Universidad de Buenos Aires

Facultad de Ciencias Exactas y Naturales

Algoritmos simbólicos para matrices polinomiales, straight–line programs y teoría de eliminación efectiva

Tesis de Licenciatura en Ciencias de la Computación

Autora: Sandra Cesaratto
Director: Guillermo Matera
Co-Directora: Rosita Wachenchauzer

Buenos Aires, Mayo 2003

Resumen

El objetivo de la presente tesis es el estudio, implementación y aplicación de algoritmos simbólicos eficientes para la resolución de problemas álgebra lineal y eliminación en geometría paramétricos.

Los algoritmos que estudiaremos siguen dos principios fundamentales: en primer lugar, utilizan la representación de polinomios multivariados por medio de straight–line programs, y en segundo lugar, poseen versiones de baja complejidad paralela no escalar. Estas dos características permiten evitar el crecimiento exponencial de los resultados intermedios típico de los algoritmos de cálculo simbólico.

La implementación de los mismos se realiza en un ambiente de programación que provee la funcionalidad básica necesaria para la manipulación de polinomios representados por programas.

Abstract

The purpose of this thesis is the development, implementation and application of efficient symbolic algorithms for linear algebra and geometric elimination problems.

In the design of our algorithms we follow two main principles: first, we use the straight–line program representation of multivariate polynomials, and second, we take care of the nonscalar parallel complexity. This allows us to avoid the intermediate expression swell typical of symbolic computation.

These algorithms are implemented in a programming environment which provides the functionality required in order to manipulate program–represented polynomials.

Contenidos

1 Introducción 4

2 Modelo computacional 8
2.1 Estructuras de Datos para la Representación de Polinomios 9

3 Matrices polinomiales y straight–line programs 12
3.1 Polinomio Característico 14
3.1.1 El Método de Samuelson 15
3.2 Polinomio Minimal 17
3.2.1 El método de Wiedemann 18
3.2.2 El Algoritmo de Berlekamp–Massey 19
3.2.3 Algoritmo para el Cálculo de Polinomio Minimal 20
3.3 Matriz Adjunta y Seudo–adjunta 21
3.4 Rango 22
3.5 Un Caso de Aplicación: las Matrices Vandermonde 24

4 Implementación 26
4.1 Representación de Straight–line Programs mediante Programas 26
4.2 El Lenguaje `PARI` 29
4.2.1 Tipos disponibles 30
4.2.2 Operaciones y Funciones 30
4.2.3 Memoria 30
4.2.4 Recursos 31
4.3 El TAD `SLP` implementado en `PARI` 31

5 Aplicaciones a Sistemas Paramétricos 32
5.1 Dimensión del Espacio de Parámetros de un Sistema de Ecuaciones Polinomiales 33
5.2 Resolución de Sistemas de Ecuaciones Polinomiales 35
5.2.1 Sistemas Paramétricos y Deformaciones 37

6 Algoritmos Implementados y Resultados Comparativos 44
6.1 Polinomio Característico con el Método de Samuelson 44
6.2 Polinomio Minimal con el Método de Wiedemann 46

6.3 Cálculo de la dimensión de una variedad 47

7 Conclusiones **50**

Referencias **52**

Capítulo 1

Introducción

El tema central de esta tesis es el tratamiento de problemas lineales o no lineales *paramétricos*. Más precisamente, el desarrollo de algoritmos eficientes para resolver problemas de álgebra lineal o de eliminación en geometría que involucran matrices o sistemas de ecuaciones polinomiales dependientes de parámetros.

El interés por este tipo de problemas proviene de la amplia gama de aplicaciones ingenieriles o científicas modeladas por éstos. Frecuentemente es necesario simplificar *simbólicamente* grandes sistemas de ecuaciones lineales o no lineales a fin de procesarlos numéricamente (ver [GV98]). Por ejemplo, este tipo de cuestiones aparece en la calibración de manipuladores en robótica [Egn96] o de cámaras en visión automática [Hor91], o en la búsqueda de estados de equilibrio en química [EM99].

Los paquetes de software de cálculo simbólico generalistas (`Maple`, `Mathematica`, `Magma`, etc.) proveen herramientas para la manipulación de polinomios multivariados y la resolución de sistemas de ecuaciones lineales y no lineales. Usualmente, los polinomios se representan en forma rala (o densa) por su vector de coeficientes. Desafortunadamente, en los paquetes de software destinados a la resolución de sistemas de ecuaciones lineales y no lineales paramétricos los resultados intermedios son típicamente expresiones polinomiales densas de alto grado. En consecuencia, la representación densa produce un fenómeno habitual en cálculo simbólico conocido como el "aumento de las expresiones intermedias" (intermediate expression swell, ver [Dia97], [Mos71], [Par95]), que inhibe la aplicación de algoritmos simbólicos en situaciones prácticas.

Debido a razones geométricas y algebraicas es imposible, al menos en el peor caso, evitar el incremento exponencial de los grados de los resultados intermedios de los algoritmos simbólicos para problemas paramétricos álgebra lineal o eliminación en geometría. De hecho, los problemas algebraicos típicos en anillos de polinomios son completos en espacio exponencial ([MM82], [May97]) y los problemas de eliminación geométricos son # P–duros ([HM93], [Koi96], [Mat97]). La situación general de los algoritmos para problemas de álgebra lineal con matrices polinomiales es similar.

En contraposición a la situación del álgebra lineal sobre un cuerpo K arbitrario (ver [BCS97] por ejemplo), el impacto del tamaño de los datos sobre dominios concretos como $\mathbb{Q}[X_1, \ldots, X_n]$ o $\mathbb{Z}$ puede ser muy importante. Por ejemplo, sobre equipamiento estándar no es posible calcular en `Maple` el polinomio característico de una matriz 4×4 cuyos coeficientes son polinomios con coeficientes enteros de grado 5 en 10 variables (ver los resultados comparativos del Capítulo 6 y [CHLM00]).

A fin de evitar este comportamiento exponencial, en esta tesis los polinomios multivariados que aparecen como coeficientes de las matrices (o sistemas de ecuaciones polinomiales) en consideración serán representados por medio de *programas* (straight–line programs). Un straight–line program que representa un conjunto $\{F_1, \ldots, F_s\}$ de polinomios de $\mathbb{Q}[X_1, \ldots, X_n]$ es un algoritmo sin ramificaciones que evalúa $F_1, \ldots, F_s$ en un punto arbitrario $x \in \mathbb{C}^n$.

Una medida básica de complejidad de un straight–line program es la cantidad de operaciones aritméticas que realiza, que denominamos su *tiempo*. Asimismo, vamos a prestar atención al tiempo paralelo *no escalar*, que es la cantidad de operaciones aritméticas "anidadas" entre polinomios no constantes. En [KP96, Lemma 14] (ver también [Mat99, Lemma 2]) se establece una cota optimal para el grado de los resultados intermedios de un straight–line program en términos del tiempo no escalar del mismo. En vistas de este resultado, solamente vamos a considerar algoritmos de álgebra lineal de bajo tiempo paralelo no escalar (típicamente polilogarítmico en el tamaño del sistema en consideración), donde el tamaño de los resultados intermedios es controlable. De esta manera, la propia representación straight–line program de los resultados intermedios constituye una "compresión" exponencial de los mismos en comparación con su representación densa o rala.

Los algoritmos de álgebra lineal que vamos a considerar satisfacen los siguientes principios: utilizan la representación straight–line program de polinomios multivariados y poseen baja complejidad paralela no escalar. Asimismo, teniendo en cuenta que nuestros algoritmos resuelven problemas paramétricos, vamos a evitar las divisiones por elementos no escalares ya que éstas producen divisiones por cero en instancias paramétricas particulares.

Estos algoritmos de álgebra lineal se aplican en la solución de dos tareas básicas de eliminación en geometría: el cálculo de la dimensión del espacio de parámetros libres asociado a un sistema de ecuaciones polinomiales y la resolución de ciertos sistemas de ecuaciones polinomiales con finitas soluciones.

Los métodos clásicos para el cálculo de la dimensión del espacio de parámetros generalmente calculan un sistema de generadores del sistema en consideración con ciertas propiedades especiales (por ejemplo una base de Gröbner [Buc85]) por medio de técnicas de reescritura que manipulan la representación densa de los polinomios de entrada. Este sistema de generadores no sólo permite obtener la dimensión del espacio de parámetros sino también una descripción explícita de la estructura

algebraica asociada al sistema. En particular, esto implica que son *completos en espacio exponencial* (ver [MM82], [May97], [Mat99]).

En lugar de seguir esta línea vamos a utilizar un algoritmo propuesto en [DFGS91], cuya complejidad es polinomial en el tamaño de la representación densa del sistema de entrada. Este algoritmo se basa en una observación simple: la dimensión del espacio de parámetros es igual a la cantidad de "grados de libertad" del sistema, es decir, la cantidad máxima de variables que se pueden especializar en un valor genérico conservando la compatibilidad del sistema. A fin de testear la compatibilidad se utilizan teoremas de ceros efectivos (ver [CGH89], [DFGS91]) que reducen el problema a cuetiones de álgebra lineal con matrices polinomiales. Así, aplicando nuestros algoritmos de álgebra lineal se obtiene un algoritmo exponencialmente más eficiente en el caso peor que los que utilizan técnicas de reescritura para determinar la dimensión del espacio de parámetros.

El segundo problema de eliminación que vamos a tratar es la resolución de ciertos sistemas de ecuaciones polinomiales con finitas soluciones por medio de técnicas de deformación. La idea es obtener un sistema paramétrico del cual nuestro sistema original representa una instancia paramétrica. Si el sistema paramétrico posee a su vez una instancia paramétrica "suave" fácil de resolver, es posible "mover" dicha solución a través del espacio de parámetros a fin de resolver el sistema original.

Este proceso se basa en una versión global de la iteración de Newton simbólica de [GHM+98], [GHH+97] para la aproximación de ramas analíticas de un sistema multidimensional. La iteración de Newton simbólica "clásica" es una versión efectiva del Teorema de la Función implícita que permite, a partir de un punto "suave" de un sistema paramétrico dado, obtener aproximaciones de las series de potencia que representan localmente el conjunto solución por medio de una iteración del operador de Newton (ver [Eis95] por ejemplo). En [GHM+98], [GHH+97] se propone una versión global de esta iteración de Newton que parte de todas las soluciones de la instancia paramétrica dada y obtiene una aproximación de todas las ramas del conjunto solución del sistema paramétrico. Conociendo cotas de grado para los polinomios que describen el conjunto solución [SS96], a partir de una aproximación de orden suficientemente alto de las series que lo describen se obtienen los polinomios buscados en forma exacta.

La complejidad de este algoritmo depende en forma cuadrática de la cantidad de soluciones de la instancia general del sistema paramétrico y en forma lineal de la complejidad del cálculo de la matriz inversa (o alternativamente de la matriz adjunta y el determinante) de la matriz Jacobiana del sistema. Por lo tanto, teniendo en cuenta que la dependencia cuadrática del invariante geométrico mencionado parece ser inevitable ([HMPW98], [CGH+03]), en los ejemplos concretos que consideramos las deformaciones conservan la estructura de la matriz jacobiana del sistema original. Así, la complejidad de nuestro algoritmo para la resolución de sistemas de ecuaciones polinomiales mejora debido a los algoritmos para el tratamiento de la

matriz Jacobiana, lo cual implica una importante mejora de la complejidad de todo el algoritmo.

La implementación utiliza un ambiente de programación [Bru99] implementado en la librería `PARI`[1], que permite la manipulación de polinomios representados por straight–line programs. Según fuera señalado en [Dia97] (ver también [GHL+00]), la representación explícita de un straight–line program por medio del correspondiente grafo de computación es ineficiente. Sin embargo, los algoritmos de eliminación que nos interesan no requieren el almacenamiento de las funciones con las que operan, sino la evaluación de las mismas sobre argumentos numéricos bien especificados. Por lo tanto, en la librería `SLP` que utilizamos el concepto teórico de straight–line programs se realiza por medio del modelo computacional de "cajas negras" (black boxes). Una caja negra representa una codificación implícita de un straight–line program por medio de un proceso de evaluación generado algorítmicamente y permite la evaluación económica en espacio del valor de la función correspondiente en una entrada numérica dada.

Hemos testeado nuestra implementación en ciertos problemas paramétricos provenientes de aplicaciones ingenieriles y científicas. La conveniencia de nuestro enfoque algorítmico se refleja en los resultados comparativos que hemos obtenido. En efecto, la diferencia de desempeño entre `Maple` y nuestra implementación en estos ejemplos muestra las enormes ventajas que se obtienen a partir de la introducción de la estructura de datos straight–line program y su correspondiente realización computacional por medio de cajas negras.

[1] `http://www.math.u-psud.fr/~belabas/pari.html`

Capítulo 2

Modelo computacional

En el ámbito de la teoría de complejidad los objetos matemáticos generalmente tienen una representación *canónica* (por ejemplo, los enteros se codifican por medio de su representación bit, los polinomios por sus coeficientes, etc.). Una vez que se fija una representación, la teoría de complejidad busca algoritmos eficientes a fin de resolver los problemas matemáticos en consideración y trata de certificar la optimalidad de los mismos.

Sin embargo, la situación es más complicada en el caso particular de los problemas de eliminación paramétricos que son el objeto de estudio de esta tesis. En este caso, la teoría de complejidad requiere la optimización *simultánea* de la representación y los algoritmos. A fin de ilustrar esta afirmación consideramos un problema clásico de eliminación. Sea $A := (A_{ij})_{1\leq i,j\leq n}$ una matriz de indeterminadas. La matriz A determina un sistema de ecuaciones lineales homogéneo cuyas solubilidad puede expresarse por medio de la siguiente fórmula $\Phi(A)$:

$$(\exists X_1)\cdots(\exists X_n)\Big(X_1 \neq 0 \vee \cdots \vee X_n \neq 0 \wedge \bigwedge_{i=1}^{n}\sum_{j=1}^{n} A_{ij}X_j = 0\Big).$$

Esta fórmula $\Phi(A)$ es equivalente a la siguiente fórmula libre de cuantificadores:

$$\det(A) \neq 0,$$

donde $\det(A) := \sum_{\sigma\in S_n} \operatorname{sign}(\sigma) A_{1\sigma(1)}\cdots A_{n\sigma(n)}$ es un polinomio de $\mathbb{Q}[A_{ij} : 1 \leq i,j \leq n]$ con $n!$ términos.

La forma habitual de verificar la validez de la fórmula $\Phi(A)$ para una especialización de A en una matriz $a := (a_{ij})_{1\leq i,j\leq n} \in \mathbb{C}^{n\times n}$ consiste en calcular $\det(a)$ y decidir si $\det(a) = 0$. Si escribimos $\det(A)$ de la forma tradicional como un polinomio ralo en las entradas de la matriz reemplazamos la fórmula $\Phi(A)$ por una fórmula de longitud $O(n!)$. En lugar de esto, en esta tesis vamos a representar el polinomio $\det(A)$ por un algoritmo que calcula el valor $\det(A)$ a partir de las variables $A_{11},\ldots,A_{nn}$. Por ejemplo, utilizando el algoritmo de Samuelson (ver la

Sección 3.1.1) vamos a representar $\det(A)$ por medio de un algoritmo de tiempo $O(n^4)$.

2.1 Estructuras de Datos para la Representación de Polinomios

Los programas generales de cálculo simbólico ofrecen a sus usuarios una variedad de herramientas para la manipulación de polinomios y funciones racionales. Estos programas generalmente se basan en la representación densa o rala de los polinomios multivariados y son sumamente ineficientes en la práctica.

En el contexto de la teoría de la eliminación en geometría este fenómeno de ineficiencia práctica tiene una explicación clara. Se puede observar que un polinomio $F \in \mathbb{Q}[X_1, \ldots, X_n]$ de grado d tiene $\binom{d+n}{n} \approx O(d^n)$ monomios distintos. Por lo tanto, la representación densa del polinomio F tiene un comportamiento exponencial con respecto a la cantidad n de variables. Además, los algoritmos usuales de eliminación tienden a producir expresiones densas de grado exponencial a partir de polinomios de entrada ralos de bajo grado. Existe un conocido ejemplo debido a D. Lazard, T. Mora, W. Masser and P. Philippon (ver [Bro87]) que ilustra este fenómeno.

De esta observación concluimos que el comportamiento exponencial de los algoritmos de eliminación es inevitable si no se utilizan estructuras de datos alternativas para representar polinomios multivariados. Una posibilidad es su codificación por medio de circuitos aritméticos y sus esquemas de evaluación asociados (*straight–line programs*) (ver [Par95], [vzG86], [vzG93], [BCS97]).

Sea $\mathbb{Q}(\underline{X}) = \mathbb{Q}(X_1, \ldots, X_n)$ el cuerpo de funciones racionales sobre $\mathbb{Q}$ en las variables $X_1, \ldots, X_n$. Un circuito aritmético β en $\mathbb{Q}(\underline{X})$ consiste de un *grafo dirigido acíclico* (*dag*), cuyos nodos tienen grado de entrada (indegree) 0 o 2 y están etiquetados de acuerdo a las reglas que detallamos a continuación.

Los nodos con grado de entrada 0 son etiquetados por elementos del conjunto $\mathbb{Q} \cup \{\underline{X}\}$, mientras que los nodos con grado de entrada 2 (llamados *nodos internos*) son etiquetados por una de las operaciones aritméticas suma, resta, multiplicación o división. Los nodos con grado de entrada 0 etiquetados por elementos de $\{\underline{X}\}$ se denominan los *nodos de entrada.* Los nodos con grado de entrada 0 etiquetados por elementos de $\mathbb{Q}$ se denominan *nodos de parámetros.* Los elementos de $\mathbb{Q}$ que aparecen de esta forma son los *parámetros* del circuito aritmético β. Finalmente, algunos nodos del circuito aritmético β se llaman *nodos de salida* (típicamente, los nodos de salida serán los nodos con grado de salida 0). Denotamos el *dag* subyacente a β por $\Gamma(\beta)$.

Si se comienza a partir de los nodos de entrada y se sigue el *dag* $\Gamma(\beta)$, a cada nodo ρ le corresponde de manera natural una función racional Q_ρ que se obtiene como

resultado de todos los pasos previos. Dadas s funciones racionales $F_1, \ldots, F_s \in \mathbb{Q}(\underline{X})$ y un circuito aritmético β en $\mathbb{Q}(\underline{X})$ con s nodos de salida, diremos que $F_1, \ldots, F_s$ son *representados* por β si $F_1, \ldots, F_s$ son las funciones racionales asociadas a los nodos de salida de β.

