Fachbereich Mathematik
AG Algebraische Geometrie und Computeralgebra
Universität Kaiserslautern

Resultantenmethoden zur Lösung algebraischer Gleichungssysteme implementiert in Singular

Diplomarbeit von Moritz Wenk

27. Juli 1999

Harry: „Während der letzten Wochen hat sich ein Team mit der Sache beschäftigt. Also, was wir rausgekriegt haben, läßt sich auf zwei Grundkonzepte reduzieren … erstens … die Leute tragen zu wenig Hüte … zweitens … Materie ist Energie; die wir normalerweise nicht wahrnehmen. Manche Energien haben einen spirituellen Ursprung, der auf die Seele der Menschen einwirkt. Die Seele existiert jedoch nicht *ab initio*, wie das orthodoxe Christentum lehrt; sie muß durch einen Prozeß der Selbstbeobachtung unter kundiger Führung ins Leben gerufen werden. Und das wiederum kommt nur vor, weil der Mensch die einzigartige Fähigkeit hat, sich von geistigen Dingen durch alltäglichen Kleinkram ablenken zu lassen.“

Max: „Wie war das mit den Hüten?“

Monty Python, *The Meaning of Life*

And now something completely different …

Inhaltsverzeichnis

Danksagungen 4

1 Einleitung 5

1.1 Allgemeines . . . 7

1.2 Polytope und ihre Eigenschaften . . . 8

1.3 Vandermonde Systeme . . . 10

2 Resultanten 14

2.1 Resultanten zweier Polynome . . . 14

2.2 Macaulay Resultanten . . . 18

2.2.1 Einführung . . . 18

2.2.2 Die Macaulay Resultante . . . 19

2.2.3 Konstruktion der Macaulay Resultante . . . 22

2.2.4 Eigenschaften der Macaulay Resultante . . . 24

2.2.5 Verallgemeinerte charakteristische Polynome . . . 27

2.2.6 Generischer linearer Koordinatenwechsel . . . 28

2.3 Sparse Resultanten . . . 28

2.3.1 Einführung . . . 28

2.3.2 Minkowski-Summen und Gemischte Volumina . . . 29

2.3.3 Die sparse Resultante . . . 32

2.3.4 Gemischte polyhedrale Subdivision . . . 34

2.3.5 Konstruktion der sparsen Resultante . . . 42

2.3.6 Eigenschaften der sparsen Resultante . . . 50

3 Lösen von Gleichungssystemen **53**

3.1 Die u-Resultante 54

3.2 Numerische Lösungen 57

3.2.1 Spezialisierung der Koeffizienten 58

3.2.2 Zuordnung der Nullstellen 59

3.2.3 Interpolation univariater Determinanten 61

3.2.4 Bestimmung der Nullstellen univariater Polynome 62

3.3 Zusammenfassender Algorithmus 63

4 Auswertung **68**

4.1 Die Testumgebung 71

4.2 Vergleich Macaulay – sparse Resultante 73

4.3 Vergleich Resultanten – FGLM-Algorithmus 78

4.4 Vergleich Resultanten – CAS 80

4.5 Zusammenfassung 82

4.6 Die Beispiele 83

5 Zur Implementierung **89**

Literaturverzeichnis **109**

Danksagungen

Mein erster und besonderer Dank geht an Prof. Dr. G. M. Greuel für die Themenstellung, die sich als interessanter aber auch umfangreicher erwiesen hat, als von mir zunächst vermutet. Er verstand es, mich durch seine gute Betreuung, Anregungen und Tips, insbesondere auch in der Endphase, immer wieder zu motivieren und zum Gelingen der Arbeit beizutragen.

Ebenso gilt mein Dank Prof. Dr. G. Pfister, der in ähnlicher Weise mit Ratschlägen zur Seite stand. Dr. Hans Schönemann danke ich für seine geduldigen Einweisungen in die Untiefen der Programmierung im SINGULAR-Kernel. Auch für die gute Zusammenarbeit mit ihm als Chef des CTM, das mir vier Jahre lang einen Arbeitsplatz geboten hat, möchte ich ihm noch einmal danken.

Besonderer Dank gebürt Wilfred Pohl, der mir mit seiner ständig weiterentwickelten Implementierung des Bareiss-Algorithmus ein schnelles Verfahren zur Berechnung sparser Determinanten bereitstellte.

Für das gute Arbeitsklima geht Dank an meine Zimmergenossen Olaf Bachmann und Tim Wichmann, der immer mit frisch gekochtem Tee aufwarten konnte, die Arbeitsgruppe jedoch leider recht bald verließ. Ebenso möchte ich dem Rest der Arbeitsgruppe Algebraische Geometrie und Computeralgebra danken.

Für die moralische und seelische Unterstützung geht Dank an Andrea Mindnich, mit der zusammen ich viele der Mathematik-Vorlesungen gehört habe und die diese Arbeit mehrmals intensiv zur Korrektur gelesen hat. Auch Sandra Bornhofen sei für das Auffinden vieler lustiger Rechtschreibefehler gedankt.

1 Einleitung

Die vorliegende Arbeit beschäftigt sich mit dem Lösen von nulldimensionalen Gleichungssystemen in n Polynomen mit n Veränderlichen. Dabei sollen multivariate Resultanten zum Einsatz kommen, die die Lösbarkeit eines Systems charakterisieren und die Berechnung der Nullstellen vereinfachen. Neben der Behandlung der Theorie zu dieser Methode war das Ziel der Arbeit, eine effiziente Implementierung für das Computeralgebrasystem SINGULAR ([GPS+99]) zu erstellen. In anschließenden Versuchen sollte die Nutzbarkeit der Methode sowie die Qualität der Implementierung untersucht werden.

Bislang wurden Abwandlungen des 1965 gefundenen Buchberger-Algorithmus verwendet, um eine reduzierte Standardbasis eines Gleichungssystems zu berechnen und so die Berechnung der Nullstellen zu ermöglichen. Vor etwa 15 Jahren wurden multipolynomiale Resultanten als Methode zum Lösen von Gleichungssystemen als Alternative zu Standardbasen „wiederentdeckt"; seitdem versucht man, schnelle Algorithmen zur Konstruktion von Resultanten und zum Lösen von Systemen unter Verwendung derselben zu entwickeln. Diese multipolynomialen Resultanten stellen eine Verallgemeinerung der Sylvesterresultante für zwei Polynome dar und wurden zunächst 1902 von F. S. Macaulay eingeführt. Die Macaulay Resultanten sind jedoch auf homogene dicht besetzte Polynome beschränkt. Vor etwa 5 Jahren wurde diese Form der Resultante von I. Gelfand, M. Kapranov und A. Zelevinsky ([GKZ94]) verallgemeinert und von ihren Beschränkungen befreit. Das Resultat dieser Bemühungen ist die sparse Resultante.

Resultantenmethoden

Zunächst wurde die Macaulay Resultante implementiert, deren Konstruktionsverfahren zwar recht einfach ist, die jedoch aufgrund ihrer Einschränkungen nicht gut zum Lösen von Gleichungssystemen geeignet ist. Die sparse Resultante, die daraufhin implementiert wurde, hat einen signifikant niedrigeren Grad für Polynome, die sparse in dem Sinne sind, daß ihr gemischtes Volumen kleiner ist, als ihre Bèzout-Zahl. Der Algorithmus verwendet eine gemischte polyhedrale Subdivision der Minkowski-Summe der Newton Polytope des Eingabesystems zur Konstruktion der sparsen Resultantenmatrix, deren Determinante ein Vielfaches ungleich Null der sparsen Resultante ist.

Numerische Lösungen

Zur Bestimmung der Nullstellen eines nulldimensionalen Systems eignet sich die u-Resultante, die ihren Namen aufgrund der besonderen Form der Eingabepolynome zur Konstruktion der Resultante trägt. Sie ermöglicht eine Spezialisierung der Resultante, so daß mit Hilfe eines Algorithmus zur

Bestimmung der komplexen Nullstellen eines univariaten Polynoms die einzelnen Koordinaten der Nullstellen des Systems bestimmt werden können. Hierbei kommt der Algorithmus von Laguerre zur Anwendung, der sich durch seine Robustheit und schnelle Konvergenz auszeichnet.

Implementiert in Singular

Von Beginn an lautete das Ziel, eine effiziente Implementierung zu erreichen. Die Verwendung der SINGULAR-eigenen Interpretersprache schied damit weitgehendst aus, stattdessen wurde der Algorithmus in C/C++ geschieben und direkt in den Kern von SINGULAR eingebunden.

SINGULAR ist ein Computeralgebrasystem, das speziell für die Anforderungen der kommutativen Algebra, der algebraischen Geometrie und der Singularitätentheorie entwickelt wurde. Die wichtigsten Objekte, mit denen SINGULAR arbeitet, sind Polynome, Ringe und Modulen, es gibt eine interaktive Shell sowie eine eigene, C-ähnliche Interpretersprache. SINGULAR bietet eine der schnellsten und allgemeinsten Implementationen zur Gröbnerbasen-Berechnung sowie Algorithmen zur polynomialen Faktorisierung, charakteristische Mengen, Syzygien-Berechnung und vieles mehr. Außerdem besitzt SINGULAR eine eigene Homepage, diese findet sich unter

`http://www.mathematik.uni-kl.de/~zca/Singular`.

Dort steht SINGULAR auch als ausführbares Programm für verschiedene Computerplatformen zur Verfügung.

Zu dieser Arbeit

Die vorliegende Arbeit gliedert sich in drei große Abschnitte:

im Kapitel 2 wird zunächst in Abschnitt 2.1 die klassische Formulierung der Resultante für zwei Polynome, die Sylvesterresultante, vorgestellt. Eine erste Verallgemeinerung bietet Abschnitt 2.2 mit der Macaulay Resultante für multivariate multipolynomiale homogene und dicht besetzte Polynome. In Abschnitt 2.3 schließlich wird die sparse Resultante eingeführt, die die beiden zuvor aufgezeigten Resultanten weiter verallgemeinert.

In Kapitel 3 wird in Abschnitt 3.2 zunächst die u-Resultante eingeführt und im darauf folgenden Abschnitt darauf eingegangen, wie man letztendlich die Nullstellen eines Systems numerisch bestimmt und wie die dabei auftretenden Hindernisse zu überwinden sind. Abschnitt 3.3 schließlich stellt einen Algorithmus vor, der alle zuvor gezeigten Verfahren, von der Konstruktion der Resultante bis hin zu den numerisch bestimmten Nullstellen, zusammenfaßt.

In den Kapiteln 4 und 5 geht es dann von der Theorie in die Praxis. Hier finden sich die Zeitmessungen und Auszüge aus dem Quelltext der Implementierung in SINGULAR.

1.1 Allgemeines

Wie üblich bezeichnen $\mathbb{Q}$, $\mathbb{R}$ bzw. $\mathbb{C}$ den Körper der rationalen, der reellen bzw. der komplexen Zahlen. $\mathbb{Z}$ bezeichnet den Körper der ganzen Zahlen, $\mathbb{N}$ den Körper der ganzen positiven Zahlen mit Null. Ist von einem Körper die Rede, so soll dieser immer die *Charakteristik 0* haben.

Definition 1: Sei $\mathbb{K}$ ein Körper. Ein **Polynom** f in $x_1, \dots, x_n$ mit Koeffizienten in $\mathbb{K}$ ist eine endliche Linearkombination von Monomen. Wir schreiben ein Polynom in der Form

$$f = \sum_a c_a x^a, \quad c_a \in \mathbb{K},$$

wobei die Summe über eine endliche Anzahl von n-tupeln $a = a_1, \dots, a_n$ geht. Die Menge aller Polynome in $x_1, \dots, x_n$ wird mit $\mathbb{K}[x_1, \dots, x_n]$ bezeichnet.

Solch ein Polynomring $\mathbb{K}[x_1, \dots, x_n]$ soll immer ein kommutativer Ring mit Einselement sein.

Definition 2: Sei $t \in \mathbb{K}[x_1, \dots, x_n]$. Existieren $a_1, \dots, a_n \in \mathbb{N}$ und ein $c \in \mathbb{K}$ mit

$$t = c \cdot x_1^{a_1} \cdots x_n^{a_n},$$

so heißt t ein **Term** und c der **Koeffizient** des Terms. Ein Term mit Koeffizient 1 heißt **Monom**.

Monome in $\mathbb{K}[x_1, \dots, x_n]$ haben also die Gestalt $x_1^{a_1} \cdots x_n^{a_n}$ mit $a_i \in \mathbb{N}$. Wir vereinbaren für sie die abkürzende Schreibweise

$$x^a := x_1^{a_1} \cdots x_n^{a_n} \ \text{ mit } \ a = (a_1, \dots, a_n) \in \mathbb{N}^n.$$

Definition 3: Der **Grad eines Monoms** x^a ist die Summe der Exponenten $a_1 + \cdots + a_n =: |a|$. Der **Grad eines Polynoms**, bezeichnet mit $\deg(f)$, ist gegeben durch das größte $|a|$, so daß der Koeffizient c_a ungleich Null ist.

Definition 4: Sei $\mathbb{K}$ ein Körper und $f_1, \dots, f_s$ Polynome in $\mathbb{K}[x_1, \dots, x_n]$. Wir definieren

$$\mathbf{V}(f_1, \dots, f_n) = \{(b_1, \dots, b_n) \in \mathbb{K}^n \mid f_i(b_1, \dots, b_n) = 0 \ \text{ für alle } \ 1 \leq i \leq s\}.$$

Die Menge $\mathbf{V}$ bezeichnen wir als die durch $f_1, \dots, f_s$ definierte **affine Varietät**.

Die affine Varietät $\mathbf{V}(f_1, \dots, f_s) \subset \mathbb{K}$ ist also die Menge aller **Lösungen** oder **Nullstellen** des Systems von Gleichungen

$$f_1(b_1, \dots, b_n) = \cdots = f_s(b_1, \dots, b_n) = 0.$$

Wir bezeichnen ein Polynom f als **homogen**, wenn alle seine Monome den gleichen Grad haben. Homogene Polynome spielen eine zentrale Rolle, wenn man algebraische Geometrie über dem **Projektiven Raum** $\mathbb{P}^n(\mathbb{K})$ betreibt. Betrachtet man dort nämlich Varietäten von Idealen, so hat dies nur Sinn, wenn man sich auf homogene Ideale beschränkt, eine Folge der homogenen Koordinaten im projektiven Raum, die nur bis auf ein skalares Vielfaches eindeutig sind. Eine Einführung in die projektive Welt, die Berechnung projektiver Varietäten, in den Zusammenhang zwischen affinen und projektiven Varietäten und vieles mehr findet man in [CLO97a].

Bei der Berechnung der Resultanten von Systemen von Gleichungen werden wir uns auf **nulldimensionale** Ideale beschränken.

Definition 5: Sei $I \subset \mathbb{K}[x_1,\ldots,x_n]$ ein Ideal. Ist $\mathbb{K}[x_1,\ldots,x_n]/I$ als $\mathbb{K}$-Vektorraum von endlicher Dimension, so heißt I **nulldimensional**.

Verwirrend erscheint auf den ersten Blick die Namensgebung „nulldimensional"; intuitiver wäre wohl „von endicher Dimension" gewesen.

1.2 Polytope und ihre Eigenschaften

Dieser Abschnitt soll lediglich einen Überblick über die im Zusammenhang mit Polytopen verwendeten Begriffe und Konventionen geben, die in Kapitel 2 benötigt werden. Sehr viel mehr Informationen zum Thema Polytope findet man in [Zie95]. Dort finden sich auch die Beweise der unten als Tatsachen hingestellten Sachverhalte.

Definition 6:

(i) Eine Menge $S \subset \mathbb{R}^n$ heißt **konvex**, wenn jedes Liniensegment, das zwei beliebige Punkte aus S verbindet, in ihr enthalten ist.

(ii) Ist eine Menge $S \subset \mathbb{R}^n$ selbst nicht konvex, so ist ihre **konvexe Hülle Conv**(S) die kleinste konvexe Menge, die S enthält.

Dies bedeutet, das alle Punkte in einer konvexen Menge $\mathrm{Conv}(S)$ durch Linearkombinationen von Punkten in S gebiltet werden können.

Sei S eine nicht notwendigerweise konvexe Teilmenge von $\mathbb{R}^n$ und $\lambda_1,\ldots,\lambda_n \in \mathbb{R}$. Dann gilt:

$$\mathrm{Conv}(S) = \{\lambda_1 s_1 + \cdots + \lambda_n s_n \mid (s_1,\ldots,s_n) \in S, \lambda_i \geq 0, \sum_{i=1}^{n} \lambda_i = 1\} \tag{1.1}$$

Linearkombinationen der Form $\lambda_1 s_1 + \cdots + \lambda_n s_n$ mit $\lambda_i \geq 0$, $\sum_{i=1}^{n} \lambda_i = 1$ werden **konvexe Kombinationen** genannt.

Definition 7: Ein **Polytop** ist die konvexe Hülle einer endlichen Menge $\mathcal{A} \subset \mathbb{R}^n$. Ist diese Menge $\mathcal{A}$ gegeben durch $\mathcal{A} = \{m_1,\ldots,m_l\} \subset \mathbb{R}^n$, so ist das korrespondierende Polytop durch

$$\mathrm{Conv}(\mathcal{A}) = \{\lambda_1 m_1 + \cdots + \lambda_l s_l \mid \lambda_i \geq 0, \sum_{i=1}^{n} \lambda_i = 1\}$$

für $\lambda_1,\ldots,\lambda_l \in \mathbb{R}$ gegeben.

In niedrigen Dimensionen sind Polytope bekannte geometrische Geblide: Ein Polytop

(i) in $\mathbb{R}$ ist ein Geradensegment,

(ii) in $\mathbb{R}^2$ ist ein Geradensegment oder ein konvexes Polygon,

(iii) in $\mathbb{R}^3$ ist ein Geradensegment, ein konvexes Polygon oder ein 3-dimensionaler Polyhedron.

Jedes Polytop hat eine wohldefinierte Dimension (siehe auch [Zie95]).

Die für uns später interessanten Polytope sind solche mit ganzzahligen Koordinaten, also konvexe Hüllen $\mathrm{Conv}(\mathcal{A})$ von ganzzahligen endlichen Teilmengen $\mathcal{A} \subset \mathbb{Z}$. Polytope dieser Form bezeichnet man auch als **Gitter-Polytope**, da ihre Eckpunkte auf den Gitterpunkten des zugrundeliegenden Koordinatensystems liegen.

Ein Polytop $Q \subset \mathbb{R}^n$ hat ein n-dimensionales Volumen, welches mit $\mathrm{Vol}_n(Q)$ bezeichnet werden soll und sich wie folgt berechnen läßt:

$$\mathrm{Vol}_n(Q) = \int \cdots \int_Q 1 da_1 \cdots da_n, \tag{1.2}$$

wobei $(a_1, \ldots, a_n) \in \mathbb{R}^n$. Man beachte, das Q genau dann ein positives Volumen hat, wenn es n-dimensional ist.

Polytope zeichnen sich durch spezielle, im Allgemeinen als **Seiten** bezeichnete Teilmengen aus. Zum Beispiel weist ein 3-dimensionales Polytop in $\mathbb{R}^3$ die folgenden Teilmengen auf:

(i) Seiten, gegeben durch Polygone, die in Ebenen liegen,

(ii) Kanten, gegeben durch Geradensegmente, die verschiedene Paare von Eckpunkten verbinden,

(iii) Eckpunkte, gegeben durch Punkte in $\mathbb{R}^3$.

Kommen wir als nächstes zur exakten Definition einer Seite eines Polytops. Dazu benötigen wir den Begriff der Hyperebene.

Definition 8: Sei $Q \subset \mathbb{R}^n$ ein Polytop und $v \in \mathbb{R}^n \backslash \{(0, \ldots, 0)\}$ ein Vektor.

(i) Eine **affine Hyperebene** in $\mathbb{R}^n$ ist durch eine Gleichung der Form

$$m \cdot v = -a$$

mit $m \in Q$ definiert. ($\cdot$ seht dabei für die Skalarmultiplikation.)

(ii) Ist

$$a_Q(v) := -\min_{m \in Q}(m \cdot v),$$

dann bezeichnet die Gleichung

$$m \cdot v = -a_Q(v)$$

die **unterstützende Hyperebene** von Q und v ist die **nach innen gerichtete Normale** dieser Hyperebene.

Mit $Q_v := Q \cap \{m \in \mathbb{R}^n \mid m \cdot v = -a_Q(v)\}$ bezeichnen wir die Seite von Q, die durch $v \in \mathbb{R}^n \backslash \{(0, \ldots, 0)\}$ bestimmt wird.

Jede Seite Q_v von $Q \subset \mathbb{R}^n$ mit $\dim(Q) = n$ ist ein Polytop mit einer Dimension kleiner als $\dim(Q)$. **Eckpunkte** sind Seiten mit Dimension 0 (also Punkte in $\mathbb{R}^n$), **Kanten** sind Seiten mit Dimension 1. Seiten der Dimension $n-1$ werden als **Facetten** bezeichnet.

Jedes n-dimensionale Polytop $Q \subset \mathbb{R}^n$ läßt sich anhand seiner Seiten charakterisieren. Ist $F \subset Q$ eine Seite des Polytops, dann ist die nach innen gerichtete Normale bis auf Multiplikation mit einer positiven Konstanten eindeutig bestimmt. Angenommen, Q hat die Seiten $F_1, \ldots, F_N \subset Q$ mit entsprechenden Normalen $v_1, \ldots, v_N \in \mathbb{R}^n$. Jede Seite F_i hat eine durch die Gleichung $m \cdot v = -a_j$ für ein a_j definierte unterstützende Hyperebene. Dann kann man zeigen, daß das Polytop Q durch die Menge

$$Q = \{m \in \mathbb{R}^n \mid m \cdot v \geq -a_j \ \text{ für alle } \ j = 1, \ldots, N\} \tag{1.3}$$

gegeben ist. Man beachte, daß $a_j = a_Q(v_j)$ ist.

Mit der folgenden Notation für Polynome $f \in \mathbb{K}[x_1, \ldots, x_n]$ wird klar, daß eine enge Beziehung zwischen Gitter-Polytopen und Polynomen besteht:

$$f = \sum_{a \in \mathbb{Z}^n_{\geq 0}} c_a x^a. \tag{1.4}$$

Das **Newton Polytop** $\mathrm{NP}(f)$ von f ist das Gitter-Polytop

$$\mathrm{NP}(f) = \mathrm{Conv}(\{a \in \mathbb{Z}^n_{\geq 0} \mid c_a \neq 0\}) \tag{1.5}$$

Mit anderen Worten, das Newton Polytop „speichert“ die Struktur eines Polynoms, daß heißt, es sagt uns, welche Monome mit Koeffizienten ungleich Null und welche nicht, wobei natürlich nur diejenigen Monome betrachtet werden, die das Newton Polytop bilden. Die eigentlichen Werte der Koeffizienten spielen dabei keine Rolle.

Natürlich kann man auch den umgekehrten Weg vom Newton Polytop zum Polynom gehen. Angenommen, wir haben eine endliche Menge

$$\mathcal{A} = \{a_1, \ldots, a_l\} \in \mathbb{Z}^n_{\geq 0} \tag{1.6}$$

von Exponenten. Dann bezeichne $L(\mathcal{A})$ die Menge aller Polynome, desen Terme ausschließlich Exponenten aus $\mathcal{A}$ haben, das heißt:

$$L(\mathcal{A}) = \{c_1 x^{a_1} + \cdots + c_l x^{a_l} \mid c_i \in \mathbb{K}\}. \tag{1.7}$$

Man beachte, daß $L(\mathcal{A})$ ein Vektorraum über $\mathbb{K}$ mit Dimension l ist.

1.3 Vandermonde Systeme

Seien $p_1, \ldots, p_n \in \mathbb{K}[x_1, \ldots, x_n]$ gegeben. Bilden diese Polynome ein lineares System von Gleichungen der Form

$$\begin{aligned}
p_1 &= x_1 + k_1 x_2 + k_1^2 x_3 + \cdots + k_1^{n-1} x_n = w_1, \\
p_2 &= x_1 + k_2 x_2 + k_2^2 x_3 + \cdots + k_2^{n-1} x_n = w_2, \\
&\vdots \\
p_n &= x_1 + k_n x_2 + k_n^2 x_3 + \cdots + k_n^{n-1} x_n = w_n,
\end{aligned} \tag{1.8}$$

mit $w_1, \ldots, w_n \in \mathbb{K}$, so bezeichnet man dies als ein **Vandermonde System** von Gleichungen.

Die Koeffizientenmatrix dieses Systems (1.8) heißt die **Vandermonde Matrix** und hat die Form

$$V_n := \begin{pmatrix} 1 & k_1 & k_1^2 & \ldots & k_1^{n-1} \\ 1 & k_2 & k_2^2 & \ldots & k_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & k_n & k_n^2 & \ldots & k_n^{n-1} \end{pmatrix}. \tag{1.9}$$

Satz 1: Für die Determinante einer Vandermonde Matrix V_n gilt

$$\det(V_n) = \prod_{1 \le i < j \le n} (k_j - k_i) \tag{1.10}$$

Daraus folgt direkt, daß die Determinante einer Vandermonde Matrix genau dann ungleich Null ist, wenn die k_i voneinander verschieden sind.

Beweis: Induktion über n. Für $n = 1$ ist die Gleichung trivialer Weise erfüllt. Nehmen wir also an, daß die Gleichung für alle Zahlen $< n$ gilt. Wir betrachten die folgende Determinante:

$$f(z) = \det \begin{pmatrix} 1 & k_1 & k_1^2 & \ldots & k_1^{n-1} \\ 1 & k_2 & k_2^2 & \ldots & k_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & k_{n-1} & k_{n-1}^2 & \ldots & k_{n-1}^{n-1} \\ 1 & z & z^2 & \ldots & z^{n-1} \end{pmatrix}. \tag{1.11}$$

Offensichtlich ist f ein Polynom vom Grad $n-1$ mit $f(k_1) = f(k_2) = \cdots = f(k_n) = 0$. Also $f(z) = C_{n-1}(z-k_1)\cdots(z-k_{n-1})$. Mit Expansion nach der letzten Zeile und unter Benutzung der Induktionshypothese erhalten wir

$$C_{n-1} = \det \begin{pmatrix} 1 & k_1 & k_1^2 & \ldots & k_1^{n-2} \\ 1 & k_2 & k_2^2 & \ldots & k_2^{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & k_{n-1} & k_{n-1}^2 & \ldots & k_{n-1}^{n-2} \end{pmatrix} = \prod_{1 \le j < i \le n-1} (k_i - k_j). \tag{1.12}$$

Die Berechnung von f in $z = k_n$ impliziert dann (1.10).

□

Damit dieser Satz gilt, brauchen wir nicht anzunehmen, daß die k_i alle verschieden sind.

Die Inverse einer Vandermonde Matrix kann man auf folgende Weise konstruieren, wobei wir nun voraussetzen wollen, daß die k_i alle voneinander verschieden sind, da die Inverse nur für $\det \neq 0$ existiert: man multipliziert eine Vandermonde Matrix mit einer allgemeinen $n \times n$ - Matrix:

$$\begin{pmatrix} 1 & k_1 & k_1^2 & \ldots & k_1^{n-1} \\ 1 & k_2 & k_2^2 & \ldots & k_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & k_n & k_n^2 & \ldots & k_n^{n-1} \end{pmatrix} \cdot \begin{pmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \ldots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \ldots & a_{nn} \end{pmatrix} \tag{1.13}$$

Das j-te Element der obersten Zeile vom Produkt dieser beiden Matrizen ist

$$a_{1j} + a_{2j}k_1 + a_{3j}k_1^2 + \cdots + a_{nj}k_1^{n-1} =: P_j(k_1). \tag{1.14}$$

Tatsächlich ist die Produktmatrix dann

$$\begin{pmatrix} P_1(k_1) & P_2(k_1) & P_3(k_1) & \dots & P_n(k_1) \\ P_1(k_2) & P_2(k_2) & P_3(k_2) & \dots & P_n(k_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ P_1(k_n) & P_2(k_n) & P_3(k_n) & \dots & P_n(k_n) \end{pmatrix} \tag{1.15}$$

Wählt man die $P_j(z)$ als

$$P_j(z) = \prod_{\substack{1 \le i \le n \\ i \ne j}} \frac{z - k_i}{k_j - k_i} \tag{1.16}$$

so sieht man, daß die Produktmatrix von (1.13) die Identitätsmatrix ist, also sind die Koeffizienten der P_j die Spalten der Inversen der Vandermonde Matrix.

Angenommen, man möchte ein Vandermonde System 1.8 lösen. Dazu definieren wir

$$P(z) = \prod_{1 \le i \le n} (z - k_i). \tag{1.17}$$

Dieses Polynom enthält $n+1$ Terme und der Koeffizient von z^n ist immer 1. Die Polynome der Form $P(z)/(z-k_j)$ können mit synthetischer Division (siehe zum Beispiel [PTVF92], Paragraph 5.8) berechnet werden; es sind die Zähler von $P_j(z)$. Der Wert von $P(z)/(z-k_j)$ in k_j ist gerade der Nenner von $P_j(z)$. Die Berechnung der x_i erfolgt dann so:

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} w_1 \mathrm{Coef}(P_1, z^0) \\ \vdots \\ w_n \mathrm{Coef}(P_1, z^{n-1}) \end{pmatrix} + \cdots + \begin{pmatrix} w_1 \mathrm{Coef}(P_n, z^0) \\ \vdots \\ w_n \mathrm{Coef}(P_n, z^{n-1}) \end{pmatrix} \tag{1.18}$$

Diesen Ansatz kann man auch auf transponierte Vandermonde Systeme wie das folgende anwenden:

$$\begin{aligned} x_1 + x_2 + x_3 + \cdots + x_n &= w_1 \\ k_1x_1 + k_2x_2 + k_3x_3 + \cdots + k_nx_n &= w_2 \\ k_1^2x_1 + k_2^2x_2 + k_3^2x_3 + \cdots + k_n^2x_n &= w_2 \\ &\vdots \\ k_1^{n-1}x_1 + k_2^{n-1}x_2 + k_3^{n-1}x_3 + \cdots + k_n^{n-1}x_n &= w_2 \end{aligned} \tag{1.19}$$

Da das Inverse einer transponierten Matrix gleich dem transponierten des Inversen ist, ergibt sich als Formel für jedes der x_i die folgende Gleichung:

$$x_i = w_1 \cdot \mathrm{Coef}(P_i, z^0) + w_2 \cdot \mathrm{Coef}(P_i, z^1) + \cdots + w_n \cdot \mathrm{Coef}(P_i, z^{n-1}) \tag{1.20}$$

Der nun folgende Algorithmus löst ein solches transponiertes Vandermonde System unter Verwendung der oben beschriebenen Verfahren.

Algorithmus 1: VANDERMONDE.

Eingabe: $(k_1,\dots,k_n) \in \mathbb{Q}$, $(w_1,\dots,w_n) \in \mathbb{Q}$ mit $\sum_{i=1}^{n} x_i k_i^{l-1} = w_l$, $l = 1,\dots,n$.

Ausgabe: $(x_1,\dots,x_n) \in \mathbb{Q}$

Algorithmus:

```
        (c_1, ..., c_n) ← (0, ..., 0)

((x_1, ..., x_n)) = VANDERMONDE((k_1, ..., k_n), (w_1, ..., w_n))

        begin
          if (n == 1)
            then
                x_1 ← w_1
            else
                for (i = 2, ..., n) do
                    xx ← −k_i
                    for (j = (n+1−i), ..., (n−1)) do
                        c_j ← c_j + xx · c_{j+1}
                    od
                    c_n ← c_n + xx
                od
                for (i = 1, ..., n) do
                    xx ← k_i
                    (t, b) ← (1, 1)
                    s ← w_n
                    for (k = n, ..., 2) do
                        b ← c_k + xx · b
                        s ← s + w_{k−1} · b
                        t ← xx · t + b
                    od
                    x_i ← s/t
                od
          fi
          return ((x_1, ..., x_n))
        end
```

2 Resultanten

2.1 Resultanten zweier Polynome

In diesem Abschnitt soll zunächst die Resultante von zwei Polynomen in einer Variablen vorgestellt werden. Dann wird gezeigt, wie man Resultanten zur Elimination von Variablen einsetzen kann und zuletzt folgt ein Satz als Vorbereitung für den nächsten Abschnitt.

Seien $f,g \in \mathbb{K}[x]$ Polynome in einer Unbekannten. Möchte man wissen, ob f und g einen gemeinsamen Faktor haben, d. h. ob ein Polynom $h \in \mathbb{K}[x]$ existiert, so daß gilt: h teilt f und g, so gibt es verschiedene Wege, dies herauszufinden.

Eine Möglichkeit ist die Zerlegung von f und g in irreduzieble Faktoren. Leider ist die Faktorisierung ein unter Umständen sehr zeitaufwendiger Prozeß. Eine andere Möglichkeit ist die Berechung des ggTs von f und g mit Hilfe des Euklidischen Algorithmus. Dieser Algorithmus benötigt Divisionen im Körper $\mathbb{K}$, was man vermeiden möchte.

Eine weiterer Weg, der ohne Divisionen auskommt, ist der folgende.

Lemma 2: Seien $f,g \in \mathbb{K}[x]$ Polynome mit $k := \deg(f) > 0$ und $l := \deg(g) > 0$. f und g haben genau dann einen gemeinsamen Faktor, wenn es Polynome $A,B \in \mathbb{K}[x]$ gibt, so daß gilt:

(i) A und B sind nicht beide gleich Null.

(ii) $\deg(A) < l$ und $\deg(B) < k$.

(iii) $Af + Bg = 0$.

Beweis: Angenommen, f und g haben einen gemeinsamen Faktor $h \in \mathbb{K}[x]$. Dann ist $f = hf_1$ und $g = hg_1$ mit $f_1, g_1 \in \mathbb{K}[x]$. $\deg(f_1) \leq k-1$, $\deg(g_1) \leq l-1$. Dann ist $g_1 \cdot f + (-f_1) \cdot g = g_1 \cdot hf_1 - f_1 \cdot hg_1 = 0$ und $A = g_1$ und $B = -f_1$ haben die geforderten Eigenschaften.

Umgekehrt, angenommen, A und B haben die obigen drei Eigenschaften und ohne Beschränkung der Allgemeinheit sei $B \neq 0$. Haben f und g keinen gemeinsamen Faktor, dann ist ihr ggT 1. Man kann also Polynome $A', B' \in \mathbb{K}[x]$ finden, so daß $A'f + B'g = 1$ ist. Jetzt multipliziert man mit B und benutzt nach (iii) $Bg = -Af$:

$$B = (A'f + B'g)B = A'Bf + B'Bg = A'Bf - B'Af = (A'B - B'A)f$$

Da $B \neq 0$, folgt aus dieser Gleichung, daß $\deg(B) \geq k$, was ein Widerspruch zu (ii) ist. Also muß ein gemeinsamer Faktor mit positivem Grad existieren.

□

Die Aussage von Lemma 2 ist nicht sehr befriedigend, da zu gegebenen f, g zunächst nicht klar ist, ob passende A und B existieren.

Die Idee besteht nun darin, die Gleichung $Af + Bg = 0$ als lineares Gleichungssystem zu betrachten. Man schreibt

$$\begin{aligned} A &= c_0 x^{l-1} + c_1 x^{l-2} \ldots + c_{l-1}, \\ B &= d_0 x^{k-1} + d_1 x^{k-2} \ldots + c_{k-1}, \end{aligned} \tag{2.1}$$

wobei man jetzt die $l+k$ Koeffizienten $c_0, \ldots, c_{k-1}, d_0 \ldots d_{l-1}$ als Unbekannte betrachtet. Ziel ist es also, $c_i, d_j \in \mathbb{K}$, c_i, d_j nicht alle gleich Null, zu finden, so daß

$$Af + Bg = 0 \tag{2.2}$$

gilt. Dies liefert einem die gesuchten Polynome A und B aus Lemma 2.

Um ein lineares Gleichungssystem zu erhalten, schreibt man f und g wie folgt:

$$\begin{aligned} f &= a_0 x^k + a_1 x^{k-1} + \ldots + a_k, \quad a_0 \neq 0, \\ g &= b_0 x^l + b_1 x^{l-1} + \ldots + b_l, \quad b_0 \neq 0, \end{aligned} \tag{2.3}$$

mit $a_i, b_j \in \mathbb{K}$.

Setzt man die Gleichungen für f, g, A und B in die Gleichung (2.2) ein und vergleicht die Koeffizienten von gleichen Potenzen von x, so erhält man das Gleichungssystem

$$\begin{array}{llll} a_0c_0 & + \quad b_0d_0 & = 0 & \text{(Koeffizient von } x^{k+l-1}) \\ a_1c_0 + a_0c_1 & + \quad b_1d_0 + b_0d_1 & = 0 & \text{(Koeffizient von } x^{k+l-2}) \\ & \quad\vdots & & \quad\vdots \\ \sum_{i+j=s} a_ic_j & + \quad \sum_{i+j=s} b_id_j & = 0 & \text{(Koeffizient von } x^{k+l-s}) \\ & \quad\vdots & & \quad\vdots \\ a_kc_{l-1} & + \quad b_ld_{k-1} & = 0 & \text{(Koeffizient von } x^{0}) \end{array} \tag{2.4}$$

mit c_i, d_j als Unbekannte und Koeffizienten $a_i, b_j \in \mathbb{K}$.

Aus der Linearen Algebra ist bekannt, daß dieses Gleichungssystem mit $k+l$ Unbekannten und $k+l$ Gleichungen genau dann eine nichttriviale Lösung besitzt, wenn die Determinante der Koeffizientenmatrix Null ist.

Definition 9: Seien $f,g \in \mathbb{K}[x]$, $k := \deg(f) > 0$ und $l := \deg(g) > 0$. Schreibe f und g als

$$f = a_0x^k + a_1x^{k-1} + \ldots + a_k, \quad a_0 \neq 0,$$
$$g = b_0x^l + b_1x^{l-1} + \ldots + b_l, \quad b_0 \neq 0,$$

mit $a_i, b_j \in \mathbb{K}$. Dann ist die **Sylvestermatrix** von f und g unter Beachtung von x die Koeffizientenmatrix des in (2.4) gegebenen Systems und wird mit $\mathrm{Syl}^x(f,g)$ bezeichnet. $\mathrm{Syl}^x(f,g)$ ist dann durch die folgende $(k+l) \times (k+l)$-Matrix gegeben:

$$\mathrm{Syl}^x(f,g) = \left(\underbrace{\begin{matrix} a_0 & & & \\ a_1 & a_0 & & \\ a_2 & a_1 & \ddots & \\ \vdots & & \ddots & a_0 \\ & \vdots & & a_1 \\ a_k & & & \\ & a_k & & \vdots \\ & & \ddots & \\ & & & a_k \end{matrix}}_{l \text{ Spalten}} \; \underbrace{\begin{matrix} b_0 & & & \\ b_1 & b_0 & & \\ b_2 & b_1 & \ddots & \\ \vdots & & \ddots & b_0 \\ & \vdots & & b_1 \\ b_l & & & \\ & b_l & & \vdots \\ & & \ddots & \\ & & & b_l \end{matrix}}_{k \text{ Spalten}} \right)$$

Die **Resultante** von f und g unter Berücksichtignung von x, $\mathrm{Res}^x(f,g)$, ist die Determinante der Sylvestermatrix, also

$$\mathrm{Res}^x(f,g) = \det(\mathrm{Syl}^x(f,g)).$$

Notation: Wird kein Wert auf die Hervorhebung der Bedeutung von x gelegt, so schreibt man auch $\mathrm{Res}(f,g)$ für die Resultante von f und g.

Beispiel 1: Seien $f,g \in \mathbb{Z}[x]$ mit $f = x^3 + 4x - 1$ und $g = 2x^2 + 3x + 7$. Dann ist

$$\mathrm{Res}^x(f,g) = \det(\mathrm{Syl}^x(f,g)) = \det \left(\underbrace{\begin{matrix} 1 & 0 \\ 0 & 1 \\ 4 & 0 \\ -1 & 4 \\ 0 & -1 \end{matrix}}_{\deg(g)=2} \; \underbrace{\begin{matrix} 2 & 0 & 0 \\ 3 & 2 & 0 \\ 7 & 3 & 2 \\ 0 & 7 & 3 \\ 0 & 0 & 7 \end{matrix}}_{\deg(f)=3} \right) = 216$$

⊣

Mit der obigen Definition erhält man die folgenden Eigenschaften von Resultanten.

Satz 3: Seinen $f,g \in \mathbb{K}[x]$.

(i) $\mathrm{Res}^x(f,g) \in \mathbb{Z}[a_0,\ldots,a_{k-1},b_0,\ldots,b_{l-1}]$.

(ii) $\mathrm{Res}^x(f,g) = 0$ genau dann, wenn f und g einen gemeinsamen Faktor haben.

(iii) Es existieren Polynome A, $B \in \mathbb{K}[x]$, so daß $Af + Bg = \mathrm{Res}^x(f,g)$.

Beweis: (i) folgt aus der Definition der Resultante und der Determinante für eine $s \times s$ - Matrix.
(ii) folgt mit Lemma 2.

(iii) siehe zum Beispiel [CLO97a], Kapitel 3, Paragraph 5, Proposition 9 (Seite 152).

□

Für $\mathbb{K} = \mathbb{C}$ haben f und g genau dann einen gemeinsamen Faktor, wenn beide mindestens eine gemeinsame Nullstelle haben, was genau dann der Fall ist, wenn ihre Resultante gleich Null ist.

Resultanten können auf viele verschiede Arten dargestellt werden. Eine sicherlich sehr bekannte Variante ist die oben definierte Darstellung als Determinante der Sylvestermatrix. Eine andere interessante Darstellungsform soll im weiteren näher erläutert werden.

