_2^4}\right) \stackrel{\text{def}}{=} (B_3, B_4, B_5)$$

is a solution of the system

$$\begin{aligned} A'_{15}(B_3, B_4, B_5) &= 0, \\ A'_{17}(B_3, B_4, B_5) &= 0, \\ A'_{19}(B_3, B_4, B_5) &= 0, \\ A'_{21}(B_3, B_4, B_5) &= 0, \\ A'_{23}(B_3, B_4, B_5) &\neq 0, \end{aligned} \tag{6.5.2}$$

with $A'_i(B_3, B_4, B_5) = A_i(1, B_3, B_4, B_5)$ for $i = 15, 17, \ldots, 23$. The coefficients A'_i are listed in Appendix D. We are thus left to study four equations in the three variables B_3, B_4, B_5.

Dimension count suggests that the subsystem

$$A'_{15}(B_3, B_4, B_5) = A'_{17}(B_3, B_4, B_5) = A'_{19}(B_3, B_4, B_5) = A'_{21}(B_3, B_4, B_5) = 0$$

could already be inconsistent; however, the situation turns out to be more complex: this system defines two irreducible components of dimension 1 (one of them is defined by $2B_3 = 3B_5$ and $2B_4 = 5B_5$), as well as 3 isolated points. This information is obtained by computing a Gröbner basis of the ideal generated by A'_{15}, A'_{17}, A'_{19}, A'_{21}, followed by a primary decomposition computation.

Then, it is straightforward to check if A'_{23} identically vanishes on these sets of points: this is done by computing the normal form of A'_{23} with respect to the above Gröbner basis, and checking if the result is 0. Since the result is 0, the system (6.5.2) admits no solution; then as a conclusion, the system (6.5.1) has no solution, with the additional constraint that $b_2 \neq 0$.

Denoting A'_{ij} by the MAPLE symbol Ap_ij, we have the Maple command as follows:

```
> with(Groebner):
> gb := gbasis([Ap_15, Ap_17, Ap_19, Ap_21], plex(B_3, B_4, B_5));
> normalf(Ap_23, gb,plex(B_3, B_4, B_5));
```

It should be noted that the above 'normalf' (normal form) is different from the normal form of Hopf bifurcation given in Eq. (6.1.3), which is a normal form of differential equations associated with Hopf singularity, a well known result in the area of differential equations and nonlinear dynamical systems. The normal form used in the above code, on the other hand, is the result of reduction of a multivariate polynomial by a Gröbner basis, a standard concept in symbolic computation.

Summarizing the above results shows that it is impossible to have $\hat{H}(8,5) = 11$. Thus, the best possibility for this case is $\hat{H}(8,5) \leq 10$. Similarly, we can show that there exist infinitely many solutions for b_2, b_3, b_4 and b_5 such that $A_{15} = A_{17} = A_{19} = 0$, but $A_{21} \neq 0$. Therefore, $\hat{H}(8,5) \leq 10$. Also note that a_2, a_4 and b_2 can be chosen as any real values satisfying $a_2\, b_3 - 2\, a_4 \neq 0$. A numerical example of such a solution is given below.

$$
\begin{aligned}
a_2 &= a_4 = 0.01, \quad b_2 = 1, \\
a_3 &= 0.00666666666666666666666666666667, \\
a_5 &= 0.02100524472160155454539186112149, \\
a_6 &= -0.09468613133323534646424316541009, \\
a_7 &= -0.17831652074496883568804454053 1, \\
a_8 &= 0.00430502346606372002536893076 5, \\
a_9 &= 0.05993325303032700714151625609 8, \\
b_3 &= -4.88458910869885173306256026034 3, \\
b_4 &= -6.22300400076436425364107239654 0, \\
b_5 &= 1.67460627729493276042802449158 9,
\end{aligned}
$$

under which we obtain the following focus values:

$$
\begin{aligned}
v_0 &= 0,\\
v_1 &= -0.5 \times 10^{-52},\\
v_2 &= 0.38137218131784479500761549267786321994583353692687 \times 10^{-51},\\
v_3 &= -0.12820076597337910923435078536400682454091359844500 \times 10^{-50},\\
v_4 &= 0.47 \times 10^{-49},\\
v_5 &= 0.6888 \times 10^{-49},\\
v_6 &= -0.5127 \times 10^{-46},\\
v_7 &= 0.44337991070933151535877434801109430389266430460906 \times 10^{-44},\\
v_8 &= -0.2922151 \times 10^{-42},\\
v_9 &= 0.18090751088619566876516361673250818001667402359248 \times 10^{-40},\\
v_{10} &= -0.10955226764885435339852638063263439952921844458418.
\end{aligned}
$$

This implies that $\hat{H}(8,5) \leq 10$.

6.6 Conclusion and Discussion

In this paper, we have presented a case study for Hilbert's 16th problem. In order to find the maximum number of small limit cycles for a polynomial planar vector field, we have to symbolically compute the focus values and then solve the coupled polynomial equations. The basic ideas and procedures used here are, as usual, to eliminate variables and reduce the number of equations. Two approaches were used.

One approach was to eliminate variables successively, by means of resultant computations, until, if possible, a univariate polynomial is obtained. This polynomial can be solved numerically, and the values of the other variables are obtained by backward substitution. However, it should be noted that this approach may introduce spurious solutions, that need to be removed in a final stage. Our second approach consisted in computing Gröbner bases, which, while potentially more expensive, introduce no additional solutions, and actually proved to be manageable for the larger systems.

Let us mention that other methods are known for such tasks: Following the ideas initiated by Ritt and Wu, triangularization methods such as the algorithm of [11] implemented in the Maple package `RegularChains` [4], would also be well adapted to the kind of problems we have to deal with.

We have shown in this paper some success in the symbolic treatment of a simplified Hilbert's 16th problem – the Liénard equation. It is also shown that the bottleneck in the computation of small limit cycles are the polynomial system solving tasks.

Bibliography

[1] Bautin, N. N. On the Number of Limit Cycles Which Appear with the Variation of Coefficients from an Equilibrium Position of Focus or Center Type. *Mat. Sbornik(N.S.)*, 30(72), 181-196, 1952.

[2] Hilbert, D. Mathematical Problems. (M. Newton, Transl.), *Bull. Amer. Math.*, 8, 437-479, 1902.

[3] Kukles, I. S. Necessary and Sufficient Conditions for the Existence of Center. *Dokl. Akad. Nauk*, 42, 160-163, 1944.

[4] Lemaire, F., Moreno Maza, M. and Xie, Y. The `RegularChains` library. *In Maple 10 Maplesoft*, Canada, 2005.

[5] Li, J. Hilbert's 16th Problem and Bifurcations of Planar Polynomial Vector Fields. *Int. J. Bifurcations & Chaos*, 13, 47-106, 2003.

[6] Li, J. and Liu, Z. Bifurcation Set and Limit Cycles Forming Compound Eyes in a Perturbed Hamiltonian System. *Publications Mathematiques*, 35, 487-506, 1991.

[7] Liénard, A. Étude des Oscillations Entrenues. *Revue Génerale de l'Électricité*, 23, 946-954, 1928.

[8] Liu, Y. and Li, J. On the Singularity Values of Complex Autonomous Differential Systems. *Sci. in China (Series A)*, 3, 245-255 (in Chinese), 1989.

[9] Lloyd, N. and Pearson, J. Symmetry in Planar Dynamical Systems. *J. Symbolic Computation*, 33, 357-366, 2002.

[10] Malkin, K. E. Criteria for Center of a Differential Equation. *Volg. Matem. Sbornik*, 2, 87-91, 1964.

[11] Moreno Maza, M. On Triangular Decompositions of Algebraic Varieties. *MEGA*, Bath, UK, 2000.

[12] Rayleigh, J. The Theory of Sound. *Dover*, New York.

[13] Smale, S. Mathematical Problems for the Next Century. *The Math. Intell.*, 20, 7-15, 1999.

[14] van der Pol, B. On Relaxation-oscillations. *Philosophical Magazine*, 7, 901-912, 1926.

[15] Yu, P. Limit Cycles in 3rd-order Planar System. *International Congress of Mathematicians*, Beijing, China, August 20-28, 2002.

[16] Yu, P. Twelve Limit Cycles in a Cubic Case of the 16th Hilbert Problem. *Workshop on Bifurcation Theory and Spatio-Temporal Pattern Formation in PDE*, Toronto, Canada, December 11-13, 2003.

[17] Yu, P. and Han, M. Twelve Limit Cycles in a 3rd-order Planar System with Z_2 Symmetry. *Communication on Applied and Pure Analysis*, 3(3), 515-526, 2004.

[18] Yu, P. and Han, M. Small Limit Cycles Bifurcating from Fine Focus Points in Cubic Order Z_2-equivariant Vector Fields. *Chaos, Solitons and Fractals*, 24(1), 329-348, 2005.

[19] Yu, P. and Han, M. Twelve Limit Cycles in a Cubic Case of the 16th Hilbert Problem. *Int. J. Bifurcation & Chaos*, 15(7), 2191-2205, 2005.

[20] Yu, P. Computation of Normal Forms Via a Perturbation Technique. *J. Sound & Vib*, 211, 19-38, 1998.

Appendix A

Computation Sequences

We append the source listing of the right-hand side of the differential equation, to show the result of Maple's automatic code generation.

$f := \mathrm{proc}(N, t, Y, YP)$
$W[1] := (-4l_1^2 l_2^2 \cos(Y[1]+Y[3])^2 m_2^2 - 16 l_1^2 l_2^2 \cos(Y[1]+Y[3])^2 m_2 m_3 - 16 l_1^2 l_2^2 \cos(Y[1]+Y[3])^2 m_3^2 + l_1^2 m_1 l_2^2 m_2 + 4 l_1^2 m_1 l_2^2 m_3 + 4 l_1^2 m_1 J_2 + 4 l_1^2 m_2^2 l_2^2 + 20 l_1^2 m_2 l_2^2 m_3 + 16 l_1^2 m_2 J_2 + 16 l_1^2 m_3^2 l_2^2 + 16 l_1^2 m_3 J_2 + 4 J_1 l_2^2 m_2 + 16 J_1 l_2^2 m_3 + 16 J_1 J_2)/(l_1 l_2 \cos(Y[1]+Y[3])(m_2+2m_3));$
$W[2] := (-2l_1^2 \cos(Y[1])\cos(Y[1]+Y[3]) m_2 - 4 l_1^2 \cos(Y[1])\cos(Y[1]+Y[3]) m_3 + \cos(Y[3]) l_1^2 m_1 + 4\cos(Y[3]) l_1^2 m_2 + 4\cos(Y[3]) l_1^2 m_3 + 4\cos(Y[3]) J_1)/(l_1 \cos(Y[1]+Y[3])(m_2+2m_3));$
$W[3] := (-2l_2^2 \cos(Y[3])\cos(Y[1]+Y[3]) m_2 - 4 l_2^2 \cos(Y[3])\cos(Y[1]+Y[3]) m_3 + \cos(Y[1]) l_2^2 m_2 + 4\cos(Y[1]) l_2^2 m_3 + 4\cos(Y[1]) J_2)/(l_2 \cos(Y[1]+Y[3])(m_2+2m_3));$

$W[4] := (-\cos(Y[1])\cos(Y[3]) W[1] + W[3] W[2] \cos(Y[1]+Y[3]) m_2 + 2W[3]W[2]\cos(Y[1]+Y[3]) m_3)/(\cos(Y[1]+Y[3])(m_2+2m_3) W[1]);$
$W[5] := (-g\cos(Y[3]) + l_1 Y[2]^2 \sin(Y[1]+Y[3]))(l_1^2 m_1 + 4 l_1^2 m_2 + 4 l_1^2 m_3 + 4 J_1)/(\cos(Y[1]+Y[3]) l_1);$

$W[6] := (-l_1 \sin(Y[1]) Y[2]^2 \cos(Y[1]+Y[3]) W[1] + l_2 \sin(Y[3]) Y[4]^2 \cos(Y[1]+Y[3]) W[1] - \cos(Y[1]) W[1] g \cos(Y[3]) + \cos(Y[1]) W[1] l_1 Y[2]^2 \sin(Y[1]+Y[3]) - 2W[3]\cos(Y[1]+Y[3]) l_1 g \cos(Y[1]) m_1 - 4W[3]\cos(Y[1]+Y[3]) l_1 g \cos(Y[1]) m_2 - 4W[3]\cos(Y[1]+Y[3]) l_1 g \cos(Y[1]) m_3 - 2W[3]\cos(Y[1]+Y[3]) l_1 l_2 Y[4]^2 \sin(Y[1]+Y[3]) m_2 - 4W[3]\cos(Y[1]+Y[3]) l_1 l_2 Y[4]^2 \sin(Y[1]+Y[3]) m_3 - W[3]\cos(Y[1]+Y[3]) W[5])/(W[4] W[1] \cos(Y[1]+Y[3]));$

