INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Combining Algebraic and Numerical Methods in Algorithm Design

by

Yanqiang Yu

A dissertation submitted to the Graduate Faculty
in Computer Science in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
The City University of New York

1998

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

Victor Pan Date: April 24, 1998

Professor Victor Pan Chair of Examining Committee

Stanley Habib Date: April 24, 1998

Professor Stanley Habib Executive Officer

Professor Charles Giardina

Professor Zhi Gang Xiang

Professor Carl Bredlau

Supervisory Committe

THE CITY UNIVERSITY OF NEW YORK

Abstract

Combining Algebraic and Numerical Methods in Algorithm Design

Author: Yanqiang Yu

Advisor: Professor Victor Pan

The dissertation summarizes our studies in developing fast and efficient algorithms by combining the techniques of numerical methods and algebraic methods. The effectiveness of our approach is proven by its applications to several common computational problems. In the case of multipoint polynomial evaluation and polynomial interpolation, by converting the problem into dense structured matrix computation, we have developed efficient approximation algorithm with numerical stability improved over existing algebraic algorithms. We also demonstrate the techniques of binary modular reduction and backward binary segmentation which bring the traditional algebraic method of modular arithmetic into numerical computation algorithms. The techniques are proven to improve significantly the overall efficiency of inner product computation and of the iterative improvement algorithm for matrix inversion. For the evaluation of sign of matrix determinant, we again combine numerical methods with algebraic methods to obtain the solution by using lower precision computations and fewer arithmetic operations. Some of numerical experiment results are included.

Key words: numerical algorithms, symbolic computation, multipoint polynomial evaluation, interpolation, modular arithmetic, inner product of vectors, matrix determinant, error analysis.

To Ying and Tiffany

Contents

List of Tables

Chapter 1

Introduction

1.1 Background

The subject of this dissertation is the design of efficient and numerically stable algorithms by using the combined power of algebraic and numerical methods. Algebraic algorithms are designed to obtain exact solutions to various computational problems by means of algebraic and symbolic manipulation and by computation that uses exact integer or rational arithmetic. Numerical methods use discrete models of computing and the numerical computations are performed in floating or, more rarely, fixed point arithmetic with rounding to represent the real and/or complex numbers in finite precision.

Numerical methods utilize the ability of modern digital computers to perform large amount of simple and standard floating point computations to arrive at the solutions; iterative improvements of the approximation results are often used by numerical methods to remove or decrease the errors introduced when real numbers are approximated with finite precision. On the other hand, special data structures and additional subroutines are of-

ten required to perform symbolic computation needed for most algebraic algorithms.

Historically, numerical algorithms and algebraic algorithms have been developed and studied by two different groups of researchers which had very little interactions between each other. It has been one of our major research interests to bridge or narrow the gap between the two by combining the power of the techniques developed in both areas in order to design new algorithms with low computational complexity and good numerical stability.

I joined Professor Pan and his research team since Fall 1993 and we have been working on various problems with a consistent research methodology. In our research, we focused on fundamental problems in large scale scientific computations, more specifically, polynomial evaluation, interpolation and matrix computations. Our general approach was to first identify the limitations and advantages of the known algorithms, which were often based solely on either algebraic or numerical methods. Then we tried to obtain improvements by designing new algorithms which combined algebraic and numerical methods to overcome the limitations that could not be managed by either method alone. To verify and further improve the efficiency, numerical stability, and usability of our algorithms, we implemented the algorithms and performed numerical experiments with various input sets.

1.2 Our Contributions

We have successfully applied the approach to several computational problems with wide ranges of practical applications. In the dissertation, I will present the results of our research with an emphasis on the general approach

we used.

Our first topic was multipoint polynomial evaluation and polynomial interpolation. The fastest known algorithm based on purely algebraic techniques suffered from severe numerical stability problems, to the extent that the produced output was frequently corrupted completely in the case when the algorithm performed numerically. Based on the conversion of the problem into dense-structured matrix computation, we have overcome this problem by designing algorithms have both low arithmetic cost and improved numerical stability. According to our numerical test, our algorithms provide correct solutions to the multipoint polynomial evaluation problem even in the cases where the output of the known fast algorithm was completely contaminated by the rounding errors.

Our second approach relies on the techniques of binary modular reduction and binary segmentation. The techniques bring the traditional algebraic method of modular arithmetic into numerical computation algorithms and significantly improve the overall efficiency of the inner product evaluation, under certain assumptions satisfied for some important problems of matrix computations, iterative improvement algorithm for matrix inversion, as well as the solution of linear partial differential equations (PDE's).

Thirdly, for the evaluation of the sign of a matrix determinant, we again find that the combination of numerical methods with algebraic methods enables us to obtain the sign of the determinant by using lower precision computations and fewer arithmetic operations. The problem and the results are fundamental to practical geometric computation.

Our contributions made to each of the above major areas of computa-

tion are of independent interest from the technical point of view. We hope that our work will serve as yet another demonstration of the power of the combination of algebraic and numerical computational techniques.

1.3 Organization of the Thesis

The next three chapters presents background of the computational problems that we study and our results. Each chapter starts with description of a specific computational problem. Then we comment on previous results, including the advantages and limitations of the known algorithms, and present our solutions using combined algebraic and numerical approaches and the results of our numerical experiments.

Chapter 2 presents improved algorithms for multipoint polynomial evaluation and polynomial interpolation. Chapter 3 presents the application of modular arithmetic to linear algebra computations. Chapter 4 contains our recent work and results on the evaluation of the sign of matrix determinant and its application to computational geometry. Some results of our work were included into [PZHY], [PYS], [EPY] and [BPY].

Chapter 5 summarizes our results, reiterates the key points of our techniques, and discusses our on-going and future research.

1.4 Acknowledgments

I would like to thank my advisor Professor Pan for his invaluable guidance and support throughout my dissertation research and preparation process. The materials presented in the dissertation are the results of joint research projects directed by Professor Pan. I definitely owe my thanks to my coau-

thors and collaborators, and to the members of the Dissertation Committee, for their contributions, comments and suggestions. I also thank Sobze Isdor, Ailong Zheng, and Colin Stewart, whom I have been working closely with, for their continuous support.

Chapter 2

Fast Multipoint Polynomial Evaluation and Interpolation via Computations with Structured Matrices

2.1 Overview

Suppose that we are given n points, $x_0, x_1, \cdots, x_{n-1}$, and the coefficients of a polynomial $p(x)$ of degree n. One may evaluate the value of polynomial at the given points, $p(x_0), p(x_1), \cdots, p(x_{n-1})$ in $2n^2$ arithmetic operations, by means of Horner's algorithm, or in $O(n \log^2 n)$, by means of the more recent algorithm of Moenck and Borodin (cf. [BP]), which, however, leads to numerical stability problems (due to recursive application of polynomial division). For $x_0, x_1, \cdots, x_{n-1}$ lying in a fixed real interval, the alternative algorithms of [R88] and [P95] approximate the values $p(x_0), p(x_1), \cdots, p(x_{n-1})$ at the cost of $O(n \log^2 n)$, with improved numerical stability. Confinement of the input points to the real interval is essential in the approaches of [R88] and [P95] since they rely on a certain result on

approximation of functions on a real line interval.

In [PSLT], it was proposed to use the properties of various classes of structured matrices (that is, Toeplitz-like, Vandermonde-like, and Cauchy-like) and the known correlation among these classes [defined via some associated linear operators of scaling and displacement (shift)] in order to solve the problem of multipoint polynomial approximation and also the converse problem of approximate polynomial interpolation. The idea of exploiting the correlation among the above matrix classes in order to improve some fundamental matrix computations was first proposed in [P90] and later on used in [GKO], but as an alternative approach to multipoint polynomial evaluation and interpolation, this idea was first applied in [PSLT]. Unlike [R88] and [P95], the approach of [PSLT] allows complex input points $x_0, x_1, \cdots, x_{n-1}$.

Our research refines and extends the latter approach. Like [PSLT], we define the original evaluation and interpolation problems by the vector equation $\vec{v} = V(\vec{x})\,\vec{p}$, where $\vec{p}$ and $\vec{v}$ are the vectors of the coefficients of the polynomial and of its values at the given points $x_0, x_1, \cdots, x_{n-1}$, respectively, and $V(\vec{x})$ is the associated Vandermonde matrix. Then, instead of rather complicated reductions of [PSLT] to the computations with Toeplitz-like and Cauchy-like matrices, we multiply $V(\vec{x})$ by a discrete Fourier transform matrix and arrive at a Cauchy-like matrix, for which approximate solutions to our original problems can be readily computed, due to a simple technique of summation reordering (cf. [R85] or [BP]).

We specify the number of arithmetic operations sufficient in order to guarantee the desired bound ϵ on the output approximation errors (this

number is shown to be $O(n \log n)$ for a large class of input sets of points and for $\log(1/\epsilon) = O(\log n)$) and then show 2 extensions of the approach, in particular, to multipoint approximation of a polynomial given by its Chebyshev decomposition (section 2.7) and to the approximation of the vector $\vec{v}$ for a given vector $\vec{p}$, where $\vec{v} = V\ \vec{p}$, for a Vandermonde-like matrix V (section 2.6).

In sections 2.2 and 2.4, we present some definitions and auxiliary results. In sections 2.3 and 2.2, we treat multipoint evaluation and interpolation, respectively.

2.2 Preliminaries

Given the coefficient vector $\vec{p} = [\ p_0, \cdots, p_{n-1}\ ]^T$ of a polynomial

$$p(x) = \sum_{i=0}^{n-1} p_i x^i,$$

the values of $p(x)$ on the set of points $\{x_0, \cdots, x_{n-1}\}$ are defined by the vector equation,

$$\vec{v} = V(\vec{x})\ \vec{p}, \tag{2.1}$$

where $\vec{x} = (x_0, \cdots, x_{n-1})^T$ and

$$V(\vec{x}) = \begin{pmatrix} 1 & x_0 & \cdots & x_0^{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n-1} & \cdots & x_{n-1}^{n-1} \end{pmatrix}, \quad x_i \neq x_j,\ i \neq j,\ i,j = 0, \cdots, n-1\ .$$

$V(\vec{x})$ is called a Vandermonde matrix. Hereafter, w_k denotes the $(1+k)$-th component of a vector $\vec{w} = [\ w_0, \cdots, w_{n-1}\ ]^T$.

The vector (2.1) defines the problems of multipoint evaluation and interpolation for a polynomial $p(x)$, that is, the problem of computing the

vector $\vec{v}$ of the values of $p(x)$, where the vectors $\vec{p}$ of the coefficients and $\vec{x}$ of points (nodes of evaluation) are given, and the problem of computing the coefficient vector $\vec{p}$, where the vectors of the values $\vec{v}$ and points (nodes of interpolation) $\vec{x}$ are given, respectively. We will approach the above two problems by using scaling and displacement (shift) operators for representation of the Vandermonde matrix $V(\vec{x})$ of (2.1). By following [GKK], [P90], [GO], and [BP], we define such an operator for a Vandermonde matrix $V = V(\vec{x})$ as follows:

$$S_{(D_{1/\vec{x}},Z_1^T)}(V) = D_{1/\vec{x}}V - VZ_1^T = \vec{g}\vec{e}^{\,(0)T} . \tag{2.2}$$

Here and hereafter, we write

$$\left.\begin{array}{lcl} \vec{e}^{\,(0)T} & = & [\,1,0,\cdots,0\,]\,, \\ D_{1/\vec{x}} & = & diag\,(\,\frac{1}{x_0},\cdots,\frac{1}{x_{n-1}}\,)\,, \\ \vec{g} & = & \left[\;\frac{1}{x_0}-x_0^{n-1},\cdots,\frac{1}{x_{n-1}}-x_{n-1}^{n-1}\;\right]^T \end{array}\right\} \tag{2.3}$$

and

$$Z_1 = \begin{pmatrix} 0 & \cdots & 0 & 1 \\ 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 \end{pmatrix} .$$

It is known from [H] and [GKO] that VF^* is a Cauchy-like matrix (compare definition 2.1 below) such that

$$S_{(D_{1/\vec{x}},D_1^*)}(VF^*) = \vec{g}\,\vec{h}^{\,T}\,,$$

where

$$\vec{h}^{\,T} = \vec{e}^{\,(0)T}F^*, \qquad D_1^* = diag(\,1,e^{-\frac{2\pi\sqrt{-1}}{n}},\cdots,e^{-\frac{2\pi\sqrt{-1}}{n}(n-1)}\,)\,,$$

$F = (\,1/\sqrt{n}\,)[\,e^{2\pi jk\sqrt{-1}/n}\,]_{j,k=0}^{n-1}$ denotes the (normalized) matrix of the discrete Fourier transform, and F^* denotes the Hermitian transpose of the

matrix F, $F^* = F^{-1}$. From [GO], we have

$$V(\vec{x})\ F^* = \frac{1}{\sqrt{n}} diag(\frac{1}{x_i} - x_i^{n-1})\ C(\vec{s}, \vec{t}\ )\ , \qquad i = 0, 1, \cdots, n-1, \quad (2.4)$$

where

$$C(\vec{s}, \vec{t}\ ) = \left[\frac{1}{s_i - t_j} \right]_{i,j=0}^{n-1} \quad (2.5)$$

is a Cauchy matrix, $s_i = \frac{1}{x_i}$, and $t_j = \epsilon^{-\frac{2\pi j\sqrt{-1}}{n}}$ for $i, j = 0, 1, \cdots, n-1$.

Here and hereafter, we use the following definition:

Definition 2.1 [GO]. *An $n \times n$ matrix C is called Cauchy-like matrix if*

$$S_{(D_{1/\vec{s}}, D_{\vec{t}})}(C) = G\ H^T\ , \quad G \in C^{n \times \alpha}\ ,\ H \in C^{n \times \alpha}\ . \quad (2.6)$$

Then, $G\ H^T$ is called a $(D_{1/\vec{s}}, D_{\vec{t}})$-generator (or a scaling generator) of C of length α, and the minimum α allowing the above representation (2.6) is called the $(D_{1/\vec{s}}, D_{\vec{t}})$-rank (or the scaling rank) of C.

Cauchy matrix of (2.5) is a special case of Cauchy-like matrices of $(D_{1/\vec{s}}, D_{\vec{t}})$-rank 1.

2.3 Multipoint Polynomial Evaluation

We extend and strengthen the results of [PSLT] on fast approximate multipoint polynomial evaluation. The algorithm, its correctness proof, and the complexity analysis are presented in the statement and the proof of the following proposition:

Proposition 2.1 *Given the coefficients of a polynomial*

$$p(x) = \sum_{i=0}^{n-1} p_i x^i$$

and the set of points $\{x_0, \cdots, x_{n-1}\}$, *the vector* $\vec{v}$ *of the equation (2.1) can be approximated within maximum error norm bound* ϵ (*that is, we may compute a vector* $\vec{v}^{\,*}$ *such that*

$$|| \vec{v}^{\,*} - \vec{v} || = \max_i | v_i^* - v_i | \leq \epsilon \quad)$$

by using $(4n-1)L + O(n \log n)$ *arithmetic operations, where*

$$L = \lceil \frac{\log(\alpha n b / ((1-q)\epsilon))}{\log(1/q)} \rceil,$$

q *is a fixed constant,* $q \geq x_+ = \max_k |x_k|$,

$$\alpha = \max_k |u_k|, \quad u_k = (\frac{1}{\sqrt{n}} F \vec{p} \,)_k, \quad \textit{and} \quad b = \max_k |x_k^n - 1| \leq 1 + q^n \ ,$$

so that $L = O(\log n)$ *if* $\log(\alpha/\epsilon) = O(\log n)$ *and* $q < 1$.

Proof: Due to (2.4), we have

$$\vec{v} = V(\vec{x}) \ \vec{p} = diag(\frac{1}{x_i} - x_i^{n-1}) \ C(\vec{s}, \vec{t} \,) \ \vec{u} = [\, v_0, v_1, \cdots, v_{n-1} \,]^T,$$

where $\vec{u} = (1/\sqrt{n}) \ F \ \vec{p}$,

$$\begin{aligned} v_i = v(x_i) = (\frac{1}{x_i} - x_i^{n-1}) \sum_{k=0}^{n-1} \frac{u_k}{\frac{1}{x_i} - t_k} &= (1 - x_i^n) \sum_{k=0}^{n-1} \frac{u_k}{1 - t_k x_i} \\ &= (1 - x_i^n) \sum_{j=0}^{\infty} A_j x_i^j \end{aligned}$$

$$A_j = \sum_{k=0}^{n-1} u_k t_k^j.$$

Due to the summation reordering, we now fix a sufficiently large natural L, approximate v_i by

$$v_i^* = v_i^{(L)} = (1 - x_i^n) \sum_{j=0}^{L-1} A_j x_i^j \ ,$$

note that $|t_k| = 1$ for all k, so that $|A_j| \leq \alpha n$, for all j, and obtain that

$$E_L = || \vec{v}^{\,*} - \vec{v} || = \max_i | v_i^* - v_i | \leq \frac{\alpha n b}{1-q} q^J \ . \tag{2.7}$$

Therefore, for $L \geq \lceil \frac{\log(\alpha n b/((1-q)\epsilon))}{\log(1/q)} \rceil$, we have $E_L \leq \epsilon$.

Evaluation of the vectors $F\,\vec{p}$ and $\vec{u} = [\, u_0, \cdots, u_{n-1} \,]^T$ and of the scalars $1 - x_i^n$,

$i = 0, 1, \cdots, n-1$, requires $O(n \log n)$ arithmetic operations; evaluation of A_j, for all j,

$j = 0, 1, \cdots, L-1$, requires $L(2n-2)$ arithmetic operations, and the subsequent evaluation of

$$v_i^* = (1 - x_i^n) \sum_{j=0}^{L-1} A_j x_i^j, \qquad \text{for all } i,\ 0 \leq i \leq n-1 \ ,$$

requires $(1+2L)n$ arithmetic operations. Therefore, we obtain an approximation to the vector $V(\vec{x})\vec{p}$ by using $L(4n-1) + O(n \log n) + n$ arithmetic operations. This is $O(n \log n)$, for $L = O(\log n)$, and if we have

$$x_+ = \max_k |x_k| \leq q < 1 \ (\text{ for a fixed constant } q\ )$$

and $\log(\alpha/\epsilon) = O(\log n)$, then $L = O(\log n)$.

Remark 2.1 Given any vector $\vec{x}$, we may choose 2 scalars $\gamma \neq 0$ and δ, such that

$$|y_i| \leq q < 1 \tag{2.8}$$

for $y_i = (x_i - \delta)/\gamma$, $x_i = \gamma y_i + \delta$. Then, by scaling and shifting the variable x, we turn it into a new variable $y = (x - \delta)/\gamma$ and thus transform the polynomial

$$p(x) = \sum_{i=0}^{n-1} p_i x^i$$

into the polynomial

$$p_{\gamma,\delta}(y) = \sum_{i=0}^{n-1} p_{\gamma,\delta,i}\, y^i = \sum_{i=0}^{n-1} p_i(\gamma y + \delta)^i = \sum_{i=0}^{n-1} \gamma^i y^i \sum_{h=i}^{n} p_h \delta^{h-i} \binom{h}{i} ,$$

whose coefficients can be computed by using $O(n \log n)$ operations, if we are given γ, δ, and the vector $\vec{p}$ (compare [BP]). We have $p(x_i) = p_{\gamma,\delta}(y_i)$. Due to (2.6), we may apply proposition 2.1 in order to approximate $p_{\gamma,\delta}(y_i)$, $i = 0, 1, \cdots, n-1$, and then extend this result to approximate evaluation of $p(x_0), p(x_1), \cdots, p(x_{n-1})$ for the set $\{x_0, x_1, \cdots, x_{n-1}\}$; the estimates of proposition 2.1 for the approximation errors should change since they will now depend on the coefficients

$$p_{\gamma,\delta,i} = \sum_{i=0}^{n-1} \gamma^i \sum_{h=i}^{n} p_h \delta^{h-i} \binom{h}{i},$$

of the polynomial $p_{\gamma,\delta}(x)$.

2.4 Some Further Definitions and Auxiliary Results

In this section, we recall some definitions and auxiliary results from [BP], [GO], and [H].

Definition 2.2 *For a vector* $\vec{x} = [\, x_0, \cdots, x_{n-1} \,]^T$, *let* $\Gamma(t) = \Gamma_{\vec{x}}(t)$ *denote the polynomial*

$$\prod_{i=0}^{n-1} (t - x_i) = t^n + \sum_{i=0}^{n-1} r_i t^{i-1}$$

and let $\vec{r} = \vec{r}(\vec{x})$ *denote the vector* $[\, r_0, \cdots, r_{n-1} \,]^T$.