Seien wieder $f,g \in \mathbb{K}[x]$. Angenommen, die Nullstellen von f und g sind $\mu_1,\dots,\mu_k$ beziehungsweise $\nu_1,\dots,\nu_l$, wobei diese Nullstellen möglicherweise aus einem Unterkörper $\mathbb{K}' \subset \mathbb{K}$ sind. Dann läßt sich die Resultante von f und g durch ein Produkt der Nullstellen und Koeffizienten ausdrücken:

$$\begin{aligned} \operatorname{Res}(f,g) &= a_0^k b_0^l \prod_{i=1}^{k}\prod_{j=1}^{l}(\mu_i - \nu_j) \\ &= a_0^k \prod_{i=1}^{k} g(\mu_j) \\ &= (-1)^{kl} b_0^l \prod_{j=1}^{l} f(\nu_j) \end{aligned} \tag{2.5}$$

Einen Beweis für die Gültigkeit dieser Darstellung findet man z. B. in [Mig92] auf Seite 106.

Bisher wurde immer mit Polynomen aus $\mathbb{K}[x]$ gearbeitet. Jetzt sollen Polynome aus $\mathbb{K}[x,y]$ in *zwei* Variablen x und y betrachtet werden.

Beispiel 2: Seien $f,g \in \mathbb{C}[x,y]$ mit $f = xy - 1$ und $g = x^2 + y^2 - 4$. Dann ist

$$\operatorname{Res}^x(f,g) = \det \begin{pmatrix} y & 0 & 1 \\ -1 & y & 0 \\ 0 & -1 & y^2-4 \end{pmatrix} = y^4 - 4y^2 + 1$$

Daraus folgt nach Lemma 3, daß $A,B \in \mathbb{C}[x,y]$ existieren mit

$$A(xy-1) + B(x^2+y^2-4) = y^4 - 4y^2 + 1.$$

Daraus folgt, daß $\operatorname{Res}^x(f,g) \in (\langle f,g\rangle \cap \mathbb{C}[y]) =: I$, d. h. $\operatorname{Res}^x(f,g)$ liegt im ersten Eleminationsideal I. Also ist $y^4 - 4y^2 + 1 = 0$ für jede gemeinsame Lösung von $f = g = 0$. Lösen der Gleichung $y^4 - 4y^2 + 1 = 0$ liefert also die y-Koordinate der Lösung von $f = g = 0$. ⊣

Man sieht also, daß man Resultanten auch zur Elimination von Variablen verwenden kann. Dies gibt auch der folgenden Satz wieder.

Satz 4: Seien $f,g \in \mathbb{K}[x_1,\dots,x_n]$ mit $\deg_{x_1}(f) > 0$ und $\deg_{x_1}(g) > 0$. Dann gilt:

(i) $\operatorname{Res}^{x_1}(f,g)$ ist im ersten Eliminationsideal $\langle f,g\rangle \cap \mathbb{K}[x_2,\dots,x_n]$ enthalten.

(ii) $\operatorname{Res}^{x_1}(f,g) = 0$ genau dann, wenn f und g einen gemeinsamen Faktor in $\mathbb{K}[x_1,\dots,x_n]$ haben.

Beweis: Da der Satz für den weiteren Verlauf nur von untergeordneter Bedeutung ist, sei auf [CLO97a], Kapitel 3, Paragraph 3, Proposition 1, verwiesen.

□

Als Abschluß dieses Abschnitts wird noch eine Variante von $\mathrm{Res}(f,g)$ betrachtet, die für den folgenden Abschnitt über Macaulay Resultanten von Bedeutung ist. Statt Polynome in einer Variablen werden jetzt *homogene* Polynome aus $\mathbb{K}[x,y]$ betrachtet.

Sind $F,G \in \mathbb{K}[x,y]$ homogene Polynome mit $\deg(F) = k$ und $\deg(G) = l$, so kann man diese schreiben als

$$\begin{aligned} F &= a_0x^k + a_1x^{k-1}y + \ldots + a_ky^k \\ G &= b_0x^l + b_1x^{l-1}y + \ldots + b_ly^l \end{aligned} \tag{2.6}$$

wobei a_0 oder b_0 oder beide gleich Null sein können.

$\mathrm{Res}(F,G)$ sei genauso wie in Definition 9 definiert. Dies ist leicht einsichtig, wenn man f und g aus (2.3) unter Verwendung geeigneter Potenzen von y homogenisiert. So erhält man F und G aus (2.6), also offensichtlich $\mathrm{Res}(F,G) = \mathrm{Res}(f,g)$. Die andere Richtung gestaltet sich etwas schwieriger, da man F und G aus (2.6) dehomogenisieren muß. Dies erreicht man, indem man z. B. $y = 1$ setzt, was allerdings nicht das gewünschte Ergebins liefert, wenn a_0 oder b_0 gleich Null ist. Trotzdem erfüllt die Resultante $\mathrm{Res}(F,G)$ die folgenden grundlegenden Eigenschaften.

Satz 5: Seien $k,l \in \mathbb{N}, k,l \neq 0$.

(i) Es existiert ein Polynom $\mathrm{Res}_{k,l} \in \mathbb{Z}[a_0,\ldots,a_k,b_0,\ldots,b_l]$, so daß $\mathrm{Res}(F,G) = \mathrm{Res}_{k,l}(a_0,\ldots,a_k,b_0,\ldots,b_l)$ für alle F,G wie in (2.6)

(ii) Ist $\mathbb{K} = \mathbb{C}$, dann ist $\mathrm{Res}(F,G) = 0$ genau dann, wenn die Gleichungen $F = G = 0$ eine nichttriviale Lösung $(x,y) \neq (0,0)$ in $\mathbb{K}^2$ haben.

Beweis: (i) ist offensichtlich eine direkte Folgerung aus der Definition der Resultante.
(ii) Ist $(u,v) \in \mathbb{C}^2$ eine nichttriviale Lösung, dann ist auch $(\lambda u, \lambda v) \in \mathbb{C}^2$ eine nichttriviale Lösung für belibige $\lambda \in \mathbb{C}$. Man betrachtet die folgenden drei Fälle:
Fall 1: $a_0 = b_0 = 0$
Die Resultante verschwindet und man erhält die nichttriviale Lösung $(1,0)$.
Fall 2: $a_0b_0 \neq 0$
$\Rightarrow$: Ist $\mathrm{Res}(F,G) = 0$, so dehomogenisiert man F und G, indem man $y = 1$ setzt und erhält die Polynome $f,g \in \mathbb{C}[x]$ mit $\mathrm{Res}(f,g) = 0$. Da $\mathbb{K} = \mathbb{C}$, folgt, daß f und g eine gemeinsame Nullstelle $x = u$ haben müssen. Also ist $(x,y) = (u,1)$ die gesuchte nichttiviale Lösung.
$\Leftarrow$: Hat man die nichtttriviale Lösung (u,v), dann folgt aus der Annahme $a_0b_0 \neq 0$, daß $v \neq 0$. Dann ist $(u/v,1)$ auch eine Lösung, d. h. u/v ist eine gemeinsame Nullstelle der dehomogenisierten Polynome. Damit folgt, daß $\mathrm{Res}(F,G) = 0$ ist.
Fall 3: Entweder $a_0 \neq 0$ oder $b_0 \neq 0$
Siehe [CLO97b], Kapitel 3, Proposition 1.7.

□

2.2 Macaulay Resultanten

2.2.1 Einführung

Die *Macaulay Resultante* für $n+1$ homogene Polynome in $n+1$ Veränderlichen geht auf Macaulays Arbeit [Mac02] von 1902 zurück und wird als die die klassische Variante der multipolynomialen

Resultante angesehen. Aus diesem Grund wird sie oft einfach auch *multipolynomialen Resultante* genannt.

Seien $F_0, \ldots, F_n \in \mathbb{C}[x_0, \ldots, x_n]$ homogene Polynome mit positiven Graden in $n+1$ Variablen. Gesucht ist eine Bedingung, die die Koeffizienten eines Gleichungssystems

$$F_0(x_0, \ldots, x_n) = F_1(x_0, \ldots, x_n) = \cdots = F_n(x_0, \ldots, x_n) = 0 \tag{2.7}$$

erfüllen müssen, damit das System eine nichttriviale Lösung besitzt.

Beispiel 3: Betrachten wir den Fall $n = 3$ und $\deg(F_0) = \deg(F_1) = \deg(F_2) = 1$, das heißt, $F_0, F_1, F_2 \in \mathbb{C}[x_0, x_1, x_2]$ sind lineare homogene Polynome. Diese bilden das Gleichungssystem

$$\begin{aligned}
F_0 &= c_{0,100}x_0 + c_{0,010}x_1 + c_{0,001}x_2 = 0,\\
F_1 &= c_{1,100}x_0 + c_{1,010}x_1 + c_{1,001}x_2 = 0,\\
F_2 &= c_{2,100}x_0 + c_{2,010}x_1 + c_{2,001}x_2 = 0.
\end{aligned} \tag{2.8}$$

Dieses 3×3 – System linearer Gleichungen hat genau dann eine nichttriviale Lösung, wenn die Determinante der Koeffizientenmatrix von (2.8) gleich Null ist, also

$$\det(c_{i,a}) = 0.$$

Man erhält also eine einzige Bedingung für die Existenz einer nichttrivialen Lösung. ⊣

Da die F_i aus (2.7) homogen und von positivem Grad sind, hat das System immer die triviale Lösung $x_0 = x_1 = \ldots = x_n = 0$. Die Lösungen, für die wir uns interessieren, sind die nichttrivialen Lösungen, also aus $\mathbb{P}^n(\mathbb{C})$.

In der Anwendung werden oft nur die Lösungen eines affinen Systems in $\mathbb{C}^n \setminus \{0\}$ benötigt. Deshalb dehomogenisieren wir, indem wir $x_0 = 1$ setzen und definieren:

$$f_i(x_1, \ldots, x_n) = F_i(1, x_1, \ldots, x_n). \tag{2.9}$$

Man beachte, daß f_i maximal den Grad $d_i := \deg(F_i)$ haben kann.

2.2.2 Die Macaulay Resultante

Jedes homogene Polynom F_i läßt sich auch als Summe der einzelnen Terme schreiben:

$$F_i = \sum_{|a|=d_i} c_{i,a}x^a = \sum_{a_0+\cdots+a_n=d_i} c_{i,a_0\ldots a_n} x_0^{a_0} \cdots x_n^{a_n} \tag{2.10}$$

mit $d_i := \deg(F_i)$, $i = 0, \ldots, n$.

Für jeden dieser möglichen Koeffizienten $c_{i,a}$ von F_i führen wir eine Variable $v_{i,a}$ ein und betrachten den Polynomring $\mathbb{Z}[v_{i,a}]$ über diesen Variablen. Ein Polynom $p \in \mathbb{Z}[v_{i,a}]$ bezeichnet man als ein *Polynom in den Koeffizienten* von $F_0, \ldots, F_n$, wobei $p(c_{i,a})$ die Zahl ist, die man durch Ersetzen aller Variablen $v_{i,a}$ durch die entsprechenden Koeffizienten $c_{i,a}$ erhält.

Notation: Oft schreiben wir auch

$$p(F_0, \dots, F_n) := p(c_{i,a}), \tag{2.11}$$

um die Abhängigkeit der Koeffizienten $c_{i,a}$ von den Polynomen $F_0, \dots, F_n$ hervorzuheben.

Mit dieser Notation formulieren wir den folgenden Satz, der die Existenz einer Macaulay Resultante garantiert und eine Hauptaussage zu der gesuchten Bedingung an die Koeffizienten von (2.7) macht.

Satz 6: Seien $F_0, \dots, F_n \in \mathbb{C}[x_0, \dots, x_n]$ homogene Polynome mit positven Graden $d_i := \deg(F_i)$. Dann existiert ein bis auf das Vorzeichen eindeutiges Polynom $\mathrm{Res} \in \mathbb{Z}[v_{i,a}]$ mit den folgenden Eigenschaften:

(i) Das System $F_0 = \cdots = F_n = 0$ hat genau dann eine nichttriviale Lösung, wenn $\mathrm{Res}(F_0, \dots, F_n) = 0$ ist.

(ii) $\mathrm{Res}(x_0^{d_0}, \dots, x_n^{d_n}) = 1$.

(iii) Res ist irreduzibel.

Dieses Polynom Res wird **Macaulay Resultante von** $F_0, \dots, F_n$ genannt.

Der folgende Beweis dieses Satzes liefert gleichzeitig auch eine geometrische Interpretation der Resultante.

Beweis: Die Elemente von $\mathbb{Z}[v_{i,a}]$ sind Polynome im affinene Raum $\mathbb{C}^N$ mit $N = \sum_{i=0}^{n} \binom{n+d_i}{d_i}$. Man betrachtet $x = (x_0, \dots, x_n)$ als die homogenen Koordinaten des komplexen projektiven Raums $\mathbb{P}^n(\mathbb{C})$. Also sind die $(v_{i,a}, x)$ die Koordinaten auf der Produktvarietät $\mathbb{C}^N \times \mathbb{P}^n(\mathbb{C})$. $\mathcal{I}$ bezeichne die Subvarietät von $\mathbb{C}^N \times \mathbb{P}^n(\mathbb{C})$ definiert durch die Gleichung

$$\sum_{a_0+\cdots+a_n=d_i} v_{i,a_0\dots a_n} x_0^{a_0} \cdots x_n^{a_n} = 0, \quad i = 0, 1, \dots, n,$$

auch Inzidenzvarietät genannt. Man beachte, daß $\mathcal{I}$ über $\mathbb{Q}$ definiert ist. Nun betrachtet man die Projektion

$$\begin{aligned} \varphi : \mathbb{C}^N \times \mathbb{P}^n(\mathbb{C}) &\to \mathbb{P}^n(\mathbb{C}) \\ (v_{i,a}, x) &\mapsto x \end{aligned}$$

Dann ist $\varphi(\mathcal{I}) = \mathbb{P}^n(\mathbb{C})$. Genauer, das Urbild $\varphi^{-1}(x)$ eines jeden Punktes $x \in \mathbb{P}^n(\mathbb{C})$ kann mit der Menge $\{v_{i,a} \in \mathbb{C}^N \mid (v_{i,a}, x) \in \mathcal{I}\}$ identifiziert werden. Dies ist ein linearer Subraum der Kodimension $n+1$ in $\mathbb{C}^N$. Es folgt, daß $\mathcal{I}$ eine abgeschlossene und irreduzible Subvarietät der Kodimension n in $\mathbb{C}^N \times \mathbb{P}^n(\mathbb{C})$ ist (siehe z. B. [Sha77], Theorem 8, Abschnitt I.6.3). Also ist $\dim(\mathcal{I}) = N - 1$. Jetzt betrachtet man die Projektion

$$\begin{aligned} \psi : \mathbb{C}^N \times \mathbb{P}^n(\mathbb{C}) &\to \mathbb{C}^N \\ (v_{i,a}, x) &\mapsto v_{i,a} \end{aligned}$$

Mit dem Hauptsatz der Eliminationstheroie (siehe z. B. [CLO97a], Kapitel 8, Paragraph 5) folgt, daß $\psi(\mathcal{I})$ eine irreduzible Subvarietät von $\mathbb{C}^N$ ist, die auch über $\mathbb{Q}$ definiert ist. Jeder Punkt $c \in \mathbb{C}^N$ kann mit einem System von Polynomen $(F_0, \dots, F_n)$ wie in (2.7) identifiziert werden. Dieses

System hat eine Nullstelle ungleich Null genau dann, wenn $c \in \psi(\mathcal{I})$. Für jedes dieser c's hat man die Ungleichung

$$\dim(\psi(\mathcal{I})) \leq \dim(\mathcal{I}) = N - 1 \leq \dim(\psi^{-1}(c)) + \dim(\psi(\mathcal{I})) \tag{2.12}$$

Die beiden Ungleichungen folgen aus Theorem 7, Teil (1) und (2), in [Sha77], Abschnitt I.6.3. Nun wählt man $c = (F_0, \ldots, F_n)$ wie folgt: Seien $F_0, \ldots, F_{n-1}$ wie in (2.7) mit endlich vielen Nullstellen in $\mathbb{P}^n(\mathbb{C})$. F_n wählt man so, daß es genau an einer dieser Nullstellen, z. B. $y \in \mathbb{P}^n(\mathbb{C})$, verschwindet, d. h. $F_n(y) = 0$. Also ist $\psi^{-1}(c) = \{(c, y)\}$ eine nulldimensionale Varietät. Für diesen speziellen Fall gilt für die beiden Ungleichungen in (2.12) Gleichheit. Damit folgt, daß $\dim(\psi(\mathcal{I})) = N - 1$. Bis jetzt wurde gezeigt, daß das Bild von $\mathcal{I}$ unter ψ eine irreduzible über $\mathbb{Z}$ definierte Hyperebene in $\mathbb{C}^N$ ist. Also existiert ein irreduzibles Polynom Res $\in \mathbb{Z}[v_{i,a}]$, eindeutig bis auf das Vorzeichen, dessen Nullstellenmenge $\psi(\mathcal{I})$ entspricht. Mit der obigen Konstruktion genügt dieses Polynom (i) und (iii). (ii) ist trivial.

□

Notation: Legt man Wert auf die Hervorhebung der Grade $d_0, \ldots, d_n$ von $F_0, \ldots, F_n$, so schreibt man auch

$$\mathrm{Res}_{d_0,\ldots,d_n}(F_0, \ldots, F_n) := \mathrm{Res}(F_0, \ldots, F_n).$$

Beispiel 4: Sind $F_0, \ldots, F_n$ lineare homogene Polynome aus $\mathbb{C}[x_0, \ldots, x_n]$, so ist die Resultante gegeben durch

$$\mathrm{Res}_{1,\ldots,1}(F_0, \ldots, F_n) = \det(c_{i,a}).$$

⊣

Jetzt noch einige nützliche Eigenschaften Macaulay Resultanten.
Die Grade $d_0, \ldots, d_n$ von $F_0, \ldots, F_n$ seien positiv und fest gewählt.

Zunächst ein Satz über den Grad einer Resultante.

Satz 7:

(i) Sei j fest gewählt, $0 \leq j \leq n$ und $|a| = d_j$. Dann ist Res $\in \mathbb{Z}_j[v_{i,a}]$, homogen in $v_{j,a}$ und vom Grad $d_0 \cdots d_{j-1} d_{j+1} \cdots d_n$, d. h.

$$\mathrm{Res}(F_0, \ldots, \lambda F_j, \ldots, F_n) = \lambda^{d_0 \cdots d_{j-1} d_{j+1} \cdots d_n} \mathrm{Res}(F_0, \ldots, F_j, \ldots, F_n).$$

(ii) Für Res $\in \mathbb{Z}[v_{i,a}]$ gilt $\deg(\mathrm{Res}) = \sum_{i=0}^n d_0 \cdots d_{i-1} d_{i+1} \cdots d_n$.

Beweis: Ein Beweis kann in [Jou91], Paragraph 2, oder [GKZ94], Kapitel 13, gefunden werden.

Als nächstes ein Satz über die Eigenschaften von Symmetrie und Multiplizität von Resultanten.

Satz 8:

(i) Ist $i < j$, dann gilt

$$\mathrm{Res}(F_0, \ldots, F_i, \ldots, F_j, \ldots, F_n) = (-1)^{d_0 \cdots d_n} \mathrm{Res}(F_0, \ldots, F_j, \ldots, F_i, \ldots, F_n).$$

(ii) Ist $F_j = F_j' F_j''$ das Produkt homogener Polynome F_j' und F_j'' vom Grad d_j' beziehungsweise d_j'', so gilt

$$\mathrm{Res}(F_0, \ldots, F_j, \ldots, F_n) = \mathrm{Res}(F_0, \ldots, F_j', \ldots, F_n) \cdot \mathrm{Res}(F_0, \ldots, F_j'', \ldots, F_n).$$

Beweis: Siehe Paragraph 5 in [Jou91] für den ersten Teil des Satzes und [CLO97b], Seite 88, für den zweiten Teil.

2.2.3 Konstruktion der Macaulay Resultante

Seien $n+1$ homogene Polynome

$$F_0, \ldots, F_n \in \mathbb{C}[x_0, \ldots, x_n] \tag{2.13}$$

mit positiven Graden $d_i := \deg(F_i)$, $0 \leq i \leq n$, gegeben, deren Resultante bestimmt werden soll.

Definition 10: Der **kritische Grad** d der Polynome $F_0, \ldots, F_n$ sei die Summe

$$d := (d_0 - 1) + (d_1 - 1) + \cdots + (d_n - 1) + 1 = \sum_{i=0}^{n} d_i - n.$$

Monome $x_0^{a_0} x_1^{a_1} \cdots x_n^{a_n}$ mit $\sum_{i=0}^{n} a_i = d$ haben die folgende für uns interessante Eigenschaft.

Lemma 9: Für jedes Monom $x^a = x_0^{a_0} x_1^{a_1} \cdots x_n^{a_n}$ mit $\deg(x^a) = \sum_{i=0}^{n} a_i = d$, d wie oben, gibt es mindestens ein $i \in [0, \ldots, n]$, so daß $x_i^{d_i} | x_0^{a_0} x_1^{a_1} \cdots x_n^{a_n}$. d ist die kleinste Zahl, für die dies gilt.

Beweis: Angenommen, für alle $i = 0, \ldots, n$ existiert kein $x_i^{d_i}$, welches $x_0^{a_0} x_1^{a_1} \cdots x_n^{a_n}$ teilt. Daraus folgt, daß $0 \leq a_j \leq d_j - 1$ für $j = 0, \ldots, n$, denn sonst würde ein $j \in [0, \ldots, n]$ existieren mit $x_i^{d_i} | x_i^{a_i}$. Mit $d = (d_0 - 1) + (d_1 - 1) + \cdots + (d_n - 1) + 1 = a_0 + a_i + \cdots + a_n$ folgt aber, daß mindestens ein $a_j \geq d_j$ sein muß.

□

Jetzt betrachten wir die Menge

$$\mathrm{MON}_d := \{x^a \mid |a| = d\} = \{x_0^{a_0} x_1^{a_1} \cdots x_n^{a_n} \mid a_0 + a_i + \cdots + a_n = d\} \tag{2.14}$$

aller Monome mit Grad d und zerlegen sie in $n+1$ Teilmengen:

$$\begin{aligned}
S_0 &= \{x^a \mid |a| = d,\ x_0^{d_0} \text{ teilt } x^a\} \\
S_1 &= \{x^a \mid |a| = d,\ x_0^{d_0} \text{ teilt nicht } x^a,\ x_1^{d_1} \text{ teilt } x^a\} \\
&\ \vdots \\
S_n &= \{x^a \mid |a| = d,\ x_0^{d_0}, \ldots, x_{n-1}^{d_{n-1}} \text{ teilen nicht } x^a,\ x_n^{d_n} \text{ teilt } x^a\}
\end{aligned} \tag{2.15}$$

Man erhält so $n+1$ paarweise disjunkte Teilmengen $S_0, \ldots, S_n$, und es gilt

$$S_0 \cup S_1 \cup \cdots \cup S_n = \mathrm{MON}_d$$

wegen Lemma 9. Wir beobachten, daß wir für jedes $x^a \in S_i$ auch

$$x^a = x_i^{d_i} \cdot \frac{x^a}{x_i^{d_i}} \tag{2.16}$$

schreiben können und das $x^a/x_i^{d_i}$ ein Monom vom Grad $d-d_i$ ist, da $x^a \in S_i$.

Lemma 10: Die Kardinalität von S_n entspricht $|S_n| = d_0 \cdot d_1 \cdots d_{n-1}$.

Beweis: Der Beweis folgt direkt aus der Konstruktion der Menge S_n. Seien $a_0, \ldots, a_{n-1}$ mit $0 \leq a_i \leq d_i - 1$ gegeben. Dann gilt $d = \sum_{i=0}^{n} d_i - n$ und $a_0 + a_1 + \cdots + a_{n-1} = d - a_n$. Damit existiert ein eindeutiges $a_n \geq d_n$ so daß $x_0^{a_0} x_1^{a_1} \cdots x_n^{a_n} \in S_n$.

□

Jetzt betrachten wir das folgende Gleichungssystem mit $\binom{n+d}{d}$ Gleichungen:

$$\begin{aligned}
\frac{x^a}{x_0^{d_0}} F_0 &= 0 \quad \text{für alle } x^a \in S_0 \\
\frac{x^a}{x_1^{d_1}} F_1 &= 0 \quad \text{für alle } x^a \in S_1 \\
&\vdots \\
\frac{x^a}{x_n^{d_n}} F_n &= 0 \quad \text{für alle } x^a \in S_n
\end{aligned} \tag{2.17}$$

Da F_i den Grad d_i hat, folgt, daß $x^a/x_i^{d_i} F_i$ homogen vom Grad d ist. Also kann jedes Polynom auf der linken Seite in (2.17) als Linearkombination der $\binom{n+d}{d}$ Monome vom Grad d dargestellt werden.

Definition 11: Die **Macaulay Resultantenmatrix** $\mathcal{DM}_n$ von (2.13) ist eine quadratische Matrix, deren Einträge die Koeffizienten des Gleichungssystems (2.17) sind. Die Zeilen und Spalten der Matrix M_n werden duch die Elemente der Menge $S_0 \cup S_1 \cup \cdots \cup S_n$ indiziert. Die Determinante dieser Koeffizientenmatrix wird mit $\mathcal{DD}_n$ bezeichnet.

Beispiel 5: Seien zum Beispiel die folgenden homogenen Polynome $F_0, F_1, F_2 \in \mathbb{K}[x,y,z]$ gegeben:

$$\begin{aligned}
F_0 &= a_1x + a_2y + a_3z = 0 \\
F_1 &= b_1x + b_2y + b_3z = 0 \\
F_2 &= c_1x^2 + c_2xy + c_3xz + c_4y^2 + c_5yz + c_6z^2 = 0
\end{aligned}$$

Dann ist $(d_0, d_1, d_2) = (1,1,2)$ und damit $d = \sum_{i=0}^{2} d_i - 2 = 2$. Die Menge $\mathrm{MON}_2 = \{x^2, xy, xz, y^2, yz, z^2\}$ besteht also aus allen Monomen vom Grad 2. Die disjunkte Zerlegung in S_0, S_1 und S_2 ergibt sich wie folgt:

$$S_0 = \{x^2, xy, xz\}, \quad S_1 = \{y^2, yz\}, \quad S_2 = \{z^2\}$$

Die Menge S_2 hat, nach Lemma 10, die Kardinalität $d_0 \cdot d_1 = 1$.

Man erhält das folgende Gleichungssystem mit $\binom{2+2}{2} = 6$ Gleichungen:

$$\begin{aligned}
&\frac{x^2}{x}F_0 = xF_0 = a_1x^2 + a_2xy + a_3xz = 0\\
&\frac{xy}{x}F_0 = yF_0 = a_1xy + a_2y^2 + a_3yz = 0\\
&\frac{xz}{x}F_0 = zF_0 = a_1xz + a_2yz + a_3z^2 = 0\\
&\frac{y^2}{y}F_1 = yF_1 = b_1xy + b_2y^2 + b_3yz = 0\\
&\frac{yz}{y}F_1 = zF_1 = b_1xz + b_2yz + b_3z^2 = 0\\
&\frac{z^2}{z^2}F_2 = F_2 = c_1x^2 + c_2xy + c_3xz + c_4y^2 + c_5yz + c_6z^2 = 0
\end{aligned}$$

Die Koeffizienten dieses Gleichungssystems bilden die folgende 6×6 – Matrix $\mathcal{DM}_2$, deren Determinante $\mathcal{DD}_2$ ist:

$$\mathcal{DD}_2 = \det(\mathcal{DM}_2) = \det \begin{array}{c} \begin{matrix} x^2 & xy & xz & y^2 & yz & z^2 \end{matrix} \\ \begin{pmatrix} a_1 & a_2 & a_3 & 0 & 0 & 0 \\ 0 & a_1 & 0 & a_2 & a_3 & 0 \\ 0 & 0 & a_1 & 0 & a_2 & a_3 \\ 0 & b_1 & 0 & b_2 & b_3 & 0 \\ 0 & 0 & b_1 & 0 & b_2 & b_3 \\ c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{pmatrix} \end{array} \begin{array}{c} \\ xF_0 \\ yF_0 \\ zF_0 \\ yF_1 \\ zF_1 \\ F_2 \end{array} \tag{2.18}$$

⊣

Satz 11: Die **Macaulay Resultantenmatrix** $\mathcal{DM}_n$ nach Definition 11 ist wohldefiniert und kann mit dem oben beschriebenen Verfahren konstruiert werden. $\mathcal{DM}_n$ ist quadratisch und hat die Dimension $\binom{n+d}{d}$.

2.2.4 Eigenschaften der Macaulay Resultante

Nun zu einigen Eigenschaften von $\mathcal{DD}_n$. Da $\mathcal{DD}_n$ die Determinante der Koeffizientenmatrix von (2.17) ist, kann man $\mathcal{DD}_n$ als ein Polynom in den Koeffizienten der F_i auffassen. Ausserdem kann man einige Aussagen über den Grad von $\mathcal{DD}_n$ machen.

Lemma 12: Mit den obigen Notationen ist die Determinante $\mathcal{DD}_n$ ein Polynom ungleich Null in $\mathbb{Z}[v_{i,a}]$ und homogen vom Grad $|S_i|$ in den Koeffizienten von F_i.

Beweis: Aus der Konstruktion folgt offensichtlich, daß $\mathcal{DD}_n \in \mathbb{Z}[v_{i,a}]$. Setzt man $F_i := x_i^{d_i}$ in der obigen Konstruktion, so ist $\mathcal{DM}_n$ die Identitätsmatrix. Also ist $\mathcal{DD}_n(F_0, \ldots, F_n) = 1$ und damit ist $\mathcal{DD}_n$ ungleich dem Null-Polynom. Direkt aus der Konstruktion folgt wiederum, daß $\mathcal{DD}_n$ homogen vom

Grad $|S_i|$ in den Koeffizienten von F_i ist.

□

Satz 13: $\mathcal{DD}_n \in \langle \text{Res} \rangle$, das heißt, die Resultante Res teilt $\mathcal{DD}_n$.

Beweis: Jede Nullstelle u von Res entspricht einem Gleichungssystem $f_1(y) = \ldots = f_n(y) = 0$ mit einer Lösung $y \in \mathbb{P}^{n-1}(\mathbb{C})$. Substituiert man in (2.17) x durch y und nimmt an, daß $u \neq 0$ ist, so sieht man, daß die Koeffizientenmatrix von (2.17) einen Vektor ungleich Null in ihrem Kern enthält. Daraus folgt, daß $\mathcal{DD}_n = 0$ in u. Also verschwindet $\mathcal{DD}_n$, wann immer Res verschwindet. Da Res irreduzibel ist (Satz 6), folgt Res teilt $\mathcal{DD}_n$.

□

Mit diesem Satz kann man $\mathcal{DD}_n$ auch so schreiben:

$$\mathcal{DD}_n = \text{Res} \cdot E_n \tag{2.19}$$

Dieser „Faktor“ E_n ist unabhängig von den Koeffizienten von F_n.

Satz 14: E_n aus (2.19) ist ein ganzzahliges Polynom in den Koeffizienten von $\overline{F}_0, \ldots, \overline{F}_{n-1}$, wobei $\overline{F}_i = F_i(0, x_1, \ldots, x_n)$.

Beweis: Da $\mathcal{DD}_n$ eine Determinante ist, ist es wie Res ein Polynom in $\mathbb{Z}[v_{i,a}]$ sein. Der „Faktor“ E_n muß also in $\mathbb{Q}[v_{i,a}]$ liegen, da man beim Teilen von $\mathcal{DD}_n$ durch Res maximal rationale Koeffizienten erhält. Da Res in $\mathbb{Z}[v_{i,a}]$ irreduzibel ist, folgt $E_n \in \mathbb{Z}[v_{i,a}]$. Da $\mathcal{DD}_n = \text{Res} \cdot E_n$ homogen in den Koeffizienten von F_n ist, folgt, daß Res und E_n ebenfalls homogen in diesen Koeffizienten ist. Mit Satz 7 haben sowohl Res als auch D_n den Grad $d_0 \cdots d_{n-1}$ in den Koeffizienten von F_n. Damit folgt direkt, daß E_n den Grad 0 in diesen Koeffizienten hat, es also nur von den Koeffizienten von $F_0, \ldots, F_{n-1}$ abhängig ist.

Bleibt noch zu zeigen, daß E_n nur von den Koeffizienten von $\overline{F}_0, \ldots, \overline{F}_{n-1}$ abhängig ist, daß heißt, die Koeffizienten von Termen von $\overline{F}_0, \ldots, \overline{F}_{n-1}$, in denen x_n einen positiven Grad hat, treten nicht in E_n auf. Um dies zu zeigen, wollen wir Macaulays Vorgehensweise aus [Mac02] verwenden. Wir sagen, daß *Gewicht* von $v_{i,a}$ sei der Exponent a_n von x_n, wobei $a_n = (a_{n1}, \ldots, a_{nn})$. Dann ist das Gewicht eines Monoms in $v_{i,a}$, zum Beispiel $v_{i1,a_1}^{m_1} \cdots v_{il,a_l}^{m_l}$, definiert durch die Summe der Gewichte der einzelnen v_{ij,a_j} multipliziert mit dem entsprechenden Exponenten. Ein Polynom in $v_{i,a}$ wird *isobarisch* genannt, wenn alle Terme des Polynoms das gleiche Gewicht haben.

Man kann zeigen, daß jeder Term in $\mathcal{DD}_n$ das Gewicht $d_0 \cdots d_n$ hat, $\mathcal{DD}_n$ also isobarisch ist. Mit $\mathcal{DD}_n = \text{Res} \cdot E_n$ sind dann Res und E_n ebenfalls isobarisch und das Gewicht von $\mathcal{DD}_n$ entspricht der Summe der Gewichte von Res und E_n. Also genügt es zu zeigen, daß E_n das Gewicht 0 hat. Beachte, daß $v_{0,a_0}, \ldots, v_{n-1,a_{n-1}}$ Gewicht 0 haben, v_{n,a_n} hingegen Gewicht d_n. Mit den Sätzen 6 und 7 folgt, daß einer der Terme in Res gleich

$$\pm v_{0,a_0}^{d_1 \cdots d_n} v_{1,a_1}^{d_0 \cdot d_2 \cdots d_n} \cdots v_{n,a_n}^{d_0 \cdots d_{n-1}}$$

ist. Dieser Term hat Gewicht $d_0 \cdots d_n$, womit folgt, daß das Gewicht von Res ebenfalls $d_0 \cdots d_n$ ist. Oben haben wir gezeigt, daß $\mathcal{DD}_n$ das gleiche Gewicht hat, womit sofort folgt, daß E_n wie gewünscht Gewicht 0 hat.

□

Die Mengen $S_0, \ldots, S_n$ und damit die Determinante $\mathcal{DD}_n$ hängen von der Anordnung der Variablen $x_0, \ldots, x_n$ ab. Die Notation $\mathcal{DD}_n$ hebt hervor, daß die Variable x_n in der Anordnung als letzte auftritt. Wählt man i fest, $0 \leq i \leq n-1$, und ordnet die Variablen so an, daß x_i als letzte auftritt, also etwa $x_0, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n, x_i$, so erhält man leicht unterschiedliche Mengen $S_0, \ldots, S_n$ und somit ein unterschiedliches Gleichungssystem (2.17). Die Determinante der Koeffizientenmatrix eines solchen Systems bezeichen wir mit $\mathcal{DD}_i$. Natürlich gibt es zahlreiche verschiedene Möglichkeiten, die Variablen so anzuordnen, daß x_i als letzte Variable auftritt. Wir wählen daraus eine beliebige Anordnung zur Berechung von $\mathcal{DD}_i$.

Beispiel 6: Seien die Polynome $F_0, F_1, F_2 \in \mathbb{K}[x, y, z]$ wie im letzten Beispiel gegeben.

Mit der Anordnung x, z, y der Variablen erhält man die folgenden Mengen S_0, S_1 und S_2:

$$S_0 = \{x^2, xz, xy\}, \ \ S_2 = \{z^2, yz\}, \ \ S_1 = \{y^2\}$$

Die Determinante $\mathcal{DD}_1$ entspricht dann der Determinante der Koeffizientenmatrix des Gleichungssystems (2.17) für die obigen Mengen S_0, S_1 und S_2. ⊣

Korollar 15: Die Determinante $\mathcal{DD}_i$ ist homogen vom Grad $|S_j|$ in den Koeffizienten von F_j und insbesondere homogen vom Grad $d_0 \cdots d_{i-1} \cdot d_{i+1} \cdots d_n$ in den Koeffizienten von F_i.

Beweis: Das Korollar folgt direkt aus Lemma 12.

□

Nun zu einer klassischen Formulierung der Resultante, die auch gleich einen ersten Ansatz zur Berechnung derselben gibt.

Satz 16: Die Resultante ist der größte gemeinsamte Teiler der Polynome $\mathcal{DD}_0, \ldots, \mathcal{DD}_n$ im Ring $\mathbb{Z}[v_{i,a}]$, also $\mathrm{Res}(F_0, \ldots, F_n) \ = \ \pm\mathrm{GCD}(\mathcal{DD}_0, \ldots, \mathcal{DD}_n)$

Die Berechnung des GCD macht hier natürlich nur für Polynome $F_0, \ldots, F_n$ mit symbolischen Koeffizienten Sinn.

Beweis: Sei G der größte gemeinsame Teiler von $\mathcal{DD}_0, \ldots, \mathcal{DD}_n$. Nach Korollar 15 ist der Grad von $\mathcal{DD}_i$ in den Koeffizienten von F_i gleich $d_0 \cdots d_{i-1} \cdot d_{i+1} \cdots d_n$. Also hat G maximal den Grad $d_0 \cdots d_{i-1} \cdot d_{i+1} \cdots d_n$ in den Koeffizienten von F_i für $i = 0, \ldots, n$. Andererseits hat Res nach Lemma 10 exakt den Grad $d_0 \cdots d_{i-1} \cdot d_{i+1} \cdots d_n$ in den Koeffizienten von F_i. Nach Satz 13 wird $\mathcal{DD}_n$ von Res geteilt, also sind Res und G gleich bis auf einen konstanten rationalen Faktor.

□

Diese Formulierung der Resultante besticht durch seine Einfachheit, erweist sich in der Praxis aber als nicht sehr handlich, da die Berechnung des GCD sehr aufwendig sein kann und man außerdem die Determinanten aller möglichen Resultantenmatizen $\mathcal{DM}_0, \ldots, \mathcal{DM}_n$ berechnen muß.

Auf F. S. Macaulay geht die Darstellung der Resultante als Quotient zweier Determinanten zurück. In seiner Arbeit [Mac02] von 1902 zeigte er, daß der „Faktor“ aus Satz 14 tatsächlich ein Minor der Resultantenmatrix $\mathcal{DM}_n$ ist. Um diesen Minor zu bestimmen, muß man wissen, welche Zeilen und Spalten der Matrix $\mathcal{DM}_n$ gestrichen werden.

Definition 12:

(i) Ein Monom x^a vom Grad $d = |a|$ heißt **reduziert**, wenn für genau ein i, $0 \leq i \leq n$, gilt, daß $x_i^{d_i}$ das Monom x^a teilt.

(ii) Streicht man alle zu reduzierten Monomen x^a gehörigen Zeilen und Spalten der Koeffizientenmatrix M_n von (2.17), so erhält man eine quadratische Submatrix $\mathcal{DM}'_n$ von $\mathcal{DM}_n$.

Beispiel 7: Im Beispiel (5) sind alle Monome bis auf xy reduzierte Monome. Damit erhält man die Submatrix $\mathcal{DD}'_2 = a_1$, indem man alles bis auf die 2. Zeile und 2. Spalte der Matrix (2.18) löscht. $\dashv$

Korollar 17: Die Anzahl der reduzierten Monome x^a in MON_d ist

$$\sum_{j=0}^{n} d_0 \cdots d_{j-1} d_{j+1} \cdots d_n.$$

Beweis: Siehe [CLO97b].

Nach F. S. Macaulay entspricht der „Faktor“ aus Satz 14 bis auf das Vorzeichen genau der Determinante $\mathcal{DD}'_n$ der Submatrix $\mathcal{DM}'_n$. Daraus ergibt sich die Formulierung für die Resultante als Quotient zweier Determinanten.

Satz 18: Seien $F_0, \ldots, F_n \in \mathbb{K}[x_0, \ldots, x_n]$ homogene Polynome und sei $\mathcal{DD}'_n \neq 0$. Dann ist die Resultante von $F_0, \ldots, F_n$ gegeben durch

$$\text{Res}_{d_0,\ldots,d_n}(F_0, \ldots, F_n) \;=\; \pm \frac{\mathcal{DD}_n}{\mathcal{DD}'_n} = \pm \frac{\det(\mathcal{DM}_n)}{\det(\mathcal{DM}'_n)} \tag{2.20}$$

Beweis: Einen Beweis für diesen Satz findet man in [Mac02].

2.2.5 Verallgemeinerte charakteristische Polynome

Unglücklicherweise kommt es häufig vor, daß der Nenner des Quotienten in (2.20) gleich Null ist, d. h. die Determinante $\mathcal{DD}'_{i_n} = 0$ ist. Canny beschreibt in seiner Arbeit über die „Generalized Characteristical Polynoms“ [Can87] eine Vorgehensweise, mit der man dies umgehen kann.

