A Complete Analysis of Clarkson's Algorithm for Safe Determinant Evaluation

Hervé Brönnimann, Mariette Yvinec

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A complete analysis of Clarkson's algorithm for safe determinant evaluation

Hervé Brönnimann, Mariette Yvinec

N° 3051

Novembre 1996

THÈME 2

ISSN 0249-6399

INRIA
SOPHIA ANTIPOLIS

A complete analysis of Clarkson's algorithm for safe determinant evaluation

Hervé Brönnimann, Mariette Yvinec*

Thème 2 — Génie logiciel
et calcul symbolique
Projet Prisme

Rapport de recherche n ° 3051 — Novembre 1996 — 36 pages

Abstract: In this paper, we give a complete and self-contained analysis of Clarkson's algorithm that performs safe and efficient determinant evaluation of an $n \times n$ matrix with integer coefficients that can be expressed with b bits. Clarkson's original paper is generally felt difficult to read. We complete the gaps in his exposition, simplifying the analysis where we can. The number of extra bits needed by this analysis is roughly equivalent to the number of bits needed by Clarkson's analysis. We show that the algorithm performs sign evaluation correctly if $b + O(n)$ bits are available. We give a table of the maximum numbers b of bits available for the entries as a function of n, when the arithmetic is performed using a floating point processor complying with the IEEE 754 standard for double precision arithmetic (with 53 bits available for the mantissa). We also gain some insight into the practical behavior of the algorithm by experimenting. In particular, we provide experimental evidence that the algorithm evaluates correctly the sign of determinants of order up to 15 with 48-bit coefficients.

Key-words: Computational geometry, exact arithmetic

(Résumé : tsvp)

Research by Hervé Brönnimann was supported in part by ESPRIT Basic Research Actions 7141 (ALCOMII) and 6546 (PROMotion).

* INRIA, B.P.93, 06902 Sophia-Antipolis cedex (France), Phone: +33 4 93 65 77 75, +33 4 93 65 77 49 E-mail: {Herve.Bronnimann,Mariette.Yvinec}@sophia.inria.fr.

Unité de recherche INRIA Sophia Antipolis
2004 route des Lucioles, BP 93, 06902 SOPHIA ANTIPOLIS Cedex (France)
Téléphone : 04 93 65 77 77 – Télécopie : 04 93 65 77 65

Une analyse exhaustive de l'algorithme de Clarkson pour calculer le signe d'un déterminant

Résumé : Nous donnons une analyse complète de l'algorithme de Clarkson qui calcule le signe du déterminant d'une matrice $n \times n$ de manière exacte, si les coefficients de la matrice sont entiers et peuvent s'exprimer avec au plus b bits. L'analyse donnée par Clarkson est généralement ressentie comme difficile d'accès. Bien que similaire, notre analyse utilise des techniques plus élémentaires, une notation plus précise et se situe à un niveau de détail plus raffiné. Le nombre de bits supplémentaires requis par notre analyse est légèrement moins bon mais comparable avec celui de l'analyse de Clarkson. Nous donnons une table du nombre b de bits sur les entrées pour quelques valeurs de n pour lequel l'algorithme est garanti de trouver le signe du déterminant, si les calculs intermédiaires s'effectuent sur 53 bits (en `double` suivant le standard 754 d'IEEE). Des expériences pratiques donnent un peu de compréhension sur le comportement réel de l'algorithme. En particulier, nous montrons expérimentalement que l'algorithme fonctionne correctement pour des entrées sur 48 bits jusqu'à l'ordre 15.