A fin de modelar el cálculo con circuitos aritméticos introducimos la noción de *pebble game* (juego de fichas). Un pebble game convierte un circuito aritmético β dado en un algoritmo secuencial (llamado *straight–line program*) y asocia a β medidas de espacio y tiempo naturales. En el grafo $\Gamma(\beta)$ del circuito aritmético β podemos jugar un pebble game sujeto a las siguientes reglas (ver [Bor93]):

P1 Cualquier nodo con grado de entrada 0 de $\Gamma(\beta)$ puede recibir un pebble.

P2 Si los nodos predecesores de un nodo ρ de $\Gamma(\beta)$ tienen ambos un pebble, entonces ρ puede recibir un pebble, ya sea agregando un nuevo pebble o moviendo algún pebble de uno de sus predecesores a ρ.

P3 Un pebble puede ser siempre removido de un nodo de $\Gamma(\beta)$.

Un pebble game termina cuando todos los nodos de salida de $\Gamma(\beta)$ han recibido un pebble. Observamos que el grafo $\Gamma(\beta)$ no determina unívocamente un pebble game (es decir, es posible jugar diferentes pebble games sobre $\Gamma(\beta)$).

Dado un pebble game sobre el grafo $\Gamma(\beta)$, tenemos las siguientes medidas de complejidad:

C1 Una medida de *espacio*, dada por el número máximo de pebbles utilizados en cualquier momento del juego.

C2 Una medida de *tiempo*, dada por el número de ubicaciones de pebbles realizados durante el juego siguiendo las reglas P1 y P2.

Cualquier pebble game fijo sobre el grafo $\Gamma(\beta)$ de β define una estrategia de evaluación de β que llamamos un *straight–line program*. Un straight–line program en $\mathbb{Q}(\underline{X})$ que calcula funciones racionales $F_1, \ldots, F_s$ es una sucesión $(Q_1, \ldots, Q_r)$ de elementos de $\mathbb{Q}(\underline{X})$ con las siguientes propiedades:

1. $\{F_1, \ldots, F_s\} \subseteq \{Q_1, \ldots, Q_r\}$.
2. Para todo $1 \leq \rho \leq r$, la función racional Q_ρ pertenece a $\mathbb{Q} \cup \{\underline{X}\}$ o existen $1 \leq \rho_1, \rho_2 < \rho$ y una operación aritmética $op_\rho \in \{+, -, *, /\}$ tal que $Q_\rho = Q_{\rho_1}\ op_\rho\ Q_{\rho_2}$.

En álgebra computacional el uso de la representación de polinomios mediante straight–line programs surgió inicialmente en relación con la verificación de identidades polinomiales: dadas dos expresiones polinomiales multivariadas enteras (construidas mediante operaciones aritméticas), se trata de decidir si representan la misma

función polinomial. En este caso el uso de la representación densa conduce necesariamente a algoritmos exponenciales, mientras que utilizando la representación por straight–line programs se diseñaron algoritmos probabilísticos eficientes (ver [Zip79], [Sch80], [HS82]).

Posteriormente se estudiaron diversos problemas relacionados con la eliminación en polinomios univariados, tratando de aprovechar al máximo la representación straight–line program (ver [HS81], [Kal88], [Kal89], [Mat95]). Finalmente se desarrollaron métodos algorítmicos eficientes para problemas de eliminación multivariados que utilizan la representación straight–line program de las entradas, los resultados intermedios y las salidas del algoritmo (ver [GH93], [FGS95], [KP96], [GHH+97], [GHM+98], [HKP+00]).

Nuestro modelo de computación se basa en el concepto de circuitos aritméticos y straight–line programs. Sin embargo, un modelo de computación basado *solo* en circuitos aritméticos y straight–line programs no es suficientemente expresivo para nuestros propósitos. Por lo tanto, vamos a ampliar nuestro modelo a fin de incluir decisiones y selecciones (sujetas a decisiones previas). Por esta razón vamos a considerar *circuitos aritmético–booleanos* (también denominados *redes aritméticas*) en lugar de circuitos aritméticos, y *árboles de computación* en lugar de straight–line programs. Un circuito aritmético–booleano es simplemente un *dag* cuyos nodos se etiquetan por una operación aritmética o por una selección (que apunta a otros nodos), sujeta a una decisión previa sobre la validez de una identidad. Asociado a un pebble game sobre un circuito aritmético–booleano tenemos un árbol de computación, que determina una estrategia de evaluación del circuito aritmético–booleano dado. En otras palabras, un árbol de computación es un straight–line program con *ramificaciones*. Tiempo y espacio de la evaluación de un árbol de computación dado se definen en forma análoga a los de straight–line programs (ver [vzG86], [vzG93] o [BCS97] por más detalles sobre la noción circuito aritmético–booleano y árbol de computación).

Capítulo 3

Matrices polinomiales y straight–line programs

En este capítulo describimos los algoritmos de álgebra lineal para matrices con entradas en $R := \mathbb{Q}[X_1, \ldots, X_n]$ que vamos a utilizar en la resolución de los problemas de eliminación del Capítulo 5.

Como señalamos en la introducción, estos algoritmos satisfacen los siguientes principios: utilizan la representación straight–line program de polinomios multivariados, poseen baja complejidad paralela no escalar y no realizan divisiones no escalares.

Asimismo, dado que nuestra intención es resolver problemas de eliminación *particulares*, provenientes de problemas prácticos que en general poseen cierta "estructura", queremos obtener el máximo provecho de esta característica. Así, nos interesan los algoritmos que se aplican a matrices generales, pero poseen un mejor comportamiento sobre matrices "especiales".

En [KS91] se propone un modelo general para tal tipo de algoritmos, que se denomina álgebra lineal de tipo *caja negra* (ver también [Wie86], [vzGG99]). Este tipo de algoritmos propone un paradigma de matriz "especial": aquella en la cual la complejidad del producto matriz–vector es asintóticamente menor que la del caso general. En consecuencia, la complejidad de tales algoritmos se mide en términos de la cantidad de recursos computacionales necesarios para calcular el producto de la matriz de entrada con un vector arbitrario.

Si A es una matriz $n \times n$ general y v es un vector con n coordenadas arbitrario, el algoritmo clásico para calcular $A \cdot v$ requiere $2n^2$ operaciones aritméticas. Por otro lado, existen muchas clases de matrices especiales donde este costo se reduce significativamente. Por ejemplo, si A es una matriz de Sylvester o una matriz de Vandermonde, el costo se reduce a $O(\mathtt{M}(n))$ o $O(\mathtt{M}(n) \log n)$ operaciones aritméticas respectivamente, donde $\mathtt{M}(n)$ representa el costo de la multiplicación de dos polinomios univariados de grado n. Cabe destacar que $\mathtt{M}(n) = O(n^{\log_2 3})$ con el al-

goritmo de Karatsuba [KO62] y $\mathtt{M}(n) = O(n \log_2 n \log_2 \log_2 n)$ con el algoritmo de Schönhage–Strassen [SS71] (cf. [BP94], [vzGG99]). Otro caso importante es el de las matrices *ralas*, es decir, aquellas donde el número de entradas no nulas es del orden $s = o(n^2)$. En tal caso, el producto matriz–vector tiene complejidad $O(s)$.

La característica distintiva de los algoritmos de álgebra lineal de tipo caja negra es que son más eficientes que los algoritmos generalistas en el caso de matrices donde el producto matriz–vector es del orden $o(n^2)$.

Habiendo definido la clase de algoritmos que nos interesa, resta discutir la clase de complejidad espacio–tiempo que necesitamos con vistas a las aplicaciones del Capítulo 5. Los problemas de eliminación paramétricos generalmente producen matrices polinomiales de grandes tamaños o con entradas polinomiales de alto grado, que generan problemas de espacio de memoria. En consecuencia, nos interesan algoritmos con "poco" espacio de memoria. A este respecto, de [Ja'83, Theorem 5.2] deducimos el siguiente resultado:

Lema 1 *Sean $A_1, \ldots, A_r$ matrices $n \times n$ y consideremos el problema de calcular el producto matricial $A_1 \cdots A_r$ con un algoritmo que calcula productos de dos matrices por vez. Supongamos que existe un algoritmo que calcula $A_1 \cdots A_r$ con espacio sublineal $\mathcal{S} = O(n^c)$, donde $0 \leq c < 1$. Entonces el tiempo $\mathcal{T}$ de dicho algoritmo satisface:*

$$\mathcal{T} = \Omega(n^{(1-c)\log r}).$$

Demostración: Dado que para n suficientemente grande $\frac{n}{S} < 1$, aplicando [Ja'83, Theorem 5.2] concluimos que $\mathcal{T} = \Omega(n^{(1-c)\,l})$ para algún $l \geq \lceil \log r \rceil$. En particular, $\mathcal{T} = \Omega(n^{(1-c)\log r})$. ∎

Deducimos entonces que el producto iterado de n matrices de $n \times n$ tiene tiempo *no polinomial* $\mathcal{T} = \Omega(n^{(1-c)\log n})$ para todo algoritmo de espacio *sublineal* $\mathcal{S} = O(n^c)$ con $0 \leq c < 1$.

Por lo tanto, desde el punto de vista del tradeoff espacio–tiempo, la única clase conveniente de algoritmos económicos en espacio es la clase de espacio lineal. Por otro lado, si bien el Lema 1 considera un tipo restringido de algoritmos para el producto iterado (la de los algoritmos que calculan el producto de dos matrices por vez), cabe destacar que no se conocen algoritmos *generales* para el producto iterado que no sean esencialmente de esta forma.

Por otra parte, teniendo en cuenta que el producto iterado de n matrices de $n \times n$ y el cálculo del determinante, el polinomio característico, o la inversa de una matriz $n \times n$ entre otros son computacionalmente equivalentes ([vzG86], [vzG93], [BCS97]), la conclusión anterior se aplica a todos los algoritmos de álgebra lineal que vamos a considerar.

3.1 Polinomio Característico

Sea $R = \mathbb{Q}[X_1, \ldots, X_n]$ y A una matriz en $R^{n \times n}$. Sea λ una indeterminada. El polinomio característico $p \in R[\lambda]$ de A se define como

$$p(\lambda) := det(\lambda I - A),$$

donde I representa la matriz identidad $n \times n$.

El cálculo de los coeficientes del polinomio característico (y por ende del determinante) nos interesa por varios motivos. En primer lugar, a partir de los coeficientes del polinomio característico se obtiene la matriz adjunta y el determinante, que nos permiten resolver sistemas de ecuaciones lineales sin divisiones no escalares.

Por otro lado, los polinomios característicos son objetos básicos de la teoría de eliminación efectiva, donde se utilizan a fines de codificar soluciones de sistemas de ecuaciones polinomiales ([GVRR97], [Rou97], [Par95], [CLO98]). A partir del cálculo del polinomio característico de ciertos endomorfismos asociados a un sistema de ecuaciones polinomiales con finitas soluciones dado se obtiene información básica sobre el conjunto solución del mismo: la cantidad de soluciones, la multiplicidad de las mismas, la cantidad de soluciones reales, etc.

De acuerdo a nuestro planteo básico, no vamos a considerar algoritmos de cálculo del polinomio característico que realicen divisiones no escalares. Por lo tanto, no vamos a considerar algoritmos que combinan estrategias de interpolación con algoritmos de cálculo del determinante en base a eliminación Gaussiana.

Los algoritmos de baja complejidad paralela no escalar más prominentes para el cálculo del polinomio característico son el de Csanky [Csa76], el de Chistov [Chi85] y el de Berkowitz [Ber84], basados en los algoritmos de Samuelson, Krylov y Leverrier (cf. [FF63]). En [MV99] (ver también [Val92]) se presenta un marco combinatorio que permite comparar estos algoritmos. A una matriz $A = (a_{ij})_{1 \leq i,j \leq n}$ se asocia el grafo dirigido completo de n vértices G_A cuyo $< i, j >$–ésimo eje tiene peso a_{ij}. Así, el determinante de A es la suma (con signo) de los pesos de las permutaciones de S_n.

Sea $p = det(\lambda I - A) := \lambda^n + p_1 \lambda^{n-1} + \cdots + p_{n-1}\lambda + p_n$ y denotemos por $G_A(\lambda)$ el grafo correspondiente a la matriz $\lambda I - A$. Observamos que cada $\sigma \in S_n$ tiene una descomposición (única) en ciclos que forman un cubrimiento de $\{1, \ldots, n\}$. Entonces el coeficiente p_l es la suma (con signo) de las contribuciones de los cubrimientos por ciclos de $\{1, \ldots, n\}$ que tienen al menos $(n - l)$ 1–ciclos [MV99, Proposition 2.2].

En [MV99] se demuestra que los algoritmos Samuelson–Berkowitz, Csanky–Leverrier y Chistov expanden estas sumas a fin de incluir más términos que se cancelan mutuamente (ver [Val92] y [MV99, Theorems 3.6 y 3.8]), con una interpretación combinatoria que muestra que el algoritmo de Samuelson–Berkowitz es el más conservativo de los tres. Asimismo, cabe destacar el benchmarking comparativo

de [Abd97], que muestra que sobre matrices con entradas enteras el algoritmo de Samuelson–Berkowitz es también el más eficiente en la práctica. Por lo tanto, en lo que sigue vamos a considerar una versión del algoritmo de Samuelson–Berkowitz con espacio lineal.

3.1.1 El Método de Samuelson

A fin de describir el método de Samuelson, dividimos la matriz original A en cuatro partes

$$A := \left(\begin{array}{c|c} M & S \\ \hline R & a_{nn} \end{array}\right),$$

es decir:

- Una matriz de 1×1 formada por el elemento a_{nn}.
- Una matriz R de $1 \times (n-1)$ que contiene los elementos $a_{n1}, a_{n2}, \ldots, a_{n\,n-1}$.
- Una matriz S de $(n-1) \times 1$ que contiene los elementos $a_{1n}, a_{2n}, \ldots, a_{n-1\,n}$.
- Una matriz M de $(n-1) \times (n-1)$ formada por las $n-1$ primeras filas y las $n-1$ primeras columnas de la matriz A.

El método relaciona los coeficientes del polinomio característico de la matriz A con los coeficientes del polinomio característico de la matriz M. Aplicando este argumento recursivamente, se obtiene un algoritmo para el cálculo del polinomio característico p.

Sea $q(\lambda)$ el polinomio característico de M, y escribamos:

$$\begin{aligned} p(\lambda) &:= \lambda^n + \sum_{k=1}^{n} p_k \lambda^{n-k}, \\ q(\lambda) &:= \lambda^{n-1} + \sum_{k=2}^{n} q_k \lambda^{n-k}. \end{aligned}$$

Tenemos la *Fórmula de Samuelson* que relaciona los coeficientes de p y q:

Lema 2 ([FF63], [Ber84, §3])

$$p(\lambda) = (a_{nn} - \lambda)\, q(\lambda) + \sum_{k=2}^{n} \left((R \cdot M^{k-2} \cdot S)\, \lambda^{n-k} \right). \tag{3.1}$$

Esta identidad puede reescribirse como un sistema de ecuaciones lineales igualando los coeficientes correspondientes a monomios con la misma potencia en ambos miembros de (3.1). La matriz de este sistema es una matriz Toeplitz $C^{(n)} := (c_{ij}^{(n)})_{1\le i\le n+1, 1\le j\le n}$ triangular inferior de dimensión $(n+1)\times n$, cuyos coeficientes se definen de la siguiente forma:

$$c_{ij}^{(n)} := \begin{cases} 0 & \text{para } j > i, \\ -1 & \text{para } j = i, \\ a_{11} & \text{para } j - 1 = i, \\ R \cdot M^{i-1-j} \cdot S & \text{para } j < i - 1. \end{cases}$$

A fin de calcular los coeficientes $c_{ij}^{(n)}$ de la matriz $C^{(n)}$ aplicamos la estrategia de Abdeljaoued [Abd97], que requiere espacio lineal y tiempo $O(n^4)$. Realizando $n-1$ reducciones recursivas de este tipo obtenemos $n-1$ matrices Toeplitz $C^{(n)}, \ldots, C^{(2)}$ triangulares inferiores cuyo producto es igual al inverso aditivo del vector de coeficientes del polinomio característico p. Teniendo en cuenta que $C^{(i-1)} \cdots C^{(2)}$ es un vector columna de $i \times 1$, vemos que el producto $C^{(i)} \cdots C^{(2)}$ se reduce recursivamente a un producto matriz–vector de la matriz Toeplitz triangular inferior $C^{(i)}$ con el vector $C^{(i-1)} \cdots C^{(2)}$. Este producto se reduce fácilmente a la convolución de la primer columna de $C^{(i)}$ y el vector $C^{(i-1)} \cdots C^{(2)}$ (cf. [BP94]), cuyo costo $\mathtt{M}(i)$ es el de la multiplicación de dos polinomios de grado i con coeficientes en R. Así obtenemos un algoritmo que calcula el polinomio característico de A con espacio $O(n)$ y tiempo $O(n^4)$, cuya descripción sintética damos a continuación:

Inicializar $P = (-1, a_{11})^t$, $Vect_1 = -1$, $Vect_2 = a_{11}$.
Para $i = 2, \ldots, n$ **Hacer**
  Obtener submatriz $M = (a_{kl})_{1\le k,l\le i-1}$.
  Obtener vector fila $R = (a_{il})_{1\le l\le i-1}$.
  Obtener vector columna $S = (a_{li})_{2\le l\le i}$.
  Para $j = 3, \ldots, i$ **Hacer**
    $Vect_j = R \cdot S$.
    $S = M \cdot S$.
  Fin Para
$P =$ convolución(P,$Vect$).
Fin Para
Retornar $-P$.

Lema 3 *El algoritmo de Samuelson es un algoritmo de álgebra lineal de tipo caja negra.*

Demostración: Sea A una matriz $n \times n$, y sean $\mathcal{S}_A$ y $\mathcal{T}_A$ el espacio y tiempo necesarios para calcular el producto matriz–vector $A \cdot v$ de A con un vector v de n coordenadas arbitrario. De la definición de las submatrices R, S y M deducimos que

el producto de loop interior $S = M \cdot S$ se reduce fácilmente a un producto matriz–vector del tipo $A \cdot \widetilde{S}$ completando eventualmente S con ceros de forma adecuada. En consecuencia, el loop interior se calcula con espacio $O(\mathcal{S}_A + i)$ y tiempo $O(i\mathcal{T}_A)$. Concluimos entonces que el algoritmo se calcula con espacio $O(\mathcal{S}_A + n)$ y tiempo $O(n^2\mathcal{T}_A + n\,\mathtt{M}(n))$. En particular, si $\mathcal{T}_A = o(n^2)$, entonces el tiempo requerido por nuestro algoritmo es $o(n^4)$, lo cual demuestra el lema. ∎

3.2 Polinomio Minimal

Sea $R := \mathbb{Q}[X_1, \ldots, X_n]$ y sea A una matriz de $R^{n\times n}$. El *polinomio minimal* de A se define como el polinomio mónico de grado mínimo m_A con coeficientes en R que anula a la matriz A, es decir, tal que $m_A(A) = 0_{n\times n}$ (donde $0_{n\times n}$ denota la matriz nula de $n \times n$).