Seien $F_0, \ldots, F_n \in \mathbb{Z}[x_0, \ldots, x_n]$. Die Determinanten $\mathcal{DD}_n$ und $\mathcal{DD}'_n$ aus Satz 18 wurden aus Matrizen erzeugt, die mit $\mathcal{DM}_n$ und $\mathcal{DM}'_n$ bezeichnet wurden.

Tritt der Fall $\det(\mathcal{DM}') = 0$ ein, so führt man nach Cannys Methode eine neue Variable s ein und betrachtet jetzt die Resultante

$$\text{Res}(F_0 - s x_0^{d_0}, F_1 - s x_1^{d_1}, \ldots, F_n - s x_n^{d_n}). \tag{2.21}$$

Also ist

$$\text{Res}(F_0 - s x_0^{d_0}, \ldots, F_n - s x_n^{d_n}) = \frac{\det(\mathcal{DM}_n - sI)}{\det(\mathcal{DM}'_n - sI)}, \tag{2.22}$$

wobei I die Identitätsmatrix ist, da $\det(\mathcal{DM}'_n - sI) \neq 0$ als charakteristisches Polynom von $\mathcal{DM}'_n$. Damit folgt, daß $\text{Res}_{d_0,\ldots,d_n}(F_0, \ldots, F_n)$ der konstante, von s unabhängige Faktor, des Polynoms (2.22) ist.

2.2.6 Generischer linearer Koordinatenwechsel

Für verschiedene „degenerierte“ Macaulay Gleichungssysteme empfiehlt sich ein „zufälliger“ linearer Koordinatenwechsel, um die Stabilität der bei der Berechung der Resultante benutzten Verfahren zu gewährleisten ([CLO97b], Paragraph 3.5, und [EMW98], Paragraph 5.3).

Man betrachtet zunächst das originale Macaulay Gleichungssystem G(X), definiert auf der Menge X der Variablen, und konstruiert ein neues Macaulay Gleichungssystem G'(X') mit einer neuen Menge X' von Variablen. Dabei ist die Beziehung zwischen X und X' durch die folgende lineare Transformation gegeben:

$$[X'] = [M][X] \leftarrow G(X) = G'(X'). \tag{2.23}$$

M ist eine zufällig gewählte Rotationsmatrix. Man löst das neue System G'(X') und erhält die Nullstellenmenge U'. Daraus berechnet man durch Invertieren der Matrix M die Nullstellenmenge U des originalen Systems G(X):

$$[U] = [M]^{-1}[U']. \tag{2.24}$$

2.3 Sparse Resultanten

Als Einführung zunächst ein paar allgemeine Worte zur sparsen Resultante. Im darauf folgenden Abschnitt werden die benötigten Hilfsmittel wie Minkowski-Summen und gemischte Volumen eingeführt, die wir zur Definition der sparsen Resultante benötigen. Der daran anschließende Abschnitt über die polyhedrale Subdivision stellt schießlich die zur Konstruktion der sparsen Resultantenmatrix benötigten Materialien bereit.

2.3.1 Einführung

Der folgende Ansatz zur Konstruktion einer Resultante berücksichtigt die kombinatorische Struktur der gegebenen Polynome eines Systems. Anstatt die Polynome anhand ihres Grades zu charakterisieren, werden die *Newton Polytope* der Polynome betrachtet, die deren Struktur wiedergeben. Das Ergebnis des Ansatzes wird die *sparse Resultante* sein, die die im letzten Abschnitt eingeführte klassische *Macaulay Resultante* verallgemeinert und die durch die Newton Polytope der Polynome gegebene Struktur ausnutzt. Der Ausdruck *sparse* soll hier keineswegs irgendwelche Anforderungen an die Eingabepolynome implizieren, sondern soll vielmehr verdeutlichen, daß der Grad der Resultante proportional zu der „sparseheit“ der Eingabepolynome ist.

Die Grundlagen für die *sparse Resultante* wurden in frühen 90'ern von Gelfand, Kapranov und Zelevinski [GKZ94] gelegt. Aufbauend darauf entwickelten unter anderen J. F. Canny, I. Z. Emiris und D. Manocha in diversen Publikationen ([CE95, CE93, BOT88]) verschiedene Algorithmen zur Konstruktion der *sparsen Resultante*. Das hier vorgestellte Konstruktionsverfahren basiert schließlich auf einer Arbeit von Canny und Emiris [CE97b] aus dem Jahre 1997.

Bei der Formulierung der Macaulay Resultante im letzten Abschnitt ergab sich aus der Betrachtung der Lösungsmenge in $\mathbb{P}^n(\mathbb{C})$ das Problem, daß möglicherweise projektive Lösungen auftreten, die keinen signifikanten Bezug zur affinen Problemstellung haben. Die Anzahl der projektiven Lösungen in $\mathbb{P}^n(\mathbb{C})$ liegt dann typischerweise über der Anzahl der gesuchten affinen Lösungen in $\mathbb{C}^n$ und man muß durch Einsetzen und Ausrechnen herausfinden, welche der gefundenen Lösungen den gesuchten entsprechen.

Da wir nur an den affinen Lösungen interessiert sind, richten wir bei der Formulierung der sparsen Resultante unsere Aufmerksamkeit ganz auf Lösungen in $(\mathbb{C}^*)^n$ mit $\mathbb{C}^* := \mathbb{C}\backslash\{0\} \subset \mathbb{C} \subset \mathbb{P}(\mathbb{C})$. Unter diesen Umständen können wir auch gleich den allgemeineren Fall betrachten, bei dem die Polynome des zu lösenden polynomialen Systems Laurent-Polynome sind, d. h.

$$f_0, f_1, \ldots, f_n \in \mathbb{C}[x_1, x_1^{-1}, \ldots, x_n, x_n^{-1}]. \tag{2.25}$$

Das polynomiale System, an dessen Lösung wir interessiert sind, ist dann

$$f_0 = f_1 = \ldots = f_n = 0. \tag{2.26}$$

Entspricht μ_i der Anzahl der Koeffizienten von f_i ungleich Null, dann läßt sich jedes Polynom f_i auch schreiben als

$$f_i = \sum_{j=1}^{\mu_i} c_{ij} x^{a_{ij}}, \qquad c_{ij} \neq 0, \qquad i = 0, \ldots, n. \tag{2.27}$$

2.3.2 Minkowski-Summen und Gemischte Volumina

Definition 13:

(i) Die endliche Menge $\mathcal{A}_i = \{a_{i1}, \ldots, a_{i\mu_i}\} \subset \mathbb{Z}^n$ aller Exponenten der Monome von f_i wird als der **Träger** von f_i bezeichnet, kurz $\mathrm{tr}(f_i) := \mathcal{A}_i$, wobei $|\,\mathrm{tr}(f_i)| = \mu_i = |\mathcal{A}_i|$.

(ii) Das **Newton Polytop** $Q_i \subset \mathbb{R}^n$ von f_i ist die konvexe Hülle von $\mathcal{A}_i$, d. h. $Q_i := \mathrm{Conv}(\mathcal{A}_i)$. Die **Menge der Eckpunkte** von Q_i sei $E_i := \{a_{i1}, \ldots, a_{im_i}\} \subset \mathcal{A}_i$ mit $m_i \leq \mu_i$.

(iii) Sind alle Träger $\mathcal{A}_i$, $i = 0, \ldots, n$, eines polynomialen Systems $f_0 = f_1 = \ldots = f_n = 0$ gleich, so bezeichnet man dieses System als **ungemischt**, sonst als **gemischt**.

Ein Programm in der Sprache C zur Berechnung konvexer Hüllen von Newton Polytopen findet man unter [ERMM93]. Der dort implementierte recht umfangreiche *Beneath-Beyond-Algorithmus* aus dem Bereich der kombinatorischen Geometrie wird in [Ede90], Kapitel 8, näher beschrieben.

Beispiel 8: Seien $f_0, f_1, f_2 \in \mathbb{Z}[x, y]$. Wir betrachten

$$\begin{aligned}
f_0 &= 4x^5y^3 + 3x^4y^5 + 2x^3 + xy + 8y^4 + 5\\
f_1 &= 3x^5y^3 + 2x^4y^2 + 3x^3y^5 + x^3 + 3x^2y^4 + x^2y + xy^3 + xy\\
f_2 &= 6x^5y^3 + 7x^4y^5 + 4x^3y + x^2y^4 + 4xy^2.
\end{aligned}$$

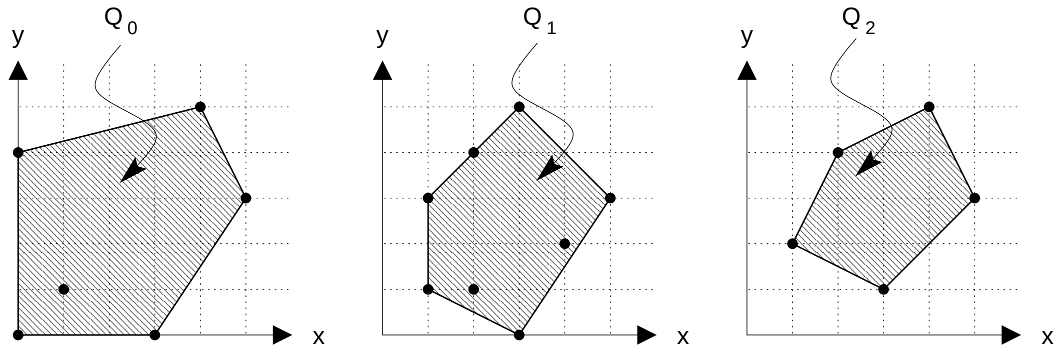

Abbildung 2.1: Die Newton Polytope Q_0, Q_1, Q_2 und die Punkte $a_{ij} \in \mathcal{A}_i$.

Die Träger von f_0, f_1, f_2 sind die Mengen

$$\begin{aligned}
\mathcal{A}_0 &= \mathrm{tr}(f_0) = \{(5,3),(4,5),(3,0),(1,1),(0,4),(0,0)\},\\
\mathcal{A}_1 &= \mathrm{tr}(f_1) = \{(5,3),(4,2),(3,5),(3,0),(2,4),(2,1),(1,3),(1,1)\},\\
\mathcal{A}_2 &= \mathrm{tr}(f_2) = \{(5,3),(4,5),(3,0),(2,4),(1,2)\}.
\end{aligned}$$

Die Mengen $\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2$ und die dazugehörigen Newton Polytope Q_0, Q_1, Q_2 sind in Abbildung 2.1 dargestellt. $\dashv$

Die Newton Polytope übertragen das Problem von der algebraischen auf eine kombinatorisch-geometrische Sichtweise, weshalb wir jetzt einige Begriffe aus der Theorie der Polytope benötigen. Eine umfangreichere Einführung in dieses Thema findet man zum Beispiel in [CLO97b], Kapitel 7 oder in [Zie95]. Zunächst zwei Operationen auf Polytopen, die wir aus den Vektorraum-Strukturen von $\mathbb{R}^n$ ableiten können.

Definition 14: Seien $A, B \subset \mathbb{R}^n$ zwei Polytope und $\lambda \in \mathbb{R}_{\geq 0}$. Die **Minkowski-Summe** von A und B, $A+B$, ist durch

$$A+B := \{a+b \mid a \in A, b \in B\}$$

definiert, wobei $a+b$ die übliche Vektorraumaddition in $\mathbb{R}^n$ ist.

Sind A und B konvexe Polytope, so ist auch $A+B$ ein konvexes Polytop ([Sch93]). Die Minkowski-Summen konvexer Polytope können also als die konvexe Hülle aller Summen $(a+b)$ der Eckpunkte von A respektive B berechnet werden.

Beispiel 9:
Anhand des folgenden Systems von drei Polynomen in zwei Veränderlichen wollen wir alle hier gezeigten Schritte bis hin zur Konstruktion der Resultantenmatrix nachvollziehen.

Seien $f_0, f_1, f_2 \in \mathbb{Z}[x,y]$. Wir betrachten das Gleichungssystem

$$\begin{aligned}
f_0 &= x+y+1 = 0\\
f_1 &= x^2+6x+3y-4 = 0\\
f_2 &= 2x+y^2-7y+5 = 0.
\end{aligned}$$

Die Trägermengen von f_0, f_1 und f_2:

$$\begin{aligned}\mathcal{A}_0 =& \operatorname{tr}(f_0) = \{(1,0),(0,1),(0,0)\},\\ \mathcal{A}_1 =& \operatorname{tr}(f_1) = \{(2,0),(1,0),(0,1),(0,0)\},\\ \mathcal{A}_2 =& \operatorname{tr}(f_2) = \{(1,0),(0,2),(0,1),(0,0)\}.\end{aligned}$$

Die Mengen der Eckpunkte E_i der Newton Polytope Q_i von $\mathcal{A}_i$ sind nach Definition 13

$$\begin{aligned}E_0 =& \{(1,0),(0,1),(0,0)\} \subset \mathcal{A}_0,\\ E_1 =& \{(2,0),(0,1),(0,0)\} \subset \mathcal{A}_1,\\ E_2 =& \{(1,0),(0,2),(0,0)\} \subset \mathcal{A}_2.\end{aligned}$$

Die Minkowski-Summe von E_0, E_1 und E_2 ist nach dieser Definition

$$\begin{aligned}E = \{&(4,0),(3,2),(3,0),(2,1),(1,3),(1,1),(2,0),(1,2),\\ &(1,0),(3,1),(2,3),(0,4),(0,2),(0,3),(0,1),(2,2),(0,0)\}\end{aligned}$$

Da die E_i, $i = 0,1,2$, konvex sind, ist die Minkowski-Summe E ebenfalls konvex.

In Abbildung (2.2) sind das Newton Polytop $Q = \operatorname{Conv}(E) \in \mathbb{R}^2$ und die Menge E dargestellt. Wie man sieht, besteht die Menge E gerade aus allen ganzzahligen Gitterpunkten in Q, es gilt also $E = Q \cap \mathbb{Z}^2$. ⊣

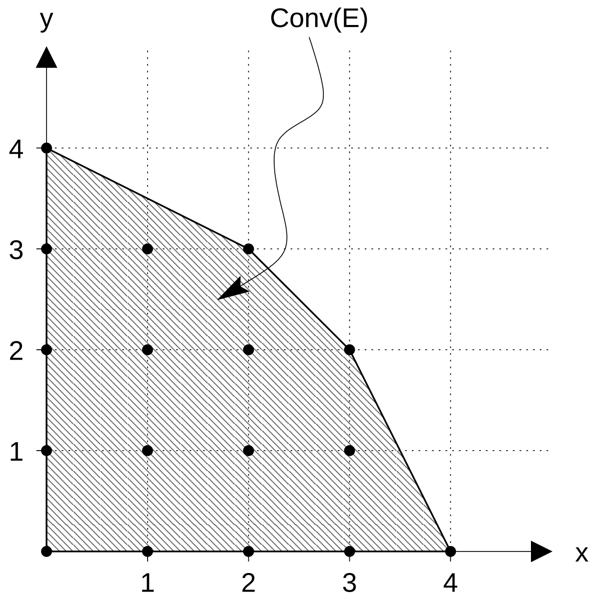

Abbildung 2.2: Die Menge E und ihre konvexe Hülle.

Definition 15: Seien $A, B \subset \mathbb{R}^n$ zwei Polytope und $\lambda \in \mathbb{R}_{\geq 0}$. Das Polytop λA ist durch

$$\lambda A := \{\lambda a \mid a \in A\},$$

definiert, wobei λa die übliche Skalarmultiplikation in $\mathbb{R}^n$ ist.

Kommen wir nun zu gemischten Volumina von Polytopen. Für ein Polytop $Q \in \mathbb{R}^n$ bezeichne $\mathrm{Vol}_n(Q)$ das Lebesgue-Maß von Q im n-dimensionalen Euklidischen Raum.

Definition 16: Seien n konvexe Polytope $Q_1, \ldots, Q_n \subset \mathbb{R}^n$ gegeben. Dann existiert eine bis auf skalare Vielfache eindeutige Funktion $\mathrm{MV}(Q_1, \ldots, Q_n) : (R^n)^n \to \mathbb{R}$, die multilinear unter Berücksichtigung der Minkowski-Addition und der skalaren Multiplikation ist. Mit anderen Worten, für $\mu, \rho \in \mathbb{R}_{\geq 0}$ und konvexe Polytope $Q'_k \in \mathbb{R}^n$ gilt

$$\mathrm{MV}(Q_1, \ldots, \mu Q_k + \rho Q'_k, \ldots, Q_n) = \mu \mathrm{MV}(Q_1, \ldots, Q_k, \ldots, Q_n) + \rho \mathrm{MV}(Q_1, \ldots, Q'_k, \ldots, Q_n). \quad (2.28)$$

Die Funktion $\mathrm{MV}(Q_1, \ldots, Q_n)$ bezeichnet man als das **gemischte Volumen** von $Q_1, \ldots, Q_n$. Damit das gemischte Volumen exakt definiert ist, benötigt man

$$\mathrm{MV}(Q_1, \ldots, Q_1) = n! \mathrm{Vol}_n(Q_1). \quad (2.29)$$

Diese Definition weicht vom klassischen gemischten Volumen um den Faktor $n!$ ab. Diese Skalierung garantiert, daß das gemischte Volumen von Polytopen mit ganzzahligen Eckpunkten wiederum ganzzahlig ist. Eine äquivalente Definition (siehe auch [Sch93]) ist die folgende:

Definition 17: Seien n konvexe Polytope $Q_1, \ldots, Q_n \subset \mathbb{R}^n$ und $\lambda_1, \lambda_2, \ldots, \lambda_n \in \mathbb{R}_{\geq 0}$ gegeben. Das n-dimensionale **gemischte Volumen** von $Q_1, \ldots, Q_n$ wird mit

$$\mathrm{MV}_n(Q_1, \ldots, Q_n)$$

bezeichnet und ist durch den Koeffizienten von $\lambda_1 \cdots \lambda_n$ in $\mathrm{Vol}_n(\lambda_1 Q_1 + \cdots + \lambda_n Q_n)$ aufgefasst als als ein Polynom in $\lambda_1 \cdots \lambda_n$ gegeben.

Andere allgemeine Eigenschaften und eine ausführlichere Einführung zu gemischten Volumina findet man zum Beispiel in [Sch93] oder in [PS93], wo auch einige andere Möglichkeiten zur Berechnung von gemischten Volumina vorgestellt werden.

Ein Programm in der Programmiersprache C zur Berechnung von gemischen Volumina findet man unter [CE97a]. Der dort verwendete *Lift-Prune-Algorithmus* wird in [CE95] näher beschrieben.

2.3.3 Die sparse Resultante

Das Theorem von Bernstein, ausführlich diskutiert und bewiesen in [CLO97b], Kapitel 7, Paragraph 5, ermöglicht es uns, eine Aussage über die Anzahl der gemeinsamen Lösungen eines polynomialen Systems zu machen.

Satz 19: (Bernsteins Theorem)
Seien $f_1, \ldots, f_n \in \mathbb{C}[x_1, x_1^{-1}, \ldots, x_n, x_n^{-1}]$ Laurent-Polynome. Die Anzahl der gemeinsamen Nullstellen des Systems $f_1 = \ldots = f_n = 0$ in $(\mathbb{C}^*)^n$ ist entweder unendlich oder durch $\mathrm{MV}_n(Q_1, \ldots, Q_n)$ nach oben beschränkt, wobei Q_i das jeweilige Newton Polytop von f_i ist. Für fast alle Spezialisierungen der Koeffizienten c_{ij} von f_i ist die Anzahl der Nullstellen exakt gleich $\mathrm{MV}_n(Q_1, \ldots, Q_n)$.

Diese obere Grenze $\mathrm{MV}_n(Q_1,\dots,Q_n)$ für die Anzahl der Nullstellen wird im Allgemeinen auch als die *obere BKK-Grenze* bezeichnet. Mit dieser Bezeichnung werden die Beiträge von Kushnirenko [Kus76] und Khovanskii [Kho78] zur Entwicklung und zum Beweis des Theorems von Bernstein gewürdigt.

Die hier vorgestellte Formulierung der sparsen Resultante ist etwas einfacher, als die in [GKZ94] vorgestellte, da wir zusätzlich annehmen wollen, daß die Koeffizienten hinreichend generisch sind.

Wir betrachten ein Polynom $f_i = \sum c_{ij} x^{a_{ij}}$, $c_{ij} \neq 0$, als den folgenden generischen Punkt in $\mathbb{P}^{\mu_i - 1}(\mathbb{C})$:

$$(c_{i1}, c_{i2}, \dots, c_{i\mu_i}) \in \mathbb{P}^{\mu_i-1}(\mathbb{C}), \quad i = 0, 1, \dots, n,$$

im Raum aller möglichen Polynome mit dem gegebenen Träger $\mathcal{A}_i$ von f_i, $\mu_i = |\mathcal{A}_i|$, bis auf skalare Vielfache. Dann ergibt sich mit dem polynomialen System 2.26 der folgende Punkt:

$$c = (c_{01}, \dots, c_{0\mu_0}, \dots, c_{n1}, \dots, c_{n\mu_n}), \quad \text{so daß} \quad c \in \mathbb{P}^{\mu_0-1}(\mathbb{C}) \times \cdots \times \mathbb{P}^{\mu_n-1}(\mathbb{C}).$$

Sei $Z_0 = Z_0(\mathcal{A}_0, \mathcal{A}_1, \dots, \mathcal{A}_n)$ die Menge aller Punkte c, so daß das dazugehörige polynomiale System eine Lösung in $(\mathbb{C}^*)^n$ hat und sei $Z = Z(\mathcal{A}_0, \mathcal{A}_1, \dots, \mathcal{A}_n)$ der Zariski-Abschluß von Z_0 im Produkt der projektiven Räume. Z ist dann eine irreduzible algebraische Menge ([PS93]). Jetzt können wir die sparse Resultante definieren.

Definition 18: Mit der obigen Notation ist die **sparse Resultante** $\mathrm{Res}(\mathcal{A}_0, \mathcal{A}_1, \dots, \mathcal{A}_n)$ des polynomialen Systems (2.26) ein Polynom in $\mathbb{Z}[c]$, also ein Polynom in den Koeffizienten der $f_0, f_1, \dots, f_n$, so daß gilt:

(i) Ist $\mathrm{codim}(Z) = 1$, so ist $\mathrm{Res}(\mathcal{A}_0, \mathcal{A}_1, \dots, \mathcal{A}_n)$ das definierende Polynom der Hyperebene Z.

(ii) Ist $\mathrm{codim}(Z) > 1$, dann ist $\mathrm{Res}(\mathcal{A}_0, \mathcal{A}_1, \dots, \mathcal{A}_n) = 1$.

Notation: Man schreibt auch $\mathrm{Res}_{\mathcal{A}_0, \mathcal{A}_1, \dots, \mathcal{A}_n}(f_0, f_1, \dots, f_n)$ für $\mathrm{Res}(\mathcal{A}_0, \mathcal{A}_1, \dots, \mathcal{A}_n)$, um die Abhängigkeit der Träger $\mathcal{A}_i$ von den Polynomen f_i hervorzuheben.

Der nächste Satz liefert uns den Zusammenhang zwischen den Lösungen von (2.26) und der sparsen Resultante aus Definition (18)

Satz 20: Seien $f_0, f_1, \dots, f_n \in \mathbb{C}[x_1, x_1^{-1}, \dots, x_n, x_n^{-1}]$ $n+1$ Laurent-Polynome in n Variablen. Dann existiert ein irreduzibles bis auf skalare Vielfache eindeutiges Polynom $\mathrm{Res}_{\mathcal{A}_0, \mathcal{A}_1, \dots, \mathcal{A}_n} \in \mathbb{Z}[c]$ mit den Eigenschaften aus Definition (18), so daß

$$(f_0, f_1, \dots, f_n) \in Z(\mathcal{A}_0, \mathcal{A}_1, \dots, \mathcal{A}_n) \Longleftrightarrow \mathrm{Res}_{\mathcal{A}_0, \mathcal{A}_1, \dots, \mathcal{A}_n}(f_0, f_1, \dots, f_n) = 0.$$

Genauer, das System $f_0 = f_1 = \cdots = f_n$ hat genau dann eine Lösung $(p_1, \dots, p_n) \in (\mathbb{C}^*)^n$, wenn

$$\mathrm{Res}_{\mathcal{A}_0, \mathcal{A}_1, \dots, \mathcal{A}_n}(f_0, f_1, \dots, f_n) = 0.$$

Beweis: Siehe [GKZ94], Kapitel 8.

Diese Definition der Resultante umfaßt alle bereits vorgestellten Definitionen. Die Macaulay Resultante aus dem vorherigen Abschnitt ist nämlich gerade

$$\mathrm{Res}_{d_0,d_1,\ldots,d_n}(F_0,F_1,\ldots,F_n) = \mathrm{Res}_{\mathcal{A}_0,\mathcal{A}_1,\ldots,\mathcal{A}_n}(f_0,f_1,\ldots,f_n),$$

wobei $\mathcal{A}_i = \{m \in \mathbb{Z}^n_{\geq 0} \mid |m| \leq d_i = \deg(F_i)\}$ und F_i die Homogenisierung von f_i ist.

Mit Hilfe des gemischen Volumens können wir auch den Grad der Resultante bestimmen.

Korollar 21: Angenommen, $Q_i = \mathrm{Conv}(\mathcal{A}_i) \subset \mathbb{Z}^n$ ist n-dimensional für $i = 0,1,\ldots,n$, und $\mathbb{Z}^n$ wird von den verschiedenen Elementen aus $\mathcal{A}_0 \cup \mathcal{A}_1 \cup \ldots \cup \mathcal{A}_n$ generiert. Fixiert man i auf einen Wert zwichen 0 und n, so ist $\mathrm{Res}_{\mathcal{A}_0,\mathcal{A}_1,\ldots,\mathcal{A}_n}$ homogen in den Koeffizienten $(c_{i1},\ldots,c_{i\mu_i})$ von f_i vom Grad

$$\deg_{f_i} \mathrm{Res}_{\mathcal{A}_0,\mathcal{A}_1,\ldots,\mathcal{A}_n} = \mathrm{MV}_n(Q_0,\ldots,Q_{i-1},Q_{i+1},\ldots,Q_n) =: \mathrm{MV}_{-i}.$$

Beweis: Folgt aus Satz 19. Siehe auch [GKZ94], Kapitel 8, oder [PS93], Abschnitt 2.

Konsequenterweise folgt dann mit diesem Korollar, daß der Grad der Resultante

$$\deg \mathrm{Res}_{\mathcal{A}_0,\mathcal{A}_1,\ldots,\mathcal{A}_n} = \sum_{i=0}^{n} \mathrm{MV}_{-i}$$

ist.

2.3.4 Gemischte polyhedrale Subdivision

Sind $n+1$ Newton Polytope eines Systems (2.26) gegeben, so sei deren Minkowski-Summe

$$Q := Q_0 + Q_1 + \cdots + Q_n \subset \mathbb{R}^n. \tag{2.30}$$

Die folgende Funktion gibt die Minkowski-Addition wieder:

$$\begin{aligned} m : (\mathbb{R}^n)^{n+1} &\to \mathbb{R}^n \\ m|_{Q_0\times Q_1\times\cdots\times Q_n} : Q_0 \times Q_1 \times \cdots \times Q_n &\to Q \\ (p_0,\ldots,p_n) &\mapsto p_0 + \cdots + p_n = q. \end{aligned} \tag{2.31}$$

Dies ist, wie man leicht sieht, eine wohldefinierte und nicht notwendigerweise injektive Abbildung.

Für jeden Punkt $q \in Q$ werden wir jetzt ein eindeutiges Inverses $(p_0,\ldots,p_n)$ in $Q_0 \times Q_1 \times \cdots \times Q_n$ definieren.

Dazu gehen wir nach dem in [Stu94] beschrieben Verfahren vor. Wir wählen $n+1$ hinreichend generische homogene *Lift-Funktionen* $l_0,\ldots,l_n \in \mathbb{Z}[x_1,\ldots,x_n]$ mit

$$\begin{aligned} l_i : \mathbb{Z}[x_1,\ldots,x_n] &\to \mathbb{Z} \\ (x_1,\ldots,x_n) &\mapsto L_{i1}x_1 + \cdots + L_{in}x_n, \quad i = 0,1,\ldots,n, \end{aligned} \tag{2.32}$$

wobei die L_{ij} hinreichend generisch zufällig aus $\mathbb{Z}$ gewählt seien. Was hierbei mit „generisch" gemeint ist, wollen wir später definieren, sobald die benötigten Begriffe zur Verfügung stehen. Die gelifteten Newton Polytope von $Q_0, \dots, Q_n$ sind dann so definiert:

$$\hat{Q}_i = \{(p, l_i(p)) \mid p \in Q_i\} \subset \mathbb{R}^{n+1}, \quad i = 0, 1, \dots, n. \tag{2.33}$$

Man beachte, daß ein geliftetet Punkt $\hat{p} \in \hat{Q}_i$ nur dann wohldefiniert ist, wenn sich der entsprechende Punkt in $\mathbb{R}^n$ als eine Summe aus Punkten in Q_i ausdrücken läßt.

Die Minkowski-Summe der gelifteten Newton Polytope ist, entsprechend (2.30),

$$\hat{Q} := \hat{Q}_0 + \hat{Q}_1 + \cdots + \hat{Q}_n \subset \mathbb{R}^{n+1}. \tag{2.34}$$

Beispiel 10: (Fortsetzung von Beispiel 9.)
Wir wählen die Lift-Funktionen l_0, l_1, l_2 wie folgt:

$$l_0(x,y) = 3x + 4y, \qquad l_1(x,y) = 6x + y, \qquad l_2(x,y) = 2x + 5y.$$

Nun liften wir die Mengen E_0, E_1 und E_2 aus $\mathbb{R}^2$ und erhalten so die Mengen

$$\begin{aligned}
\hat{E}_0 &= \{(1,0,l_0(1,0)), (0,1,l_0(0,1)), (0,0,l_0(0,0))\} = \{(1,0,3), (0,1,4), (0,0,0)\},\\
\hat{E}_1 &= \{(2,0,l_1(2,0)), (0,1,l_1(0,1)), (0,0,l_1(0,0))\} = \{(2,0,12), (0,1,1), (0,0,0)\},\\
\hat{E}_2 &= \{(1,0,l_2(1,0)), (0,2,l_2(0,2)), (0,0,l_2(0,0))\} = \{(1,0,2), (0,2,10), (0,0,0)\}
\end{aligned}$$

aus $\mathbb{R}^{n+1}$.

Die Minkowski-Summe von $\hat{E}_0$, $\hat{E}_1$ und $\hat{E}_2$ ist

$$\begin{aligned}
\hat{E} = \{&(4,0,17), (3,2,25), (3,1,18), (3,0,15), (3,0,14), (2,3,26), (2,2,22), (2,1,16), (2,1,6),\\
&(2,0,12), (2,0,5), (1,3,14), (1,2,13), (1,2,7), (1,1,6), (1,1,4), (1,1,3), (1,0,3),\\
&(1,0,2), (0,4,15), (0,3,14), (0,3,11), (0,2,10), (0,2,5), (0,1,4), (0,1,1), (0,0,0)\}
\end{aligned}$$

Die konvexe Hülle $\mathrm{Conv}(\hat{E})$ entspricht offensichtlich gerade der Menge $\hat{Q}$, die sich aus der Minkowski-Summe von $\hat{Q}_i = \mathrm{Conv}(\hat{E}_i)$, $i = 0, 1, 2$, berechnet. ⊣

Kommen wir als nächstes zur Definition der unteren Hülle eines konvexen Polytops.

Definition 19: Gegeben sei ein konvexes Newton Polytop in $\mathbb{R}^{n+1}$ mit positivem $(n+1)$-dimensionalen Volumen. Die **untere Hülle** dieses Polytops unter Berücksichtigung des Vektors $(0, \dots, 0, 1) \in \mathbb{R}^{n+1}$ ist die Vereinigung aller n-dimensionaler Seiten respektive der Facetten, deren nach innen gerichteter Normalenvektor eine positive letzte Komponente hat.

Jetzt kann der Begriff der hinreichenden Generizität der Lift-Funktionen näher erläutert werden.

Definition 20: Wir betrachten einen Eckpunkt $\sum_{i=0}^{n}(p_i, l_i(p_i))$ auf der unteren Hülle von $\hat{Q}$, welcher durch die Menge der Eckpunte $\{p_0, \dots, p_n\}$ mit $p_i \in Q_i$ definiert ist. Sei $\{q_0, \dots, q_n\}$ eine davon verschiedene Eckpunktmenge mit $q_i \in Q_i$, so daß $\sum_{i=0}^{n} p_i = \sum_{i=1}^{n} q_i$. Das durch die Funktionen $l_0, \dots, l_n$ definierte Liften ist genau dann hinreichend generisch, wenn die gelifteten Bilder zweier beliebiger Punkte in Q verschieden sind, also

$$\sum_{i=0}^{n}(p_i, l_i(p_i)) \neq \sum_{i=1}^{n}(q_i, l_i(q_i)) \text{ respektive } \sum_{i=0}^{n} l_i(p_i) \neq \sum_{i=1}^{n} l_i(q_i). \tag{2.35}$$

Da die rein geometrische Berechnung der unterhen Hülle einer konvexen Hülle recht aufwendig sein kann, wollen wir sie direkt aus der Minkowki-Summe von Polytopen herleiten.

Die Projektion eines Punktes $p \in \hat{Q}$ auf die ersten n Koordinaten sei definiert durch

$$\begin{aligned} \pi : \mathbb{R}^{n+1} &\to \mathbb{R}^n \\ (p_1, \dots, p_{n+1}) &\mapsto (p_1, \dots, p_n). \end{aligned} \tag{2.36}$$

Sowohl $\hat{Q}$ als auch die untere Hülle von $\hat{Q}$ werden von π auf Q abgebildet, wobei die zweite Abbildung injektiv, die erste hingegen möglicherweise nicht injektiv ist.

Die Projektion auf die $(n+1)$-te Koordinate sei durch

$$\begin{aligned} h : \mathbb{R}^{n+1} &\to \mathbb{R} \\ (p_1, \dots, p_{n+1}) &\mapsto p_{n+1} \end{aligned} \tag{2.37}$$

definiert und gibt die „Höhe“ des gelifteten Polytops an.

Mit diesen Notationen ist die folgende Funktion wohldefiniert:

$$\begin{aligned} s : Q &\to \hat{Q} \\ q &\mapsto \hat{q} \in \pi^{-1}(q) \cap \hat{Q}, \text{ so daß } h(\hat{q}) \text{ minimal ist.} \end{aligned} \tag{2.38}$$

Die untere Hülle von $\hat{Q}$ ist dann gerade $s(Q)$.

Beispiel 11: (Fortsetzung von Beispiel 9.)
Wir „berechnen“ die untere Hülle $s(E)$ von $\hat{E}$, indem wir für Punkte $e = (e_1, e_2, e_3)$ in $\hat{E}$ mit jeweils gleichen e_1- und e_2- Koordinaten denjenigen mit der kleinsten „Höhe“, also der kleinsten e_3-Koordinate, wählen.

$$\begin{aligned} s(E) = \{&(4,0,17),(3,2,25),(3,1,18),(3,0,14),(2,3,26), \\ &(2,2,22),(2,1,6),(2,0,5),(1,3,14),(1,2,7),(1,1,3), \\ &(1,0,2),(0,4,15),(0,3,11),(0,2,5),(0,1,1),(0,0,0)\} \end{aligned}$$

Die untere Hülle von $\hat{E}$ wird in Abbildung 14 graphisch dargestellt. ⊣

Die generische Forderung an l_i war, daß jeder Punkt $\hat{q}$ auf der unteren Hülle von $\hat{Q}$ als *eindeutige Summe* von Punkten $\hat{q} = \hat{q}_0 + \cdots + \hat{q}_n$ mit $\hat{q}_i \in \hat{Q}_i$ ausgedrückt werden kann. Mit dieser Forderung definieren wir das eindeutige Inverse für die Funktion m aus (2.31) wie folgt:

$$\begin{aligned} m^{-1}|_Q : Q &\to Q_0 \times \cdots \times Q_n \\ q &\mapsto (q_0, \ldots, q_n), \end{aligned} \tag{2.39}$$

wobei $s(q) = \hat{q}_0 + \cdots + \hat{q}_n \in \mathbb{R}^{n+1}$. Die Summe der Projektionen dieser Punkte $\hat{q}_i$ bezüglich π ist $q = q_0 + \cdots + q_n \in \mathbb{R}^{n+1}$, was aus der Konstruktion folgt.

Definition 21: Die eindeutige Minkowski-Summe $s(q) = \hat{q} = \hat{q}_0 + \cdots + \hat{q}_n$, $\hat{q}_i \in \hat{Q}_i$, eines Eckpunktes $\hat{q}$ der unteren Hülle von $\hat{Q}$ wird als **optimal** bezeichnet. Die Projektionen der Summanden unter π bilden eine eindeutige optimale Summe, die den entsprechenden Punkt in Q darstellt. Die Summanden sind dabei durch $m^{-1}|_Q(q)$ gegeben.

Beispiel 12: (Fortsetzung von Beispiel 9.)
Die optimale Minkowski-Summe für $(1,1) \in E$ ist $e_{12} + e_{21} = (0,1) + (1,0) = (1,1)$ da $s((1,1)) = \hat{e}_{12} + \hat{e}_{21} = (0,1,1) + (1,0,2) = (1,1,3)$ mit $h(1,1,3) = 3$ minimal. $\dashv$

Definition 22: Sei $Q \subset \mathbb{R}^n$ ein Newton Polytop der Dimension n. Eine **polyhedrale Subdivision** Δ von Q besteht aus einer Familie endlich vieler Polytope $R_0, \ldots, R_s$, so daß $Q = R_0 \cup \ldots \cup R_s$ und für $i \neq j$ ist $R_i \cap R_j$ eine Seite von R_i und R_j. Die Polytope R_i werden die **Zellen** der Subdivision Δ genannt. Zellen mit Dimension n werden als **maximale Zellen** oder **Facetten** bezeichnet.

Beispiel 13: Abbildung 2.3 zeigt drei Möglichkeiten, ein Quadrat in kleinere Teile zu zerlegen. Die

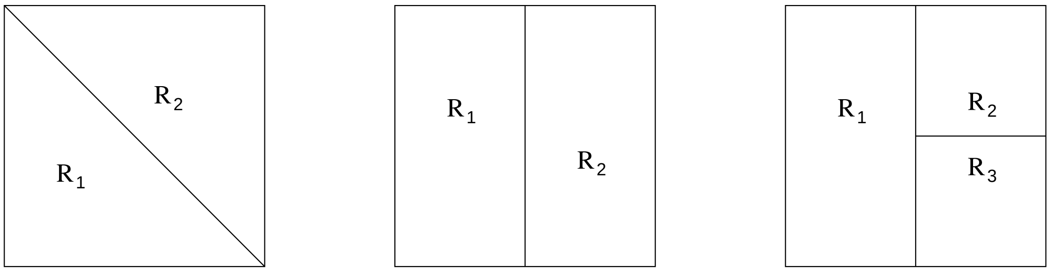

Abbildung 2.3: Zerlegungen eines Quadrates

ersten beiden sind polyhedrale Subdivisionen, die dritte aber nicht, da $R_1 \cap R_2$ keine Seite von R_1 ist, genauso wie $R_1 \cap R_3$ keine Seite von R_1 ist. $\dashv$

Betrachten wir jetzt die natürliche polyhedrale Subdivision der unteren Hülle $s(Q)$ von $\hat{Q}$ in ihre Seiten S_j. Die Familie aller Facetten der unteren Hülle wollen wir mit $\hat{\Delta}$ bezeichnen.

$\hat{Q}$ ist durch die Minkowski-Summe der gelifteten Newton Polytope $\hat{Q}_i$ definiert. Auf die gleiche Weise kann jede der Seiten S_j von $\hat{Q}$ durch eine Minkowski-Summe von $n+1$ Seiten der korrespondierenden gelifteten Newton Polytope ausgedrückt werden, also durch die Summe der Seiten $\hat{F}_0 + \cdots + \hat{F}_n = S_j$ mit $\hat{F}_i \subset \hat{Q}_i$.

Die Summe der Dimensionen $\dim(\hat{F}_i)$, ist dabei größer oder gleich der Dimension der entsprechenden Seite S_j in $\hat{\Delta}$,

Beispiel 14: (Fortsetzung von Beispiel 9.)
Zunächst wollen wir eine Triangulierung der untere Hülle $s(Q) \in \mathbb{R}^3$ von $\hat{Q}$ vornehmen, sie also in ihre Seiten S_j zerlegen. Eine Seite S_j soll dabei durch jeweils drei Punkte $p,q,w \in \mathbb{Z}^3$ und ihren Normalenvektor $v \in \mathbb{Z}^3$ repräsentiert werden, was wir so schreiben:

$$[(p_1,p_2,p_3),(q_1,q_2,q_3),(w_1,w_2,w_3),(v_1,v_2,v_3)].$$

Wir erhalten 15 Seiten:

$$\begin{aligned}
S_0 &= [(0,3,11),(0,2,5),(1,2,7),(-2,-4,1)] \\
S_1 &= [(0,3,11),(1,3,14),(1,2,7),(-2,-4,1)] \\
S_2 &= [(0,2,5),(0,1,1),(1,2,7),(-2,-1,1)] \\
S_3 &= [(0,1,1),(1,2,7),(1,1,3),(-2,-1,1)] \\
S_4 &= [(1,0,2),(0,3,11),(1,3,14),(-2,-5,1)] \\
S_5 &= [(1,3,14),(1,2,7),(2,2,22),(-3,-4,1)] \\
S_6 &= [(1,0,2),(2,0,5),(4,0,17),(-12,-13,1)] \\
S_7 &= [(1,0,2),(3,0,14),(4,0,17),(-12,-13,1)] \\
S_8 &= [(1,0,2),(2,0,5),(2,2,22),(-6,-7,1)] \\
S_9 &= [(1,0,2),(1,3,14),(2,2,22),(-6,-7,1)] \\
S_{10} &= [(1,1,3),(2,2,22),(2,1,6),(-3,-1,1)] \\
S_{11} &= [(1,2,7),(1,1,3),(2,2,22),(-3,-1,1)] \\
S_{12} &= [(2,2,22),(2,1,6),(0,0,0),(-6,-1,1)] \\
S_{13} &= [(2,2,22),(4,0,17),(0,0,0),(-10,-9,1)] \\
S_{14} &= [(2,0,5),(2,2,22),(4,0,17),(-10,-9,1)]
\end{aligned}$$

Wie man sieht, haben einige der Seiten gleiche Normalenvektoren. Diese Seiten lassen sich zu Facetten der unteren Hülle zusammenfassen. Natürlich ist eine Seite, deren Normalenvektor einzigartig ist, auch eine Facette.