Definition 2.3 *For a given vector* $\vec{u} = [\, u_0, u_1, \cdots, u_{n-1} \,]^T$ *and a given scalar* $\phi \neq 0$, *an* $n \times n$ *matrix* $C_{(\phi,n)}(\vec{u}) = [c_{i,j}]$ *is called a* ϕ*-circulant matrix if* $c_{i,j} = u_{(i-j) \bmod n}$ *for* $i \geq j$, $c_{i,j} = \phi u_{(i-j) \bmod n}$ *for* $i < j$.

We have the following well-known equation (compare [BP], page 144):

$$V^T(\vec{t}\,) = J\ Z_f^{-1}(\vec{r} + f\vec{e}^{\,(0)})\ V^{-1}(\vec{t}\,)\ diag(\ a_i\ )$$

or, equivalently,

$$V^{-1}(\vec{t}\,) = Z_f(\vec{r} + f\vec{e}^{\,(0)})\ J\ V^T(\vec{t}\,)\ diag(\ a_i^{-1}\ )\ , \tag{2.9}$$

provided that $f \neq x_i^n, \quad i = 0, 1, \cdots, n-1,$

$$a_i = \Gamma'_{\vec{x}}(x_i)(\ f - x_i^n\ )\ , \qquad i = 0, 1, \cdots, n-1, \tag{2.10}$$

$\Gamma(t) = \Gamma_{\vec{x}}(t)$ and $\vec{r} = \vec{r}(\vec{x})$ are defined in definition 2.2, $Z_f(\vec{r} + f\vec{e}^{\,(0)})$ denotes the f-circulant matrix that has the first column $\vec{r} + f\vec{e}^{\,(0)}$, and

$$J = \begin{pmatrix} 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 1 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \end{pmatrix}.$$

2.5 Polynomial Interpolation Algorithm

According to the equations (2.1) and (2.9), the polynomial interpolation problem is equivalent to computing the following vector:

$$\vec{p} = V^{-1}(\vec{x})\ \vec{v} = Z_f(\vec{r} + f\vec{e}^{\,(0)})\ J\ V^T(\vec{x})\ diag(a_i^{-1})\ \vec{v}\ . \tag{2.11}$$

Here, $a_i = \Gamma'_{\vec{x}}(x_i)(f - x_i^n)$, according to (2.10); $V(\vec{x})$ is a Vandermonde matrix satisfying (2.2), and $\vec{g}$, $\vec{e}^{\,(0)}$, and $D_{1/\vec{x}}$ are defined by (2.3).

Proposition 2.2 *FV^T is a Cauchy-like matrix such that*

$$S_{(D_1, D_{1/\vec{x}})}(FV^T) = D_1 FV^T - FV^T D_{1/\vec{x}} = -F\vec{e}^{\,(0)T}\vec{g}^{\,T} = \vec{g}\,'\,\vec{b}\,'^{\,T},$$

where

$$\begin{aligned} \vec{g}\,' &= -F\vec{e}^{\,(0)^T} = [\,-1,-1,\cdots,-1\,]^T, \\ \vec{b}\,' &= \vec{g}^T = \left[\ \tfrac{1}{x_0} - x_0^{n-1},\ \cdots,\ \tfrac{1}{x_{n-1}} - x_{n-1}^{n-1}\ \right], \\ D_1 &= diag(\ 1, e^{\frac{2\pi\sqrt{-1}}{n}}, \cdots, e^{\frac{2\pi\sqrt{-1}}{n}(n-1)}\ ) . \end{aligned}$$

Proof: Indeed, (2.2) implies that

$$S_{(Z_1,D_{1/\vec{x}})}(V^T) = Z_1V^T - V^TD_{1/\vec{x}} = \vec{e}^{\,(0)}\ \vec{g}^{\,T} .$$

Since $F^*\ F = I,\quad F\ Z_1\ F^* = D_1$. we obtain

$$\begin{aligned} FS_{(Z_1,D_{1/\vec{x}})}(V^T) &= FZ_1F^*FV^T - FV^TD_{1/\vec{x}} \\ &= D_1FV^T - FV^TD_{1/\vec{x}} = F\vec{e}^{\,(0)}\ \vec{g}^{\,T} , \end{aligned}$$

and proposition 2.2 follows.

From section 3 of [GO], we have

$$S_{(D_{1/\vec{x}},D_{\vec{t}})}(\ C(\vec{s},\vec{t}\,)\ ) = \vec{s}\ (-\vec{g}\,') .$$

By combining the latter equation with proposition 2.2, we obtain the following result:

Corollary 2.1

$$FV^T = -C(\vec{s},\vec{t}\,)\ diag(\ \frac{1}{x_i} - x_i^{n-1}\ ) ,$$

where $\vec{s} = (s_j)_{j=0}^{n-1}$, $s_j = e^{\frac{2\pi\sqrt{-1}}{n}(n-1)}$, $j = 0,1,\cdots,n-1$, *and* $\vec{t} = (t_j)_{j=0}^{n-1} = (\frac{1}{x_j})_{j=0}^{n-1}$.

Proposition 2.3 *Given 3 sets of values:*

i) $\{ x_i : i = 0, 1, \cdots, n-1;\ x_i \neq x_j \text{ for } i \neq j \}$,

ii) $\{ v_i : i = 0, 1, \cdots, n-1 \}$, *and*

iii) $\{ r_i : i = 0, 1, \cdots, n-1 \}$

(where the values $\{x_i\}$ *and* $\{r_i\}$ *are related according to definition 2.2, one may approximate within* ϵ *the coefficient vector* $\vec{p} = [\ p_0, \cdots, p_{n-1}\ ]$ *of the polynomial*

$$p(x) = \sum_{i=0}^{n-1} p_i x^i$$

(that is, one may compute a vector $\vec{p}^{\,*}$ *such that* $\| \vec{p}^{\,*} - \vec{p} \| \leq \epsilon$ *) by using* $O(n \log n)$ *arithmetic operations if*

$$\max_i ||x_i|| \leq q, \quad q \text{ is a fixed constant, } q < 1, \qquad \log(r\alpha/\epsilon) = O(\log n)\ ,$$

$$\alpha = \max_i |y_i|\ , \qquad y_i = \frac{1 - x_i^n}{a_i x_i}\ v_i\ , \qquad \text{and} \qquad r = \max_{i>0} |r_i|\ ,$$

for $r_1, \cdots, r_{n-1}$ *of definition 2.2.*

Proof: Due to corollary 2.1, we have

$$V^T = -F^*\ C(\vec{s}, \vec{t}\,)\ diag(\ \frac{1}{x_i} - x_i^{n-1}\ )\ .$$

By combining the latter equation with (2.11), we obtain that

$$\vec{p} = -Z_f(\vec{r} + f\vec{e}^{\,(0)})\ J\ F^*\ C(\vec{s}, \vec{t}\,)\ diag(\ \frac{1 - x_i^n}{a_i x_i}\ )\ \vec{v}\ .$$

Let

$$\vec{y} = diag(\ \frac{1 - x_i^n}{a_i x_i}\ )\ \vec{v} = [\ y_0, \cdots, y_{n-1}\ ]^T,$$

so that

$$y_i = \frac{1 - x_i^n}{a_i x_i} v_i\ , \qquad \vec{p} = -Z_f(\ \vec{r} + f\vec{e}^{\,(0)}\ )\ J\ F^*\ C(\vec{s}, \vec{t}\,)\ \vec{y}\ , \qquad (2.12)$$

and consider the following vector equation:

$$\vec{w} = \begin{pmatrix} w_0 \\ \vdots \\ w_{n-1} \end{pmatrix} = C(\vec{s},\vec{t}\,) \begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} \frac{y_0}{s_0-t_0} + \cdots + \frac{y_{n-1}}{s_0-t_{n-1}} \\ \vdots \\ \frac{y_0}{s_{n-1}-t_0} + \cdots + \frac{y_{n-1}}{s_{n-1}-t_{n-1}} \end{pmatrix}. \tag{2.13}$$

The i-th coordinate of the vector $\vec{v}$ can be re-written as follows:

$$w_i = \sum_{k=0}^{n-1} \frac{y_k}{s_i - t_k} = \sum_{k=0}^{n-1} \frac{y_k}{s_i - \frac{1}{x_k}} = -\sum_{k=0}^{n-1} \frac{y_k x_k}{1 - s_i x_k} = -\sum_{k=0}^{n-1} y_k x_k \sum_{j=0}^{\infty} (s_i x_k)^j \ .$$

We write $x_+ = \max_k |x_k|$. If $x_+ < 1$, we have $|s_i x_k| \le x_+ < 1$ for all pairs (i,k), $1 \le i$,

$k \le n$, since $s_h = e^{\frac{2\pi h\sqrt{-1}}{n}}$, $h = 0, 1, \cdots, n-1$. Then, the series $\sum_{j=0}^{\infty}(s_i x_k)^j$ converges for

all k, and we have

$$| \sum_{j=0}^{\infty} (s_i x_k)^j | \le \sum_{j=0}^{\infty} x_+^j = \frac{1}{1-x_+} \quad .$$

Now, we write

$$B_j = \sum_{k=0}^{n-1} y_k x_k^{j+1}, \qquad w_i = -\sum_{j=0}^{\infty}\sum_{k=0}^{n-1} y_k x_k^{j+1} s_i^j = -\sum_{j=0}^{\infty} B_j s_i^j \quad .$$

By means of limiting the number of the summation terms $B_j s_i^j$ to a certain number L, we obtain an approximation of the vector $\vec{w}$ by the vector

$$\vec{w}^{\,*} = \vec{w}^{\,(L)} = [\ w_0^*, w_1^*, \cdots, w_{n-1}^*\ ]^T$$

such that

$$w_i^* = -\sum_{j=0}^{L-1} B_j s_i^j \ .$$

For the error norm,

$$\epsilon_L = || \ w^* - w \ || = \max_i | \ w_i^* - w_i \ | \ ,$$

we obtain the bound

$$\epsilon_L \leq \frac{n\alpha q^{L+1}}{1-q} ,$$

similarly to (2.7). From (2.12) and (2.13), we obtain that

$$\vec{p} = -Z_f(\vec{r} + f\vec{e}^{\,(0)}) \; J \; F^* \; \vec{w} .$$

We write

$$\vec{p}^{\,*} = -Z_f(\vec{r} + f\vec{e}^{\,(0)}) \, JF^* \, \vec{w}^{\,*} , \qquad E^* = E_L^* = || \, \vec{p}^{\,*} - \vec{p} \, || = \max_i | \, p_i^* - p_i \, |$$

and obtain that $E^* \leq || \; Z_f(\; \vec{r} + f\vec{e}^{\,(0)} \;) \; ||_\infty \; ||J||_\infty \; ||F^*||_\infty \; e_L$,

where $||A||_\infty$ denotes the row norm of a matrix $A = [a_{ij}]$,

$$||A||_\infty = \max_i \sum_j |a_{ij}| \qquad (\text{compare [GL], p.57}) .$$

We have

$$||J||_\infty = 1, \qquad ||F||_\infty = \sqrt{n}, \qquad || \; Z_f(\; \vec{r} + f\vec{e}^{\,(0)} \;) \; ||_\infty \leq n \max_{i>0} |r_i| ,$$

if we choose f so as to decrease or cancel the first component of the vector $\vec{r} + f\vec{e}^{\,(0)}$. Therefore,

$$E^* \leq \max_{i>0} |r_i| \; n^{3/2} \; e_L \leq \frac{r n^{5/2} \alpha q^{L+1}}{1-q},$$

where $r = || \; r(\vec{x}) \; ||_\infty = \max_{i>0} |r_i|$. By assumption, $q < 1$, and therefore, we have $E^* < \epsilon$ for a fixed positive ϵ, if

$$L \geq \lceil \; \frac{\log(\; r\alpha n / (\; (1-q)\epsilon \;) \;)}{\log(1/q)} \; \rceil .$$

Thus, given the assumptions of proposition 2.3, we may choose a value L such that $L = O(\log n)$.

We will now complete the proof of proposition 2.3 by showing that $O(n \log n + nL)$ arithmetic operations suffice for the evaluation of the vector $\vec{p}^{\,*}$. The transition from $\vec{w}^{\,*}$ to $\vec{p}^{\,*}$ is reduced to performing discrete Fourier transforms (by means of $O(n \log n)$ operations) [BP], and we will only need to estimate the arithmetic cost of computing the vector $\vec{w}^{\,*}$. We observe that $2n-1$ arithmetic operations suffice for computing each B_j. If all the B_j are available, then for any fixed integer i, $2(L-1)$ arithmetic operations suffice for computing w_i^*, so that all w_i^*, $i = 0, 1, \cdots, n-1$, can be computed by using

$$2(L-1)n + L(2n-1) = 4Ln - 2n - L$$

arithmetic operations.

2.6 Multiplication of Vector by Vandermonde-like Matrix

In this section, we will extend the results of section 2.3 to the class of Vandermonde-like matrices, which includes Vandermonde matrices as a special subclass.

Definition 2.4 [GO]. *V is an $n \times n$ Vandermonde-like matrix if*

$$\nabla_{(D_{1/\vec{x}}, Z_1^T)}(V) = D_{1/\vec{x}}V - VZ_1^T = GB^T \ , \tag{2.14}$$

where $G = [\ \vec{g}_1, \vec{g}_2, \cdots, \vec{g}_\beta\ ] \in C^{n \times \beta}$, $B \in C^{n \times \beta}$, and $D_{1/\vec{x}}$ and Z_1^T are defined as in the previous sections. GB^T is a (nonunique) $(D_{1/\vec{x}}, Z_1^T)$-generator of V of length β, and the minimum possible β allowing the representation (2.14) *is the $(D_{1/\vec{x}}, Z_1^T)$-rank of V.*

It can be easily verified that for V satisfying (2.14), VF^* is a Cauchy-like matrix such that

$$\nabla_{(D_{1/\vec{x}},D_1^*)}(VF^*) = GH^*,$$

where $H^* = B^TF^* = [\ h_1, \cdots, h_\beta\ ]^*$ and the superscript '*' stands for Hermitian transpose. By following [GKO], [GO], and [H], we obtain that

$$VF^* = \sum_{m=1}^{\beta} diag(g_m)\ C(\frac{1}{\vec{x}}, \vec{t}\ )\ diag(h_m)\ ,$$

where $1/\vec{x} = (\frac{1}{x_i})_{i=0}^{n-1}$, $\vec{t} = (t_k)_{k=0}^{n-1}$, and $t_k = e^{-\frac{2\pi k\sqrt{-1}}{n}}$, $0 \le k \le n-1$.

Then, for any vector $\vec{p} \in C^{n\times 1}$, we have

$$V\vec{p} = \sum_{m=1}^{\beta} diag(g_m)\ C(\frac{1}{\vec{x}}, \vec{t}\ )\ diag(h_m)\ F\ \vec{p}\ .$$

Denote

$$V_m = diag(h_m)\ F\ \vec{p} = \begin{pmatrix} v_{m,0} \\ \vdots \\ v_{m,n-1} \end{pmatrix}, \qquad \vec{g}_m = \begin{pmatrix} g_{m,0} \\ \vdots \\ g_{m,n-1} \end{pmatrix}, \qquad 1 \le m \le \beta.$$

Then, we have

$$V\vec{p} = \begin{pmatrix} \sum_{m=1}^{\beta} g_{m,0} \sum_{k=0}^{n-1} \frac{v_{m,k}}{\frac{1}{x_0} - t_k} \\ \vdots \\ \sum_{m=1}^{\beta} g_{m,n-1} \sum_{k=0}^{n-1} \frac{v_{m,k}}{\frac{1}{x_{n-1}} - t_k} \end{pmatrix} = \begin{pmatrix} u(x_0) \\ \vdots \\ u(x_{n-1}) \end{pmatrix},$$

where

$$u(x_i) = \sum_{m=1}^{\beta} g_{m,i} \sum_{k=0}^{n-1} \frac{v_{m,k}}{\frac{1}{x_i} - t_k} \qquad \text{if} \quad |x_i| < q < 1.$$

Furthermore,

$$u(x_i) = \sum_{m=1}^{\beta} g_{m,i} \sum_{k=0}^{n-1} \frac{v_{m,k}x_i}{1 - t_kx_i} \ = \ \sum_{m=1}^{\beta} g_{m,i} \sum_{k=0}^{n-1} v_{m,k}x_i \sum_{j=0}^{\infty} (t_kx_i)^j$$

$$= \sum_{m=1}^{\beta} g_{m,i} \sum_{j=0}^{\infty} A_{j,m} x_i^{j+1} .$$

Write

$$A_{j,m} = \sum_{k=0}^{n-1} v_{m,k}\, t_k^j, \qquad m = 1, 2, \cdots, \beta,$$

and define multipoint polynomial approximations as follows:

$$u^*(x_i) = \sum_{m=1}^{\beta} g_{m,i} \sum_{j=0}^{L-1} A_{j,m} x_i^{j+1} , \quad \text{and} \qquad E_L(x_i) = \mid u(x_i) - u^*(x_i) \mid .$$

We have

$$E_L(x_i) = \mid \sum_{m=1}^{\beta} g_{m,i} \sum_{j=L}^{\infty} A_{j,m} x_i^{j+1} \mid \leq \frac{n\alpha\beta b q^{L+1}}{1-q} ,$$

where $x_+ = \max_i |x_i| \leq q$, as in sections 2.2 and 2.3, $\alpha = \max_{m,k} |v_{m,k}|$, $b = \max_{m,k} |g_{m,k}|$. Therefore, for

$$L \geq \lceil \frac{\log(\, n\alpha\beta b/(\,(1-q)\epsilon\,)\,)}{\log(1/q)} \rceil - 1 , \tag{2.15}$$

we have $E_L(x_i) \leq \epsilon$.

Next, we estimate the computational cost of approximation of the values $u(x_i)$. Computing $A_{j,m}$ for all $j = 0, 1, \cdots, L-1, \quad m = 1, 2, \cdots, \beta$, involves $L(2n-2)\beta$ arithmetic operations. Subsequent computation of $g_{mi} \sum_{j=0}^{L-1} A_{j,m} x_i^{j+1}$ involves $2L+1$ arithmetic operations. Thus, for all $1 \leq i \leq n$, $1 \leq m \leq \beta$, computing $u^*(x_i)$ uses

$$L(n-2)\beta + 2nL\beta + (\beta-1)n + O(\beta n \log n) + n\beta = (4n-2)L\beta + O(\beta n \log n)$$

arithmetic operations. Here, $O(n \log n)$ arithmetic operations are needed to compute V_m for each m, $m = 1, 2, \cdots, \beta$.

We have proved the following result:

Proposition 2.4 *For an $n \times n$ Vandermonde-like matrix V, given with its generator GB^T, $G \in C^{n\times\beta}$, $B \in C^{n\times\beta}$, and for a given vector $\vec{p} \in C^{n\times 1}$, the vector $V\vec{p}$ can be approximated by $u^*(x_i)$, within the error bound $|| \vec{u}^* - \vec{u} || \leq \epsilon$ (for any given $\epsilon > 0$), at the cost of performing $L(4n-2)\beta + O(\beta n \log n)$ arithmetic operations, for L satisfying (2.15). In particular, if $q < 1$, $\log(\alpha\beta b/\epsilon) = O(\log n)$, for q, α, and b defined as in propositions 2.1 and 2.3, then, the cost bound is $O(n\beta \log n)$.*

2.7 Multiplication of Vector by Chebyshev-Vandermonde-like Matrix

Definition 2.5 (compare [GO1]). *For $n = 0, 1, 2, \cdots$, recursively define Chebyshev polynomials of the first kind $T_n(x)$ and Chebyshev polynomials of the second kind $U_n(x)$ as follows:*

$$\begin{aligned} T_0(x) &= 1, \qquad T_1(x) = x \ , \\ T_n(x) &= T_{n-1}(x) - T_{n-2}(x) \ , \end{aligned}$$

$$\begin{aligned} U_0(x) &= 1, \qquad U_1(x) = 2x \ , \\ U_n(x) &= 2xU_{n-1}(x) - U_{n-2}(x) \ . \end{aligned}$$

Define the Chebyshev-Vandermonde matrices as follows:

$$V_T(\vec{x}) = \begin{pmatrix} T_0(x_0) & T_1(x_0) & \cdots & T_{n-1}(x_0) \\ T_0(x_1) & T_1(x_1) & \cdots & T_{n-1}(x_1) \\ \vdots & \vdots & \vdots & \vdots \\ T_0(x_{n-1}) & T_1(x_{n-1}) & \cdots & T_{n-1}(x_{n-1}) \end{pmatrix},$$

$$V_U(\vec{x}) = \begin{pmatrix} U_0(x_0) & U_1(x_0) & \cdots & U_{n-1}(x_0) \\ U_0(x_1) & U_1(x_1) & \cdots & U_{n-1}(x_1) \\ \vdots & \vdots & \vdots & \vdots \\ U_0(x_{n-1}) & U_1(x_{n-1}) & \cdots & U_{n-1}(x_{n-1}) \end{pmatrix}.$$

Fact 2.1 *The vectors* $V_T(\vec{x})\ \vec{p}$ *and* $V_U(\vec{x})\ \vec{p}$, *for* 2 *Chebyshev-Vandermonde matrices* $V_T(\vec{x})$ *and* $V_U(\vec{x})$, *are the vectors of the values of the* 2 *polynomials,*

$$P_T(x) = \sum_{k=0}^{n-1} p_k T_k(x) \qquad \text{and} \qquad P_U(x) = \sum_{k=0}^{n-1} p_k U_k(x)\ , \tag{2.16}$$

respectively, at the points $x_0, \cdots, x_{n-1}$.