$W[7] := (-l_2 g \cos(Y[3])W[1]m_2 - 2l_2 g \cos(Y[3])W[1]m_3 + l_1 l_2 Y[2]^2 \sin(Y[1]+$
$Y[3])W[1]m_2 + 2l_1 l_2 Y[2]^2 \sin(Y[1]+Y[3])W[1]m_3 - 2l_2^2 m_2 l_1 g \cos(Y[1])m_1 - 4l_2^2$
$m_2^2 l_1 g \cos(Y[1]) - 20 l_2^2 m_2 l_1 g \cos(Y[1])m_3 - 2l_2^3 m_2^2 l_1 Y[4]^2 \sin(Y[1]+Y[3]) - 12 l_2^3$
$m_2 l_1 Y[4]^2 \sin(Y[1]+Y[3])m_3 - l_2^2 m_2 W[5] - l_2^2 m_2 W[2]W[6] - 8l_2^2 m_3 l_1 g \cos(Y[1])$
$m_1 - 16 l_2^2 m_3^2 l_1 g \cos(Y[1]) - 16 l_2^3 m_3^2 l_1 Y[4]^2 \sin(Y[1]+Y[3]) - 4l_2^2 m_3 W[5] - 4l_2^2 m_3$
$W[2]W[6] - 8J_2 l_1 g \cos(Y[1])m_1 - 16 J_2 l_1 g \cos(Y[1])m_2 - 16 J_2 l_1 g \cos(Y[1])m_3 - 8$
$J_2 l_1 l_2 Y[4]^2 \sin(Y[1]+Y[3])m_2 - 16 J_2 l_1 l_2 Y[4]^2 \sin(Y[1]+Y[3])m_3 - 4J_2 W[5] - 4$
$J_2 W[2]W[6] + l_2 \cos(Y[3])W[6]W[1])/(W[1]l_1 l_2 \cos(Y[1]+Y[3])(m_2+2m_3));$

$YP[1] := Y[2]; YP[2] := W[7]; YP[3] := Y[4];$
$YP[4] := 8(-l_1 g(1/2m_1 + m_2 + m_3)\cos(Y[1]) - l_1 l_2 Y[4]^2 \sin(Y[1]+Y[3])(1/2m_2+$
$m_3) - 1/4W[5] - 1/4W[2]W[6])/W[1];$

endproc

Appendix B

Coefficients A_i from Maple

The computer output of the coefficients A_i's from Maple program is listed below.

```
A2 := a2;
A3 := a3-2/3*a2*b2;
A4 := a4-1/2*a2*b3+2/3*a2*b2^2-a3*b2;
A5 := a5+7/6*a2*b3*b2-7/9*a2*b2^3-2/5*a2*b4+7/6*a3*b2^2-3/4*a3*b3-4/3*a4*b2;
A6 := a6+16/15*a2*b4*b2+80/81*a2*b2^4-20/9*a2*b3*b2^2-1/3*a2*b5+1/2*a2*b3^2+2*a3*b3*b2
      -40/27*a3*b2^3-3/5*a3*b4+16/9*a4*b2^2-a4*b3-5/3*a5*b2;
A7 := a7-11/5*a2*b4*b2^2+a2*b5*b2-143/108*a2*b2^5-33/16*a2*b3^2*b2+143/36*a2*b3*b2^3
      +9/10*a2*b3*b4-2/7*a2*b6-33/8*a3*b3*b2^2+143/72*a3*b2^4+27/32*a3*b3^2+9/5*a3*b4*b2
      -1/2*a3*b5-4/5*a4*b4-22/9*a4*b2^3+3*a4*b3*b2-5/4*a5*b3+5/2*a5*b2^2-2*a6*b2;
A8 := a8-224/81*a3*b2^5-3/7*a3*b6-7/3*a7*b2+112/27*a2*b4*b2^3+5/6*a2*b5*b3+35/6*a2*b2^2*b3^2
      -1/4*a2*b7+2/5*a2*b4^2+448/243*a2*b2^6-5/8*a2*b3^3-4*a2*b4*b3*b2-20/9*a2*b5*b2^2
      +20/21*a2*b6*b2-15/4*a3*b3^2*b2-560/81*a2*b3*b2^4+70/9*a3*b3*b2^3+3/2*a3*b3*b4
      -4*a3*b4*b2^2+5/3*a3*b5*b2-20/3*a4*b3*b2^2+8/3*a4*b4*b2+280/81*a4*b2^4+5/4*a4*b3^2
      -2/3*a4*b5-100/27*a5*b2^3-a5*b4+25/6*a5*b3*b2-3/2*a6*b3+10/3*a6*b2^2;
A9 := a9+55/32*a5*b3^2-5/6*a5*b5-143/27*a6*b2^3-6/5*a6*b4+77/18*a7*b2^2-7/4*a7*b3+11/3*a5*b4*b2
      -12155/864*a3*b3*b2^4+11/7*a3*b6*b2+11/5*a4*b3*b4-143/24*a4*b3^2*b2+715/54*a4*b3*b2^3
      +22/9*a4*b5*b2-715/72*a5*b3*b2^2+11/2*a6*b3*b2-286/45*a4*b4*b2^2+33/50*a3*b4^2
      -143/128*a3*b3^3+46189/11664*a3*b2^6-3/8*a3*b7-2431/486*a4*b2^5-4/7*a4*b6
      +3575/648*a5*b2^4-8/3*a8*b2+143/12*a2*b2^2*b3*b4-143/36*a2*b5*b2*b3-143/20*a3*b4*b3*b2
      -143/75*a2*b4^2*b2+11/14*a2*b3*b6+46189/3888*a2*b2^5*b3-12155/864*a2*b2^3*b3^2
      +715/162*a2*b5*b2^3-2431/324*a2*b4*b2^4-143/63*a2*b6*b2^2+11/12*a2*b7*b2
      -46189/17496*a2*b2^7-2/9*a2*b8+11/15*a2*b5*b4-143/80*a2*b3^2*b4+715/192*a2*b3^3*b2
      +11/8*a3*b5*b3+715/64*a3*b2^2*b3^2+143/18*a3*b4*b2^3-143/36*a3*b5*b2^2;
A10:= a10+2*a4*b5*b3+6/5*a3*b5*b4-3*a9*b2-4928/243*a2*b2^6*b3-14*a2*b3^3*b2^2+3/4*a2*b7*b3
      -224/27*a2*b5*b2^4+128/27*a2*b6*b2^3-7/3*a2*b7*b2^2+1792/135*a2*b4*b2^5+8/9*a2*b8*b2
      +280/9*a2*b2^4*b3^2-7/4*a2*b5*b3^2+24/35*a2*b4*b6-42/25*a2*b3*b4^2+448/75*a2*b4^2*b2^2
      -84/25*a3*b4^2*b2+9/7*a3*b3*b6+224/9*a3*b2^5*b3-28*a3*b2^3*b3^2+224/27*a3*b5*b2^3
      -224/15*a3*b4*b2^4-4*a3*b6*b2^2+3/2*a3*b7*b2-63/20*a3*b3^2*b4+7*a3*b3^3*b2
      +56/3*a4*b2^2*b3^2+1792/135*a4*b4*b2^3-56/9*a4*b5*b2^2-224/9*a4*b3*b2^4+16/7*a4*b6*b2
      +3*a5*b3*b4-35/4*a5*b3^2*b2+560/27*a5*b3*b2^3-28/3*a5*b4*b2^2+10/3*a5*b5*b2-14*a6*b3*b2^2
      +24/5*a6*b4*b2+7*a7*b3*b2+56/5*a2*b4*b2*b3^2-56/15*a2*b4*b2*b5-4*a2*b6*b3*b2
      +112/9*a2*b5*b3*b2^2+112/5*a3*b2^2*b3*b4-7*a3*b5*b2*b3-1408/243*a3*b2^7-1/3*a3*b8
```

```
      +1/3*a2*b5^2+7/8*a2*b3^4+2816/729*a2*b2^8-1/5*a2*b9-448/15*a2*b4*b2^3*b3+24/25*a4*b4^2
      -7/4*a4*b3^3+1792/243*a4*b2^6-1/2*a4*b7-224/27*a5*b2^5-5/7*a5*b6
      +224/27*a6*b2^4+9/4*a6*b3^2-a6*b5-196/27*a7*b2^3-7/5*a7*b4
      +16/3*a8*b2^2-2*a8*b3-56/5*a4*b4*b3*b2;
A11:= a11+26/15*a4*b5*b4+1105/216*a2*b7*b2^3-20995/768*a3*b3^3*b2^2-195/16*a6*b3^2*b2
      +1105/84*a2*b6*b2^2*b3-26/7*a2*b6*b2*b4-4199/96*a2*b3^2*b2^2*b4+1105/96*a2*b3^2*b2*b5
      +29393/432*a2*b3*b2^4*b4-20995/648*a2*b3*b2^3*b5-65/16*a2*b7*b2*b3+221/18*a2*b5*b4*b2^2
      -13/4*a2*b5*b3*b4+221/20*a2*b3*b2*b4^2-4199/72*a3*b4*b2^3*b3+663/32*a3*b4*b2*b3^2
      -13/2*a3*b4*b2*b5-195/28*a3*b6*b3*b2+1105/48*a3*b5*b3*b2^2+221/6*a4*b2^2*b3*b4
      -65/6*a4*b5*b2*b3-65/4*a5*b4*b3*b2-10/3*a10*b2-65/27*a2*b8*b2^2-676039/10368*a2*b3^2*b2^5
      -20995/3072*a2*b3^4*b2-676039/15552*a3*b2^6*b3-96577/8748*a4*b2^7-4/9*a4*b8+13/10*a5*b4^2
      -325/128*a5*b3^3+146965/11664*a5*b2^6-5/8*a5*b7-4199/324*a6*b2^5-6/7*a6*b6+91/32*a7*b3^2
      +7735/648*a7*b2^4-7/6*a7*b5-8/5*a8*b4-260/27*a8*b2^3-9/4*a9*b3+13/2*a9*b2^2-26/5*a4*b4^2*b2
      +39/10*a6*b3*b4-13/25*a2*b4^3-2414425/419904*a2*b2^9-2/11*a2*b10+13/24*a3*b5^2
      +3315/2048*a3*b3^4+2414425/279936*a3*b2^8-3/10*a3*b9+146965/3456*a2*b3^3*b2^3
      +2414425/69984*a2*b3*b2^7+13/20*a2*b7*b4-4199/270*a2*b4^2*b2^3-65/36*a2*b5^2*b2
      +13/21*a2*b5*b6-20995/2268*a2*b6*b2^4+29393/1944*a2*b5*b2^5+13/15*a2*b9*b2
      +221/64*a2*b4*b3^3-195/112*a2*b6*b3^2+13/18*a2*b8*b3-676039/29160*a2*b2^6*b4+39/32*a3*b7*b3
      -20995/1296*a3*b5*b2^4+1105/126*a3*b6*b2^3-65/16*a3*b7*b2^2+29393/1080*a3*b4*b2^5
      +13/9*a3*b8*b2+146965/2304*a3*b2^4*b3^2-195/64*a3*b5*b3^2+39/35*a3*b4*b6-117/40*a3*b3*b4^2
      +221/20*a3*b4^2*b2^2+13/7*a4*b3*b6+29393/648*a4*b2^5*b3-20995/432*a4*b2^3*b3^2
      +1105/81*a4*b5*b2^3-4199/162*a4*b4*b2^4-130/21*a4*b6*b2^2+13/6*a4*b7*b2-39/8*a4*b3^2*b4
      +1105/96*a4*b3^3*b2+65/24*a5*b5*b3+5525/192*a5*b2^2*b3^2+1105/54*a5*b4*b2^3
      -325/36*a5*b5*b2^2-104975/2592*a5*b3*b2^4+65/21*a5*b6*b2+1105/36*a6*b3*b2^3-13*a6*b4*b2^2
      +13/3*a6*b5*b2-455/24*a7*b3*b2^2+91/15*a7*b4*b2+26/3*a8*b3*b2;
A12:= a12+49/10*a7*b3*b4+28/45*a2*b8*b4+7/2*a6*b5*b3+448/125*a2*b2*b4^3+7/3*a5*b5*b4
      -14080/729*a5*b2^7-5/9*a5*b8+42/25*a6*b4^2-7/2*a6*b3^3+4928/243*a6*b2^6-3/4*a6*b7
      -1568/81*a7*b2^5-a7*b6-49/3*a7*b3^2*b2-4/3*a8*b5+448/27*a8*b2^4+7/2*a8*b3^2-9/5*a9*b4
      -112/9*a9*b2^3+70/9*a10*b2^2-5/2*a10*b3-11/3*a11*b2+1232/9*a3*b3*b2^4*b4
      -560/9*a3*b3*b2^3*b5-7*a3*b7*b2*b3+112/5*a3*b5*b4*b2^2-28/5*a3*b5*b3*b4
      +504/25*a3*b3*b2*b4^2-896/9*a4*b4*b2^3*b3+168/5*a4*b4*b2*b3^2-448/45*a4*b4*b2*b5
      -32/3*a4*b6*b3*b2+112/3*a4*b5*b3*b2^2+56*a5*b2^2*b3*b4-140/9*a5*b5*b2*b3-112/5*a6*b4*b3*b2
      +112/5*a2*b3*b4*b5*b2-28*a2*b3^3*b4*b2-140/3*a2*b3^2*b2^2*b5-224/5*a2*b3*b4^2*b2^2
      +6160/81*a2*b3*b2^4*b5-896/27*a2*b4*b2^3*b5-112/27*a2*b8*b2*b3+14*a2*b7*b2^2*b3
      -56/15*a2*b7*b4*b2-19712/135*a2*b2^5*b3*b4-16/5*a2*b6*b3*b4-320/9*a2*b6*b3*b2^3
      +12*a2*b6*b3^2*b2+64/5*a2*b6*b4*b2^2-32/9*a2*b6*b5*b2-32/5*a3*b6*b2*b4-84*a3*b3^2*b2^2*b4
      +21*a3*b3^2*b2*b5+7/9*a4*b5^2+21/8*a4*b3^4+36608/2187*a4*b2^8-2/5*a4*b9
      -256256/19683*a3*b2^9-3/11*a3*b10+24*a3*b6*b2^2*b3+28/3*a3*b7*b2^3-112/27*a3*b8*b2^2
      -1232/9*a3*b3^2*b2^5-105/8*a3*b3^4*b2+770/9*a3*b3^3*b2^3+18304/243*a3*b3*b2^7
      +21/20*a3*b7*b4-448/15*a3*b4^2*b2^3-28/9*a3*b5^2*b2+a3*b5*b6-160/9*a3*b6*b2^4
      +2464/81*a3*b5*b2^5+7/5*a3*b9*b2+63/10*a3*b4*b3^3-3*a3*b6*b3^2+7/6*a3*b8*b3
      -19712/405*a3*b2^6*b4-19712/243*a4*b2^6*b3-140/3*a4*b3^3*b2^2+7/4*a4*b7*b3
      -2240/81*a4*b5*b2^4+128/9*a4*b6*b2^3-56/9*a4*b7*b2^2+19712/405*a4*b4*b2^5+56/27*a4*b8*b2
      +3080/27*a4*b2^4*b3^2-14/3*a4*b5*b3^2+8/5*a4*b4*b6-112/25*a4*b3*b4^2+448/25*a4*b4^2*b2^2
      -112/15*a5*b4^2*b2+5/2*a5*b3*b6+6160/81*a5*b2^5*b3-700/9*a5*b2^3*b3^2+560/27*a5*b5*b2^3
      -1120/27*a5*b4*b2^4-80/9*a5*b6*b2^2+35/12*a5*b7*b2-7*a5*b3^2*b4+35/2*a5*b3^3*b2
      +42*a6*b2^2*b3^2+448/15*a6*b4*b2^3-112/9*a6*b5*b2^2-560/9*a6*b3*b2^4+4*a6*b6*b2
      +392/9*a7*b3*b2^3-784/45*a7*b4*b2^2+49/9*a7*b5*b2+112/15*a8*b4*b2-224/9*a8*b3*b2^2
      +21/2*a9*b3*b2-112/125*a3*b4^3+1232/9*a2*b3^2*b4*b2^3+7/12*a2*b7*b5-112/45*a2*b9*b2^2
      -14/9*a2*b3*b5^2+7/2*a2*b5*b3^3-112/75*a2*b5*b4^2-7/4*a2*b7*b3^2+32032/243*a2*b2^6*b3^2
```

```
      +28/33*a2*b10*b2-280/27*a2*b7*b2^4+1408/81*a2*b6*b2^5+448/81*a2*b8*b2^3
      -19712/729*a2*b5*b2^6+385/12*a2*b3^4*b2^2-3080/27*a2*b3^3*b2^4+126/25*a2*b3^2*b4^2
      -128128/2187*a2*b2^8*b3+146432/3645*a2*b2^7*b4+4928/135*a2*b4^2*b2^4+56/9*a2*b5^2*b2^2
      +7/10*a2*b9*b3+2/7*a2*b6^2-21/16*a2*b3^5+512512/59049*a2*b2^10-1/6*a2*b11;
A13:= ... (  80 lines)
A15:= ... ( 165 lines)
A17:= ... ( 145 lines)
A19:= ... ( 600 lines)
A21:= ... (1080 lines)
A23:= ... (1880 lines)
```