Im oberen Teil von Abbildung 14 sieht man sehr schön die Triangulierung der unteren Hülle von $\hat{Q}$ in ihre Seiten. Haben mehrere Seiten den gleichen Normalenvektor, bilden gemeinsam also eine Facette von $s(Q)$, so sind sie in der selben Farbe dargestellt. Jede Farbe makiert also eine maximale Zelle der polyhedralen Subdivision $\hat{\Delta}$ von $s(Q)$.

Die 9 Facetten der unteren Hülle sind die maxiamlen Zellen ihrer polyhedralen Subdivision $\hat{\Delta}$:

$$\begin{aligned}
\hat{R}_0 &= \{(0,2,5),(0,1,1),(1,1,3),(1,2,7)\}, \\
\hat{R}_1 &= \{(0,1,1),(0,0,0),(1,1,3),(1,0,2)\}, \\
\hat{R}_2 &= \{(0,4,15),(0,2,5),(1,2,7)\}, \\
\hat{R}_3 &= \{(1,2,7),(1,1,3),(2,1,6)\}, \\
\hat{R}_4 &= \{(0,4,15),(1,3,14),(3,2,25),(2,3,26)\}, \\
\hat{R}_5 &= \{(0,4,15),(1,3,14),(2,1,6),(1,2,7)\}, \\
\hat{R}_6 &= \{(1,0,2),(2,1,6),(2,0,5),(1,1,3)\}, \\
\hat{R}_7 &= \{(2,1,6),(2,0,5),(4,0,17)\}, \\
\hat{R}_8 &= \{(2,1,6),(3,2,25),(4,0,17),(1,3,14)\}.
\end{aligned}$$

Jede dieser Zellen $\hat{R}_i$ kann durch eine Minkowski-Summe $\hat{F}_0+\hat{F}_1+\hat{F}_2$ dargestellt werden. Ein $\hat{F}_i \subset \hat{Q}_i$ ist dabei eine Seite von $\hat{Q}_i$. Eine Seite der Dimension $< \dim(\hat{Q}) = 3$ wurde als Facette ($\dim = 2$), Kante ($\dim = 1$) oder Eckpunkt ($\dim = 0$) von $\hat{Q}_i$ bezeichnet.

Wir erinnern uns, daß die Eckpunktmengen $\hat{E}_i$ der geliften Newton Polytope $\hat{Q}_i$ folgende Gestalt haben:

$$\hat{E}_0 = \{(1,0,3),(0,1,4),(0,0,0)\},\ \hat{E}_1 = \{(2,0,12),(0,1,1),(0,0,0)\},$$
$$\hat{E}_2 = \{(1,0,2),(0,2,10),(0,0,0)\}.$$

So läßt sich zum Beispiel $\hat{R}_0$ durch die Minkowski-Summe aus zwei Kanten und einem Eckpunkt darstellen:

$$\begin{aligned}\hat{R}_0 =& \hat{F}_0+\hat{F}_1+\hat{F}_2\\ =&\{e_{02},e_{03}\}+\{e_{12}\}+\{e_{21},e_{23}\}\\ =&\{(0,1,4),(0,0,0)\}+\{(0,1,1)\}+\{(1,0,2),(0,0,0)\}\\ =&\{(0,2,5),(0,1,1),(1,1,3),(1,2,7)\}.\end{aligned}$$

⊣

Die Forderung der Generizität an die Lift-Funktionen impliziert die folgende Eigenschaft:

$$\dim \hat{F}_0+\cdots+\dim \hat{F}_n = \dim \sum_{i=0}^{n} \hat{F}_i. \tag{2.40}$$

Desweiteren ist die Familie $\hat{F}_0,\ldots,\hat{F}_n$ für eine Seite in $\hat{\Delta}$ eindeutig definiert durch $m^{-1}|_Q$.

Das Bild von $\hat{\Delta}$ unter π induziert also eine polyhedrale Subdivision Δ von Q, deren maximalen Zellen eine eindeutige Darstellung $F_0+\cdots+F_n$ haben, wobei F_i die zu $\hat{F}_i$ korrespondierende Seite von Q_i ist. Sind die Lift-Funktionen generisch, so wird die Subdivision auch als gemischt bezeichnet.

Definition 23: Sei $Q = Q_1+\ldots+Q_n \subset \mathbb{R}^n$ ein Newton Polytop der Dimension n. Eine polyhedrale Subdivision $R_1,\ldots,R_s$ von Q ist eine **gemischte** Subdivision, wenn jede Zelle R_i der Subdivision als Minkowski-Summe

$$R_i = F_0+\cdots+F_n$$

geschrieben werden kann und es gilt $\dim(F_1)+\cdots+\dim(F_n) = n$. Dabei ist jedes der F_i eine Seite von Q_i, $i = 0,\ldots,n$.

Die folgende Definition vervollständigt die Definition 22 der Subdivision.

Definition 24: Läßt sich eine maximale Zelle der Subdivision $\hat{\Delta}$ der unteren Hülle von $\hat{Q}$ als Minkowski-Summe $\hat{F}_0+\cdots+\hat{F}_n$ schreiben, dann wird die Minkowski-Summe $F_0+\cdots+F_n$ der korrespondierenden maximalen Zelle in der Subdivision Δ von Q als **optimal** bezeichnet.

Abbildung 2.4: Oben im Bild: die untere Hülle $s(Q)$ von $\hat{Q}$, die Triangulierung (dargestellt durch die schwarzen Kanten) und die maxiamlen Zellen der Subdivision $\hat{\Delta}$ von $s(Q)$ (dargestellt durch die Einfärbung); unten im Bild: die Projektion in die Ebene und die daraus resultierende polyhedrale Subdivision Δ von Q.

Liegt ein Punkt q in einer Zelle $F_0 + \cdots + F_n$, so liegen die Punkte q_i seiner optimalen Summe in F_i. Ebenso liegen die Summanden der eindeutigen Summe eines Punktes $\hat{q}$ auf einer Facette $\hat{F}_0 + \cdots + \hat{F}_n$ der unteren Hülle von $\hat{Q}$ in $\hat{F}_i$.

Lemma 22: Sei R eine beliebige Zelle beliebiger Dimension aus der gemischten Subdivision Δ von $Q \subset \mathbb{R}^n$. Ist die optimale Summe dieser Zelle $F_0 + \cdots + F_n$, F_i eine Seite von Q_i, so ist mindestens eines der F_i 0-dimensional, das heißt, ein Eckpunkt. Dies gilt sowohl für maximale Zellen als auch für Punkte.

Beweis: Da das Liften linear ist, verändert es nicht die Dimension der Seiten. Also gilt für die optimalen Summanden F_i einer Zelle in Δ, daß $\sum_i \dim(F_i) = \sum_i \dim(\hat{F}_i) \leq n$. Da die Dimension nicht negativ ist, folgt das Lemma.

□

Definition 25: Eine maximale Zelle einer gemischten Subdivision Δ, die sich optimal als Minkowski-Summe $F_0 + \cdots + F_n$ ausdrücken läßt, wobei genau ein F_i 0-dimensional ist, wird **gemischte Zelle** genannt. Die verbleibenden F_j, $j \neq i$, sind also Kanten, d. h. $\dim(F_j) \geq 1$. Alle maximalen Zellen, die nicht gemischt sind, werden als **ungemischte Zellen** bezeichnet.

Beispiel 15: (Fortsetzung von Beispiel 9.)
Projeziert man die untere Hülle $s(Q)$ unter Berücksichtigung der Facetten in die Ebene, so erhält man die polyhedrale Subdivision Δ von $Q = \mathrm{Conv}(E)$. Dabei ergibt sich die folgende Aufteilung in maximale Zellen R_i der Subdivision:

$$
\begin{aligned}
R_0 =& \{(0,2),(0,1),(1,1),(1,2)\},\\
R_1 =& \{(0,1),(0,0),(1,1),(1,0)\},\\
R_2 =& \{(0,4),(0,2),(1,2)\},\\
R_3 =& \{(1,2),(1,1),(2,1)\},\\
R_4 =& \{(0,4),(1,3),(3,2),(2,3)\},\\
R_5 =& \{(0,4),(1,3),(2,1),(1,2)\},\\
R_6 =& \{(1,0),(2,1),(2,0),(1,1)\},\\
R_7 =& \{(2,1),(2,0),(4,0)\},\\
R_8 =& \{(2,1),(3,2),(4,0),(1,3)\}.
\end{aligned}
$$

Die polyhedrale Subdivision Δ von Q in die maximalen Zellen R_i ist auch im unteren Teil von Abbildung 14 dargestellt.

Jede dieser Zellen R_i kann durch eine Minkowski-Summe $F_0 + \cdots + F_n$ dargestellt werden. Ein $F_i \subset Q_i$ ist dabei eine Seite von Q_i. Wir erinnern uns, das eine Seite mit einer Dimension kleiner $dim(Q) = 2$ eine Facette $(\dim = 1)$ beziehungsweise eine Kante $(\dim = 1)$ oder ein Eckpunkt $(\dim = 0)$ von Q_i sein kann.

Wir erinnern uns, daß die Eckpunktmengen E_i der Newton Polytope Q_i folgende Gestalt haben:

$$E_0 = \{(1,0)(0,1)(0,0)\}, \;\; E_1 = \{(2,0)(0,1)(0,0)\}, \;\; E_2 = \{(1,0)(0,2)(0,0)\}.$$

So läßt sich zum Beispiel R_0 als die Minkowski-Summe aus zwei Kanten und einem Eckpunkt darstellen:

$$R_0 = F_0 + F_1 + F_2 = \{e_{02}, e_{03}\} + \{e_{12}\} + \{e_{21}, e_{23}\} = \{(0,1),(0,0)\} + \{(0,1)\} + \{(1,0),(0,0)\}$$

Dabei gilt $\dim(F_0) + \dim(F_1) + \dim(F_2) = 1 + 1 + 0 = 2$. Außerdem handelt es sich hierbei gemäß Definition 24 um eine *optimale* Minkowski-Summe, da die F_i mit den $\hat{F}_i$ der Zelle $\hat{R}_0$ aus dem letzten Beispiel korrespondieren.

Die Zelle R_3 läßt sich durch eine Minkowski-Summe von einer Facette und zwei Punkten darstellen:

$$R_3 = F_0 + F_1 + F_2 = \{e_{01}, e_{02}, e_{03}\} + \{e_{12}\} + \{e_{21}\} = \{(1,0),(0,1),(0,0)\} + \{(0,1)\} + \{(1,0)\}$$

Auch hier ergibt die Summe der Dimensionen $\sum_{i=0}^{2} \dim(F_i) = 2$.

Ebenso lassen sich die verbleibenden Zellen durch eine entsprechende Minkowski-Summe mit $\sum_{i=0}^{2} \dim(F_i) = 2$ ausdrücken, es handelt sich hierbei laut Definition 23 also um eine *gemischte* polyhedrale Subdivision.

Die Zelle R_0 ist eine *gemischte* Zelle, da genau ein F_i der darstellenden optimalen Minkowski-Summe 0-dimensional ist. Bei R_3 handelt es sich hingegen um eine *ungemsichte* Zelle (Definition 25). ⊣

Lemma 23: Sei Δ eine gemischte Subdivision von Q. Dann ist

$$\mathrm{MV}_{-i} = \sum_{R = F_0 + \cdots + F_{i-1} + a_{ij} + F_{i+1} + \cdots + F_n} \mathrm{Vol}(R), \quad i = 0, \ldots, n,$$

wobei jedes F_k eine Kante von Q_k ist und $a_{ij} \in \mathcal{A}_i$.

Beweis: Siehe [Stu94].

Software in Form eines Programms in der Sprache C zur Berechnung gemischter Subdivionen von Polytopen findet man unter [Emi94].

2.3.5 Konstruktion der sparsen Resultante

Basierend auf den erabeiteten Eigenschaften der gemischten polyhedralen Subdivision wollen wir jetzt die quadratische *sparse Resultantenmatrix* $\mathcal{SM}$ definieren und konstruieren. Der Vollständigkeit halber sei noch erwähnt, daß die Matrix $\mathcal{SM}$ wegen der zugrundeliegenden Newton Polytope des definierenden Gleichungssystems manchmal auch *Newton Matrix* genannt wird.

Die Zeilen und Spalten der Matrix $\mathcal{SM}$ werden wir durch eine Teilmenge $\mathcal{E}$ der ganzzahligen Gitterpunkte in Q indizieren. Um jedem Gitterpunkt in Q eine eindeutige Zelle zuordnen zu können, permutiert man Q mit Hilfe eines Vektors $\delta \in \mathbb{R}^n$ so, daß die ganzzahligen Gitterpunkte, die zuvor auf den Eckpunkten der maximalen Zellen lagen, jetzt entweder im Inneren einer maximalen Zelle zu liegen kommt oder sich außerhalb von Δ befindet. Aus den Gitterpunkten innerhalb maximaler Zellen ergibt sich die Menge $\mathcal{E}$.

Definition 26: Sei $\delta \in \mathbb{R}^n \backslash \{0\}$ ein hinreichend klein gewählter Vektor. Wir definieren $\mathcal{E}$ als die folgende Menge ganzzahliger Gitterpunkte:

$$\mathcal{E} = (Q + \delta) \cap \mathbb{Z}^n \subset \mathbb{Z}^n.$$

Diese Menge $\mathcal{E}$ steht in bijektiever Korrespondenz zu einer Menge von Laurent-Polynomen in n Variablen, ein Punkt in $\mathcal{E}$ läßt sich folglich als Exponent eines Monoms auffassen.

Sei Δ_δ die gemischte polyhedrale Subdivision, die man erhält, wenn man alle maximalen Zellen in der gemischte Subdivision Δ von Q um den Vektor δ verschiebt, also

$$\Delta_\delta := \{R + \delta \mid R \in \Delta\}. \tag{2.41}$$

Bevor wir zur eigentlichen Definition der Resultantenmatrix kommen, wollen wir uns mit der Berechnung der Menge $\mathcal{E}$ beschäftigen.

Definition 27: Sei $Q \in \mathbb{R}^n$ ein konvexes Polytop und $\delta \in \mathbb{R}^n \backslash \{0\}$. Der **$\delta$-Abstand** eines Punktes $p \in Q \cap \mathbb{Z}^n$ ist das maximale $s \in \mathbb{R}_{\geq 0}$, so daß $p + s\delta \in Q$. Anders gesagt, der **δ-Abstand** ist der Abstand von p zum Rand von Q in Richtung δ.

Ganzzahlige Punkte auf dem Rand des Polytops Q haben folglich den δ-Abstand 0 und Punkte ausserhalb von Q einen negativen δ-Abstand.

Der folgende Algorithmus wird *Mayan-Pyramid-Algorithmus* genannt und zählt alle Punkte in $\mathcal{E}$ auf. Er stellt eine Abwandlung des in [CE95], Seite 127ff, vorgestellten Algorithmus dar. Der Mayan-Pyramid-Algorithmus arbeitet rekursiv und berechnet in der k-ten Stufe den Bereich der Werte der k-ten Koordinate in $\mathcal{E}$, wobei die $(k-1)$-te Koordinaten auf die Koordinaten des $(k-1)$-dimensionalen Vektors $u \in \mathbb{Z}^{k-1}$ fixiert sind. Bei jedem rekursiven Aufruf „steigt" der Algorithums sozusagen um eine „Dimensionsstufe" höher, woraus sich der Name ergibt.

Algorithmus 2: MayanPyramid.

Eingabe: Sei E_i die Menge aller Eckpunkte der Newton Polygone $Q_i = \mathrm{Conv}(\mathrm{tr}(f_i))$ eines Systems (2.26), $i = 0, \ldots, n$.
Sei $\delta \in \mathbb{R}^n \backslash \{0\}$ generisch gewählt. Sei $E = E_0 + \cdots + E_n$.

Ausgabe: $\mathcal{E} = (Q + \delta) \cap \mathbb{Z}^n \subset \mathbb{Z}^n$, mit $Q = Q_0 + \cdots + Q_n$.

Algorithmus:

$n \leftarrow$ Anzahl der Variablen
- Die Anzahl der Variablen entspricht der Dimension der Exponentenvektoren des Systems (2.26) und damit der Anzahl der Koordinaten.

$\mathcal{E} \leftarrow \{\}$
$dim \leftarrow 1$
$p \leftarrow (0)$

```
            • p ∈ ℤ^dim.

(ℰ) = MayanPyramid(ℰ, dim, p, E_0, ..., E_n, δ)

    begin
      mx ← maximale dim-te Koordinate in E
      mn ← minimale dim-te Koordinate in E
         • Berechnet wird (p_0, ..., p_{dim-1}, s) für s maximal beziehungsweise mi-
           nimal mittels linearer Optimierung (siehe unten).

      if (dim < n)
        then
            for (q = mn, ..., mx) do
              p ← (p_0, ..., p_{dim-1}, q)
              MayanPyramid(ℰ, dim + 1, p, E_0, ..., E_n, δ)
                 • Der Algorithmus „klettert“ auf die nächsthöhere „Dimensionsstufe“
            od
      fi

      if (dim = n)
           • Dimension n erreicht, der Algorithmus terminiert.
        then
            for (q = mn, ..., mx) do
              p ← (p_0, ..., p_{n-1}, q)
              s ← δ-Abstand von p
                 • Berechnet wird das maximale s, so daß p + sδ ∈ Q. Verwendet wird
                   lineare Optimierung, siehe unten.
              if (s > 0)
                then
                    ℰ ← ℰ ∪ {p}
              fi
            od
            return (ℰ)
      fi

      return (ℰ)
    end
```

Die minimalen und maximalen k-ten Koordinaten mn und mx wollen wir durch lineare Optimierung bestimmen. Hier sollen nur die Zielfunktion und die erforderlichen Nebenbedingungen angegeben werden. Wie man solch ein lineares Programm mit Hilfe des *Simplex-Algorithmus* berechnet, gehört zu den Grundlagen der Praktischen Mathematik (siehe zum Beispiel [Sch97]).

Zur Berechnung von mn wählen wir für ein $k > 1$ die folgende Zielfunktion:

$$\text{minimiere } s \in \mathbb{R}:$$
$$(p_1, \ldots, p_{k-1}, s) = \sum_{i=0}^{n} \sum_{j=0}^{m_i} \lambda_{ij} e_{ij}^k, \tag{2.42}$$

mit $\lambda_{ij} \geq 0$ für alle $i \in \{0, \ldots, n\}$, $j \in \{1, \ldots, m_i\}$. Dabei war $m_i = |E_i|$ mit $E_i = \{e_{i1}, \ldots, e_{im_i}\} \subset \mathcal{A}_i$ als Eckpunktmengen und $m_i \leq \mu_i = |\mathcal{A}_i|$, $i = 0, \ldots, n$. Mit e_{ij}^k ist der Vektor $(e_{ij1}, \ldots, e_{ijk}) \in \mathbb{Z}^k$ gemeint, der aus den ersten k Koordinaten von $e_{ij} \in \mathbb{Z}^n$ besteht.

Als Nebenbedingungen betrachten wir die Summen

$$\sum_{j=1}^{m_i} \lambda_{ij} = 1, \quad \text{für alle } \; i \in \{0, \ldots, n\}. \tag{2.43}$$

Die Berechnung von mx erfolgt mit dem selben linearen Programm, nur daß diesmal s maximiert wird.

Die Berechnung des δ-Abstandes eines Punktes p erfolgt ebenfalls durch lineare Optimierung. Wir wählen als Zielfunktion

$$\text{minimiere } s \in \mathbb{R}:$$
$$(p_1, \ldots, p_n) + sp = \sum_{\substack{l=0 \\ l \neq i}}^{n} \sum_{j=1}^{m_i} \lambda_{lj} p_{lj}. \tag{2.44}$$

Als Nebenbedingungen betrachten wir

$$\sum_{j=1}^{m_i} \lambda_{ij} = 1, \quad \text{für alle } \; i \in \{0, \ldots, n\}. \tag{2.45}$$

Die Rekursion terminiert, wenn $k = n$ oder $[mn, mx]$ für ein beliebiges k leer ist.

Nun, da wir in der Lage sind, $\mathcal{E}$ aufzuzählen, kommen wir zur eigentlichen Konstruktion der Matrix $\mathcal{SM}$.

Sei $\hat{\Delta}$ eine gemischte Subdivision der unteren Hülle $s(Q)$ von $\hat{Q}$, Δ die korrespondierende gemischte Subdivision von Q und $\hat{\Delta}_\delta = \{R + \delta \mid R \in \hat{\Delta}\}$. Jetzt definieren wir eine Funktion, die uns Auskunft darüber gibt, wie die Elemente von $\mathcal{SM}$ zu wählen sind.

Definition 28: (Zeileninhaltsfunktion)
Sei $p \in \mathcal{E}$ ein Punkt aus dem Inneren einer gemischten Zelle $R = \delta + F_0 + \cdots + F_n$ einer gemischten polyhedralen Subdivision Δ_δ. Die **Zeileninhaltsfunktion** ist wie folgt definiert:

$$\begin{aligned} \mathrm{ZIF}: \mathcal{E} &\to \{0, \ldots, n\} \times \mathbb{N} \\ p &\mapsto (i, j), \quad j \in \{1, \ldots, m_i\}, \end{aligned}$$

wobei $i \in \{0, \ldots, n\}$ die *größte* Zahl sei, so daß F_i ein Eckpunkt ist und $F_i = e_{ij} \in E_i$ für ein $j \in \{1, \ldots, m_i\}$. Das heißt, aus $\mathrm{RCF}(p) = (i, j)$ folgt, daß $\dim(F_k) > 0$ für alle $k > i$.

Die Zeileninhaltsfunktion ordnet jedem Punkt $p \in \mathcal{E}$ also ein Zahlenpaar (i, j) zu, welches sich aus der optimalen Minkowski-Summe der Zelle R ergibt, in der der Punkt p liegt. Die erste Komponente des Zahlenpaars ergibt sich aus dem Index der 0-dimensionalen Seite F_i der Summe $\delta + F_0 + \cdots + F_n$. Mit Lemma 22 wissen wir, daß die optimale Minkowski-Summe einer Zelle mindestens ein solches 0-dimensionales F_i enthält.

Bei der Zuordnung spielt die erste Gleichung $f_0 = 0$ des Eingabesystems $f_0 = \ldots = f_n = 0$ eine besondere Rolle. Wenn es nämlich mehrere 0-dimensionale F_i gibt, so wählt die Zeileninhaltsfunktion davon das mit dem größeren Index aus und vermeidet es so, einem Punkt $p \in \mathcal{E}$ dem Newton Polytop Q_0 von f_0 zuzuordnen. Aufgrund dieser Tatsache nennen wir $\mathcal{SM}$ *assoziiert zu* f_0 und schreiben $\mathcal{SM}_0$.

Der folgende Algorithmus berechnet für jeden Punkt $p \in \mathcal{E}$ das Paar (i, j).

Algorithmus 3: ZIF.

Eingabe: Seien $Q_0, \ldots, Q_n$ die Newton Polytope eines Systems (2.26), $\mathcal{E} = (Q + \delta) \cap \mathbb{Z}^n \subset \mathbb{Z}^n$, mit $Q = Q_0 + \cdots + Q_n$, und $\delta \in \mathbb{R}^n \setminus \{0\}$ generisch gewählt. Seien $l_0, \ldots, l_n$ Lift-Funktionen wie in (2.32).

Ausgabe: Das Paar (i, j) für jeden Punkt $p \in \mathcal{E}$.

Algorithmus: Angenommen, Q_i hat die Eckenmenge $E_i = \{e_{i1}, \ldots, e_{im_i}\}$ für $m_i \leq \mu_i$. $p \in R + \delta$ ist äquivalent zu $p - \delta \in R$ für eine maximale Zelle R von Δ_δ.

Gesucht ist für alle $p \in \mathcal{E}$ die optimale Summe von $p - \delta$ ausgedrückt durch Punkte $p_i \in Q_i$.

Die Berechnung der optimalen Summe von $p - \delta$ erfolgt widerum mittels linearer Optimierung. Als Zielfunktion wählen wir

$$\text{minimiere} \quad \sum_{i=0}^{n} \sum_{j=1}^{m_i} \lambda_{ij} l_i(e_{ij}) \tag{2.46}$$

Wir fordern also, daß der geliftete Punkt $s(p - \delta)$ auf der unteren Hülle von $\hat{Q}$ liegt.

Unsere Nebenbedingungen seinen gegeben durch

$$p - \delta = \sum_{i=0}^{n} p_i = \sum_{i=0}^{n} \sum_{j=1}^{m_i} \lambda_{ij} e_{ij} \tag{2.47}$$

mit

$$\lambda_{ij} \geq 0 \text{ für } 1 \leq j \leq m_i, \text{ und} \sum_{j=1}^{m_i} \lambda_{ij} = 1 \text{ für alle } i \in \{0, \ldots, n\}. \tag{2.48}$$

Für jedes i entsprechen die positiven λ_{ij} in der optimalen Lösung den Eckpunkten e_{ij}, die sich zu $p_i \in Q_i$ in der optimalen Minkowskisumme von $p - \delta$ aufsummieren.

Liegen mehrere Punkte in der gleichen maximalen Zelle $R = F_0 + \cdots + F_n$, so wird jede Seite F_i in der optimalen Summe von R durch alle voneinander verschiedenen Eckpunkte von Q_i, die als Summanden in optimalen Summen von Punkten in R auftreten, definiert.

Das maximale i für das F_i ein Eckpunkt e_{ij} ist, definiert $(i,j) = \mathrm{ZIF}(p)$ für alle $p \in R$.

Definition 29: Die **sparse Resultantenmatrix** $\mathcal{SM}_0$ ist eine quadratische Matrix, deren Zeilen und Spalten durch die Elemente der Menge $\mathcal{E}$ indiziert sind. Das Element von $\mathcal{SM}_0$ in der Zeile p und der Spalte q für beliebige $p,q \in \mathcal{E}$ mit $\mathrm{ZIF}(p) = (i,j)$, $i \in \{0,\ldots,n\}$, $j \in \{1,\ldots,m_i\}$, ist

$$\mathcal{SM}_{pq} = \begin{cases} c_{ik} & \text{für } q-p+e_{ij} \in E_i \text{ für ein } k \in \{1,\ldots,m_i\}, \\ 0 & \text{für } q-p+e_{ij} \notin E_i. \end{cases}$$

Ein Element auf der Diagonalen ist $\mathcal{SM}_{pp} = c_{ij}$ mit $(i,j) = \mathrm{ZIF}(p)$ für alle $p \in \mathcal{E}$. Die Definition impliziert, daß eine Zeile p von $\mathcal{SM}_0$, die durch $p \in \mathcal{E}$ indiziert wird, aus den Koeffizienten des monomialen Vielfachen des Polynoms f_i besteht:

$$x^{p-e_{ij}} f_i \quad \text{mit} \quad (i,j) = \mathrm{ZIF}(p),\ i = 0,\ldots,n. \tag{2.49}$$

Die Anzahl der Spalten von $\mathcal{SM}_1$ ist mindestens so groß, wie die Anzahl $|\mathcal{E}|$ der Zeilen. Es ist offensichtlich, daß $p - e_{ij} + e_{ik} \in Q \cap \mathbb{Z}^n$ für beliebige $e_{ik} \in E_i$. Um zu zeigen, daß die Matrix wohldefiniert ist, müssen wir das eben gesagte nach $Q+\delta$ übertragen. Das macht das folgende Lemma.

Lemma 24: Mit der obigen Notation ist $p - e_{ij} + e_{ik} \in \mathcal{E}$ für alle $e_{ik} \in E_i$.

Beweis: Da $p \in \mathcal{E}$ haben wir $p \in R+\delta$ für eine Zelle R, die sich als Minkowski-Summe $F_0 + \cdots + F_{i-1} + e_{ij} + F_{i+1} + \cdots + F_n$ schreiben läßt, wobei F_k eine Seite von Q_k ist. Dies impliziert, daß $p - e_{ij} + e_{ik}$ in

$$R - e_{ij} + e_{ik} + \delta = F_1 + \cdots + F_{i-1} + e_{ik} + F_{i+1} + \cdots + F_n + \delta \tag{2.50}$$

liegt, was wiederum eine Teilmenge von $Q+\delta$ für beliebige $e_{ik} \in E_i$ ist.

□

Jetzt können wirden folgenden Satz formulieren.

Satz 25: Die sparse Resultantenmatrix $\mathcal{SM}_0$ nach Definition 29 ist wohldefiniert. $\mathcal{SM}_0$ ist quadratisch und hat die Dimension $|\mathcal{E}|$.

Beispiel 16: (Fortsetzung von Beispiel 9.)
Im letzten Beispiel haben wir die polyhedrale Subdivision Δ von Q in die maximalen Zellen R_i bestimmt.

Nun wollen wir Δ um den Vektor $\delta = (0.25, 0.167) \in \mathbb{R}^2$ verschieben und erhalten so $\Delta_\delta = \{R+\delta \mid R \in \Delta\}$ und $\mathcal{E} = (Q+\delta) \cap \mathbb{Z}^2$ mit

$$\mathcal{E} = \{(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2)\}.$$

In Abbildung 2.5 sehen wir die gemischte Subdivision Δ_δ, die gemischten Zellen $R_0,\ldots,R_8$ der Subdivision und die Punkte $p \in \mathcal{E}$.

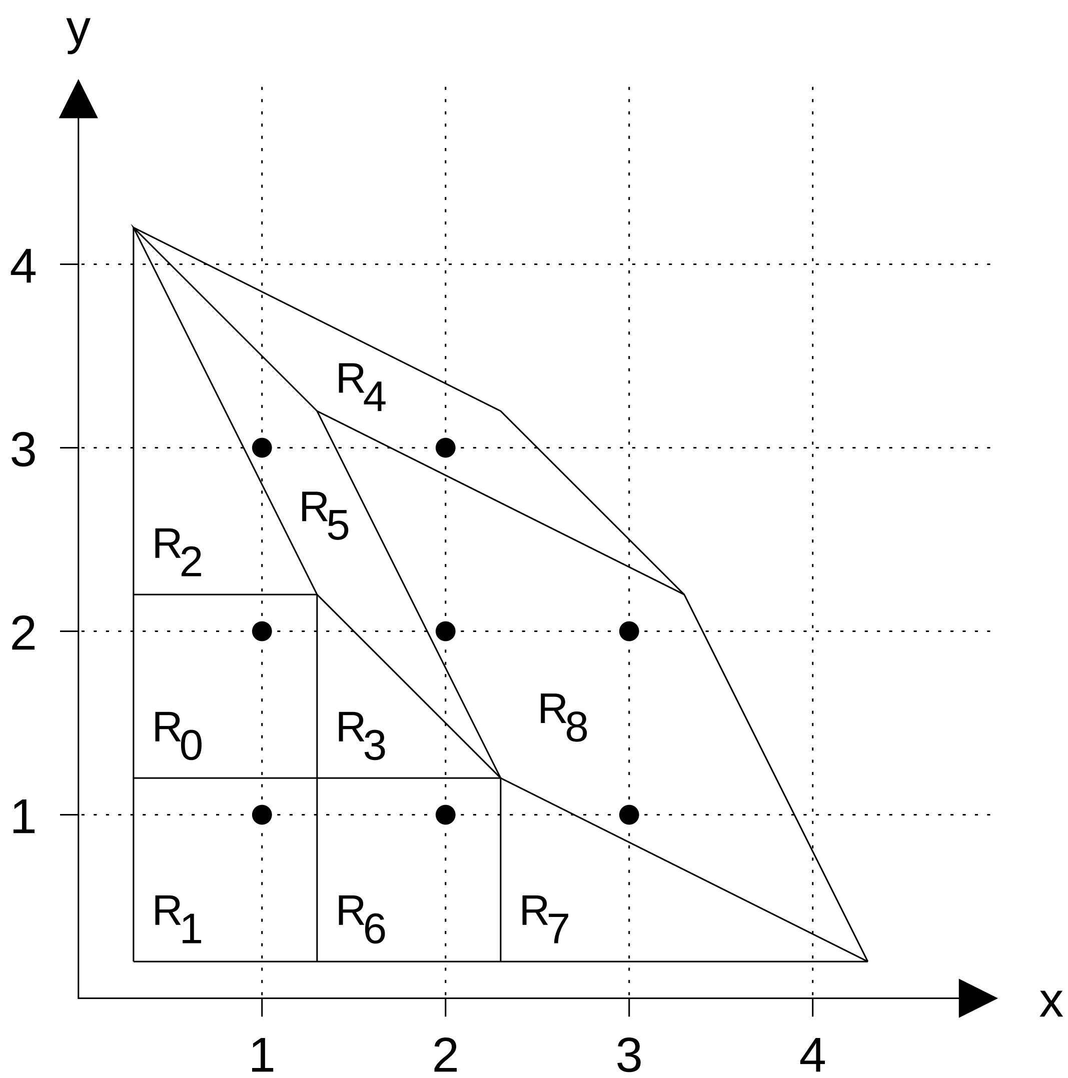

Abbildung 2.5: Die gemischten Zellen R_i der gemischten Subdivision Δ_δ von $Q+\delta$ und die ganzzahligen Gitterpunkte (die Menge $\mathcal{E}$) innerhalb der Zellen.

Der Punkt $(1,2) \in \mathcal{E}$ beispielsweise liegt in der Zelle R_0, deren optimale Minkowski-Summe

$$R_0 = F_0 + F_1 + F_2 = \{(0,1),(0,0)\} + \{(0,1)\} + \{(1,0),(0,0)\}$$

war. Offensichtlich ist F_1 die 0-dimensionale Seite mit dem größten Index, die Zeileninhaltsfunktion liefert also für $\mathrm{ZIF}((1,2)) = (1,2)$ zurück, was dem Punkt $(0,1) \in E_1$ entspricht.

Wenden wir die Zeileninhaltsfunktion auf alle Punkte $p \in \mathcal{E}$ an und konstruieren die Zeilen der Matrix $\mathcal{SM}_0$ wie in (2.49), so erhalten wir:

$$\begin{aligned}
&\text{1. Zeile: } \mathrm{ZIF}((1,1)) = (0,3), \text{ entspricht } e_{03} = (0,0) \in E_0\\
&\qquad \longrightarrow \frac{x^{p_1} y^{p_2}}{x^{e^1_{03}} y^{e^2_{03}}} \cdot f_0 = xy \cdot f_0 = x^2y + xy^2 + xy\\
&\text{2. Zeile: } \mathrm{ZIF}((1,2)) = (1,2), \text{ entspricht } e_{12} = (0,1) \in E_1\\
&\qquad \longrightarrow \frac{x^{p_1} y^{p_2}}{x^{e^1_{12}} y^{e^2_{12}}} \cdot f_0 = \frac{xy^2}{y} \cdot f_1 = x^3y + 6x^2y + 3xy^2 - 4xy\\
&\qquad \vdots\\
&\text{6. Zeile: } \mathrm{ZIF}((2,3)) = (2,2), \text{ entspricht } e_{22} = (0,2) \in E_2\\
&\qquad \longrightarrow \frac{x^2y^3}{y^2} \cdot f_2 = 2x^3y + x^2y^3 - 7x^2y^2 + 5x^2y\\
&\text{7. Zeile: } \mathrm{ZIF}((3,1)) = (0,1), \text{ entspricht } e_{01} = (1,0) \in E_0\\
&\qquad \longrightarrow \frac{x^3y}{x} \cdot f_0 = x^3y + x^2y^2 + x^2y\\
&\text{8. Zeile: } \mathrm{ZIF}((3,2)) = (0,1), \text{ entspricht } e_{01} = (1,0) \in E_0\\
&\qquad \longrightarrow \frac{x^3y^2}{x} \cdot f_0 = x^3y^2 + x^2y^3 + x^2y^2
\end{aligned}$$

Daraus ergibt sich mit Definiton 29 die sparse 8×8 - Resultantenmatrix, deren Spalten durch die zu den Punkten in $\mathcal{E}$ korrespondierenden Monome, geordnet in lexikographischer Ordnung, indiziert sind:

$$\mathcal{SM}_0 = \begin{array}{c} \begin{array}{cccccccc} xy & xy^2 & xy^3 & x^2y & x^2y^2 & x^2y^3 & x^3y & x^3y^2 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0\\ -4 & 3 & 0 & 6 & 0 & 0 & 1 & 0\\ 0 & -4 & 3 & 0 & 6 & 0 & 0 & 1\\ 5 & -7 & 1 & 2 & 0 & 0 & 0 & 0\\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 5 & -7 & 1 & 2 & 0\\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

$\dashv$

2.3.6 Eigenschaften der sparsen Resultante

Analog zu Abschnitt 2.2.4 wollen wir zeigen, daß die Determinante $\mathcal{SD}_0$ der sparsen Resultantenmatrix $\mathcal{SM}_0$ aus Definition 29 ein Vielfaches der sparsen Resultante $\mathrm{Res}_{\mathcal{A}_0,\mathcal{A}_1,\dots,\mathcal{A}_n}$ ist. Zunächst jedoch werden wir zeigen, daß der Grad von $\mathcal{SD}_0$ in den Koeffizienten eines jeden der Polynome $f_0,\dots,f_n$ gleich dem entsprechenden Grad der sparsen Resultante ist.

Definition 30: Seien feste ganzzahlige Vektoren $e_1,\dots,e_n \in \mathbb{Z}^n$ gegeben. Ein **halb-offenens Integral-Parallelotop** $\mathcal{H}$ ist definiert durch die Menge

$$\mathcal{H} = \{r_1e_1 + \dots + r_ne_n \mid r_1,\dots,r_n \in [0,1)\}.$$

Lemma 26: Die Anzahl der ganzzahligen Gitterpunkte in einem halb-offenen Integral-Parallelotop entspricht seinem Volumen $\mathrm{Vol}(\mathcal{H})$.

Beweis: Mit [Sta80], Seite 335, folgt, daß die Anzahl der Gitterpunkte $n!\mathrm{Vol}(S)$ ist, wobei S der Simplex mit der Eckpunktmenge bestehend aus dem Ursprung und den Eckpunkten von $e_1,\dots,e_n$ ist. Das Volumen des Parallelotops $\mathcal{H}$ ist bekannterweise gerade $n!\mathrm{Vol}(S)$.

□

Satz 27: Für ein beliebiges $\delta \in \mathbb{R}^n$ ist die Anzahl der ganzzahligen Gitterpunkte in $\mathcal{H}+\delta$ gleich dem Volumen $\mathrm{Vol}(\mathcal{H})$.

Beweis: Man beachte, daß $\mathcal{H}$ um $\delta \cdot t$ verschoben ist, wobei t zwischen 0 und 1 variiert. Desweitern beobachtet man, daß für jede Facette, die offen beziehungsweise geschlossen ist, die gegenüberliegende Facette geschlossen beziehungsweise offen ist und daß gegenüberliegende Facetten gegeneinander um einen ganzzahligen Vektor v verschoben sind. Wird H also bewegt und ein Gitterpunkt p „betritt“ $\mathcal{H}$, so „verläßt“ ein korrespondierender Punkt $p+v$ $\mathcal{H}$ und umgekehrt. Die Anzal der Gitterpunkte in $\mathcal{H}$ bleibt also konstant. Mit Lemma 26 entspricht die Anzahl der Gitterpunkte in $\mathcal{H}$ dem Volumen von $\mathcal{H}$.

□

Eine gemischte Zelle aus Δ_δ ist die Minkowski-Summe von n Kanten und einem Eckpunkt, also ein Parallelotop in $\mathbb{R}^n$.

Lemma 28: Die Anzahl der Spalten in $\mathcal{SM}_0$, die durch alle ganzzahligen Gitterpunkte in einer gemischten Zelle indiziert werden, entspricht dem Volumen der Zelle.

Beweis: Die Perturbation δ garantiert, daß alle ganzzahligen Gitterpunkte in $\mathcal{E}$ im relativen Inneren einer gemischten Zelle liegen. Die Anzahl dieser Punkte ist gleich der Anzahl der Punkte in dem halb-offenen Parallelotop definiert durch die n Kanten und den Eckpunkt in der optimalen Summe der Zelle. Da jeder Punkt in $\mathcal{E}$ genau einer Zeile der Matrix entspricht, folgt das Lemma mit Satz 27.

□

Satz 29: Die Determinante $\mathcal{SD}_0$ ist ein Vielfaches ungleich Null der sparsen Resultante $\mathrm{Res}_{\mathcal{A}_0,\mathcal{A}_1,\dots,\mathcal{A}_n}$, das heißt,

$$\mathrm{Res}_{\mathcal{A}_0,\mathcal{A}_1,\dots,\mathcal{A}_n} = C \cdot \mathcal{SD}_0.$$

Betrachtet man $\mathcal{SD}_0$ als Polynom in den Koeffizienten von f_0, so ist der Grad von $\mathcal{SD}_0$ durch das gemischte Volumen

$$\deg(\mathcal{SD}_0) = \mathrm{MV}_{-0} = \mathrm{MV}_n(Q_1,\dots,Q_n)$$

gegeben. Dies ist gleich dem Grad der sparsen Resultante in den entsprechenden Koeffizienten.