[KO] defines linear operators associated with Chebyshev-Vandermonde matrices $V_T(\vec{x})$ and $V_U(\vec{x})$, which leads to operator representation of these matrices in the form GB^T for $G \in C^{n\times\alpha}$, $B \in C^{n\times\alpha}$, $\alpha \leq 2$. Having applied discrete Fourier transform, we obtain Cauchy-like matrices $V_T(\vec{x})F^*$ and $V_U(\vec{x})F^*$ (cf. [KO]). Then, application of the techniques of the previous section gives us fast algorithms for multipoint approximation of the polynomial $P_T(x)$ and $P_U(x)$ of (2.16). We will next specify this approach for the classes of polynomial $P_T(x)$, Chebyshev-Vandermonde matrices $V_T(\vec{x})$, and their extension to Chebyshev-Vandermonde-like matrices. We will omit the similar treatment of polynomial $P_U(x)$, matrices $V_U(\vec{x})$, and their Chebyshev-Vandermonde-like extension.

Definition 2.6 (compare [KO]). *Given a matrix* $R \in C^{n\times n}$, *such that*

$$\nabla_{(D_{1/\vec{x}},W)}(R) = GB^T,\ for\quad G,\ B \in C^{n\times\alpha}\ , \tag{2.17}$$

where

$$W = 2\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor}(-1)^{i-1}\ (Z_0^T)^{2i-1}, \qquad \text{and} \qquad Z_0 = \begin{pmatrix} 0 & \cdots & 0 & 0 & 0 \\ 1 & \cdots & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 & 0 \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix},$$

then R is called a Chebyshev-Vandermonde-like matrix, the pair of matrices $\{G, B\}$ is called a $(D_{1/\vec{x}}, W)$-generator of R of length α (not uniquely defined for any R and α), and the smallest length α over all possible $(D_{1/\vec{x}}, W)$-generators is called the $(D_{1/\vec{x}}, W)$-rank of R.

At the end of this section, we will show that the matrix $R = V_T(\vec{x})$ satisfies (2.17) for $\alpha \leq 2$. The following lemma states how a Chebyshev-Vandermonde-like matrix can be transformed into a Cauchy-like matrix:

Lemma 2.1 (compare [KO]). *Let a matrix R be given by its ($2D_x, Z_1 + Z_1^T$)-generator, so that*

$$\nabla_{(2D_{\vec{x}}, Z_1 + Z_1^T)}(R) = 2D_{\vec{x}}R - R(Z_1 + Z_1^T) = GB^*$$

for $G,\ B \in C^{n \times \alpha}$, then RF^* is a Cauchy-like matrix such that

$$\nabla_{(2D_{\vec{x}}, D_{cos})}(RF^*) = 2D_{\vec{x}}(RF^*) - (RF^*)D_{cos} = GH^*,$$

where

$$G = [\ \vec{g}_1, \cdots, \vec{g}_\alpha\ ], \quad H = [\ \vec{h}_1^*, \cdots, \vec{h}_\alpha^*\ ] = FB, \quad D_{\vec{x}} = diag(\ x_0, \cdots, x_{n-1}\ )\ ;$$

Z_1, F^*, and F are defined as above, and

$$D_{cos} = diag\ (\ 2,\ 2cos(\tfrac{\pi}{n}),\ 2cos(\tfrac{2\pi}{n}),\ \cdots,\ 2cos(\tfrac{(n-1)\pi}{n})\ )\ .$$

Next, we will approximate the vector $R\ \vec{p}$, where $\vec{p} \in C^{n\times 1}$, R is an $n \times n$ Chebyshev-Vandermonde-like matrix. We will apply the reduction to Cauchy-like matrices given by lemma 2.1. We have the following equations [GO]:

$$\begin{aligned} RF^{*} &= \sum_{m=1}^{\alpha} diag(g_m)\ C(2\vec{x},\vec{t}\,)\ diag(h_m), \\ R\ \vec{p} &= \sum_{m=1}^{\alpha} diag(g_m)\ C(2\vec{x},\vec{t}\,)\ diag(h_m)\ F\ \vec{p}\,, \end{aligned}$$

where $\vec{x} = (x_i)_{i=1}^{n}$, $\vec{t} = (t_j)_{j=1}^{n}$ and $t_j = 2cos(\frac{j-1}{n}\pi)$.

Denote $\vec{v}_m = diag(h_m)\ F\ \vec{p} = [\ v_{m,0}, \cdots, v_{m,n-1}\ ]$, $m = 1, 2, \cdots, \alpha$.

Then $O(n\alpha \log n)$ arithmetic operations suffice in order to compute $\vec{v}_m$ for all m. Since

$$C(2\vec{x},\vec{t}\,)\ \vec{v}_m = \begin{pmatrix} \sum_{k=0}^{n-1} \frac{v_{m,k}}{2x_0 - t_k} \\ \vdots \\ \sum_{k=0}^{n} \frac{v_{m,k}}{2x_{n-1} - t_k} \end{pmatrix},$$

we have

$$\begin{aligned} R\ \vec{p} &= \sum_{m=1}^{\alpha} diag(g_m)\ C(2\vec{x},\vec{t}\,)\ \vec{v}_m \\ &= \begin{pmatrix} \sum_{m=1}^{\alpha} g_{m,0} \sum_{k=0}^{n-1} \frac{v_{m,k}}{2x_1 - t_k} \\ \vdots \\ \sum_{m=1}^{\alpha} g_{m,n-1} \sum_{k=0}^{n-1} \frac{v_{m,k}}{2x_{n-1} - t_k} \end{pmatrix} = \begin{pmatrix} u(x_0) \\ \vdots \\ u(x_{n-1}) \end{pmatrix}, \end{aligned}$$

where

$$\vec{g}_m = [\ g_{m,0}, \cdots, g_{m,n-1}\ ]^T$$

, and

$$u(x_i) = \sum_{m=1}^{\alpha} g_{m,i} \sum_{k=0}^{n-1} \frac{v_{m,k}}{2x_i - t_k}.$$

If $|\frac{2x_i}{t_k}| < q < 1$ for all i and k, then, we have

$$u(x_i) = -\sum_{m=1}^{\alpha} g_{m,i} \sum_{k=0}^{n-1} \frac{v_{m,k}}{t_k} \sum_{j=0}^{\infty} (\ \frac{2x_i}{t_k}\ )^j = -\sum_{m=1}^{\alpha} g_{mi} \sum_{j=0}^{\infty} A_{j,m} (2x_i)^j\ ,$$

where

$$A_{j,m} = \sum_{k=0}^{n-1} \frac{v_{m,k}}{t_k^{j+1}} .$$

Let

$$A = \max_{m,i} |g_{m,i}|, \qquad b = \max_{m,k} |\frac{v_{m,k}}{t_k}| , \tag{2.18}$$

and

$$u^*(x_i) = - \sum_{m=1}^{\alpha} g_{m,i} \sum_{j=0}^{L-1} A_{j,m} (2x_i)^j .$$

Then, we have

$$E_L(x_i) = \mid u(x_i) - u^*(x_i) \mid = \mid \sum_{m=1}^{\alpha} g_{m,i} \sum_{j=L}^{\infty} A_{j,m} (2x_i)^j \mid \leq \frac{\alpha A n b q^L}{1-q} \leq \epsilon ,$$

for any

$$L \geq \lceil \frac{\log(\alpha A n b / ((1-q)\epsilon))}{\log(1/q)} \rceil . \tag{2.19}$$

Now, we estimate the cost of computing the approximations $u^*(x_i)$ to $u(x_i)$, $i = 0, 1, \cdots, n-1$, where

$$u^*(x_i) = - \sum_{m=1}^{\alpha} g_{m,i} \sum_{j=0}^{L-1} A_{j,m} (2x_i)^j .$$

Computing $A_{j,m}$ requires $(2n-1)L\alpha$ arithmetic operations for all j and m, computing

$$g_{m,i} \sum_{j=0}^{L-1} A_{j,m} (2x_i)^j$$

takes on $n(2L-2)$ arithmetic operations for all i, and therefore, the overall cost of computing the vector $[\ u^*(x_0), \cdots, u^*(x_{n-1})\ ]^T$, which approximates the vector $R\ \vec{p}$, is

$$\alpha L(2n-2) + \alpha n(2L-1) + n(\alpha - 1) + O(n\alpha \log n) + n\alpha \ = 4\alpha n L + O(n\alpha \log n) .$$

This is $O(n\alpha \log n)$, for $L = O(\log n)$, and we may choose $L = O(\log n)$ satisfying (2.19) if $q < 1$ and $\log(\alpha A b/\epsilon) = O(\log n)$. We have proved the following proposition:

Proposition 2.5 *Given an $n \times n$ Chebyshev-Vandermonde-like matrix R and a vector*

$\vec{p} \in C^{n\times 1}$, the vector $R\,\vec{p}$ can be approximated by the vector

$$\vec{u}^{\,*}(x) = [\, u^*(x_0), \cdots, u^*(x_{n-1}) \,]^T$$

such that $\mid u(x_i) - u^*(x_i) \mid \leq \epsilon$ for all i and for a fixed $\epsilon > 0$. The arithmetic complexity of computing such an approximation vector is bounded by $O(\alpha n \log n)$, if $q < 1$ and if $\log(\alpha A n b/\epsilon) = O(\log n)$, where A and b are defined by (2.18) and q is a fixed constant. $q \geq x_+ = \max_i |x_i|$, as in propositions 2.1 and 2.3.

If $R = V_T(x)$ is a Chebyshev-Vandermonde matrix, then

$$\nabla_{(2D_{\vec{x}}, Z_1 + Z_1^T)}(R) = GB^T\ ,$$

where

$$G = \begin{pmatrix} x_0 - T_{n-2}(x_0) & T_{n-1}(x_0) - 1 \\ \vdots & \vdots \\ x_{n-1} - T_{n-2}(x_{n-1}) & T_{n-1}(x_{n-1}) - 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 1 \end{pmatrix}\ .$$

Therefore, in this case, we have $\alpha = 2$,

$$H = FB = \begin{pmatrix} 1 & 1 \\ 1 & e^{\frac{2\pi\sqrt{-1}}{n}(n-1)} \\ \vdots & \vdots \\ 1 & e^{\frac{2\pi\sqrt{-1}}{n}(n-1)^2} \end{pmatrix},\ \vec{v}_m = diag(h_m)\, F\, \vec{p} = diag(h_m) \begin{pmatrix} b_0^* \\ \vdots \\ b_{n-1}^* \end{pmatrix},$$

$$\vec{v}_1 = [\, b_0^*, \cdots, b_{n-1}^* \,]^T, \qquad \vec{v}_2 = [\, b_0^*, \cdots, b_{n-1}^* e^{\frac{2\pi\sqrt{-1}}{n}(n-1)^2} \,]^T.$$

Thus, we have proved the following result:

Corollary 2.2 *Given an $n \times n$ Chebyshev-Vandermonde matrix $V_T(\vec{x})$ and a vector*

$\vec{p} \in C^{n \times 1}$, let q be a fixed constant,

$$q \geq x_+ = \max_i |x_i| \ , \qquad b = \max_{m,k} | \ \frac{v_{m,k}}{t_k} \ | \ ,$$

and $A = \max(\ | \ x_i - T_{n-1}(x_i) \ |, \ | \ T_n(x_i) - 1 \ | \ )$ for $0 \leq i \leq n-1$. *Then, the vector $V_T(\vec{x})\ \vec{p}$ can be approximated by a vector $\vec{u}^*(\vec{x})$ such that*

$$| \ u(x_i) - u^*(x_i) \ | \leq \epsilon \qquad \textit{for a given } \epsilon > 0 \textit{ and for all } i \ ,$$

and $O(\ n(L + \log n)\ )$ arithmetic operations suffice in order to compute $u^*(x_i)$,

$i = 0, 1, \cdots, n-1$, where

$$L = \lceil \ \frac{\log(\ Abn/(\ (1-q)\epsilon \ ) \ )}{\log(1/q)} \ \rceil \ .$$

Furthermore, $L = O(\log n)$, so that $O(n \log n)$ arithmetic operations suffice for the evaluation of $u^*(x_i)$, $i = 0, 1, \cdots, n-1$, if $q < 1$ and if $\log(Ab/\epsilon) = O(\log n)$.

2.8 Numerical experiments

The algorithms find approximated results by using lower computational complexity. In practical numeral computation, approximate algorithms are often found to be more favorable than some of the exact algorithms, since the later may be affected by some practical limitations, such as limited floating point precision and finite memory resources. The following numerical experiment results demonstrate the behavior of the implementations of two multipoint

polynomial evaluation algorithms, the one proposed by Moenck and Borodin [BP], and the one introduced in section 2.3 of this chapter.

The implementations of the two algorithms are coded in C++ and the experiment programs were compiled and linked using Microsoft Visual C++. The programs were executed on a 386 and a Pentium PC and identical results were obtained from both platforms. Arithmetic functions are performed by Microsoft FP87 emulation library functions without utilizing a math coprocessor. In the following three cases, algorithm 1 denotes the one by Moenck and Borodin [BP] and algorithm 2 denotes the one proposed in section 2.3. Direct computation results are given in all cases as references. All the input, output and intermediate results are stored in double precision binary representation. For algorithm 2, $L = 40$ was used for all the cases to obtain the results with desired precision.

Experiment 2.1 A simple example shows how the exact algorithm fails in a somewhat extreme case where two adjacent input values vary widely. The coefficients of the polynomial and the input values are chosen in a way that the results can be easily verified. The detail steps of Moenck-Borodin algorithm are given. Notice that in Step 2 and step 3, due to limited precision, the least significant digits of the intermediate results are lost and greatly affects the reliability of the final output. ie., two identical input, $x_0 = x_2 = 1.5$, actually result in different output, $v_0 = 8.125$ and $v_2 = 6$, because of the adjacent input value.

Polynomial:

$$P(x) = x^3 + x^2 + x + 1$$

Input vector:

$$\mathrm{x} = (1.5 \quad 20 \quad 1.5 \quad 1 \times 10^8)$$

Direct Computation Output:

$$\mathrm{v} = (8.125 \quad 8421 \quad 8.125 \quad 1 \times 10^{24})$$

Algorithm 1 Output:

Step 1: Compute coefficient vectors for $P_0(x) = (x - x_0)(x - x_1)$ and $P_1(x) = (x - x_2)(x - x_3)$:

$$\mathrm{p}_0 = (1 \quad 21.5 \quad 30); \quad \mathrm{p}_1 = (1 \quad 1.000000015 \times 10^8 \quad 1.5 \times 10^8)$$

Step 2: Compute $M_i(x) = P(x) \bmod P_i(x)$, $i = 0, 1$.

$$\mathrm{m}_0 = (469.75 \quad 616); \quad \mathrm{m}_1 = (1.000000215 \times 10^{16} \quad 1.500000075 \times 10^{16})$$

Step 3: Compute output vector v, where $v_i = M_{\lfloor \frac{i}{2} \rfloor}(x) \bmod (x - x_i)$, $i = 0, 1, 2, 3$:

$$\mathrm{v} = (8.125 \quad 8421 \quad 6 \quad 1 \times 10^{24})$$

Algorithm 2 Output:

$$\mathrm{v} = \begin{pmatrix} (8.125 + 1.99483 \times 10^{-16} i) \\ (8421 + 1.52282 \times 10^{-12} i) \\ (8.125 + 1.99483 \times 10^{-16} i) \\ (1 \times 10^{24} + 2.00008 \times 10^{8} i) \end{pmatrix}^T$$

Experiment 2.2 The example in Experiment 2.1 may seem unusual because of the fact that input values varies very widely. However, in polynomials with higher degrees, similar results may occur even when the differences among the inputs are of much smaller scale. The following example selects

Input	Direct	Algorithm 1
1.5	1311.68	$2.19902 \times 10^{+012}$
100	$1.0101 \times 10^{+030}$	$1.0101 \times 10^{+030}$
2	65535	$4.29497 \times 10^{+009}$
3	$2.15234 \times 10^{+007}$	$1.07374 \times 10^{+010}$
4	$1.43166 \times 10^{+009}$	$1.42954 \times 10^{+009}$
5	$3.8147 \times 10^{+010}$	$3.81427 \times 10^{+010}$
6	$5.64222 \times 10^{+011}$	$5.64214 \times 10^{+011}$
7	$5.53882 \times 10^{+012}$	$5.53881 \times 10^{+012}$
8	$4.02107 \times 10^{+013}$	$4.02107 \times 10^{+013}$
9	$2.31628 \times 10^{+014}$	$2.31628 \times 10^{+014}$
10	$1.11111 \times 10^{+015}$	$1.11111 \times 10^{+015}$
11	$4.59497 \times 10^{+015}$	$4.59497 \times 10^{+015}$
12	$1.68077 \times 10^{+016}$	$1.68077 \times 10^{+016}$
13	$5.54514 \times 10^{+016}$	$5.54514 \times 10^{+016}$
14	$1.67535 \times 10^{+017}$	$1.67535 \times 10^{+017}$
15	$4.69172 \times 10^{+017}$	$4.69172 \times 10^{+017}$

Table 2.1: Algorithm 1 output of Experiment 2.2

a simple polynomial of degree 15 and a set of 16 input values range from 1.5 to 100, notice how more than one of the output values of algorithm 1 are corrupted because of an adjacent input of larger magnitude.

Polynomial:

$$\begin{aligned} P(x) &= x^{15}+x^{14}+x^{13}+x^{12}+x^{11}+x^{10}+ \\ &\quad x^9+x^8+x^7+x^6+x^5+x^4+x^3+x^2+x+1 \end{aligned}$$

Input vector:

x = (1.5 100 2 3 4 5 6 7 8 9 10 11 12 13 14 15)

Output:

See table 2.1 and 2.2.

Experiment 2.3 Similar to Experiment 2.2 except that the polynomial coefficients has been arbitrarily selected.

Input	Direct	Algorithm 2
$(1.5\ 100\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)^T$	$(1311.68\ 1.0101 \times 10^{+030}\ 65535\ 2.15234 \times 10^{+007}\ 1.43166 \times 10^{+009}\ 3.8147 \times 10^{+010}\ 5.64222 \times 10^{+011}\ 5.53882 \times 10^{+012}\ 4.02107 \times 10^{+013}\ 2.31628 \times 10^{+014}\ 1.11111 \times 10^{+015}\ 4.59497 \times 10^{+015}\ 1.68077 \times 10^{+016}\ 5.54514 \times 10^{+016}\ 1.67535 \times 10^{+017}\ 4.69172 \times 10^{+017})^T$	$(1311.68 - 3.0773 \times 10^{-13}i\ 1.0101 \times 10^{+030} + 1.4558 \times 10^{+015}i\ 65535 + 3.94771 \times 10^{-12}i\ 2.15234 \times 10^{+007} + 9712240000i\ 1.43166 \times 10^{+009} + 96895500i\ 3.8147 \times 10^{+010} + 313546i\ 5.64222 \times 10^{+011} + 0.000520682i\ 5.53882 \times 10^{+012} + 0.00552181i\ 4.02107 \times 10^{+013} + 0.0423665i\ 2.31628 \times 10^{+014} + 0.25441i\ 1.11111 \times 10^{+015} + 1.26063i\ 4.59497 \times 10^{+015} + 5.35069i\ 1.68077 \times 10^{+016} + 19.9938i\ 5.54514 \times 10^{+016} + 67.1483i\ 1.67535 \times 10^{+017} + 205.96i\ 4.69172 \times 10^{+017} + 584.302i)^T$

Table 2.2: Algorithm 2 output of Experiment 2.2

Polynomial:

$$P(x) = 2.1x^{15} + x^{14} + 10x^{13} + 11x^{12} + 12x^{11} + 15x^{10} + 1.4x^9 + 5x^8 + 6x^7 + 7x^6 + 8x^5 + 6.3x^4 + 1.2x^3 + 3x^2 + 3.3x + 2$$

Input vector:

x = (1.5 100 2 3 50 5 30 7 20 9 10 70 77 55 42 15)

Output:

See table 2.3 and 2.4.