Appendix C

Coefficients A_i for Case $\hat{H}(8,5)$

The coefficients A_i, $i = 15,\ 17,\ 19,\ 21\,23$ used for the case $\hat{H}(8,5)$ are listed below.

```
A15 := -68/50625*(-2*a4+a2*b3)
       /(39600*b2^6*b3^2-6561*b5*b4^2+40500*b3^3*b2^4-97200*b3^2*b4*b2^3+87480*b3*b4^2*b2^2
         +55350*b3*b2^4*b5-50220*b4*b2^3*b5+3240*b2^5*b3*b4+11040*b2^7*b4-18225*b5^2*b2^2
         -6480*b4^2*b2^4-26244*b2*b4^3-78600*b5*b2^6+34000*b2^8*b3+10935*b3*b4*b5*b2)
       *(885735*b4^5*b5*b3+708588*b4^7-19000000*b2^9*b5^3-1728000*b2^9*b4^4+1216000*b2^12*b4^3
         +18225000*b2^6*b5*b4*b3^4-19950000*b2^10*b5*b4*b3^2-43302600*b2^3*b5*b3*b4^4
         -19683000*b2^2*b5^2*b3*b4^3+14762250*b2^2*b5*b3^3*b4^3-11390625*b2^5*b5^4
         -3149280*b4^6*b2^3+1360800*b2^6*b4^5+41229000*b2^6*b5*b3*b4^3+1140000*b2^11*b4^2*b3^2
         -5670000*b2^8*b4^3*b3^2+2736000*b2^10*b4^3*b3-16402500*b2^3*b4^2*b5*b3^4
         -14580000*b2^5*b5*b4^4-26244000*b2^5*b3^2*b4^4-1944000*b2^7*b4^4*b3
         +25650000*b2^8*b5^2*b4*b3+17812500*b2^9*b5*b3^4+32062500*b2^9*b5^2*b3^2
         +37462500*b2^6*b5^3*b4+2375000*b2^11*b5*b3^3+7873200*b4^5*b2^2*b5
         +4428675*b4^4*b2*b5^2+14580000*b2^6*b3^3*b4^3+26973000*b2^4*b5^2*b4^3
         +15746400*b2^4*b3*b4^5+34171875*b2^5*b3^2*b5^3-11390625*b2^5*b3^4*b5^2
         +6834375*b2^4*b5*b4*b3^5-34171875*b2^4*b5^3*b3*b4-6840000*b2^9*b5*b4^2*b3
         -45562500*b2^8*b5*b4*b3^3+107325000*b2^6*b5^2*b4*b3^2-69255000*b2^5*b4^2*b5*b3^3
         +24603750*b2^3*b5^2*b3^2*b4^2-5904900*b5*b2*b3^2*b4^4+5467500*b2^4*b3^4*b4^3
         +11809800*b2^2*b3^2*b4^5-13122000*b2^3*b3^3*b4^4+8550000*b2^9*b4^2*b3^3
         +22800000*b2^10*b5^2*b4-9120000*b2^11*b5*b4^2-30375000*b2^7*b5^2*b3^3
         +9072000*b2^8*b5*b4^3-48093750*b2^7*b5^3*b3+8201250*b2^3*b5^3*b4^2
         -26325000*b2^7*b5^2*b4^2-4723920*b3*b2*b4^6+85293000*b2^4*b5*b3^2*b4^3
         -98415000*b2^5*b3*b5^2*b4^2-7290000*b2^7*b5*b4^2*b3^2):
A17 := -19/151875*(-2*a4+a2*b3)
       /(39600*b2^6*b3^2-6561*b5*b4^2+40500*b3^3*b2^4-97200*b3^2*b4*b2^3+87480*b3*b4^2*b2^2
         +55350*b3*b2^4*b5-50220*b4*b2^3*b5+3240*b2^5*b3*b4+11040*b2^7*b4-18225*b5^2*b2^2
         -6480*b4^2*b2^4-26244*b2*b4^3-78600*b5*b2^6+34000*b2^8*b3+10935*b3*b4*b5*b2)
       *(73600000*b4^3*b2^14+10628820*b5^2*b4^5-79488000*b4^4*b2^11-1150000000*b5^3*b2^11
         -191056320*b4^6*b2^5-1207500000*b4*b5*b2^12*b3^2+414000000*b5*b2^11*b4^2*b3
         -1746866250*b5^2*b2^3*b4^2*b3^3+82012500*b4*b5^2*b2^4*b3^4-2727189000*b5^2*b2^4*b4^3*b3
         +2234840625*b4*b5^3*b2^4*b3^2-458810730*b3*b5*b2^2*b4^5+2480058000*b5*b2^3*b4^4*b3^2
         +2740311000*b5*b2^6*b4^3*b3^2-6834375000*b4*b5^2*b2^6*b3^3+1417176000*b5^2*b4^3*b2^2*b3^2
         +1033357500*b5*b2^3*b4^2*b3^5-4107915000*b5^2*b2^7*b4^2*b3+3674160000*b5*b2^5*b4^2*b3^4
```

```
        -418263750*b5^3*b2^3*b4^2*b3-1267875000*b4*b5*b2^10*b3^3+372008700*b5*b4^4*b2*b3^3
        +1551474000*b5*b2^8*b4^3*b3-349865325*b3*b2*b4^4*b5^2-430565625*b4*b5*b2^4*b3^6
        +1435218750*b3^4*b2^7*b5^2+297606960*b4^6*b2*b3^2-246037500*b4*b5^4*b2^4
        +990984375*b3*b2^5*b5^4-55801305*b5*b4^5*b3^2+2059425000*b4*b5^3*b2^8
        +7452202500*b5^2*b2^5*b4^2*b3^2-4344840000*b5*b2^7*b4^2*b3^3-4753444500*b5*b2^4*b4^3*b3^3
        +4668637500*b4*b5*b2^8*b3^4-842906250*b5^4*b2^7-652050000*b4^2*b2^9*b3^4
        +69000000*b4^2*b2^13*b3^2+1102248000*b4^4*b2^5*b3^3+99792000*b4^4*b2^9*b3
        -496011600*b4^5*b2^4*b3^2+414000000*b4^2*b2^11*b3^3-44641044*b3*b4^7+29760696*b2^2*b4^7
        +1615950000*b5^2*b2^6*b4^3-1516320000*b4^4*b2^7*b3^2
        -1196370000*b5^2*b2^9*b4^2-783432000*b5*b2^7*b4^4+1387530000*b4^3*b2^8*b3^3
        +869551200*b4^5*b2^6*b3+477640800*b5*b2^4*b4^5+431932500*b5^3*b2^5*b4^2
        +370656000*b5*b2^10*b4^3-552000000*b5*b2^13*b4^2-1358437500*b3^5*b2^9*b5
        +717609375*b3^5*b2^5*b5^2+1940625000*b3^2*b2^11*b5^2-2289515625*b3^3*b2^5*b5^3
        -3804187500*b3^3*b2^9*b5^2-861131250*b4*b5*b2^6*b3^5-1185937500*b3*b2^9*b5^3
        +4586625000*b3^2*b2^7*b5^3+143750000*b3^3*b2^13*b5+862500000*b3^4*b2^11*b5
        +50932800*b4^5*b2^8+1380000000*b4*b5^2*b2^12-744017400*b4^5*b2^2*b3^3
        -3910781250*b4*b5^3*b2^6*b3+55200000*b4^3*b2^12*b3+4588650000*b4*b5^2*b2^8*b3^2
        -517500000*b4*b5^2*b2^10*b3-344452500*b4^3*b2^4*b3^5-688905000*b4^3*b2^6*b3^4
        -583740000*b4^3*b2^10*b3^2+826686000*b4^4*b2^3*b3^4-59049000*b5^3*b4^3*b2^2
        +243741150*b5^2*b2^3*b4^4-930021750*b5*b2^2*b4^3*b3^4-1708484400*b5*b2^5*b3*b4^4
        +185220000*b5*b2^9*b4^2*b3^2):
A19 := -23/3645000*(-2*a4+a2*b3)
      /(39600*b2^6*b3^2-6561*b5*b4^2+40500*b3^3*b2^4-97200*b3^2*b4*b2^3+87480*b3*b4^2*b2^2
        +55350*b3*b2^4*b5-50220*b4*b2^3*b5+3240*b2^5*b3*b4+11040*b2^7*b4-18225*b5^2*b2^2
        -6480*b4^2*b2^4-26244*b2*b4^3-78600*b5*b2^6+34000*b2^8*b3+10935*b3*b4*b5*b2)
      *(3348078300*b3^2*b4^7-714256704*b4^7*b5+4185097875*b3^3*b4^5*b5+5753600000*b2^16*b4^3
        -89900000000*b2^13*b5^3-64233000000*b2^11*b5*b3^2*b4^2+24273000000*b2^13*b3^3*b4^2
        -50895000000*b2^12*b3^2*b4^3+24377760000*b2^11*b3*b4^4+11237500000*b2^15*b5*b3^3
        -43152000000*b2^15*b5*b4^2+491062500000*b2^10*b5^2*b3^2*b4+5394000000*b2^15*b3^2*b4^2
        -4315200000*b2^14*b3*b4^3-2232052200*b4^5*b5^2*b3-921456000*b2^10*b4^5
        +569227500000*b2^10*b5*b4*b3^4+64360980000*b2^10*b5*b3*b4^3+107880000000*b2^14*b4*b5^2
        -94395000000*b2^14*b4*b5*b3^2+32298750000*b2^12*b5*b4*b3^3-202275000000*b2^12*b5^2*b3*b4
        -2338560000*b2^13*b4^4+97092000000*b2^13*b5*b3*b4^2+151706250000*b2^13*b5^2*b3^2
        -166017600000*b2^9*b5^2*b3*b4^2-499632300000*b2^9*b5*b3^3*b4^2+50568750000*b2^13*b3^4*b5
        -1051200000*b2^12*b5*b4^3+799129800000*b2^7*b5*b3^4*b4^2-50864062500*b2^9*b5^4
        -198196875000*b2^11*b5*b3^5+86226120000*b2^8*b5*b3^2*b4^3-996360750000*b2^8*b5^2*b3^3*b4
        -498180375000*b2^8*b5*b4*b3^5-584369437500*b2^8*b5^3*b3*b4-872276202000*b2^6*b5*b3^3*b4^3
        +119601562500*b2^9*b3^6*b5-37289808000*b2^9*b5*b4^4+218472187500*b2^7*b3*b5^4
        -148541040000*b2^9*b3^2*b4^4+62845632000*b2^8*b3*b4^5-191872800000*b2^8*b3^4*b4^3
        +64831428000*b2^8*b5^2*b4^3-13648500000*b2^11*b5^2*b4^2-482456250000*b2^11*b5^2*b3^3
        +42140625000*b2^11*b3*b5^3-95134500000*b2^11*b3^4*b4^2+25259850000*b2^6*b3^5*b4^3
        -12542532480*b2^7*b4^6+439495875000*b2^6*b5^3*b3^2*b4+6833937600*b2^6*b5*b4^5
        -188925307200*b2^6*b3^2*b4^5+31574812500*b2^6*b5*b4*b3^6-599867856000*b2^6*b5^2*b3*b4^3
        -89047350000*b2^6*b4*b5^4+11481750000*b2^5*b5^5+486443475000*b2^6*b5^2*b3^4*b4
        +57408750000*b2^9*b3^5*b4^2+82406250000*b2^10*b4*b5^3+189880200000*b2^10*b3^3*b4^3
        +417453750000*b2^9*b3^4*b5^2+411370312500*b2^9*b5^3*b3^2-62001450000*b2^3*b3^5*b4^4
        +25833937500*b2^4*b3^6*b4^3+20105003520*b2^3*b5*b4^6+32292421875*b2^4*b5*b4*b3^7
        +65445975000*b2^4*b5^4*b3*b4-8266860000*b2^3*b4^2*b5^4+1428513408*b2*b4^8