Beweis: Haben die Gleichungen $f_0 = f_1 = \cdots = f_n = 0$ eine Lösung in $(\mathbb{C}^*)^n$, so hat das Gleichungssystem aus (2.49) eine nichttriviale Lösung. Also hat die sparse Koeffizientenmatrix $\mathcal{SM}_0$ von (2.49) die Determinante 0. Mit Satz 20 folgt, daß $\mathcal{SD}_0$ für alle Elemente der Menge $Z(\mathcal{A}_0,\mathcal{A}_1,\dots,\mathcal{A}_n)$ Null ist. Da die sparse Resultante gerade die irreduzibel definierende Gleichung dieser Menge ist, muß sie $\mathcal{SD}_0$ teilen.
Um zu zeigen, daß $\mathcal{SD}_0$ ungleich Null ist, benötigen wir Polynome $f_0,\dots,f_n$ für die gilt $D_0 \neq 0$. Dazu führt man eine neue Variable γ ein und schreibt:

$$f_i = \sum_{a\in\mathcal{A}_i} \gamma^{l_i(a)} x^a,\ i = 0,\dots,n,$$

wobei die $l_i : \mathbb{Z}^n \to \mathbb{Z}$ die Lift-Funktionen aus der gemischten polyhedralen Subdivision von $Q = Q_0 + \cdots + Q_n$ sind. In [CE95],Paragraph 6, wird gezeigt, daß für diese Wahl der f_i gilt: $\mathcal{SD}_0 \neq 0$. Bleibt noch der Grad der Determinante betrachtet als Polynom in den Koeffizienten von f_0. Die Zeileninhaltsfunktion aus Definition 28 wählt f_0 genau für die gemischten Zellen aus, zu denen Q_0 einen Eckpunkt beiträgt. Nach Lemma 23 entspricht das Gesamtvolumen dieser Zellen $\mathrm{MV}_{-0} = \mathrm{MV}_n(Q_1,\dots,Q_n)$. Also entspricht die Anzahl der Zeilen, die Koeffizienten von f_0 enthalten, exakt MV_{-0}. Mit Satz 21 folgt der Zusammenhang zwischen MV_{-0} und dem entsprechenden Grad der sparsen Resultante.

□

Der Grad von $\mathcal{SD}_0$ respektive $\mathrm{Res}_{\mathcal{A}_0,\mathcal{A}_1,\dots,\mathcal{A}_n}$ ist also an das assoziierte Polynom f_0 gebunden.

Der Grad von $\mathcal{SD}_0$ in den Koeffizienten eines f_j ist gleich der Anzahl der Zeilen der sparsen Resultantenmatrix M_0, welche durch monomiale Vielfache von f_j gegeben sind.

Lemma 30: Sei $j \in \{1,\dots,n\}$ fest gewählt. Betrachtet man $\mathcal{SD}_0$ als Polynom in den Koeffizienten von f_j, so gilt:

$$\deg_{f_j}(\mathcal{SD}_0) \geq \mathrm{MV}_{-j} = \mathrm{MV}_n(Q_1,\dots,Q_{j-1},Q_{j+1},\dots,Q_n).$$

Dies ist gleich dem entsprechenden Grad der Resultante $\mathrm{Res}_{\mathcal{A}_0,\mathcal{A}_1,\dots,\mathcal{A}_n}$.

Beweis: Für jedes f_j, $j \geq 1$, ist die Anzahl der Zeilen in $\mathcal{SM}_0$, die ein monomiales Vielfaches von f_j enthalten, mindestens so groß, wie die Anzahl der ganzzahligen Gitterpunkte in den gemischten Zellen, zu denen Q_j einen Eckpunkt beiträgt, da die Zeileninhaltsfunktion für diese Zellen einen Eckpunkt aus $\mathcal{A}_j$ wählen muß. Die Anzahl der Punkte in diesen Zellen ist MV_{-j}. Mit Satz 21 folgt wieder der Zusammenhang zwischen MV_{-j} und dem entsprechenden Grad der sparsen Resultante.

□

Wir wollen die sparse Resultante $\mathrm{Res}_{\mathcal{A}_0,\mathcal{A}_1,\dots,\mathcal{A}_n}$ jetzt wie in Satz 16 als GCD der Determinaten $\mathcal{SD}_0,\dots,\mathcal{SD}_n$ ausdrücken, wobei $\mathcal{SD}_k$ jeweils die Determinante der zu f_k assoziierten Matrix $\mathcal{SM}_k$ sei. Die Zeileninhaltsfunktion nach Definition 28 liefert nur die zu f_0 assoziierte Matrix $\mathcal{SM}_0$ und deren Determinante $\mathcal{SD}_0$, weshalb wir hierfür eine allgemeinere Version der Zeileninhaltsfunktion benötigen.

Definition 31: (allgemeine Zeileninhaltsfunktion)
Sei $k \in \{1,\dots,n\}$ fest gewählt. Sei $p \in \mathcal{E}$ ein Punkt aus einer maximalen gemischten Zelle $\delta + F_0 + \cdots + F_n$ einer gemischten Subdivision Δ_δ von Q wie in Definition 28. Die **allgemeine Zeileninhaltsfunktion** ist dann wie folgt definiert:

$$\begin{aligned} \mathrm{aZIF} : \mathcal{E} &\to \{0,\dots,n\} \times \mathbb{N} \\ p &\mapsto (i,j), \quad j \in \{1,\dots,m_i\}, \end{aligned}$$

mit

$$i = \begin{cases} j, & \text{falls } \dim F_k \geq 0 \ \forall k \neq j \\ min(k \neq j \ : \ F_k \text{ ist ein Eckpunkt}) & \text{sonst} \end{cases}$$

Damit erhält man die entsprechenden Matrizen $\mathcal{SM}_1,\dots,\mathcal{SM}_n$.

Mit Satz 14 und Satz 16 können wir folgern:

Satz 31: $\mathrm{Res}_{\mathcal{A}_0,\mathcal{A}_1,\dots,\mathcal{A}_n} = \pm \mathrm{GCD}(\mathcal{SD}_0,\dots,\mathcal{SD}_n)$.

Zwei Methoden zur Berechnung der sparsen Resultante mit Hilfe des GCD's findet man in [CE97b], Paragraph 9.1 und 9.2. Natürlich macht auch hier eine Berechnung des GCD's nur Sinn, wenn man Polynome $f_0,\dots,f_n$ mit symbolischen Koeffizienten hat.

Eine bislang offene Frage ist, ob die sparse wie die Macaulay Resultante analog zu (2.20) als exakter rationaler Quotient zweier Determinanten dargestellt werden kann. Laut [CE97b], Paragraph 13, ist dies für Spezialfälle bereits möglich.

3 Lösen von Gleichungssystemen

Im letzten Kapitel wurden zwei Methoden zur Konstruktion und Berechnung von Resultanten multipolynomialer Systeme vorgestellt. Im nächsten Abschnitt wird die u-Resultante eingeführt und zur Bestimmung der gemeinsamen Lösungen eines Gleichungssystems verwendet. In den darauf folgenden Abschnitten werden einige Methoden zur Vereinfachung der Berechnung und Bestimmung der numerischen Lösungen eines Systems vorgestellt.

Seien $f_1,\dots,f_n \in \mathbb{C}[x_1,\dots,x_n]$ Polynome, die ein nulldimensionales Ideal $I := \langle f_1,\dots,f_n\rangle$ bilden. Wir suchen die Varietät $\mathbf{V}(I)$, also die Nullstellen des Gleichungssystems

$$f_1(x_1,\dots,x_n) = \cdots = f_n(x_1,\dots,x_n) = 0. \tag{3.1}$$

Ist ein homogenes Sytstem

$$F_1(x_0,x_1,\dots,x_n) = \cdots = F_n(x_0,x_1,\dots,x_n) = 0 \tag{3.2}$$

mit $F_1,\dots,F_n \in \mathbb{C}[x_0,x_1,\dots,x_n]$ gegeben, so dehomogenisieren wir wie schon in (2.9), indem wir $x_0 = 1$ setzen und definieren:

$$\begin{aligned} f_i(x_1,\dots,x_n) &= F_i(1,x_1,\dots,x_n),\\ \overline{F}_i(x_1,\dots,x_n) &= F_i(0,x_1,\dots,x_n). \end{aligned} \tag{3.3}$$

Man beachte, daß f_i maximal vom Grad $\deg(F_i)$ ist. In $\mathbb{P}^n(\mathbb{C})$ liegt der affine Raum $\mathbb{C}^n \subset \mathbb{P}^n(\mathbb{C})$ definiert durch $x_0 = 1$, die Lösungen des affinen Systems (3.1) sind also genau die Lösungen des homogenen Systems (3.2), die in $\mathbb{C}^n \subset \mathbb{P}^n(\mathbb{C})$ liegen.

Die Lösungen des homogenen Systems

$$\overline{F}_1(x_1,\dots,x_n) = \cdots = \overline{F}_n(x_1,\dots,x_n) = 0 \tag{3.4}$$

kann man, anschaulich gesagt, als die Lösungen betrachten, die im Unendlichen liegen. Man sagt, das System (3.1) hat keine Lösungen im Unendlichen, wenn $\overline{F}_1(x_1,\dots,x_n) = \cdots = \overline{F}_n(x_1,\dots,x_n) = 0$ keine nichttrivialen Lösungen hat. Nach Satz 6 ist dies gleichbedeutend mit der Bedingung

$$\mathrm{Res}(\overline{F}_1,\dots,\overline{F}_n) \neq 0. \tag{3.5}$$

3.1 Die u-Resultante

Die **u-Resultante**, die nun eingeführt werden soll, wurde erstmals von van der Waerden [vdW50] verwendet. Dabei handelt es sich nicht um eine neue Variate der Resultantenmatrix, vielmehr bezieht sich der Name auf die besondere Form der Eingabepolynome. Die u-Resultante ermöglicht das Lösen von polynomialen Gleichungssystemen der Art (3.1) unter Verwendung sowohl der multipolynomialen Resultante als auch der sparsen Resultante aus dem vorangegangenen Kapitel.

In den vergangenen 10 Jahren wurden verschiedene Arbeiten über die Verwendbarkeit der u-Resultante zum Lösen von Systemen geschrieben und diverse Algorithmen veröffentlicht. Einige dieser Arbeiten sind [Stu98, SZ94, CM91, CM92, Man93, Man94] und [CLO97b].

Betrachten wir also ein Gleichungssystem der Form

$$f_1(x_1,\dots,x_n) = \dots = f_n(x_1,\dots,x_n) = 0. \tag{3.6}$$

wie in (3.1).

Die Idee der u-Resultante besteht darin, dem Ideal $I := \langle f_1,\dots,f_n\rangle$ ein weiteres Polynom $f_0 \in \mathbb{C}(u_0,u_1,\dots,u_n)[x_1,\dots,x_n]$ hinzuzufügen. Der Einfachheit halber wählt man für f_0 eine lineare Gleichung

$$f_0 = u_0 + u_1x_1 + \dots + u_nx_n = 0, \tag{3.7}$$

wobei $u_0,u_1,\dots,u_n$ unabhängige Variablen sind. Man erhält so das System

$$f_0(x_1,\dots,x_n) = f_1(x_1,\dots,x_n) = \dots = f_n(x_1,\dots,x_n) = 0 \tag{3.8}$$

mit Polynomen $f_0,f_1,\dots,f_n \in \mathbb{C}(u_0,u_1,\dots,u_n)[x_1,\dots,x_n]$ von denen man nun sowohl die multipolynomial Resultante als auch die sparse Resultante berechnen kann. Als Resultante $\mathrm{Res}(f_0,f_1,\dots,f_n)$ erhält man offensichtlich ein Polynom in den unabhängigen Variablen $u_0,u_1,\dots,u_n$ des hinzugefügent Polynoms $f_0 = 0$, also

$$\mathrm{Res}(f_0,f_1,\dots,f_n) \in \mathbb{C}(u_0,u_1,\dots,u_n). \tag{3.9}$$

Der nun folgende Satz zeigt, wie man durch die Hinzunahme von f_0 zu I die Varietät $\mathbf{V}(I)$ bestimmt.

Satz 32: Seien $f_1,\dots,f_n \in \mathbb{K}[x_1,\dots,x_n]$ Polynome mit maximalem Grad $d_1,\dots,d_n$ und sei $\mathbb{K}$ ein algebraisch abgeschlossener Körper. Sei $\mathrm{Res}(\overline{F}_1,\dots,\overline{F}_n) \neq 0$, das heißt, es existiert keine Lösung im Unendlichen. Ist $f_0 = u_0 + u_1x_1 + \dots + u_nx_n$, wobei $u_0,\dots,u_n$ unabhängige Variablen sind, dann existiert eine Konstante $C \neq 0$, so daß

$$\begin{aligned}\mathrm{Res}(f_0,f_1,\dots,f_n) &= C \prod_{p\in\mathbf{V}(f_1,\dots,f_n)} (f_0(p))^{m(p)} \\ &= C \prod_{p\in\mathbf{V}(f_1,\dots,f_n)} (u_0 + a_1u_1 + \dots + a_nu_n)^{m(p)}\end{aligned}$$

wobei $p \in \mathbf{V}(f_1,\dots,f_n)$ mit $p = (a_1,a_2,\dots,a_n)$ und $m(p)$ die Multiplizität von p ist.

Beweis: Siehe [CLO97b], Kapitel 4.

Beispiel 17: Wir betrachten die Polynome $f_1, f_2 \in \mathbb{C}[x,y]$ mit

$$f_1 = x^2 + 6x + 3y - 4,$$
$$f_2 = 2x + y^2 - 7y + 5.$$

Gesucht sind die Lösungen von $f_1 = f_2 = 0$.

Zur Berechnung der u-Resultante fügen wir das lineare Polynom $f_0 \in \mathbb{C}(u_0, u_1, u_2)[x,y]$

$$f_0 = u_0 + u_1 x + u_2 y$$

zu f_1, f_2 hinzu und können jetzt die u-Resultante $\mathrm{Res}(f_0, f_1, f_2)$ bestimmen.

Konstruieren wir die multipolynomiale Resultantenmatrix mit dem Verfahren von Macaulay, so erhalten wir die multipolynomiale (10×10) - Resultantenmatrix $\mathcal{DM}_0$ mit

$$\mathcal{DM}_0 = \begin{pmatrix} u_0 & u_1 & u_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & u_0 & 0 & u_1 & u_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & u_0 & 0 & u_1 & u_2 & 0 & 0 & 0 & 0 \\ -10 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & u_0 & 0 & 0 & u_1 & u_2 & 0 \\ -16 & 0 & 0 & 1 & 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & -10 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & -10 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & -16 & 0 & 0 & 0 & 0 & 1 & 1 & 2 & 0 \\ 0 & 0 & -16 & 0 & 0 & 0 & 0 & 1 & 1 & 2 \end{pmatrix}.$$

Die Submatrix $\mathcal{DM}'_0$ von $\mathcal{DM}_0$ ist der konstante Faktor 1.

Als u-Resultante erhält man das Polynom

$$\begin{aligned} \mathrm{Res}_{1,d_1,\ldots,d_n}(f_0, f_1, \ldots, f_n) &= \pm \frac{\det(\mathcal{DM}_0)}{\det(\mathcal{DM}_0)} \\ &= \pm(u_0 + u_1 - 3u_2)(u_0 - u_1 + 3u_2)(u_0 + 2\sqrt{2}u_1 + \sqrt{2}u_2)(u_0 - 2\sqrt{2}u_1 - \sqrt{2}u_2), \end{aligned}$$

welches bereits in seine Linearfaktoren zerlegt wurde.

Konstruieren wir die sparse Resultantenmatrix, so erhalten wir $\mathcal{SM}_0$ mit

$$\mathcal{SM}_0 = \begin{pmatrix} -16 & 0 & 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & -16 & 0 & 2 & 0 & 0 & 1 & 0 & 1 & 0 \\ -10 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -10 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -16 & 0 & 2 & 0 & 1 & 1 \\ 0 & u_0 & u_2 & 0 & 0 & u_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & u_0 & u_2 & 0 & 0 & u_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & u_0 & u_2 & 0 & u_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & u_0 & u_2 & 0 & u_1 & 0 \\ 0 & 0 & 0 & 0 & -10 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

und derselben Dimension wie bei der multipolynomialen Resultantenmatrix.

Als u-Resultante erhält man das gleiche Polynom wie oben:

$$\mathrm{Res}_{\mathcal{A}_0,\mathcal{A}_1,\ldots,\mathcal{A}_n}(f_0,f_1,\ldots,f_n) = \pm\det(sM_0) = \pm\mathrm{Res}_{1,d_1,\ldots,d_n}(f_0,f_1,\ldots,f_n)$$

Die Koeffizienten der Linearfaktoren von $\mathrm{Res}(f_0,f_1,\ldots,f_n)$ ergeben 4 Punkte

$$(1,1,-3),\ (1,-1,3),\ (1,2\sqrt{2},\sqrt{2}),\ (1,-2\sqrt{2},-\sqrt{2})$$

in $\mathbb{P}^2(\mathbb{C})$. Die gesuchten Nullstellen von $f_1 = f_2 = 0$ in $\mathbb{C}^2$ sind dann

$$(1,-3),\ (-1,3),\ (2\sqrt{2},\sqrt{2}),\ (-2\sqrt{2},-\sqrt{2})$$

$\dashv$

Satz 32 liefert ein Polynom $\mathrm{Res}(f_0,f_1,\ldots,f_n) \in \mathbb{K}[u_0,\ldots,u_n]$, das sich in Linearfaktoren der Form

$$u_0 + a_1u_1 + \cdots + a_nu_n \tag{3.10}$$

zerlegen läßt, wobei $(a_1,\ldots,a_n)$ mit den Lösungen des Original-Systems (3.1) übereinstimmen.

Damit hat man, wie wir bereits im Beispiel gesehen haben, einen ersten Ansatz zur Bestimmung der Nullstellen eins Systems der Form (3.1). Man berechnet zunächst die u-Resultante des Systems und erhält so ein Polynom Res, welches man dann in seine Linearfaktoren der Form (3.10) zerlegt. Anschließend „liest“ man die Nullstellen ab.

Kommen wir zur Berechnung der Resultante von (3.1). Wir erinnern uns, daß $\mathcal{SD}_0$ die Determinante der sparsen Resultantenmatrix $\mathcal{SM}_0$ nach Abschnitt 2.3 bezeichnet, wohingegen $\mathcal{DD}_0$ für die Determinante der multipolynomialen Resultantenmatrix $\mathcal{DM}_0$ nach Abschnitt 2.2 steht. Ist von $\mathcal{M}_0$ oder $\mathcal{D}_0$ die Rede, so ist es belanglos, ob hier die sparse oder die multipolynomiale Resultantenmatrix beziehungsweise deren Determinante verwendet wird.

Zur Berechnung kann man, wie wir im Beispiel bereits gesehen haben, einerseits die multipolynomiale Resultante nach Satz 18 verwenden, wobei man $f_0,f_1,\ldots,f_n$ zuvor homogenisieren muß:

$$\mathrm{Res}_{1,d_1,\ldots,d_n}(F_0,F_1,\ldots,F_n) \;=\; \pm\frac{\mathcal{DD}_0}{\mathcal{DD}_0'}. \tag{3.11}$$

Man beachte, daß hier f_0 beziehungsweise F_0 das ausgezeichnete Polynom ist und mit $\mathcal{DD}_0$ und $\mathcal{DD}_0'$ gerechnet wird.

Das folgende Lemma besagt, daß der Nenner $\mathcal{DD}_0'$ des Quotienten in (3.11) ein konstanter nummerischer Faktor aus $\mathbb{C}$ ist.

Lemma 33: Ist $\mathcal{DD}_0'$ aus (3.11) ungleich Null, so ist $\mathcal{DD}_0'$ unabhängig von den Koeffizienten $u_0,\ldots,u_n$ von f_0 beziehungsweise F_0. Damit ist $\mathcal{DD}_0' \in \mathbb{C}$, also ist die u-Resultante bis auf einen konstanten numerischen Faktor gleich der Determinante D_0^m.

Beweis: $\mathcal{DD}_i$ mit $i = 0$ aus (3.11) bedeutet, daß wir x_0 als letzte Variable bei der Konstruktion der Mengen $S_0, \ldots, S_n$ aus (2.15) betrachten. Also besteht S_0 aus den $d_0 \cdots d_n$ Monomen

$$S_0 = \{x_0^{a_0} x_1^{a_1} \cdots x_1^{a_1} | 0 \leq a_i \leq d_i - 1 \text{ für } i > 0, \sum_{i=0}^{n} a_i = d\},$$

wobei $d = 1 + d_1 + \cdots + d_n - n$ der kritische Grad nach Definition 10 ist. Alle Monome $x^a \in S_0$ sind nur durch $x_0^{d_0} = x_0$ teilbar, da alle Multiplikatoren $x_j^{a_j}$, $j = 1, \ldots, n$, des Monoms einen Grad $< d_i = deg(f_i)$ haben. Also sind nach Definition 12 alle Monome in S_0 reduziert. Bei der Konstruktion der Matrix $\mathcal{DM}_0$ indizieren die Elemente in S_0 alle Zeilen der Matrix mit monomialen Vielfachen von f_0. Bei der Konstruktion von $\mathcal{DM}'_0$ aus $\mathcal{DM}_0$ werden gerade diese Zeilen gestrichen, da sie von reduzierten Monomen indiziert werden. Also enthält $\mathcal{DM}'_0$ nur Koeffizienten von $f_1, \ldots, f_n$ und damit ist $\mathcal{DD}'_0$ unabhängig von den Koeffizienten von f_0.

□

Die Berechnung von $\text{Res}_{1,d_1,\ldots,d_n}(F_0, F_1, \ldots, F_n)$ läßt sich also vereinfachen, indem man den konstanten Faktor $\mathcal{DD}'_0$ vernachlässigt und sagt

$$\text{Res}_{1,d_1,\ldots,d_n}(F_0, F_1, \ldots, F_n) \; = C \cdot \mathcal{DD}_0, \;\; C \in \mathbb{C} \backslash \{0\}. \tag{3.12}$$

Voraussetzung dafür ist natürlich $\mathcal{DD}'_0 \neq 0$.

Andererseits kann man zur Berechung der u-Resultante auch die sparse Resultante verwenden. Diese ist vorteilhafter, da bei der multipolynomialen Resultante oft der Fall $\mathcal{DD}'_0 \neq 0$ eintritt, wenn die Polynome des Systems (3.1) nicht ausreichend dicht besetzt sind. Zusätzlich liefert die sparse Formulierung der Resultantenmatrix im Allgemeinen kleinere Matrizen und reduziert damit den Rechenaufwand. Mit Satz 29 ist die u-Resultante dann

$$\text{Res}_{\mathcal{A}_0,\mathcal{A}_1,\ldots,\mathcal{A}_n}(f_0, f_1, \ldots, f_n) \; = C \cdot \mathcal{SD}_0, \;\; C \in \mathbb{C} \backslash \{0\}. \tag{3.13}$$

Im weiteren Verlauf werden wir nicht mehr die eigentliche Resultante, sondern nur noch ihr Vielfaches $\mathcal{D}_0$ betrachten.

Notation: Da $\mathcal{D}_0$, wie die Resultante auch, ein Polynom in $\mathbb{C}[u_0, u_1, \ldots, u_n]$ ist, schreiben wir

$$\mathcal{D}_0(u_0, u_1, \ldots, u_n) \tag{3.14}$$

zur Hervorhebung der Abhängigkeit der Determinante $\mathcal{D}_0$ von $u_0, u_1, \ldots, u_n$.

3.2 Numerische Lösungen

Der im vorherigen Abschnitt beschriebene Ansatz zum Lösen eines Gleichungssystems der Form (3.1) erscheint auf den ersten Blick einleuchtend und einfach.

In der Praxis treten jedoch zwei Probleme auf: zum einen muß man zur Berechnung der u-Resultante Res die symbolische Determinante einer Matrix berechnen, deren Größe stark vom Grad und der Anzahl der Eingabe-Polynome abhängt. Für die multipolynomiale Resultante ist die Dimension der

Resultantenmatrix M durch $\binom{n+d}{d}$ nach Satz 11 und für die sparse Resultante durch $|\varepsilon|$ nach Satz 25 gegeben. Hat man die u-Resultante berechnet, muß man diese noch mit Hilfe einer multivariaten Faktorisierung in ihre Linearfaktoren zerlegen.

Diese beiden Rechenschritte können sich für große Eingabe-Systeme als sehr zeitaufwendig erweisen. Hier soll deshalb eine alternative Vorgehensweise vorgestellt werden, wie sie auch in [CLO97b] und [CM93] vorgeschlagen und beschrieben wird.

3.2.1 Spezialisierung der Koeffizienten

Betrachten wir das System $f_0 = f_1 = \cdots = f_n = 0$, $f_0, \ldots, f_n$ wie oben, das wir zur Berechnung der u-Resultante benutzt haben.

Die Idee besteht darin, einige der Koeffizienten $u_0, \ldots, u_n$ des zu (3.1) hinzugenommenen Polynoms

$$f_0 = u_0 + u_1 x_1 + \cdots + u_n x_n \tag{3.15}$$

zu spezialisieren, ihnen also konkrete Werte zuzuweisen. So verringert man die Anzahl der Variablen in der Resultante $\mathrm{Res} \in \mathbb{C}(u_0, \ldots, u_n)$ und erleichtert so das Berechnen der Determinante der Resultantenmatrix. Außerdem kann man so die multivariate Faktorisierung von Res umgehen, wie wir sehen werden.

Betrachtet man beispielsweise die folgende Spezialisierung

$$\begin{aligned} u_1 = u_2 = \cdots = u_{n-1} &= 0, \\ u_n &= -1, \end{aligned} \tag{3.16}$$

und wendet diese auf Satz 32 an, so ergibt sich die Gleichung

$$\mathrm{Res}(u_0 - x_n, f_1, \ldots, f_n) = C \prod_{p_i \in \mathbf{V}(f_1, \cdots, f_n)} (u_0 - a_{in})^{m(p_i)}, \tag{3.17}$$

wobei a_{in} die x_n-Koordinate von $p_i = (a_{i1}, \ldots, a_{in}) \in \mathbf{V}(f_1, \ldots, f_n)$ ist.

Diese derart spezialisierte u-Resultante ist offensichtlich ein univariates Polynom in u_0, dessen Nullstellen exakt den x_n-Koordinaten der Lösungen von (3.1) entsprechen.

Benutzen wir das Vielfache $\mathcal{D}_0(u_0, \ldots, u_n)$ der Resultante zur Darstellung derselben, so ergibt sich aus der obigen Spezialisierung

$$\mathcal{D}_0(u_0, 0, \ldots, 0, -1), \tag{3.18}$$

was wiederum ein univariates Polynom in u_0 ist und bis auf einen konstanten Faktor der Resultante $\mathrm{Res}(u_0 - x_n, f_1, \ldots, f_n)$ entspricht.

Allgemein definieren wir die Spezialisierung der $u_0, \ldots, u_n$ zur Berechnung der x_i-Koordiniaten der Lösung wie folgt:

$$\mathcal{D}_0^i(u_0) := \mathcal{D}_0(u_0, \ldots, u_n)_{u_1 = \cdots = u_{i-1} = 0, u_i = -1, u_{i+1} = \cdots = u_n = 0} \tag{3.19}$$

Da $\mathcal{D}_0^i$ durch eine symbolische Determinante gegeben ist, die nur von einer Variablen u_0 abhängt, ist die Berechnung um einiges leichter als im multivariaten Fall.

Benutzt man numerische Techniken wie zum Beispiel den Root-Finder nach Laguerre's Methode für univariate Polynome kleinen bis mittleren Grades (siehe Paragraph 3.2.4), so erhält man mit dem obigen Verfahren der Spezialisierung eine recht effiziente Methode zur Berechnung der x_i-Koordinaten eines Systems (3.1).

3.2.2 Zuordnung der Nullstellen

Hat man mit Hilfe der u-Resultante und der oben beschriebenen Methode zur Spezialisierung der Koeffizienten $u_0, \ldots, u_n$ die verschiedenen x_i-Koordinaten, $i = 1, \ldots, n$, der Lösung des Systems (3.1) bestimmt, so ergibt sich das Problem, welche der Koordinaten eine der Lösungen von (3.1) bilden. Das Vorgehen hierzu schließt sich nahtlos an die Spezialisierung der Koeffizienten an und wird ebenfalls in [CM93] beschrieben.

Angenommen, wir haben bereits $\mathcal{D}_0^1$ und $\mathcal{D}_0^2$ und damit die x_1- und x_2-Koordinaten der Lösung bestimmt, wobei $L_1 = \{a_1^{(1)}, \ldots, a_r^{(1)}\}$ beziehungsweise $L_2 = \{a_1^{(2)}, \ldots, a_r^{(2)}\}$ die entsprechende Menge der Nullstellen sei und $r = \deg(\mathcal{D}_0^1) = \deg(\mathcal{D}_0^2)$ deren Anzahl. Das Problem liegt darin, jedem $\alpha^1 \in L_1$ ein entsprechendes $\alpha^2 \in L_2$ zuzuordnen, so daß diese die 1. und 2. Koordinate einer Lösung $(\alpha^1, \alpha^2, *, \ldots, *)$ von (3.1) bilden.

Wir betrachten jetzt eine generische Kombination dieser beiden Koordinaten, um damit die Zuordnung zu den einzelnen Nullstellen vorzunehmen. Es sei

$$\hat{\mathcal{D}}_0^2(u_0) := \mathcal{D}_0(u_0, k_1, k_2, 0, \ldots, 0), \tag{3.20}$$

wobei $k_1, k_2 \in \mathbb{K}$ zwei positive, zufällig gewählte Zahlen seien. Mit Satz 32 erhalten wir:

$$\hat{\mathcal{D}}_0^2(u_0) = C \cdot \operatorname{Res}(u_0 + k_1 x_1 + k_2 x_2, f_1, \ldots, f_n) = C \cdot \prod (u_0 + (a_i^1 \cdot k_1 + a_j^2 \cdot k_2))^m. \tag{3.21}$$

$\hat{L}_2 = \{b_1^{(2)}, \ldots, b_r^{(2)}\}$ sei die Menge der Nullstellen von $\hat{\mathcal{D}}_0^2(u_0)$, wobei $b_i = a_i^{(1)} \cdot k_1 + a_j^{(2)} \cdot k_2$.

Um die Zugehörigkeit der einzelnen Elemente $\alpha^{(1)} \in L_1$ und $\alpha^{(2)} \in L_2$ zu bestimmen, berechnen wir alle Kombinationen der Form

$$\alpha^{(1)} \cdot k_1 + \alpha^{(2)} \cdot k_2 \tag{3.22}$$

und vergleichen diese mit den Elementen aus $\hat{L}_2$.

Da es sich bei der Abbildung der x_1- und x_2-Koordinaten auf u_1 und u_2 um eine generische Abbildung handelt, dürfen wir annehmen, daß es sich bei einer Übereinstimmung der Form

$$\beta^{(2)} = \alpha^{(1)} \cdot k_1 + \alpha^{(2)} \cdot k_2 \tag{3.23}$$

für ein $\beta^{(2)} \in \hat{L}_2$ um die Abbildung der Nullstellen $\alpha^{(1)}$ und $\alpha^{(2)}$ auf die x_1- und x_2-Koordinate handelt.

Beispiel 18: Wir betrachten das Gleichungssystem

$$f_1 = x^2 + 6x + 3y - 4 = 0, \quad f_2 = 2x + y^2 - 7y + 5 = 0$$

aus dem letzen Beispiel. Für $\mathcal{D}_0^1(u_0)$ und $\mathcal{D}_0^2(u_0)$ erhalten wir

$$\begin{aligned}\mathcal{D}_0^1(u_0) &= (u_0-1)(u_0+1)(u_0-2\sqrt{2})(u_0+2\sqrt{2}),\\ \mathcal{D}_0^2(u_0) &= (u_0+3)(u_0-3)(u_0-\sqrt{2})(u_0+\sqrt{2}),\end{aligned}$$

und für $\hat{\mathcal{D}}_0^2(u_0)$ mit $k_1=2$ und $k_2=4$

$$\hat{\mathcal{D}}_0^2(u_0) = -2(u_0-10)(u_0+10)(u_0^2-128)$$

Als Nullstellenmengen L_1, L_2 und $\hat{L}_2$ erhalten wir also

$$L_1 = \{-1,1,-2\sqrt{2},2\sqrt{2}\},\ \ L_2=\{-3,3,-\sqrt{2},\sqrt{2}\},\ \ \hat{L}_2=\{-10,10,-\sqrt{128},\sqrt{128}\}$$

So ist zum Beispiel, gemäß der Gleichung (3.23),

$$(1\cdot 2)+(-3\cdot 4) = -10 \in \hat{L}_2 \ \text{ aber } \ (-1\cdot 2)+(-3\cdot 4) = -14 \notin \hat{L}_2.$$

1 ist also die erste und -3 die zweite Koordinate der Nullstelle $(1,-3)$. -1 und -3 gehören jedoch nicht zusammen. ⊣

Auf die gleiche Weise ordnet man einander die x_3-, …, x_n-Koordinaten zu, wobei

$$\hat{\mathcal{D}}_0^j(u_0) := \mathcal{D}_0(u_0,k_1,k_2,\dots,k_j,0,\dots,0),\ \ j=2,\dots,n, \tag{3.24}$$

mit zufällig gewählten positiven Zahlen $k_1,\dots,k_j \in \mathbb{K}$. Man erhält allgemein die ersten l Koordinaten einer Lösung $(\alpha^{(1)},\dots,\alpha^{(l)},*,\dots,*)$ wenn für ein $\beta^{(l)} \in \hat{L}_l$ gilt

$$\beta^{(l)} = \alpha^{(1)}\cdot k_1+\cdots+\alpha^{(l)}\cdot k_l. \tag{3.25}$$

Wir erhalten damit den folgenden Algorithmus zur Zuordnung der Nullstellen.

Algorithmus 4: ZDN.

Eingabe:
$L_i = \{a_1^{(i)},\dots,a_r^{(i)}\}$ als die Mengen der Nullstellen von $\mathcal{D}_0^i(u_0)$, $i=1,\dots,n$, $r\in\mathbb{N}$
$\hat{L}_j = \{b_1^{(j)},\dots,b_r^{(j)}\}$ als Nullstellen von $\hat{\mathcal{D}}_0^j(u_0)$, $j=2,\dots,n$,
$K_j = \{k_1^{(j)},\dots,k_j^{(j)}\}$ mit $k_t^{(s)}\in\mathbb{Z}$.

Ausgabe: $L_i = \{\alpha_1^{(i)},\dots,\alpha_r^{(i)}\}$, $i=1,\dots,n$, so daß $(\alpha_s^{(1)},\dots,\alpha_s^{(n)})$ eine Nullstelle von (3.1) ist, $s=1,\dots,r$.

Algorithmus:

$(L_1,\dots,L_n) = \mathrm{ZDN}(L_1,\dots,L_n,\hat{L}_2,\dots,\hat{L}_n,K_2,\dots,K_n)$

begin

for $(ik=2,\dots,n)$ do

for $(ir = 1, \ldots, r)$ do

$tmp \leftarrow a_{ir}^{(1)} \cdot k_1^{(ik)} + \cdots + a_{ir}^{(ir-1)} \cdot k_{ik-1}^{(ik)}$

for $(t = ir, \ldots, r)$ do

if $(tmp + a_t^{(ir)} \cdot k_{ik}^{(ik)} \in \hat{L}_{ik})$
then

$vertausche(a_t^{(ir)}, a_{ir}^{(ir)})$

fi

od

od

od

return $(L_1, \ldots, L_n)$

end

3.2.3 Interpolation univariater Determinanten

Auch die Berechnung univariater Determinanten der Form $\mathcal{D}_0^i(u_0)$ beziehungsweise $\hat{\mathcal{D}}_0^j(u_0)$ ist noch eine zeitaufwendige Aufgabe. Allerdings ist mit Lemma 12 für multipolynomiale Resultanten und mit Satz 29 für sparse Resultanten der Grad der Polynome $\mathcal{D}_0^i(u_0)$ bzw. $\hat{\mathcal{D}}_0^j(u_0)$ bekannt, da diese jeweils homogene Polynome in den Koeffizienten $u_0, \ldots, u_n$ von f_0 sind.

Steht also kein leistungsfähiges Verfahren zur symbolischen Berechnung von Determinanten mit einer Variablen zur Verfügung, so bietet sich hier ein Verfahren zur numerischen Interpolation der Polynome $\mathcal{D}_0^i(u_0)$ und $\hat{\mathcal{D}}_0^j(u_0)$ geradezu an. Solche Verfahren für uni- und multivariate dichte und sparse Polynome werden zum Beispiel in [Zip90] vorgestellt.

Wir wollen Polynome der Form

$$\mathcal{D}_0^i(u_0) = a_d u_0^d + a_{d-1} u_0^{d-1} + \cdots + a_1 u_0 + a_0 \tag{3.26}$$

interpolieren, wobei d der Grad von $\mathrm{Res}(f_0, \ldots, f_n)$ als homogenes Polynom in den Koeffizienten von f_0 ist.

Folgendes sollte man sich zunächst verdeutlichen: einerseits kann man $\mathcal{D}_0^i$ als ein uns unbekanntes *Polynom* in $\mathbb{C}(u_0)$ auffassen, andererseits aber auch als die *Determinante der Resultantenmatrix*

$\mathcal{M}_0(u_0)$, die uns aus dem vorherigen Kapitel bekannt ist. Setzt man in $\mathcal{M}_0(u_0)$ für u_0 einen Wert ein, so erhält man eine rein numerische Matrix, deren Determinante ebenfalls ein numerischer Wert ist und sich leicht berechnen läßt.

Zur Interpolation des Polynoms $\mathcal{D}_0^i(u_0)$ wählt man $n+1$ verschiedene Punkte $z_0,\dots,z_n \in \mathbb{Z}$ und berechnet so

$$w_i := \det(\mathcal{M}_0(z_i)), \quad i = 1,\dots,n \tag{3.27}$$

als die Werte des Polynoms $\mathcal{D}_0^i(u_0)$ in den Punkten $z_0,\dots,z_n \in \mathbb{Z}$.

Jetzt läßt sich das folgende System aufstellen:

$$\begin{aligned}
\mathcal{D}_0^i(z_0) &= a_d z_0^d + a_{d-1} z_0^{d-1} + \cdots + a_1 z_0 + a_0 = w_0,\\
\mathcal{D}_0^i(z_1) &= a_d z_1^d + a_{d-1} z_1^{d-1} + \cdots + a_1 z_1 + a_0 = w_1,\\
&\vdots\\
\mathcal{D}_0^i(z_n) &= a_d z_n^d + a_{d-1} z_n^{d-1} + \cdots + a_1 z_n + a_0 = w_n,
\end{aligned} \tag{3.28}$$

wobei $a_0,\dots,a_n$ Koeffizienten aus $\mathbb{K}$ sind.

Dies ist offensichtlich ein Vandermondesystem wie in (1.8), welches sich schnell mit dem dort beschriebenen Verfahren lösen läßt.

3.2.4 Bestimmung der Nullstellen univariater Polynome

Will man die Nullstellen eines univariaten Polynoms $f \in \mathbb{C}[x]$ numerisch bestimmen, so bieten sich viele verschiedene Verfahren an. Hier wurde der Algorithmus von Laguerre ([PTVF92]) gewählt, ein iteratives Verfahren, welches innerhalb weniger Iterationsschritte gegen die Nullstellen von f konvergiert. Der Algorithmus eignet sich insbesondere zur Berechnung der komplexen Nullstellen univariater Polynome niedrigen bis mittleren Grades ($\deg(f) \lesssim 60$).

Für Polynome mit ausschließlich reellen Nullstellen ist gezeigt worden, daß der Algorithmus unabhängig vom Startpunkt gegen eine Nullstelle konvergiert. Für Polynome mit komplexen Nullstellen ist in der Theorie bislang wenig über die Konvergenz des Algorithmus bewiesen worden, jedoch legt die praktische Erfahrung nahe, daß nur in außergewöhnlichen Fällen keine Konvergenz auftritt.

Zur Motivation des Algorithmus von Laguerre, der hier nicht weiter bewiesen werden soll, dienen die folgenden Zusammenhänge zwischen dem Polynom und seinen Nullstellen und Ableitungen.

f sei ein univariates Polynom in $\mathbb{C}[x]$ vom Grad d mit

$$f(x) = (x-x_1)(x-x_2)\cdots(x-x_d) \tag{3.29}$$

und den Nullstellen $x_1,\dots,x_d \in \mathbb{C}$.

$$\ln|f(x)| = \ln|x-x_1| + \ln|x-x_2| + \cdots + \ln|x-x_d| \tag{3.30}$$

$$\frac{d\ln|f(x)|}{dx} = \frac{1}{x-x_1} + \frac{1}{x-x_2} + \cdots + \frac{1}{x-x_d} = \frac{f'}{f} =: g \tag{3.31}$$

$$\frac{d^2\ln|f(x)|}{d^2x} = \frac{1}{(x-x_1)^2} + \frac{1}{(x-x_2)^2} + \cdots + \frac{1}{(x-x_d)^2} = \left(\frac{f'}{f}\right)^2 - \frac{f''}{f} =: h \tag{3.32}$$

$\frac{d}{dx}$ bzw. $\frac{d^2}{d^2x}$ stehen dabei für die erste bzw. zweite Ableitung nach x.