De su definición se deduce que el polinomio minimal de una matriz A tiene grado menor o igual que el polinomio característico de A (de hecho, lo divide). Por lo tanto, en aplicaciones tales como el cálculo de la matriz adjunta o la resolución de sistemas de ecuaciones lineales puede resultar una alternativa interesante al cálculo del polinomio característico.

Por otro lado, observamos que el conjunto de ceros de cualquier polinomio no nulo con coeficientes en $K := \mathbb{Q}(X_1, \ldots, X_n)$ que anula a A contiene todos los autovalores de A (ver por ejemplo [BP94, Remark 2.1.1]). Por lo tanto, en las aplicaciones a la teoría de eliminación efectiva donde se codifican soluciones de sistemas de ecuaciones polinomiales por medio de polinomios característicos, es posible reemplazar éstos por polinomios minimales sin "perder soluciones". Este enfoque en teoría de eliminación efectiva corresponde a lo que [Laz93] y [Par95] entre otros denominan el "punto de vista geométrico" (en contraposición al punto de vista algebraico): el objeto central de estudio es el conjunto de soluciones de un sistema de ecuaciones polinomiales dado, y no el sistema que lo define. De hecho, de acuerdo con [GH01], el reemplazo de polinomios característicos por minimales en teoría de eliminación efectiva permitió una "enorme reducción del grado, la altura y también la complejidad straight–line program de los polinomios" que aparecían como resultados intermedios (ver [GH93], [GHS93], [FGS95]).

En el cálculo de los coeficientes del polinomio minimal vamos a seguir el enfoque de Wiedemann [Wie86] para la resolución de sistemas de ecuaciones lineales ralos. En [KS91] el enfoque de Wiedemann se generalizó al contexto más general de algoritmos de álgebra lineal tipo caja negra. En la sección siguiente vamos a exhibir una adaptación del algoritmo de Wiedemann que no realiza divisiones no escalares.

3.2.1 El método de Wiedemann

Sea $K := \mathbb{Q}(X_1, \ldots, X_n)$ y sea V un K–espacio vectorial. Una sucesión $(a_i)_{i\geq 0}$ de elementos de V se dice $(a_i)_{i\geq 0}$ *linealmente generada* sobre K si y sólo si existen elementos $c_0, \cdots, c_n \in K$ tales que $n \geq 0$, $c_k \neq 0$ para algún k con $0 \leq k \leq n$ y se satisface la siguiente relación:

$$c_0 a_j + c_1 a_{j+1} + \cdots + c_n a_{j+n} = 0 \text{ para todo } j \geq 0.$$

El polinomio $c_0 + c_1\lambda + \cdots + c_n\lambda^n$ se dice un *polinomio generador* de $(a_i)_{i\geq 0}$. El conjunto de todos los polinomios generadores para $(a_i)_{i\geq 0}$ en $K[X]$ tiene un único polinomio mónico de grado mínimo que lo genera, llamado el *polinomio mínimo generador* de la sucesión $(a_i)_{i\geq 0}$.

Sea $A \in K^{n\times n}$ una matriz cuadrada. Del teorema de Hamilton–Cayley se deduce que la sucesión $(A^i)_{i\geq 0}$ está generada linealmente y su polinomio mínimo m_A es el polinomio minimal de la matriz A.

Sea $b \in K^{n\times 1}$ un vector columna. La sucesión $(A^i b)_{i\geq 0}$ también está generada linealmente y su polinomio minimal $m_{A,b}$, que no necesariamente coincide con m_A, es un divisor de m_A. Análogamente, para todo vector $u \in K^n$ la sucesión $(uA^i b)_{i\geq 0}$ está generada linealmente y su polinomio minimal $m^u_{A,b}$ es un divisor de $m_{A,b}$.

Teorema 4 [KS91, Theorem 1] *Sea d el grado de $m_{A,b}$, y sea W el subespacio formado por todos los polinomios de grado menor que d en $K[\lambda]$. Entonces existe una función lineal suryectiva $\ell : K^{1\times n} \to W$ tal que para todo $u \in K^{1\times n}$ la identidad $m^u_{A,b} = m_{A,b}$ se satisface si y sólo si* $\operatorname{mcd}(M_{A,b}, \ell(u)) = 1$.

Se deduce entonces que la probabilidad que valga la identidad $m_{A,b} = m^u_{A,b}$ para u elegido aleatoriamente es esencialmente la probabilidad que un polinomio de grado menor a m sea coprimo con $m_{A,b}$. Más precisamente, se tiene el siguiente resultado:

Lema 5 [KS91, Lemma 2] *Sean $A \in K^{n\times n}$, $b \in K^n$ y $S \subset K$ un subconjunto finito. Sea $u \in S^{1\times n}$ un vector fila y $b \in S^n$ un vector columna elegidos en forma aleatoria y uniforme. Entonces tenemos:*

$$Prob(m^u_{A,b} = m_A) \geq 1 - \frac{2\,\mathrm{gr}(m_A)}{\#S}.$$

El problema es entonces encontrar el polinomio mínimo que genera la sucesión de recurrencia lineal $(uA^i b)_{i\geq 0}$. Sea $a_j := uA^j b$ para $j = 0, \ldots, 2n-1$ y $a := (a_0, \ldots, a_{2n-1})$. Para $0 \leq N \leq 2n-1$ definimos $L_N(a)$ como el grado mínimo de un polinomio generador de $a_0, a_1, \ldots, a_{N-1}$. Es claro que $L_N(a) \leq N$.

Teorema 6 [Mas69, Theorem 1] *Si algún polinomio de grado mínimo L genera la sucesión $a_0, a_1, \ldots a_{N-1}$ pero no la sucesión $a_0, \ldots, a_{N-1}, a_N$, entonces todo polinomio que genera la sucesión $a_0, \ldots, a_{N-1}, a_N$ tiene longitud $L' \geq N + 1 - L$.*

Dado que el polinomio mínimo que genera la sucesión $(a_i)_{i\geq 0}$ tiene grado a lo sumo n, se deduce que el polinomio mínimo generador de la sucesión $(a_0, \ldots, a_{2n-1})$ es el polinomio mínimo de la sucesión $(a_i)_{i\geq 0}$.

3.2.2 El Algoritmo de Berlekamp–Massey

A fin de calcular el polinomio mínimo generador de $(a_0, \ldots, a_{2n-1})$ aplicamos el algoritmo de Berlekamp–Massey [Mas69]. El algoritmo de Berlekamp–Massey es un procedimiento recursivo que calcula el polinomio mínimo generador de las sucesiones $a_0, a_1, \ldots, a_{N-1}$ para $N = 1, \ldots, 2n-1$.

Para cada $N = 1, \ldots, 2n-1$, definimos el polinomio de conexión de la sucesión $a_0, \ldots, a_N$ como el polinomio

$$C^{(N)}(\lambda) := 1 + c_1^{(N)}\lambda + c_2^{(N)}\lambda^2 + \cdots + c_{L_N(a)}^{(N)}\lambda^{L_N(a)}$$

de grado mínimo $L_N(a)$ en λ que genera la sucesión $a_0, \ldots, a_N$. Por convención, definimos $C^{(0)}(\lambda) := 1$ para la sucesión de longitud $N = 0$.

Supongamos inductivamente que hemos calculado $L_N(a)$ y el polinomio de conexión $C^{(N)}$ para $N = 1, 2, \ldots, m$. Se trata ahora de calcular $L_{m+1}(a)$ y el polinomio de conexión $C^{(m+1)}$. Entonces, por hipótesis inductiva se deduce que:

$$a_i + \sum_{i=1}^{L_m(a)} c_i^{(m)} \cdot a_{j-i} = \begin{cases} 0 & \text{para } j = L_m(a), \ldots, m-1, \\ d_m & \text{para } j = m, \end{cases} \tag{3.2}$$

donde d_m se denomina la m–ésima *discrepancia*. Si d_m=0 entonces $C^{(m)}$ genera también los primeros $m+1$ dígitos de a de manera que $L_{m+1}(a) = L_m(a)$. Por lo tanto, podemos tomar $C^{(m+1)}(\lambda) = C^{(m)}(\lambda)$. Por otro lado, si $d_m \neq 0$ tenemos el siguiente resultado:

Lema 7 [Mas69, Theorem 2] *Si un polinomio de grado $L_m(a)$ genera la sucesión $a_0, a_1, \ldots, a_{m-1}$ pero no genera $a_0, a_1, \ldots, a_m$, entonces*

$$L_{m+1}(a) = \max\{L_m(a), m + 1 - L_m(a)\}.$$

Sea $k < m$ el mayor número entero tal que $L_k(a) < L_m(a)$ y $C^{(k)}$ el polinomio de conexión correspondiente. Tenemos entonces el siguiente resultado:

Teorema 8 [Mas69, Theorem 3] *El conjunto de polinomios*

$$\{C^{(m)}(\lambda) + Q(\lambda)\lambda^{m-k}C^{(k)}(\lambda) : \operatorname{gr}(Q(\lambda)) < 2L_m - n\}$$

es el conjunto de polinomios de conexión de todas las sucesiones de longitud mínima $L_{m+1}(a)$ generadas linealmente a partir de $a_0, \ldots, a_{m-1}$.

Del Teorema 8 se deduce que el polinomio $C^{(m+1)}$ buscado es de la forma $C^{(m)} + Q\,\lambda^{m-k}C^{(k)}$ para algún polinomio $Q \in K[\lambda]$ de grado menor que $2L_m - n$. Eligiendo $Q := d_m d_k^{-1}$ se comprueba fácilmente que el polinomio $C^{(m+1)}$ así obtenido genera la sucesión $a_0, \ldots, a_m$ y por lo tanto, es el polinomio de conexión buscado. A

fin de evitar divisiones, vamos a calcular el múltiplo $d_k C^{(m+1)}(\lambda) = d_k C^{(m)}(\lambda) + d_m \lambda^{m-k} C^{(k)(\lambda)}$.

Inicializar $C_0(\lambda) = 1$, $L = 0$, $B_0(\lambda) = 1$, $b = 1$, $e = 1$, $aux = 1$.
Para $N = 0 \ldots 2n-1$ **Hacer**
  Asignar $d = a_N + \sum_{i=1}^{L} c_i a_{N-i}$.
  Si $d = 0$ **Entonces**
    Asignar $e = e + 1$.
  Sino
    Si $2L > N$ **Entonces**
      Asignar $C(\lambda) = C(\lambda) - d\lambda^e B(\lambda)$, $e = e + 1$, $aux = aux \cdot b$.
    Sino
      Asignar $T(\lambda) = C(\lambda)$, $C(\lambda) = bC(\lambda) - d\lambda^e B(\lambda)$, $B(\lambda) = T(\lambda)$.
      Asignar $aux = aux \cdot b$, $L = N + 1 - L$, $b = d$, $e = 1$.
    Fin Si
  Fin Si
Fin Para
Retornar $C(\lambda)$.

El espacio que requiere este algoritmo es esencialmente el necesario para almacenar la representación densa de los polinomios B, C, T, es decir $O(n)$. Dado que cada ciclo del loop principal requiere $O(n)$ operaciones aritméticas, concluimos que el algoritmo requiere $O(n^2)$ operaciones aritméticas en total.

3.2.3 Algoritmo para el Cálculo de Polinomio Minimal

Combinando el algoritmo de Berlekamp–Massey con las ideas de Wiedemann obtenemos el siguiente algoritmo para el cálculo del polinomio minimal de una matriz $A \in R^{n \times n}$:

Obtener en forma aleatoria un vector fila $u \in S^{1 \times n}$.
Obtener en forma aleatoria un vector columna $v \in S^n$.
Asignar $a_j = uA^j v$ para $j = 0, \cdots, 2n-1$.
Calcular el polinomio mínimo generador de $(a_0, \ldots, a_{2n-1})$.

Sea A una matriz de $n \times n$ y sean S_A y $\mathcal{T}_A$ el espacio y tiempo necesarios para calcular el producto matriz–vector $A \cdot v$ de A con un vector v de R^n arbitrario. El cálculo de $a_0, \ldots, a_{2n}$ requiere espacio $O(S_A + n)$ y tiempo $O(n\mathcal{T}_A)$. Por lo tanto, el algoritmo se ejecuta con espacio $O(S_A + n)$ y tiempo $O(n\mathcal{T}_A + n^2)$.

3.3 Matriz Adjunta y Seudo–adjunta

La matriz adjunta $adj(A) := (b_{ij})_{1 \leq i,j \leq n}$ de una matriz A de $n \times n$ es la matriz definida por $b_{ij} := (-1)^{i+j} \det(M_{ji})$, donde M_{ji} es la submatriz de A de dimensión $(n-1) \times (n-1)$ que se obtiene a partir de A eliminando su j–ésima fila y su i–ésima columna.

La matriz adjunta es importante para nuestros propósitos dado que $adj(A) \cdot A = A \cdot adj(A) = \det(A)I$, donde I denota la matriz identidad $n \times n$. Por lo tanto, la matriz adjunta $adj(A)$ es un múltiplo de la matriz inversa de A cuyas entradas pertenecen a R.

Describimos a continuación un procedimiento que permite calcular la matriz adjunta de una matriz A no singular a partir de los coeficientes de su polinomio característico $p(\lambda) := \lambda^n + p_1\lambda^{n-1} + \cdots + p_n \in R[\lambda]$. Por el teorema de Hamilton–Cayley [Ja'83, Theorem 8.15] tenemos:

$$A^n + p_1 A^{n-1} + \cdots + p_{n-1}A + p_0 I = 0_{n \times n}.$$

Multiplicando ambos términos de esta identidad por $adj(A)$ deducimos:

$$adj(A)A(A^{n-1} + p_1 A^{n-2} + \cdots + p_{n-1}I) = -p_0\, adj(A). \tag{3.3}$$

Dado que $adj(A) \cdot A = \det(A) \cdot I$ y $\det(A) \neq 0$ (ya que A es no singular), reemplazando en (3.3) tenemos:

$$adj(A) = -(A^{n-1} + p_1 A^{n-2} + \cdots + p_{n-1}I). \tag{3.4}$$

A fin de calcular el miembro derecho de (3.4), aplicamos la regla de Horner a cada columna de $adj(A)$: la j–ésima columna $b^{(j)}$ de $adj(A)$ se obtiene a partir de la identidad $b^{(j)} = -(A^{(n-1)} + p_1 A^{n-2} + \cdots + p_{n-1}I)e^{(j)}$, donde $e^{(j)}$ es el j–ésimo vector canónico de R^n. En conclusión, el algoritmo para el cálculo de la matriz adjunta puede describirse sintéticamente de la siguiente forma:

Calcular los coeficientes $(p_1, \ldots, p_n)$ del polinomio característico de A.
Para $i = 1, \ldots, n$ **Hacer**
  Inicializar Vect=i–ésima columna de A.
  Para $j = 1, \ldots, n-2$ **Hacer**
    $Vect = Vect - p_j\, e^{(i)}$.
    Calcular el producto $Vect = A \cdot Vect$.
  Fin Para
  Retornar $-(Vect - p_{n-1}\, e^{(i)})$ como la i–ésima columna de A.
Fin Para

Sea dada una matriz de entrada $A \in R^{n \times n}$ tal que el producto matriz–vector $A \cdot v$ de A con un vector v de R^n arbitrario requiere espacio $\mathcal{S}_A$ y tiempo $\mathcal{T}_A$. Entonces

los coeficientes del polinomio característico de A se calculan con espacio $O(\mathcal{S}_A + n)$ y tiempo $O(n^2\mathcal{T}_A + n\,\mathtt{M}(n))$ (ver la Sección 3.1.1). Por otro lado, el loop principal se ejecuta con espacio $O(\mathcal{S}_A + n)$ y tiempo $O(n\mathcal{T}_A)$. En consecuencia, el algoritmo requiere espacio $O(\mathcal{S}_A + n)$ y tiempo $O(n^2\mathcal{T}_A + n\,\mathtt{M}(n))$.

Como hemos comentado, la matriz adjunta de una matriz $A \in R^{n\times n}$ nos interesa dado que nos provee un múltiplo escalar de la inversa de A con entradas en R. En este sentido, también podemos utilizar el polinomio minimal $m_A \in R[\lambda]$ de A en lugar del polinomio característico (o un múltiplo escalar del mismo). De hecho, si $m_A := \lambda^r + c_1\lambda^{r-1} + \ldots + c_r$, entonces

$$m_A(A) = A^r + c_1 A^{r-1} + \cdots + c_r I = A(A^{r-1} + c_1 A^{r-2} + \cdots + c_{r-1} I) + c_r I = 0_{n\times n}.$$

En consecuencia, tenemos

$$A(-A^{r-1} - c_1 A^{r-2} - \cdots - c_{r-1} I) = c_r I. \tag{3.5}$$

Definimos *seudo–adjunta* $sadj(A)$ y *seudo–determinante* $sdet(A)$ de A a

$$sadj(A) := -A^{r-1} - c_1 A^{r-2} - \cdots - c_{r-1} I \in R^{n\times n} \quad \text{y} \quad sdet(A) := c_r \in R$$

respectivamente. De (3.5) deducimos que $A \cdot sadj(A) = sdet(A) \cdot I$. Teniendo en cuenta que $sdet(A) \neq 0$ si A es no singular (dado que $sdet(A)$ divide a $det(A)$), concluimos que $sadj(A)$ es el $sdet(A)$–múltiplo de la matriz inversa de A y tiene entradas en R. Cabe destacar que mediante una simple adaptación del algoritmo para el cálculo de la matriz adjunta de A obtenemos un algoritmo que calcula la seudo–adjunta de A y el seudo–determinante de A con espacio $O(\mathcal{S}_A + n)$ y tiempo $O(n\mathcal{T}_A + n^2)$.

3.4 Rango

Sea A una matriz de $n \times n$ con entradas R. En esta sección consideramos el problema de calcular el rango de A. Para tal fin, vamos a seguir las ideas de [KS91], que proponen un algoritmo probabilístico eficiente de tipo caja negra.