        +1204052850000*b2^7*b5^2*b3^2*b4^2+46625090400*b2^3*b3^2*b4^6+190166484375*b2^5*b3^4*b5^3
        +55597389120*b2^5*b3*b4^6+16196047200*b2^7*b4^4*b5*b3-53820703125*b2^5*b3^6*b5^2
```

```
          -8266860000*b2^5*b3^4*b4^4-122711203125*b2^5*b3^2*b5^4+88276286700*b2^5*b5^2*b4^4
          -4993183440*b2^4*b4^7-503761781250*b2^7*b5^3*b3^3+131281965000*b2^7*b5^3*b4^2
          +278011440000*b2^7*b3^3*b4^4-52624687500*b2^7*b3^5*b5^2-22320522000*b2*b3^3*b4^6
          -190757794500*b2^4*b4^5*b5*b3+55801305000*b2^2*b4^5*b3^4+2480058000*b2^2*b5^2*b4^5
          -15177954960*b2^2*b3*b4^7-46294416000*b2^4*b3^3*b4^5-4464104400*b2*b4^4*b5^3
          +298295865000*b2^4*b5^2*b3^2*b4^3+246627990000*b2^4*b5*b3^4*b4^3
          -17222625000*b2^4*b5^2*b4*b3^5-171795684375*b2^4*b5^3*b3^3*b4-16717428000*b2^4*b5^3*b4^3
          +157587018750*b2^3*b5^2*b4^2*b3^4-48567802500*b2^3*b3*b4^4*b5^2
          -37889775000*b2^5*b3*b5^3*b4^2-182559825000*b2^5*b3^5*b5*b4^2
          +589868017200*b2^5*b5*b4^4*b3^2-646652160000*b2^5*b5^2*b3^3*b4^2
          +27280638000*b2^2*b3*b5^3*b4^3+69751631250*b2^2*b5*b4^3*b3^5
          +6324147900*b2^2*b5*b4^5*b3^2-27900652500*b2*b3^4*b4^4*b5
          +35805837375*b2*b3^2*b4^4*b5^2-77501812500*b2^3*b5*b4^2*b3^6
          -114496011000*b2^3*b5*b4^4*b3^3+1550036250*b2^3*b5^3*b4^2*b3^2
          -130203045000*b2^2*b4^3*b3^3*b5^2+6547353120*b2*b4^6*b5*b3):
A21 := -23/1968300000*(-2*a4+a2*b3)
      /(39600*b2^6*b3^2-6561*b5*b4^2+40500*b3^3*b2^4-97200*b3^2*b4*b2^3+87480*b3*b4^2*b2^2
          +55350*b3*b2^4*b5-50220*b4*b2^3*b5+3240*b2^5*b3*b4+11040*b2^7*b4-18225*b5^2*b2^2
          -6480*b4^2*b2^4-26244*b2*b4^3-78600*b5*b2^6+34000*b2^8*b3+10935*b3*b4*b5*b2)
      *(-358099892578125*b2^5*b3^5*b5^3+321886248046875*b2^5*b5^4*b3^3+297538935552*b4^9
          -174630750000000*b2^16*b4*b5*b3^2+299367000000000*b2^15*b5*b3*b4^2
          +675851343750000*b2^11*b3^6*b5-34045466400000*b2^11*b5*b4^4
          +210309506156250O*b2^8*b5^3*b3^2*b4+1065982072500000*b2^8*b5*b4*b3^6
          +2922925500000000*b2^8*b5^2*b3^4*b4-1759512577500000*b2^8*b5^2*b3*b4^3
          -2564468545500000*b2^8*b5*b3^3*b4^3-832037315625000*b2^4*b4^3*b3^3*b5^2
          +1586941875000O0*b2^4*b3*b5^3*b4^3-124058258437500*b2^4*b5*b4^3*b3^5
          +52313723437500*b2^4*b4*b3^6*b5^2-221156957812500*b2^4*b4*b5^4*b3^2
          -54182070703125*b2^4*b4*b5*b3^8+288901845703125*b2^4*b4*b3^4*b5^3
          +705069417712500*b2^4*b5*b4^5*b3^2+76892205386250*b2^2*b3^3*b4^5*b5
          -402070108500000*b2^11*b5^2*b3*b4^2-1001406888000000*b2^11*b5*b3^3*b4^2
          +3294032062500000*b2^9*b5*b3^4*b4^2+3152445682500000*b2^9*b5^2*b3^2*b4^2
          +238092129000000*b2^9*b4^4*b5*b3+86451891300000*b2^10*b5*b3^2*b4^3
          -3242183760000000*b2^10*b5^2*b3^3*b4-2435522478750000*b2^10*b5*b4*b3^5
          -1315247675625000*b2^10*b5^3*b3*b4-79831200000000*b2^17*b5*b4^2
          +20789375000000*b2^17*b5*b3^3+9978900000000*b2^17*b3^2*b4^2
          +199578000000000*b2^16*b4*b5^2+29936700000000*b2^15*b3^3*b4^2
          +62368125000000*b2^15*b3^4*b5+280656562500000*b2^15*b5^2*b3^2
          +324408645000000*b2^11*b3^5*b4^2-23949360000000*b2^16*b3*b4^3
          +1765405631250000*b2^11*b3^4*b5^2-9751920480000*b2^12*b4^5-5617597090500*b3^3*b4^7
          -573956280000*b5^3*b4^5-30720515625000*b2^11*b5^4-117033272718750*b2^2*b3^6*b5*b4^3
          -111801900375000*b2^2*b3^2*b5^3*b4^3+269042006250000*b2^2*b5^2*b4^3*b3^4
          +46813309087500*b2*b5*b4^4*b3^5-448819147125000*b2^7*b3*b5^3*b4^2
          -217691519625000O*b2^7*b3^5*b5*b4^2+1534199308200000*b2^7*b5*b4^4*b3^2
          -673575750000000*b2^14*b5^2*b3*b4+317066062500000*b2^14*b5*b4*b3^3
          -90753966993600*b2^3*b4^6*b5*b3+130036969687500*b2^3*b3^7*b5*b4^2
          +238819620881250*b2^3*b3^2*b5^2*b4^4+30557857500000*b2^3*b3*b5^4*b4^2
          +80131338281250*b2^3*b3^3*b5^3*b4^2-320608390781250*b2^3*b5^2*b4^2*b3^5
          +1345708257187500*b2^5*b5^2*b4^2*b3^4-460619843737500*b2^5*b3*b4^4*b5^2
          -2018094053400000*b2^5*b5*b4^4*b3^3+80911892250000*b2^5*b5*b4^2*b3^6
          +867282187500*b2^5*b5^3*b4^2*b3^2+512008593750000*b2^11*b5^3*b3^2
```

```
      +3471287581440*b5*b4^7*b3+6779858557500*b5^2*b4^5*b3^2-7021996363125*b5*b4^5*b3^4
      -38139394806000*b2^2*b4^5*b5^2*b3+68808487500000*b2^7*b5^5-28489063975200*b2^6*b4^7
      -166315000000000*b2^15*b5^3+4660416000000*b2^15*b4^4+10644160000000*b2^18*b4^3
      -20599440480000*b2^9*b4^6+3801886398720*b2^3*b4^8-32130072554400*b2*b3^2*b5*b4^6
      +23197399650000*b2*b3*b5^3*b4^4-80309038865625*b2*b5^2*b4^4*b3^3
      -405805723200000*b2^11*b3^2*b4^4+639783182812500*b2^9*b3*b5^4
      +483956795250000*b2^9*b5^3*b4^2+1043815734000000*b2^9*b3^3*b4^4
      -158813056250000*b2^9*b5^3*b3^3-110593856250000*b2^9*b3^6*b4^2
      -230403867187500*b2^9*b3^7*b5-852048365625000*b2^9*b3^5*b5^2
      +138835483920000*b2^10*b3*b4^5-874920285000000*b2^10*b3^4*b4^3
      +9105355800000*b2^10*b5^2*b4^3-57995565840000*b2^8*b5*b4^5
      +454647775500000*b2^8*b3^5*b4^3-679890186000000*b2^8*b3^2*b4^5
      -373919591250000*b2^8*b4*b5^4+297739820250000*b2^4*b4^5*b3^4
      +13286025000000*b2^4*b4*b5^5+11864951766000*b2^4*b5^2*b4^5-43345656562500*b2^4*b4^3*b3^7
      -6916173174000*b2^4*b3*b4^7-93626618175000*b2^2*b3^5*b4^5+413248521600*b2^2*b4^7*b5
      +70123108509000*b2^2*b3^2*b4^7+3188646000000*b2^2*b5^4*b4^3+37450647270000*b2*b3^4*b4^6
      -9174117179520*b2*b4^8*b3+3099363912000*b2*b5^2*b4^6+226039572000000*b2^7*b3*b4^6
      +1255252570312500*b2^7*b3^4*b5^3-693530505000000*b2^7*b3^4*b4^4
      -36121380468750*b2^7*b3^6*b5^2-907442683593750*b2^7*b3^2*b5^4
      +351782941275000*b2^7*b5^2*b4^4-73401552000000*b2^14*b5*b4^3
      -79372710000000*b2^14*b3^2*b4^3+104029575750000*b2^3*b3^6*b4^4
      -215233605000000*b2^3*b4^6*b3^3-254914533000000*b2^3*b5^3*b4^4
      +90077974041600*b2^5*b5*b4^6-64584843750000*b2^5*b3*b5^5
      -166447321200000*b2^5*b3^5*b4^4-73102662000000*b2^5*b4^2*b5^4
      +903034511718750*b2^5*b3^7*b5^2-104077405440000*b2^5*b3^2*b4^6
      +57672648000000*b2^13*b3*b4^4-1265988656250000*b2^13*b5^2*b3^3
      +327432656250000*b2^13*b3*b5^3-240302700000000*b2^13*b3^4*b4^2
      +173673315000000*b2^13*b5^2*b4^2-500630625000000*b2^13*b5*b3^5
      -69522324300000*b2^6*b5^3*b4^3+451405985400000*b2^6*b3^3*b4^5
      +17338262625000*b2^6*b3^6*b4^3+527573655000000*b2^12*b3^3*b4^3
      -64020037500000*b2^12*b4*b5^3-4127746533750000*b2^7*b5^2*b3^3*b4^2
      -14348907000000*b2^3*b3^4*b5*b4^4-475508070000000*b2^13*b5*b3^2*b4^2
      -593016411465000*b2^6*b4^5*b5*b3+2341484759250000*b2^6*b5^2*b3^2*b4^3
      +2794095344250000*b2^6*b5*b3^4*b4^3-761178515625000*b2^6*b5^2*b4*b3^5
      -1117394683593750*b2^6*b5^3*b3^3*b4+21672828281250*b2^6*b5*b4*b3^7
      +608212901250000*b2^6*b5^4*b3*b4+170265385800000*b2^12*b5*b3*b4^3