Input	Direct	Algorithm 1
$(1.5\ 100\ 2\ 3\ 50\ 5\ 30\ 7\ 20\ 9\ 10\ 70\ 77\ 55\ 42\ 15)^T$	$(6962.47\ 2.11101 \times 10^{30}\ 255709\ 5.97974 \times 10^{7}\ 6.48221 \times 10^{25}\ 8.58207 \times 10^{10}\ 3.07765 \times 10^{22}\ 1.17973 \times 10^{13}\ 7.13181 \times 10^{19}\ 4.84203 \times 10^{14}\ 2.31235 \times 10^{15}\ 1.00475 \times 10^{28}\ 4.19381 \times 10^{28}\ 2.70455 \times 10^{26}\ 4.75383 \times 10^{24}\ 9.69772 \times 10^{17})^T$	$(4.39805 \times 10^{12}\ 2.11101 \times 10^{30}\ 1.28849 \times 10^{10}\ 2.57698 \times 10^{10}\ 6.48221 \times 10^{25}\ 8.58993 \times 10^{10}\ 3.07765 \times 10^{22}\ 1.17973 \times 10^{13}\ 7.13181 \times 10^{19}\ 4.84232 \times 10^{14}\ 2.31282 \times 10^{15}\ 1.00475 \times 10^{28}\ 4.19381 \times 10^{28}\ 2.70455 \times 10^{26}\ 4.75383 \times 10^{24}\ 9.69777 \times 10^{17})^T$

Table 2.3: Algorithm 1 output of Experiment 2.3

Input	Direct	Algorithm 2
$(1.5\ 100\ 2\ 3\ 50\ 5\ 30\ 7\ 20\ 9\ 10\ 70\ 77\ 55\ 42\ 15)^T$	$(6962.47\ 2.11101 \times 10^{30}\ 255709\ 5.97974 \times 10^{7}\ 6.48221 \times 10^{25}\ 8.58207 \times 10^{10}\ 3.07765 \times 10^{22}\ 1.17973 \times 10^{13}\ 7.13181 \times 10^{19}\ 4.84203 \times 10^{14}\ 2.31235 \times 10^{15}\ 1.00475 \times 10^{28}\ 4.19381 \times 10^{28}\ 2.70455 \times 10^{26}\ 4.75383 \times 10^{24}\ 9.69772 \times 10^{17})^T$	$(6962.47 - 8.45502 \times 10^{-12}i\ (2.11101 \times 10^{30} - 7.9277 \times 10^{15}i\ (255709 - 3.54152 \times 10^{-10}i\ (5.97974 \times 10^{7} - 1.08648 \times 10^{-7}i\ (6.48221 \times 10^{25} - 2.39968 \times 10^{11}i\ (8.58207 \times 10^{10} - 0.000217322i\ (3.07765 \times 10^{22} - 1.11633 \times 10^{8}i\ (1.17973 \times 10^{13} - 0.0343946i\ (7.13181 \times 10^{19} - 251662i\ (4.84203 \times 10^{14} - 1.5167i\ (2.31235 \times 10^{15} - 7.41649i\ (1.00475 \times 10^{28} - 3.75053 \times 10^{13}i\ (4.19381 \times 10^{28} - 1.56835 \times 10^{14}i\ (2.70455 \times 10^{26} - 1.00389 \times 10^{12}i\ (4.75383 \times 10^{24} - 1.74988 \times 10^{10}i\ (9.69772 \times 10^{17} - 3321.8i)^T$

Table 2.4: Algorithm 2 output of Experiment 2.3

Chapter 3

Modular (Residue) Arithmetic for Linear Algebra Computations in the Real Field

3.1 Overview

In this chapter, we present the application our combined approaches to decreasing the precision of some important computations in linear algebra. By using fewer bits for representing the numbers involved in the computations. we decrease the time-complexity and the space-complexity of the computations without affecting the output precision.

To take advantage of decreasing the precision of computation, we need, of course, a computer (such as MASPAR) that performs faster if the computation goes with a lower precision. Due to the recent progress in the data compression area, we should expect that more and more computers of this kind will be around. We also recall the recent specific progress in data compression for basic matrix operations [P,a], [P91], [P92], [BP], which implies a

possible acceleration of computations by the factor of the order P/b, showing the ratio of the single machine precision, P, and of the actually needed bit-precision of the operand, b.

We rely on the observation that some major computations in linear algebra (such as the solution of descretized PDEs by means of multigrid methods and iterative improvement of an approximate solution to a linear system of equations, where the input coefficients are represented with fewer bits) involve inner products whose magnitudes are substantially less than the magnitudes of some coordinates of the two input vectors. Then, in many cases, we may chop the most significant digits in the representation of these coordinates (and thus decrease the precision of the computations) without affecting the output errors. Such an idea must be counter-intuitive for a numerical analyst, who views the cancellation of the most significant digits of the operands as a major disaster of numerical computing, because of the implied contamination of the output. Furthermore, many numerical analysts believe that the usual scheme for iterative improvement of approximate solution to the linear system of equations cannot produce correct output unless the residual vector is computed with double precision. Our techniques enable us to compute the correct solution by performing the computations of this algorithm with a precision, which is substantially *smaller* than the *single* machine precision.

On the other hand, the power of our approach should be less surprising to the designers of algebraic algorithms, who are familiar with using reduction modulo an integer m as a common means of decreasing the precision of computations [G],[YG]. Unlike the previous works, however, we were able

to utilize this approach within some customary schemes of numerical computing. To achieve this, we had to combine the reduction modulo m with numerical techniques of rounding-off (in order to truncate the least significant digits too) which involved reduction modulo noninteger positive m. Furthermore, we had to recognize which of the most significant bits of the operand did not affect the output and to recover the output after the deletion of these bits of the operands. To achieve all this, we have introduced new techniques which we call *backward modular reduction* (b.m.r) and *backward binary segmentation* (b.b.s.). The b.b.s. combines the algebraic techniques of b.m.r. with the customary numerical techniques of truncation of the least significant bits. Both b.m.r and b.b.s. require to estimate the range in which we may truncate the operands depending on the estimated magnitudes of the output values and of their allowed approximation errors. (The analysis goes from the output values back to the operands, thus motivating the adjectives "backward".)

Besides their applications to computations in linear algebra, our modular reduction techniques can be useful in some other numerical computations, for instance (as suggested by a referee), for calculating some special functions to a limited precision.

In the next two sections, we apply the algebraic techniques of backward modular reduction to the summation and the computation of the inner product of two vectors. In section 3.4, we complement these techniques with the numerical techniques of truncation (chopping) and apply the resulting b.b.s. process to the computation of the inner product. We combine b.b.s. with a modification of the iterative improvement algorithm presented in its gener-

alized version in section 3.5. In section 3.6, we give an example of further extensions by applying the b.b.s. process to the Gauss-Seidel iteration. We also show application of the b.b.s. techniques to the solution of PDE's by means of multigrid algorithms (see section 3.7 and compare [PR], [PR,a]). Section 3.8 contains the results of our numerical experiments.

3.2 Backward modular reduction for summation

Definition 3.1 *For a positive m and a real r, define the unique real number $r \bmod m$ such that $0 \leq r \bmod m < m$, $r = r \bmod m + jm$, for some integer j. Let $\mathbf{R}/m$ denote the additive group of real numbers reduced modulo m, that is, $0 \leq r^{(i)} \leq m$ if $r^{(i)} \in \mathbf{R}/m$, $i = 1, 2$, and in $\mathbf{R}/m$ the sum of $r^{(1)}$ and $r^{(2)}$ equals $(r^{(1)} + r^{(2)}) \bmod m$.*

Proposition 3.1 *For any pair of a positive m and a real $r \in \mathbf{R}/m$, we have*

$$r = \begin{cases} r \bmod m & \text{if } r \bmod m < m/2, \\ (r \bmod m) - m & \text{otherwise.} \end{cases}$$

We will apply modular reduction for $m = 2^h$, and for integers h, thus using binary representation of real numbers and binary logarithms, except that in our Examples 3.2, 3.3, and 3.4, we will let $m = 10^h$ which should show that the decimal case, and more generally the b-ary case for any integer $b \geq 2$, can be treated similarly.

Now suppose that h bits is a single precision allowed when we compute the sum $r = \sum_{i=1}^{k} r^{(i)}$ of k integers $r^{(1)}, \cdots, r^{(k)}$ and that

$$2|r| < m = 2^h \tag{3.1}$$

though $|r|^{(i)} > 2^h$ for some i, $1 \leq i \leq k$. Then, we may still obtain r by computing modulo m, with a single precision of h bits, and by applying Proposition 3.1 at the end. Furthermore, we may achieve the same goal when we work with rational summands $r^{(1)} \cdots, r^{(k)}$, where the sum r has the form

$$r = 2^d z, \ 2|z| < m, \tag{3.2}$$

for 2 integers d and z.

Example 3.1 Let $k = 3$,

$$\begin{aligned} r^{(1)} &= (0.1111111111111)_2, \\ r^{(2)} &= -(0.010101010111)_2, \\ r^{(3)} &= -(0.1010101010111)_2, \end{aligned}$$

Then, subtractions with the precision of 13 bits give us

$$r = r^{(1)} + r^{(2)} + r^{(3)} = -(0.11)_2 2^{-10} .$$

On the other hand, knowing in advance that $2|r| < 2^{-9}$, we may set $m = 2^{-9}$ and compute $r^{(i)} \bmod m$ and $r \bmod m$ in $\mathbf{R}/m$, with 4-bit precision, as follows:

$$\begin{aligned} r^{(1)} \bmod m &= (0.1111)_2 2^{-9}, \\ r^{(2)} \bmod m &= (0.001)_2 2^{-9}, \\ r^{(3)} \bmod m &= (0.1001)_2 2^{-9}, \end{aligned}$$

$$r \bmod m = ((r^{(1)} \bmod m) + (r^{(2)} \bmod m) + (r^{(3)} \bmod m)) \bmod m$$

$$= ((0.1111)_2 2^{-9} + (0.001)_2 2^{-9} + (0.1001)_2 2^{-9}) \bmod m$$
$$= (1.101)_2 2^{-9} \bmod m$$
$$= (0.101)_2 2^{-9}.$$

Since

$$r \bmod m = (0.101)_2 2^{-9} > m/2,$$

Proposition 3.1 implies that

$$r = -m + r \bmod m =$$

$$-2^{-9} + (0.101)_2 2^{-9} = -(0.011)_2 2^{-9} = -(0.11)_2 2^{-10}\ .$$

In the next example we use the decimal representation.

Example 3.2 Let

$$k = 3,\ r^{(1)} = 3.1416048,\ r^{(2)} = -2.718288,\ r^{(3)} = -0.4233216.$$

Then subtractions with 8 decimals give us $r = -(0.48)10^{-5}$. However, knowing in advance that $|r| < (0.5)10^{-5}$, we may set $m = 10^{-5}$ and perform the computation with 3 decimals:

$r^{(1)} \bmod m = (0.48)10^{-5}$,

$r^{(2)} \bmod m = (0.2)10^{-5}$,

$r^{(3)} \bmod m = (0.84)10^{-5}$,

$r \bmod m = ((r^{(1)} \bmod m) + (r^{(2)} \bmod m) + (r^{(3)} \bmod m)) \bmod m =$

$((1.52)10^{-5}) \bmod m =$

$(0.52)10^{-5}$.

Apply Proposition 3.1 and obtain

$$r = -m + (r \bmod m) = -(0.48)10^{-5}.$$

We will use the name *backward modular reduction (b.m.r.)* for the techniques applied in Examples 3.1 and 3.2 since it extends the reduction modulo m *backward*, from the sum r to the summands $r^{(i)}$.

3.3 Backward modular reduction for the computation of the inner product of two vectors

Computation of the inner product

$$r = \sum_{i=1}^{k} r^{(i)}, \quad r^{(i)} = u^{(i)} v^{(i)},$$

of two vectors

$$\vec{v} = (v^{(i)}), \quad \vec{u} = (u^{(i)}), \quad i = 1, \cdots, k,$$

is a basic operation of linear algebra.

To compute r, we may first multiply $u^{(i)}$ and $v^{(i)}$ pairwise, for all i, and then sum the products. Knowing in advance a bound (3.1) on $|r|$, we may apply b.m.r. at the summation stage. Let us extend b.m.r. also to the multiplication stage provided that, in addition to (3.1), we have

$$u^{(i)} = 2^{c(i)} z^{(i)}, \qquad v^{(i)} = 2^{d(i)} w^{(i)}, \tag{3.3}$$

where $c(i)$, $d(i)$, $z^{(i)}$ and $w^{(i)}$ are given integers, $i = 1, \ldots, k$. Then

$$\begin{aligned} r^{(i)} \bmod 2^h &= u^{(i)} v^{(i)} \bmod 2^h \\ &= ((u^{(i)} \text{ trunc } 2^{h-d(i)})(v^{(i)} \text{trunc } 2^{h-c(i)})) \bmod 2^h, \end{aligned} \tag{3.4}$$

where

$$x \text{ trunc } m = \begin{cases} x \bmod m & if\ x \geq 0 \\ -((-x) \bmod m) & \text{otherwise.} \end{cases} \tag{3.5}$$

Thus, x trunc 2^g denotes the value of x with its leftmost bits truncated up to the bit corresponding to 2^g. Due to (3.4), we may reduce $u^{(i)}$ or $-u^{(i)}$ modulo $2^{h-d(i)}$ and $v^{(i)}$ or $-v^{(i)}$ modulo $2^{h-c(i)}$ when we compute $r^{(i)}$ mod 2^h. That is, we again extend the modular reduction *backwards*, this time from the product $r^{(i)}$ to the multiplicands $u^{(i)}$ and $v^{(i)}$. If $|u^{(i)}| \geq 2^{h-d(i)}$ and/or $|v^{(i)}| \geq 2^{h-c(i)}$, then such a backward modular reduction decreases the binary lengths of $u^{(i)}$ and/or $v^{(i)}$. The next example demonstrates these techniques in the decimal case.

Example 3.3 Let $k = 3$, $u^{(1)} = 0.4176$, $v^{(1)} = 7.523$, $u^{(2)} = 1.8877$, $v^{(2)} = -1.44$, $u^{(3)} = 1.248$, $v^{(3)} = -0.3392$. Then $r^{(1)}$, $r^{(2)}$, $r^{(3)}$, and r take on the same values as in Example 3.2. so that $m = 10^{-5}$, $h = -5$. We next represent $u^{(i)}$ and $v^{(i)}$ as in (3.3):

$u^{(1)} = (10^{-4})4176, \quad v^{(1)} = (10^{-3})7523,$

$u^{(2)} = (10^{-4})18877, \quad v^{(2)} = -(10^{-2})144,$

$u^{(3)} = (10^{-3})1248, \quad v^{(3)} = -(10^{-4})3392,$

so that,

$c(1) = -4, \quad d(1) = -3,$

$c(2) = -4, \quad d(2) = -2,$

$c(3) = -3, \quad d(3) = -4.$

Now, by following (3.4) but replacing the powers of 2 by the powers of 10, we obtain that

$h - c(1) = -1, \; h - d(1) = -2,$

$u^{(1)}\text{trunc } 10^{-2} = (0.76)10^{-2}$, $v^{(1)}\text{trunc } 10^{-1} = (0.23)10^{-1}$,

$r^{(1)}\text{mod } 10^{-5} = ((0.1748)10^{-3}) \text{ mod } 10^{-5} = (0.48)10^{-5}$,

$h - c(2) = -4$, $h - d(2) = -3$,

$u^{(2)}\text{trunc } 10^{-3} = (0.7)10^{-3}$, $v^{(2)}\text{trunc } 10^{-1} = (0.6)10^{-1}$,

$r^{(2)}\text{mod } 10^{-5} = ((0.42)10^{-4}) \text{ mod } 10^{-5} = (0.2)10^{-5}$,

$h - c(3) = -2$, $h - d(3) = -1$,

$u^{(3)}\text{trunc } 10^{-1} = (0.48)10^{-1}$, $v^{(3)}\text{trunc } 10^{-2} = (0.8)10^{-3}$,

$r^{(3)}\text{mod } 10^{-5} = ((0.384)10^{-4}) \text{ mod } 10^{-5} = (0.84)10^{-5}$.

This gives us the same values of $r^{(i)} \text{ mod } m$ (for $i = 1, 2, 3$) as in Example 3.2.

3.4 Backward binary segmentation for the computation of the inner product

The algebraic technique of b.m.r. has enabled us to get rid of the most significant bits of the input and intermediate values in the computation of the inner products. Next, we will further decrease the precision of computing by adding the customary numerical techniques of truncating the least significant bits of the same values [A], [CdB].

Example 3.4 Assume that, for the same input as in Example 3.3, we should compute the inner product r on a computer that chops all values to 5 floating point decimal digits. If we apply the straightforward numerical algorithm, we obtain

$r^{(1)*} = f\ell((0.4176)(7.523)) = (0.31416)10^{1}$,

$r^{(2)*} = -f\ell((1.8877)(1.44)) = -(0.27182)10^{1}$,

$r^{(3)*} = -fl((1.248)(0.3392)) = -0.42332,$

$r^* = fl(r^{(1)*} + r^{(2)*} + r^{(3)*}) = (0.8)10^{-4}.$

Here, the rounding errors have completely contaminated the correct output value $r = -(0.48)10^{-5}$, which, however, can be correctly computed if we operate with the same number of bits but apply the b.m.r., as described in Examples 3.2 and 3.3.

The input values $u^{(i)}$ and $v^{(i)}$, $i = 1, 2, 3$, of Examples 3.2, 3.3, and 3.4 can be assumed to be real input values truncated to a few floating point decimals. Let us formalize and extend such a process of 2-side truncation, which accentuates the power of both of its elements, that is, of b.m.r. and customary numerical truncation.

Definition 3.2 *binary segments. Let g and h denote two integers, $g < h$; let q_i denote 0 or 1 for all i; let $S[g,h]$ denote the binary segment of real numbers can be represented as $\pm \sum_{i=g}^{h-1} q_i 2^i$, and let $S_{g,h}(q)$ denote the projection of a real number $q = \pm \sum_{i=-\infty}^{h-1} q_i 2^i$, $|q| < 2^h$, into the binary segment $S[g,h]$, that is, $S_{g,h}(q) = \pm \sum_{i=g}^{h-1} q_i 2^i$.*

Now, assume that the equations (3.3) hold, that, moreover,

$$|w^{(i)}| \leq 2^{a(i)}, \quad i = 1, \ldots, k, \tag{3.6}$$

for some fixed integers $a(i)$, and that $r = \sum_{i=1}^{k} u^{(i)} v^{(i)}$ satisfies the inequality

$$|r| + 2^t < 2^{h-1}. \tag{3.7}$$

for 2 fixed integers h and t. Then, consider the following algorithm for the inner product where we do not apply b.m.r to the $v^{(i)}$ values.

Algorithm 3.1

Input: *integers* $a(i)$, $d(i)$, h, *and* t, *a natural* k, *and real* $u^{(i)}$, $v^{(i)}$, $i = 1, \ldots, k$, *satisfying (3.3), (3.6), and (3.7).*

Output: *an approximation* r^* *to* r *such that*

$$|r^* - r| \leq 2^t. \tag{3.8}$$

Computations: *successively compute:*

1. $g = \lfloor \log(2^t/k) \rfloor - 1 = t - 1 - \lceil \log k \rceil$,
2. $\tilde{u}^{(i)} = S_{g-a(i)-d(i)}(u^{(i)} \bmod 2^{h-d(i)})$, $\quad i = 1, \ldots, k$,
3. $\tilde{r}^{(i)} = S_g((\tilde{u}^{(i)} v^{(i)}) \bmod 2^h)$, $\quad i = 1, \ldots, k$,
4. $\tilde{r} = \left(\sum_{i=1}^{k} \tilde{r}^{(i)}\right) \bmod 2^h$, *by performing additions in* $\mathbf{R}/2^h$,
5. $r^* = \begin{cases} \tilde{r} & \text{if } |\tilde{r}| < 2^{h-1} \\ \tilde{r} - 2^h & \text{otherwise.} \end{cases}$

The computation is performed with $(h-g+c)$-bit precision at stage 2 and with $(h-g)$-bit precision at stages 3 and 4. We will call this technique the *backward binary segmentation (b.b.s.)* process since it extends the output bound (3.7) backward, to the operands, restricting their binary values to certain segments.

Correctness proof. Let us deduce (3.8) so as to prove correctness of Algorithm 3.1. We would have had $\tilde{r} = r \bmod 2^h$ [due to (3.4)], if we excluded chopping by replacing g by $-\infty$ in stages 2 and 3, and then we would have had $r^* = r$. To deduce (3.8), it remains to estimate the errors due to chopping. For completeness we will supply these simple routine

estimates. In stage 2, chopping errors are less than $2^{g-a(i)-d(i)}$ (see definition 3.2). This bound turns into 2^g after multiplication of $\bar{u}^{(i)}$ by the integer $v^{(i)}$ since $|v^{(i)}| \leq 2^{a(i)+d(i)}$, due to (3.3) and (3.6). In stage 3 the latter error bound grows to 2^{g+1}, due to chopping. In stage 4, the k errors, each having its magnitude bounded by 2^{g+1}, are added with (or subtracted from) each other, which gives the overall error bound $k2^{g+1}$ for the approximation of $r \bmod 2^h$ by $\bar{r}$, and we observe that $2^t \geq k2^{g+1}$, for g defined at stage 1.