Sean $U, L \in R^{n\times n}$ dos matrices Toeplitz triangulares de la siguiente forma:

$$U := \begin{pmatrix} 1 & u_2 & \ldots & u_n \\ & 1 & \ddots & \vdots \\ & & \ddots & u_2 \\ & & & 1 \end{pmatrix}, \quad L := \begin{pmatrix} 1 & & & \\ v_2 & \ddots & & \\ \vdots & \ddots & 1 & \\ v_n & \ldots & v_2 & 1 \end{pmatrix}, \tag{3.6}$$

donde $u_2, \ldots, u_n, w_2, \ldots, w_n$ son elementos de $\mathbb{Q}$ elegidos aleatoriamente. Sea $\tilde{A} \in R^{n\times n}$ la matriz $\tilde{A} := U \cdot A \cdot L$. Tenemos el siguiente resultado:

Teorema 9 [KS91, Theorem 2] *Sea r el rango de A. Si los coeficientes de U y L son elegidos aleatoriamente en un subconjunto finito $S \subset \mathbb{Q}$, entonces las submatrices principales $\tilde{A}_{ii}$ de $\tilde{A}$ para $i = 1, \ldots, r$ son no singulares con probabilidad al menos $1 - \frac{r(r+1)}{\#S}$.*

Por lo tanto, el problema se reduce a determinar el rango de la matriz $\tilde{A}$. Teniendo en cuenta que los menores principales son no nulos para $i = 1, \ldots, r$, el rango de $\tilde{A}$ (y por ende el de A) se podría determinar por medio de un proceso de búsqueda binaria de la mayor submatriz principal no nula. Desafortunadamente, este proceso agrega un factor $\log n$ a la complejidad del algoritmo. El siguiente resultado permite evitar este inconveniente:

Lema 10 [KS91, Lemma 2] *Sea D una matriz diagonal de $R^{n \times n}$ cuyas entradas se elijen aleatoriamente en un conjunto finito $S \subset \mathbb{Q}$. Entonces el grado del polinomio minimal de la matriz $\tilde{A} \cdot D$ es igual a la cantidad* $\mathrm{rango}(A) + 1$ *con probabilidad al menos $1 - \frac{n(n-1)}{2\#S}$.*

Combinando el Lema 10 con el algoritmo de la Sección 3.2.3 obtenemos un algoritmo de tipo caja negra que calcula el rango de una matriz $A \in R^{n \times n}$ dada. Este algoritmo se puede describir sintéticamente de la siguiente manera:

> Inicializar $S = \{1, \ldots, 2n^2\}$.
> Elegir aleatoriamente u, v en S^{n-1} y w en S^n.
> Inicializar $U = U(u)$ y $L = L(v)$ según (3.6).
> Inicializar $D = \mathrm{diag}(w)$.
> Asignar $B = U \cdot A \cdot L \cdot D$.
> Calcular el polinomio minimal m_B de B.
> Retornar $\mathrm{gr}(m_B) - 1$.

Sea $A \in R^{n \times n}$ una matriz de entrada tal que el producto matriz–vector $A \cdot b$ de A con un vector $b \in R^n$ arbitrario requiere espacio $\mathcal{S}_A$ y tiempo $\mathcal{T}_A$. El costo del algoritmo descripto es esencialmente el necesario para calcular el polinomio minimal de la matriz B. Siguiendo el algoritmo de la Sección 3.2.3, la parte más costosa es el cálculo de la sucesión $(b \cdot B^j \cdot c^t)_{0 \le j \le 2n-1}$, donde $b, c \in R^n$ son vectores elegidos aleatoriamente. Observamos que U y L son matrices Toeplitz triangulares, y por lo tanto el producto matriz–vector de U y L con un vector arbitrario de R^n se reduce fácilmente a una convolución (cf. [BP94]), cuyo costo denotamos por $\mathtt{M}(n)$. Asimismo, el producto de D con un vector arbitrario de R^n tienen costo a lo sumo $O(n)$. Concluimos que la sucesión $(b \cdot B^j \cdot c^t)_{0 \le j \le 2n-1}$ se calcula con espacio $O(\mathcal{S}_A + n)$ y tiempo $O(n\mathcal{T}_A + n\,\mathtt{M}(n))$. Por lo tanto, el algoritmo de cálculo de rango que hemos descripto requiere espacio $O(\mathcal{S}_A + n)$ y tiempo $O(n\mathcal{T}_A + n\,\mathtt{M}(n))$.

3.5 Un Caso de Aplicación: las Matrices Vandermonde

En esta sección vamos a ilustrar el comportamiento de los algoritmos de álgebra lineal descriptos a lo largo de este capítulo en una clase de matrices en particular: las matrices Vandermonde.

Una matriz Vandermonde $V := V(v) \in R^{n\times n}$ es una matriz de la siguiente forma:

$$V := \begin{pmatrix} 1 & v_0 & \dots & v_0^{n-1} \\ 1 & v_1 & \dots & v_1^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & v_{n-1} & \dots & v_{n-1}^{n-1} \end{pmatrix},$$

donde $v := (v_0, \dots, v_{n-1})$ es un vector de elementos de R arbitrario. Las matrices Vandermonde aparecen en relación a importantes cuestiones de cálculo científico tales como la interpolación, la reconstrucción modular polinomial, el Teorema Chino del Resto, la evaluación de Hermite y multipunto, FFT, etc. (cf. [Pan01]).

Nuestro propósito es mostrar como la multiplicación matriz–vector de una matriz Vandermonde V de $R^{n\times n}$ con un vector arbitrario de R^n puede realizarse con espacio $O(n)$ y tiempo $O(\mathtt{M}(n)\log n)$. Por lo tanto, concluimos que las matrices Vandermonde forman una clase de matrices conveniente para la aplicación de algoritmos de álgebra lineal de tipo caja negra.

Sea $b := (b_0, \dots, b_{n-1})$ un vector arbitrario de R^n y sea $p \in R[\lambda]$ el polinomio $p := b_0 + b_1\lambda + \cdots + b_{n-1}\lambda^{n-1}$. Entonces tenemos $V(v) \cdot b^t = (p(v_0), \dots, p(v_{n-1}))^t$. Por lo tanto, el problema del cálculo matriz–vector es equivalente a un problema de *evaluación multipunto*: dado (el vector de coeficientes de) un polinomio p de grado $n-1$ y n nodos $v_0, \dots, v_{n-1}$, se trata de calcular los valores $p(v_0), \dots, p(v_{n-1})$.

El algoritmo de Moenck–Borodin que resuelve el problema de la evaluación multipunto se basa en dos simples observaciones (cf. [Pan01]):

- $p(\lambda) \equiv p(a) \bmod (\lambda - a)$ para todo polinomio $p \in R[\lambda]$ y todo $a \in R$.
- $r(\lambda) \bmod q(\lambda) \equiv \Big(r(\lambda) \bmod (p(\lambda)q(\lambda))\Big) \bmod q(\lambda)$ para todo $p, q, r \in R[\lambda]$.

Sea $k := \lceil \log_2 n \rceil$. Definamos $r_0^{(k)} := p(\lambda)$, $m_j^{(0)} := \lambda - v_j$ para $j = 0, \dots, n-1$ y $m_j^{(0)} := 1$ para $j = n, \dots, 2^k - 1$. Con estas notaciones podemos describir el algoritmo de Moenck–Borodin:

Para $h = 0, \ldots, k-2$ **Hacer**
    Para $j = 0, \ldots, 2^{k-h}$ **Hacer**
        Asignar $m_j^{(h+1)} = m_{2j}^{(h)} m_{2j+1}^{(h)}$.
    Fin Para
Fin Para
Para $h = k-1, k-2, \ldots, 0$ **Hacer**
    Para $j = 0, \ldots, \min\{n, \lceil n/2^h \rceil - 1\}$ **Hacer**
        Asignar $r_j^{(h)} = r_{\lfloor j/2 \rfloor}^{(h+1)} \bmod m_j^{(h)}$.
    Fin Para
Fin Para
Retornar $p(v_i) = r_i^{(0)}$ para $i = 0, \ldots, n-1$.

En [Pan01] se demuestra la correctitud del algoritmo y el estimación $O(n)$ y $O(\mathtt{M}(n) \log_2 n)$ para la complejidad en espacio y tiempo del mismo respectivamente. Combinando este algoritmo con los de las Secciones 3.2.3 y 3.3 se obtienen algoritmos que calculan el polinomio minimal, el seudo–determinante y la seudo–adjunta de una matriz Vandermonde de $n \times n$ con espacio $O(n)$ y tiempo $O(n\,\mathtt{M}(n) \log n)$.

Capítulo 4

Implementación

En este capítulo se describirá cómo se han utilizado straight–line programs con el lenguaje `PARI` a los efectos de la implementación.

4.1 Representación de Straight–line Programs mediante Programas

Luego de haberse establecido en forma teórica la potencia de la representación de polinomios multivariados por medios de straight–line programs (ver la Sección 2.1 por un suscinto recorrido de este proceso), la necesidad de una implementación adecuada de dicha representación resultó evidente. En un primer momento se propusieron implementaciones en las cuales el grafo asociado a cada straight–line program se representaba en forma explícita (ver por ejemplo [FIKY88], [CLM96], [Häg96], [CHLM00]). Como se observa por ejemplo en [Dia97], la representación explícita del grafo asociado a un straight–line program es muy costosa como para permitir el tratamiento eficiente de problemas de tamaños reales.

El punto de inflexión a este enfoque provino de la observación que los grandes straight–line programs que aparecen en los problemas de álgebra computacional generalmente se componen de varios straight–line programs más pequeños. Dado que estamos interesados sólo en la evaluación de la función que resulta de tal composición, es posible representar cada straight–line program como una *caja negra* que contiene un programa a evaluar y datos de entrada que permiten su evaluación [Dia97].

En su tesis de Licenciatura [Bru98] (ver también [BHMW02]), Nicolás Bruno diseñó un lenguaje que permite la representación de straight–line programs. El lenguaje se denomina `MILONGA` (`Modularized Implementation Language Oriented to Nonlinear Geometric Applications`) y tiene las siguientes características:

- Los straight–line programs son ciudadanos de primera categoría: se permiten straight–line programs como parámetros de funciones y como resultados.

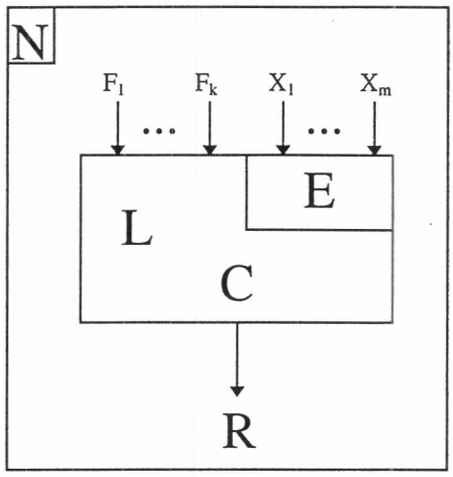

Figure 4.1: Componentes de una función

- Se admite la posibilidad de reducir la aridad del polinomio, sustituyendo algunas variables por valores (*evaluación parcial*).
- Es posible cambiar el anillo sobre el cual se evalúa un straight–line program: una misma expresión se puede evaluar por ejemplo sobre enteros y polinomios.
- Es posible manipular los straight–line programs (por ejemplo, calcular su derivada respecto de una variable) dentro del propio programa.
- En el caso de una evaluación módulo un entero o un polinomio, es posible de cambiar el módulo de la aritmética dentro del propio programa.

Una *función* en `MILONGA` admite parámetros y siempre devuelve un resultado. Las funciones tienen dos tipos de parámetros: fijos y extras. A fin de calcular la cantidad de parámetros extra de una función se provee una expresión de los parámetros fijos. Esto permite que las funciones tengan una cantidad variable de argumentos. Las funciones parcialmente evaluadas pueden ser usadas a su vez como parámetros de funciones y pueden devolverse como resultados.

Gráficamente (ver la Figura 4.1), una función en `MILONGA` puede verse como una caja N donde:

- $F_1, \ldots, F_k$ son los k parámetros fijos.
- E es una expresión entera en los parámetros fijos. Una vez evaluada, este número indica cuántos parámetros variables se deben considerar.
- $X_1, \ldots, X_m$ son los parámetros extra. El valor de m se conoce en tiempo de ejecución una vez que se instancian todos los parámetros fijos, de modo tal que la expresión E puede evaluarse.

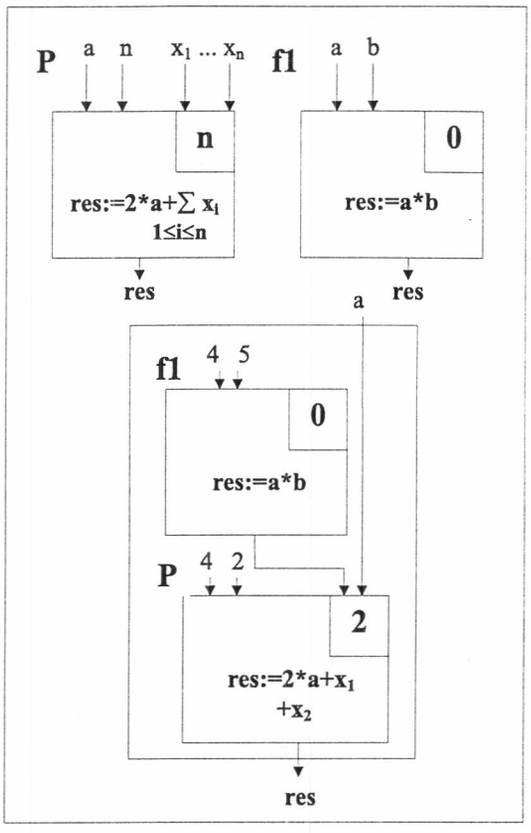

Figure 4.2: Combinación de funciones

- $L_1, \ldots, L_k$ son las variables locales.
- C es el cuerpo de la función.
- R es el resultado.

Por ejemplo, si se define la siguiente función[1]:

```
p y n x1 ...  xn = 2 * y + x1 + ... + xn
f1 a b = a * b
```

entonces la expresion `p 4 2 (f1 4 5)` representa la función parcialmente evaluada que puede verse en la figura 4.2.

Sólo cuando se terminan de instanciar todos los parámetros fijos, en tiempo de ejecución, la expresión E puede evaluarse y puede calcularse la aridad de la presente activación de la función N.

[1]Esto es sólo un pseudocódigo.

4.2 El Lenguaje PARI

El problema que tenía MILONGA era que, por tratarse de un lenguaje construido desde cero, requería la implementación de una librería completa de algoritmos de aritmética, álgebra de polinomios y álgebra lineal desde cero. A fin de evitar una tal implementación, y a fines de tener un banco de pruebas para los algoritmos de eliminación en geometría, se decidió entonces implementar un tipo abstracto straight–line program, que respetara la filosofía de MILONGA, a un lenguaje suficientemente poderoso. Para esta implementación [Bru99], se eligió el lenguaje PARI, ya que posee primitivas muy poderosas de teoría de números.

PARI-GP es un sistema de cálculo simbólico cuyo principal objetivo es facilitar los cálculos de teoría de números. Es *free software* y a partir de la versión 2.1.0 se distribuye bajo los términos de GNU General Public License. Corre en la mayoría de los sistemas operativos comúnmente utilizados. PARI-GP fue originalmente desarrollado en 1987 por un grupo liderado por Henry Cohen en el Laboratorio A2X de la Universidad de Bordeaux y ahora es mantenido por Karim Belabas en la Universidad Paris–Sud Orsay, con la ayuda de muchos ayudantes en forma voluntaria.

El sistema PARI es un paquete de software que puede realizar cálculos formales sobre tipos recursivos a alta velocidad. Si bien apunta principalmente a aplicaciones de teoría de números (factorización de enteros, cálculos sobre curvas elípticas y aplicaciones en teoría de números algebraica), puede ser utilizado en otras aplicaciones que requieran velocidad. Por ejemplo, posee funciones para la manipulación de matrices, polinomios, series, números algebraicos y posee implementaciones de varias funciones especiales. Cabe destacar que si bien PARI posee extensas librerías de programas de cálculo simbólico, no está suficientemente desarrollado en comparación con otros sistemas generalistas de cálculo simbólico como AXIOM, Macsyma, Maple, Mathematica o Reduce (por ejemplo en la manipulación de polinomios multivariados o la integración formal). Por otro lado, las tres principales ventajas de PARI son su velocidad (que puede ser entre 5 y 100 veces mejor en muchos cálculos), la posibilidad de utilizar directamente tipos de datos que son familiares a los matemáticos y el extenso módulo de teoría de números que no tiene equivalente en los sistemas antes mencionados.

Los operadores de PARI se encuentran sobrecargados de manera que, por ejemplo, el mismo operador de suma puede aplicarse tanto a un número entero como a matrices.

PARI puede utilizarse de dos maneras diferentes:

- Como una librería, que puede ser llamada desde otro lenguaje de alto nivel (escrito por ejemplo en C, C++, Pascal o Fortran).

- Como una sofisticada calculadora programable, llamada gp, que contiene la mayoría de las instrucciones de control del tipo de las de un lenguaje standard

como C. Puede ser programada en un lenguaje llamado GP.

En nuestra implementación hemos utilizado PARI 2.1.5[2], que fue extendido con el tipo SLP [Bru99].

4.2.1 Tipos disponibles

En lo que concierne a los tipos, la clave en PARI es la recursividad: la mayoría de sus tipos se construyen recursivamente. Por ejemplo, si bien el tipo básico complex existe, sus componentes pueden ser de cualquier otro tipo. Por supuesto, es necesario utilizar esta característica con cuidado a fin de evitar obtener resultados erróneos.

PARI logra esto considerando todo objeto como tipo GEN en principio, y permitiendo definir cada objeto como un tipo específico dentro de GEN.

Los tipos más utilizados en este trabajo son:

- t_INT: Enteros.
- t_POL: Polinomios.
- t_VEC: Vectores fila.
- t_COL: Vectores columna.
- t_MAT: Matrices.

4.2.2 Operaciones y Funciones

Además de las operaciones aritméticas básicas $+, -, *, /$, PARI provee funciones aritméticas para operar sobre ideales, integrales, discriminantes, etc. Además provee varias funciones para operar con polinomios y series de potencia. El uso de estas funciones se ve ampliamente facilitado por las numerosas funciones de conversión que provee el lenguaje, tanto para conversión entre tipos de PARI como entre tipos de PARI y C en ambos sentidos.

Las operaciones de PARI son muy poderosas al momento de realizar cálculos con matrices y vectores, que son los tipos más utilizados en esta tesis. Por ejemplo, a fin de multiplicar dos matrices A y B basta con la sóla instrucción gmul(A, b).

4.2.3 Memoria

PARI utiliza un stack cuyo tamaño es configurable con la función pari_init. PARI provee diversas instrucciones para realizar *garbage collection*, de manera que en cualquier momento de la programación podemos (con cuidado) liberar el espacio utilizado en el stack, "limpiando" por ejemplo variables intermedias que ya no se utilizan.

[2] http://www.math.u-psud.fr/~belabas/pari.html

4.2.4 Recursos

Además de la información disponible en su página web[3] y los manuales, existen mailing lists y foros que contienen gran cantidad de información y ejemplos de utilización.