      +165845272500000*b2^12*b5^2*b3^2*b4+1140072468750000*b2^12*b5*b4*b3^4):
A23 := -29/94478400000*(-2*a4+a2*b3)
      /(39600*b2^6*b3^2-6561*b5*b4^2+40500*b3^3*b2^4-97200*b3^2*b4*b2^3+87480*b3*b4^2*b2^2
        +55350*b3*b2^4*b5-50220*b4*b2^3*b5+3240*b2^5*b3*b4+11040*b2^7*b4-18225*b5^2*b2^2
        -6480*b4^2*b2^4-26244*b2*b4^3-78600*b5*b2^6+34000*b2^8*b3+10935*b3*b4*b5*b2)
      *(202095765433800000*b2^9*b5*b4^4*b3^2+42568902396900000*b2^5*b3^6*b4^4
        +16441455937500000*b2^5*b3^2*b5^5-49813252013671875*b2^5*b3^4*b5^4
        -10394685430114500O*b2^4*b3^3*b4^5*b5-2915123946120000*b2^4*b4^5*b5^2*b3
        -9785954574000000*b2^6*b4^3*b3^7+15370472650896000*b2^6*b3*b4^7
        -4853186400000000*b2^18*b3*b4^3-19038129807600000*b2^5*b3^3*b4^6
        -45086816710800000O*b2^6*b4^3*b3^3*b5^2-24627985677900000*b2^6*b4^5*b3^4
        +11656272600000000*b2^6*b4*b5^5+17208612278640000*b2^6*b5^2*b4^5
        +1787778432000000*b2^17*b4^4+67442158725120*b4^7*b5^2-2998796904000000*b2^11*b4^6
        +594496740370500*b3^4*b4^7-110684484025344*b4^9*b3+153729697500000000*b2^9*b5^5
```

```
+319755176073216*b2^2*b4^9+1294183040000000*b2^20*b4^3-797161500000000*b2^5*b5^6
-762948603878400*b2^5*b4^8-5883039115416000*b2^8*b4^7-1489930977600000*b2^14*b4^5
+6914797031250000*b2^13*b5^4-20221610000000000*b2^17*b5^3
+704824052715000000*b2^11*b5*b3^4*b4^2-714837292663680*b3^2*b4^7*b5
+247949112960000*b3*b5^3*b4^5+7431209254631125*b3^5*b4^5*b5
-1080903164310000*b4^5*b5^2*b3^3-19588589752387500O*b2^5*b3^5*b5^2*b4^2
+102042132556275000*b2^5*b3^2*b5^2*b4^4+50958457200000000*b2^17*b5*b3*b4^2
+2090706920478720*b2^2*b5*b4^7*b3+105147497250000000*b2^14*b5*b4*b3^4
-1329694274700000O0*b2^13*b5^2*b3*b4^2-1910034666835200*b2*b3*b5^2*b4^6
-3963311602470000*b2*b3^5*b4^6-21232690500000000*b2^18*b4*b5*b3^2
-1253470275000000O0*b2^15*b5*b3^2*b4^2+15616145070612000*b2^2*b5^2*b4^5*b3^2
+58634642997450000*b2^9*b5^2*b4^4+33098145480000000*b2^5*b3*b5^4*b4^2
+4708645488115200*b2^4*b4^7*b5+1668060438750000O*b2^4*b3^2*b4^7
+4587166206562500*b2^4*b3^8*b4^3-63386296672500000*b2^4*b4^5*b3^5
-255091680000000*b2^4*b5^4*b4^3-14844403456524000*b2^2*b3^3*b4^7
-1341680200128000*b2^2*b5^3*b4^5+9908279006175000*b2^2*b3^6*b4^5
-4201910967628800*b2^3*b3*b4^8-11009198895750000*b2^3*b3^7*b4^4
-516652485004800*b2^3*b4^6*b5^2+43796594879820000*b2^3*b4^6*b3^4
+573956280000000*b2^3*b5^5*b4^2+1819944900000000*b2^17*b3^3*b4^2
+34123966875000000*b2^17*b5^2*b3^2+3791551875000000*b2^17*b3^4*b5
+24068368900800000*b2^12*b3*b4^5-209521543725000000*b2^12*b3^4*b4^3
+3311371206000000*b2^12*b5^2*b4^3+44057517058080000*b2^9*b3*b4^6
+332185172765625000*b2^9*b3^4*b5^3-281497211820000000*b2^9*b3^4*b4^4
+27529289648437500*b2^9*b3^8*b5+89090024976562500*b2^9*b3^6*b5^2
+13214059031250000*b2^9*b3^7*b4^2-252176238140625000*b2^9*b3^2*b5^4
+57339577582031250*b2^4*b3^9*b4*b5-57247355961000000*b2^4*b3^2*b5^3*b4^3
+160035153384375000*b2^4*b3^4*b5^2*b4^3+37576200206250000*b2^4*b3^3*b5^4*b4
-29536701922265625*b2^4*b3^5*b5^3*b4-66961566000000000*b2^4*b3*b5^5*b4
-28635037531875000*b2^4*b3^6*b5*b4^3-8340302193750000*b2^4*b3^7*b4*b5^2
-21558013110405000*b2^2*b5*b4^5*b3^4-2410616376000000*b2^2*b3*b5^4*b4^3
+22147537954500000*b2^2*b3^3*b5^3*b4^3+123853487577187500*b2^2*b3^7*b5*b4^3
-34528851082125000*b2^2*b3^5*b4^3*b5^2+32393733720525000*b2^3*b3^5*b4^4*b5
-13761498619687500*b2^3*b3^8*b4^2*b5-65985242452875000*b2^3*b3^3*b5^2*b4^4
+16248702286800000*b2^3*b3*b5^3*b4^4+85831717936320000*b2^3*b4^6*b5*b3^2
-3228504075000000*b2^3*b3^2*b5^4*b4^2-21063747072656250*b2^3*b3^4*b5^3*b4^2
+40658973194531250*b2^3*b3^6*b4^2*b5^2-20380727358000000*b2^12*b5*b3^2*b4^3
-733504405950000000*b2^12*b5^2*b3^3*b4-531167319731250000*b2^12*b5*b4*b3^5
-115071706856250000*b2^12*b5^3*b3*b4-196633334025000000*b2^9*b3*b5^3*b4^2
-1106942297085000000*b2^9*b5^2*b3^3*b4^2-867043629540000000*b2^9*b3^5*b5*b4^2
+12132966000000000*b2^19*b3^2*b4^2-9706372800000000*b2^19*b5*b4^2
+2527701250000000*b2^19*b5*b3^3+242659320000000000*b2^18*b4*b5^2
-9556596263671875*b2^5*b3^8*b5^2-13590646981200000*b2^5*b5^3*b4^4
+42570292447265625*b2^5*b5^3*b3^6+5352929712000000*b2^15*b3*b4^4
-2015823825000000O0*b2^15*b5^2*b3^3+701437096875000O0*b2^15*b3*b5^3
-346636017000000O0*b2^15*b3^4*b4^2+491245762500000O0*b2^15*b5^2*b4^2
-72215836875000000*b2^15*b5*b3^5+13285174694400000*b2^7*b5*b4^6
-37068009750000000*b2^7*b3*b5^5+738295906194000O0*b2^7*b3^5*b4^4
-38373583140000000*b2^7*b4^2*b5^4+203874053625000O0*b2^7*b3^7*b5^2
-198352464672656250*b2^7*b3^5*b5^3+200361560765625000*b2^7*b5^4*b3^3
-85823810094240000*b2^7*b3^2*b4^6+194712306030000000*b2^10*b3^5*b4^3
```

```
-137634282918000000*b2^10*b3^2*b4^5-67583439450000000*b2^10*b4*b5^4
+86072528812500000*b2^11*b3*b5^4+72540134617500000*b2^11*b5^3*b4^2
+232165945116000000*b2^11*b3^3*b4^4-217740658781250000*b2^11*b5^3*b3^3
-58980847612500000*b2^11*b3^6*b4^2-122876765859375000*b2^11*b3^7*b5
-32580695812500000*b2^11*b3^5*b5^2+6673245588000000*b2^8*b5^3*b4^3
+215501876416800000*b2^8*b3^3*b4^5-55952101383750000*b2^8*b3^6*b4^3
+2259808215517440*b2*b4^8*b3^2-245965520056320*b2*b4^8*b5
+309936391200000*b2*b5^4*b4^4+83393323200000000*b2^14*b3^3*b4^3
-39644191125000000*b2^14*b4*b5^3-18764736480000000*b2^16*b5*b4^3
-4901153940000000*b2^16*b3^2*b4^3+77721350850000000*b2^13*b3^5*b4^2
-12789229218750000*b2^13*b5^3*b3^2+39920105559375000*b2^13*b3^4*b5^2
+1619194809375000000*b2^13*b3^6*b5-5616062424000000*b2^13*b5*b4^4
-69843023064000000*b2^13*b3^2*b4^4+72127526814000000*b2^6*b3*b5^3*b4^3
-41841751605637500*b2^6*b5*b4^3*b3^5+77811930466875000*b2^6*b4*b3^6*b5^2
-17051505918750000*b2^6*b4*b5^4*b3^2-12232443217500000*b2^6*b4*b5*b3^8
+18088341773906250*b2^6*b4*b3^4*b5^3+306437776922340000*b2^6*b5*b4^5*b3^2
+21597097938750000*b2^5*b3^3*b5^3*b4^2+308969274868200000*b2^5*b4^4*b5*b3^4
-66126957041376000*b2^5*b3*b4^6*b5+22630019952375000*b2^5*b3^7*b4^2*b5
+718964939109375000*b2^7*b5^2*b4^2*b3^4-214040998251900000*b2^7*b3*b4^4*b5^2
-12878295066720000*b2^10*b5*b4^5-689053114575000000*b2^7*b5*b4^4*b3^3
+300238522971750000*b2^7*b5*b4^2*b3^6+25695836651250000*b2^7*b5^3*b4^2*b3^2
+4563416070000000000*b2^10*b5^3*b3^2*b4+506119435593750000*b2^10*b5*b4*b3^6
+840833648325000000*b2^10*b5^2*b3^4*b4-263423209590000000*b2^10*b5^2*b3*b4^3
-468905848735500000*b2^10*b5*b3^3*b4^3+555628788067500000*b2^11*b5^2*b3^2*b4^2
+42850659949200000*b2^11*b4^4*b5*b3-71570748478590000*b2^8*b4^5*b5*b3
+748402673985000000*b2^8*b5^2*b3^2*b4^3+930846369285000000*b2^8*b5*b3^4*b4^3
-442159576301250000*b2^8*b5^2*b4*b3^5-401823093684375000*b2^8*b5^3*b3^3*b4
-129403392370312500*b2^8*b5*b4*b3^7+225748755675000000*b2^8*b5^4*b3*b4
+6667971520276800*b2*b3^3*b4^6*b5-5256004634100000*b2*b3^2*b5^3*b4^4
-4954139503087500*b2*b3^6*b5*b4^4+10921625722715625*b2*b3^4*b4^4*b5^2
+352233663750000000*b2^14*b5^2*b3^2*b4+53597862486000000*b2^14*b5*b3*b4^3
-118296418500000000*b2^16*b5^2*b3*b4+70840297125000000*b2^16*b5*b4*b3^3
-54130405410000000*b2^13*b5*b3^3*b4^2);
```