Mots-clé : Géométrie algorithmique, arithmétique exacte

1 Introduction

Computing the sign of a determinant is a geometric primitive used by many geometric predicates: orientation test, in-circle and in-sphere tests, etc. If computed naively, it is well-known that roundoff errors may lead to the wrong sign and possibly to inconsistencies, causing the algorithm to fail. An algorithm that avoids these inconsistencies is called a *robust* algorithm. Robustness can be achieved in several ways [11, 12, 14]. Fortune and Van Wyk [9], Yap [15], Burnikel and coll. [3], Avnaim and coll. [1] and many others advocate the use of exact arithmetic to avoid all robustness issues. Unfortunately, naive implementations of exact arithmetic can be quite slow and two approaches have been designed, which strengthen one another.
To take advantage of the usually good precision of the fast floating-point implementation, arithmetic filters are designed that can tell whether the answer they compute is safe. In most cases, this will provide a fast and reliable answer. This approach is used in the LN package by Fortune and Van Wyk [9] and has been shown experimentally to provide a substantial speed-up. Devillers and Preparata investigate the theoretical behavior of some filters [5].
In near-degenerate cases, however, we must resort to exact arithmetic. Burnikel [2] and Yap [15] provide extremely powerful exact arithmetic on algebraic numbers. To not pay the full price of these multi-precision packages, exact implementation of particular geometric predicates have been designed. Clarkson [4], and Avnaim and coll. [1] have provided two completely different methods for safely evaluating the sign of the determinant of a matrix with integer coefficients. Shewchuk [13] derives exact implementations of common geometric predicates with entries not necessarily integral, but with a wide range of exponents. All these techniques are adaptive, in that their running time depends on the input.
In this report, we focus on Clarkson's method and provide a complete analysis, simplifying Clarkson's original exposition wherever we can. We close the gaps in the argumentation. In particular, Clarkson's algorithm does not terminate if the determinant is null. Although not published, this feature is fixed in the robust code written by Clarkson to compute convex hulls in any dimension, which uses his algorithm. We discuss this issue in section 8. We also provide some experimental insight into the behavior of the algorithm.

2 Overview of the algorithm

Throughout this paper, the vectors are in $\mathbb{R}^n$, where n is fixed and small ($n < 32$). The dot product of two vectors x and y is denoted by $x \cdot y$. A linear combination of $x_1, \ldots, x_n$ is denoted by $LC(x_1, \ldots, x_n)$ when the coefficients don't really matter.

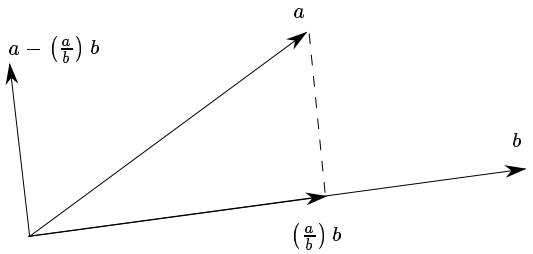
Reduction of two vectors. The basic operation that will be used in the algorithm is the reduction of a vector a by a vector b. This operation takes two vectors a and b ($b \neq 0$) and returns a scalar $\left(\frac{a}{b}\right)$, computed by:

$$\left(\frac{a}{b}\right) := \frac{a \cdot b}{b \cdot b} = \left(\frac{a}{\|b\|}\right) \cdot \left(\frac{b}{\|b\|}\right).$$

Obviously, the following inequality is valid for any two vectors a and b ($b \neq 0$):

$$\left|\frac{a}{b}\right| \leq \frac{\|a\|}{\|b\|}.$$

The geometric meaning of this operation is depicted in the figure below. Its main interest is that the vector $a - \left(\frac{a}{b}\right)b$ is orthogonal to b.