4.3 El TAD SLP implementado en PARI

La interfaz que permite utilizar straight–line programs con PARI es una librería (`libSLP.a`) desarrollada por N. Bruno [Bru99]. Esta librería puede incluirse en la compilación por medio de las opciones estándar `include` de C++.

Las funciones que se exportan de este módulo son:

- `setExtras (int ne)` devuelve `ne` como la cantidad de extras.
- `getRemainExtras();` devuelve la cantidad de extras que le faltan al *slp* para saturarse.
- `deleteExtras(int ne)` remueve `ne` cantidad de extras.
- `aplicar (GEN o)` aplica la función `o`.
- `TSLP& apVec (GEN v);` aplica el *slp* al vector de extras `v`.
- `GEN ejecutar()=0;` permite el llamado dentro de un *slp* de la rutina PARI correspondiente.

En este trabajo los distintos tipos de parámetros que recibe un straight–line program se han utilizado de la siguiente forma:

- Parámetros fijos: se han utilizado para definir la dimensión de las matrices que un straight–line program utiliza, y eventualmente determinar la cantidad de parámetros extras del mismo. El valor de estos parámetros se "pasa" en la *primera saturación* del straight–line program de siguiente manera:

  $$\texttt{nombre\_tipo\_slp nombre\_variable}(F_1, \ldots, F_n)$$

  donde `nombre_tipo_slp` es el nombre dado al tipo que construimos y `nombre_variable` define una variable de tipo `nombre_tipo_slp`.

- Parámetros extras: se han utilizado para instanciar las variables de las entradas polinomiales de las matrices construidas por el straight–line program correspondiente. Al "pasarse" estas variables tiene lugar la *segunda saturación* del straight–line program. Estos valores se "pasan" de la siguiente forma:

  $$\texttt{nombre\_variable} \ll X_1 \ll \ldots \ll X_n$$

[3] `http://www.math.u-psud.fr/~belabas/pari.html`

Capítulo 5

Aplicaciones a Sistemas Paramétricos

El estudio de sistemas polinomiales multivariados es un tema central de la matemática computacional, que aparece en relación a numerosos problemas científicos y técnicos (ver [CS98], [Stu02]).

Varias aplicaciones en cálculo simbólico, robótica, modelado molecular, modelado geométrico y visión computacional entre otros, utilizan ecuaciones polinomiales en la representación de sus objetos de estudio y en la definición de restricciones. En consecuencia, varias operaciones geométricas en estas aplicaciones se reducen al estudio de sistemas de ecuaciones polinomiales.

En particular, el tratamiento de sistemas *paramétricos* (es decir, sistemas definidos por ecuaciones polinomiales que dependen de parámetros) resulta particularmente duro, tanto desde el punto de vista teórico ([HMPW98], [CGH$^+$03]) como práctico [GV98].

En la literatura computacional se han propuesto varios algoritmos numéricos y simbólicos (ver [AG90], [CS98], [CLO92]). Los métodos numéricos para sistemas paramétricos (ver [AG90], [Rhe98]) se basan en versiones efectivas del Teorema de la Función Implícita y, si bien son buenos para el análisis local, tienen problemas para dar una idea del comportamiento cualitativo global del espacio solución. Por otro lado, los métodos simbólicos generalmente se basan en técnicas de reescritura (bases de Gröbner o resultantes) que tienen una complejidad doblemente exponencial en el peor caso [Giu84]. Este hecho se refleja en la práctica, donde las implementaciones existentes sólo funcionan bien para "pequeños" sistemas polinomiales (ver [Hof90] por un ejemplo concreto del ámbito de modelado geométrico).

En lo que sigue, vamos a aplicar los algoritmos de álgebra lineal tipo caja negra del Capítulo 3 a fin de desarrollar algoritmos para dos problemas críticos de sistemas paramétricos: la determinación de la dimensión del espacio de parámetros, y la resolución de sistemas de ecuaciones polinomiales de dimensión cero.

5.1 Dimensión del Espacio de Parámetros de un Sistema de Ecuaciones Polinomiales

La descripción del comportamiento *cualitativo* del conjunto de soluciones de un sistema paramétrico es una cuestión de suma importancia. Una parte importante de esta descripción pasa por la determinación de invariantes que representan ciertas características geométricas del conjunto solución del sistema en consideración, tales como la cantidad de "grados de libertad", la cantidad de "componentes", etc.

Sea $V \subset \mathbb{C}^n$ una *variedad algebraica*, es decir, el conjunto solución de un sistema de ecuaciones polinomiales dado. Diremos que V es *irreducible* si V no se puede escribir como una unión no trivial de dos variedades algebraicas. La *dimensión* de V es la longitud r de la máxima cadena de inclusiones $V_0 \subset V_1 \subset \cdots \subset V_r = V$, donde cada V_i es una variedad irreducible, y representa la cantidad de "grados de libertad" en V. Así, la dimensión de un conjunto finito es cero, la de una curva es uno, etc.

Conocer la dimensión de una variedad V dada es un dato clave para su posterior tratamiento, tanto simbólico como numérico. A efectos de su determinación, los algoritmos de cálculo simbólico habituales calculan un sistema de ecuaciones polinomiales que define a V con ciertas características particulares (por ejemplo, una base de Gröbner), a partir del cual la dimensión de V se obtiene en forma inmediata. Como ya hemos señalado anteriormente, tales algoritmos tienen espacio exponencial y tiempo doblemente exponencial. Por otro lado, en [GH93] (ver también [GHL$^+$00]) se propone un algoritmo probabilístico rápido que determina la dimensión de una variedad definida por polinomios *homogéneos*. Este algoritmo reduce el problema a la determinación de la suryectividad de ciertas matrices con entradas polinomiales y por lo tanto puede ser tratado utilizando algoritmos de álgebra lineal de tipo caja negra.

Sean $F_1, \ldots, F_s \in \mathbb{Q}[X_1, \ldots, X_n]$ polinomios homogéneos de grados $d_1, \ldots, d_s$. Elijamos aleatoriamente un subconjunto $\{i_1, \ldots, i_m\} \subset \{1, \ldots, n\}$. A fin de simplificar notaciones, supongamos sin pérdida de generalidad $\{i_1, \ldots, i_m\} = \{1, \ldots, m\}$. Para $1 \leq i \leq s$, definimos $F_i^{(m)} := F_i(X_0X_1, \ldots, X_0X_m, X_{m+1}, \ldots, X_n)$, donde X_0 es una nueva indeterminada. Sea $R^{(m)} := \mathbb{Q}[X_1, \ldots, X_m]$. Observamos que $F_i^{(m)}$ es un polinomio homogéneo en $R^{(m)}[X_0, X_{m+1}, \ldots, X_n]$ (es decir, como polinomio con coeficientes en $R^{(m)}$) de grado d_i.

Para $r \geq 0$, sea $\mathcal{H}_r$ el subconjunto de $R^{(m)}[X_0, X_{m+1}, \ldots, X_n]$ formado por el polinomio nulo y todos los polinomios homogéneos de grado r. Sea $N := 1 + \sum_{i=1}^{s}(d_i - 1)$, y sea

$$\begin{array}{rccl} \phi_m : & \mathcal{H}_{N-d_1} \times \cdots \times \mathcal{H}_{N-d_s} & \to & \mathcal{H}_N \\ & (H_1, \ldots, H_s) & \mapsto & H_1F_1^{(m)} + \cdots + H_sF_s^{(m)} \end{array}$$

Entonce el sistema $F_1 = F_2 = \cdots = F_s = 0$ tiene dimensión m si y sólo si ϕ_m es una función suryectiva y ϕ_{m+1} no lo es ([Laz81], [GH93]).

Teniendo en cuenta que ϕ_m es una función $R^{(m)}$–lineal y la $R^{(m)}$–dimensión de $\mathcal{H} := \mathcal{H}_{N-d_1} \times \cdots \times \mathcal{H}_{N-d_s}$ es mayor que la de $\mathcal{H}_N$, concluimos que ϕ_m es suryectiva si y sólo si la matriz ϕ_m asociada a un par de bases cualesquiera de $\mathcal{H}$ y $\mathcal{H}_N$ es de rango máximo. A fin de testear esta condición, observamos que el conjunto de monomios $\mathcal{B}_r = \{X_0^{\alpha_0} X_{m+1}^{\alpha_1} \cdots X_n^{\alpha_{m-n}} : \alpha_0 + \cdots + \alpha_{m-n} = r\}$ es una base de $\mathcal{H}$ de cardinal $\binom{n-m+r}{r}$. En consecuencia, $\mathcal{B}_{N-d_1} \times \cdots \times \mathcal{B}_{N-d_s}$ es una base de $\mathcal{H}$ y $\mathcal{B}_N$ es una base de $\mathcal{H}_n$. Además, la matriz $\mathcal{M}$ de ϕ_m en las bases $\mathcal{B}, \mathcal{B}_N$ es fácil de determinar: usando la notación $\underline{X} := (X_0, X_{m+1}, \ldots, X_n)$, si $(\underline{X}^{\alpha^{(1)}}, \ldots, \underline{X}^{\alpha^{(s)}})$ es un elemento de $\mathcal{B}$, $\underline{X}^{(\beta)}$ es un elemento de $\mathcal{B}_N$ y $\mathcal{J} := \{i \in \{0, m+1, \ldots, n\} : \beta^{(i)} := \beta - \alpha^{(i)} \in (\mathbb{Z}_{\geq 0})^{n-m+1}\}$, entonces el coeficiente de $\mathcal{M}$ correspondiente a $(\underline{X}^{\alpha^{(1)}}, \ldots, \underline{X}^{\alpha^{(s)}})$ es $\sum_{i \in J} a_i$, donde a_i es el coeficiente de $\underline{X}^{\beta^{(i)}}$ en $F_i^{(m)}$.

Teniendo construida la matriz $\mathcal{M}$, y teniendo en cuenta que sus entradas pertenecen a $R^{(m)} = \mathbb{Q}[X_1, \ldots, X_m]$, concluimos que $\mathcal{M}$ es de rango máximo si y sólo si el determinante $\Delta := \det(\mathcal{M} \cdot \mathcal{M}^t) \in R^{(m)}$ es distinto de cero.

A fin de determinar la validez de esta última condición, aplicando el algoritmo de Samuelson (ver la Sección 3.1.1), el polinomio Δ se calcula con un straight–line program que utiliza espacio $O((nd)^n)$ y tiempo $O((nd)^{4n})$. Una forma ingenua de chequear la identidad $\Delta = 0$ consiste en interpolar Δ a fin de obtener su representación densa. Desafortunadamente, dado que Δ tiene grado menor o igual que $N_\Delta := 2N\binom{n-m+N}{N} = O((nd)^n)$, esta interpolación requeriría $O((dn)^{n^2})$ operaciones aritméticas en $\mathbb{Q}$, o sea una cantidad exponencial en el tamaño de la la representación densa de la entrada. En lugar de esto, vamos a aplicar el test de Zippel–Schwartz ([Zip79], [Sch80]): elegimos aleatoriamente una m–upla $(\gamma_1, \ldots, \gamma_m)$ en el conjunto $\{1, \ldots, 4N_\Delta\}$ y calculamos $\Delta(\gamma_1, \ldots, \gamma_m)$. Entonces, si $\Delta(\gamma_1, \ldots, \gamma_m) = 0$ concluimos que $\Delta = 0$ (obviamente, si $\Delta(\gamma_1, \ldots, \gamma_m) \neq 0$ es claro que $\Delta \neq 0$). La probabilidad de que nuestra respuesta sea errónea se acota por $1/4$, y el test se puede repetir a fin de reducir esta probabilidad de error (esta reducción es exponencial con respecto a la cantidad de testeos realizados [BDG88]).

En conclusión, el algoritmo que hemos descripto cálcula la dimensión de la variedad algebraica definida por los polinomios $F_1, \ldots, F_s$ y utiliza espacio $O((nd)^n)$ y tiempo $O((nd)^{4n})$, es decir, es *polinomial* en la *representación densa* de la entrada. Cabe destacar que el problema de decidir si la dimensión es mayor o igual que -1 (es decir, si la variedad es vacía) es un conocido problema NP–duro y # P–duro ([BCSS98], [Koi96], [HM93], [Mat97]). Nuestro algoritmo se puede describir sintéticamente de la siguiente manera:

Para $m = 0 \ldots n-1$ **Hacer**
  Obtener $F_1^{(m)}, \ldots, F_s^{(m)}$.

Determinar la matriz $\mathcal{M}_m$ de ϕ_m.
Elegir aleatoriamente $\gamma_1, \ldots, \gamma_m \in \{1, \ldots, 4N_{\Delta_m}\}$.
Calcular $\Delta_m(\gamma_1, \ldots, \gamma_m)$.
Si $\Delta_m(\gamma_1, \ldots, \gamma_m) = 0$ **Entonces**
    Parar y devolver como salida m.
Fin Si
Fin Para

5.2 Resolución de Sistemas de Ecuaciones Polinomiales

Supongamos que tenemos un sistema paramétrico tal que todas las instancias paramétricas son variedades algebraicas de dimensión cero. En [GHH+97], [GHM+98] se propone una versión global de la iteración de Newton simbólica que permite describir la variedad algebraica definida por el sistema paramétrico dado a partir de una instancia paramétrica "suave" (ver también [GLS01], [HMW01], [HKP+00], [Sch03], [BMWW03]).

Sea $V \subset \mathbb{C}^{m+n}$ la variedad algebraica definida por un sistema paramétrico de dimensión m que consiste de n ecuaciones $F_1(\underline{T}, \underline{X}), \ldots, F_n(\underline{T}, \underline{X})$ en $n+m$ variables $\underline{T} := (T_1, \ldots, T_m), \underline{X} := (X_1, \ldots, X_n)$. Supongamos que las variables $\underline{T}$ representan los parámetros de nuestro sistema.

Sea $\pi : V \to \mathbb{C}^m$ la proyección canónica sobre el espacio de parámetros, es decir, $\pi(\underline{t}, \underline{x}) := \underline{t}$ para todo $(\underline{t}, \underline{x}) \in \mathbb{C}^{m+n}$. Supongamos que π es *finito*, es decir, $\pi^{-1}(\underline{t})$ es un conjunto finito para todo $\underline{t} \in \mathbb{C}^m$, y no hay ramas que se van "al infinito" (es decir, la imagen de todo subconjunto cerrado y acotado de V es un conjunto acotado).

Supongamos además que existe una hipersuperficie H de $\mathbb{C}^m$ (es decir, el conjunto de ceros de un polinomio en $\mathbb{C}[\underline{T}]$ no nulo) tal que para todo $\underline{t} \in \mathcal{U} := \mathbb{C}^m \setminus H$ todos los puntos de $\pi^{-1}(\underline{t})$ son "suaves", es decir, la matriz Jacobiana de $F_1, \ldots, F_n$ con respecto a $X_1, \ldots, X_n$ es no singular en todos los puntos de $\pi^{-1}(\underline{t})$.

Finalmente, supongamos "dada" una instancia paramétrica $\pi^{-1}(\underline{t_0})$ con $\underline{t} \in \mathcal{U} \cap \mathbb{Q}^m$. Esto significa que tenemos:

1. Una forma lineal $U \in \mathbb{Q}[\underline{X}]$ que "separa" los puntos de $\pi^{-1}(\underline{t_0})$, es decir, que satisface la condición $U(\underline{x}^{(j)}) \neq U(\underline{x}^{(k)})$ para todo par de puntos distintos $(\underline{t_0}, \underline{x}^{(j)}), (\underline{t_0}, \underline{x}^{(k)})$ de $\pi^{-1}(\underline{t_0})$ (una tal forma lineal se denomina un *elemento primitivo* de $\pi^{-1}(\underline{t_0})$).

2. El *polinomio minimal* de U en $\pi^{-1}(\underline{t_0})$, es decir, el polinomio $q := \prod_{i=1}^{\delta} (Y - U(\underline{x}^{(i)}))$ cuyas raíces corresponden a los valores que toma U en los puntos de $\pi^{-1}(\underline{t_0})$.

3. Polinomios $v_1, \ldots, v_n \in \mathbb{Q}[Y]$ que *parametrizan* los puntos de $\pi^{-1}(\underline{t_0})$ por los ceros de q, es decir, que satisfacen las condiciones $q'(U(\underline{x})) \cdot x_j = v_j(U(\underline{x}))$ para todo punto $(\underline{t_0}, \underline{x}) := (\underline{t_0}, x_1, \ldots, x_n)$ de $\pi^{-1}(\underline{t_0})$ y $\mathrm{gr}_Y(v_j) < \delta$.

Los polinomios de los ítems 1, 2 y 3 se denominan una *solución geométrica* de $\pi^{-1}(\underline{t_0})$ (ver [GHM+98], [GHH+97]). Esta noción de solución geométrica tiene una larga historia que se remonta a Kronecker [Kro82]. Se podría considerar a [CG83], [GM89] como las primeras referencias donde esta noción se utilizó implícitamente en cálculo simbólico.

En estas condiciones, aplicando la iteración de Newton de [GHH+97], [GHM+98] (en la versión de [GLS01]), obtenemos aproximaciones a los vectores de series de potencia en $\mathbb{Q}[\underline{T} - \underline{t_0}]^n$ que describen a V. A partir de una aproximación de orden suficientemente grande es posible calcular los polinomios que definen una solución geométrica de V considerando a U como elemento primitivo, es decir:

1. El polinomio minimal $Q \in \mathbb{Q}[\underline{T}, Y]$ de U en V, es decir, el (único) polinomio de grado δ mónico en Y tal que $Q(\underline{T}, U(\underline{X}))$ se anula en todo punto de V.

2. Polinomios $V_1, \ldots, V_n \in \mathbb{Q}[\underline{T}, Y]$ que parametrizan a V por los ceros de Q, es decir, tales que $\frac{\partial Q}{\partial Y}(\underline{T}, U(X))X_j - V_j(\underline{T}, U(X))$ se anula en todo punto de V y $\mathrm{gr}_Y(V_j) < \delta$ para $j = 1, \ldots, n$.

Observamos que $Q(\underline{t_0}, Y) = q(Y)$ y $V_j(\underline{t_0}, Y) = v_j(Y)$ para $j = 1, \ldots, n$. Cabe destacar que a partir de los polinomios $Q, V_1, \ldots, V_n$ se obtiene una solución geométrica de cada instancia paramétrica $\pi^{-1}(\underline{t})$. En efecto, evaluando $Q, V_1, \ldots, V_n$ en $T = \underline{t}$, y limpiando eventualmente los polinomios $Q(\underline{t}, Y), V_1(\underline{t}, Y), \ldots, V_n(\underline{t}, Y)$ de los factores múltiples de $Q(\underline{t}, Y)$ (ver [GLS01] por detalles) se obtiene una descripción de $\pi^{-1}(\underline{t})$.