Ausgehend von diesen Gleichungen macht der Algorithmus eine sehr gewagte Annahme ([Act90]): es wird angenommen, daß die gesuchte Nullstelle den Abstand a vom bislang angeratenen Wert x hat, wohingegen *alle anderen Nullstellen* den Abstand b von x haben.

$$x - x_1 = a; \quad x - x_i = b, \ \ i = 2, 3, \ldots, d. \tag{3.33}$$

Dann kann man (3.31) und (3.32) wie folgt ausdrücken:

$$\frac{1}{a} + \frac{d-1}{b} = g \tag{3.34}$$

$$\frac{1}{a^2} + \frac{d-1}{b^2} = h \tag{3.35}$$

was als Lösung für a

$$a = \frac{n}{g \pm \sqrt{(d-1)(dh-g^2)}} \tag{3.36}$$

liefert. Das Vorzeichen wird so gewählt, daß der Zähler am größten ist. Da der Faktor innerhalb der Quadratwurzel negativ sein kann, kann a komplex sein.

Der Algorithmus arbeitet iterativ auf die folgende Weise: für ein angeratenes x wird a nach (3.36) berechnet. Der nächste Iterationsschritt wird für $x = x - a$ ausgeführt. Dies wird solange fortgesetzt, bis a hinreichend klein, x also mit der gewünschten Genauigkeit bestimmt ist.

3.3 Zusammenfassender Algorithmus

Nun haben wir Algorithmen zur Konstruktion von Macaulay und sparsen Resultantenmatrizen, einen Algorithmus zur Bestimmung der Nullstellen univariater Polynome und wissen, wie man mit Hilfe der u-Resultante die Nullstellen eines Gleichungssystems bestimmen kann.

All diese Methoden wollen wir jetzt in einem Algorithmus zusammenfassen, der für ein eingegebenes Gleichungssystem die komplexen Nullstellen desselben berechnet.

Da die Beispiele bei der Beschreibung der Konstruktionsverfahren zur sparsen Resultantenmatrix bereits recht ausführlich waren, wird jeweils auf das zum gerade ausgeführten Schritt des Algorithmus passende Beispiel verwiesen. Die Macaulay Resultante wollen wir dabei vernachlässigen, da sie, wie die Auswertung der gerechneten Beispiele im nächsten Kapitel zeigen wird, mehr historische Bedeutung hat.

Algorithmus 5: UResSolve.

Eingabe: n Polynome in n Veränderlichen, $f_1,\ldots,f_n \in \mathbb{K}[x_1,\ldots,x_n]$.

Voraussetzung: $\mathbb{K}$ ist ein Körper der Charakteristik 0, zum Beispiel $\mathbb{Q},\mathbb{R}$ oder $\mathbb{C}$ und $I = \langle f_1,\ldots,f_n\rangle$ ein nulldimensionales Ideal.

Ausgabe: Liste aller komplexen Nullstellen des Gleichungssystems $f_1 = f_2 = \cdots = f_n = 0$.

Algorithmus:

Schritt 1: Konstruktion der Resultantenmatrix
Aufstellen der Resultantenmatrix $\mathcal{M}(u_0,\ldots,u_n)$ für $f_0, f_1,\ldots,f_n$, wobei $f_1,\ldots,f_n$ die Eingabepolynome sind und f_0 das lineare Polynom $f_0 = u_0 + u_1x_1 + \cdots + u_nx_n$ mit $u_0,\ldots,u_n$ unabhängige Variablen.

Bemerkung 1.1: Aus Effizienzgründen wird die sparse Resultantenmatrix (Abschnitt 2.3) verwendet, als Alternative steht jedoch auch die Macaulay Resultantenmatrix (Abschnitt 2.2), auch multipolynomiale Resultantenmatrix genannt, zur Verfügung.

Schritt 1.1: Berechnung der Newton Polytope
Sei $Q_i = \mathrm{Conv}(\mathrm{tr}(f_i))$ das Newton Polytop von f_i, $i = 0,\ldots,n$. Die Minkowsi-Summe von $Q_0,\ldots,Q_n$ bezeichnen wir mit Q.

• Siehe Beispiel 9 auf Seite 30.

Schritt 1.2: Liften der Newton Polytope
Seien $l_i = L_{i_1}x_1 + \cdots + L_{i_n}x_n$ generische Linearformen mit zufällig gewählten $L_{i_j} \in \mathbb{K}$. Dann ist $\hat{Q}_i := \{(p, l_i(p) \mid p \in Q_i\}$ und

$$\hat{Q} = \hat{Q}_0 + \cdots + \hat{Q}_n \subset \mathbb{R}^{n+1}, \tag{3.37}$$

wobei $\hat{Q}$ wieder konvex ist.

• Siehe Beispiel 10 auf Seite 35.

Schritt 1.3: Bestimmung der unteren Hülle
Bestimme die untere Hülle $s(Q)$ von $\hat{Q}$. Durch die Triangulierung der unteren Hülle, das heißt, die Zerlegung in ihre Seiten, erhält man eine natürliche polyhedrale Subdivision von $\hat{Q}$. Mit $\hat{\Delta}$ bezeichnen wir die Menge aller Facetten von $s(Q)$. $\hat{\Delta}$ ist also die polyhedrale Subdivision von $s(Q)$ in maximale Zellen.

Die untere Hülle $s(Q)$ von $\hat{Q}$ ist dabei durch die Menge aller Seiten von $\hat{Q}$ gegeben, deren innerer Normalenvektor eine positive letzte Koordinate hat.

• Siehe Beispiel 11 auf Seite 36 zur Berechnung der unteren Hülle und insbesondere Beispiel 14 auf Seite 38 zur Triangulierung und Subdivision.

Schritt 1.4: Projektion
Bei der Projektion von $s(Q)$ von $\mathbb{R}^{n+1}$ nach $\mathbb{R}^n$ unter Beachtung der Subdivision ergibt sich die polyhedrale Subdivision Δ von Q.

- Siehe Beispiel 15 auf Seite 41.

Schritt 1.4: Shiften von Δ
Um jedem Gitterpunkt in Q eine eindeutige Zelle zuordnen zu können, permutiert man Q mit Hilfe eines Vektors $\delta \in \mathbb{R}^n$ so, daß die ganzzahligen Gitterpunkte, die zuvor auf den Eckpunkten der maximalen Zellen lagen, jetzt entweder im Inneren einer maximalen Zelle zu liegen kommt oder sich außerhalb von Δ befindet. Es sei

$$\mathcal{E} = (Q+\delta) \cap \mathbb{Z}^n \subset \mathbb{Z}^n.$$

Schritt 1.5: Aufstellen der Resultantenmatrix
Für jeden Gitterpunkt $e = (e_1, \ldots, e_n) \in \mathcal{E}$ führt man die folgenden Schritte aus und konstruiert so die Resultantenmatrix:

Schritt 1.5.1:
Bestimme die *optimale* Minkowski-Summe der Zelle $R_j = F_0 + \ldots + F_n$, die den Punkt e enthält. Dabei ist F_i eine Seite von Q_i.

Schritt 1.5.2:
Bestimme dasjenige 0-dimensionale $F_i = q = (q_1, \ldots, q_n) \in \mathrm{tr}(f_i)$ mit dem größten Index i. (Es existiert mindestens ein solches F_i.)

Schritt 1.5.3:
Eine Zeile der quadratischen sparsen Resultantenmatrix $\mathcal{SM}$ ergibt sich aus den Koeffizienten des Polynoms

$$g = \frac{x_1^{e_1} x_2^{e_2} \cdots x_n^{e_n}}{x_1^{q_1} x_2^{q_2} \cdots x_n^{q_n}} \cdot f_i. \tag{3.38}$$

- Zur Konstruktion der Matrix siehe Beispiel 16 auf Seite 47.

Bemerkung 1.2: Man kann zeigen, daß jeder Exponentenvektor eines Monoms in g auch in $\mathcal{E}$ enthalten ist. $\mathcal{M}$ hat die Dimension $|\mathcal{E}|$ und ihre Spalten werden durch die lexikographisch geordneten Elemente von $\mathcal{E}$ indiziert.

Bemerkung 1.3: Tatsächich führt man Schritt 1.5.1 bis 1.5.3 nicht wirklich aus, sondern verwendet Algorithmus 2 zur Berechnung der Menge $\mathcal{E}$ und Algorithmus 3 zur Bestimmung des Punktes $q \in \mathrm{tr}(f_i)$. Beide Algorithmen basieren auf linearer Optimierung und vermeiden so die aufwendige Berechnung der Minkowski-Summen und der unteren Hülle $s(Q)$.

Bemerkung 1.4: Intern wird die Matrix $\mathcal{M}$ unter Berücksichtigung ihrer sparsen Struktur gespeichert, das heißt, es werden nur die Einträge ungleich Null berücksichtigt.

Schritt 2: Erste Spezialisierung von $\mathcal{M}(u_0, \ldots, u_n)$
Die Spezialisierung der Resultantenmatrix $\mathcal{M}(u_0, \ldots, u_n)$ in $u_1, \ldots, u_n$ sei definiert durch

$$\mathcal{M}^i(u_0) := \mathcal{M}(u_0, 0, \ldots, 0, -1, 0, \ldots, 0), \tag{3.39}$$

wobei u_i durch -1 ersetzt wurde und $u_1 = \ldots = u_{i-1} = u_{i+1} = \ldots = u_n = 0$, $i = 1, \ldots, n$.

Die Determinante von $\mathcal{M}^i(u_0)$ sei $\mathcal{D}^i(u_0) := \det(\mathcal{M}^i(u_0))$.

Wir spezialisieren die Resultantenmatrix $\mathcal{M}$ in n Schritten:

$$\begin{array}{lrcl} \textbf{Schritt 2.1:} & \mathcal{D}^1(u_0) & = & \det(\mathcal{M}^i(u_0)) \\ \textbf{Schritt 2.2:} & \mathcal{D}^2(u_0) & = & \det(\mathcal{M}^2(u_0)) \\ \vdots & & \vdots & \\ \textbf{Schritt 2.n:} & \mathcal{D}^n(u_0) & = & \det(\mathcal{M}^n(u_0)) \end{array}$$

Bemerkung 2.1: In diesem wie auch dem nächsten Schritt gilt es, ein möglichst effizientes Verfahren zur Berechnung der Determinante einer sparsen Matrix zu verwenden. Konkret kommt hier eine Variante des *Algorithmus von Bareiss* zum Einsatz, die mit Hinblick auf die Berechnung der Determinante optimiert wurde.

Schritt 3: Zweite Spezialisierung von $\mathcal{M}(u_0,\dots,u_n)$
Die Spezialisierung der Resultantenmatrix $\mathcal{M}(u_0,\dots,u_n)$ in $u_1,\dots,u_n$ sei definiert durch

$$\hat{\mathcal{M}}^j(u_0) := \mathcal{M}(u_0,k_1,\dots,k_j,0,\dots,0), \tag{3.40}$$

wobei $k_j \in \mathbb{K}$ zufällig gewählte Zahlen sind,$j = 2,\dots,n$. Aus Effizienzgründen wählt man k_j möglichst klein und beschränkt sich auf $k_j \in \mathbb{N}$.

Die Determinante von $\hat{\mathcal{M}}^j(u_0)$ sei $\hat{\mathcal{D}}^j(u_0) := \det(\hat{\mathcal{M}}^j(u_0))$.

Wir spezialisieren die Resultantenmatrix $\mathcal{M}$ in $n-1$ Schritten:

$$\begin{array}{lrcl} \textbf{Schritt 3.1:} & \hat{\mathcal{D}}^2(u_0) & = & \det(\hat{\mathcal{M}}^2(u_0)) \\ \textbf{Schritt 3.2:} & \hat{\mathcal{D}}^3(u_0) & = & \det(\hat{\mathcal{M}}^3(u_0)) \\ \vdots & & \vdots & \\ \textbf{Schritt 3.n-1:} & \hat{\mathcal{D}}^n(u_0) & = & \det(\hat{\mathcal{M}}^n(u_0)) \end{array}$$

Bemerkung 3.1: Die Spezialisierung der Resultantenmatrix wie in Schritt 3 und 4 entspricht dem in den Abschnitten 3.2.1 und 3.2.2 besprochenen Verfahren.

Schritt 4: Berechnung aller Nullstellen
Berechnung sämtlicher komplexer Nullstellen von $\mathcal{D}^i(u_0)$, $i = 1,\dots,n$, und $\hat{\mathcal{D}}^j(u_0)$, $j = 2,\dots,n$, aus Schritt 2 und 3 unter Verwendung des *Algorithmus von Laguerre* aus Abschnitt 3.2.4.

Sei $L_i = \{a_1^{(i)},\dots,a_r^{(i)}\}$ die Menge aller komplexen Nullstellen von $\mathcal{D}^i(u_0)$, $i = 1,\dots,n$ und $r = \deg(\mathcal{D}^i)$. Alle Elemente aus L_i repräsentieren also die i-te Koordinate einer Lösung (p_1, ... ,p_n) von $f_1 = \ldots = f_n = 0$. Als offensichtliches Problem ergibt sich hier, welche einzelnen $p \in L_i$ mit anderen $q \in L_k$ gemeinsam eine Lösung von $f_1 = \ldots = f_n = 0$ bilden.

Sei $\hat{L}_j = \{b_1^{(j)},\dots,b_r^{(j)}\}$ die Menge aller komplexen Nullstellen von $\hat{\mathcal{D}}^j(u_0)$, $j = 2,\dots,n$.

Bemerkung 4.1: Allgemein gilt hier $r = \deg(\mathcal{D}^i) = \deg(\hat{\mathcal{D}}^j)$ für $i \in [1,\dots,n]$, $j \in [2,\dots,n]$.

Schritt 5: Zuordnung der Nullstellen
Zuordnung der in Schritt 4 berechneten Koordinaten der Nullstellen (Abschnitt 3.2.2).

Zwei Elemente $\alpha^{(1)} \in L_1$ und $\alpha^{(2)} \in L_2$, sind die 1. und 2. Koordinate einer Nullstelle $(\alpha^{(1)}, \alpha^{(2)}, *, \ldots, *)$ von $f_1 = \ldots = f_n = 0$ genau dann, wenn für ein $\beta^{(2)} \in \hat{L}_2$ gilt

$$\alpha^{(1)} \cdot k_1 + \alpha^{(2)} \cdot k_2 = \beta^{(2)}. \tag{3.41}$$

Analog gilt für die u-te Koordinate

$$\alpha^{(1)} \cdot k_1 + \cdots + \alpha^{(u)} \cdot k_u = \beta^{(u)}. \tag{3.42}$$

für ein $\beta^{(u)} \in \hat{L}_u$ und $\alpha^{(i)} \in L_i$, $i = 1, \ldots, u$.

4 Auswertung

Eines der Ziele der vorliegenden Arbeit war es, einen auf Resultanten basierenden Algorithmus zur Bestimmung der Nullstellenmenge nulldimensionaler Systeme in SINGULAR ([GPS+99]) zu implementieren und die praktische Anwendbarkeit der u-Resultanten-Methode zu überprüfen. In diesem Kapitel steht die Auswertung im Mittelpunkt, die den Nutzen der Resultanten-basierten Methode untersuchen soll. Alle in dieser Arbeit vorgestellten Algorithmen wurden in SINGULAR implementiert. Zu Beginn wird kurz erkärt, über welche Kommandos die Algorithmen zu erreichen sind. Im nächsten Abschnitt wird motiviert, anhand welcher Beispiele die Frage nach dem Nutzen beantwortet werden soll. Die Ergebnisse der verschiedenen Tests werden dann in den verbleibenden Abschnitten diskutiert. Die verwendeten Beispiele finden sich ganz am Ende dieses Kapitels.

SINGULAR besitzt zwar eine eigene Interpretersprache, aus Geschwindigkeitsgründen wurde jedoch von Beginn an eine Realisierung in C/C++ im SINGULAR-Kern selbst angestrebt. Ergebnis dieser Bemühungen ist das neue SINGULAR-Kommando `uressolve`. Die Syntax lautet

```
list = uressolve( ideal i, int k, int l, int m );
```

Dieses Kommando berechnet mit Hilfe der im 3. Kapitel beschriebenen u-Resultante unter Verwendung der sparsen Resultante (`k=0`) aus Abschnitt 2.3 oder der multipolynomialen Resultante (`k=1`) aus Abschnitt 2.2 die Nullstellen einens gegebenen nulldimensionalen Ideals, wobei das Ideal aus ebensovielen Polynomen bestehen muß, wie der zugrundeliegende Polynomring Variablen besitzt. Wie wir im Rahmen der Auswertung sehen werden, ist die Verwendung der sparsen Formulierung der Resultante der multipolynomialen vorzuziehen. `l` gibt die Präzision der Gleitkommazahlen des Ergebnisses an, sofern $\mathbb{K} = \mathbb{Q}$ und `m` die Anzahl der Iterationen bei der Approximation der Nullstellen mit dem Laguerre Algorithmus.

Zusätzlich wurden die Kommandos `mpresmat`, `vandermonde`, `convhull`, `simplex` und `laguerre` implementiert, die dem Benutzer verschiedene Schritte des von `uressolve` verwendeten Algorithmus zur Verfügung stellen. Im Einzelnen sind dies:

- `matrix = mpresmat( ideal i, int k );`
  Führt Schritt 1 des Algorithmus 5 aus, liefert also wahlweise die sparse oder Macaulay Resultantenmatrix zurück.

  Zu den einzelnen Parametern siehe Berschreibung der Biblioteks-Funktion

  ```
  mp_res_mat( i [, k ] );
  ```

  weiter unten. Man beachte, daß die Parameter hier nicht optional sind.

- `list = laguerre( poly p, int l, int m );`
  Implementierung des in Abschnitt 3.2.4 vorgestellten Algorithmus von Laguerre zur Berechnung aller komplexen Nullstellen eines univariaten Polynoms. Wird auch in Schritt 4 von Algorithmus 5 verwendet.

  Zu den einzelnen Parametern siehe Berschreibung der Biblioteks-Funktion

  `laguerre_solve( p [, l, m ] );`

  weiter unten. Man beachte, daß die Parameter hier nicht optional sind.

- `poly = vandermonde( ideal p, ideal w, int d );`
  Implementiert das in Abschnitt 3.2.3 vorgeschlagene Verfahren zur Interpolation von Polynomen, deren Wert an verschiedenen Punkten bekannt ist. Siehe auch Algorithmus 1.

  Siehe Berschreibung der Biblioteks-Funktion

  `interpolate( p, w, d );`

  weiter unten. Man beachte, daß die Parameter hier nicht optional sind.

- `ideal = convhull( ideal i );`
  Berechnet die konvexen Hüllen der Träger der Polynome im Ideal `i`, also deren Newton Polytope. Liefert das Eingabeideal zurück, wobei die Koeffizienten der Monome, deren Träger kein Eckpunkt eines Newton Polytops ist, auf Null gesetzt werden.

- `list = simplex( matrix m, int cons, int n, int c1, int c2, int c3 );`
  Implementation des Simplex-Algorithmus. Die Matrix `m` enthält das Simplex-Tableau. `cons` gibt die Anzahl der Nebenbedingungen an, `n` die Anzahl der Basisvariablen (Variablen auf der rechten Seite der (Un-)Gleichungen). `c1` gibt die Anzahl der linearen Ungleichungen der Form $\leq$ vor, `c2` für $\geq$ und `c3` für $=$. Man beachte, daß $cons = c1 + c2 + c3$ sein muß. Die Anzahl der Zeilen von `m` ist gerade $cons + 1$, die Anzahl der Spalten $n + 1$.

  Zurückgeliefert wird eine Liste mit den folgenden Einträgen:

  1. die Matrix `m` nach der Ausführung des Simplex-Algorithmus. Der Eintrag a_{11} der Matrix enthält den Wert des Optimums, sofern eins gefunden werden konnte;
  2. ein Integer, der die folgenden Werte annehmen kann:
     - 0: es wurde eine endliche Lösung gefunden;
     - 1: die Zielfunktion ist nicht beschränkt, weshalb keine Lösung gefunden werden konnte;
     - -1: es existiert keine Lösung.
  3. einen `intvect`. Der Wert i an der j-ten Koordinate des Vektors gibt die Nichtbasisvariable an, die jetzt durch die $j+1$-te Zeile von `m` repräsentiert wird, $j = 1, \ldots, cons$;
  4. einen `intvect`;
  5. die Anzahl der Nebenbedingungen als Integer;
  6. die Anzahl der Variablen als Integer.

  Zur Definition der verwendeten Begriffe und einer Beschreibung des Simplex-Algorithmus sei auf [Sch97], Kapitel 2, verwiesen. Eine Implementation in C findet man zum Beispiel in [PTVF92], Kapitel 10.

Um dem SINGULAR-Benutzer die Anwendung einiger der vorgestellten Kommandos zu erleichtern, wurde die Bibliotek `solve.lib` erstellt, die die oben aufgeführten Kommandos um eine Hilfestellung und kleine Beispiele erweitert. Außerdem genügt es bei fast allen Befehlen, das Gleichungssystem in Form eines Ideals anzugeben, da die anderen Parameter optional sind.

Im folgenden die Befehle, die `solve.lib` dem Benutzer zur Verfühgung stellt:

- `list = ures_solve( ideal i [, int k, int l, int m ] );`
  i ist ein nulldimensionales Ideal, das aus ebensovielen Polynomen bestehen muß, wie der definierte Polynomring Variablen enthält. Diese Variablen müssen sämtlich in den Polynomen vorkommen. Gibt eine Liste aller (komplexen) Nullstellen von $i = 0$ zurück. Die einzelnen Elemente der Liste werden durch den SINGULAR-Datentyp `number` repräsentiert, falls $\mathbb{K} = \mathbb{C}$, sonst durch `string`'s.

  Die Parameter `k`, `l` und `m` sind optional und bieten die folgenden Möglichkeiten:

  - `k=0`: verwende die sparse Resultantenmatrix nach Gelfand, Kapranov and Zelevinsky (Defaultwert);
  - `k=1`: verwende die Resultantenmatrix nach Macaulay;
  - `l>0`: definiert die Präzision bei der Bestimmung der komplexen Nullstellen, falls $\mathbb{K} = \mathbb{Q}$ (Defaultwert ist 30);
  - `m=0,1,2`: gibt die Anzahl der Iterationen bei der Approximation der Nullstellen mit dem Laguerre Algorithmus an (Defaultwert ist 2).

- `modul = mp_res_mat( ideal i [, int k ] );`
  i ist ein nulldimensionales Ideal, das aus ebensovielen Polynomen bestehen muß, wie der definierte Polynomring Variablen enthält. Diese Variablen müßen sämtlich in den Polynomen vorkommen. Liefert die Resultantenmatrix als Modul zurück.

  Der Parameter `k` ist optional und bieten die folgenden Möglichkeiten:

  - `k=0`: verwende die sparse Resultantenmatrix nach Gelfand, Kapranov and Zelevinsky (Defaultwert);
  - `k=1`: verwende die Resultantenmatrix nach Macaulay;

- `list = laguerre_solve( poly p [, int l, int m ] );`
  Liefert eine Liste aller (komplexen) Nullstellen des univariaten Polynoms p zurück. Die einzelnen Elemente der Liste werden durch den SINGULAR-Datentyp `number` repräsentiert, falls $\mathbb{K} = \mathbb{C}$, sonst durch `string`'s.

  Die Parameter `l` und `m` sind optional und bieten die folgenden Möglichkeiten:

  - `l>0`: definiert die Präzision bei der Bestimmung der komplexen Nullstellen, falls $\mathbb{K} = \mathbb{Q}$ (Defaultwert ist 30);
  - `m=0,1,2`: gibt die Anzahl der Iterationen bei der Approximation der Nullstellen mit dem Laguerre Algorithmus an (Defaultwert ist 2).

- `poly = interpolate( ideal p, ideal w, int d );`
  Interpoliert ein Polynom f vom Grad $d := deg(f)$ anhand des Berechnungspunkts $p =$

$(p_1,\ldots,p_n)$, n Anzahl der Variablen, und der Ergebnisse w, so daß für $i = 0,1,\ldots,(d+1)^n$ gilt

$$f((p_1^i, p_2^i, \ldots, p_n^i)) = w[i].$$

Bei der Programmierung der numerischen Algorithmen im SINGULAR-Kernel wurde für die Darstellung der Gleitkommazahlen die GNU MP Library verwendet, die bereits bei der Darstellung der ganzen und rationalen Zahlen in SINGULAR zur Anwendung kommt. Mit Hilfe dieser Library wurden sowohl die reellen als auch die komplexen Zahlen implementiert, so daß dem Benutzer von SINGULAR jetzt zwei neue Polynomringe zur Verfügung stehen, wobei die Genauigkeit der Nachkommastellen beliebig vorgegeben werden kann. Die Ringe lassen sich mit

```
ring name = (real,<Nachkommastellen>), <Variablen>,<Ordnung>;
```

beziehungsweise

```
ring name = (real,<Nachkommastellen>, <Komplexe Einheit>),
                 <Variablen>,<Ordnung>;
```

definieren.

4.1 Die Testumgebung

Bei der Auswertung stand die Frage im Mittelpunkt, ob der auf der Resultante basierende Algorithmus zum Lösen von Gleichungssystemen eine effektive Alternative zu den Algorithmen bietet, die reduzierte Standartbasen verwenden. Zum Vergleich bietet sich hier der ebenfalls im SINGULAR-Kern implementierte FGLM-Algorithmus ([Wic97]) an, der sehr effektiv bei der Berechnung von Gröbnerbasen nulldimensionaler Ideale ist. Durch suksessives Lösen und rekursives Einsetzen der so gewonnenen Nullstellen erhält man sämtliche Lösungen.

Ausserdem soll mit populären Computeralgebrasystemen wie Mathematica und Maple verglichen werden, wobei möglichst die Befehle zur Anwendung kommen sollten, die laut Anleitung in der Lage sind, alle komplexen Nullstellen zu finden. Zusätzlich wird noch mit REDUCE verglichen, dessen Lösungsalgorithmus Gröbnerbasen verwendet und ebenfalls numerische Lösungen liefert.

Die für die Zeitvergleiche verwendeten Ideale finden sich, zum Teil mit Quellenangaben, am Ende des Kapitels. Gerechnet wurde immer über den angegebenen Körper der Koeffizienten. Sämtliche in den Tabellen aufgeführten Zeiten sind in Sekunden angegeben, wobei Angaben von unter einer Sekunde mit Vorsicht zu genießen sind, da sie im Rahmen der Meßgenauigkeit liegen. Die Rechenzeit pro Beispiel wurde auf 5 Stunden (18000 Sekunden) beschränkt.

Gerechnet wurde auf einer Hewlet Packard HP 160C (Serie 9000/780, CPU PA-RISC 8000, 160MHz) mit 512 MByte Hauptspeicher unter HPUX 10.20. SINGULAR wurde mit dem frei verfügbaren Compiler *egcs* in der Version 1.0.3 und den Optionen `-O6 -fomit-frame-pointer -mschedule=7200` übersetzt, um eine möglichst gute Anpassung des Programms an die CPU zu erreichen.

In Tabelle 4.1 sind die Daten der nulldimensionalen Beispiele aufgeführt, die Spalten enthalten der Reihenfolge nach:

- die Anzahl der Nullstellen (entspricht der Vektorraumdimension),
- die Anzahl der Variablen,
- die Größe der sparsen Resultantenmatrix $\mathcal{SM}_0$,
- die Größe der Macaulay Resultantenmatrix $\mathcal{DM}_0$,
- die Größe der Macaulay Resultanten-Submatrix $\mathcal{DM}'_0$,
- die Anzahl der Nullstellen des homogenisierten Ideals in $\mathbb{P}^n(\mathbb{C})$.

Dabei entspricht die Anzahl der Variablen der Anzahl der im Ideal enthaltenen Polynome.

Ist ein „$\times$“ statt einer Zahl eingetragen, so konnte die Resultantenmatrix aufgrund Speicherplatzmangels nicht aufgestellt werden, was bei der Konstruktion der Macaulay Resultantenmatrix des öfteren eintrat.

			Resultante			
			sparse	Macaulay		
Beispiel	#Nullstellen	#Var	$\lvert\mathcal{SM}_0\rvert$	$\lvert\mathcal{DM}_0\rvert$	$\lvert\mathcal{DM}'_0\rvert$	dim
1	4	2	10	10	2	4
2	20	3	127	560	366	120
3	4	2	8	10	2	4
4	8	3	26	35	15	8
5	8	3	35	35	15	8
7	40	3	158	220	142	40
8	28	2	56	66	27	28
9	8	3	35	35	15	8
11	24	3	115	165	96	36
12	24	3	115	120	70	24
13	6	3	26	35	18	6
14	24	4	216	330	256	24
15	8	3	51	56	32	9
16	83	3	337	455	302	90
18	5	2	16	28	9	12
20	16	3	112	286	174	64
23	168	6	5860	$\times$	$\times$	$\times$
24	16	4	68	126	78	16
26	56	2	99	136	56	64
27	16	4	126	126	78	16
28	42	10	13613	$\times$	$\times$	$\times$
29	46	8	7529	$\times$	$\times$	$\times$
30	48	4	599	1365	1053	144
31	23	2	48	253	110	121
33	120	7	6322	6435	5859	$\times$
40	20	3	128	560	366	120
41	2	3	10	20	8	4

Fortsetzung auf der nächsten Seite

Fortsetzung

Beispiel	#Nullstellen	#Var	Resultante sparse $\lvert\mathcal{SM}_0\rvert$	Macaulay $\lvert\mathcal{DM}_0\rvert$	$\lvert\mathcal{DM}'_0\rvert$	dim
42	30	10	8846	×	×	×
43	48	4	601	1365	1053	144
45	6	4	55	70	42	8
46	73	4	613	715	526	81
51	64	3	258	455	290	100
54	156	6	5487	×	×	×
55	819	7	49275	×	×	×
57	46	8	7529	×	×	×
58	120	4	1365	1365	1091	120
59	16	4	126	126	78	16
60	30	3	121	286	174	64
61	28	3	127	286	174	64
62	31	3	163	286	174	64

Tabelle 4.1: Die Daten der verwendeten Beispiele.

Man sieht, daß die Resultantenmatrix schon für recht kleine Beispiele eine ansehnliche Größe erreichen kann. Hier kommt einem die Sparseheit der Matrix entgegen, so daß sich der benötigte Speicherplatz in Grenzen hält. Abbildung 4.1 zeigt die sparse Resultantenmatrix $\mathcal{SM}_0$ zu Beispiel 30. Es handelt sich hierbei um eine 601×601-Matrix mit 361201 möglichen Einträgen, wovon 4062 ungleich Null sind, was einem Besetzungsgrad von etwa 1,125% entspricht. Die Matrix kann also mit Recht als sparse bezeichnet werden.

4.2 Vergleich Macaulay – sparse Resultante

Zunächst, bevor in den nächsten Abschnitten mit anderen Methoden verglichen wird, soll die Effizienz der in SINGULAR implementierten Resultanten-Methode unter Verwendung der sparsen und der Macaulay Resultante gegenübergestellt werden. Die Macaulay Resultante läßt sich nur für homogene Ideale konstruieren, weshalb hierfür die Ideale aus den Beispielen jeweils unter Hinzunahme einer Variable homogenisiert werden. Gerechnet wurde mit einer Genauigkeit von 30 Stellen für Fließkommazahlen.

In den Tabellen 4.2 und 4.3 werden die Zeiten aufgeführt, die zum Lösen der Beispiele unter Verwendung der Macaulay und der sparsen Resultante gebraucht werden. Die zweite Spalte enthält jeweils die benötigte Gesammtzeit, während in den restlichen Spalten die Zeiten für die einzelnen Schritte von Algorithmus 5 aufgeführt sind. Noch einmal zur Erinnerung:

- **Schritt 1:** Konstruktion der Resultantenmatrix.

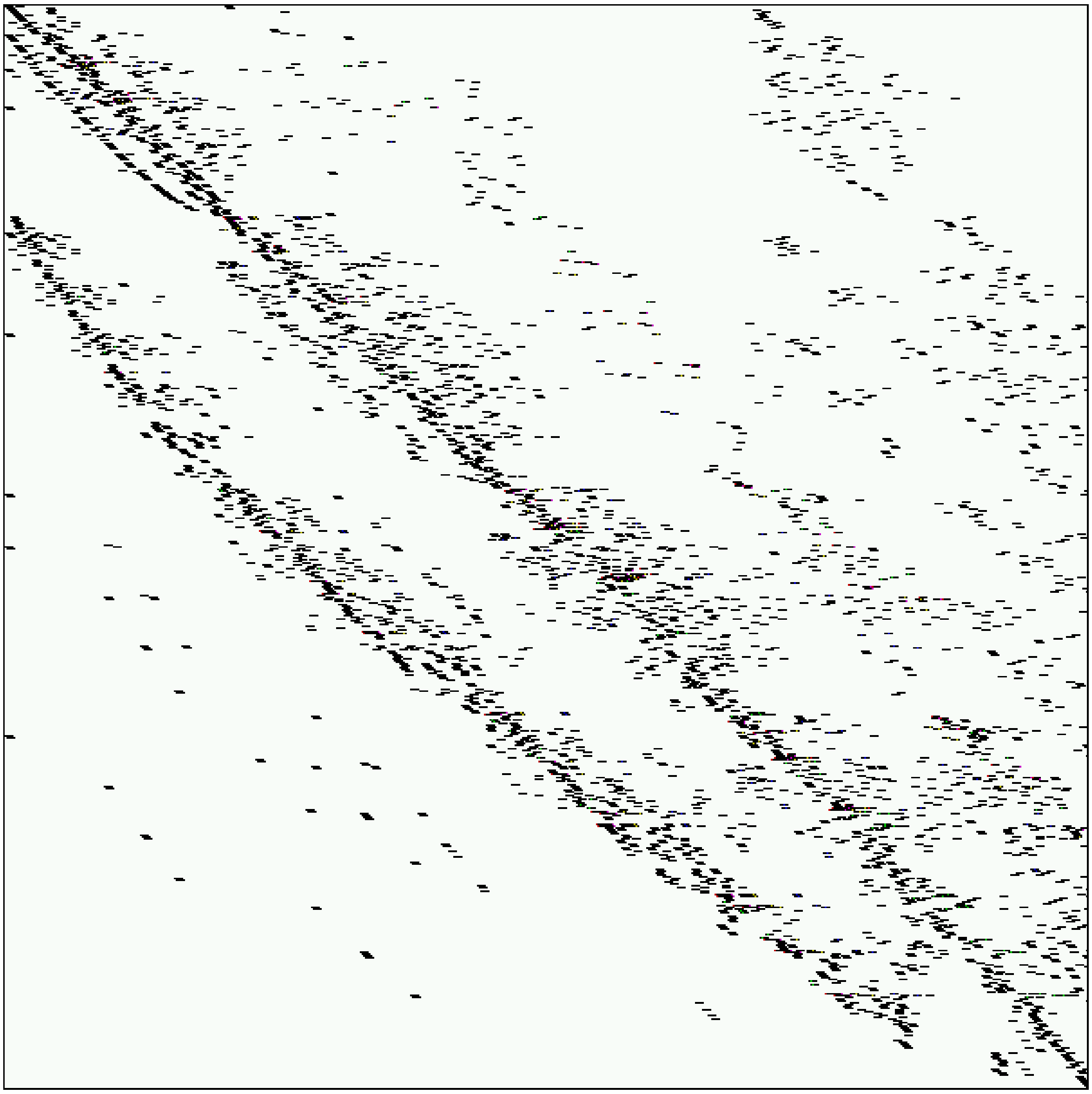

Abbildung 4.1: Die sparse 601 × 601-Resultantenmatirx zu Beispiel 30.

- **Schritt 2:** Erste Spezialisierung von $\mathcal{M}(u_0,\ldots,u_n)$.
- **Schritt 3:** Zweite Spezialisierung von $\mathcal{M}(u_0,\ldots,u_n)$.
- **Schritt 4:** Berechnung aller (komplexen) Nullstellen.
- **Schritt 5:** Zuordnung der Nullstellen.

Ist bei einem der Schritte des Algorithmus ein Fehler aufgetreten, so wird dies durch einen der folgenden Einträge dargestellt:

- *mem*
  Dieser Schritt konnte aufgrund Speichermangels nicht zu Ende geführt werden.
- *time*
  Die vorgegebene Rechenzeit von 5 Stunden wurde überschritten.
- *itr*
  Der Algorithmus von Laguerre 3.2.4 terminierte nicht innerhalb der vorgegebenen Anzahl von Iterationsschritten (Schritt 4).
- *m* (als Index)
  Die Zuordnung der einzelnen Koordinaten der Nullstellen war nicht erfolgreich (Schritt 5).
- *sing*
  Die Matrix $\mathcal{DM}'_0$ ist singulär. (Dies kann natürlich nur bei der Verwendung von Macaulay Resultanten auftreten.)

Die restlichen Einträge sind dann mit einem $\times$ gekennzeichnet.

In der letzten Zeile ist jeweils der Durchschnitt der einzelnen Zeiten angegeben, wobei Zeilen, in denen ein $\times$ vorkommt, nicht berücksichtigt wurden. Die Durchschnittszeiten geben einen Überblick darüber, wie sich die Zeiten für die einzelnen Schritte zueinander verhalten.

Zuerst die Zeiten bei der Verwendung der Macaulay Resultante:

		Algorithmus 5, Macaulay Resultante					
Bsp.	#Var	Summe	Schritt 1	Schritt 2	Schritt 3	Schritt 4	Schritt 5
1	2	0,26	0,01	0,03	0,14	0,08	0,00
2	3	$\times$	0,76	*sing*	$\times$	$\times$	$\times$
3	2	0,26	0,01	0,03	0,14	0,08	0,00
4	3	11,01	0,00	3,93	6,56	0,48	0,02
5	3	13,33	0,01	5,64	7,09	0,56	0,02
7	3	$\times$	0,13	*sing*	$\times$	$\times$	$\times$
8	2	$\times$	0,02	*sing*	$\times$	$\times$	$\times$
9	3	15,49	0,01	6,02	8,89	0,54	0,02
11	3	$\times$	0,08	*time*	$\times$	$\times$	$\times$
12	3	$\times$	0,04	*sing*	$\times$	$\times$	$\times$
13	3	6,52	0,00	2,43	3,72	0,33	0,01

Fortsetzung auf der nächsten Seite

Fortsetzung

Bsp.	#Var	Algorithmus 5, Macaulay Resultante Summe	Schritt 1	Schritt 2	Schritt 3	Schritt 4	Schritt 5
14	4	×	0,29	*sing*	×	×	×
15	3	×	0,01	*sing*	×	×	×
16	3	×	0,53	*sing*	×	×	×
18	2	×	0,01	*sing*	×	×	×
20	3	×	0,20	*sing*	×	×	×
23	6	×	*mem*	×	×	×	×
24	4	2531,40	0,05	978,50	1548,96	2,93	0,13
26	2	×	0,05	*sing*	×	×	×
27	4	6481,22	0,07	3358,29	3112,50	2,72	0,12
28	10	×	*mem*	×	×	×	×
29	8	×	*mem*	×	×	×	×
30	4	×	4,45	*sing*	×	×	×
31	2	×	0,13	*sing*	×	×	×
33	7	×	*mem*	×	×	×	×
40	3	×	0,75	*sing*	×	×	×
41	3	0,99	0,00	0,23	0,67	0,09	0,00
42	10	×	*mem*	×	×	×	×
43	4	×	4,36	*sing*	×	×	×
45	4	×	0,01	*sing*	×	×	×
46	4	×	1,22	*sing*	×	×	×
51	3	×	0,49	*sing*	×	×	×
54	6	×	*mem*	×	×	×	×
55	7	×	*mem*	×	×	×	×
57	8	×	*mem*	×	×	×	×
58	4	×	4,33	*sing*	×	×	×
59	4	6480,33	0,07	3357,87	3112,01	2,74	0,12
60	3	×	0,20	*sing*	×	×	×
61	3	×	0,20	*sing*	×	×	×
62	3	×	0,20	*sing*	×	×	×
Durchschnitt		1554,08	0,02	771,30	780,07	1,05	0,04

Tabelle 4.2: Zeiten für Algorithmus 5 (Macaulay Resultante)

Wir erinnern uns, daß die Macaulay Resultante als Quotinent zweier Determinanten $\det(\mathcal{DM}_0)/\det(\mathcal{DM}'_0)$ dargestellt wurde. Beim Betrachten der Tabelle 4.2 fällt auf, daß die Submatrix $\mathcal{DM}'_0$, die in Schritt 1 aus der Matrix $\mathcal{DM}_0$ hergeleitet wird, sehr oft singulär, eine weitere Berechnung also nicht möglich ist.

Die Dimension von $\mathcal{DM}_0$ war $\binom{n+d}{d}$ nach Satz 12, wird also abhängig vom Grad des Eingabesystems sehr schnell sehr groß. Oftmals war es dager gar nicht möglich, die Matrix $\mathcal{DM}_0$ zu konstruieren, da nicht genügend Speicherplatz vorhanden war.

Betrachtet man die Beispiele, für die eine Berechnung möglich war, so stellt man fest, daß es sich hierbei durchweg um dicht besetzte Polynome handelt. Die Macaulay Resultante ist also vornehmlich für dichte polynomiale Systeme geeignet.