Gram-Schmidt reductions. We give a complete analysis of an algorithm by Clarkson [4]. The algorithm is based on the Gram-Schmidt orthonormalization algorithm, which computes an orthogonal basis $C = (c_1, \ldots, c_n)$ given an independent family of vectors $A = (a_1, \ldots, a_n)$. The family C has the additional property that $c_k = a_k + \mathrm{LC}(a_1, \ldots, a_{k-1})$, and similarly that $a_k = c_k + \mathrm{LC}(c_1, \ldots, c_{k-1})$. If A is not independent, then there is a subfamily of A such that $\mathrm{LC}(a_1, \ldots, a_k) = 0$. For such a minimal index k, at the k=th reduction step that computes a_k, the algorithm can output the coefficients in the linear combination $\mathrm{LC}(a_1, \ldots, a_k) = 0$. The Gram-Schmidt algorithm is given in figure 1a. The vector $c_k^{(j)}$ is orthogonal to $(a_j, \ldots, a_{k-1})$, so we can think of the reduction as projecting onto the orthogonal

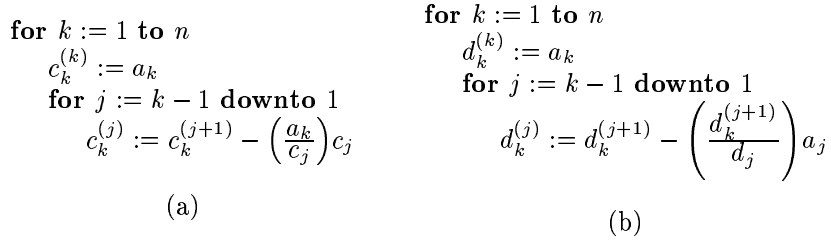


Figure 1: The (a) Gram-Schmidt and (b) modified Gram-Schmidt reductions.

of a_k, then (a_k, a_{k-1}), etc., by at each time subtracting the component of a_k along c_j, $j = k, \ldots, 1$. Since $c_1, \ldots, c_{k-1}$ are orthogonal, the resulting c_k must also be orthogonal to these vectors. It is either null, or it extends the orthogonal family C.

Modified Gram-Schmidt reductions. Clarkson also uses a modified Gram-Schmidt reduction [4], given in figure 1b. Even though the values of $c_k^{(j)}$ and $d_k^{(j)}$ differ for $j = k-1, \ldots, 2$, this procedure computes exactly the same base $\{c_1, \ldots, c_k\} = \{d_1, \ldots, d_k\}$. Indeed, when computing d_k, we already know by induction that $a_j = d_j + \mathrm{LC}(d_1, \ldots, d_{j-1})$, so that $\frac{a_j}{d_j} = 1$ for all $j < k$; after the j-th iteration in the loop, we have

$$\left(\frac{d_k^{(j)}}{d_j}\right) = \left(\frac{d_k^{(j+1)}}{d_j}\right) - \left(\frac{d_k^{(j+1)}}{d_j}\right)\left(\frac{a_j}{d_j}\right) = 0.$$

The major difference is that d_k is computed as a linear combination: $d_k = a_k - \mathrm{LC}(a_1, \ldots, a_{k-1})$, whereas c_k was computed as a linear combination $c_k = a_k - \mathrm{LC}(c_1, \ldots, c_{k-1})$. If the coefficients in $\mathrm{LC}(a_1, \ldots, a_{k-1})$ are rounded, but the linear combinations are computed using exact arithmetic, the computed orthogonal family D has exactly the same determinant as A.

Clarkson's algorithm. The idea behind Clarkson's algorithm is to compute a family B that approximates C using floating point arithmetic with $b + O(n)$ bits, but not from the original family A because that can lead to unsafe evaluations. In order to enforce that the determinant will be computed with a small relative error, we must amplify the component of a_k orthogonal to $a_1, \ldots, a_{k-1}$. One simple way to do this is to multiply a_k by a huge factor, but this also increases the numbers of bits required to

for $k := 1$ **to** n
  loop
  $b_k^{(k)} := a_k$
  for $j := k-1$ **downto** 1
    $b_k^{(j)} := fl\left(b_k^{(j+1)} - fl\left(fl\left(\frac{a_k}{b_j}\right) b_j\right)\right)$
  if $fl\,(a_k \cdot a_k) \leq 2\,fl\left(b_k^{(1)} \cdot b_k^{(1)}\right)$
    $b_k := b_k^{(1)}$
    exit loop
  Compute some integer $s \geq 2$
  $a_k^{(k)} := s \times a_k$
  for $j := k-1$ **downto** 1
    $a_k^{(j)} := a_k^{(j+1)} - \left\lceil fl\left(\frac{a_k^{(j+1)}}{b_j}\right)\right\rfloor a_j$
  $a_k := a_k^{(1)}$
  end loop