Appendix D

Coefficients A'_i for Case $\hat{H}(8,5)$

The coefficients A'_i, $i = 15, 17, 19, 21$ used for the case $\hat{H}(8,5)$ are listed below.

```
A15':= -4723920*B3*B4^6+5467500*B3^4*B4^3+11809800*B3^2*B4^5+34171875*B3^2*B5^3+22800000*B5^2*B4
       +8550000*B4^2*B3^3+1140000*B4^2*B3^2-11390625*B3^4*B5^2+7873200*B4^5*B5+2736000*B4^3*B3
       +17812500*B5*B3^4-26244000*B3^2*B4^4+4428675*B4^4*B5^2+37462500*B5^3*B4
       +26973000*B5^2*B4^3-30375000*B5^2*B3^3-14580000*B5*B4^4+32062500*B5^2*B3^2
       +15746400*B3*B4^5+8201250*B5^3*B4^2+9072000*B5*B4^3-5670000*B4^3*B3^2-48093750*B5^3*B3
       -9120000*B5*B4^2+14580000*B3^3*B4^3+2375000*B5*B3^3-13122000*B3^3*B4^4-26325000*B5^2*B4^2
       -1944000*B4^4*B3+885735*B4^5*B5*B3+18225000*B5*B4*B3^4-19950000*B5*B4*B3^2
       -43302600*B5*B3*B4^4-19683000*B5^2*B3*B4^3+14762250*B5*B3^3*B4^3+41229000*B5*B3*B4^3
       -16402500*B4^2*B5*B3^4+25650000*B5^2*B4*B3+6834375*B5*B4*B3^5-34171875*B5^3*B3*B4
       -6840000*B5*B4^2*B3-45562500*B5*B4*B3^3+107325000*B5^2*B4*B3^2-69255000*B4^2*B5*B3^3
       +24603750*B5^2*B3^2*B4^2-5904900*B5*B3^2*B4^4+85293000*B5*B3^2*B4^3
       -98415000*B3*B5^2*B4^2-7290000*B5*B4^2*B3^2-1728000*B4^4-11390625*B5^4+1216000*B4^3
       -3149280*B4^6+1360800*B4^5+708588*B4^7-19000000*B5^3:
A17':= 344452500*B4^3*B3^5+2289515625*B3^3*B5^3+59049000*B5^3*B4^3-826686000*B4^4*B3^4
       +744017400*B4^5*B3^3-10628820*B5^2*B4^5+44641044*B3*B4^7+688905000*B3^4*B4^3
       -1417176000*B5^2*B4^3*B3^2+496011600*B3^2*B4^5-4586625000*B3^2*B5^3-1380000000*B5^2*B4
       -414000000*B4^2*B3^3-69000000*B4^2*B3^2-1435218750*B3^4*B5^2-477640800*B4^5*B5
       -55200000*B4^3*B3-862500000*B5*B3^4+1516320000*B3^2*B4^4-243741150*B4^4*B5^2
       -2059425000*B5^3*B4-1615950000*B5^2*B4^3+3804187500*B5^2*B3^3+783432000*B5*B4^4
       -1940625000*B5^2*B3^2-869551200*B3*B4^5-431932500*B5^3*B4^2-370656000*B5*B4^3
       +583740000*B4^3*B3^2+1185937500*B5^3*B3+552000000*B5*B4^2-1387530000*B3^3*B4^3
       -143750000*B5*B3^3-1102248000*B3^3*B4^4+1196370000*B5^2*B4^2-99792000*B4^4*B3
       +458810730*B4^5*B5*B3-4668637500*B5*B4*B3^4+1207500000*B5*B4*B3^2+1708484400*B5*B3*B4^4
       +2727189000*B5^2*B3*B4^3+4753444500*B5*B3^3*B4^3-1551474000*B5*B3*B4^3
       -3674160000*B4^2*B5*B3^4+517500000*B5^2*B4*B3+861131250*B5*B4*B3^5+3910781250*B5^3*B3*B4
       -414000000*B5*B4^2*B3+1267875000*B5*B4*B3^3-4588650000*B5^2*B4*B3^2
       +4344840000*B4^2*B5*B3^3-7452202500*B5^2*B3^2*B4^2-2480058000*B5*B3^2*B4^4
       -2740311000*B5*B3^2*B4^3+4107915000*B3*B5^2*B4^2-185220000*B5*B4^2*B3^2+79488000*B4^4
       +842906250*B5^4-73600000*B4^3+191056320*B4^6-50932800*B4^5-29760696*B4^7+1150000000*B5^3
       +652050000*B4^2*B3^4-717609375*B3^5*B5^2+1358437500*B3^5*B5-990984375*B3*B5^4
       -297606960*B4^6*B3^2+246037500*B4*B5^4+55801305*B5*B4^5*B3^2+1746866250*B5^2*B4^2*B3^3
       -82012500*B4*B5^2*B3^4-2234840625*B4*B5^3*B3^2+6834375000*B4*B5^2*B3^3
```

```
        -1033357500*B5*B4^2*B3^5+418263750*B5^3*B4^2*B3-372008700*B5*B4^4*B3^3
        +349865325*B3*B4^4*B5^2+430565625*B4*B5*B3^6+930021750*B5*B4^3*B3^4:
A19':= 190166484375*B3^4*B5^3+119601562500*B3^6*B5+25259850000*B4^3*B3^5-503761781250*B3^3*B5^3
        -16717428000*B5^3*B4^3-53820703125*B3^6*B5^2-122711203125*B3^2*B5^4-8266860000*B4^4*B3^4
        -46294416000*B4^5*B3^3+3348078300*B3^2*B4^7-22320522000*B3^3*B4^6+25833937500*B3^6*B4^3
        -62001450000*B3^5*B4^4-714256704*B4^7*B5-4464104400*B4^4*B5^3+2480058000*B5^2*B4^5
        +55597389120*B3*B4^6-15177954960*B3*B4^7-8266860000*B4^2*B5^4-191872800000*B3^4*B4^3
        +55801305000*B4^5*B3^4+298295865000*B5^2*B4^3*B3^2+20105003520*B5*B4^6
        -188925307200*B3^2*B4^5+411370312500*B3^2*B5^3+107880000000*B5^2*B4+24273000000*B4^2*B3^3
        +5394000000*B4^2*B3^2+417453750000*B3^4*B5^2+6833937600*B4^5*B5-4315200000*B4^3*B3
        +50568750000*B5*B3^4-148541040000*B3^2*B4^4+88276286700*B4^4*B5^2+82406250000*B5^3*B4
        +64831428000*B5^2*B4^3-482456250000*B5^2*B3^3-37289808000*B5*B4^4+151706250000*B5^2*B3^2
        +62845632000*B3*B4^5+131281965000*B5^3*B4^2-1051200000*B5*B4^3-50895000000*B4^3*B3^2
        +42140625000*B5^3*B3-43152000000*B5*B4^2+189880200000*B3^3*B4^3+11237500000*B5*B3^3
        +278011440000*B3^3*B4^4-13648500000*B5^2*B4^2+24377760000*B4^4*B3-190757794500*B4^5*B5*B3
        +569227500000*B5*B4*B3^4-94395000000*B5*B4*B3^2+16196047200*B5*B3*B4^4
        -599867856000*B5^2*B3*B4^3-872276202000*B5*B3^3*B4^3+64360980000*B5*B3*B4^3
        +799129800000*B4^2*B5*B3^4-202275000000*B5^2*B4*B3-498180375000*B5*B4*B3^5
        -584369437500*B5^3*B3*B4+97092000000*B5*B4^2*B3+32298750000*B5*B4*B3^3
        +491062500000*B5^2*B4*B3^2-499632300000*B4^2*B5*B3^3+1204052850000*B5^2*B3^2*B4^2
        +589868017200*B5*B3^2*B4^4+86226120000*B5*B3^2*B4^3-166017600000*B3*B5^2*B4^2
        -64233000000*B5*B4^2*B3^2-2338560000*B4^4-50864062500*B5^4+5753600000*B4^3-12542532480*B4^6
        -921456000*B4^5-4993183440*B4^7-89900000000*B5^3-95134500000*B4^2*B3^4
        -52624687500*B3^5*B5^2-198196875000*B3^5*B5+218472187500*B3*B5^4+46625090400*B4^6*B3^2
        -89047350000*B4*B5^4+6324147900*B5*B4^5*B3^2-646652160000*B5^2*B4^2*B3^3
        +486443475000*B4*B5^2*B3^4+439495875000*B4*B5^3*B3^2-996360750000*B4*B5^2*B3^3
        -182559825000*B5*B4^2*B3^5-37889775000*B5^3*B4^2*B3-114496011000*B5*B4^4*B3^3
        -48567802500*B3*B4^4*B5^2+31574812500*B4*B5*B3^6+246627990000*B5*B4^3*B3^4
        +57408750000*B4^2*B3^5+4185097875*B3^3*B4^5*B5-2232052200*B4^5*B5^2*B3
        +32292421875*B5*B4*B3^7+65445975000*B5^4*B3*B4+1428513408*B4^8+11481750000*B5^5
        -17222625000*B5^2*B4*B3^5-171795684375*B5^3*B3^3*B4+157587018750*B5^2*B4^2*B3^4
        +27280638000*B3*B5^3*B4^3+69751631250*B5*B4^3*B3^5-27900652500*B3^4*B4^4*B5
        +35805837375*B3^2*B4^4*B5^2-77501812500*B5*B4^2*B3^6+1550036250*B5^3*B4^2*B3^2
        -130203045000*B4^3*B3^3*B5^2+6547353120*B4^6*B5*B3:
A21':= -1255252570312500*B3^4*B5^3-675851343750000*B3^6*B5-454647775500000*B4^3*B3^5
        +158813105625000*B3^3*B5^3-90303451171875*B3^7*B5^2+69522324300000*B5^3*B4^3
        +36121380468750*B3^6*B5^2+907442683593750*B3^2*B5^4+693530505000000*B4^4*B3^4
        -451405985400000*B4^5*B3^3-70123108509000*B3^2*B4^7+215233605000000*B3^3*B4^6
        -17338262625000*B3^6*B4^3+166447321200000*B3^5*B4^4-413248521600*B4^7*B5
        +25491453300000*B4^4*B5^3+43345656562500*B4^3*B3^7-11864951766000*B5^2*B4^5
        -226039572000000*B3*B4^6+6916173174000*B3*B4^7+5617597090500*B3^3*B4^7
        -3471287581440*B5*B4^7*B3+93626618175000*B3^5*B4^5+573956280000*B5^3*B4^5
        +73102662000000*B4^2*B5^4+874920285000000*B3^4*B4^3-297739820250000*B4^5*B3^4
        -321886248046875*B5^4*B3^3-2341484759250000*B5^2*B4^3*B3^2-6779858557500*B5^2*B4^5*B3^2
        -90077974041600*B5*B4^6+679890186000000*B3^2*B4^5-512008593750000*B3^2*B5^3
        -199578000000000*B5^2*B4-29936700000000*B4^2*B3^3-9978900000000*B4^2*B3^2
        -1765405631250000*B3^4*B5^2+57995565840000*B4^5*B5+23949360000000*B4^3*B3
        -62368125000000*B5*B3^4+405805723200000*B3^2*B4^4-351782941275000*B4^4*B5^2
        +64020037500000*B5^3*B4-9105355800000*B5^2*B4^3+1265988656250000*B5^2*B3^3
        +34045466400000*B5*B4^4-280656562500000*B5^2*B3^2-138835483920000*B3*B4^5
```

```
-483956795250000*B5^3*B4^2+73401552000000*B5*B4^3+79372710000000*B4^3*B3^2
-327432656250000*B5^3*B3+79831200000000*B5*B4^2-527573655000000*B3^3*B4^3
-20789375000000*B5*B3^3-1043815734000000*B3^3*B4^4-173673315000000*B5^2*B4^2
-57672648000000*B4^4*B3+593016411465000*B4^5*B5*B3-1140072468750000*B5*B4*B3^4
+174630750000000*B5*B4*B3^2-238092129000000*B5*B3*B4^4+1759512577500000*B5^2*B3*B4^3
+2564468545500000*B5*B3^3*B4^3-170265385800000*B5*B3*B4^3-3294032062500000*B4^2*B5*B3^4
+673575750000000*B5^2*B4*B3+2435522478750000*B5*B4*B3^5+1315247675625000*B5^3*B3*B4
-299367000000000*B5*B4^2*B3-317066062500000*B5*B4*B3^3-1658452725000000*B5^2*B4*B3^2
+1001406888000000*B4^2*B5*B3^3-3152445682500000*B5^2*B3^2*B4^2-1534199308200000*B5*B3^2*B4^4
-86451891300000*B5*B3^2*B4^3+402070108500000*B3*B5^2*B4^2+475508070000000*B5*B4^2*B3^2
-4660416000000*B4^4+30720515625000*B5^4-10644160000000*B4^3+20599440480000*B4^6
+9751920480000*B4^5+28489063975200*B4^7+166315000000000*B5^3+240302700000000*B4^2*B3^4
+852048365625000*B3^5*B5^2+500630625000000*B3^5*B5-639783182812500*B3*B5^4
+104077405440000*B4^6*B3^2+373919591250000*B4*B5^4-705069417712500*B5*B4^5*B3^2
+4127746533750000*B5^2*B4^2*B3^3-2922925500000000*B4*B5^2*B3^4-2103095061562500*B4*B5^3*B3^2
+3242183760000000*B4*B5^2*B3^3+2176915196250000*B5*B4^2*B3^5+448819147125000*B5^3*B4^2*B3
+2018094053400000*B5*B4^4*B3^3+460619843737500*B3*B4^4*B5^2-1065982072500000*B4*B5*B3^6
-2794095344250000*B5*B4^3*B3^4-324408645000000*B4^2*B3^5-76892205386250*B3^3*B4^5*B5
+38139394806000*B4^5*B5^2*B3-21672828281250*B5*B4*B3^7-608212901250000*B5^4*B3*B4
-3801886398720*B4^8-68808487500000*B5^5+761178515625000*B5^2*B4*B3^5
+1117394683593750*B5^3*B3^3*B4-1345708257187500*B5^2*B4^2*B3^4-158694187500000*B3*B5^3*B4^3
+124058258437500*B5*B4^3*B3^5+14348907000000*B3^4*B4^4*B5-238819620881250*B3^2*B4^4*B5^2
-80911892250000*B5*B4^2*B3^6-867282187500*B5^3*B4^2*B3^2+832037315625000*B4^3*B3^3*B5^2
+90753966993600*B4^6*B5*B3-297538935552*B4^9+7021996363125*B5*B4^5*B3^4
+230403867187500*B3^7*B5+9174117179520*B4^8*B3-37450647270000*B3^4*B4^6
-13286025000000*B4*B5^5-3188646000000*B5^4*B4^3-3099363912000*B5^2*B4^6
+110593856250000*B3^6*B4^2+64584843750000*B3*B5^5-104029575750000*B3^6*B4^4
+358099892578125*B3^5*B5^3-52313723437500*B4*B3^6*B5^2+221156957812500*B4*B5^4*B3^2
+54182070703125*B4*B5*B3^8-288901845703125*B4*B3^4*B5^3+117033272718750*B3^6*B5*B4^3
+111801900375000*B3^2*B5^3*B4^3-269042006250000*B5^2*B4^3*B3^4-46813309087500*B5*B4^4*B3^5
-130036969687500*B3^7*B5*B4^2-30557857500000*B3*B5^4*B4^2-80131338281250*B3^3*B5^3*B4^2
+320608390781250*B5^2*B4^2*B3^5+32130072554400*B3^2*B5*B4^6
-23197399650000*B3*B5^3*B4^4+80309038865625*B5^2*B4^4*B3^3;
```