3.5 Application of b.b.s. to generalized iterative improvement algorithm

Let us apply our b.b.s. tools in order to bound the precision of computations in the well known algorithm [A], [W] for iterative improvement of a solution to a nonsingular linear system of n equations,

$$A\vec{x} = \vec{f}.$$

Hereafter, we will assume the matrix and vector norm $\|.\| = \|.\|_\infty$. Let the input consist of a vector $\vec{f}$, a pair of $n \times n$ matrices A and C (the latter approximating A^{-1}, so that

$$\|I - CA\|_\infty \leq 2^{-b} < 1, \tag{3.9}$$

for some fixed positive scalar b), and some initial vector $\vec{x}(0)$ (say, for $\vec{x}(0) = \vec{0}$). Then, the iterative improvement algorithm successively computes the vectors

$$\vec{r}(p) = \vec{f} - A\vec{x}(p-1), \tag{3.10}$$

$$\vec{e}(p) = C\vec{r}(p), \tag{3.11}$$

$$\vec{x}(p) = \vec{x}(p-1) + \vec{e}(p), \tag{3.12}$$

for $p = 1, 2, \ldots$. It can be easily shown that, for $p = 1, 2, \ldots$, we have

$$\begin{aligned} \vec{x} - \vec{x}(p) &= (I - CA)(\vec{x} - \vec{x}(p-1)) = (I - CA)^p(\vec{x} - \vec{x}(0)), \\ \vec{r}(p) &= A(\vec{x} - \vec{x}(p-1)), \\ \vec{e}(p) &= CA(\vec{x} - \vec{x}(p-1)). \end{aligned}$$

Therefore, if (3.9) holds, then $\vec{r}(p)$ and $\vec{e}(p)$ converge to $\vec{0}$ with the speed of a geometric progression, as $p \to \infty$. It is customary to use the double precision of computation at the stage (3.10) in order to ensure such a rapid convergence (see [GL], pp.126-127, or [A], pp467-471). We, however, observe that the precision of computing can be controlled and decreased by means of using the b.b.s. process. Indeed, apart from an addition and a subtraction of two pairs of vectors, the computation by (3.10) - (3.12) amounts to two multiplications of $n \times n$ matrices A and C by vectors, that is, to the evaluation of $2n$ inner products of $2n$ pairs of vectors. It remains

a) to show that the convergence of the iteration (3.10)-(3.12) with the speed of geometric progression will be preserved even if the vectors $\vec{r}(p)$ in (3.11) and $\vec{e}(p)$ in (3.12) are replaced by their numerical approximations,

$$\vec{r^*}(p) = \vec{r}(p) + \Delta\vec{r}(p), \quad \vec{e^*}(p) = \vec{e}(p) + \Delta\vec{e}(p), \tag{3.13}$$

respectively, for $p = 1, 2, \ldots$;

b) to bound the norms $||A\vec{x^*}(p)||$ and $||\vec{e^*}(p)||$ of the two vectors approximating the vectors $\vec{e}(p)$ and $A\vec{x}(p)$ in this process for $p = 1, 2, \cdots$.

Routine error analysis shows that requirement a) is satisfied, where

$$\|\Delta\vec{r}(p)\| \le 2^{g^*(p)}, \quad \|\Delta\vec{e}(p)\| \le 2^{g(p)}, \tag{3.14}$$

$$g^*(p) = g^* - bp, \quad g(p) = g - bp, \tag{3.15}$$

$p = 1, 2, \ldots$, for b of (3.9) and for some fixed scalars g and g^*.

$$\|\vec{r^*}(p)\| \le 2^{h^*(p)}, \quad \|\vec{e^*}(p)\| \le 2^{h(p)}, \tag{3.16}$$

$$h^*(p) = h^* + \lceil \log p \rceil - bp, \quad h(p) = h + \lceil \log p \rceil - bp, \tag{3.17}$$

$p = 1, 2, \ldots$. The relations (3.16) and (3.17) imply rapid convergence of $\|\vec{x} - \vec{x}(p)\|$ to 0, with the speed of a geometric progression. Next, we will assume that b, g, g^*, h and h^* have been precomputed, and then, we will satisfy the requirement b) above provided that for every pair (i, j) the entry $a_{i,j}$ of the input matrix A lies in a fixed binary segment $S[g_{ij}(A), h_{ij}(A)]$ of a moderately small length $h_{ij}(A) - g_{ij}(A)$, that is,

$$a_{i,j} = \pm \sum_{k=g_{i,j}(A)}^{h_{i,j}(A)-1} q_k 2^k,$$

where q_k takes values 0 and 1. Hereafter, $\ell_i(A)$ denotes $\max_j |h_{ij}(A) - g_{ij}(A)|$.

Remark 3.1 *The latter assumption about $a_{i,j}$ is needed in order to bound the precision of computing the product of A by $\vec{x}(p-1)$ in (3.10). This assumption holds, for instance, for many linear systems obtained by means of the discretization of linear PDE's with constant coefficients. Generally,*

if A is a well-conditioned matrix, we may decrease $h_{ij}(A) - g_{ij}(A)$ by chopping the entries of A and/or by applying the routine technique of algebraic segmentation, described in [EPY].

The above assumptions enable us to apply the b.b.s. process [based on Algorithm 3.1] in order to bound the precision of computations performed according to (3.10) and (3.11) with $\vec{r^*}(p)$ replacing $\vec{r}(p)$ in (3.11) and with $\vec{e}(p)$ and $\vec{r}(p)$ evaluated with error vectors $\Delta\vec{e}(p)$ and $\Delta\vec{r}(p)$ that satisfy (3.13)-(3.17).

Hereafter, let $c_i(W)$ denote the number of nonzero entries of row i of a matrix W. By applying the analysis from sections 3.2-3.4, whose tedious but straightforward elaboration we will omit, we arrive at the following bounds on the precision of the operands (for $p = 1, 2, \ldots$):

(a) in the representation of each operand of any addition or subtraction involved in the evaluation of the i-th component of $\vec{r}(p)$ [according to (3.10)], it suffices to use

$$d_i^*(p) = h^*(p) - g^*(p) + \lceil \log c_i(A) \rceil = h^* - g^* + \log p + \lceil \log c_i(A) \rceil \text{ bits;}$$

(b) in the representation of each component of $\vec{x}(p-1)$ when this component is multiplied by an entry of row i of A [according to (3.10)], it suffices to use $d_i^*(p) + \ell_i(A)$ bits;

(c) in the representation of each operand of any addition or subtraction involved in the evaluation of the i-th component of $\vec{e^*}(p)$ [according to (3.11)], it suffices to use

$$d_i(p) = h(p) - g(p) + \lceil \log c_i(C) \rceil = h - g + \lceil \log p \rceil + \lceil \log c_i(C) \rceil \text{ bits;}$$

(d) in the representation of any entry of row i of C when this entry is multiplied by a component of $\vec{r}(p)$ [according to (3.11)], it suffices to use

$$d_i(p) + h^*(p) - g^*(p) = d_i(p) + h^* - g^* + \lceil \log p \rceil \text{ bits.}$$

Thus, the b.b.s. process enables us to compute the solution vector $\vec{x}$, within error norm of the order of 2^{-p}, after p calls to the loop (3.10)-(3.12), even though only the order of $(\log i)$-bit precision is needed in the computations in the i-th call for this loop, for $i = 1, 2, \ldots$. In particular, even if $p = 60$ (which is much greater than what is usually needed in practice), then, still $\log p < 6$, whereas over 50 bits of single machine precision are usually allowed on modern computers performing matrix computations.

For comparison, if we perform the iteration based on (3.10) - (3.12) but do not use the b.b.s. process, then we generally must increase, at least to $H^*(p) - g(p)$ and $H(p) - g(p)$, the precision of the computations performed according to (3.10) and (3.11), respectively, where

$$H^*(p) = \log \max \left\{ \|\vec{f}\|, \max_{i,j} |a_{ij} x_j(p-1)| \right\},$$

$$A = [a_{ij}], \quad \vec{x}(p-1) = [x_j(p-1)],$$

$$H(p) = \log \max_{i,j} |c_{ij} r_j^*(p)|,$$

$$C = [c_{ij}], \quad \vec{r^*}(p) = [r_j^*(p)], \quad i, j = 1, \ldots, n;\ p = 1, 2, \ldots,$$

so that $H^*(p) - h^*(p)$ and $H(p) - h(p)$ are bounded from below by $H^* + bp - \log p$ and by $\log \max_{i,j} |c_{ij}| + H$, respectively, for two constants H^* and H.

3.6 An example of an extension of the b.b.s. process

To exemplify various possible extensions, assume that A denotes a real symmetric matrix filled with short binary numbers and having 1's on its diagonal, so that

$$A = L^* + I + U^*, \tag{3.18}$$

with $L^* = (U^*)^T$ being a proper lower triangular matrix.

Gauss-Seidel's iteration for $A\vec{x} = \vec{f}$ takes the following form (cf. [GL], [IK], or [A]):

$$\vec{x}(p+1) = \vec{f} - L^*\vec{x}(p+1) - U^*\vec{x}(p), \quad p = 0, 1, \ldots.$$

Rewrite this vector equation as follows:

$$\begin{aligned} \Delta\vec{x}(p+1) &= -L^*\Delta\vec{x}(p+1) - U^*\Delta\vec{x}(p), \quad p = 0, 1, \ldots, \\ \vec{x}(p) &= \sum_{i=0}^{p} \Delta\vec{x}(i), \quad p = 0, 1, \ldots., \end{aligned} \tag{3.19}$$

The iteration converges to the solution if and only if the matrix A satisfies (3.18) and is positive definite [[IK], pages 70-71] or [[GL], page 509]. In this case,

$$\|\Delta\vec{x}(p)\|_\infty < 3^{g-b^*p}, \quad p = 0, 1, \ldots, \tag{3.20}$$

where g is a fixed constant and 2^{-b^*} is the spectral radius of the matrix $B = (L^* + I)^{-1}U^*$, $2^{-b^*} \leq \| B \|_\infty$, $b^* > 0$. Estimating b^* generally takes a substantial amount of work, but for some important classes of the input matrices A, a good positive lower bound $\hat{b}$ on b^* is readily available. Then,

due to the rapid decrease of the error norm of (3.20), an application of the b.b.s. process enables us to decrease the precision of the computations.

The reader may easily check that the analysis and its results are similar for several other well-known iterative techniques, such as Jacobi's, SOR, SSOR.

3.7 Extension to the solution of piecewise linear PDE's

The results of the previous sections enable us to apply the b.b.s. technique in order to decrease the precision required in the computation of the solution of piecewise linear partial differential equations (PDE's) by means of multigrid methods (compare [PR], [PRa]). Let us show this, by outlining the multigrid approach and by observing its similarity to the iterations of section 3.5 and 3.6. For a given PDE, and for a fixed sequence of d-dimensional grids $G_0 \subset G_1 \subset G_1 \ldots \subset G_n$, define $n+1$ linear systems of difference equations,

$$D_i \vec{u}_i = \vec{b}_i, \tag{3.21}$$

by discretizing the PDE over the grids G_i, for $i = 0, 1, \cdots, n$. The N_i-dimensional vector $\vec{u}_i$ that satisfies (3.21) approximates the solution to the given PDE on the grid G_i, where $N_i = |G_i|$ denotes the number of vertices of the grid G_i, for $i = 0, 1, \ldots, n$. Let $u_i(\vec{x})$ for $\vec{x} \in G_i$ denote the respective component of the vector $\vec{u}_i$. Define the operators P_i of prolongation of $u_i(\vec{x})$ from G_{i-1} to G_i (such operators usually amount to interpolation by

averaging) and let

$$e_i(\vec{x}) = u_i(\vec{x}) - P_i u_i(\vec{x}), \quad \vec{x} \in G_i, \tag{3.22}$$

denote the prolongation errors of these operators for $i = 1, 2, \ldots, n$. Furthermore, define the vectors $P_i \vec{u}_{i-1}$ and $\vec{e}_i$ with the components $P_i u_{i-1}(\vec{x})$ and $e_i(\vec{x})$, respectively, for $\vec{x}$ ranging on G_i, and then define the residual vectors

$$D_i \vec{e}_i = \vec{r}_i \tag{3.23}$$

or, equivalently,

$$\vec{r}_i = \vec{b}_i - D_i P_i \vec{u}_{i-1}, \tag{3.24}$$

$i = 1, \ldots, n$.

We now recall the customary loop (V-cycle) of the multigrid algorithm for solving the system (3.21), for $i = n$, by means of successive evaluation of the following values, defined at stage i ($i = 1, \ldots, n$), for all $\vec{x} \in G_i$ [where, say, we fix $u_0(\vec{x}) = 0$ for $\vec{x} \in G_0$]:

a) $\hat{u}_{i-1}(\vec{x})$ (by prolongation of $u_{i-1}(\vec{x})$ from the grid G_{i-1}),

b) $r_i(\vec{x})$ [from (3.24)],

c) $e_i(\vec{x})$ [from (3.23)],

d) $u_i(\vec{x})$ [from (3.22)].

The vector equations (3.23) and (3.24) define the computational pattern of the iterative improvement of the solution to a linear system (3.21). Furthermore, in the case of a piecewise- linear PDE with constant coefficients, the

entries of the matrix D_i are "short" binary values, each represented with $O(1)$ bits. The only difference with the usual application of the iterative improvement scheme is the stage of the solution of the linear system (3.23). In the iterative improvement algorithms, this system is usually solved by direct methods, whereas the multigrid algorithms solve it by means of iterative methods (say, of Gauss-Seidel's or SSOR type in the symmetric case).

Furthermore, the number of iterations required in order to solve the linear system (3.23) arising in a multigrid algorithm is typically bounded from above by a fixed constant, which corresponds to setting $p = O(1)$ in section 3.5. Thus, by applying techniques exemplified in section 3.6 to solving the system (3.23) and the techniques of section 3.5 to computing $\vec{r}_i$ from the vector equation (3.24), we decrease the precision of these computation to $O(1)$ bits, without affecting the accuracy of the output.

3.8 Numerical Experiments

In this section we present the results of some numerical experiments designed in order to compare the performance of two implementations of the generalized algorithm of section 3.5 for the iterative improvement of the solution of a linear system of equations. That is, we tested a customary implementation and one using the b.m.r./b.b.s. techniques.

We have run our experiments on a general purpose computer that relies on fixed precision representation of floating point numbers and uses floating point hardware logic for acceleration of numerical computations. Since the b.m.r./b.b.s. techniques rely on using variable precision representation of numbers, we could not directly compare the CPU time, executable size or

run-time memory consumption of the b.m.r./b.b.s. algorithm with that of the customary algorithm. Instead, we emulated both algorithms with a high level language using special data structures and then approximately measured the bit-complexity as follows.

Definition 3.3 *For floating point numbers r_1 and r_2 represented with the precision of p_1 and p_2 bits, respectively, we define the bit-complexity of their addition c^+ and multiplication c^*, as follows.*

$$c^+(r_1, r_2) = \max\{p_1, p_2\}, \quad c^*(r_1, r_2) = p_1 \times p_2.$$

In our experiments, the input to the algorithm consists of an $n \times n$ matrix A, a matrix C that approximates A^{-1}, an n-dimensional vector $\vec{f}$, and an error bound $\epsilon > 0$. The program calculates and outputs $\vec{x}(p)$ such that $||\vec{x}(p) - \vec{x}|| < \epsilon$, where $\vec{x}$ is the solution of the linear system $A\vec{x} = \vec{f}$.

We have implemented the algorithm in ANSI C language and the program was compiled, linked and run on a SUN Sparc station running SUN OS version 4.

We store floating point numbers in a C structure consisting of a sign, a mantissa, an exponent, and a precision value. All the input, output and intermediate results have been stored in this format, and all the arithmetic operations needed for the experiments (such as addition, multiplication and modular reduction) have been implemented with C functions. The approximate inverse matrices have been calculated by using PLU decomposition with partial pivoting.

In the remainder of this section, we show only the input and output of our experiments, with complexity estimates based on the above definition.

These results confirm the theory by showing a consistent decrease of the bit-complexity in the transition from the customary implementation to the b.m.r./b.b.s. implementation. In particular, we report a decrease of 6–17%. This is shown in the respective tables by the ratio of the additive and multiplicative complexities, respectively, between the two algorithms. We use the decimal representation for the sake of clarity.

Experiment 3.1 This example input is from [DB], pp. 184–185, and has input:

$$\begin{aligned}
\epsilon &= +(0.10)_{10} \times 10^{-4}, \\
A &= \begin{pmatrix} +(0.20000)_{10} \times 10^{0} & +(0.16667)_{10} \times 10^{0} & +(0.14286)_{10} \times 10^{0} \\ +(0.16667)_{10} \times 10^{0} & +(0.14286)_{10} \times 10^{0} & +(0.12500)_{10} \times 10^{0} \\ +(0.14286)_{10} \times 10^{0} & +(0.12500)_{10} \times 10^{0} & +(0.11111)_{10} \times 10^{0} \end{pmatrix}, \\
C &= \begin{pmatrix} +(0.21618)_{10} \times 10^{4} & -(0.57597)_{10} \times 10^{4} & +(0.37001)_{10} \times 10^{4} \\ -(0.57597)_{10} \times 10^{4} & +(0.15793)_{10} \times 10^{5} & -(0.10362)_{10} \times 10^{5} \\ +(0.37001)_{10} \times 10^{4} & -(0.10362)_{10} \times 10^{5} & +(0.69087)_{10} \times 10^{4} \end{pmatrix}, \\
\vec{f} &= (+(0.50953)_{10} \times 10^{0} \quad +(0.43453)_{10} \times 10^{0} \quad +(0.37897)_{10} \times 10^{0})^{T}.
\end{aligned}$$

The initial and final approximations to $\vec{x}$ are

$$\begin{aligned}
\vec{x}(0) &= (+(0.90000)_{10} \times 10^{0} \quad +(0.90000)_{10} \times 10^{0} \quad +(0.90000)_{10} \times 10^{0})^{T}, \\
\vec{x}(4) &= (+(0.99999)_{10} \times 10^{0} \quad +(0.99999)_{10} \times 10^{0} \quad +(0.99999)_{10} \times 10^{0})^{T}.
\end{aligned}$$

Both the customary as well as the b.m.r./b.b.s. algorithms produce the same output after 4 iterations. However, the second method has lower additive and multiplicative complexity at every iteration, as shown in table 3.1.

Experiment 3.2 This example is from [J], p. 52, and has input:

$$\begin{aligned}
\epsilon &= +(0.10)_{10} \times 10^{-3}, \\
A &= \begin{pmatrix} +(0.1230)_{10} \times 10^{1} & +(0.4560)_{10} \times 10^{1} & +(0.9870)_{10} \times 10^{1} \\ -(0.9610)_{10} \times 10^{1} & +(0.6020)_{10} \times 10^{1} & +(0.1110)_{10} \times 10^{2} \\ +(0.7310)_{10} \times 10^{1} & +(0.2890)_{10} \times 10^{1} & +(0.5040)_{10} \times 10^{1} \end{pmatrix}, \\
C &= \begin{pmatrix} +(0.1452)_{10} \times 10^{-1} & -(0.4629)_{10} \times 10^{-1} & +(0.7351)_{10} \times 10^{-1} \\ -(0.1082)_{10} \times 10^{1} & +(0.5508)_{10} \times 10^{0} & +(0.9062)_{10} \times 10^{0} \\ +(0.5995)_{10} \times 10^{0} & -(0.2487)_{10} \times 10^{0} & -(0.4278)_{10} \times 10^{0} \end{pmatrix}, \\
\vec{f} &= (+(0.4120)_{10} \times 10^{1} \quad +(0.5340)_{10} \times 10^{1} \quad -(0.3560)_{10} \times 10^{1})^{T}.
\end{aligned}$$

iteration	customary		b.m.r/b.s.s.	
i	c_i^+	c_i^*	c_i^+	c_i^*
1	285	1575	261	1350
2	510	2925	453	2475
3	735	4275	629	3600
4	960	5625	802	4725
total	2490	14400	2145	12150
ratio	1	1	0.86	0.84

Table 3.1: Complexity of Experiment 3.1

iteration	customary		b.m.r/b.s.s.	
i	c_i^+	c_i^*	c_i^+	c_i^*
1	228	1008	228	864
2	408	1872	379	1584
3	588	2736	530	2304
4	768	3600	681	3024
total	1992	9216	1818	7776
ratio	1	1	0.91	0.84

Table 3.2: Complexity of Experiment 3.2

The initial and final approximations to $\vec{x}$ are

$$\begin{aligned}\vec{x}(0) &= (-(0.4491)_{10} \times 10^0 \quad -(0.4743)_{10} \times 10^1 \quad +(0.2665)_{10} \times 10^1)^T, \\ \vec{x}(4) &= (-(0.4490)_{10} \times 10^0 \quad -(0.4743)_{10} \times 10^1 \quad +(0.2665)_{10} \times 10^1)^T,\end{aligned}$$

where the same output is obtained by both customary and b.m.r./b.b.s. algorithms. Again, the second algorithm is faster at every iteration, as seen in table 3.2.