El algoritmo de [GLS01] procede "levantando" (recursivamente) los polinomios $q, v_1, \ldots, v_n$ a $Q, V_1, \ldots, V_n$. A tal efecto, el $(k+1)$–ésimo paso de iteración de Newton comienza con aproximaciones de orden 2^k de $Q, V_1, \ldots, V_n$, es decir, polinomios $Q^{(k)}, V_1^{(k)}, \ldots, V_n^{(k)} \in \mathbb{Q}[\underline{T}, Y]$ tales que $Q \equiv Q^{(k)} \bmod (\underline{T}^{2^k+1})$ y $V_j \equiv V_j^{(k)} \bmod (\underline{T}^{2^k+1})$ para $j = 1, \ldots, n$ y obtiene aproximaciones de orden 2^{k+1} por aplicación del operador de Newton

$$N_F(\underline{Z})^t = \underline{Z}^t - D_F(\underline{Z})^{-1} F(\underline{Z})^t,$$

donde $\underline{Z} := (Z_1, \ldots, Z_n)$ y D_F representa la matriz Jacobiana de $F := (F_1, \ldots, F_n)$ respecto de $\underline{X}$.

Si los polinomios $F_1, \ldots, F_n$ vienen dados por un straight–line program que utiliza espacio $\mathcal{S}$ y tiempo $\mathcal{T}$, entonces el algoritmo descripto utiliza espacio $O(\mathcal{S}n\delta^2)$ y tiempo (salvo factores polilogarítmicos) $O((n\mathcal{T} + n^\Omega)\delta^2)$, donde $O(n^\Omega)$ representa el costo de evaluar la inversa de la matriz jacobiana D_F. Con las notaciones anteriores, el algoritmo puede describirse sintéticamente de la siguiente manera (ver [GLS01] por su justificación):

$k = 0$, $Q = q$, $\underline{V} = \underline{v}$.
Para $k = 0 \ldots \lceil \log_2 \delta \rceil$ **Hacer**
  $k = 2k$.
  /* *Iteración de Newton*
  $\underline{V} = \underline{V} - D_F(\underline{V})^{-1} F(\underline{V}) \bmod Q$.
  $\Delta = U(\underline{V}) - Y$.
  /* *Actualización de* V *y* Q
  $\underline{V} = \underline{V} - (\frac{\partial V}{\partial Y} \Delta \bmod Q)$.
  $Q = Q - (\frac{\partial Q}{\partial Y} \Delta \bmod Q)$.
Fin Para
Retornar $(Q, \underline{V})$.

Existen fuertes evidencias sobre la optimalidad del factor δ^2. En [HMPW98], [CGH+03] se demuestra que la dependencia en forma al menos lineal de δ es consecuencia de la "robustez algebraica" o la "universalidad" de los algoritmos de eliminación conocidos (por ejemplo, de la "información" codificada en la salida de los mismos). Por otro lado, la presencia del factor δ^2 se debe a la necesidad de introducir un parámetro adicional de "deformación". Por lo tanto, posibles mejoras son solamente factibles en casos particulares donde la inversa de la matriz Jacobiana D_F puede calcularse más rápidamente.

5.2.1 Sistemas Paramétricos y Deformaciones

El algoritmo de la sección anterior puede aplicarse a la resolución de sistemas de ecuaciones polinomiales por medio de deformaciones. La idea es la siguiente: dado un sistema de ecuaciones polinomiales de dimensión cero, se trata de hallar un sistema paramétrico del cual este sistema es una instancia paramétrica. Si el sistema paramétrico hallado satisface las condiciones de la sección anterior y tiene una instancia paramétrica "fácil", entonces el algoritmo descripto calcula eficientemente la solución del sistema original.

En las siguientes secciones vamos a ver dos familias de ejemplos donde la combinación de nuestros algoritmos de álgebra lineal tipo caja negra con esta técnica permite mejorar el costo del algoritmo generalista de [GLS01].

Sistemas de Wright

Sea $n \in \mathbb{N}$ fijo. Un sistema de Wright ([KM83], [Wri85], [Zul88]) es el definido por polinomios $F_1, \ldots, F_n \in \mathbb{Q}[X_1, \ldots, X_n]$ del tipo:

$$F_i = X_i^2 - 2X_i + \alpha_i + \sum_{j=1}^{n} X_j, \quad (1 \leq i \leq n) \tag{5.1}$$

donde $\alpha_1, \ldots, \alpha_n$ son racionales no nulos fijos.

Supongamos fijos $\alpha_1, \ldots, \alpha_n \in \mathbb{Q} \setminus \{0\}$. A fin de desarrollar nuestra técnica de deformación necesitamos la terminología sobre bases de Gröbner que introducimos a continuación (cf. [CLO92], [BW93]).

Un *orden monomial* $>$ en $\mathbb{Q}[X_1, \ldots, X_n]$ es un buen orden sobre el conjunto los monomios $\{X^m := X^{m_1} \cdots X_n^{m_n} : m_1 \geq 0, \ldots, m_n \geq 0\}$ que es compatible con el producto, es decir, tal que $X^{m^{(1)}} > X^{m^{(2)}}$ implica $X^{m^{(1)}+m} > X^{m^{(2)}+m}$ para todo $m \in (\mathbb{Z}_{\geq 0})^n$. Para $m := (m_1, \ldots, m_n) \in (\mathbb{Z}_{\geq 0})^n$ definimos $|m| := \sum_{j=1}^n m_j$. En lo que sigue vamos a utilizar el *orden graduado lexicográfico directo* $>$ según el cual $X^{m^{(1)}} > X^{m^{(2)}}$ sii $|m^{(1)}| > |m^{(2)}|$ o $|m^{(1)}| = |m^{(2)}|$ y $X^{m^{(1)}} > X^{m^{(2)}}$ según el orden lexicográfico definido por $X_1 > \cdots > X_n$ [CLO92].

Sea $>$ un orden monomial. Para cada $F \in \mathbb{Q}[x_1, \ldots, x_n]$ definimos el *monomio inicial* $in_>(F)$ de F con respecto a $>$ como el monomio de mayor orden que aparece en F con coeficiente no nulo.

El *ideal* I generado por polinomios $H_1, \ldots, H_s$ de $\mathbb{Q}[X_1, \ldots, X_n]$ es el conjunto de polinomios que se escriben como combinación lineal de $H_1, \ldots, H_n$ con coeficientes en $\mathbb{Q}[x_1, \ldots, x_n]$, es decir:

$$I := \{F \in \mathbb{Q}[x_1, \ldots, x_n] : (\exists P_1) \cdots (\exists P_s) F = P_1 H_1 + \cdots + P_s H_s\}.$$

Los polinomios $H_1, \ldots, H_s$ se denominan un sistema de generadores de I. Dado un ideal $I \subset \mathbb{Q}[x_1, \ldots, x_n]$ y un orden monomial $>$, definimos el *ideal inicial* $in_>(I)$ de I con respecto a $>$ como el ideal generado por todos los monomios iniciales $in_>(F)$ de los polinomios de I. Una *base de Gröbner* de un ideal I es un sistema de generadores $F_1, \ldots, F_s$ de I tal que $in_>(F_1), \ldots, in_>(F_s)$ genera el ideal $in_>(I)$. Una base de Gröbner $H_1, \ldots, H_s$ se dice reducida sii $in_>(H_i)$ no divide a $in_>(H_j)$ para todo $i \neq j$.

Lema 11 *Los polinomios $F_1, \ldots, F_n$ de (5.1) forman una base de Gröbner reducida (del ideal que generan) según el orden graduado lexicográfico directo y definen una variedad algebraica de dimensión cero con a lo sumo 2^n soluciones complejas.*

Demostración: Observamos que $in_>(F_i) = X_i^2$ para $1 \leq i \leq n$, dado que éste es el único monomio de grado 2 de F_i (y todos los otros monomios tienen grado menor). Por lo tanto, los monomios $in_>(F_1), \ldots, in_>(F_n)$ son coprimos entre sí. Del primer criterio de Buchberger [BW93] se deduce que $F_1, \ldots, F_n$ forman una base de Gröbner reducida.

Por último, demostramos que la variedad algebraica definida por $F_1, \ldots, F_n$ es de dimensión cero y tiene a lo sumo 2^n soluciones complejas. Observamos que existen exactamente 2^n monomios que no son divisibles por algunos de los monomios $in_>(F_1), \ldots, in_>(F_n)$: los monomios en los cuales todas las variables aparecen con grado a lo sumo 1. De esto se deduce que el sistema $F_1 = F_2 = \cdots = F_n = 0$ tiene a lo sumo 2^n soluciones complejas [CLO92]. ∎

A fin de resolver el sistema definido en (5.1) introducimos un sistema paramétrico del cual el conjunto de soluciones de $F_1 = F_2 = \cdots = F_n = 0$ constituye una instancia particular. Para esto, introducimos una nueva variable (parámetro) $\mathcal{E}$ y definimos los siguientes polinomios de $\mathbb{Q}[\mathcal{E}, X_1, \ldots, X_n]$:

$$G_i := X_i^2 - X_i + \mathcal{E}\Big(\alpha_i - X_i + \sum_{j=1}^{n} X_j\Big) \quad (1 \leq i \leq n).$$

Observamos que $G_i(1, \underline{X}) = F_i$ para $1 \leq i \leq n$ (es decir, los polinomios $G_1, \ldots, G_n$ "deforman" $F_1, \ldots, F_n$ a un sistema paramétrico). A fin de describir el comportamiento de este último sistema, tenemos el siguiente resultado:

Lema 12 *Para todo $\varepsilon \in \mathbb{C}$, los polinomios $G_1(\varepsilon, \underline{X}), \ldots, G_n(\varepsilon, \underline{X})$ forman una base de Gröbner reducida respecto del orden graduado lexicográfico directo y definen una variedad algebraica de dimensión cero con a lo sumo 2^n soluciones complejas.*

Demostración: Sea $\varepsilon \in \mathbb{C}$ fijo. Teniendo en cuenta que $in_>(G_i(\varepsilon, \underline{X})) = X_i^2$ para $1 \leq i \leq n$, con los argumentos de la demostración del Lema 11 deducimos el enunciado. ∎

Sea $V \subset \mathbb{C}^{n+1}$ la variedad algebraica definida por los polinomios $G_1, \ldots, G_n$, y sea $\pi : V \to \mathbb{C}$ la proyección $\pi(\varepsilon, \underline{x}) := \varepsilon$. Como hemos señalado, nuestra intención es calcular una solución geométrica del sistema de Wright $\pi^{-1}(1)$. Para esto, queremos aplicar la iteración de Newton simbólica de la Sección 5.2 a partir de una solución geométrica de $\pi^{-1}(0)$. Por lo tanto, es necesario demostrar que los polinomios $G_1, \ldots, G_n$ satisfacen las condiciones requeridas para la aplicación de la iteración de Newton.

En primer lugar, observamos que $G_1, \ldots, G_n$ forman una base de Gröbner para el orden graduado lexicográfico de $\mathbb{Q}[\mathcal{E}, \underline{X}]$ definido por $X_1 > \ldots > X_n > \mathcal{E}$. Dado que $in_>(G_i) = X_i^2$ según este orden para $1 \leq i \leq n$, vemos que los monomios que no son divisibles por ninguno de los monomio $in_>(G_1), \ldots, in_>(G_n)$ son de la forma $\mathcal{E}^\beta \underline{X}^\alpha$ con $\alpha \in \{0,1\}^n$. Esto implica que el morfismo π es finito (ver por ejemplo [Sha94, §I.5.3]).

Por otro lado, el Lema 12 demuestra que $\pi^{-1}(\varepsilon)$ tiene a lo sumo 2^n puntos para todo $\varepsilon \in \mathbb{C}$. Además, de la expresión explícita de $G_1(0, \underline{X}), \ldots, G_n(0, \underline{X})$ fácilmente deducimos que $\pi^{-1}(0)$ tiene exactamente 2^n soluciones. Por lo tanto, [Sha94, Theorem II.6.3.4] demuestra que $\pi^{-1}(\varepsilon)$ tiene exactamente 2^n puntos para todo $\varepsilon \in \mathbb{C}$ salvo finitos valores. Combinando esta observación con el Lema 12, de [CLO98, §4.2, Corollary 2.5] deducimos que para todo $\varepsilon \in \mathbb{C}$ salvo finitos valores, los puntos de $\pi^{-1}(\varepsilon)$ son suaves.

Estamos entonces en condiciones de aplicar la iteración de Newton simbólica de la Sección 5.2. Para esto necesitamos una solución geométrica de $\pi^{-1}(0)$, es decir,

del conjunto solución del sistema $G_1(0,\underline{X}) = \cdots = G_n(0,\underline{X}) = 0$. Observamos que $x \in \mathbb{C}^n$ es una solución de $G_1(0,\underline{X}) = \cdots = G_n(0,\underline{X}) = 0$ si y sólo si $x_i \in \{0,1\}$ para $1 \leq i \leq n$. Sea $U := \sum_{i=1}^{n} 2^{i-1} X_i$. Entonces el polinomio minimal de U es $q := \prod_{j=0}^{2^n-1}(Y-j)$. Los polinomios $v_1,\ldots,v_n$ que parametrizan las variables $X_1,\ldots,X_n$ por los ceros de q se obtienen por interpolación con $O(n2^n \log n)$ operaciones aritméticas (cf. [BP94]). Así obtenemos la entrada necesaria para aplicar la iteración de Newton.

A fin de estudiar el costo de la iteración de Newton, observamos que los polinomios $G_1,\ldots,G_n$ se calculan por un straight–line program con espacio $O(n)$ y tiempo $O(n)$, y las entradas de la matriz Jacobiana D_G de $G_1,\ldots,G_n$ con respecto a $X_1,\ldots,X_n$ se calculan con espacio $O(1)$ y tiempo $O(n)$. De hecho,

$$D_G = \begin{pmatrix} 2X_1 - 1 - \mathcal{E} & & \mathcal{E} & \cdots & \mathcal{E} \\ & \mathcal{E} & 2X_2 - 1 - \mathcal{E} & \ddots & \vdots \\ & \vdots & \ddots & \ddots & \mathcal{E} \\ & \mathcal{E} & \cdots & \mathcal{E} & 2X_n - 1 - \mathcal{E} \end{pmatrix}.$$

De la expresión explícita de D_G deducimos que, dados $(\varepsilon,\underline{x}) := (\varepsilon,x_1,\ldots,x_n) \in \mathbb{C}^{n+1}$ y $w \in \mathbb{C}^n$ arbitrarios, el producto $D_G(\varepsilon,\underline{x}) \cdot w$ se calcula con espacio $O(1)$ y tiempo $O(n)$. Por lo tanto, aplicando el algoritmo de Wiedemann (ver la Sección 3.2.1) deducimos que la inversa de la matriz D_G se calcula con espacio $O(n)$ y tiempo $O(n^2)$. Finalmente, aplicando la iteración de Newton simbólica de la Sección 5.2 obtenemos un algoritmo que resuelve un sistema de Wright dado con espacio $O(n2^{2n})$ y tiempo $O(n^4 2^{2n} \log n)$.

Cabe destacar que los algoritmos que utilizan la representación densa de polinomios multivariados (bases de Gröbner, resultantes) tienen complejidad (del peor caso) de orden $\Omega(2^{n^2})$. Por otro lado, mediante una aplicación directa del algoritmo de [GLS01] obtenemos un algoritmo que requiere tiempo $O(n^6 2^{2n} \log n)$. En consecuencia, nuestro algoritmo mejora la complejidad asintótica de ambos.

Discretizaciones de la Ecuación del Calor

En esta sección vamos a tratar una familia de sistemas que proviene de problemas de discretización de la ecuación del calor unidimensional con términos de absorción no lineales (polinomiales) y términos de reacción en el borde [Pao92].

En la literatura del análisis numérico para ecuaciones diferenciales ha sido frecuentemente estudiado el comportamiento asintótico de una discretización en relación con el de la ecuación original ([CFQ91], [Lev90], [FF96]). Este comportamiento viene descripto fundamentalmente por las soluciones estacionarias de la discretización en consideración, que son aquellas que no varían con el tiempo [Hen81]. En el caso de la ecuación del calor, si consideramos una discretización de segundo orden en espacio con una malla uniforme, manteniendo la variable temporal continua, las soluciones

estacionarias son las soluciones reales positivas de los sistemas de ecuaciones polinomiales que vamos a analizar a continuación ([BR01], [FGR02], [DM02]).

Sean $d, n \geq 2$ fijos. El tipo de sistemas que vamos a considerar está definido por polinomios $F_1, \ldots, F_n \in \mathbb{Q}[\underline{X}] := \mathbb{Q}[X_1, \ldots, X_n]$ de la siguiente forma:

$$\begin{array}{rcll} F_1 & := & 2(n-1)^2(X_2 - X_1) - X_1^d, & \\ F_i & := & (n-1)^2(X_{i+1} - 2X_i + X_{i-1}) - X_i^d, & (2 \leq i \leq n-1) \\ F_n & := & 2(n-1)^2(X_{n-1} - X_n) + 2(n-1)\alpha - X_n^d. & \end{array} \tag{5.2}$$

A fin de resolver el sistema $F_1 = \cdots = F_n = 0$, vamos a definir un sistema paramétrico en donde éste aparece como una instancia paramétrica particular. Para esto, introducimos una nueva indeterminada $\mathcal{E}$ y definimos los siguientes polinomios de $\mathbb{Q}[\mathcal{E}, \underline{X}]$:

$$\begin{array}{rcl} G_1 & := & 2(n-1)^2\mathcal{E}(X_2 - X_1) + (\mathcal{E}-1)p(X_1) - X_1^d \\ G_i & := & (n-1)^2\mathcal{E}(X_{i+1} - 2X_i + X_{i-1}) + (\mathcal{E}-1)p(X_i) - X_i^d, \quad (2 \leq i \leq n-1) \\ G_n & := & 2(n-1)^2\mathcal{E}(X_{n-1} - X_n) + 2(n-1)\alpha\mathcal{E} + (\mathcal{E}-1)p(X_n) - X_n^d, \end{array}$$

donde $p := \prod_{j=0}^{d-1}(Z - j) - Z^d$.

Observamos que $G_i(1, \underline{X}) = F_i$ para $1 \leq i \leq n$. A fin de describir el comportamiento de este último sistema, tenemos el siguiente resultado:

Lema 13 *Para todo $\varepsilon \in \mathbb{C}$, los polinomios $G_1(\varepsilon, \underline{X}), \ldots, G_n(\varepsilon, \underline{X})$ forman una base de Gröbner y definen una variedad algebraica de dimensión cero con a lo sumo d^n soluciones complejas.*

Demostración: Sea $\varepsilon \in \mathbb{C}$ fijo, y sea $>$ el orden graduado lexicográfico directo de $\mathbb{C}[X_1, \ldots, X_n]$. Observamos que $in_>(G_i(\varepsilon, \underline{X})) = X_i^d$ para $1 \leq i \leq n$. Por lo tanto, $in_>(G_1(\varepsilon, \underline{X})), \ldots, in_>(G_n(\varepsilon, \underline{X}))$ son coprimos entre sí, y aplicando el primer criterio de Buchberger [BW93] vemos que $G_1(\varepsilon, \underline{X}), \ldots, G_n(\varepsilon, \underline{X})$ forman una base de Gröbner.