Nun die Zeiten bei der Verwendung der sparsen Resultante:

		Algorithmus 5, sparse Resultante					
Bsp.	#Var	Summe	Schritt 1	Schritt 2	Schritt 3	Schritt 4	Schritt 5
1	2	0,28	0,01	0,00	0,20	0,07	0,00
2	3	15,36	0,09	1,21	8,27	5,63	0,15
3	2	0,27	0,00	0,00	0,19	0,08	0,00
4	3	1,14	0,03	0,01	0,60	0,48	0,02
5	3	1,49	0,03	0,12	0,78	0,55	0,01
7	3	39,95	0,10	5,78	21,12	11,91	1,02
8	2	3,93	0,02	0,02	0,35	3,33	0,19
9	3	1,23	0,03	0,06	0,61	0,52	0,01
11	3	10,60	0,07	0,75	3,17	6,36	0,24
12	3	29,60	0,09	9,15	15,79	4,31	0,25
13	3	0,81	0,02	0,02	0,43	0,33	0,01
14	4	45,63	0,31	17,41	21,71	5,81	0,38
15	3	1,26	0,04	0,08	0,60	0,52	0,02
16	3	4807,28	0,25	513,10	4231,29	54,23	8,38
18	2	0,31	0,01	0,00	0,16	0,13	0,01
20	3	4,09	0,06	0,62	1,57	1,69	0,14
23	6	×	30,10	*mem*	×	×	×
24	4	5,19	0,08	0,31	1,73	2,93	0,13
26	2	41,56	0,03	2,36	17,39	20,44	$1,32_m$
27	4	20,21	0,25	6,82	10,31	2,70	0,12
28	10	×	301,20	*time*	×	×	×
29	8	×	88,65	*time*	×	×	×
30	4	281,48	0,86	28,76	143,03	106,18	$2,61_m$
31	2	3,07	0,01	0,02	0,23	2,70	0,11
33	7	×	58,55	*mem*	×	×	×
40	3	15,74	0,10	1,56	8,24	5,68	0,15
41	3	0,69	0,01	0,00	0,64	0,02	0,01
42	10	×	161,26	*time*	×	×	×
43	4	195,31	0,84	27,85	102,98	61,00	$2,61_m$
45	4	1,86	0,10	0,13	1,22	0,40	0,01
46	4	5847,86	0,76	559,33	5067,49	211,37	$8,87_m$
51	3	×	0,15	54,87	417,97	8,90	itr
54	6	×	28,15	*mem*	×	×	×
55	7	×	2864,80	*mem*	2,70	×	×
57	8	×	153,72	*time*	×	×	×
58	4	×	1,95	6133,82	×	×	×
59	4	18,75	0,25	7,41	8,24	2,72	0,12

Fortsetzung auf der nächsten Seite

Fortsetzung

Bsp.	#Var	Algorithmus 5, sparse Resultante Summe	Schritt 1	Schritt 2	Schritt 3	Schritt 4	Schritt 5
60	3	12,22	0,07	0,69	4,22	6,78	0,45
61	3	17,17	0,08	3,59	7,49	5,62	0,38
62	3	55,62	0,10	12,93	34,84	7,25	0,49
Durchschnitt		382,67	0,16	40,00	323,83	17,72	0,94

Tabelle 4.3: Zeiten für Algorithmus 5 (sparse Resultante)

Betrachten wir die Beispiele, bei denen es wegen Speichermangel oder Überschreitung der maximalen Rechenzeit zu einem Abbruch kam, so stellen wir fest, daß die Resultantenmatrix hierbei durchweg sehr groß ist. Auch wenn diese Matrizen aufgrund ihrer sparsen Struktur noch in den Speicher passen, so nimmt die Berechnung ihrer Determinanten nach der Spezialisierung in Schritt 2 und 3 aus Algorithmus 5 übermäßig viel Zeit in anspruch. Diese großen Matrizen treten bereits bei relativ kleinen Systemen mit 8 oder 10 Variablen und mittlerem Grad auf.

Auffällig ist auch, daß die für Schritt 3 benötigte Zeit im Durchschnitt dreimal so lang ist, wie die Zeit für Schritt 2. Die Berechnung der Koordinaten der Nullstellen (Schritt 2 und Schritt 4) beansprucht also weniger Zeit, als die Zuordnung der Nullstellen (Schritt 3 und Schritt 5). Offensichtlich liegt hier eine Schwachstelle des Algorithmus 5.

Vergleichen wir beide Tabellen miteinander, so fällt auf, daß die Zeiten bei der Verwendung der sparsen Resultante zum Teil erheblich unter den Zeiten bei der Verwendung der Macaulay Resultante liegen, sofern es überhaupt möglich war, die Macaulay Resultante zu konstruieren. Das Ergebnis fällt also eindeutig zugunsten der sparsen Resultante aus.

4.3 Vergleich Resultanten – FGLM-Algorithmus

Nun wollen wir Algorithmus 5 mit dem FGLM-Algorithmus vergleichen. Der FGLM-Algorithmus berechnet zunächst eine Basis für das Eingabesystem bezüglich einer „Ausweichordnung“, für die sich die Berechnung als einfach erweist, und transformiert diese dann in eine Gröbnerbasis bezüglich der gewünschten Ordnung.

Dies geschiet mit dem SINGULAR-Befehl

```
ideal = stdfglm( ideal, "dp" );
```

wobei als „Ausweichordnung“ die grad-reverse-lexikographische Ordnung und als gewünschte Zielordnung die lexikographische Ordnung verwendet wird. Die genaue Definition dieser Ordnungen findet man zum Beispiel in [Wic97] oder [CLO97a].

Unter Verwendung des Laguerre-Algorithmus aus Abschnitt 3.2.4 und rekursivem Einsetzen der so gewonnenen Koordinaten der Nullstellen in die Gröbnerbasis erhält man alle komplexen Nullstellen.

Durch das wiederholte Einsetzen und die Verwendung von Gleitkommazahlen mit einer vorgegebenen Genauigkeit können Fehler auftreten, die um so größer ausfallen können, je mehr Variablen das Eingabesystem aufweist. Diese Rundungsfehler können bei der Verwendung des Algorithmus 5 nicht auftreten.

Leider mußte hier aus technischen Gründen[1] eine Einschränkung bezüglich der Struktur der Gröbnerbasis gemacht werden: die Basis muß über genausoviele Elemente verfügen, die der zugrundeliegende Polynomring Variablen hat. Der Eintrag *err* statt einer Zeitangabe kennzeichnet Gröbnerbasen, bei denen dies nicht der Fall war. Ist bei einem der Schritte des Algorithmus ein Fehler aufgetreten, so wird dies durch einen der folgenden Einträge dargestellt:

In der dritten Spalte von Tabelle 4.4 sind noch einmal die Zeiten aufgeführt, die Algorithmus 5 zum Lösen der Beispiele benötigt. In Spalte 4 ist die Zeit aufgeführt, die mit dem eben beschriebenen Verfahren unter Verwendung des FGLM-Algorithmus zum Lösen der Beispiele benötigt wird. Spalte 5 enthält die Zeiten, die der Befehlt `stdfglm` zur Ausführung benötigt und Spalte 6 die Zeiten, die für das suksessive Lösen und Einsetzten benötigt werden.

Die letzte Zeile enthält wieder die Durchschnittszeiten, die einen direkten Vergleich ermöglichen.

Auch hier wurde mit einer Rechengenauigkeit von 30 Stellen gerechnet und eine Zeitbeschränkung von 5 Stunden vorgegeben. Ein Überschreiten dieser Zeitbeschränkung wird durch den Eintrag *time* gekennzeichnet.

		Alg 5, sparse	FGLM-Algorithmus		
Bsp.	#Var	Summe	Summe	stdfglm	solve
1	2	0,28	0,10	0,01	0,09
2	3	15,36	2,24	0,49	1,75
3	2	0,27	0,10	0,01	0,09
4	3	1,14	0,39	0,02	0,37
5	3	1,49	0,46	0,06	0,40
7	3	39,95	20,34	12,57	7,77
8	2	3,93	3,49	0,45	3,04
9	3	1,23	0,43	0,03	0,40
11	3	10,60	0,98	0,13	0,85
12	3	29,60	4,87	2,26	2,61
13	3	0,81	0,29	0,02	0,27
14	4	45,63	5,03	1,71	3,32
15	3	1,26	0,43	0,05	0,38
16	3	4807,28	1035,05	990,90	44,15
18	2	0,31	0,15	0,02	0,13
20	3	4,09	×	*time*	×
23	6	×	×	102,90	err
24	4	5,19	1,97	0,24	1,73
26	2	41,56	86,49	73,71	12,78
27	4	20,21	4,29	2,72	1,57

Fortsetzung auf der nächsten Seite

[1]Der Befehl `interred` funktioniert bei der Verwendung von komplexen Zahlen in SINGULAR noch nicht.

Fortsetzung

Bsp.	#Var	Alg 5, sparse Summe	FGLM-Algorithmus Summe	stdfglm	solve
28	10	×	529,58	524,95	4,63
29	8	×	501,78	496,19	5,59
30	4	281,48	×	0,50	err
31	2	3,07	×	0,49	err
33	7	×	×	6,65	err
40	3	15,74	2,25	0,49	1,76
41	3	0,69	0,15	0,02	0,13
42	10	×	41,35	32,58	8,77
43	4	195,31	×	0,54	err
45	4	1,86	0,51	0,06	0,45
46	4	5847,86	×	6,60	err
51	3	×	23,55	7,65	15,89
54	6	×	×	49,87	err
55	7	×	×	*time*	×
57	8	×	1637,43	1615,11	22,31
58	4	×	5,52	3,42	2,10
59	4	18,75	4,28	2,74	1,54
60	3	12,22	21,27	11,07	10,20
61	3	17,17	×	3,27	err
62	3	55,62	×	13,70	err
Durchschnitt		382,67	135,68	130,33	5,35

Tabelle 4.4: Zeiten sparse Resultante — FGLM - Algorithmus

4.4 Vergleich Resultanten – CAS

Um ein Maß für die Qualität der Implementation der Resultanten-Methoden in SINGULAR zu erhalten und außerdem die korrekte Funktionsweise zu überprüfen, sollte diese mit anderen Computeralgebrasystemen (CAS) verglichen werden. Hierzu wurden die folgenden Systeme hinzugezogen:

- **Mathematica 2.2**
  Kommando:

  ```
  NSolve[{<Liste der Gleichungen>},{<Liste der Variablen>},30]
  ```

- **Maple V Release 3**
  Kommando:

  ```
  evalf(solve({<Liste der Gleichungen>},{<Liste der Variablen>}),30);
  ```

- **REDUCE 3.5**
  Kommando:

```
solve(<Liste der Gleichungen>,<Liste der Variablen>);
```

mit den Optionen

```
on time; on complex; on rounded; on multiplicities; precision 30;
```

Leider war aus der Dokumentation der CAS nicht immer ersichtlich, mit welchem Lösungsverfahren gearbeitet wird. Bei dem Vergleich muß man unter anderem auch beachten, daß die Kommandos der CAS sehr viel mehr können, als nur nulldimensionale polynomiale Gleichungssysteme zu lösen und daß die Lösungsverfahren aufgrund des Umfangs der CAS möglicherweise nicht so ausgereift und spezialisiert sind, wie die Resultanten-Methode, die auf nulldimensionale Systeme zugeschnitten ist.

Gerechnet wurde auf einer Hewlet Packard HP-9000/735 (CPU PA-RISC 7100, 99 MHz) mit 424 MByte Hauptspeicher unter HPUX 9.01a, da nur auf diesem Rechner sowohl Mathematica als auch Maple zur Verfügung standen. Es sei angemerkt, daß eine HP C160 mit dem zur Kompilierung von SINGULAR verwendeten *egcs* etwa um den Faktor 1,3 schneller ist, als eine HP-9000/735.

Bsp.	SINGULAR	Mathematica	Maple	REDUCE
1	0,28	0,90	0,70	0,59
3	0,27	0,90	0,87	1,16
4	1,14	9,27	2,25	231,65
5	1,49	127,33	143,92	210,96
7	39,95	*time*	*time*	*time*
11	10,60	*time*	2,33$_{(3)}$	4,63
12	29,60	*time*	$>$ 644,12	×
14	45,63	*time*	$>$ 1203,39	×
16	4807,28	×	$>$ 1700,34	×
24	5,19	7,62	16,70$_{(1)}$	×
26	41,56	23,40	2,68$_{(1)}$	×
27	20,21	*time*	$>$ 1413,26	×
31	3,07	13,90	22,28$_{(1)}$	×
40	5,61	14,00	×	×
41	0,69	1,00	×	×
43	195,31	×	7,08$_{(16)}$	×
45	1,86	2,58	1,56$_{(3)}$	×
60	12,22	*time*	×	×
61	17,17	*time*	71,07$_{(3)}$	×
62	55,62	*time*	4183,22$_{(2)}$	×

Tabelle 4.5: Vergleich verschiedener CAS mit SINGULAR

Tabelle 4.5 zeigt die Ergebnisse des Vergleichs zwischen SINGULAR und den oben genannten Computeralgebrasystemen. Ist eine Zeitangabe mit einem Index versehen, so gibt dieser die Anzahl der gefundenen Nullstellen an. Die tatsächliche Anzahl kann man der zweiten Spalte der Tabelle 4.1 entnehmen.

Gerechnet wurde auch hier mit einer Genauigkeit von 30 Stellen. Vorgegeben wurde eine maximale Rechenzeit von 2 Stunden (7200 Sekunden). Ein Überschreiten dieser Zeitgrenze wird durch den Eintrag *time* makiert. Die Zeiten sind auch hier wieder in Sekunden angegeben.

Die Tabelle 4.5 zeigt, daß SINGULAR-Implementation den anderen CAS deutlich überlegen ist. Bei den kleineren Beispielen liefern auch die CAS noch gute Ausführungszeiten, jedoch ergibt sich bei den größeren Beispielen eine eindeutige Verschiebung zugunsten von SINGULAR. Auch die Berücksichtigung des Faktors 1,3 als Geschwindigkeitsunterschied zwischen den zugrundeliegenden Rechnerplatformen bringt hier keine großen Veränderungen, da die Laufzeitunterschiede teilweise beträchtlich sind. Bei der Verwendung von Maple fällt auf, daß dies zwar recht schnell ist, aber nicht immer alle komplexen Nullstellen findet.

4.5 Zusammenfassung

Aus all diesen Ergebnissen kann man schließen, daß die Verwendung von Resultanten wie in Algorithmus 5 zum Lösen relativ kleiner Gleichungssysteme kleinen bis mittleren Grades eine durchaus brauchbare Methode darstellt. Insbesondere im Vergleich mit den populären Computeralgebrasystemen zeigt sich die Überlegenheit der Resultanten-Methode.

Eine offensichtiche Schäche des Algorithmus liegt in der Zuordnung der Koordinaten der Nullstellen. Die Koordinaten selbst lassen sich in akzeptabler Zeit bestimmen und die Zeit, die in Schritt 5 für die Zuordnung selbst benötigt wird, ist vernachlässigbar. Jedoch nimmt die Vorarbeit, die in Schritt 3 geleistet werden muß, um die Zuordnung zu ermöglichen, mehr Zeit in Anspruch, als alle anderen Schritte. Dies ist unter anderem darauf zurückzuführen, daß das Einsetzen der Zufallszahlen $k_1, k_2, \dots$ in Schritt 3 die Berechnung der Determinanten aufwendiger macht.

Generell kann die Handhabung der schnell sehr großen Resultantenmatrizen Probleme bereiten. Der Speicherplatzbedarf läßt sich durch die sparse Darstellung der Matrizen teilweise in Griff bekommen. Die Berechnung der Determinante stellt jedoch auch ausgereifte, die Sparseheit ausnutzende Algorithmen auf eine harte Probe. Deshalb wäre es von Vorteil, Verfahren zu entwickeln, die die Größe der Resultantenmatrix weiter reduzieren und andere Methoden zur Extraktion der Nullstellen aus der Matrix wie in Algorithmus 5 zu entwickeln und zu testen.

Bei den Tests zeigte es sich, daß der Algorithmus an den folgenden Stellen Probleme bereiten kann. Zum einen können große Gleichungssysteme oder ein hoher Grad zu großen Resultantenmatrizen und dem daraus resultierenden Speicherplatzmangel führen. Dies ist bei der Macaulay Resultantenmatrix häufiger der Fall, als bei der sparsen Resultantenmatrix.

Wählt man δ bei der Konstruktion der sparsen Resultantenmatrix aus Schritt 1.4 bzw. Definition 26 falsch, so kann es vorkommen, daß man Punkte $p \in \mathcal{E}$ hat, die die Zeilenzuordnungsfunktion keiner Zelle der Subdivision zuordnen kann. Die Konstruktion der Matrix muß dann mit einem anderen δ erneut versucht werden. Der gleiche Fall kann eintreten, wenn die Linearkombinationen aus Schritt 1.2 bzw. (2.32) nicht generisch gewählt sind.

Eine schlechte Konditionierung des Gleichungssystems durch unzureichend generische Koeffizienten kann dazu führen, daß die Resultantenmatrix singulär ist, bei der Spezialisierung und Berechnung der Determinanten (Schritte 3 und 4) also nur Nullen herauskommen. Dies führt dazu, daß man als multiple Nullstelle die Null erhält. Bei ungünstiger Lage der Nullstellen kann es zudem vorkommen, das der Algorithums von Laguerre in Schritt 4 nicht innerhalb der vorgegebenen Anzahl von Iterationsschritten terminiert.

Bei Schritt 5, der Zuordnung der Nullstellen, können schließlicht numerische Probleme zutage treten, da man hier mit Gleitkommazahlen arbeitet und diese miteinander vergleicht. So kann es zum Beispiel vorkommen, daß eine genaue Zuordnung einzelner Koordinaten einer Nullstelle nicht möglich ist, da die Rechengenauigkeit zu klein gewählt ist.

Wie man sieht, erweist sich die hier getestet und implementierte Kombination aus algebraischer und numerischer Vorgehensweise als durchaus brauchbar, wobei man allerdings die vielen Fallstricke berücksichtigen muß, die sich bei der Konstruktion der Matrizen und der Verwendung der numerischen Algorithmen ergeben.

4.6 Die Beispiele

- **Beispiel 1:** (aus [CLO97b]), $\mathbb{Q}[x_1,x_2]$

$$I = \langle x_1^2+x_2^2-10, x_1^2+x_1x_2+2x_2^2-16\rangle$$

- **Beispiel 2:** (aus [CLO97b]), $\mathbb{Q}[a,b,c]$

$$I = \langle a^2bc+ab^2c+abc^2+abc+ab+ac+bc, a^2b^2c+a^2bc+ab^2c^2+abc+a+bc+c,\\ a^2b^2c^2+a^2b^2c+ab^2c+abc+ac+c+1\rangle$$

- **Beispiel 3:** (aus [EMW98]), $\mathbb{Q}[x,y]$

$$I = \langle x^2+6x+3y-4, 2x+y^2-7y+5\rangle$$

- **Beispiel 4:** (aus [EMW98]), $\mathbb{Q}[x,y,z]$

$$I = \langle x^2+6x+3y+6z-4, 2x+y^2-7y+2z+5, x^2+y^2+z^2-1\rangle$$

- **Beispiel 5:** (aus [CM93]), $\mathbb{Q}[x_1,x_2,x_3]$

$$I = \langle 6x_1^2-4x_1x_2-8x_1x_3+3x_2^2+10x_2x_3+13x_3^2-1,\\ -2x_1^2-4x_1x_3+x_1-2x_2^2-4x_2x_3+x_2-4x_3^2,\\ -5x_1^2+2x_1x_2+8x_1x_3+x_1-2x_2^2-10x_2x_3-x_2\\ -13x_3^2\rangle$$

- **Beispiel 7:** (aus [CM93]), $\mathbb{Q}[x,y,z]$

$$I = \langle 2x^2+7xz+4x+3y^2+9yz+5z^2, 7x^3+8xy+18xz+3y^4z+12y^2+10z^3+12,\\ 3x^4+xyz+3xz^2+6y^3+2yz^2+4z^2+5\rangle$$

- **Beispiel 8:** (aus [Stu98]), $\mathbb{Q}[x,y]$

$$I = \langle x^4+y^4-1, x^5y^2-4x^3y^3+x^2y^5-1\rangle$$

- **Beispiel 9:** (aus [Stu98]), $\mathbb{Q}[x,y,z]$

$$I = \langle 2x^2+4xy+7xz+3y^2+9yz+5z^2+2, 7x^2+8xy+18xz+12y^2+3yz+10z^2+12,\\ 3x^2+xy+3xz+6y^2+2yz+4z^2+5\rangle$$

- **Beispiel 11:** (aus [MKF93]), $\mathbb{Q}[x_1,x_2,x_3]$

$$\begin{aligned} I = \langle & \frac{1}{4}x_1^4 - x_1^3x_2 + x_1^3 + \frac{3}{2}x_1^2x_2^2 + 3x_1^2x_2 - x_1x_2^3 \\ & + 3x_1x_2^2 + \frac{1}{4}x_2^4 + x_2^3 + x_3^4 - 1, \\ & \frac{3}{2}x_1^2 + x_1x_2 + \frac{3}{2}x_2^2 + x_3^3 - 1, \\ & \frac{1}{5}x_1^2 + \frac{2}{5}x_1x_2 + \frac{1}{5}x_2^2 + \frac{1}{2}x_3^3 - 1\rangle \end{aligned}$$

- **Beispiel 12:** (aus [MKF93]), $\mathbb{Q}[x,y,z]$

$$\begin{aligned} I = \langle & 3x^4 + 2x^2y^2 + 3x^2y + 4x^2z^2 + xyz + xz^2 + 6y^2z + 5z^4, \\ & x^3 + x^2y + 3xyz + 5xy + 3x + 2y^3 + 6y^2z + yz + 1, 7xy + x + 3yz + 4y + 2z + 10\rangle \end{aligned}$$

- **Beispiel 13:** (aus [MKF93]), $\mathbb{Q}[x_1,x_2,x_3]$

$$\begin{aligned} I = \langle & 16x_1^2 + 3x_2^2 + 5x_3^3 - 1, 5x_1^2 + 2x_1 + 3x_2^2 + 4x_3^2 - 1, \\ & -4x_1 + x_2 + x_3 - 3\rangle \end{aligned}$$

- **Beispiel 14:** (aus [MKF93]), $\mathbb{Q}[x_1,x_2,x_3,x_4]$

$$\begin{aligned} I = \langle & 16x_1^2 + 3x_2^2 + 5x_3^4 + x_4^3 - 4x_4 - 1, \\ & 5x_1^3 + 2x_1x_4 + 4x_1 + 3x_2^2 + x_2 + 4x_3^2x_4 + x_3 + 2x_4 \\ & -1, \\ & -4x_1^2 + 4x_1 + x_2^2 + x_2 + x_3^2 + x_3 + x_4^2 + x_4 - 3, -4x_1 + x_2 + x_3 + x_4\rangle \end{aligned}$$

- **Beispiel 15:** (aus [MKF93]), $\mathbb{Q}[x,y,z]$

$$\begin{aligned} I = \langle & 2x^2z + 3xy + 7xz + 4y^2z + 25y^2 + 4yz^2 + 2y - 3z, x^2y + 15x^2 + 3xyz + xy + xz^2 + 9yz + 7, \\ & 4x + 5y + 6\rangle \end{aligned}$$

- **Beispiel 16:** (aus [MKF93]), $\mathbb{Q}[x,y,z]$

$$\begin{aligned} I = \langle & x^4 + 6x^3y + 2x^3z + 3x^3 + 12x^2y^2 + 12x^2yz + 57xy^5 + 36xyz + 7xz^3 + 29y^2 + 12yz^3 \\ & + 25yz^2 + 2yz - 3y + 3z^4 + 7z^3 - 3z, \\ & x^4 + 16x^3 + 6x^2y^3 + 5x^2z^3 + 10xy^2 + 11y^2z + 25y^2 + 7yz^2 + 40yz + z^3 + 15z^2 + 7, \\ & 9x^3 + 10y^2 + 5z^3 + 6\rangle \end{aligned}$$

- **Beispiel 18:** (aus [MKF93]), $\mathbb{Q}[x,y]$

$$I = \langle 7x^2y^2 + x^2y + x + y, xy^2 + x + y + 1\rangle$$

- **Beispiel 20:** (aus [MKF93]), $\mathbb{Q}[t_1,t_2,t_3]$

$$\begin{aligned} I = \langle & 4t_2^2t_3^2 + 2t_2^2 + 5t_2t_3 + 3t_3^2 + 1, \\ & 4t_1^2t_3^2 + 3t_1^2 + 5t_1t_3 + 2t_3^2 + 1, \\ & 4t_1^2t_2^2 + 2t_1^2 + 5t_1t_2 + 3t_2^2 + 1\rangle \end{aligned}$$

- **Beispiel 23:** (aus [MKF93]), $\mathbb{Q}[x_1,x_2,x_3,x_4,x_5,x_6]$

$$\begin{aligned}I = \langle & x_1+x_2+x_3+x_4+x_5+x_6, x_1x_2+x_1x_6+x_2x_3+x_3x_4+x_4x_5+x_5x_6,\\ & x_1x_2x_3+x_1x_2x_6+x_1x_3x_4+x_1x_5x_6\\ & +x_2x_3x_4+x_4x_5x_6,\\ & x_1x_2x_3x_4+x_1x_2x_3x_6+x_1x_2x_5x_6\\ & +x_1x_4x_5x_6+x_2x_3x_4x_5+x_3x_4x_5x_6,\\ & x_1x_2x_3x_4x_5+x_1x_2x_3x_4x_6\\ & +x_1x_2x_3x_5x_6+x_1x_2x_4x_5x_6\\ & +x_1x_3x_4x_5x_6+x_2x_3x_4x_5x_6,\\ & x_1x_2x_3x_4x_5x_6-1\rangle\end{aligned}$$

- **Beispiel 24:** (aus [EMW98]), $\mathbb{Q}[x,y,z,w]$

$$I = \langle 5x^2+6x+3y+6z+2w+3, 4x+4y^2+3z+2w+4, x-11z^2+3z+7w-9, x+3y+2z-3w^2+13\rangle$$

- **Beispiel 26:** (aus [Wic97]), $\mathbb{Q}[x,y]$

$$\begin{aligned}I = \langle & 45x^8+3x^7+39x^6+30x^5+13x^4+41x^3+5x^2+46x+7,\\ & 49x^7y+4x^6+31x^4+27x^3y^2+33x^2y+24x^2+35xy^7+37xy^6+20xy^4+9y^7+49y+43\rangle\end{aligned}$$

- **Beispiel 27:** (aus [Wic97]), $\mathbb{Q}[w,x,y,z]$

$$\begin{aligned}I = \langle & 8w^2+5wx-4wy+2wz+3w+5x^2+2xy-7xz-7x+7y^2-8yz-7y+7z^2-8z+8,\\ & 3w^2-5wx-3wy-6wz+9w+4x^2+2xy-2xz+7x+9y^2+6yz+5y+7z^2+7z+5,\\ & -2w^2+9wx+9wy-7wz-4w+8x^2+9xy-3xz+8x+6y^2-7yz+4y-6z^2+8z+2,\\ & 7w^2+5wx+3wy-5wz-5w+2x^2+9xy-7xz+4x-4y^2-5yz+6y-4z^2-9z+2\rangle\end{aligned}$$

- **Beispiel 28:** (aus [Wic97]), $\mathbb{Q}[a,b,c,d,u,v,w,x,y,z]$

$$\begin{aligned}I = \langle & 16aw+18bv+20cu, -80d+180y+855z, 7av+8bu, 210z-210,\\ & 40ay+44bx+48cw+52dv+280u, 27ax+20bw+33cv+36du,\\ & 55az+60by+65cx+70dw+80u+375v, -170a+78bz+84cy+90dx+102v+480w,\\ & -114c+136dz+152x+720y, -144b+105cz+112dy+126w+595x\rangle\end{aligned}$$

- **Beispiel 29:** (aus [Wic97]), $\mathbb{Q}[a,b,u,v,w,x,y,z]$

$$\begin{aligned}I = \langle & -162a^2u+6abz+8auy+10avx+48aw+14bx+16uw,\\ & -162a^2v-312ab+15auz+18avy+24aw+84ax+24by+27ux+30vw,\\ & -240a-64v+112y+420z,\\ & -162a^2-284av+50ay+180az-112b+55uz+60vy+70w+260x,\\ & 66az-1056a-90u+78vz+90x+336y, 136z-136,\\ & -162a^2b+2aby+3aux+4avw+6bw,\\ & -648a^2+300au+28avz+128ay+36bx+36bz+40uy+44vx+192w\rangle\end{aligned}$$

- **Beispiel 30:** (aus [Wic97]), $\mathbb{Q}[x,y,z,t]$

$$\begin{aligned} I = \langle & 2xyt-2x+y^2z-z, xt^2-x+2yzt-2z, \\ & -x^3z+4x^2yt+4x^2+4xy^2z+4xz+2y^3t-10y^2-10yt+2, \\ & -xz^3+4xzt^2+4xz+4yz^2t+2yt^3-10yt+4z^2-10t^2+2\rangle \end{aligned}$$

- **Beispiel 31:** (aus [Wic97]), $\mathbb{Q}[x,y]$

$$I = \langle 5x^4+3x^2y^9+y^9, 9x^3y^8+9xy^8+11y^{10}\rangle$$

- **Beispiel 33:** (aus [Wic97]), $\mathbb{Q}[a,b,c,d,e,f,g]$

$$\begin{aligned} I = \langle & a^2+a+2bg+2cf+2de, 2ab+b+2cg+2df+e^2, 2ac+b^2+c+2dg+2ef, \\ & 2ab+2ad+2bc+b+2cg+2df+d+e^2+2eg, 2ae+2bd+c^2+e+2fg, \\ & 2af+2be+2cd+f+g^2, 2ag+2bf+2ce+d^2+g\rangle \end{aligned}$$

- **Beispiel 40:** (aus [Aut95]), $\mathbb{Q}[x,y,z]$

$$\begin{aligned} I = \langle & x^2yz+xy^2z+xyz^2+xyz+xy+xz+yz, x^2y^2z+x^2yz+xy^2z^2+xyz+x+yz+z, \\ & x^2y^2z^2+x^2y^2z+xy^2z+xyz+xz+z+1\rangle \end{aligned}$$

- **Beispiel 41:** (aus [Aut95]), $\mathbb{Q}[y,z,x]$

$$I = \langle 2yz+2yx-y, 2y^2+2z^2+x^2-x, 2y+2z+x-1\rangle$$

- **Beispiel 42:** (aus [Aut95]), $\mathbb{Q}[x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9,x_{10}]$

$$\begin{aligned} I = \langle & -10x_1x_8-320000x_1+64x_2x_7+10x_7x_9 \\ & +11x_7x_{10}, \\ & 160000x_1-32x_2x_7-5x_2x_8-5x_2x_{10}-5000x_2, \\ & -x_3x_8+x_6x_8+210x_6+x_9x_{10}+1300000, -x_4x_8+700000, -2x_5+x_{10}^2, \\ & -x_6x_8-210x_6+x_7x_9, \\ & 320000x_1-64x_2x_7-10x_7x_9-11x_7x_{10}-16x_7 \\ & +7000000, \\ & -10x_1x_8-10x_2x_8+10x_2x_{10}+20000x_2-10x_3x_8 \\ & -10x_4x_8+14x_5-10x_6x_8+11x_7x_{10}, \\ & x_4x_8-x_7x_9-x_9x_{10}-410x_9, \\ & 10x_2x_8-10x_2x_{10}+10x_3x_8+10x_6x_8+1400x_6 \\ & +10x_7x_9-11x_7x_{10}-10x_9x_{10}-10x_{10}^2-4200x_{10}\rangle \end{aligned}$$

- **Beispiel 43:** (aus [Aut95]), $\mathbb{Q}[y,z,x,t]$

$$\begin{aligned} I = \langle & y^2z+2yxt-z-2x, 2y^3t+4y^2zx-10y^2+4yx^2t-10yt-zx^3+4zx+4x^2+2, \\ & 2yzt-2z+xt^2-x, 4yz^2t+2yt^3-10yt-z^3x+4z^2+4zxt^2+4zx-10t^2+2\rangle \end{aligned}$$

- **Beispiel 45:** (aus [Aut95]), $\mathbb{Q}[t,z,y,x]$

$$I = \langle t^2-t+2z^2+2y^2+2x^2, 2tz+2zy-z+2yx, 2ty+z^2+2zx-y, t+2z+2y+2x-1\rangle$$

- **Beispiel 46:** (aus [Aut95]), $\mathbb{Q}[t,x,y,z]$

$$I = \langle t^2x+xy^2+xz^2+x-1, t^2y+x^2y+yz^2+y-1, t^2z+x^2z+y^2z+z-1,\\ tx^2+ty^2+tz^2+t-1\rangle$$

- **Beispiel 51:** (aus [Wic97]), $\mathbb{Q}[z,y,x]$

$$I = \langle z^2+zy^2x+y^4+y^2-2yx+x^2, z^3yx+zy^2x+y^4-y^2x^3-2yx, z^4y+y^4x-2yx^2-3\rangle$$

- **Beispiel 54:** (aus [Wic97]), $\mathbb{Q}[a,b,c,d,x,f]$

$$I = \langle a+b+c+d+x+f, ab+af+bc+cd+dx+xf, abc+abf+axf+bcd+cdx+dxf,\\ abcd+abcf+abxf+adxf+bcdx+cdxf,\\ abcdx+abcdf+abcxf+abdxf+acdxf+bcdxf, abcdxf-1\rangle$$

- **Beispiel 55:** (aus [Wic97]), $\mathbb{Q}[a,b,c,d,e,f,g]$

$$I = \langle a+b+c+d+e+f+g, ab+ag+cd+de+ef+fg, abc+abg+afg+bcd+cde+def+efg,\\ abcd+abcg+abfg+aefg+bcde+cdef+defg,\\ abcde+abcdg+abcfg+abefg+adefg+bcdef+cdefg,\\ abcdef+abcdeg+abcdfg+abcefg+abdefg+acdefg+bcdefg, abcdefg-1\rangle$$

- **Beispiel 57:** (aus [Wic97]), $\mathbb{Q}[a,b,x,y,z,u,v,w]$

$$I = \langle -162a^2u+6abz+10axv+8ayu+48aw+14bx+16uw,\\ -162a^2v-312ab+84ax+18ayv+15azu+24aw+24by+27xu+30vw,\\ -240a+112y+420z-64v,\\ -162a^2+50ay+180az-284av-112b+260x+60yv+55zu+70w,\\ 66az-1056a+90x+336y+78zv-90u, 136z-136,\\ -162a^2b+2aby+3axu+4avw+6bw,\\ -648a^2+128ay+28azv-300au+36bx+36bz+44xv+40yu+192w\rangle$$

- **Beispiel 58:** (aus [Wic97]), $\mathbb{Q}[x,y,z,t]$

$$I = \langle 2x^2-2y^2+2z^2-2t^2-1, 2x^3-2y^3+2z^3-2t^3-1, 2x^4-2y^4+2z^4-2t^4-1,\\ 2x^5-2y^5+2z^5-2t^5-1\rangle$$

- **Beispiel 59:** (aus [Wic97]), $\mathbb{Q}[w,x,y,z]$

$$I = \langle 8w^2+5wx-4wy+2wz+3w+5x^2+2xy-7xz-7x+7y^2-8yz-7y+7z^2-8z+8,\\ 3w^2-5wx-3wy-6wz+9w+4x^2+2xy-2xz+7x+9y^2+6yz+5y+7z^2+7z+5,\\ -2w^2+9wx+9wy-7wz-4w+8x^2+9xy-3xz+8x+6y^2-7yz+4y-6z^2+8z+2,\\ 7w^2+5wx+3wy-5wz-5w+2x^2+9xy-7xz+4x-4y^2-5yz+6y-4z^2-9z+2\rangle$$

- **Beispiel 60:** (aus [Wic97]), $\mathbb{Q}[x,y,z]$

$$I = \langle xy^3-2xz^3+y^4+yz^2-z^3, x^3y+2x^2y+2xy^2z, 2x^3y-3x^2y+yz^3+2\rangle$$

- **Beispiel 61:** (aus [Wic97]), $\mathbb{Q}[x,y,z]$

$$I = \langle 3xy^3 + x + y^4 + yz^2, -x^2z + 3xyz^2 + 2y^3z + 2yz^2 + z^2, 3x^3 + xy^2 - 2xz^3 + yz^2 \rangle$$

- **Beispiel 62:** (aus [Wic97]), $\mathbb{Q}[x,y,z]$

$$I = \langle x^3z + 6x^2 - 2xz^3 + y^4 + yz, 5x^2y^2 + y^3z + 3yz^3 + z^3, -x^2z - 2xyz^2 + 4y^4 + 2y^2z \rangle$$

5 Zur Implementierung

Ein Ziel der vorliegenden Arbeit bestand darin, den in Kapitel 3 vorgestellten Algorithmus 5 für das Computeralgebrasystem SINGULAR zu implementieren. Aus diesem Grund erfolgte die Implementierung als Kern-Funktionen in C/C++. Die einzelnen Schritte des Algorithmus verteilen sich über die folgenden Dateien:

- `mpr_global.h`
  In dieser Datei befinden sich einige globale Einstellungen und Hilfsmittel zum Debuggen.
- `mpr_base.h, mpr_base.cc`
  Hier ist die Klasse `resMatrixBase` implementiert, von der sich die Klassen `resMatrixSparse` und `resMatrixDense` ableiten. Diese implementieren jeweils den Algorithmus zur Konstruktion der sparsen und der Macaulay Resultantenmatrix. Außerdem findet sich hier die Klasse `uResultant`, die die Schritte von Algorithmus 5 ausführt und die benötigten Unter-Algorithmen aufruft.
- `mpr_numeric.h, mpr_numeric.cc`
  Hier sind der Algorithmus von Laguerre (Abschnitt 3.2.4) in der Klasse `rootContainer` und das „matchen“ der Nullstellen nach dem in Abschnitt 3.2.2 beschriebenen Verfahren in der Klasse `rootArranger` implementiert. Die Klasse `vandermonde` löst Vandermonde-Systeme nach dem in Algorithmus 1 beschriebenen Verfahren. Als reine C-Funktion findet man hier außerdem eine Implementation des Simplex-Algorithmus zum Lösen linearer Optimierungsaufgaben.
- `mpr_inout.h, mpr_inout.cc`
  Hier sind die Funktionen enthalten, die vom SINGULAR-Interpreter aufgerufen werden. Sie führen Vorarbeiten und Konsistenz-Prüfungen durch, um schließlich die entsprechenden Algorithmen aufzurufen.
- `mpr_complex.h, mpr_complex.cc`
  Die Klasse `gmp_float` stellt die Funktionalität der GNU Multiple Precision Arithmetic Library zur Verfügung. Basierend darauf implementiert die Klasse `gmp_complex` die komplexen Zahlen, bestehend aus zwei `gmp_float`'s.
- `gnumpfl.h, gnumpfl.cc, gnumpc.h, gnumpc.cc`
  Die Anbindung der Klassen `gmp_float` und `gmp_complex` an SINGULAR. Damit ergeben sich die Körper der reellen und der komplexen Zahlen beliebig definierbarer Genauigkeit.

Da der Quelltext in den einzelnen Dateien etwa 7100 Zeilen in C/C++ umfaßt, werden hier nur die wichtigsten Header-Dateien und Klassen wiedergegeben.