Experiment 3.3 This example is from [GL], p. 147, and has the input:

$$\begin{aligned}\epsilon &= +(0.10)_{10} \times 10^{-2}, \\ A &= \begin{pmatrix} +(0.986)_{10} \times 10^3 & +(0.579)_{10} \times 10^3 \\ +(0.409)_{10} \times 10^3 & +(0.237)_{10} \times 10^3 \end{pmatrix}, \\ C &= \begin{pmatrix} -(0.757)_{10} \times 10^{-1} & +(0.185)_{10} \times 10^0 \\ +(0.131)_{10} \times 10^0 & -(0.315)_{10} \times 10^0 \end{pmatrix},\end{aligned}$$

iteration i	customary c_i^+	customary c_i^*	b.m.r/b.s. c_i^+	b.m.r/b.s. c_i^*
1	90	252	90	216
2	162	468	152	396
3	234	684	202	556
total	486	1404	444	1168
ratio	1	1	0.91	0.83

Table 3.3: Complexity of Experiment 3.3

$$\vec{f} = (+(0.235)_{10} \times 10^3 \quad +(0.107)_{10} \times 10^3)^T.$$

The initial and final approximations to solution $\vec{x}$ are:

$$\vec{x}(0) = (+(0.200)_{10} \times 10^1 \quad -(0.290)_{10} \times 10^1)^T,$$
$$\vec{x}(3) = (+(0.200)_{10} \times 10^1 \quad -(0.300)_{10} \times 10^1)^T,$$

where the same output is obtained by both algorithms. The savings due to the b.m.r./b.s. algorithm are reported in table 3.3.

Experiment 3.4 This is a random input.

$$\epsilon = +(0.10)_{10} \times 10^{-2},$$
$$A = \begin{pmatrix} +(0.6000)_{10} \times 10^1 & +(0.1100)_{10} \times 10^2 & +(0.2000)_{10} \times 10^2 \\ +(0.3500)_{10} \times 10^2 & +(0.6800)_{10} \times 10^2 & +(0.1330)_{10} \times 10^3 \\ +(0.2600)_{10} \times 10^3 & +(0.5170)_{10} \times 10^3 & +(0.1030)_{10} \times 10^4 \end{pmatrix},$$
$$C = \begin{pmatrix} -(0.6525)_{10} \times 10^1 & +(0.5051)_{10} \times 10^1 & -(0.5255)_{10} \times 10^0 \\ +(0.7500)_{10} \times 10^1 & -(0.5000)_{10} \times 10^1 & +(0.5000)_{10} \times 10^0 \\ -(0.2117)_{10} \times 10^1 & +(0.1235)_{10} \times 10^1 & -(0.1173)_{10} \times 10^0 \end{pmatrix},$$
$$\vec{f} = (-(0.1440)_{10} \times 10^1 \quad +(0.7523)_{10} \times 10^1 \quad -(0.3392)_{10} \times 10^0)^T.$$

The initial and final approximations to solution $\vec{x}$ are as follows, obtained by the customary and b.m.r./b.s. algorithms respectively:

$$\vec{x}(0) = (+(0.4758)_{10} \times 10^2 \quad -(0.4859)_{10} \times 10^2 \quad +(0.12379)_{10} \times 10^2)^T,$$
$$\vec{x}(4) = (+(0.4757)_{10} \times 10^2 \quad -(0.4858)_{10} \times 10^2 \quad +(0.12377)_{10} \times 10^2)^T,$$
$$\vec{x}(4) = (+(0.4757)_{10} \times 10^2 \quad -(0.4858)_{10} \times 10^2 \quad +(0.12381)_{10} \times 10^2)^T.$$

Note that the two outputs are different, though within the same error bound. The complexities do differ, as seen in table 3.4.

iteration	customary		b.m.r/b.b.s.	
i	c_i^+	c_i^*	c_i^+	c_i^*
1	230	1056	230	912
2	411	1968	392	1680
3	592	2880	555	2448
4	773	3792	709	3171
total	2006	9696	1886	8211
ratio	1	1	0.94	0.85

Table 3.4: Complexity of Experiment 3.4

Chapter 4

Certified Numerical Computation of the Sign of a Matrix Determinant

4.1 Overview

4.1.1 The problem and background.

The classical problem of computing det A, the determinant of an $n \times n$ matrix A, has long history (see e.g. [Mu], [Ma], [Du], [Fo], [R52], [E67], [B68], [P88], [BP]). Recently, it turned out that some of the most fundamental problems of computational geometry (such as the computation of convex hulls and Voronoi diagrams) are reduced to the computation of $\det A$ or, more precisely, its sign, that is, testing whether $\det A = 0$, $\det A > 0$, or $\det A < 0$. [BEPP97], [BEPP98], [BKMNSU], [EC], [Em], [FVW], [Y], [YD].

In many areas of computational geometry, lower dimensional problems must be solved, and then n ranges between 2 and 10, usually staying below 5. In this class of applications, the matrix A is filled with "long" numbers, representing the real data with a high precision (and thus allowing to treat

the important case of a nearly singular input). In another major class of applications [DLa], [DL97], [ES], [AF], [FR], [MA], n is large (say, in the range from 100 to 500), whereas the matrix A is filled with relatively short integers (say, represented with 5 to 10 bits). Such applications include the computation of the orientation of a polyhedron or an algebraic variety in a high-dimensional space (for instance, such computations are required in the area of convex optimization in statistical physics and chemistry).

In both cases we may apply the well known methods to compute $\det A$ based on the triangular ($PLUP_1$) or orthogonal (QR) factorization of the matrix A. High speed of these computations is ensured as they are performed numerically, with a fixed (single or double) precision, which currently has much faster computer implementation than rational, integer, and multiple precision arithmetic. The major problem, however, is to certify that the output is correct in the presence of rounding errors.

Substantial advance in this area was the paper [C], though the correctness certification of the output of the proposed algorithm (based on the modified Gram-Schmidt method) complicated and slowed down the computation. The algorithm of [ABDPY] competes with one of [C] for $n \leq 4$ but does not work well for larger n.

Recent progress reported in [BEPP97], [BEPP98] relies on using symbolic algorithm that computes $\det A$ modulo several primes $p_1, \ldots, p_k$ such that their product exceeds $|\det A|$. In this chapter propose effective algorithms for the recovery of the sign of $\det A$ from these data, based on some novel application and extensions of the Chinese remainder algorithm, which [BEPP97] and [BEPP98] reduce to single or double precision computation.

The algorithms of [BEPP97] and [BEPP98] seem to be among the currently best ones for the problem. Their bottleneck is the relatively expensive computation of $(\det A) \bmod m_i$, for $i = 1, \ldots, k$. The number k of the pairwise relatively prime moduli m_i involved and, consequently, the computational cost decrease if $|\det A|$ is shown to be smaller.

The algorithms of [PYS] complement ones of [BEPP97], [BEPP98] by computing $\det A$ numerically. The correctness of the output is certified unless the algorithm establishes a relatively small upper bound on $|\det A|$. This would be an ideal example of effective combination of symbolic and numerical techniques, but numerical experiments show that the techniques of [PYS] give too rough bounds, greatly exceeding the actual value of $|\det A|$. because the range for the values of $|\det A|$ is huge (from 0 to D^+ with D^+ on the level of $||A||^n$), and the known techniques of error analysis only guarantee rounding error bounds of order $D^+n^2\epsilon$, ϵ being the *machine epsilon* (also called *unit roundoff*). Such bounds can be large even where $|\det A|$ is actually small (cf. section 4.8.3). Therefore, the roundoff error bounds for computing $\det A$ may exceed the value $|\det A|$ substantially.

4.1.2 Our results.

Our most recent achievement was to improve the analysis and the estimates of [PYS] by using new techniques. Our algorithm 4.1 computes $\det A$ numerically and either certifies that its sign has been computed correctly or shows which increase of the precision of computing should yield the certified output. If a very large increase of the precision is required, then the transition to the symbolic approach is motivated, and in section 4.6 we compute

or estimate from above the value of $|\det A|$. The computations involve order of n^3 arithmetic operations (cf. sections 4.3, 4.5 and 4.6). Numerical experiments reported in section 4.7 confirm the efficiency of our approach. We briefly examine other numerical techniques as well as modifications of our approach in section 4.8.

Our major technical novelty versus [PYS] is the certification of the sign of $\det A$ without estimating the roundoff error of computing $\det A$. Instead, we estimate the minimum distance N from the matrix $\tilde{L}\tilde{U}$ to a singular matrix, $\tilde{L}$ and $\tilde{U}$ being the computed approximations to the factors L and U in the triangular ($PLUP_1$) factorization of A. The idea is that $\det(A+E)$ does not change its sign when E ranges in the ball of radius N centered in the origin. On the other hand, since $N = 1/||\tilde{U}^{-1}\tilde{L}^{-1}||$ and since the matrices $\tilde{L}$ and $\tilde{U}$ are the two available triangular matrices, it is not hard to obtain a certified and quite tight upper bound on N at the cost of performing $O(n^3)$ arithmetic operations and comparisons. The combination of numerical and algebraic (residue) computation in section 4.8.5 also has some technical novelty.

4.1.3 The order of presentation

We present our results in the following order. In the next section, we recall some known estimates for the errors of Gaussian elimination due to roundoff. In section 4.3, we relate the certification of the sign of $\det A$ in the presence of roundoff errors to the minimum distance from A to a singular matrix. Then we propose an algorithm for computing and certifying the sign of $\det A$ based on this relation. In section 4.4, we show how to use the computed information in the case where the algorithm does not produce a certified

correct output. In section 4.5, we elaborate the stage of estimating the minimum distance to a singular matrix, which is a major block of our algorithm of section 4.3. In section 4.6, we complement the algorithm by presenting some techniques for computing or estimating from above $|\det A|$. In section 4.7, we present the results of our numerical experiments. In section 4.8, we comment on some variations of our approach and some alternatives. In particular, we indicate some reasons for the deficiency of an approach of [PYS] and of one based on the Barrlund and Sun theorem; we also point out the modifications of our approach that use Gauss-Jordan, $PLDM^TP_1$, QR (rather than $PLUP_1$) factorizations of A or the LDL^T factorization of A^TA. In the section 4.8.6, we recall the techniques for computing $\tilde{U}^{-1}\tilde{L}^{-1}$.

4.1.4 Definitions

Hereafter, $\vec{A}_k$ denotes the k-th column vector of A, $w_{i,j}$ denote the (i,j)-th entry of a matrix $W = (w_{i,j})$. $\|.\| = \|.\|_h$ denotes a fixed operator matrix norm, in particular we will use the 2-norm $\|.\|_2$, the row norm $\|W\|_\infty = \max_i \sum_j |w_{i,j}|$, and the column norm $\|W\|_1 = \max_j \sum_i |w_{i,j}|$ for a matrix $W = (w_{i,j})$ (cf. [GL96] or [H96]). The transpose of W is denoted $W^T = (w_{j,i})$, whereas $|W|$ denotes the matrix $(|w_{i,j}|)$. We write $|W| \leq |V|$ iff $V = (v_{i,j})$ and $|w_{i,j}| \leq |v_{i,j}|$ for all i and j. I denotes the $n \times n$ identity matrix. $\mathrm{diag}(w_{i,i})$ denotes the diagonal matrix with the diagonal entries $w_{i,i}$, $i = 1, \ldots, n$. $\det W$ and $\mathrm{sign}(\det W)$ denote the determinant of a square matrix W and its sign, respectively. A triangular matrix will be called *unit* (respectively, *proper*) triangular if its diagonal is filled with ones (respectively, zeros).

4.2 Roundoff errors of factorization by Gaussian elimination.

$\det A$ and its sign for a given matrix A can be immediately obtained from the triangular ($PLUP_1$) factorization of A, but the problem is to analyze the effect of the roundoff errors when the factorization is computed numerically. In this analysis we will apply some known estimates, which we will recall in this section.

Theorem 4.1 (cf. [H96], Theorem 9.3, page 175). *Let*

$$A = PA'P_1,\ \tilde{A} = A' + E = \tilde{L}\tilde{U}, \tag{4.1}$$

where A, A', $\tilde{A}$, E, P, P_1, $\tilde{L}$, and $\tilde{U}$ are $n \times n$ matrices, P and P_1 are permutation matrices, $\tilde{L} = (\tilde{l}_{i,j})$ is a unit lower triangular matrix (so that $\det \tilde{L} = 1$), and $\tilde{U} = (\tilde{u}_{i,j})$ is an upper triangular matrix, $\tilde{L}$ and $\tilde{U}$ are computed numerically, by means of Gaussian elimination (with complete pivoting) applied to the matrix A with unit roundoff ϵ (machine epsilon). Then

$$|E| \le \gamma_n |\tilde{L}| \cdot |\tilde{U}|,\ \gamma_n = n\epsilon/(1 - n\epsilon).$$

Two special cases of this result cover Gaussian elimination with partial pivoting (for $P_1 = I$) and with no pivoting (for $P_1 = P = I$).

By (4.1) we have

$$\det A = (\det P)(\det A') \det P_1\ , \tag{4.2}$$

$$\det \tilde{A} = \det \tilde{U} = \tilde{u}_{1,1}\tilde{u}_{2,2} \cdots \tilde{u}_{n,n}\ . \tag{4.3}$$

Since $\det P$ and $\det P_1$ are readily available (they equal 1 or -1), the equations (4.2) and (4.3) define $\mathrm{sign}(\det A)$ provided that the diagonal entries of $\tilde{U}$ are available and that ϵ is sufficiently small to guarantee that

$$\mathrm{sign}(\det A') = \mathrm{sign}(\det \tilde{A}) \ . \tag{4.4}$$

In the next section we will show how to verify (4.4) by using the following corollary of theorem 4.1.

Corollary 4.1 *Under the assumptions of theorem 4.1, we have*

$$\|E\| \leq e = \|(|\tilde{L}| \cdot |\tilde{U}|)\|\gamma_n \tag{4.5}$$

for any fixed operator matrix norm.

The computation of the row and column norms of $|\tilde{L}| \cdot |\tilde{U}|$ involves only $O(n^2)$ arithmetic operations, since

$$\|(|\tilde{L}| \cdot |\tilde{U}|)\|_\infty = \|(|\tilde{L}|\vec{u})\|_\infty, \|(|\tilde{L}| \cdot |\tilde{U}|)\|_1 = \|(\vec{l}^{\,T}|\tilde{U}|)\|_1$$

, where $\vec{l}$ and $\vec{u}$ are two vectors with the components $l_j = \sum_i |\tilde{l}|_{i,j}$, $u_i = \sum_j |\tilde{u}|_{i,j}$, where $|\tilde{l}|_{i,j}$ and $|\tilde{u}|_{i,j}$ denote the (i,j)-th entries of the matrices $|\tilde{L}|$ and $|\tilde{U}|$, respectively.

By corollary 4.1, the rounding error of the computation of the $PLUP_1$ factorization of A is bounded in terms of γ_n and the norm of the matrix $|L| \cdot |U|$. It is known that in the case of using complete pivoting, the (k,j)-th entries $\tilde{a}_{k,j}, \tilde{l}_{k,j}$ and $\tilde{u}_{k,j}$ of the matrices $\tilde{A}, \tilde{L}$ and $\tilde{U}$ of (4.1), respectively, satisfy the bounds

$$|\tilde{l}_{k,j}| \leq 1, \ |\tilde{u}_{k,j}| \leq \rho_k \max_{j,h} |\tilde{a}_{g,h}|,$$

for all k and j, where $\rho_k \leq k^{1/2}(2 \cdot 3^{1/2} \cdots k^{1/(k-1)})^{1/2} \leq 1.8k^{(\ln k)/4}$ (cf. [GL96], p.119). The same bounds are known in the case of partial partial pivoting, but theoretically only for $\rho_k \leq 2^{k-1}$. In practice, however, $\rho_k = O(k)$ even in the case of using partial pivoting (cf. [GL96], p.116). Some improvement of the worst case error bound (even versus the case of complete pivoting) can be obtained by means of symmetrization (see section 4.8.5).

4.3 Certification of the sign in terms of the smallest distance to a singular matrix.

The following sufficient condition for (4.4),

$$|\det \tilde{U}| = |\det \tilde{A}| > |(\det A) - \det(P\tilde{A}P_1)| = e_d,$$

can be verified based on the straightforward crude estimate :

$$e_d \leq n^2 e_+ D_+,$$

where e_+ denotes the maximum absolute value of the entries of E and $D_+ = \prod_{k=1}^{n}(||\vec{A}_k||_2 + ne_+)$. This estimate is based on Hadamard's bound, $|\det A| \leq \prod_{k=1}^{n} ||\vec{A}_k||_2$ (cf. [H96], p.287). It is not easy to compute e_+, but we may replace e_+ by its upper bound $e^* = \gamma_n \max_{i,j}(|\tilde{L}| \cdot |\tilde{U}|)_{i,j}$ implied by theorem 4.1. Here $(|\tilde{L}| \cdot |\tilde{U}|)_{i,j}$ denotes the (i,j)-th entry of the matrix $|\tilde{L}| \cdot |\tilde{U}|$. Then we obtain the following crude estimate:

$$e_d \leq e_d^+ = D_+ n^2 e^*, \; D_+ = \prod_{k=1}^{n}(||\vec{A}_k||_2 + ne^*). \tag{4.6}$$

Our more refined techniques for the verification of equation (4.4) will rely on the next two results.

Proposition 4.1 *For two given matrices W and $W+\Delta$, for a fixed matrix norm $\|.\|$ and for all singular matrices S, let*

$$\max\{\|W-S\|, \|W+\Delta-S\|\} > \|\Delta\|.$$

Then

$$\operatorname{sign}(\det W) = \operatorname{sign}(\det(W+\Delta)) \ . \tag{4.7}$$

Proof. Unless the equation (4.7) holds, there must exist a real t, $0 \leq t \leq 1$, such that $S = W + t\Delta$ is a singular matrix. Clearly,

$$\|W-S\| = t\|\Delta\| \leq \|\Delta\|,$$

$$\|W+\Delta-S\| = (1-t)\|\Delta\| \leq \|\Delta\|,$$

and (4.7) follows.

The next theorem is due to Gastinel, 1966 (see [K66]), but its particular case where $\|.\| = \|.\|_2$ was obtained by Eckart and Young in 1939 [EY].

Theorem 4.2 *For any fixed nonsingular matrix W and any fixed operator matrix norm $\|.\|$, we have $\frac{1}{\min\|W-S\|} = \|W^{-1}\|$, where the minimum is over all singular matrices S.*

Combining proposition 4.1 and theorem 4.2 implies the next result.

Corollary 4.2 *Under the assumptions of theorem 4.2, if*

$$1/\min\{\|W^{-1}\|, \ \|(W+\Delta)^{-1}\|\} > \|\Delta\|$$

, then (4.7) holds.

Apply corollary 4.2 to $W = A'$ and $\Delta = \tilde{A} - A'$ to obtain the next result.

Corollary 4.3 *The equation (4.4) holds if*

$$\|E\| < 1/\min\{\|(A')^{-1}\|,\ \|\tilde{A}^{-1}\|\}$$

.

Now we are ready to propose an algorithm for computing sign(det A), where at the second stage we rely on the bound (4.6), which suffices for a large class of inputs, and at the next two stages we rely on corollary 4.3.

Algorithm 4.1

Input: *an $n \times n$ matrix A, a fixed matrix norm ($||\cdot||_\infty$ or $||\cdot||_2$), and a unit roundoff ϵ.*

Output: *either the certified value of sign* (det A) *or (see also section 4.4) FAILURE.*

Computations.

1. *Apply Gaussian elimination (with complete, partial, or no pivoting) using the unit roundoff ϵ, to compute the matrices $\tilde{L}$ and $\tilde{U}$ of (4.1).*

2. *Compute an upper bound on e_d (in particular, we may use the simple crude bound e_d^+ of (4.6)) and check if $|\det \tilde{U}|$ exceeds this bound. If so, compute and output sign* (det A) *based on (4.2)-(4.4).*

3. *Otherwise, estimate the norm $N = \|\tilde{A}^{-1}\| = \|\tilde{U}^{-1}\tilde{L}^{-1}\|$ from above and/or below (see section 4.5) to decide whether*

$$eN < 1. \tag{4.8}$$

If the latter inequality is verified, compute and output sign (det A) *based on (4.2)-(4.4). Otherwise output FAILURE.*

The complexity of the computations by the algorithm is dominated at its stages 1 and 3. We refer the reader to [GL96] and [BP] on the complexity of stage 1 and to section 4.5 on the complexity of stage 3. In both cases we need $O(n^3)$ arithmetic operations. At stage 1, we may also need $O(n^3)$ or $O(n^2)$ comparisons for complete or partial pivoting, respectively.