Figure 2: Clarkson's algorithm.

represent a_k. The solution of choice is to increase a_k by an appropriate integer factor s, and then reduce sa_k by $a_1, \ldots, a_{k-1}$ using the modified Gram-Schmidt reductions, so as to amplify only the orthogonal component without changing the sign of the determinant. (The choice of the appropriate value is discussed in section 5.2.) Doing this a number of times will eventually lead to discover that $a_k = \mathrm{LC}(a_1, \ldots, a_{k-1})$, or to reduce a_k into some vector b_k that well approximates c_k. The process is depicted in figure 3. If the reduction of sa_k is computed using exact arithmetic, the value of the determinant of A is only multiplied by s, and its sign does not change, so the algorithm is correct. The algorithm is summarized in figure 2. In the algorithm, the notation $fl\,(x)$ denotes the real which is the nearest floating point approximation of a real x, and is extended component-wise to vectors for linear functions. The integer part of a real x is denoted by $\lceil x \rfloor$ and is defined as the smallest integer greater than or equal to $x - \frac{1}{2}$.

Remark. We have changed Clarkson's algorithm. In [4], Clarkson uses modified Gram-Schmidt reductions both for b_k and a_k. In fact, there is no need to use the

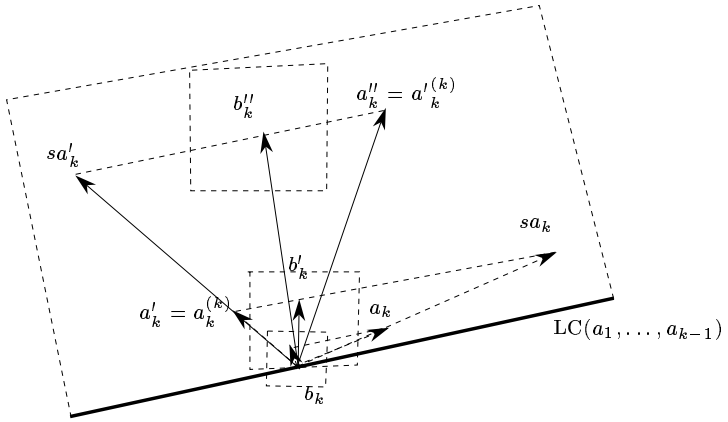


Figure 3: The process of reducing vector a_k is shown. At any time, a_k remains in the bounding box, but its component along the orthogonal of $(a_1, \ldots, a_{k-1})$ increases at each step. The computed vector b_k lies in the square box. At the first and second steps, the box does not ensure that the sign of the determinant is known safely, but it becomes so after the third reduction of a_k.

modified orthonormalization for b_k, and the analysis is simpler if not. So we have changed the reductions of b_k and we use the usual Gram-Schmidt algorithm.

Overview of the analysis. To analyze the algorithm, we proceed in three stages. In a preliminary section, we first recall the tools at hand, such as the relative error in computing dot products and reductions using floating point arithmetic.
Our first task is to bound the growth of $\left\|b_k^{(j)}\right\|$ in the approximate reduction process, and to use this bound to derive a bound on the relative error with respect to the vectors c_k (section 4). More precisely, our duty is to bound $\|b_k - c_k\|$, so as to control the error made by the approximate reduction. For this, we introduce the error quantities

$$E = B - C, \qquad e_k = b_k - c_k, \qquad e_k^{(j)} = b_k^{(j)} - c_k^{(j)}.$$

Our second task is to evaluate the size of the vectors a_k and $a_k^{(j)}$, so as to bound the precision of the integer arithmetic needed by the algorithm (section 5). The bound on $a_k^{(j)}$ is given in terms of the norm of the matrix C, or more exactly of the first k

columns of this matrix. We introduce the quantities

$$S_k^c = \sum_{1 \le j < k} c_j \cdot c_j, \qquad S_k^b = fl\left(\sum_{1 \le j < k} fl\,(b_j \cdot b_j)\right).$$