Zuletzt wird noch der Quelltext der SINGULAR-Bibliotek `solve.lib` abgebildet.

mpr_base.h

```
//-> class resMatrixBase
/**
 * Base class for sparse and dense u-Resultant computation
 */
class resMatrixBase
{
public:
  /* state of the resultant */
  enum IStateType { none, ready, notInit, fatalError, sparseError };

  resMatrixBase() : istate(notInit), totDeg(0) {}
  virtual ~resMatrixBase() {}

  virtual const ideal getMatrix()
    { return NULL; }
  virtual const ideal getSubMatrix()
    { return NULL; }

  virtual const poly getUDet( const number* evpoint )
    { return NULL; }

  virtual const number getDetAt( const number* evpoint )
    { return NULL; }
  virtual const number getSubDet()
    { return NULL; }

  virtual const long getDetDeg()
    const { return totDeg; }

  virtual const IStateType initState()
    const { return istate; }

protected:
  IStateType istate;

  ideal gls;
  int linPolyS;
  ring sourceRing;

  int totDeg;

private:
  /* disables the copy constructor */
  resMatrixBase( const resMatrixBase & );
};
//<-

//-> class uResultant
/**
 * Base class for solving 0-dim poly systems using u-resultant
 */
class uResultant
{
public:
  enum resMatType { none, sparseResMat, denseResMat };

  uResultant( const ideal _gls,
              const resMatType _rmt= sparseResMat,
              BOOLEAN extIdeal= true );
  ~uResultant();

  poly interpolateDense( const number subDetVal= NULL );

  /* Interpolates n+1 determinat polys for coeff specializations. */
  rootContainer ** interpolateDenseSP( BOOLEAN matchUp= false,
                                       const number subDetVal= NULL );

  /* Uses Bareiss */
  rootContainer ** specializeInU( BOOLEAN matchUp= false,
                                  const number subDetVal= NULL );

  resMatrixBase * accessResMat() { return resMat; }

private:
  /* deactivated copy constructor */
```

```
  uResultant( const uResultant & );

  ideal extendIdeal( const ideal gls,
                     poly linPoly,
                     const resMatType rmt );
  poly linearPoly( const resMatType rmt );
  int nextPrime( const int p );

  ideal gls;
  int n;

  resMatType rmt;        // sparse or dense resultant matrix ?
  resMatrixBase *resMat; // pointer to base resultant matrix class
};
//<-
```

Aus mpr_base.cc

```
//-------------- sparse resultant matrix -----------------------------

//-> definitions

//#define mprTEST
//#define mprMINKSUM

#define MAXPOINTS      1000
#define MAXINITELEMS   256
#define LIFT_COOR      100000000
#define SCALEDOWN      10000.0
#define MAXSEED        1024 //512
#define MINVDIST       0.0
#define RVMULT         0.00005 // 0.000000005
#define MAXVARS        100
//<-

//-> sparse resultant matrix

/* set of points,
 * each point represents an exponent vector of an monomial */
class pointSet;

/* Linear Program stuff */
struct linProg
{
  mprfloat **LiPM;
  int *izrov, *iposv;
  int LiPM_cols,LiPM_rows;
};

typedef linProg *linProgP;

/* sparse resultant matrix class */
class resMatrixSparse : virtual public resMatrixBase
{
public:
  resMatrixSparse( const ideal _gls, const int special = SNONE );
  ~resMatrixSparse();

  // public interface according to base class resMatrixBase
  const ideal getMatrix();

  /** Fills in resMat[][] with evpoint[] and gets determinant
   * uRPos[i][1]: row of matrix
   * uRPos[i][idelem+1]: col of u(0)
   *  uRPos[i][2..idelem]: col of u(1) .. u(n)
   *  i= 1 .. numSet0
   */
  const number getDetAt( const number* evpoint );

  const poly getUDet( const number* evpoint );
```

```
private:
  resMatrixSparse( const resMatrixSparse & );

  void randomVector( const int dim, mprfloat shift[] );

  /** Row Content Function
   * Finds the largest i such that F[i] is a point,
   * F[i]= a[ij] in A[i] for some j.
   * Returns -1 iff the point vert does not lie in a cell
   */
  int RC( pointSet **pQ, pointSet *E, int vert, mprfloat shift[] );

  /* Remaps a result of LP to the according point set Qi.
   * Returns false iff remaping was not possible, otherwise true.
   */
  bool remapXiToPoint( const int indx,
                       pointSet **pQ,
                       int *set, int *vtx );

  /** create coeff matrix
   * uRPos[i][1]: row of matrix
   * uRPos[i][idelem+1]: col of u(0)
   *  uRPos[i][2..idelem]: col of u(1) .. u(n)
   *  i= 1 .. numSet0
   * Returns the dimension of the matrix or -1 in case of an error
   */
  int createMatrix( pointSet *E );

  pointSet * minkSumAll( pointSet **pQ, int numq, int dim );
  pointSet * minkSumTwo( pointSet *Q1, pointSet *Q2, int dim );

private:
  ideal gls;

  int n, idelem;     // number of variables, polynoms
  int numSet0;       // number of elements in S0
  int msize;         // size of matrix

  intvec *uRPos;

  ideal rmat;        // sparse matrix representation

  linProg LP;
};
//<-

//-> typedefs and structs
poly monomAt( poly p, int i );

typedef unsigned int Coord_t;

struct setID
{
  int set;
  int pnt;
};

struct onePoint
{
  Coord_t * point; // point[0] is unused, max dim is MAXVARS+1
  setID rc;                 // filled in by Row Content Function
  struct onePoint * rcPnt; // filled in by Row Content Function
};

typedef struct onePoint * onePointP;
//<-

//-> class pointSet
class pointSet
{
private:
  onePointP *points;     // set of onePoint's, index [1..num]
  bool lifted;

public:
  int num;     // number of elements in points
  int max;     // maximal entries in points
  int dim;     // dimension, i.e. valid coord entries in point
  int index;   // should hold unique identifier of point set
```

```
pointSet( const int _dim,
          const int _index= 0,
          const int count= MAXINITELEMS );
 ~pointSet();

// pointSet.points[i] equals pointSet[i]
inline const onePointP operator[] ( const int index );

/** Adds a point to pointSet, copy vert[0,...,dim] to
 * point[num+1][0,...,dim]. Returns false, iff additional
 * memory was allocated ( i.e. num >= max )
 * else returns true
 */
bool addPoint( const onePointP vert );

bool addPoint( const Exponent_t * vert );

bool addPoint( const Coord_t * vert );

/* Removes the point at intex indx */
bool removePoint( const int indx );

/** Adds point to pointSet, iff pointSet \cap point = \emptyset.
 * Returns true, iff added, else false.
 */
bool mergeWithExp( const onePointP vert );

bool mergeWithExp( const Exponent_t * vert );

/* Adds support of poly p to pointSet, iff
 * pointSet \cap point = \emptyset. */
void mergeWithPoly( const poly p );

/* Returns the row polynom multiplicator in vert[] */
void getRowMP( const int indx, Exponent_t * vert );

/* Returns index of supp(LT(p)) in pointSet. */
int getExpPos( const poly p );

/** sort lex
   */
  void sort();

  /** Lifts the point set using sufficiently generic linear lifting
   * homogeneous forms l[1]..l[dim] in Z. Every l[i] is of the form
   * L1x1+...+Lnxn, for generic L1..Ln in Z.
   *
   * Lifting raises dimension by one!
   */
  void lift( int *l= NULL );     // !! increments dim by 1
  void unlift() { dim--; lifted= false; }

private:
  pointSet( const pointSet & );

  /** points[a] < points[b] ? */
  inline bool smaller( int, int );

  /** points[a] > points[b] ? */
  inline bool larger( int, int );

  /** Checks, if more mem is needed ( i.e. num >= max ),
   * returns false, if more mem was allocated, else true
   */
  inline bool checkMem();

  int seed;
};
//<-

//-> class convexHull
/* Compute convex hull of given exponent set */
class convexHull
{
public:
  convexHull( linProgP _pLP ) : pLP(_pLP) {}
  ~convexHull() {}

  /** Computes the point sets of the convex hulls of
   * the supports given by the polynoms in gls.
```

```
   * Returns Q[].
   */
   pointSet ** newtonPolytopes( ideal gls );

private:
  /** Returns true iff the support of poly pointPoly is inside the
   * convex hull of all points given by the  support of poly p.
   */
  bool inHull(poly p, poly pointPoly, int m, int site);

private:
  pointSet **Q;
  int n;
  linProgP pLP;
};
//<-

//-> class mayanPyramidAlg
/* Compute all lattice points in a given convex hull */
class mayanPyramidAlg
{
public:
  mayanPyramidAlg( linProgP _pLP ) : n(pVariables), pLP(_pLP) {}
  ~mayanPyramidAlg() {}

  /** Drive Mayan Pyramid Algorithm.
   * The Alg computes conv(Qi[]+shift[]).
   */
  pointSet * getInnerPoints( pointSet **_Qi, mprfloat _shift[] );

private:

  /** Recursive Mayan Pyramid algorithm for directly computing
   * MinkowskiSum lattice points for (n+1)-fold MinkowskiSum
   * of given point sets Qi[]. Recursively for range of dim:
   * dim in [0..n); acoords[0..var) fixed.
   * Stores only MinkowskiSum points of udist > 0: done
   * by storeMinkowskiSumPoints.
   */
  void runMayanPyramid( int dim );

  /**  Compute v-distance via Linear Programing
   * Linear Program finds the v-distance of the point in accords[].
   * The v-distance is the distance along the direction v to
   * boundary of Minkowski Sum of Qi (here vector v is
   * represented by shift[]).
   * Returns the v-distance or -1.0 if an error occured.
   */
  mprfloat vDistance( Coord_t * acoords, int dim );

  /** LP for finding min/max coord in MinkowskiSum,
   * given previous coors.
   * Assume MinkowskiSum in non-negative quadrants
   * coor in [0,n); fixed coords in acoords[0..coor)
   */
  void mn_mx_MinkowskiSum( int dim, Coord_t *minR, Coord_t *maxR );

  /**  Stores point in E->points[pt], iff v-distance != 0
   * Returns true iff point was stored, else flase
   */
  bool storeMinkowskiSumPoint();

private:
  pointSet **Qi;
  pointSet *E;
  mprfloat *shift;

  int n,idelem;

  Coord_t acoords[MAXVARS+2];

  linProgP pLP;
};
//<-

//------------- dense resultant matrix -----------------------------

//-> dense resultant matrix
//
class resVector;
```

```
/* dense resultant matrix */
class resMatrixDense : virtual public resMatrixBase
{
public:
  /**
   * _gls: system of multivariate polynoms
   * special: -1 -> resMatrixDense is a symbolic matrix
   *    0,1, ... -> resMatrixDense ist eine u-Resultante,
   *  wobei special das
   *                        lineare u-Polynom angibt
   */
  resMatrixDense( const ideal _gls, const int special = SNONE );
  ~resMatrixDense();

  /** column vector of matrix, index von 0 ... numVectors-1 */
  resVector *getMVector( const int i );

  /** Returns the matrix M in an usable presentation */
  const ideal getMatrix();

  /** Returns the submatrix M' of M in an usable presentation */
  const ideal getSubMatrix();

  /** Evaluate the determinant of the matrix M at the point evpoint
   * where the ui's are replaced by the components of evpoint.
   * Uses singclap_det from factory.
   */
  const number getDetAt( const number* evpoint );

  /** Evaluates the determinant of the submatrix M'.
   * Since the matrix is numerically, no evaluation point is needed.
   * Uses singclap_det from factory.
   */
  const number getSubDet();

private:
  /** deactivated copy constructor */
  resMatrixDense( const resMatrixDense & );

  /** Generate the "matrix" M. Each column is presented by a resVector
   * holding all entries for this column.
   */
  void generateBaseData();

  /** Generates needed set of monoms, split them into sets S0, ... Sn
   * and check if reduced/nonreduced and calculate size of submatrix.
   */
  void generateMonomData( int deg, intvec* polyDegs , intvec* iVO );

  /** Recursively generate all homogeneous monoms of
   * pVariables variables of degree deg.
   */
  void generateMonoms( poly m, int var, int deg );

  /** Creates quadratic matrix M of size numVectors for later use.
   * u0, u1, ...,un are replaced by 0.
   * Entries equal to 0 are not initialized ( == NULL)
   */
  void createMatrix();

private:
  resVector *resVectorList;

  int veclistmax;
  int veclistblock;
  int numVectors;
  int subSize;

  matrix m;
};
//<-

//-> struct resVector
/* Holds a row vector of the dense resultant matrix */
struct resVector
{
public:
  void init()
  {
```

```
    isReduced = FALSE;
    elementOfS = SFREE;
    mon = NULL;
  }
  void init( const poly m )
  {
    isReduced = FALSE;
    elementOfS = SFREE;
    mon = m;
  }

  /** index von 0 ... numVectors-1 */
  poly getElem( const int i );

  /** index von 0 ... numVectors-1 */
  number getElemNum( const int i );

  // variables
  poly mon;
  poly dividedBy;
  bool isReduced;

  /** number of the set S mon is element of */
  int elementOfS;

  /** holds the index of u0, u1, ..., un, if (elementOfS == linPolyS)
   *  the size is given by pVariables
   */
  int *numColParNr;

  /** holds the column vector if (elementOfS != linPolyS) */
  number *numColVector;

  /** size of numColVector */
  int numColVectorSize;

  number *numColVecCopy;
};
//<-
```

mpr_inout.h

```
/** solve a multipolynomial system using the u-resultant
 * Input ideal must be 0-dimensional and pVariables == IDELEMS(ideal).
 * Resultant method can be MPR_DENSE, which uses Macaulay Resultant (good
 * for dense homogeneous polynoms) or MPR_SPARSE, which uses
 * Sparse Resultant (Gelfand, Kapranov, Zelevinsky).
 * Arguments 4: ideal i, int k, int l, int m
 *   k=0: use sparse resultant matrix of Gelfand, Kapranov and Zelevinsky
 *   k=1: use resultant matrix of Macaulay (k=0 is default)
 *   l>0: defines precision of fractional part if ground field is Q
 *   m=0,1,2: number of iterations for approximation of roots (default=2)
 * Returns a list containing the roots of the system.
 */
BOOLEAN nuUResSolve( leftv res, leftv arg1, leftv arg2, leftv arg3 );

/** returns module representing the multipolynomial resultant matrix
 * Arguments 2: ideal i, int k
 *   k=0: use sparse resultant matrix of Gelfand, Kapranov and Zelevinsky
 *   k=1: use resultant matrix of Macaulay (k=0 is default)
 */
BOOLEAN nuMPResMat( leftv res, leftv arg1, leftv arg2 );

/** find the (complex) roots an univariate polynomial
 * Determines the roots of an univariate polynomial using Laguerres'
 * root-solver. Good for polynomials with low and middle degree (<40).
 * Arguments 3: poly arg1 , int arg2 , int arg3
 *  arg2>0: defines precision of fractional part if ground field is Q
 *  arg3: number of iterations for approximation of roots (default=2)
 * Returns a list of all (complex) roots of the polynomial arg1
 */
BOOLEAN nuLagSolve( leftv res, leftv arg1, leftv arg2 );
```

```
/** compute polynomial f with values given by v at points p1,..,pN
 * derived from p;
 * more precisely: consider p as point in K^n and v as N elements
 * in K, let p1,..,pN be the points in K^n obtained by evaluating
 * all monomials of degree 0,1,..,N at p in lexicographical order,
 * then the procedure computes the polynomial f satisfying
 *
 *                        f(pi) = v[i].
 *
 * arg1: ideal with point p
 * arg2: ideal with points w
 * arg3: degree of poly f
 */
BOOLEAN nuVanderSys( leftv res, leftv arg1, leftv arg2, leftv arg3 );

/** compute Newton Polytopes of input polynomials
 */
BOOLEAN loNewtonP( leftv res, leftv arg1 );

/** Implementation of the Simplex Algorithm.
 * For args, see class simplex.
 */
BOOLEAN loSimplex( leftv res, leftv args );
```

mpr_numeric.h

```
//-> vandermonde system solver
/**
 * vandermonde system solver for interpolating polynomials
 * from their values
 */
class vandermonde
{
public:
  vandermonde( const long _cn,
               const long _n,
               const long _maxdeg,
               number *_p,
               const bool _homog = true );

  ~vandermonde();

  /** Solves the Vandermode linear system
   *    \sum_{i=1}^{n} x_i^k-1 w_i = q_k, k=1,..,n.
   * Any computations are done using type number to get high
   *  pecision results.
   * q n-tuple of results (right hand of equations)
   * returns: w n-tuple of coefficients of resulting polynomial,
   * lowest deg first
   */
  number * interpolateDense( const number * q );

  poly numvec2poly(const number * q );

private:
  void init();

private:
  long n;       // number of variables
  long cn;      // real number of coefficients of poly to interpolate
  long maxdeg;  // degree of the polynomial to interpolate
  long l;       // max number of coefficients in poly of deg maxdeg
                //   = (maxdeg+1)^n

  number *p;    // evaluation point
  number *x;    // coefficients, determinend by init() from *p

  bool homog;
};
//<-

//-> rootContainer
```

```
/**
 * complex root finder for univariate polynomials based on
 * laguerre's algorithm
 */
class rootContainer
{
public:
  enum rootType { none, cspecial, cspecialmu, det, onepoly };

  rootContainer();
  ~rootContainer();

  void fillContainer( number *_coeffs, number *_ievpoint,
                      const int _var, const int _tdg,
                      const rootType _rt, const int _anz );

  bool solver( const int polishmode= PM_NONE );

  poly getPoly();

  //gmp_complex & operator[] ( const int i );
  inline gmp_complex & operator[] ( const int i ) {
    return *theroots[i];
  }
  gmp_complex & evPointCoord( const int i );

  inline gmp_complex * getRoot( const int i ) {
    return theroots[i];
  }

  bool swapRoots( const int from, const int to );

  int getAnzElems() { return anz; }
  int getLDim() { return anz; }
  int getAnzRoots() { return tdg; }

private:
  rootContainer( const rootContainer & v );

  /** Given the degree tdg and the tdg+1 complex coefficients
   * ad[0..tdg] (generated from the number coefficients
   * coeffs[0..tdg]) of the polynomial this routine successively
   * calls laguer() and finds all m complex roots in roots[0..tdg].
   * The bool var <polish> should be input as true if polishing
   * (also by laguer()) is desired, false if the roots will be
   * subsequently polished by other means.
   */
  bool laguer_driver( gmp_complex ** a,
                      gmp_complex ** roots,
                      bool polish = true );

  /** Given the degree m and the m+1 complex coefficients
   * a[0..m] of the polynomial, and given the complex value x,
   * this routine improves x by Laguerre's method until it converges,
   * within the achievable roundoff limit, to a root of the given
   * polynomial. The number of iterations taken is
   * returned at its.
   */
  void laguer(gmp_complex ** a, int m, gmp_complex * x, int * its);

  int var;
  int tdg;

  number * coeffs;
  number * ievpoint;
  rootType rt;

  gmp_complex ** theroots;

  int anz;
  bool found_roots;
};
//<-

//-> class rootArranger
class rootArranger
{
public:
  rootArranger( rootContainer ** _roots,
                rootContainer ** _mu,
```

```
                const int _howclean = PM_CORRUPT );
  ~rootArranger() {}

  void solve_all();
  void arrange();

  lists listOfRoots( const unsigned int oprec );

  const bool success() { return found_roots; }

private:
  rootArranger( const rootArranger & );

  rootContainer ** roots;
  rootContainer ** mu;

  int howclean;
  int rc,mc;
  bool found_roots;
};
//<-

//-> simplex computation
//   (used by sparse matrix construction)
#define SIMPLEX_EPS 1.0e-12

/** Linear Programming / Linear Optimization using Simplex - Algorithm
 *
 * On output, the tableau LiPM is indexed by two arrays of integers.
 * ipsov[j] contains, for j=1..m, the number i whose original variable
 * is now represented by row j+1 of LiPM.(left-handed vars in solution)
 * (first row is the one with the objective function)
 * izrov[j] contains, for j=1..n, the number i whose original variable
 * x_i is now a right-handed variable, rep. by column j+1 of LiPM.
 * These vars are all zero in the solution. The meaning of n<i<n+m1+m2
 * is the same as above.
 */
class simplex
{
public:
  int m;          // number of constraints, make sure m == m1 + m2 + m3!
  int n;          // # of independent variables
  int m1,m2,m3;   // constraints <=, >= and ==
  int icase;      // == 0: finite solution found;
                  // == +1 objective funtion unbound; == -1: no solution
  int *izrov,*iposv;

  mprfloat **LiPM; // the matrix (of size [m+2, n+1])

  /** #rows should be >= m+2, #cols >= n+1
   */
  simplex( int rows, int cols );
  ~simplex();

  BOOLEAN mapFromMatrix( matrix m );
  matrix mapToMatrix( matrix m );
  intvec * posvToIV();
  intvec * zrovToIV();

  void compute();

private:
  simplex( const simplex & );

  void simp1( mprfloat **a,
      int mm, int ll[], int nll, int iabf, int *kp, mprfloat *bmax );
  void simp2( mprfloat **a,
      int n, int l2[], int nl2, int *ip, int kp, mprfloat *q1 );
  void simp3( mprfloat **a,
      int i1, int k1, int ip, int kp );

  int LiPM_cols,LiPM_rows;
};
//<-
```

mpr_complex.h

```
/** Set size of mantissa to <bytes> bytes and guess the number of
 * output digits.
 *
 * Set internal gmp floating point precision to (bytes+EXTRABYTES)*8
 *  bits and set external precision (i.e. any number smaller
 * than this is treated as ZERO) to bytes*8 bits.
 * The precision in bytes is the size of the mantissa!
 * Guesses the number of digits <-> precision
 */
void setGMPFloatPrecBytes( unsigned long int bytes );
unsigned long int getGMPFloatPrecBytes();

// Sets the lenght of the mantissa to <digits> digits
void setGMPFloatDigits( size_t digits );
size_t getGMPFloatDigits();

//-> class gmp_float
/**
 * @short wrapper class for GNU Multi Precision Floats
 */
class gmp_float
{
public:
  gmp_float( const int v = 0 )
  {
    mpf_init2( t, gmp_default_prec_bits );
    mpf_set_si( t, (signed long int) v );
  }
  gmp_float( const long v )
  {
    mpf_init2( t, gmp_default_prec_bits );
    mpf_set_si( t, (signed long int) v );
  }
  gmp_float( const mprfloat v ) // double
  {
  mpf_init2( t, gmp_default_prec_bits );
  mpf_set_d( t, (double) v );
}
gmp_float( const mpf_t v )
{
  mpf_init2( t, gmp_default_prec_bits );
  mpf_set( t, v );
}
gmp_float( const mpz_t v ) // gnu mp Z
{
  mpf_init2( t, gmp_default_prec_bits );
  mpf_set_z( t, v );
}
gmp_float( const gmp_float & v ) // copy constructor
{
  //mpf_init2( t, mpf_get_prec( v.t ) );
  mpf_init2( t, gmp_default_prec_bits );
  mpf_set( t, v.t );
}

~gmp_float()
{
  mpf_clear( t );
}

friend gmp_float operator + ( const gmp_float & a,
                              const gmp_float & b );
friend gmp_float operator - ( const gmp_float & a,
                              const gmp_float & b );
friend gmp_float operator * ( const gmp_float & a,
                              const gmp_float & b );
friend gmp_float operator / ( const gmp_float & a,
                              const gmp_float & b );

inline gmp_float & operator += ( const gmp_float & a );
inline gmp_float & operator -= ( const gmp_float & a );
inline gmp_float & operator *= ( const gmp_float & a );
inline gmp_float & operator /= ( const gmp_float & a );

friend bool operator == ( const gmp_float & a,
```

```
                           const gmp_float & b );
  friend bool operator  > ( const gmp_float & a,
                           const gmp_float & b );
  friend bool operator  < ( const gmp_float & a,
                           const gmp_float & b );
  friend bool operator >= ( const gmp_float & a,
                           const gmp_float & b );
  friend bool operator <= ( const gmp_float & a,
                           const gmp_float & b );

  friend gmp_float operator - ( const gmp_float & a );

  gmp_float & operator = ( const gmp_float & a );
  gmp_float & operator = ( const mpz_t & a );
  gmp_float & operator = ( const mprfloat a );
  gmp_float & operator = ( const long a );

  inline int sign();    // t>0:+1, t==0:0, t<0:-1
  inline bool isZero();  // t == 0 ?
  inline bool isOne();   // t == 1 ?
  inline bool isMOne();  // t == -1 ?

  inline bool setFromStr( char * in );

  // access
  inline const mpf_t *mpfp() const;

  inline operator double();
  inline operator double() const;

  inline operator int();
  inline operator int() const;

public:
  static void setPrecision( const unsigned long int prec ) {
    gmp_default_prec_bits= prec;
  }
  static void setEqualBits( const unsigned long int prec ) {
    gmp_needequal_bits= prec;
  }
  static const unsigned long int getPrecision() {
    return gmp_default_prec_bits;
  }
  static const unsigned long int getEqualBits() {
    return gmp_needequal_bits;
  }

private:
  static unsigned long int gmp_default_prec_bits;
  static unsigned long int gmp_needequal_bits;

  mpf_t t;
};

// built-in functions of GMP
gmp_float abs( const gmp_float & );
gmp_float sqrt( const gmp_float & );
gmp_float hypot( const gmp_float &, const gmp_float & );
//gmp_float pow( const gmp_float &, int & );

// simulated functions using double functions
gmp_float sin( const gmp_float & );
gmp_float cos( const gmp_float & );
gmp_float log( const gmp_float & );
gmp_float exp( const gmp_float & );

gmp_float max( const gmp_float &, const gmp_float & );

gmp_float numberToFloat( number num );
char *floatToStr( const gmp_float & r, const unsigned int oprec );
//<-

//-> class gmp_complex
/**
 * @short gmp_complex numbers based on
 */
class gmp_complex
{
private:
```

```
  gmp_float r, i;

public:
  gmp_complex( const gmp_float re= 0.0, const gmp_float im= 0.0 )
  {
    r= re;
    i= im;
  }
  gmp_complex( const mprfloat re, const mprfloat im = 0.0 )
  {
    r= re;
    i= im;
  }
  gmp_complex( const long re, const long im )
  {
    r= re;
    i= im;
  }
  gmp_complex( const gmp_complex & v )
  {
    r= v.r;
    i= v.i;
  }
  ~gmp_complex() {}

  friend gmp_complex operator + ( const gmp_complex & a,
                                  const gmp_complex & b );
  friend gmp_complex operator - ( const gmp_complex & a,
                                  const gmp_complex & b );
  friend gmp_complex operator * ( const gmp_complex & a,
                                  const gmp_complex & b );
  friend gmp_complex operator / ( const gmp_complex & a,
                                  const gmp_complex & b );

  // gmp_complex <operator> real
  inline friend gmp_complex operator + ( const gmp_complex & a,
                                         const gmp_float b_d );
  inline friend gmp_complex operator - ( const gmp_complex & a,
                                         const gmp_float b_d );
  inline friend gmp_complex operator * ( const gmp_complex & a,
                                         const gmp_float b_d );
  inline friend gmp_complex operator / ( const gmp_complex & a,
                                         const gmp_float b_d );

  gmp_complex & operator += ( const gmp_complex & a );
  gmp_complex & operator -= ( const gmp_complex & a );
  gmp_complex & operator *= ( const gmp_complex & a );
  gmp_complex & operator /= ( const gmp_complex & a );

  inline friend bool operator == ( const gmp_complex & a,
                                   const gmp_complex & b );

  inline gmp_complex & operator = ( const gmp_complex & a );

  // access to real and imaginary part
  inline gmp_float real() const { return r; }
  inline gmp_float imag() const { return i; }

  inline void real( gmp_float val ) { r = val; }
  inline void imag( gmp_float val ) { i = val; }
};

// Returns absolute value of a gmp_complex number
//
inline gmp_float abs( const gmp_complex & c )
{
  return hypot(c.real(),c.imag());
}

gmp_complex sqrt( const gmp_complex & x );

inline gmp_complex numberToGmp_Complex( number num )
{
  if (rField_is_long_C()) {
    return *(gmp_complex*)num;
  } else {
    return gmp_complex( numberToFloat(num) );
  }
}
```

```
char *complexToStr( const gmp_complex & c,
                    const  unsigned int oprec );
//<-
```

solve.lib

```
///////////////////////////////////////////////////////////////////////////////

version="$Id: solve.lib,v 1.12 1999/07/08 10:18:13 wenk Exp $";
info="
LIBRARY: solve.lib     PROCEDURES TO SOLVE POLYNOMIAL SYSTEMS
AUTHOR:  Moritz Wenk,  email: wenk@mathematik.uni-kl.de

PROCEDURES:
 ures_solve(i,..);      find all roots of 0-dimensional ideal i with resultants
 mp_res_mat(i,..);      multipolynomial resultant matrix of ideal i
 laguerre_solve(p,..);  find all roots of univariate polynom p
 interpolate(i,j,d);    interpolate poly from evaluation points i and results j
";

///////////////////////////////////////////////////////////////////////////////

proc ures_solve( ideal gls, list # )
"USAGE:   ures_solve(i[,k,l,m]); i ideal, k,l,m integers
         k=0: use sparse resultant matrix of Gelfand, Kapranov and Zelevinsky
         k=1: use resultant matrix of Macaulay (k=0 is default)
         l>0: defines precision of fractional part if ground field is Q
         m=0,1,2: number of iterations for approximation of roots (default=2)
ASSUME:  i is a zerodimensional ideal with
         nvars(basering) = ncols(i) = number of vars actually occuring in i
RETURN:  list of all (complex) roots of the polynomial system i = 0,
         of type number if the ground field is the complex numbers,
         of type string if the ground field is the rational or real numbers
EXAMPLE: example ures_solve; shows an example
"
{
  int typ=0;
  int polish=2;
  int prec=30;

  if ( size(#) >= 1 )
  {
    typ= #[1];
    if ( typ < 0 || typ > 1 )
    {
      ERROR("Valid values for second parameter k are:
      0: use sparse Resultant (default)
      1: use Macaulay Resultant");
    }
  }
  if ( size(#) >= 2 )
  {
    prec= #[2];
    if ( prec == 0 ) { prec = 30; }
    if ( prec < 0 )
    {
      ERROR("Third parameter l must be positive!");
    }
  }
  if ( size(#) >= 3 )
  {
    polish= #[3];
```

```
    if ( polish < 0 || polish > 2 )
    {
      ERROR("Valid values for fourth parameter m are:
      0,1,2: number of iterations for approximation of roots");
    }
  }

  if ( size(#) > 3 )
  {
    ERROR("only three parameters allowed!");
  }

  return(uressolve(gls,typ,prec,polish));

}
example
{
  "EXAMPLE:";echo=2;
  // compute the intersection points of two curves
  ring rsq = 0,(x,y),lp;
  ideal gls=  x2 + y2 - 10, x2 + xy + 2y2 - 16;
  ures_solve(gls);
  // result is a list (x,y)-coordinates as strings

  // now with complex coefficient field, precision is 20 digits
  ring rsc= (real,20,I),(x,y),lp;
  ideal i = (2+3*I)*x2 + (0.35+I*45.0e-2)*y2 - 8, x2 + xy + (42.7)*y2;
  list l= ures_solve(i);
  // result is a list of (x,y)-coordinates of complex numbers
  l;
  // check the result
  subst(subst(i[1],x,l[1][1]),y,l[1][2]);
}
///////////////////////////////////////////////////////////////////////////////

proc laguerre_solve( poly f, list # )
"USAGE:   laguerre_solve( p[,l,m]); f poly, l,m integers
         l>0: defines precision of fractional part if ground field is Q
         m=0,1,2: number of iterations for approximation of roots (default=2)
ASSUME:  p is an univariate polynom
RETURN:  list of all (complex) roots of the polynomial p;
         of type number if the ground field is the complex numbers,
         of type string if the ground field is the rational or real numbers
EXAMPLE: example laguerre_solve; shows an example
"
{
  int polish=2;
  int prec=30;

  if ( size(#) >= 1 )
  {
    prec= #[1];
    if ( prec == 0 ) { prec = 30; }
    if ( prec < 0 )
    {
      ERROR("Fisrt parameter must be positive!");
    }
```

```
  }
  if ( size(#) >= 2 )
  {
    polish= #[2];
    if ( polish < 0 || polish > 2 )
    {
      ERROR("Valid values for third parameter are:
      0,1,2: number of iterations for approximation of roots");
    }
  }
  if ( size(#) > 2 )
  {
    ERROR("only two parameters allowed!");
  }

  return(laguerre(f,prec,polish));

}
example
{
  "EXAMPLE:";echo=2;
  // Find all roots of an univariate polynomial using Laguerre's method:
  ring rs1= 0,(x,y),lp;
  poly f = 15x5 + x3 + x2 - 10;
  laguerre_solve(f);

  // Now with 10 digits precision:
  laguerre_solve(f,10);

  // Now with complex coefficients, precision is 20 digits:
  ring rsc= (real,20,I),x,lp;
  poly f = (15.4+I*5)*x^5 + (25.0e-2+I*2)*x^3 + x2 - 10*I;
  list l = laguerre_solve(f);
  l;
  // check result, value of substituted poly should be near to zero
  subst(f,x,l[1]);
  subst(f,x,l[2]);
}
///////////////////////////////////////////////////////////////////////////////

proc mp_res_mat( ideal i, list # )
"USAGE:   mp_res_mat(i[,k]); i ideal, k integer
         k=0: sparse resultant matrix of Gelfand, Kapranov and Zelevinsky
         k=1: resultant matrix of Macaulay (k=0 is default)
ASSUME:  nvars(basering) = ncols(i)-1 = number of vars actually occuring in i,
         for k=1 i must be homogeneous
RETURN:  module representing the multipolynomial resultant matrix
EXAMPLE: example mp_res_mat; shows an example
"
{
  int typ=2;

  if ( size(#) == 1 )
  {
    typ= #[1];
    if ( typ < 0 || typ > 1 )
    {
```

```
      ERROR("Valid values for third parameter are:
      0: sparse resultant (default)
      1: Macaulay resultant");
    }
  }
  if ( size(#) > 1 )
  {
    ERROR("only two parameters allowed!");
  }

  return(mpresmat(i,typ));

}
example
{
  "EXAMPLE:";echo=2;
  // compute resultant matrix in ring with parameters (sparse resultant matrix)
  ring rsq= (0,u0,u1,u2),(x1,x2),lp;
  ideal i= u0+u1*x1+u2*x2,x1^2 + x2^2 - 10,x1^2 + x1*x2 + 2*x2^2 - 16;
  module m = mp_res_mat(i);
  print(m);
  // computing sparse resultant
  det(m);

  // compute resultant matrix (Macaulay resultant matrix)
  ring rdq= (0,u0,u1,u2),(x0,x1,x2),lp;
  ideal h=  homog(imap(rsq,i),x0);
  h;

  module m = mp_res_mat(h,1);
  print(m);
  // computing Macaulay resultant (should be the same as above!)
  det(m);

  // compute numerical sparse resultant matrix
  setring rsq;
  ideal ir= 15+2*x1+5*x2,x1^2 + x2^2 - 10,x1^2 + x1*x2 + 2*x2^2 - 16;
  module mn = mp_res_mat(ir);
  print(mn);
  // computing sparse resultant
  det(mn);
}
///////////////////////////////////////////////////////////////////////////////

proc interpolate( ideal p, ideal w, int d )
"USAGE:   interpolate(p,v,d); p,v ideals, d int
ASSUME:  ground field K is the rational numbers,
         p and v consist of numbers of the ground filed K, p must have
         n elements, v must have N=(d+1)^n elements where n=nvars(basering)
         and d=deg(f) (f is the unknown polynomial in K[x1,...,xn])
COMPUTE: polynomial f with values given by v at points p1,..,pN derived from p;
         more precisely: consider p as point in K^n and v as N elements in K,
         let p1,..,pN be the points in K^n obtained by evaluating all monomials
         of degree 0,1,...,N at p in lexicographical order,
         then the procedure computes the polynomial f satisfying f(pi) = v[i]
RETURN:  polynomial f of degree d
NOTE:    mainly useful for n=1, with f satisfying f(p^(i-1)) = v[i], i=1..d+1
```

```
EXAMPLE: example interpolate; shows an example
"
{
  return(vandermonde(p,w,d));
}
example
{
  "EXAMPLE:";  echo=2;
  ring r1 = 0,(x),lp;
  // determine f with deg(f) = 4 and
  // v = values of f at points 3^0, 3^1, 3^2, 3^3, 3^4
  ideal v=16,0,11376,1046880,85949136;
  interpolate( 3, v, 4 );

  ring r2 = 0,(x,y),dp;
  // determine f with deg(f) = 3 and
  // v = values of f at 16 points (2,3)^0=(1,1),...,(2,3)^15=(2^15,3^15)
  // valuation point (2,3)
  ideal p = 2,3;
  ideal v= 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16;
  poly ip= interpolate( p,v,3 );
  ip;
  // compute poly at point 2,3, result must be 2
  subst(subst(ip,x,2),y,3);
  // compute poly at point 2^15,3^15, result must be 16
  subst(subst(ip,x,2^15),y,3^15);
}
///////////////////////////////////////////////////////////////////////////////
```

Literaturverzeichnis

[Act90] F. S. Acton. *Numerical Methods That Work*. Mathematical Association of America, Washington, 1990.

[Aut95] Various Authors. PoSSo: Polynomial System Solving, 1993 – 1995. Homepage aviable under http://posso.dm.unipi.it/.

[BOT88] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polinomial interpolation. *ACM Symposium on Theory of Computing*, pages 301–309, 1988.

[Can87] John F. Canny. Generalised characteristic polynomials. *Journal of Symbolic Computation*, 9:241–250, 1987.

[CE93] John F. Canny and Ioannis Z. Emiris. An efficient algorithm for the sparse mixed resultant. *Algebraic Algorithms and Error Correction Codes*, pages 89–104, 1993.

[CE95] John F. Canny and Ioannis Z. Emiris. Efficient incremental algorithms for the sparse resultant and the mixed volume. *Journal of Symbolic Computation*, 20:117–149, 1995.

[CE97a] John F. Canny and Ioannis Z. Emiris. Program for mixed volume computations, 1997. Aviable under ftp://robotics.eecs.Berkeley.edu/pub/MixedVolume.

[CE97b] John F. Canny and Ioannis Z. Emiris. A subdivision-based algorithm for the sparse resultant. *preprint*, 1997.

[CLO97a] David Cox, John Little, and Donald O'Shea. *Ideals, Varieties and Algorithms*. Springer Verlag, Berlin - Heidelberg - New York, second edition, 1997.

[CLO97b] David Cox, John Little, and Donald O'Shea. *Using Algebraic Geometry*. Springer Verlag, Berlin - Heidelberg - New York, 1997.

[CM91] John F. Canny and Dinesh Manocha. Efficient techniques for multipolynomial resultant algorithms. *Porceedings of International Symposium on Symbolic and Algebraic Computation*, pages 86–95, 1991.

[CM92] John F. Canny and Dinesh Manocha. Multipolynomial resultant algorithms and linear algebra. *Porceedings of International Symposium on Symbolic and Algebraic Computation*, pages 158–167, 1992.

[CM93] John F. Canny and Dinesh Manocha. Multipolynomial resultant algorithms. *Journal of Symbolic Computation*, 15:99–122, 1993.

[Ede90] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer Verlag, Berlin, Heidelberg, New York, London, Paris, Tokyo, 1990.

[Emi94] Ioannis Z. Emiris. Program for computations of subdivisions, 1994. Aviable under ftp://robotics.eecs.Berkeley.edu/pub/MixedVolume/subdiv.

[EMW98] Ioaniss Z. Emiris, Dinesh Manocha, and Aaron Wallack. MARS: A Maple/Matlab/C resultant-based solver. *Porceedings of International Symposium on Symbolic and Algebraic Computation*, pages 244–251, 1998. Software aviable under http://www.cs.unc.edu/ geom/MARS/.

[ERMM93] Ioannis Z. Emiris, H. Rosenberg, E. P. Mucke, and Dinesh Manocha. Program for computations of convex hulls, 1993. Aviable under ftp://robotics.eecs.Berkeley.edu/pub/ConvexHull.

[Gib87] William Gibson. *Neuromancer*. Heyne Verlag, 1987.

[GKZ94] I. Gelfand, M. Kapranov, and A. Zelevinsky. *Discriminants, Resultants and Multidimensional Determinants*. Birkhäuser, Boston – Basel – Berlin, 1994.

[GPS+99] G.-M. Greuel, G. Pfister, H. Schoenemann, et al. SINGULAR. *A Computer Algebra System for Commutative Algebra, Algebraic Geometry and Singularity Theory*. Fachbereich Mathematik und Zentrum fuer Computeralgebra, Universitaet Kaiserslautern, 1999. Programm-Version 1.3.3.

[Jou91] J. Jouanolou. Le formalisme du résultant. *Advances in Math.*, 90:117–263, 1991.

[Kho78] A. G. Khovanskii. Newton polyhedra and the genus of complete intersection. *Funktsional'nyi Analiz i Ego Prilozheniya*, 12(1):51–61, Jan. - Mar. 1978.

[Kus76] A. G. Kushnierenko. Newton polytopes an the bèzout theorem. *Funktsional'nyi Analiz i Ego Prilozheniya*, 10(3):51–61, Jul. - Sep. 1976.

[Mac02] F. S. Macaulay. On some formula elimination. *Proceedings of London Mathematical Society*, pages 3–27, May 1902.

[Man93] Dinesh Manocha. Efficient algorithms for multipolynomial resultant. *The Computer Journal*, 36(5):485–496, 1993.

[Man94] Dinesh Manocha. Solving systems of polynomial equations. *IEEE Computer Graphics and Applications*, pages 46–55, March 1994.

[Mig92] Maurice Mignotte. *Mathematics for Computer Algebra*. Springer Verlag, Berlin - Heidelberg - New York, 1992.

[MKF93] Hirokazu Murao, Hidetsune Kobayashi, and Tetsuro Fujise. On factorizing the symbolic u-resultant — application of the *ddet* operator —. *Journal of Symbolic Computation*, 15:123–142, 1993.

[PS93] P. Pedersen and Bernd Sturmfels. Product fomulas for resultants and chow forms. *Mathematische Zeitschrift*, 214:377–396, 1993.

[PTVF92] W. H. Press, S. A. Teukolky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C. The Art of Scientific Computation.* Cambridge University Press, second edition, 1992.

[Sch93] R. Schneider. *Convex Bodies: The Brunn-Minkowski Theroy.* Cambridge University Press, 1993.

[Sch97] Hans Rudolf Schwarz. *Numerische Mathematik.* B. G. Teubner, Stuttgard, 1997.

[Sha77] I. Shafarevich. *Basic Algebraic Geometry.* Springer Verlag, Berlin - Heidelberg - New York, 1977.

[Sta80] R. P. Stanley. Decompositions of rational convex polyhedra. In J. Srivastava, editor, *Combinatorial Mathematics, Optimal Designs and Their Applications*, volume 6 of *Anals of Discrete Mathematics*, pages 333 – 342. North-Holland, Amsterdam, 1980.

[Stu94] Bernd Sturmfels. On the newton polytope of the resultant. *Journal of Algebraic Combinatorics*, 3:207–236, 1994.

[Stu98] Bernd Sturmfels. Introduction to resultants. *Proceedings of Symposia in Applied Mathematics*, 53:25–39, 1998.

[SZ94] B. Sturmfels and A. Zelevinsky. Multigraded resultants of sylvester type. *Journal of Algebra*, 163:115–127, 1994.

[vdW50] B. L. van der Waerden. *Modern Algebra.* F. Ungar Publisching Co., New York, third edition, 1950. The capter on Elemination Theory is included in the first three German editions and the 1950 Englich translation, but all alter editions omit this chapter.

[Wic97] Tim Wichmann. Der FGLM-Algorithmus: verallgemeinert und implementiert in SINGULAR. Master's thesis, Universität Kaiserslautern, Fachbereich Mathematik, November 1997.

[Zie95] G. Ziegler. *Lectures on Polytopes.* Springer Verlag, New York, 1995.

[Zip90] R. E. Zippel. Interpolating polynoms from their values. *Journal of Symbolic Computation*, 9:375–403, 1990.

Kaiserslautern, 27. Juli 1999

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbstständig erstellt und keine anderen als die angegebenen Hilfsmittel verwendet habe.