4.4 Recipes in the case of FAILURE.

Suppose that algorithm 4.1 has output FAILURE and that an upper bound N^+ on N and the value $f = eN^+$ are available. Then we have several options:

1. Repeat the computation but with the unit roundoff $\epsilon_{new} = c\epsilon_{old}/f$ for some heuristic choice of $c < 1$. The value c can be adapted dynamically, depending on the resulting change of the value eN^+. If the latter value changes proportionally to ϵ_{new} (as can be expected unless A is a very ill-conditioned matrix), then (4.8) holds for $\epsilon = \epsilon_{new} = c\epsilon_{old}/f$ and for any $c < 1$. Recomputation of $\tilde{L}$ and $\tilde{U}$ for ϵ_{new} can be simplified. since several leading digits in the representation of the computed values stay invariant, and only the remaining trailing digits must be recomputed (compare [EPY]).

2. Improve the upper estimate N^+ for $N = \|\tilde{U}^{-1}\tilde{L}^{-1}\|$ at stage 3 (see the next section).

3. If the value f is too large so that numerical computations with a unit roundoff $\epsilon_{new} < \epsilon_{old}/f$ become too expensive, shift to the symbolic algorithms of [BEPP97] and [BEPP98]. In this case, one needs an

a priori upper bound on $|\det A|$. Hadamard's inequality, $|\det A| \leq \prod_{k=1}^{n} ||\vec{A}_k||_2$, or the bound $|\det A| \leq e_d^+ + |\det \tilde{U}|$ can be used, but it may pay to refine these bounds by performing some additional computations (see sections 4.6 and 8.5).

4.5 Estimating the minimum distance to a singular matrix.

4.5.1 Estimating the distance from above.

To estimate from above the minimum distance from the matrix $\tilde{A}$ to a singular matrix or, equivalently, to estimate $N = \|\tilde{U}^{-1}\tilde{L}^{-1}\|$ from below, singular matrix, we may apply the simple iterative algorithm of [CMSW79], which, according to [GL96], "produces a good order-of-magnitude " lower bound N^- on N at the cost of performing $O(jn^2)$ arithmetic operations in j iterations (practically, j is much smaller than n). If (4.3) does not hold even for N replaced by N^-, then we may apply the recipes of section 4.4, for some heuristic choice of N^+ and $f = eN^+$.

4.5.2 Two-sided estimates with randomization.

Recall that

$$N = \|\tilde{A}^{-1}\|_2 = \|\tilde{U}^{-1}\tilde{L}^{-1}\|_2 = \sigma_1(\tilde{U}^{-1}\tilde{L}^{-1}) = 1/\sigma_n(\tilde{L}\tilde{U}) = 1/\sigma_n(\tilde{A})$$

, $\sigma_k(W)$ denoting the k-th largest singular value of a matrix W. Dixon in [D83] shows that with a probability at least $P_{Dixon}(k, \Theta) = 1 - 0.8\Theta^{-k/2}n^{1/2}$ the lower bound $N_k^- = (\vec{x}^T(\tilde{A}\tilde{A}^T)\vec{x})^{1/2k} \leq ||\tilde{A}^{-1}||_2 = 1/\sigma_n(\tilde{A})$ is also an upper bound on N within the factor $\Theta > 1$, that is, $\Theta N_k^- \geq ||\tilde{A}^{-1}||_2$, for

$k \geq 1$ and a random choice of a vector $\vec{x}$ on the unit sphere $S_n = \{\vec{x} : \vec{x}^T\vec{x} = 1\}$, under the uniform probability distribution on S_n. Note that in our case the computation of N_k^- costs $O(kn^2)$ since $\tilde{A} = \tilde{L}\tilde{U}$ and since the matrices $\tilde{L}$ and $\tilde{U}$ are triangular and are already available. The Dixon estimate relies on the application of the power method to approximating the smallest eigenvalue of $\tilde{A}\tilde{A}^T$, which is the smallest singular value of $\tilde{A}$.

An alternative approach produces two-sided estimates for N by means of Lanczos algorithm, which converges faster than the power method ([GL96], ch.9). Lanczos algorithm computes σ_n^*, an upper estimate for $\sigma_n(\tilde{L}\tilde{U})$, $\sigma_n^* \geq \sigma_n(\tilde{L}\tilde{U})$. For a fixed positive $\Theta > 1$ and for nonsingular $\tilde{L}$ and $\tilde{U}$,the estimates for the probability $P_{Lanczos}(l, \Theta)$ of having $\sigma_n^*/\sigma_n(\tilde{L}\tilde{U}) \leq 1/\Theta$ can be expressed as a function in the number l of Lanczos iterations for a random choice of the initial vector for the Lanczos process (cf. [KW92], [KW94]). The cost of performing each Lanczos iteration is $O(n^2)$, since $\tilde{L}$ and $\tilde{U}$ are two available triangular matrices.

4.5.3 Deterministic lower estimates for the distance.

Section 8.3 of [H96], pages 159-161, shows some techniques for rapid computation of some crude upper bounds on $||T^{-1}||$ for triangular matrices T, and this can be immediately translated into rapid computation of some crude upper bounds on $N = \|\tilde{U}^{-1}\tilde{L}^{-1}\| \leq \|\tilde{U}^{-1}\| \cdot \|\tilde{L}^{-1}\|$. To yield more refined upper bounds on N, one may actually compute the inverses $\tilde{U}^{-1}$ and $\tilde{L}^{-1}$, then their product X and finally (an upper bound on) the norm $N = \|\tilde{U}^{-1}\tilde{L}^{-1}\|$. Detailed presentation of such computations can be found in [H96], sections 13.2-13.3, pages 265-275. For reader's convenience, we

sketch some of these algorithms in the appendix.

Assuming the computation with unit roundoff ϵ, [H96] presents the estimates for the residual norms $||\tilde{A}X - I|| = r(X)$, $||\tilde{A}X - I|| = r^*(X)$, and the error norms $||\tilde{A}^{-1} - X|| = e(X)$ for the computed approximations X to $\tilde{U}^{-1}\tilde{L}^{-1} = \tilde{A}^{-1}$. The estimates are given in the form

$$r(X) \leq c_n\epsilon||\tilde{L}|| \cdot ||\tilde{U}|| \cdot ||X||,$$

$$r^*(X) \leq c_n^*\epsilon||\tilde{L}|| \cdot ||\tilde{U}|| \cdot ||X||,$$

$$e(X) \leq c_n'\epsilon||\tilde{L}|| \cdot ||\tilde{U}|| \cdot ||X|| \cdot ||\tilde{A}^{-1}||.$$

Here c_n, c_n^* and c_n' are constants independent of ϵ and $\tilde{A}$, which can be elaborated by using the error analysis techniques of [H96].

Instead of applying these estimates, we may directly compute $\tilde{A}X - I$ or $X\tilde{A} - I$ (in $O(n^3)$ operations) and arrive at the residual norms $r(X)$ or $r^*(X)$ within the roundoff error bound $\gamma_n||\tilde{A}|| \cdot ||X||$, $\gamma_n = \epsilon n/(1 - \epsilon n)$ (cf. [H96], page 78). For computing the row and column norms of the matrix, we may apply recursive pairwise summation, whose relative roundoff error is at most $\gamma_{\lceil \log_2 n \rceil}$ (cf. [H96], page 92). If $r = \min\{r(X), r^*(X)\} < 1$, we immediately estimate that $e(X) \leq r||X||/(1 - r)$.

Remark 5.1. One may try to improve the approximation to $\tilde{A}^{-1}$ by $X = X_0$ by applying Newton's iteration, $X_{i+1} = X_i(2I - AX_i)$, whose i-th iterative step for every i squares the residual matrices $\tilde{A}X_i - I$ and $X_i\tilde{A} - I$ and consequently squares the upper bounds on their norms (cf. [PS]).

4.6 Estimating the magnitude of the determinant.

One may refine the upper bounds of section 4.4 on $|\det A|$ by relying on the following well known fact (cf. section 4.8.5 for an alternative way to the refinement):

Fact 4.1

$$|\det A| = \prod_{i=1}^{k} \sigma_i \ ,$$

where $\sigma_1, \ldots, \sigma_r$ *are the singular values of* A, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, $\sigma_{r+i} = 0$ *for* $i = 1, \ldots, n-r$, $r = \mathrm{rank} A$.

Consequently, $d^+ = \prod_{i=1}^{n} \sigma_i^+ \geq |\det A|$ if $\sigma_i^+ \geq \sigma_i$, $i = 1, \ldots, n$, and our problem is reduced to approximating $\sigma_1, \ldots, \sigma_n$ from above. The known SVD algorithms ([GL96], section 8.6 and p.254) enable us to solve the latter task in $7n^3/3 + O(n^2)$ operations. A faster though more crude solution may rely on approximating from above a few smallest values $\sigma_n, \sigma_{n-1}, \ldots$, by means of Lanczos algorithm and on applying the readily available upper bound $\sigma = \min\{\|\tilde{A}\|_1, \|\tilde{A}\|_\infty\}\|$ on all other σ_i.

4.7 Numerical Experiments.

In this section, we compare the results of the numerical experiments performed in [PYS] with those based on algorithm 4.1. The comparison clearly favors the latter algorithm.

In the experiments, we compute numerically the determinants of $n \times n$ matrices A for $2 \leq n \leq 10$, based on computing the LU, PLU, and $PLUP_1$ factorizations of A. The input matrices A have been composed by using

the following steps to produce random non-singular matrices either with determinants ± 1 or with relatively small known determinants:

1. For an auxiliary pair of lower and upper triangular matrices $L^{(0)}$ and $U^{(0)}$, respectively, let their non-zero off-diagonal entries be random integers in the interval $(-10, 10)$.

2. Either let the diagonal entries $l_{i,i}^{(0)}$ and $u_{i,i}^{(0)}$ be also chosen at random in the same way (see table 4.1) or set $l_{i,i}^{(0)} = u_{i,i}^{(0)} = 1$ for all i (see table 4.2). In the former case, if $l_{i,i}^{(0)} = 0$ or $u_{i,i}^{(0)} = 0$, for some i, $1 \leq i \geq n$, then set the entry to 1 to avoid arriving at a singular matrix.

3. Compute $A = L^{(0)}U^{(0)}$.

4. Swap a random pair of rows in the matrix A.

5. m times repeat step 4, where m is a random integer in the range $[0, n)$.

The algorithms have been implemented with C++ and built as a console application with Microsoft Visual C++ 5.0 compiler and linker. All numerical operations have been performed with double precision floating point arithmetic. The double precision representation of a number uses 64 bits: 1 for the sign, 11 for the exponent, and 52 for the mantissa. Its range is $\pm 1.7 \times 10^{308}$ with at least 15 decimal digits of precision. The test results have been collected on a Pentium-100MHz PC, running under Windows 95's DOS session. The system pseudo-random number generator functions **srand()** and **rand** () have been used to generate input matrices.

Tables 4.1 and 4.2 present the results of our experiments. 1000 random matrices of each size (from 2 to 10 in the case of small determinants and

from 2 to 10 in the case of determinants 1) have been tested. The relative errors $e_0 = \frac{|\det \tilde{A} - \det A'|}{|\det A'|}$, $e_{pys} = \frac{n^2 a^+ \epsilon}{|\det A'|}$ and $e_{new} = eN$ have been evaluated in each case, and their average values have been presented in the tables. The values of $|\det A|$ and $\frac{1}{\|\tilde{A}^{-1}\|} = \frac{1}{\|\tilde{U}^{-1}\tilde{L}'^{-1}\|}$ (average over the results of both approaches [PYS] and algorithm 4.1) have also been computed and presented in tables 4.1 and 4.2. Two integer counters V_{pys} and V_{new} have been used to keep track of the number of cases where the [PYS] algorithm and our new algorithm failed to verify the computation results, respectively. More specifically, whenever division by zero occurred in the [PYS] algorithm or whenever we observed that $e_{pys} \geq 1$, the counter V_{pys} was incremented by one, and this case was excluded from the average count of the data represented in the first five columns of the tables. Likewise, whenever division by zero occurred in algorithm 4.1 or whenever we observed that $e_{new} \geq 1$, the counter V_{new} was incremented by one and the case was similarly excluded from the average count of the data.

The tables show that the number of cases rejected by the new algorithm are consistently smaller than that of the [PYS] algorithm. For the accepted cases, the relative error estimates obtained by the new algorithm are much closer to but no smaller than the "true" relative errors.

Experiment 4.1 Comparison of average estimated errors of 1,000 random matrices. Results are shown in table 4.1.

Experiment 4.2 Comparison of average estimated errors of 1,000 random matrices with ± 1 determinants. Results are shown in table 4.2.

Alg.	e_0	e_{pys}	e_{new}	$\lvert\det A\rvert$	$\frac{1}{\lVert A^{-1}\rVert}$	V_{pys}	V_{new}
Size 2							
1	1.958e-16	1.707e-13	5.906e-14	8.781e+2	9.036	102	102
2	2.625e-16	4.454e-14	6.266e-14	8.845e+2	9.262	3	3
3	1.935e-16	3.189e-14	5.084e-14	8.834e+2	9.253	0	0
Size 3							
1	2.636e-15	2.322e-10	4.828e-12	2.400e+4	3.360	76	72
2	2.404e-15	4.011e-11	1.705e-12	2.408e+4	3.438	3	3
3	2.184e-15	1.077e-11	1.160e-12	2.402e+4	3.441	0	0
Size 4							
1	2.265e-14	4.030e-4	3.705e-10	6.943e+5	1.483	62	64
2	1.363e-14	1.756e-9	4.177e-11	6.991e+5	1.481	1	1
3	1.041e-14	1.678e-9	1.525e-11	6.984e+5	1.480	0	0
Size 5							
1	1.797e-13	3.935e-3	3.761e-9	1.563e+7	0.572	56	40
2	8.941e-14	5.184e-6	2.116e-10	1.530e+7	0.565	0	0
3	7.952e-14	6.080e-7	1.437e-10	1.530e+7	0.565	0	0
Size 6							
1	9.590e-13	1.980e-2	1.255e-8	4.585e+8	0.262	134	37
2	1.078e-12	5.420e-5	2.438e-9	4.097e+8	0.240	0	0
3	9.341e-13	1.005e-5	1.132e-9	4.097e+8	0.240	0	0
Size 7							
1	7.345e-13	5.393e-2	4.752e-8	3.433e+10	0.180	317	37
2	9.060e-12	5.940e-3	3.327e-8	2.473e+10	0.142	4	0
3	1.046e-11	7.431e-4	1.502e-8	2.463e+10	0.141	0	0
Size 8							
1	4.674e-12	1.175e-1	1.751e-5	6.615e+11	0.080	619	29
2	3.221e-11	2.448e-2	1.061e-6	5.662e+11	0.061	37	0
3	2.304e-11	1.404e-2	5.653e-8	5.541e+11	0.060	16	0
Size 9							
1	9.134e-13	2.053e-1	1.586e-8	2.701e+13	0.062	839	19
2	1.055e-11	6.904e-2	5.593e-8	1.201e+13	0.028	160	0
3	1.356e-11	3.917e-2	6.251e-8	1.108e+13	0.026	89	0
Size 10							
1	9.726e-13	2.240e-1	7.454e-9	1.963e+15	0.064	964	21
2	5.562e-12	1.304e-1	2.899e-8	8.986e+14	0.019	425	0
3	9.713e-12	1.013e-1	5.244e-8	7.089e+14	0.016	271	0

Table 4.1: Results of Experiment 4.1 (Algorithm 1, 2 and 3 stand for versions of algorithm 4.1 using LU, PLU and $PLUP_1$ factorization methods, respectively.)

Alg.	e_0	e_{pys}	e_{new}	$\lvert\det A\rvert$	$\frac{1}{\lVert A^{-1}\rVert}$	V_{pys}	V_{new}
Size 2							
1	7.904e-16	1.809e-12	1.656e-13	1.000	1.000	92	92
2	9.345e-16	1.036e-13	5.165e-13	1.000	1.000	2	2
3	9.268e-16	1.034e-13	5.155e-13	1.000	1.000	0	0
Size 3							
1	2.326e-14	2.800e-8	3.315e-10	1.000	1.000	85	76
2	3.583e-14	1.064e-9	4.950e-11	1.000	1.000	4	3
3	2.828e-14	6.154e-10	3.819e-11	1.000	1.000	1	0
Size 4							
1	1.004e-12	1.924e-3	5.165e-8	1.000	1.000	54	53
2	7.900e-13	1.285e-5	5.243e-9	1.000	1.000	0	0
3	6.655e-13	4.342e-7	2.049e-9	1.000	1.000	0	0
Size 5							
1	2.608e-11	3.990e-2	1.504e-5	1.000	1.000	152	52
2	2.507e-11	9.849e-4	3.504e-7	1.000	1.000	1	0
3	2.143e-11	2.059e-4	1.021e-7	1.000	1.000	0	0
Size 6							
1	2.809e-10	2.083e-1	3.227e-5	1.000	1.000	626	41
2	4.969e-10	1.319e-1	1.051e-5	1.000	1.000	104	0
3	4.670e-10	8.723e-2	3.762e-6	1.000	1.000	21	0
Size 7							
1	2.245e-9	3.712e-1	4.674e-4	1.000	1.000	992	41
2	4.464e-9	4.782e-1	1.818e-3	1.000	1.000	937	0
3	2.534e-9	4.393e-1	3.227e-5	1.000	1.000	885	0
Size 8							
1	NA	NA	NA	NA	NA	1000	114
2	NA	NA	NA	NA	NA	1000	8
3	NA	NA	NA	NA	NA	1000	0
Size 9							
1	NA	NA	NA	NA	NA	1000	349
2	NA	NA	NA	NA	NA	1000	54
3	NA	NA	NA	NA	NA	1000	34
Size 10							
1	NA	NA	NA	NA	NA	1000	713
2	NA	NA	NA	NA	NA	1000	281
3	NA	NA	NA	NA	NA	1000	242

Table 4.2: Results of Experiment 4.2 (Algorithm 1, 2 and 3 stand for versions of algorithm 4.1 using LU, PLU and $PLUP_1$ factorization methods, respectively. NA stands for "not available" in the case where computation of $\det A$ by the algorithm of [PYS] failed in all cases.)

4.8 Modifications and alternative approaches.

4.8.1 LDM^T-factorization.

Instead of LU-factorization of A', one may rely on its LDM^T-factorization, where L and M are unit lower triangular matrices and D is a diagonal matrix. To obtain LDM^T factorization, one may compute the matrices $D = \mathrm{diag}(u_{i,i})$ and $M^T = D^{-1}U$ and then substitute $U = DM^T$ into the LU-factorization, alternatively one may compute the LDM^T factorization directly, by the Gauss-Jordan transformation. In both cases our analysis can be easily extended (cf. [H96], pp.275-281).

4.8.2 Solution by estimating the magnitude of the diagonal perturbation.

To prove that equation (4.4) holds, it suffices to verify that the roundoff causes the perturbation of the diagonal entries of the factor U of $A' = LU$ with relative errors less than 1. The theorem of Barrlund and Sun (see [H96], page 194) gives the following sufficient condition: $\|\tilde{G}\|_2 < 1$ and simultaneously $\mathrm{diag}(|\tilde{G}|(I - |\tilde{G}|)^{-1}|\tilde{U}|) < \mathrm{diag}|\tilde{U}|$ for $\tilde{G} = \tilde{L}^{-1}(\tilde{A} - A')\tilde{U}^{-1}$.

This approach has some similarity to one of sections 4.3–4.5 (in particular $\|\tilde{G}\|$ can be estimated similarly to N^+ of section 4.5 and appendix) but requires to satisfy stronger assumptions than ones of sections 4.3–4.5 (in particular, the former assumptions imply the latter ones) and also involves a more complicated expression, $|\tilde{G}|(I - |\tilde{G}|)|U|$, whose computation has larger roundoff errors.

4.8.3 The straightforward approach of [PYS].

Unlike our approach, the paper [PYS] relies on the following equations:

$$\delta = \det(A' + E) - \det A' = \sum_{i,j} e_{i,j}\tilde{D}_{i,j}, \tag{4.9}$$

where $E = (e_{i,j}) = \tilde{A} - A'$, $\tilde{D}_{i,j} = \det \tilde{A}_{i,j}$, $\tilde{A}_{i,j}$ is the $(n-1) \times (n-1)$ matrix obtained by replacing the first $j-1$ columns of A' by ones of $\tilde{A}$ and by deleting the i-th row and the j-th column of the resulting matrix, for $i, j = 1, \ldots, n$. Then the relations $|\delta| \leq n^2 e_+ \max_{i,j} |\tilde{D}_{i,j}| \leq n^2 e_+ (\|A\| + ne_+)^{n-1}$ for a fixed matrix norm are deduced. This upper estimate for $|\delta|$, however, is overly pessimistic because it relies on the rough bound $|\tilde{D}_{i,j}| \leq (\|A\| + ne_+)^{n-1}$ (which could be only slightly improved by using the Hadamard inequality, cf. (4.6)) and on ignoring possible cancellation in the summation in (4.9). Generally, it is hard to obtain a sharp estimate for δ, and our approach benefits of proposing a solution that avoids estimating δ.