In fact, S_k^b is an approximation of S_k^c, and we can bound the relative error using the bounds established in the first part of the analysis (see section 5.4).
Finally, once we know that the algorithm computes the sign correctly if $b+O(n)$ bits are available for the floating point and exact integer arithmetic, we need only show that the algorithm terminates and that its running time is roughly $O(bn^3)$.

3 Floating point arithmetic.

A binary floating point representation on b bits of a real number is a representation of the form $\pm 1.d_1 d_2 \ldots d_b \times 2^e$, where each d_i denotes a single binary digit and e denotes the exponent. On most machines, the precision is fixed and b is a constant (52 for the IEEE 754 standard). We do not address the issues of overflow and underflow so the exponent e may vary in the range $]-\infty, \infty[$. Underflows and overflows do not occur in our implementation because we consider integer input. Goldberg [10] surveys floating point formats in detail, particularly the IEEE 754 standard.
This representation implies that only a subset $\mathbb{F}$ of the reals in $\mathbb{R}$ can be represented. On most machines, *exact rounding* is provided: this means that a number x is represented by its closest element in $\mathbb{F}$, which we denote $fl\,(x)$ (ties are broken arbitrarily). When performing an arithmetic operation, the result is rounded exactly. If $\oplus$, $\ominus$, $\otimes$ and $\oslash$ denote the four operations as implemented by the machine, this means that for any numbers x and y in $\mathbb{F}$, we have

$$x \oplus y = fl\,(x+y),\ x \ominus y = fl\,(x-y),\ x \otimes y = fl\,(xy),\ x \oslash y = fl\,(x/y).$$

The *unit roundoff* $\mathbf{u}$ bounds the relative precision to which a number is approximated (if no underflow occurs). Thus for any number x in $\mathbb{R}$, we have $|fl\,(x) - x| \le \mathbf{u} \cdot |x|$. This means that the result of a machine operation is approximated with relative error $\mathbf{u}$. We will make heavy use of this property in our error analysis.
We have bounds on the relative errors made for the four operations. It remains to obtain bounds for the two basic vector operations used by the algorithm: dot products and reduction. The error made when computing a dot product is well known, and can be found in the book by Forsythe and Moler [7].

Lemma 3.1 *Assume that* $n\mathbf{u} \leq 0.01$*, and that dot products of vectors in* $\mathbb{F}^n$ *are computed using the following scheme:*

$$fl\,(x \cdot y) = fl\,(fl\,(x_1 y_1) + fl\,(fl\,(x_2 y_2) + \cdots)).$$

Then we have

$$|fl\,(x \cdot y) - x \cdot y| \leq 1.01 n\mathbf{u}\|x\|\|y\|.$$

Remark 1. Since the hypothesis $n\mathbf{u} \leq 0.01$ is as good as $n\mathbf{u} < 2^{-46}$ for $n < 32$ and $\mathbf{u} = 2^{-53}$, we may wonder what happens on the factor 1.01 if we use a better bound. In fact, the number 1.01 decreases but of course remains greater than 1, so the impact on the following analysis is negligible.

We will also need a bound on the floating point approximation on the reduction factors. This bound follows immediately from the above lemma because a reduction factor is simply a quotient of two dot products.