Demostramos ahora la segunda afirmación del enunciado. Teniendo en cuenta que $in_>(G_i(\varepsilon, \underline{X})) = X_i^d$ para $1 \leq i \leq n$, concluimos que existen exactamente d^n monomios no divisibles por alguno de los monomios $in_>(G_1(\varepsilon, \underline{X})), \ldots, in_>(G_n(\varepsilon, \underline{X}))$: aquellos en los cuales todas las variables aparecen con grado a lo sumo $d-1$. Deducimos entonces que el sistema $G_1(\varepsilon, \underline{X}) = \cdots = G_n(\varepsilon, \underline{X}) = 0$ tiene a lo sumo d^n soluciones complejas. ∎

Sea $V \subset \mathbb{C}^{n+1}$ la variedad algebraica definida por los polinomios $G_1, \ldots, G_n$, y sea $\pi : V \to \mathbb{C}$ la proyección $\pi(\varepsilon, \underline{x}) := \varepsilon$. Como hemos señalado, nuestra intención es calcular una solución geométrica del sistema $\pi^{-1}(1)$. Para esto, queremos aplicar la iteración de Newton simbólica de la Sección 5.2 a partir de una solución geométrica

de $\pi^{-1}(0)$. Por lo tanto, es necesario demostrar que los polinomios $G_1, \ldots, G_n$ satisfacen las condiciones requeridas para la aplicación de la iteración de Newton.

En primer lugar, observamos que $G_1, \ldots, G_n$ forman una base de Gröbner para el orden graduado lexicográfico de $\mathbb{Q}[\mathcal{E}, \underline{X}]$ definido por $X_1 > \ldots > X_n > \mathcal{E}$. Dado que $in_>(G_i) = X_i^d$ según este orden para $1 \leq i \leq n$, vemos que los monomios que no son divisibles por ninguno de los monomios $in_>(G_1), \ldots, in_>(G_n)$ son de la forma $\mathcal{E}^\beta \underline{X}^\alpha$ con $\alpha \in \{0, \ldots, d-1\}^n$. Esto implica que el morfismo π es finito (ver por ejemplo [Sha94, §I.5.3]).

Por otro lado, el Lema 13 demuestra que $\pi^{-1}(\varepsilon)$ tiene a lo sumo d^n puntos para todo $\varepsilon \in \mathbb{C}$. Además, de la expresión explícita de $G_1(0, \underline{X}), \ldots, G_n(0, \underline{X})$ fácilmente deducimos que $\pi^{-1}(0)$ tiene exactamente d^n soluciones. Por lo tanto, [Sha94, Theorem II.6.3.4] demuestra que $\pi^{-1}(\varepsilon)$ tiene exactamente d^n puntos para todo $\varepsilon \in \mathbb{C}$ salvo finitos valores. Combinando esta observación con el Lema 12, de [CLO98, §4.2, Corollary 2.5] deducimos que para todo $\varepsilon \in \mathbb{C}$ salvo finitos valores, los puntos de $\pi^{-1}(\varepsilon)$ son suaves.

Estamos entonces en condiciones de aplicar la iteración de Newton simbólica de la Sección 5.2. Para esto necesitamos una solución geométrica de $\pi^{-1}(0)$, es decir, del conjunto solución del sistema $G_1(0, \underline{X}) = \cdots = G_n(0, \underline{X}) = 0$. Observamos que $\underline{x} := (x_1, \ldots, x_n) \in \mathbb{C}^n$ es solución de este sistema si y sólo si $G_i(0, \underline{x}) = -x_i^d - p(x_i) = \prod_{j=0}^{d-1}(x_i - j) = 0$, es decir si y sólo si $x_i \in \{0, \ldots, d-1\}$ para $1 \leq i \leq n$. Sea $U := \sum_{j=1}^n d^{j-1} X_j$. Entonces el polinomio minimal de U es $q := \prod_{j=0}^{d^n-1}(Z - j)$, y las parametrizaciones $v_1, \ldots, v_n$ de las variables $X_1, \ldots, X_n$ por los ceros de q se obtienen por interpolación con espacio $O(d^n)$ y tiempo $O(nd^n \log d \log\log d)$ (cf. [BP94]).

Por último, observamos que los polinomios $G_1, \ldots, G_n$ se calculan con espacio $O(n)$ y tiempo $O(nd)$, y la matriz jacobiana D_G de $G_1, \ldots, G_n$ con respecto a $X_1, \ldots, X_n$ se calcula con complejidades del mismo orden. Explícitamente, tenemos:

$$D_G := \begin{pmatrix} f(\mathcal{E}, X_1) & 2(n-1)^2\mathcal{E} & & \\ (n-1)^2\mathcal{E} & f(\mathcal{E}, X_2) & (n-1)^2\mathcal{E} & \\ \ddots & \ddots & \ddots & \\ & (n-1)^2\mathcal{E} & f(\mathcal{E}, X_{n-1}) & (n-1)^2\mathcal{E} \\ & & 2(n-1)^2\mathcal{E} & f(\mathcal{E}, X_n) \end{pmatrix},$$

siendo $f(\mathcal{E}, Z) := -2(n-1)^2\mathcal{E} + (\mathcal{E}-1)p'(Z) - dZ^{d-1}$. En particular, vemos que D_G es tridiagonal y, por lo tanto, dados $(\varepsilon, \underline{x}) \in \mathbb{C}^{n+1}$ y $\underline{w} \in \mathbb{C}^n$ arbitrarios, el producto $D_G(\varepsilon, \underline{x}) \cdot \underline{w}$ se calcula con espacio $O(1)$ y tiempo $O(n)$. Por lo tanto, aplicando el algoritmo de Wiedemann (ver la Sección 3.2.1) deducimos que la inversa de la matriz D_G se calcula con espacio $O(n)$ y tiempo $O(n^2)$. Finalmente, aplicando la iteración de Newton simbólica de la Sección 5.2 obtenemos un algoritmo que resuelve un sistema de tipo (5.2) dado con espacio $O(nd^{2n})$ y tiempo $O(n^2(d+n^2)d^{2n} \log dn)$.

Cabe destacar que este algoritmo tiene mejor comportamiento asintótico que los basados en técnicas de reescritura y el algoritmo de [GLS01] (comparar con los comentarios al final de la Sección 5.2.1).

Capítulo 6

Algoritmos Implementados y Resultados Comparativos

En este capítulo presentamos una parte del código de los algoritmos implementados (la que corresponde fundamentalmente a su definición en la librería SLP), y resultados comparativos. Las implementaciones fueron realizadas en un entorno `Linux Redhat 7.3`, `PARI 2.1.5` en una plataforma Intel con las siguientes características:

- Procesador 850 Mhz,
- 128 Mb memoria RAM.

A fin de medir los tiempos en el entorno PARI extendido con SLP, hemos utilizado la función de tiempo `time` disponible en C++ y la función de tiempo de `Maple`. Los resultados de estas mediciones se expresan en segundos.

La diferencia de desempeño entre `Maple` y PARI extendido con la librería SLP ilustra las enormes ventajas que pueden obtenerse a partir de la introducción de estructuras de datos alternativas, adaptadas al problema en consideración. En efecto, la representación straight–line program que utilizamos en PARI extendido con SLP es evidentemente superior a la representación densa que utiliza `Maple`.

6.1 Polinomio Característico con el Método de Samuelson

En esta sección describimos nuestra implementación en PARI extendido con SLP del algoritmo de la Sección 3.1.1 para el cálculo del polinomio característico de una matriz con entradas en un anillo de polinomios $\mathbb{Q}[X_1, \ldots, X_n]$. A continuación presentamos el código de este algoritmo:

```
BEGIN_SLP(PCARSAMSLP)
        TSLP_pointer alfa;
```

```
        public:
        PCARSAMSLP(TSLP_pointer alfa0)  {
        alfa=alfa0;
        setExtras(alfa0->getRemainExtras());
        }

END_SLP


GEN PCARSAMSLP::ejecutar() {
GEN salida;


    salida=polcarsaml((GEN) alfa->apVec(x));

return(salida);

}
```

A fin de testear el desempeño de nuestra implementación, calculamos el polinomio característico de una sucesión de matrices A cuyas entradas son polinomios en 3 variables de grado a lo sumo 5 con coeficientes enteros elegidos aleatoriamente en $[-99, 99]$ y a lo sumo 6 términos no nulos. Los parámetros extra de la función correspondiente son los valores de las 3 variables de los polinomios que aparecen como entradas de la matriz A. La siguiente tabla muestra los tiempos de ejecución de nuestra implementación del algoritmo de la Sección 3.1.1 y de la función `charpoly` de `Maple`.

Table 6.1: Polinomio característico de matrices polinomiales.

Dimensión	PARI extendido	Maple
2	0.00	0.00
4	0.00	0.00
6	0.00	1.00
10	0.00	1.02
20	0.00	6.00
30	0.01	34.00
50	0.07	486.00

6.2 Polinomio Minimal con el Método de Wiedemann

En esta sección describimos nuestra implementación en PARI extendido con SLP del algoritmo de la Sección 3.2.3 para el cálculo del polinomio minimal de una matriz con entradas en un anillo de polinomios $\mathbb{Q}[X_1, \ldots, X_n]$. Este algoritmo se aplica a matrices Vandermonde (ver la Sección 3.5):

$$V := (v_{i,j})_{1 \leq i,j \leq n} := \begin{pmatrix} 1 & X_1 & \cdots & X_1^n \\ 1 & X_2 & \cdots & X_2^n \\ \vdots & \vdots & & \vdots \\ 1 & X_n & \cdots & X_{n-1}^n \end{pmatrix},$$

donde $X_1, \ldots, X_n$ representan parámetros extra. Cada entrada $v_{i,j}$ se genera de la siguiente manera:

$$v_{i,j} = \begin{cases} 1 & \text{para } j = 1, \\ X_i \cdot v_{i,j-1} & \text{para } j = 2, \ldots, n. \end{cases}$$

A continuación presentamos el código de este algoritmo:

```
BEGIN_SLP(POLMINSLP)
       TSLP_pointer alfa;

       public:
       POLMINSLP(TSLP_pointer alfa0) {
       alfa=alfa0;
       setExtras(alfa0->getRemainExtras());

       }

END_SLP

GEN POLMINSLP::ejecutar(){
 GEN salida;

        salida=polmin((GEN) alfa->apVec(x));

        return(salida);

       }
```

A fin de testear el desempeño de nuestra implementación calculamos el polinomio minimal de una sucesión de matrices A cuyas entradas son polinomios en 3 variables de grado a lo sumo 5 con coeficientes enteros elegidos aleatoriamente en $[-99, 99]$ y a lo sumo 6 términos no nulos. Los parámetros extra de la función correspondiente son los valores de las 3 variables de los polinomios que aparecen como entradas de la matriz A. La siguiente tabla muestra los tiempos de ejecución de nuestra implementación del algoritmo de la Sección 3.2.3 y de la función `minpoly` de `Maple`.

Table 6.2: Polinomio minimal de matrices Vandermonde.

Dimensión	PARI extendido	Maple
2	0.00	0.00
4	0.00	0.00
6	0.00	0.00
8	0.01	1.00
10	0.02	1.00
15	0.09	15.00

6.3 Cálculo de la dimensión de una variedad

En esta sección describimos nuestra implementación en `PARI` extendido con `SLP` del algoritmo de la Sección 5.1 para el cálculo de la dimensión del espacio de parámetros de un sistema de ecuaciones polinomiales. A continuación presentamos el código de este algoritmo:

```
BEGIN_SLP(func_slp)
long cantecuac,cantvbles;

public:
func_slp(long cantecuac0, long cantvbles0) {
  cantecuac=cantecuac0;
  cantvbles=cantvbles0;
  setExtras(cantvbles);
}
END_SLP
```

```
GEN func_slp::ejecutar()  {
GEN vecresul=cgetg(cantecuac+1,t_VEC);

        vecresul[1]=(long) \
        gadd( \
          gadd( \
            gmul( \
              gmul(gdeux,gpowgs((GEN)x[1],2)), \
              (GEN)x[2] \
            ), \
            gmul( \
              gmul(stoi(3),(GEN)x[1]), \
              gpowgs((GEN)x[2],2) \
            ) \
          ), \
          gpowgs((GEN)x[2],3) \
        );

            vecresul[2]=(long) gadd(gsub( gmul(gneg(gdeux),gpowgs((GEN)x[1],2)),
                                  gmul((GEN)x[1],(GEN)x[2]) ), gmul(stoi(5),
                                  gpowgs((GEN)x[2],2)));

......
......
for (i=supraindice+1;i<=cantvbles;i++) {
  grados[i]=poldegree(mon,varn((GEN)x[i]));

......
.....
printf("%s%s\n","Matriz final es ",GENtostr(matfin));
} /* fin supraindices */

return();

} /* fin ejecutar */
```

A fin de testear el desempeño de nuestra implementación calculamos la dimensión de sistemas de ecuaciones polinomiales definidos por dos polinomios en dos variables cuyos grados se indican.

Table 6.3: Algoritmo de cálculo de la dimensión.

Grados	PARI extendido	Maple
15 y 10	0.00	1.00
30 y 20	0.00	2.00
60 y 40	0.00	16.00

Capítulo 7

Conclusiones

Como hemos señalado anteriormente, una amplia gama de aplicaciones requiere la resolución de sistemas de ecuaciones lineales y no lineales paramétricos. Éste es un problema particularmente duro desde el punto de vista computacional, que exige el desarrollo de herramientas teóricas y prácticas para resolverlo.

Los paquetes de software actualmente existentes para este tipo de cálculos (`Maple`, `AXIOM`, `Mathematica`, etc.) aplican técnicas de reescritura, basadas en la representación densa de polinomios multivariados, que resultan sumamente ineficientes en la práctica (ver los resultados comparativos del Capítulo 6). Por el contrario, combinando algoritmos *seminuméricos* de álgebra lineal y eliminación en geometría con la representación straight–line program de polinomios multivariados, en esta tesis hemos obtenido una gran mejora en el tiempo de cálculo y en el espacio de memoria necesarios para resolver ciertos problemas paramétricos particulares.

Desde el punto de vista teórico, la combinación de la versión adaptada de algoritmos de álgebra lineal de tipo caja negra del Capítulo 3 con las técnicas de eliminación seminuméricas del Capítulo 5 nos ha permitido mejorar la complejidad asintótica de los más eficientes algoritmos conocidos para la resolución de ciertos problemas paramétricos particulares. Asimismo, cabe destacar que el paradigma de álgebra lineal que hemos utilizado parece ser sumamente flexible, y por lo tanto aplicable a una amplia gama de problemas de eliminación "estructurados".

Desde el punto de vista práctico, en nuestra implementación hemos sacado provecho del hecho que los algoritmos seminuméricos que hemos utilizado y desarrollado no requieren el almacenamiento de las funciones con las que operan, sino mas bien su evaluación. Así hemos evitado la representación explícita del grafo de los straight–line programs correspondientes. Por otro lado, la utilización de `PARI` nos ha provisto múltiples herramientas para manipulación de matrices y polinomios.

En los resultados comparativos del Capítulo 6 hemos ilustrado la diferencia de desempeño entre `Maple` y `PARI` extendido con la librería `SLP`. Así se aprecian las ventajas en tiempo de cálculo que se obtienen a partir de la utilización de estructuras

de datos alternativas: en los ejemplos del Capítulo 6, `Maple` muestra un comportamiento exponencial, en tanto que nuestra implementación muestra un comportamiento de tipo cuadrático o cúbico.

Por último, cabe destacar que nuestro enfoque admite aún posteriores optimizaciones. En particular, queda pendiente el tratamiento de los enteros que aparecen como resultados intermedios por medio de técnicas modulares. Tanto en el caso del álgebra lineal [Dix82], como en el de la eliminación en geometría [Tri85], la introducción de tales técnicas ha permitido importantes mejoras de los tiempos de cálculo en la práctica.

Referencias

[Abd97] J. Abdeljaoued. The Berkowitz algorithm, Maple and computing the characteristic polynomial in an arbitrary commutative ring. *Computer Algebra: The Maple Technical Newsletter*, 4(3):21–32, 1997.

[AG90] E.L. Allgower and K. Georg. *Numerical continuation methods: An Introduction*, volume 13 of *Series in Computational Mathematics*. Springer Verlag, Heidelberg, 1990.

[BCS97] P. Bürgisser, M. Clausen, and M.A. Shokrollahi. *Algebraic Complexity Theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften*. Springer, Berlin Heidelberg New York, 1997.

[BCSS98] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer, New York Berlin Heidelberg, 1998.

[BDG88] J.L. Balcázar, J. Díaz, and J. Gabarró. *Structural complexity I*, volume 11 of *EATCS*. Springer, 1988.

[Ber84] S.J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18:147–150, 1984.

[BHMW02] N. Bruno, J. Heintz, G. Matera, and R. Wachenchauzer. Functional programming concepts and straight–line programs in computer algebra. *Mathematics and Computers in Simulation*, 60(6):423–463, 2002.

[BMWW03] A. Bompadre, G. Matera, R. Wachenchauzer, and A. Waissbein. Polynomial equation solving by lifting procedures for ramified fibers. To appear in Theoretical Computer Science, 2003.

[Bor93] A. Borodin. Time space tradeoffs (getting closer to the barriers?). In *4th International Symposium on Algorithms and Computation, ISAAC '93, Hong Kong, December 15-17, 1993*, volume 762 of *Lecture Notes in Computer Science*, pages 209–220, Berlin Heidelberg New York, 1993. Springer.

[BP94] D. Bini and V. Pan. *Polynomial and matrix computations*. Progress in Theoretical Computer Science. Birkhäuser, Boston, 1994.

[BR01] J. Fernandez Bonder and J. Rossi. Blow-up vs. spurious steady solutions. *Proceedings of the American Mathematical Society*, 129(1):139–144, 2001.

[Bro87] D.W. Brownawell. Bounds for the degree in the Nullstellensatz. *Annals of Mathematics*, 126:577–591, 1987.

[Bru98] N. Bruno. Esquemas de compilación de circuitos aritméticos uniformes representados por medio de funciones generatrices. Master's thesis, FaMAF, Universidad de Córdoba, Argentina, 1998.

[Bru99] N. Bruno. La clase `tslp` extiende a `pari/gp`. Comunicación personal, 1999.