Appendix E

LU Maple Codes

E.1 LULEM Code

```
# Signature utilities

module Sig()

    export ModuleApply, UnVeil, Veil, LastUsed, forgetVeil;
    local SIG, UnVeilTable, NextLabel;

    LastUsed := table('sparse');
    UnVeilTable := table():

    NextLabel := proc(label::symbol,vars::set)
        LastUsed[label] := LastUsed[label] + 1;
        label[LastUsed[label],`if`(nargs=2, vars, NULL)];
    end proc;

    Veil := proc(x,p) local A, label;
    label := op(procname);
    LastUsed[label] := LastUsed[label] + 1;
    A := label[LastUsed[label]];
    UnVeil(A,1) := x;
    A;
    end:

    SIG := proc(f,p::posint,A)
        local t, sig;
        option remember;
        if f::'rational' then
            f mod p;
        elif f::'symbol' then
            SIG(f,p,A) := rand() mod p;
        elif f::'indexed' then
            if type(f, A[anything]) then
```

```
                SIG(UnVeil(f,1),p,A)
            else
                SIG(f,p,A) := rand() mod p;
            end if;
        elif f::'+' then
             add(SIG(t,p,A) mod p, t=f) mod p;
        elif f::'*' then
             sig := 1;
             for t in f do
                 sig := sig*SIG(t,p,A) mod p;
             od;
             sig;
        elif f::(anything^rational) then
             SIG(op(1,f),p,A) &^ op(2,f) mod p;
        elif f::(anything^polynom) then
             sig := numtheory['phi'](p);
             t := SIG(op(2, f), sig, A);
             SIG(op(1,f),p,A) &^ t mod p;
        else
            error "expressions involving %1 not done", f;
        fi;
    end:

    ModuleApply := SIG;

    UnVeil := proc(x,n::{nonnegint,identical(infinity)})
        option remember;
        if nargs=1 then
            UnVeil(x,1)
        elif (nargs=2 and n=0) then
            x
        elif x::atomic then
            x
        else
            map(UnVeil, x, n-1)
        end if;
    end proc;

    forgetVeil := proc(label::symbol)
        local i;
        subsop(4=NULL, eval(SIG));
        LastUsed[label] := 0;
        UnVeilTable := table();
    end proc:

end module:

ZeroStrategy_Signature := proc(f,p,A)
    Sig(f,p,A)
end:
```

```
SquareLUwithpivoting := proc(xA::Matrix, Strategy_LEM::procedure,
Q::symbol, Prim::posint, Strategy_Pivots::procedure,
Strategy_Zero::procedure)

    local A, n, k, i, ii, j, m, L, U, mltip, l, r, kp, p, temp,
    normalize, row, flag, z;

    A := copy(xA);
    n := LinearAlgebra[RowDimension](A):
    m := LinearAlgebra[ColumnDimension](A);
    if m<>n then
        error "The input matrix should be square!"
    end if;
    L := Matrix(LinearAlgebra[IdentityMatrix](n),
                shape=triangular[lower]);

# L is lower matrix with 1's as diagonal entries

    U := Matrix(n,n,shape=triangular[upper]):

# U is upper matrix.

    r := Vector(n, k->k );

# create pivot vector

    normalize := z ->`if`(Strategy_LEM(z) > 0,
                          Sig:-Veil[Q](z,Prim), z);

    row := 1;
    for kp from 1 to m while row <= n do

# loop over columns
# We find the pivot element among the entries
# A[r[kp],kp], A[r[kp+1],kp], .... A[r[n],kp] according to
# different pivoting strategies.
# Interchange entries in pivot vector.

        p := row;
        flag := evalb(Strategy_Zero(A[r[row],kp], Prim, Q) = 0);
        for i from row+1 to n do
            if (flag or Strategy_Pivots(A[r[i],kp], A[r[p],kp]))
              and not (Strategy_Zero(A[r[i],kp], Prim, Q)=0) then
                p := i;
                break;

# once we've found a pivot -- not ``best'' !

            fi;
```

```
        od;

# only when the pivot is not equal to zero, we will do the row
# elimination (the whole if statement). else we will continue with
# next column.

        if Strategy_Zero(A[r[p],kp], Prim, Q)=0 then
            WARNING("the matrix appears to be singular.");
        else
            if (p <> row) then
                (r[p], r[row]) := (r[row], r[p]);
            end if;

            userinfo(3, SquareLUwithpivoting,' kp',kp,'r',r);

# Now do Gauss elimination steps to get new A and also keep L
# information in new A. Packing everything into the A matrix
# during the computation of the factors.
# The reason is that this code can be ported to restricted
# memory environments, or equally, applied to very large
# matrices in a large memory environment.

            for i from row+1 to n do
                mltip := normalize(A[r[i],kp]/A[r[row],kp]);
                A[r[i],kp] := mltip;
                for j from kp+1 to m do
                    z := A[r[i],j] - mltip*A[r[row],j];
                    A[r[i], j] := 'if'(Strategy_LEM(z) > 0,
                        Sig:-Veil[Q](z,Prim), z);
                end do;
            end do;
        userinfo(3, SquareLUwithpivoting, ' A ', A);
        end if;
        row := row + 1;
    end do:
    userinfo( 2, SquareLUwithpivoting, 'r', r, 'A',A);

# Seperate new A into L and U

    for i from 1 to n do
        for j from 1 to m do
            if i <= j then
                U[i, j] := A[r[i], j];
            else
                L[i, j] := A[r[i], j];
            end if;
        end do;
    end do;

    L,U, r;
```

```
end proc:

# Solve the linear system Ax=b given the LU decomposition(PA=LU)
# The pivot vector r returned from SquareLUwithpivoting

SolveSquareLUpivot := proc( A, r, b, Strategy_LEM, P, Q) local y,
x, i, s, j, n, normalizer;

   userinfo( 2, SolveSquareLUpivot, ' b ', b, ' r ', r );
   n := LinearAlgebra[RowDimension](A);

# get number of rows in matrix A

   y := Vector(n);

# create vector for solution of Ly=Pb

   x := Vector(n);

# create vector for solution of Ux=y

   normalizer:=y->'if'(Strategy_LEM(y)>0,Sig:-Veil[Q](y,P),y);

# do forward substitution to solve Ly=Pb

   userinfo( 3, SolveSquareLUpivot, ' n ', n, ' y ', y );
   y[1] := normalizer( b[r[1]] );
   for i from 2 to n do
       y[i]:=normalizer(b[r[i]])-add(normalizer(A[r[i],j]*y[j]),
            j=1..i-1) ;
   od;

# do backward substitution to solve Ux=y

   x[n] := normalizer( y[n]/A[r[n],n] );
   for i from n-1 to 1 by -1 do
      s := normalizer(y[i])-add(normalizer(A[r[i],j]*x[j]),
            j=i+1..n);
      x[i] := normalizer( s/ A[r[i],i] );
   od;

   x;

# return solution vector

end proc:


#Veiling strategies.
```

```
LEMStrategy_n := proc(x)
    nops(indets(x)) - 4;
end proc;

LEMStrategy_L:= proc(x)
    length(x) -50;
end proc;

LEMStrategy_LB:= proc(x)
    length(x) - 260;
end proc;


LEMStrategy_Ls:= proc(x)
    length(x) - 120;
end proc;

#Pivoting strategies.

PivotStrategy_Llength := proc( x, y )
    evalb( length(x) - length(y) > 0 );
end proc;

PivotStrategy_Slength := proc( x , y )
    evalb( length(x) - length(y) < 0 );
end proc;

PivotStrategy_Lindets := proc ( x, y)
    evalb ( (nops(indets(x)) - nops(indets(y))) > 0)
end proc;

PivotStrategy_Sindets := proc ( x, y)
    evalb ( (nops(indets(x)) - nops(indets(y))) < 0)
end proc;

PivotStrategy_numeric := proc ( x, y)
    evalb ( (abs(x) - abs(y) ) > 0);
end proc;

#Zero recognition strategies.

ZeroStrategy_length := proc( x, zero, K_LEM )
    length(x)
end proc;

ZeroStrategy_normalizer := proc( x, zero, K_LEM)
    Normalizer(x)
end proc;
```

E.2 MapleFix Code

```
kernelopts(opaquemodules=false);

LU:=ToInert(eval(LinearAlgebra:-LA_Main:-LUDecomposition)):

res := indets(LU, '_Inert_INEQUAT'(specfunc(anything,
              _Inert_FUNCTION), _Inert_INTPOS(0)))[1]:

LU2 := eval(LU, res =_Inert_NOT(_Inert_FUNCTION
           (_Inert_NAME("Testzero"), op([1,2],res)))):

unprotect(LinearAlgebra:-LA_Main:-LUDecomposition):

LinearAlgebra:-LA_Main:-LUDecomposition := FromInert(LU2):
```

Appendix F

Fraction-free LU Maple Codes

F.1 Completely FFLU Code

```
CFFLUint := proc(A)
local i, j, k, L, U, DD, n, m, oldpivot, Ukk, Uik, r, P,
kpivot, Notfound, swap;

use LinearAlgebra in

U := Copy(A);
# Without this, any changes to U will be also changes in A.
(n,m) := Dimension(U):
oldpivot := 1;
L:=IdentityMatrix(n,n,'compact'=false);
P := IdentityMatrix(n, n, 'compact'=false);
DD:=ZeroVector(n,'compact'=false);

for k from 1 to n-1 do

   if U[k,k] = 0 then
    kpivot := k+1;
        Notfound := true;
   while kpivot < (n+1) and Notfound do
        if U[kpivot, k] <> 0 then
            Notfound := false;
        else
            kpivot := kpivot +1;
        end if;
   end do:
   if kpivot = n+1 then
        error "Matrix is rank deficient";
   else
        swap := U[k, k..n];
        U[k,k..n] := U[kpivot, k..n];
        U[kpivot, k..n] := swap;
```

```
        swap := P[k, k..n];
        P[k, k..n] := P[kpivot, k..n];
        P[kpivot, k..n] := swap;
    end if:
    end if:

    L[k,k]:=U[k,k];
    DD[k] := oldpivot * U[k, k];
    Ukk := U[k,k];
    for i from k+1 to n do
        L[i,k] := U[i,k];
        Uik := U[i,k];
        for j from k+1 to m do
           U[i,j] := iquo((Ukk*U[i,j]-U[k,j]*Uik), oldpivot);
        end do;
        U[i,k] := 0;
    end do;
    oldpivot:= U[k,k];
end do;
DD[n]:= oldpivot;

end use;
P, L, DD, U;

end proc:

CFFLUuni := proc(A)
local i, j, k, L, U, DD, n, m, oldpivot, Ukk, Uik, q, P,
kpivot, Notfound, swap;
use LinearAlgebra in

U := Copy(A);
# Without this, any changes to U will be also changes in A.
(n,m) := Dimension(U):
oldpivot := 1;
L:=IdentityMatrix(n,n,'compact'=false);
DD:=ZeroVector(n,'compact'=false);
P := IdentityMatrix(n, n, 'compact'=false);

for k from 1 to n-1 do

    if U[k,k] = 0 then
     kpivot := k+1;
        Notfound := true;
     while kpivot < (n+1) and Notfound do
        if U[kpivot, k] <> 0 then
            Notfound := false;
        else
            kpivot := kpivot +1;
        end if;
```

```
    end do:
    if kpivot = n+1 then
        error "Matrix is rank deficient";
    else
        swap := U[k, k..n];
        U[k,k..n] := U[kpivot, k..n];
        U[kpivot, k..n] := swap;
        swap := P[k, k..n];
        P[k, k..n] := P[kpivot, k..n];
        P[kpivot, k..n] := swap;
    end if:
   end if:

   L[k,k]:=U[k,k];
   DD[k] := oldpivot * U[k, k];
   Ukk := U[k,k];
   for i from k+1 to n do
       L[i,k] := U[i,k];
       Uik := U[i,k];
       for j from k+1 to m do
          divide((Ukk*U[i,j]-U[k,j]*Uik), oldpivot, 'q');
      U[i,j]:= q;
       end do;
       U[i,k] := 0;
   end do;
   oldpivot:= U[k,k];
end do;
DD[n]:= oldpivot;

end use;
P, L, DD, U;

end proc:

CFFLUint_OneBlock := proc(A)
local i, j, k, L, U, DD, n, m, oldpivot, Ukk, Uik, r, P,
       kpivot, Notfound, swap;
use LinearAlgebra in

U := Copy(A);
(n,m) := Dimension(U):
oldpivot := 1;
L:=IdentityMatrix(n,n,'compact'=false);
P := IdentityMatrix(n, n, 'compact'=false);
DD:=ZeroVector(n,'compact'=false);

for k from 1 to n-1 do

  if U[k,k] = 0 then
       kpivot := k+1;
```

```
        Notfound := true;
        while kpivot < (n+1) and Notfound do
             if U[kpivot, k] <> 0 then
                  Notfound := false;
             else
                  kpivot := kpivot +1;
             end if;
        end do:
        if kpivot=n+1 then
           error "Matrix is rank deficient";
        else
           swap := U[k, k..n];
           U[k,k..n] := U[kpivot, k..n];
           U[kpivot, k..n] := swap;
           swap := P[k, k..n];
           P[k, k..n] := P[kpivot, k..n];
           P[kpivot, k..n] := swap;
        end if;
   end if:

   L[k,k]:=U[k,k];
   DD[k] := oldpivot * U[k, k];
   Ukk := U[k,k];
   for i from k+1 to n do
       L[i,k] := U[i,k];
       Uik := U[i,k];
       U[i,k+1..m] := (Ukk*U[i,k+1..m]-U[k,k+1..m]*Uik)/oldpivot;
       U[i,k] := 0;
   end do;
   oldpivot:= U[k,k];
end do;
DD[n]:= oldpivot;

end use;
P, L, DD, U;

end proc;
```