4.8.4 QR factorization versus $PLUP_1$ factorization.

The proposed algorithms and their analysis can be immediately extended based on the QR rather than $PLUP_1$ factorization of A. Some estimates for the cost of computing the factorization and for the perturbation of the input, which would accommodate the roundoff errors, can be taken from [PYS], but they can be refined by carefully estimating from above the norm $||R^{-1}Q^T||$ (cf. [H96], chapter 18). The analysis gives preference to relying on the Householder transformation in order to compute the QR factorization. (Actually, we only need the diagonal entries of R for our purpose of

the sign computation.) Relying on the QR factorization has advantage over using triangular factorizations when the sign of $\det A$ must be computed for a dynamically updated matrix A (see [GGMS], [PYS]). To decrease the roundoff errors, one may apply a scaled version of QR factorization, making it free of square root computation (cf. [C]).

4.8.5 Sign determination via LDL^T factorization of A^TA.

Computation of the LDL^T-factorization of the symmetric matrix A^TA is simple and has very good numerical stability (cf. [GL96], pp.138-139, and [H96], pp. 207-209). Namely, the matrix of the roundoff errors of computing A^TA has norm bounded from above by $\gamma_n||A|| \cdot ||A^T||$ ([H96], p.78), and the matrix of the roundoff errors of computing the LDL^T factors of A^TA is bounded by $\frac{\gamma_{n+1}}{1-\gamma_{n+1}} \sum_i a_{i,i}^2$. Thus, such a factorization may serve as a basis for good numerical approximation of $(\det A)^2 = \det(A^TA) = \prod_i D_{i,i}$ and then of $|\det A|$.

Let $|\tilde{d}|$ denote the approximation to $|\det A|$ computed in this way and let Δ be a strict upper bound on the approximation error, so that $d = \det A$ lies in the ranges $(-|\tilde{d}| - \Delta, -|\tilde{d}| + \Delta)$ and/or $(|\tilde{d}| - \Delta, |\tilde{d}| + \Delta)$. Now if $|\tilde{d}| < \Delta$, then we arrive at an upper bound $|\det A| < 2\Delta$ (cf. section 4.4). Otherwise we choose a prime $p \geq 4\Delta$ and let $a \bmod p$ (for a real a) denote a unique real number in the semi-open interval $[-p/2, p/2)$. Then we deduce that $(2|\tilde{d}|) \bmod p \geq 2\Delta$ and compute the values $|\tilde{d}|_p = |\tilde{d}| \bmod p$ and $d_p = (\det A) \bmod p$. (To obtain d_p, we first compute (modulo p) a PLU factorization of A and $\mathrm{sign}(\det P)$ and then compute $d_p = ((\prod_i (u_{i,i} \bmod$

$p))\text{sign}(\det P))$ mod p. Note that the computation modulo p only requires $2\lceil \log_2 p \rceil$-bit precision.) Clearly, we have either

$$-\Delta < \det A - |\tilde{d}| = (d_p - |\tilde{d}|_p) \bmod p < \Delta$$

or

$$-\Delta < \det A + |\tilde{d}| = (d_p + |\tilde{d}|_p) \bmod p < \Delta.$$

These two cases cannot occur simultaneously since

$$(2|\tilde{d}|) \bmod p = (2|\tilde{d}|_p) \bmod p \geq 2\Delta,$$

and we may easily test which of them actually occurs. In the former case, $\det A > |\tilde{d}| - \Delta \geq 0$, and we output $\text{sign}(\det A) = 1$. Otherwise, $\det A < |\tilde{d}| - \Delta \leq 0$, and we output $\text{sign}(\det A) = -1$.

4.8.6 Computation of the inverse.

To compute the inverse $\tilde{A}^{-1} = \tilde{U}^{-1}\tilde{L}^{-1}$, we may rely on the following simple observations (cf. [H96], pp.170-174).

Proposition 4.2 *Let $T = (t_1, \ldots, t_n)$ be an $n \times n$ proper lower (respectively, upper) triangular matrix (with zeros on its diagonal). Let T_i denote the matrix obtained by zeroing all the columns of T except for its i-th (respectively, $(n+1-i)$-th) column, $i = 1, ..., n$. Then*

$$I + T = (I + T_1)(I + T_2) \cdots (I + T_{n-1}),$$

$$(I + T_i)^{-1} = I - T_i, \quad i = 1, \ldots, n-1.$$

Corollary 4.4 *Under the assumptions of proposition 4.2, we have*

$$(I + T)^{-1} = (I - T_{n-1})(I - T_{n-2}) \cdots (I - T_1). \tag{4.10}$$

Based on corollary 4.4, we may compute $\tilde{L}^{-1}$ and $\hat{U}^{-1}$ where $\hat{U}$ is a unit triangular matrix obtained from $\tilde{U}$ by scaling the rows and/or columns of $\tilde{U}$. The overall roundoff error estimate for computing $\tilde{L}^{-1}$ depends on whether we multiply the matrices $I - T_i$ in (4.10) for $T = \hat{L} - I$ from left to right or from right to left and similarly for $T = \hat{U} - I$. (In the latter stage the variation also depends on the choice of the scaling of $\tilde{U}$, which defines the transition to the matrix $\hat{U}$.)

Chapter 5

Summary

Although the three problems we studied are seemingly unrelated, the approaches we have taken to solve them have much in common, that is, we rely on combining algebraic and numerical methods to improve the algorithm design. We are encouraged by the results we have achieved and strongly believe that further study in this direction will allow us to discover more efficient algorithms in many areas of practical applications. We hope that our work will motivate the two groups of researchers and algorithm designers in the areas of algebraic and numerical computing, respectively, to increase their interactions with each other.

In the near future, our ongoing research will focus on further enhancements of our existing algorithms and the ways of widening the application of our technique to other computational problems. In the problems of multipoint polynomial evaluation and interpolation, we will try to extend the results to a larger class of inputs, that is, we will try to relax the restrictions on the class of input values for which our algorithms work efficiently. In the area of modular arithmetic in linear algebra, immediate extension of our

techniques will be to find their applications to other basic linear algebra operations. In the case of the evaluation of the sign of matrix determinant, we will continue our work on improving on the error estimates in our numerical algorithms to lower the possibility of false rejections of potentially correct results.

Bibliography

[A] K.E. Atkinson, *Introduction to Numerical Analysis*, Wiley, 1978.

[ABDPY] F. Avnaim, J.-D. Boissonnat, O. Devillers, F. P. Preparata, M. Yvinec, Evaluating Signs of Determinants Using Single-Precision Arithmetic, to appear in *Algorithmica.*

[AF] D. Avis, K. Fukuda, A Pivoting Algorithm for Convex Hulls and Vertex Enumeration of Arrangements and Polyhedra, *Discrete Comput. Geometry*, 8, 295–313, 1992.

[AHU] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, 1974.

[B68] E. H. Bareiss, Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination, *Math. of Comp.*, 22, 565–578, 1968.

[BEPP97] H. Brönnimann, I. Z. Emiris, V. Y. Pan, S. Pion, Computing Exact Geometric Predicates Using Modular Arithmetic

with Single Precision, *Proc. 13th Ann. ACM Symp. on Computational Geometry*, ACM Press, New York, 1997.

[BEPP98] H. Brönnimann, I. Z. Emiris, V. Y. Pan, S. Pion, Sign Determination in Residue Number Systems, *Theoretical Computer Science*, 1998 (to appear).

[BKMNSU] C. Burnikel, J. Könnemann, K. Mehlhorn, S. Näher, S. Schirra, C. Uhrig, Exact Geometric Computation in *LEDA*, *Proc. 11th Ann. ACM Symp. on Computational Geometry*, C18–C19, 1995. Package available at *http://www.mpi-sb.mpg.de/LEDA/leda.html.*

[BP] D. Bini and V. Y. Pan, *Polynomial and Matrix Computations, Volume 1: Fundamental Algorithms*, Birkhauser, Boston, 1994.

[BPY] D. Bini, V. Y. Pan and Y. Yu, Certified Numerical Computation of the Sign of a Matrix Determinant, *Submitted*, 1998.

[C] K.L. Clarkson, Safe and Effective Determinant Evaluation, *Proc. 33rd Annual IEEE Symp. on Foundations of Computer Science*, 387-395, IEEE Computer Society Press, 1992.

[CdB] S.D. Conte, C. de Boor, *Elementary Numerical Analysis: An Algorithmic Approach*, McGraw Hill, 1980.

[CKa] D. G. Cantor, E. Kaltofen, On Fast Multiplication of Polynomials over Arbitrary Rings, *Acta Inf.*, 28, 7, 697–701, 1991.

[CMSW79] A. K. Cline, C. B. Moler, G. W. Stewart, J. H. Wilkinson, An Estimate for the Condition Number of a Matrix, *SIAM J. Numerical Analysis*, 16, 368–375, 1979.

[D] P. Duhamel, Implementation of "Split Radis" FFT Algorithms for Complex, Real and Real Symmetric Data, *IEEE Trans. Acoust., Speech, Signal Processing*, 34, pp.285-295, 1986.

[D83] JJ. D. Dixon, Estimating Extremal Eigenvalues and Conditional Numbers of Matrices, *SIAM J. on Numerical Analysis*, 20, 4, 812-814, 1983.

[DB] G. Dahlquist, Å. Björck, *Numerical Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[DLa] M. Deza, M. Laurent, Applications of Cut Polyhedra, *J. of Computational and Applied Math.*, 55, 1, 191–216 and 55, 2, 217–247, 1994.

[DL97] M. M. Deza, M. Laurent, *Geometry of Cuts and Metrics*, Springer, Berlin, 1997.

[Du] E. Durand, *Solutions Numériques des Équations Algébriques. Vol. II: Systèmes de Plusieurs Équations. Valeurs Propres des Matrices*, Masson et Cie, Paris, 1961.

[E67] J. Edmonds, Systems of Distinct Representatives and Linear Algebra, *J. Res. Nat. Bur. Standards*, Sect. B, 71, 4, 241–245, 1967.

[EC] I.Z. Emiris, J.F. Canny, A General Approach to Removing Degeneracies, *SIAM J. Computing*, 24(3), 650–664, 1995.

[Em] I.Z. Emiris, A Complete Implementation for Computing General Dimensional Convex Hulls, *Intern. J. Computational Geom. & Applications*, 1997, to appear. (Preliminary version as Tech. Report 2551, INRIA Sophia-Antipolis, France, 1995.)

[EPY] I. Z. Emiris, V. Pan and Y. Yu, Modular Arithmetic for linear Algebra Computations in the Real Field, *J. Symbolic Computation*, 11, 1997.

[ES] R. M. Erdahl, V. H. Smith (editors), *Density Matrices and Density Functionals*, Proc. of the A. John Coleman Symp., Reidel, Dordrecht, 1987.

[EY] C. Eckart, G. Young, A Principal Axis Transformation for Non-Hermitian Matrices, *Bull. Amer. Math. Society* (New Series), 45, 118–121, 1939.

[Fo] L. Fox, *An Introduction to Numerical Linear Algebra*, Clarendon Press, Oxford, 1964.

[FP] M.J. Fischer, M.S. Paterson, String Matching and Other Products, *SIAM-AMS Proc.*, 7, 113–125, 1974.

[FR] K. Fukuda, V. Rosta, Combinatorial Face Enumeration in Convex Polytopes, *Computational Geometry, Theory and Applications*, 4, 191–198, 1994.

[FVW] S. Fortune, C. J. Van Wyk, Efficient Exact Arithmetic for Computational Geometry, *Proc. 9th Ann. ACM Symp. on Computational Geometry*, 163–172, 1993.

[G] R.T. Gregory, *Error-Free Computation: Why It Is Needed and Methods for Doing It*, Krieger, New York, 1980.

[GGMS] P.E. Gill, G.H. Golub, W. Murray, M.A. Saunders, Methods for Modifying Matrix Factorizations, *Math. of Computation*, 28, 505-535, 1974.

[GKK] I. Gohberg, T. Kailath, and I. Koltracht, Efficient Solution of Linear Systems of Equations with Recursive Structure, *Linear Algebra and Its Appl.*, 80, pp. 81-113, 1986.

[GKO] I. Gohberg, T. Kailath, and V. Olshevsky, Fast Gaussian Elimination with Partial Pivoting for Matrices with Displacement Structure, *submitted for publication*, 1994.

[GL] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland, 1989.

[GL96] G.H. Golub, C.F. Van Loan, *Matrix Computations*, Johns Hopkins Univ. Press, Baltimore, MD, 1996 (third edition).

[GO] I. Gohberg and V. Olshevsky, Complexity of Multiplication with Vectors for Structured Matrices, *Linear Algebra and Its Appl.*, 202, pp. 163-192, 1994.

[GO1] I. Gohberg and V. Olshevsky, Fast Inversion of Chebyshev-Vandermonde Matrices, *Numerische Math.*, 67, 1, pp. 71-92, 1994.

[H] G. Heinig, Inversion of Generalized Cauchy Matrices and Other Classes of Structured Matrices, *IMA Volume: Linear Algebra for Signal Processing in Math and Its Applications*, 69, pp. 95-114, Springer, 1994.

[H96] N.J. Higham, Accuracy and Stability of Numerical Algorithms, SIAM Publications, Philadelphia, 1996.

[IK] E. Isaacson, H. Keller, *Analysis of Numerical Methods*, Wiley, New York, 1966.

[J] R.L.Johnston, *Numerical Methods : A Software Approach*, John Wiley & Sons, New York, 1982.

[K66] W. Kahan, Numerical Linear Algebra, *Canadian Math. Bull.*, 9, 757–801, 1966.

[Kn] D.E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, 2, Addison-Wesley, Reading, Massachusetts, 1981.

[KO] T. Kailath and V. Olshevsky, Displacement Structure Approach to Chebyshev-Vandermonde and Related Matrices, to appear in *Integral Equations and Operator Theory.*

[KW92] J. Kuczyński, H. Woźniakowski, Estimating the Largest Eigenvalue by the Power and Lanczos Algorithms with a Random Start, *SIAM J. on Matrix Analysis and Applications*, 13, 4, 1094-1122, 1992.

[KW94] J. Kuczyński, H. Woźniakowski, Probabilistic Bounds on the Extremal Eigenvalues and Condition Number by the Lanczos Algorithm, *SIAM J. on Matrix Analysis and Applications*, 15, 2, 672-691, 1994.

[Ma] R. H. Macmillan, A New Method for the Numerical Evaluation of Determinants, *J. Roy. Aeronaut. Soc.*, 59, 772ff, 1955.

[MA] R. E. M. Moore, I. O. Angell, Voronoi Polygons and Polyhedra, *J. of Computational Physics*, 105, 301–305, 1993.

[MC] R.T. Moenck, J.H. Carter, Approximate Algorithms to Derive Exact Solutions to Systems of Linear Equations, *Proc. EUROSAM, Lecture Notes in Computer Science*, 72, 65-73, Springer, Berlin, 1979.

[Mu] T. Muir, The Theory of Determinants in Historical Order of Development, four volumes bound as two (I, 1693–1841; II, 1841–1860; III, 1861–1880; IV, 1881–1900), Dover, New York, 1960; *Contributions to the History of Determinants*, 1900–1920, Blackie and Son, London, 1930 and 1950.

[N] H. J. Nussbaumer, Fast Polynomial Transform Algorithms for Digital Convolution, *IEEE Trans. on Acoust., Speech, Signal Processing*, 28, 2, pp.205-215, 1980.

[Of62] Yu. Ofman, On the Algorithmic Complexity of Discrete Functions, *Dokl. Acad. Nauk SSSR*, 145, 1, 48–51, 1962. (English translation in *Soviet Physics-Dokl.*,7, 7, 589–591, 1963.)

[Of65] Y.P. Ofman, Universal Automaton (in Russian), *Moscow Math Soc. (Trydy)*, 14, 186–199, 1965.

[P84] V.Y. Pan, *How to Multiply Matrices Faster*, Lecture Notes in Computer Science, 179, Springer, Berlin, 1984.

[P88] V.Y. Pan, Computing the Determinant and the Characteristic Polynomial of a Matrix via Solving Linear Systems

of Equations, *Information Processing Letters*, 28, 2, 71-75, 1988.

[P90] V. Y. Pan, Computations with Dense Structured Matrices, *Math. of Computation*, 55, pp. 179-190, 1990.

[P91] V. Y. Pan, Complexity of Algorithms for Linear Systems of Equations, *in Computer Algorithms for Solving Linear Algebraic Equations, The State of the Art*, edited by E. Spedicato, NATO ASI Series, Series F: Computer and Systems Sciences, 77, pp.27-56, Springer, Berlin, 1991.

[P92] V. Y. Pan, Complexity of Computations with Matrices and Polynomials, *SIAM Review.*, 34, 2, pp. 225–262, 1992.

[P92a] V. Y. Pan, Can We Utilize the Cancellation of the Most Significant Digits?, Tech. Report TR-92-061, *The International Computer Science Institute,* Berkeley, CA, 1992.

[P92b] V.Y. Pan, Parametrization of Newton's Iteration for Computations with Structured Matrices and Applications, *Computers and Math. (with Applications)*, 24, 3, 61-75, 1992.

[P93] V. Y. Pan, Decreasing the Displacement Rank of a Matrix, *SIAM Journal of Matrix Analysis and Appl.*, 14, 1, pp. 118-121, 1993.

[P93a] V. Y. Pan, On Binary Segmentation for Matrix and Vector Operations, *Computer and Math. (with Applications)*, 25, 3, pp. 69-71, 1993.

[P95] V. Y. Pan, An Algebraic Approach to Approximate Evaluation of a Polynomial on a Set of Real Points, *Advances in Computational Math.*, 3, pp. 41-58, 1995.

[P,a] V. Y. Pan, Algebraic and Numerical Techniques for the Computation of Matrix Determinant, manuscript, 1994 (revised in 1996).

[PR] V. Y. Pan, J. Reif, Compact Multigrid, *SIAM J. on Sci. and Statist. Comp.*, 13, 1, pp.119–127, 1992.

[PR,a] V. Y. Pan, J. Reif, Generalized Compact Multigrid, *Computer and Math. (with Applications)*, 25, 9, pp. 3–5, 1993.

[PS] V. Y. Pan, R. Schreiber, An Improved Newton Iteration for the Generalized Inverse of a Matrix, with Applications, *SIAM J. Sci. Stat. Comput.*, 12, 5, 1109-1131, 1991.

[PSLT] V. Y. Pan, A. Sadikou, E. Landowne and O. Tiga, A New Approach to Fast Polynomial Interpolation and Multipoint Evaluation, *Computers Math. with Appl.*, 25, 9, pp. 25-30, 1993.

[PYS] V. Y. Pan, Y. Yu, C. Stewart, Algebraic and Numerical Techniques for the Computation of Matrix Determinants, *Computers & Math. (with Applications)*, 34, 1, 43-70, 1997.

[PZHY] V. Y. Pan, A. Zheng, X. Huang and Y. Yu, Fast Multipoint Polynomial Evaluation and Interpolation via Computations with Structured Matrices, *Annals of Numerical Mathematics*, 4, pp. 483-510, 1997.

[R52] J. B. Rosser, A Method of Computing Exact Inverses of Matrices with Integer Coefficients, *J. Res. Na. Bur. Standards*, Sect. B, 49, 349–358, 1952.

[R85] V. Rokhlin, Rapid Solution of Integral Equations of Classical Potential Theory, *Journal of Computational Physics*, 60, pp. 187-207, 1985.

[R88] V. Rokhlin, A Fast Algorithm for the Discrete Laplace Transformation, *Journal of Complexity*, 4, pp. 12-32, 1988.

[W] J. M. Wilkinson, *Algebraic Eigenvalue Problem,* Clarendon Press, Oxford, 1965.

[Win] S. Winograd, *Arithmetic Complexity of Computations*, SIAM, Philadelphia, 1980.

[Y] C. Yap, Towards Exact Geometric Computation, *Computational Geometry, Theory and Applications,* 7, 3–23, 1997.

[YD] C. K. Yap, T. Dubhe, The Exact Computation Paradigm, D. Du and F. Hwang, editors, *Computing in Euclidean Geometry*. World Scientific Press, 1995.

[YG] D.M. Young, R.T. Gregory, *A Survey of Numerical Mathematics II*, Addison-Wesley, Reading, Mass., 1973.