[Buc85] B. Buchberger. Gröbner bases: An algoritmic method in polynomial ideal theory. In N. K. Bose et al, editor, *Multidimensional System Theory*, pages 374–383. Reidel, Dordrecht, 1985.

[BW93] T. Becker and V. Weispfenning. *Gröbner bases. A computational approach to commutative algebra*, volume 141 of *Graduate Texts in Mathematics*. Springer, Berlin Heidelberg New York, 1993.

[CFQ91] M. Chipot, M. Fila, and P. Quittner. Stationary solutions, blow up and convergence to stationary solutions for semilinear parabolic equations with nonlinear boundary conditions. *Acta Mathematica Universitatis Comenianae*, 60(1):35–103, 1991.

[CG83] A.L. Chistov and D.Y. Grigoriev. Subexponential time solving systems of algebraic equations. I, II. LOMI preprints E-9-83, E-10-83, Steklov Institute, Leningrad, 1983.

[CGH89] L. Caniglia, A. Galligo, and J. Heintz. Some new effectivity bounds in computational geometry. In T. Mora et al., editor, *Applied Algebra, Algebraic Algorithms and Error Correcting Codes, Proceedings of AAECC-6*, volume 357 of *Lecture Notes in Computer Science*, pages 131–152, Berlin Heidelberg New York, 1989. Springer.

[CGH+03] D. Castro, M. Giusti, J. Heintz, G. Matera, and L.M. Pardo. The hardness of polynomial equation solving. To appear in Foundations of Computational Mathematics, 2003.

[Chi85] A.L. Chistov. Fast parallel calculation of the rank of matrices over a field of arbitrary characteristic. In *Proceedings 5th International Conference on Fundamentals of Computation Theory, Cottbus, Germany, 1985*, volume 199 of *Lecture Notes in Computer Science*, pages 63–69, Berlin Heidelberg New York, 1985. Springer.

[CHLM00] B. Castaño, J. Heintz, J. Llovet, and R. Martínez. On the data structure straight–line program and its implementation in symbolic computation. *Mathematics and Computers in Simulation*, 51:497–528, 2000.

[CLM96] B. Castaño, J. Llovet, and R. Martínez. Polynomials, matrices and algorithms with straight–line programs. In J. Heintz M. Giusti and L.M. Pardo, editors, *Proceedings of TERA'96*, pages 12–14, Santander, Spain, April 1996. Universidad de Cantabria, Facultad de Ciencias, `http://hilbert.matesco.unican.es/tera/`.

[CLO92] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms: an introduction to computational algebraic geometry and commutative algebra*. Undergraduate Texts in Mathematics. Springer, Berlin Heidelberg New York, 1992.

[CLO98] D. Cox, J. Little, and D. O'Shea. *Using algebraic geometry*, volume 185 of *Graduate Texts in Mathematics*. Springer, Berlin Heidelberg New York, 1998.

[CS98] D. Cox and B. Sturmfels. *Applications of computational algebraic geometry*, volume 53 of *Proceedings of Symposia in Applied Mathematics*. American Mathematical Society, Providence, Rhode Island, 1998.

[Csa76] L. Csanky. Fast parallel matrix inversion algorithms. *SIAM Journal of Computing*, 5(4):618–623, 1976.

[DFGS91] A. Dickenstein, N. Fitchas, M. Giusti, and C. Sessa. The membership problem for unmixed polynomial ideals is solvable in single exponential time. *Discrete Applied Mathematics*, 33:73–94, 1991.

[Dia97] A. Diaz. *FOXBOX: A system for manipulating symbolic objects in black box representation*. PhD thesis, Rensselaer Polytechnic Institute, New York, 1997.

[Dix82] J. Dixon. Exact solution of linear equations using p-adic expansions. *Numerische Mathematik*, 40:137–141, 1982.

[DM02] E. Dratman and G. Matera. Deformation techniques for counting the real solutions of specific polynomial equation systems. In P. D'Argenio and G. Matera, editors, *Proceedings Workshop Argentino de Informática Teórica, WAIT'02, Santa Fe, Argentina, September 2002*, volume 31 of *Anales Jornadas Argentinas de Informática e Investigación Operativa*, pages 42–52, Buenos Aires, 2002. SADIO.

[Egn96] S. Egner. Semi–numerical solution to 6/6–stewart–platform kinematics based on symmetry. *Applicable Algebra in Engineering, Communication and Computing*, 7(6):449–468, 1996.

[Eis95] D. Eisenbud. *Commutative Algebra with a View Toward Algebraic Geometry*, volume 150 of *Graduate Texts in Mathematics*. Springer, Berlin Heidelberg New York, 1995.

[EM99] I. Emiris and B. Mourrain. Computer algebra methods for studying and computing molecular conformations. *Algorithmica*, 25(2-3):372–402, 1999. Special Issue on Algorithms for Computational Biology.

[FF63] D.K. Faddeev and V.N. Faddeeva. *Computational methods of linear algebra*. Freeman, San Francisco, 1963.

[FF96] M. Fila and J. Filo. Blow up on the boundary: A survey. In S. Janeczko et al., editor, *Singularities and Differential Equations*, volume 33 of *Banach Center Publications*, pages 67–78, Warsaw, Poland, 1996. Polish Academy of Sciences, Institute of Mathematics.

[FGR02] R. Ferreira, P. Groisman, and J.D. Rossi. Numerical blow–up for a nonlinear problem with a nonlinear boundary condition. *Mathematical models and methods in applied sciences*, 12(4):461–484, 2002.

[FGS95] N. Fitchas, M. Giusti, and F. Smietanski. Sur la complexité du théorème des zéros. In J. Guddat et al, editor, *Approximation and Optimization in the Caribbean II, Proceedings 2nd International Conference on Non–Linear Optimization and Approximation*, volume 8 of *Approximation and Optimization*, pages 247–329. Peter Lange Verlag, Frankfurt am Main, 1995.

[FIKY88] T. S. Freeman, G. M. Imirzian, E. Kaltofen, and L. Yagati. DAGWOOD - A system for manipulating polynomials given by straight-line programs. *ACM Transactions on Mathematical Software*, 14(3):218–240, September 1988.

[GH93] M. Giusti and J. Heintz. La détermination des points isolés et de la dimension d'une variété algébrique peut se faire en temps polynomial. In D. Eisenbud and L. Robbiano, editors, *Computational Algebraic Geometry and Commutative Algebra*, volume XXXIV of *Symposia Matematica*, pages 216–256, Cambridge, 1993. Cambridge University Press.

[GH01] M. Giusti and J. Heintz. Kronecker's smart, little black–boxes. In A. Iserles R. Devore and E. Süli, editors, *Proceedings of Foundations of Computational Mathematics, FoCM'99, Oxford 1999*, volume 284 of *London Mathematical Society Lecture Notes Series*, pages 69–104, Cambridge, 2001. Cambridge University Press.

[GHH+97] M. Giusti, K. Hägele, J. Heintz, J.E. Morais, J.L. Montaña, and L.M. Pardo. Lower bounds for Diophantine approximation. *Journal of Pure and Applied Algebra*, 117,118:277–317, 1997.

[GHL+00] M. Giusti, K. Hägele, G. Lecerf, J. Marchand, and B. Salvy. Computing the dimension of a projective variety: The Projective Noether Maple Package. *Journal of Symbolic Computation*, 30(3):291–307, 2000.

[GHM+98] M. Giusti, J. Heintz, J.E. Morais, J. Morgenstern, and L.M. Pardo. Straight–line programs in geometric elimination theory. *Journal of Pure and Applied Algebra*, 124:101–146, 1998.

[GHS93] M. Giusti, J. Heintz, and J. Sabia. On the efficiency of effective Nullstellensätze. *Computational Complexity*, 3:56–95, 1993.

[Giu84] M. Giusti. Some effectivity problems in polynomial ideal theory. In John Fitch, editor, *Proceedings of the 3rd International Symposium on Symbolic and Algebraic Computation EUROSAM 84 (Cambridge, England, July 9-11, 1984)*, volume 174 of *Lecture Notes in Computer Science*, pages 159–171, Berlin Heidelberg New York, 1984. Springer.

[GLS01] M. Giusti, G. Lecerf, and B. Salvy. A Gröbner free alternative for polynomial system solving. *Journal of Complexity*, 17(1):154–211, 2001.

[GM89] P. Gianni and T. Mora. Algebraic solution of systems of polynomial equations using Gröbner bases. In L. Huguet and A. Poli, editors, *Proceedings 5th International Symposium on Applied Algebra, Algebraic Algorithms and Error–Correcting Codes, AAECC–5, Menorca, Spain, June 15–19, 1987*, volume 356 of *Lecture Notes in Computer Science*, pages 247–257, Berlin Heidelberg New York, 1989. Springer.

[GV98] L. González-Vega. The needs of industry for polynomial system solving: A report from the FRISCO project. *SAC Newsletter*, 3:21–46, may 1998.

[GVRR97] L. González-Vega, F. Rouillier, and M.-F. Roy. Symbolic recipes for polynomial system solving. In *Some Tapas of Computer algebra*, pages 34–65. Springer, Berlin Heidelberg New York, 1997.

[Häg96] K. Hägele. slp - *a* C++ *classlib (A prototype for straight–line programs)*. Departamento de Matemáticas, Estadística y Computación, Universidad de Cantabria, `http://hilbert.matesco.unican.es/tera/soft.html`, 1996.

[Hen81] D. Henry. *Geometric theory of semilinear parabolic equations*, volume 840 of *Lecture Notes in Mathematics*. Springer, Berlin Heideilberg, 1981.

[HKP+00] J. Heintz, T. Krick, S. Puddu, J. Sabia, and A. Waissbein. Deformation techniques for efficient polynomial equation solving. *Journal of Complexity*, 16(1):70–109, 2000.

[HM93] J. Heintz and J. Morgenstern. On the intrinsic complexity of elimination theory. *Journal of Complexity*, 9:471–498, 1993.

[HMPW98] J. Heintz, G. Matera, L.M. Pardo, and R. Wachenchauzer. The intrinsic complexity of parametric elimination methods. *Electronic Journal of SADIO*, 1(1):37–51, 1998.

[HMW01] J. Heintz, G. Matera, and A. Waissbein. On the time–space complexity of geometric elimination procedures. *Applicable Algebra in Engineering, Communication and Computing*, 11(4):239–296, 2001.

[Hof90] C. Hoffman. Algebraic and numeric techniques for offsets and blends. In W. Dahmen, M. Gasca, and C. Micchelli, editors, *Computation of Curves and Surfaces*, pages 499–528, Dordrecht, 1990. Kluwer Academic Publishers.

[Hor91] B.K.P. Horn. Relative orientation revisited. *Journal of Optical Society of America*, 8(10):1630–1638, 1991.

[HS81] J. Heintz and M. Sieveking. Absolute primality of polynomials is decidable in random polynomial–time in the number of variables. In Shimon Even and Oded Kariv, editors, *ICALP 81: Proceedings 8th International Colloquium on Automata, Languages and Programming, Acre (Akko), Israel, July 13-17, 1981*, volume 115 of *Lecture Notes in Computer Science*, pages 16–28. Springer, 1981.

[HS82] J. Heintz and C. P. Schnorr. Testing polynomials which are easy to compute. In *International Symposium on Logic and Algorithmic, Zurich 1980*, volume 30 of *Monographie de l'Enseignement Mathématique*, pages 237–254, 1982.

[Ja'83] J. Ja'Ja'. Time–space tradeoffs for some algebraic problems. *Journal of the Association for Computing Machinery*, 30(3):657–667, 1983.

[Kal88] E. Kaltofen. Greatest common divisors of polynomials given by straight-line programs. *Journal of the Association for Computing Machinery*, 35(1):231–264, 1988.

[Kal89] E. Kaltofen. Factorization of polynomials given by straight-line programs. In S. Micali, editor, *Randomness in Computation*, volume 5 of *Advances in Computing Research*, pages 375–412. JAI Press Inc., 1989.

[KM83] M. Kojima and S. Mizuno. Computation of all solutions to a system of polynomial equations. *Math. Programming*, 25:131–157, 1983.

[KO62] A. Karatsuba and Yu. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Dokl*, 7:595–596, 1962.

[Koi96] P. Koiran. Hilbert's Nullstellensatz is in the polynomial hierarchy. *Journal of Complexity*, 12(4):273–286, 1996.

[KP96] T. Krick and L.M. Pardo. A computational method for Diophantine approximation. In L. González-Vega and T. Recio, editors, *Algorithms in Algebraic Geometry and Applications, Proceedings of MEGA '94*, volume 143 of *Progress in Mathematics*, pages 193–254, Basel, 1996. Birkhäuser.

[Kro82] L. Kronecker. Grundzüge einer arithmetischen Theorie de algebraischen Grössen. *J. reine angew. Math.*, 92:1–122, 1882.

[KS91] E. Kaltofen and B. Saunders. On Wiedemann's method for solving sparse linear systems. In H.F. Mattson, T. Mora, and T.R.N. Rao, editors, *Proceedings of the 9th International Symposium on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, AAECC-9, New Orleans, LA, USA, October 7-11, 1991*, volume 539 of *Lecture Notes in Computer Science*, pages 29–38, Berlin Heidelberg New York, 1991. Springer.

[Laz81] D. Lazard. Résolution des systèmes d'équations algébriques. *Theoretical Computer Science*, 15:77–110, 1981.

[Laz93] D. Lazard. Systems of algebraic equations (algorithms and complexity). In *Computational Algebraic Geometry and Commutative Algebra*, volume XXXIV of *Symposia Matematica*, pages 216–256. Cambridge University Press, 1993.

[Lev90] H.A. Levine. The role of critical exponents in blow up theorems. *SIAM Reviews*, 32:262–288, 1990.

[Mas69] J. Massey. Shift–register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, IT–15(1):122–127, 1969.

[Mat95] G. Matera. Integration of multivariate rational functions given by straight–line programs. In G. Cohen, H. Giusti, and T. Mora, editors, *Proceedings of the 11th International Symposium on Applied Algebra, Algebraic Algorithms and Error–Correcting Codes, AAECC–11*, volume 948 of *Lecture Notes in Computer Science*, pages 347–364, Berlin Heidelberg New York, 1995. Springer.

[Mat97] G. Matera. *Sobre la complejidad en espacio y tiempo de la eliminación geométrica.* PhD thesis, Universidad de Buenos Aires, Argentina, 1997.

[Mat99] G. Matera. Probabilistic algorithms for geometric elimination. *Applicable Algebra in Engineering, Communication and Computing*, 9(6):463–520, 1999.

[May97] E. Mayr. Some complexity results for polynomial ideals. *Journal of Complexity*, 13(3):303–325, 1997.

[MM82] E. Mayr and A. Meyer. The complexity of the word problem for commutative semigroups. *Advances in Mathematics*, 46:305–329, 1982.

[Mos71] J. Moses. Algebraic simplification: A guide for the perplexed. *Communications ACM*, 14:548–560, 1971.

[MV99] M. Mahajan and V. Vinay. Determinant: old algorithms, new insight. *SIAM Journal on Discrete Mathematics*, 12(4):474–490, 1999.

[Pan01] V. Pan. *Structured matrices and polynomials. Unified superfast algorithms.* Birkhäuser, Boston, 2001.

[Pao92] C.V. Pao. *Nonlinear parabolic and elliptic equations.* Plenum Press, 1992.

[Par95] L.M. Pardo. How lower and upper complexity bounds meet in elimination theory. In G. Cohen, H. Giusti, and T. Mora, editors, *Applied Algebra, Algebraic Algorithms and Error Correcting Codes, Proceedings of AAECC–11*, volume 948 of *Lecture Notes in Computer Science*, pages 33–69, Berlin Heidelberg New York, 1995. Springer.

[Rhe98] W.C. Rheinboldt. *Methods for solving systems of nonlinear equations*, volume 70 of *CBMS–NSF Regional Conference Series in Applied Mathematics.* SIAM, Philadelphia, 1998.

[Rou97] F. Rouillier. Solving zero–dimensional systems throught rational univariate representation. *Applicable Algebra in Engineering, Communication and Computing*, 9(5):433–461, 1997.

[Sch80] J.T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the Association for Computing Machinery*, 27(4):701–717, 1980.

[Sch03] E. Schost. Computing parametric geometric resolutions. *Applicable Algebra in Engineering, Communication and Computing*, 13:349–393, 2003.

[Sha94] I.R. Shafarevich. *Basic Algebraic Geometry: Varieties in Projective Space.* Springer, Berlin Heidelberg New York, 1994.

[SS71] A. Schönhage and V. Strassen. Schnelle Multiplikation groβer Zahlen. *Computing*, 7:281–292, 1971.

[SS96] J. Sabia and P. Solernó. Bounds for traces in complete intersections and degrees in the Nullstellensatz. *Applicable Algebra in Engineering, Communication and Computing*, 6(6):353–376, 1996.

[Stu02] B. Sturmfels. *Solving Systems of Polynomial Equations.* CBMS Regional Conference Series in Mathematics. American Mathematical Society, Providence, RI, 2002.

[Tri85] W. Trinks. On improving approximate results of buchberger's algorithm. In B. Caviness, editor, *Proceedings of EUROCAL'85*, volume 204 of *Lecture Notes in Computer Science*, pages 608–611, Berlin Heidelberg New York, 1985. Springer.

[Val92] L. Valiant. Why is boolean complexity theory difficult? In M.S. Paterson, editor, *Boolean Complexity Theory*, volume 169 of *London Mathematical Society Lecture Note Series*, Cambridge, 1992. Cambridge University Press.

[vzG86] J. von zur Gathen. Parallel arithmetic computations: a survey. In B. Rovan J. Gruska and J. Wiedermann, editors, *Proceedings of the 12th Symposium on Mathematical Foundations of Computer Science, Bratislava, Czechoslovakia, August 25–29, 1996*, volume 233 of *Lecture Notes in Computer Science*, pages 93–112, Berlin Heidelberg New York, August 1986. Springer.

[vzG93] J. von zur Gathen. Parallel linear algebra. In J.H. Reif, editor, *Synthesis of Parallel Algorithms*, pages 573–617. Morgan Kaufmann, Los Altas, CA, 1993.

[vzGG99] J. von zur Gathen and J. Gerhard. *Modern computer algebra.* Cambridge University Press, Cambridge, 1999.

[Wie86] D. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, IT–32(1):54–62, 1986.

[Wri85] A.H. Wright. Finding all solutions to a system of polynomial equations. *Mathematics of Computation*, 44:125–133, 1985.

[Zip79] R. Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM '79: Proceedings of International Symposium on Symbolic and Algebraic Computation, Marseille 1979*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979.

[Zul88] W. Zulehner. A simple homotopy method for determining all isolated solutions to polynomial systems. *Mathematics of Computation*, 50(161):167–177, 1988.