F.2 Partially FFLU Code

```
PFFLUint := proc(A)
local i, j, k, L, U, n, m, oldpivot, Ukk, Uik, P,
kpivot, Notfound, swap;
use LinearAlgebra in

U := Copy(A);
# Without this, any changes to U will be also changes in A.
(n,m) := Dimension(U):
oldpivot := 1;
```

```
L:=IdentityMatrix(n,n,'compact'=false);
P := IdentityMatrix(n, n, 'compact'=false);

for k from 1 to n-1 do

   if U[k,k] = 0 then
    kpivot := k+1;
        Notfound := true;
    while kpivot < (n+1) and Notfound do
        if U[kpivot, k] <> 0 then
            Notfound := false;
        else
            kpivot := kpivot +1;
        end if;
    end do:
    if kpivot = n+1 then
        error "Matrix is rank deficient";
    else
        swap := U[k, k..n];
        U[k,k..n] := U[kpivot, k..n];
        U[kpivot, k..n] := swap;
        swap := P[k, k..n];
        P[k, k..n] := P[kpivot, k..n];
        P[kpivot, k..n] := swap;
    end if:
   end if:

   L[k,k]:=1/oldpivot;
   Ukk := U[k,k];
   for i from k+1 to n do
       L[i,k] := U[i,k]/(oldpivot * U[k, k]);
       Uik := U[i,k];
       for j from k+1 to m do
          U[i,j]:=iquo((Ukk*U[i,j]-U[k,j]*Uik), oldpivot, 'r');
       end do;
       U[i,k] := 0;
   end do;
   oldpivot:= U[k,k];
end do;
L[n,n]:= 1/oldpivot;

end use;
P, L, U;

end proc;

PFFLUuni := proc(A)
local i, j, k, L, U, n, m, oldpivot, Ukk, Uik, P, kpivot,
Notfound, swap;
use LinearAlgebra in
```

```
U := Copy(A);
# Without this, any changes to U will be also changes in A.
(n,m) := Dimension(U):
oldpivot := 1;
L:=IdentityMatrix(n,n,'compact'=false);
P := IdentityMatrix(n, n, 'compact'=false);

for k from 1 to n-1 do

   if U[k,k] = 0 then
    kpivot := k+1;
        Notfound := true;
    while kpivot < (n+1) and Notfound do
        if U[kpivot, k] <> 0 then
            Notfound := false;
        else
            kpivot := kpivot +1;
        end if;
    end do:
    if kpivot = n+1 then
        error "Matrix is rank deficient";
    else
        swap := U[k, k..n];
        U[k,k..n] := U[kpivot, k..n];
        U[kpivot, k..n] := swap;
        swap := P[k, k..n];
        P[k, k..n] := P[kpivot, k..n];
        P[kpivot, k..n] := swap;
    end if:
   end if:

   L[k,k]:=1/oldpivot;
   Ukk := U[k,k];
   for i from k+1 to n do
       L[i,k] := normal(U[i,k]/(oldpivot * U[k, k]));
       Uik := U[i,k];
       for j from k+1 to m do
          divide((Ukk*U[i,j]-U[k,j]*Uik), oldpivot, 'q');
      U[i,j]:=q;
       end do;
       U[i,k] := 0;
   end do;
   oldpivot:= U[k,k];
end do;
L[n,n]:= 1/oldpivot;

end use;
P, L, U;
```

```
end proc;
```

F.3 Table 2.4.2: matrices with integer entries

F.3.1 Completely FFLU Benchmark

```
with(LinearAlgebra):
read"FFLU.txt";
FD := fopen(intCFFLU_log, APPEND);
kernelopts(gcfreq=2*10^7):
ii := 30;

 for i from 1 to ii do
    n := 10 * i;
    A := LinearAlgebra[RandomMatrix](n,n,
                        generator = -10^100..10^100):
    gc();
    st := time():
    timelimit(2000, CFFLUint(A)):
    time_CFFLU:= time() - st;
    fprintf (FD, "%a, %a\n", n, time_CFFLU);
    fflush(FD);
end do:

fclose(FD);
```

F.3.2 Partially FFLU Benchmark

```
with(LinearAlgebra):
read("FFLU.txt");
FD := fopen(intPFFLU_log, APPEND);
kernelopts(gcfreq=2*10^7):
ii := 30;

 for i from 1 to ii do
    n := 10 * i;
    A := LinearAlgebra[RandomMatrix](n,n,
                         generator = -10^100..10^100):
    gc();
    st := time():
    timelimit(2000, PFFLUint(A)):
    time_PFFLU:= time() - st;
    fprintf (FD, "%a, %a\n", n, time_PFFLU);
    fflush(FD);
end do:

fclose(FD);
```

F.3.3 MapleLU Benchmark

```
with(LinearAlgebra):
FD := fopen(intMapleLU_log, APPEND);
kernelopts(gcfreq=2*10^7):
ii := 30;

for i from 1 to ii do
    n := 10 * i;
    A := LinearAlgebra[RandomMatrix](n,n,
                        generator = -10^100..10^100):
    gc();
    st := time():
    timelimit(2000, LinearAlgebra[LUDecomposition](A,
                                    method=FractionFree)):
    time_MapleLU:= time() - st;
    fprintf (FD, "%a, %a\n", n, time_MapleLU);
    fflush(FD);
end do:

fclose(FD);
```

F.4 Table 2.4.4: 5 by 5 matrices with integer of different length entries

F.4.1 Completely FFLU Benchmark

```
with(LinearAlgebra):
read"FFLU.txt";
FD := fopen(intCFFLU_len_log,APPEND);
kernelopts(gcfreq=2*10^7):
ii := 30;

for i from 1 to ii do
    n := 5;
    l := 200*i;
    A := LinearAlgebra[RandomMatrix](n,n, generator = -10^l..10^l):
    gc();
    st := time():
    timelimit(2000, CFFLUint(A)):
    time_CFFLU:= time() - st;
    fprintf (FD, "%a, %a\n", l, time_CFFLU);
    fflush(FD);
end do:

fclose(FD);
```

F.4.2 Partially FFLU Benchmark

```
with(LinearAlgebra):
read("FFLU.txt");
FD := fopen(intPFFLU_len_log, APPEND);
kernelopts(gcfreq=2*10^7):
ii := 30;

for i from 1 to ii do
    n := 5;
    l := 200 * i;
    A := LinearAlgebra[RandomMatrix](n,n, generator = -10^l..10^l):
    gc();
    st := time():
    timelimit(2000, PFFLUint(A)):
    time_PFFLU:= time() - st;
    fprintf (FD, "%a, %a\n", l, time_PFFLU);
    fflush(FD);
end do:

fclose(FD);
```

F.4.3 Maple Benchmark

```
with(LinearAlgebra):
FD := fopen(intMapleLU_len_log,APPEND);
kernelopts(gcfreq=2*10^7):
ii := 30;

for i from 1 to ii do
    n := 5;
    l := 200 * i;
    A := LinearAlgebra[RandomMatrix](n,n, generator = -10^l..10^l):
    gc();
    st := time():
    timelimit(2000, LinearAlgebra[LUDecomposition](A,
                                        method=FractionFree)):
    time_MapleLU:= time() - st;
    fprintf (FD, "%a, %a\n", l, time_MapleLU);
    fflush(FD);
end do:

fclose(FD);
```

F.5 Table 2.4.6: matrices with univariate polynomial entries

F.5.1 Completely FFLU Benchmark

```
with(LinearAlgebra):
read"FFLU.txt";
FD := fopen(randpolyCFFLU_log, APPEND);
kernelopts(gcfreq=2*10^7):
ii := 10;

for i from 1 to ii do
    n := 5 * i;
    poly :=proc() randpoly([x],degree=4); end proc:
    A := LinearAlgebra[RandomMatrix](n,n, generator = poly):
    gc();
    st := time():
    timelimit(2000, CFFLUuni(A)):
    time_CFFLU:= time() - st;
    fprintf (FD, "%a, %a\n", n, time_CFFLU);
    fflush(FD);
end do:

fclose(FD);
```

F.5.2 Partially FFLU Benchmark

```
with(LinearAlgebra):
read("FFLU.txt");
FD := fopen(randpolyPFFLU_log,APPEND);
kernelopts(gcfreq=2*10^7):
ii := 10;

for i from 1 to ii do
    n := 5 * i;
    poly :=proc() randpoly([x],degree=4); end proc:
    A := LinearAlgebra[RandomMatrix](n,n, generator = poly):
    gc();
    st := time():
    timelimit(2000, PFFLUuni(A)):
    time_PFFLU:= time() - st;
    fprintf (FD, "%a, %a\n",n,time_PFFLU);
    fflush(FD);
end do:

fclose(FD);
```

F.5.3 MapleLU Benchmark

```
with(LinearAlgebra):
FD := fopen(randpolyMapleLU_log,APPEND);
kernelopts(gcfreq=2*10^7):
ii := 10;

for i from 1 to ii do
    n := 5 * i;
    poly :=proc() randpoly([x],degree=4); end proc:
    A := LinearAlgebra[RandomMatrix](n,n, generator = poly):
    gc();
    st := time():
    timelimit(2000, LinearAlgebra[LUDecomposition](A,
                                    method=FractionFree)):
    time_MapleLU:= time() - st;
    fprintf (FD, "%a, %a\n", n, time_MapleLU);
    fflush(FD);
end do:

fclose(FD);
```

F.6 Fraction-free QR factoring

```
FFQR := proc(AA)
local A, B, C, a, i, j, k, L, U, D, n, m, Q, R;
A := AA;
n := LinearAlgebra[RowDimension](A):
m := LinearAlgebra[ColumnDimension](A):
B := Matrix(m,m);
C := Matrix(m,m+n);
Q := Matrix(n,m);
R := Matrix(m,m);
B := Transpose(A).A;

for i from 1 to m do
     for j from 1 to m do
         C[i,j] := B[i,j];
     end do:
     for j from m+1 to m+n do
          C[i, j] := A[j-m, i];
     end do;
end do:


L, D, U := CFFLU(C);

for i from 1 to m do
    for j from 1 to n do
```

```
        Q[j,i] := U[i,m+j];
    end do:
end do:

for i from 1 to m do
    for j from 1 to m do
        R[j,i] := L[i,j];
    end do:
end do:

return Q, D, R;
end proc;
```

VITA

Name: Wenqin Zhou

Post-secondary Education and Degree :

Beijing University of Aeronautics and Astronautics
Beijing, China
1995-1999 B.S.

Beijing University of Aeronautics and Astronautics
Beijing, China
1999-2002 M.E.

The University of Western Ontario
London, Ontario, Canada
2003-2007 Ph.D.

Honours and Awards:

Western Graduate Research Scholarship
2003-2006

Special University Scholarship
2003-2006

International Graduate Student Scholarship
2003-2006

Related Work Experience:

Teaching Assistant, Department of Applied Mathematics
The University of Western Ontario
2003-2007

Research Assistant, Department of Applied Mathematics
The University of Western Ontario
2003-2006

Research Assistant, School of Science
Beijing University of Aeronautics and Astronautics
1999-2002

Publications:

Wenqin Zhou, J. Carette, D.J. Jeffrey, M.B. Monagan. Hierarchical representations with signatures for large expression management, Artificial Intelligence and Symbolic Computation 2006, Springer-Verlag LNAI 4120, pp. 254-268, (2006).

Wenqin Zhou, David J. Jeffrey, Robert M. Corless. Fraction-free forms of LU and QR matrix factors, Proceedings of Transgressive Computing, pp. 443-446, (2006).

Marc Moreno Maza, Eric Schost, Wenqin Zhou. Primary decomposition of zero dimensional ideals: Putting Monico's algorithm into practice, Proceedings of Transgressive Computing, pp. 419-428, (2006).

Wenqin Zhou, David J. Jeffrey, Gregory J. Reid. Symbolic Preprocessing for the Numerical Simulation of Multibody Dynamic Systems. International workshop on Symbolic-Numeric Computation, Proceedings of SNC, pp 343-354, (2005).

Wenqin Zhou, David J. Jeffrey, Gregory J. Reid. An Algebraic Method for Analyzing Open-Loop Dynamic Systems, ICCS 2005, LNCS 3516, pp. 586-593, (2005).

Wenqin Zhou, David J. Jeffrey, Gregory J. Reid, Chad Schmitke and John McPhee. Implicit Reduced Involutive Forms and Their Application to Engineering Multibody Systems, IWMM 2004, LNCS 3519, pp. 31-43, (2004).