Lemma 3.2 *Assuming that* $n\mathbf{u} \leq 0.01$*, and that reduction factors of vectors in* $\mathbb{F}^n$ *are computed using the following scheme:*

$$fl\left(\tfrac{x}{y}\right) = fl\left(\frac{fl\,(x \cdot y)}{fl\,(y \cdot y)}\right),$$

then we have

$$\left|fl\left(\tfrac{x}{y}\right) - \frac{x}{y}\right| \leq \frac{\|x\|}{\|y\|} 1.01(2n+2)\mathbf{u}.$$

Remark 2. As is also stated in [7], the relative error made when computing a dot product can be substantially decreased by using quadruple precision for the computation of $x \cdot y$, then rounding back to double precision. If we assume that the dot products are computed in this way, although double precision is completely encapsulated within the dot product computation and may not be otherwise assumed in the algorithm, then this relative error becomes less than $1.01\mathbf{u}$, saving a factor of n. Also, the bound on the error incurred when computing reduction factors drops to $3.03\mathbf{u}$. We will see, however, that the impact on the number of extra bits needed is negligible.

4 Bounding the error $E = B - C$

The computation of b_j is referred to as the j-th stage of the algorithm. In this section, we assume that all the values $b_1, \ldots, b_{k-1}$ have been computed in the first $k-1$ stages of the algorithm. In particular, since the algorithm exited the j-th stage because $fl\,(a_j \cdot a_j) \leq 2fl\,(b_j \cdot b_j)$, we must have for any $1 \leq j < k$,

$$(1 - 1.01n\mathbf{u}){a_j}^2 \leq 2(1 + 1.01n\mathbf{u}){b_j}^2.$$

So we can assume by induction that

$${a_j}^2 \leq 2{b_j}^2 \frac{(1 + 1.01n\mathbf{u})}{(1 - 1.01n\mathbf{u})} \leq 2{b_j}^2(1 + 2.05n\mathbf{u}).$$

Since we assume $n\mathbf{u} \leq 0.01$, we have:

Lemma 4.1 *During the k-th stage of the algorithm we have, for all $1 \leq j < k$,*

$${a_j}^2 \leq 2.05{b_j}^2, \qquad \|a_j\| \leq 1.44\|b_j\|.$$

If $fl\,(a_k \cdot a_k) \leq 2fl\left(b_k^{(1)} \cdot b_k^{(1)}\right)$ is true, then also ${a_k}^2 \leq 2.05{b_k}^2$, or $\|a_k\| \leq 1.44\|b_k\|$. Otherwise, we find that ${a_k}^2 \geq 1.95{b_k}^2$, or $\|a_k\| \geq 1.39\|b_k\|$.

Clarkson gives a complicated analysis of the error made on the b_k computed by a modified Gram-Schmidt reduction. Using usual Gram-Schmidt reductions yields a straightforward bound. But we first need a weak bound on the norm of $b_k^{(j)}$. We introduce another notation that gives the roundoff error when computing $b_k^{(j)}$ from $b_k^{(j+1)}$:

$$b_k^{(j)} = fl\left(b_k^{(j+1)} - fl\left(fl\left(\frac{a_k}{b_j}\right)b_j\right)\right),$$

$$\varepsilon_j = b_k^{(j)} - \left(b_k^{(j+1)} - \left(\frac{a_k}{b_j}\right)b_j\right).$$

Lemma 4.2 *Assume that $(n+2)\mathbf{u} \leq 0.01$, and that $2^n\mathbf{u} \leq 1$. At step j of the reduction of b_k, we have*

$$\|\varepsilon_j\| \leq 5(n+2)\mathbf{u}\|a_k\|,$$

$$\left\|b_k^{(j)}\right\| \leq 1.03(k - j + 2)\|a_k\|.$$

This bound is obtained by standard error analysis and its proof is given in the appendix for completeness.
We now bound the error $e_k^{(j)}$, and finally $e_k = b_k - c_k$. Let us assume inductively that $\|e_j\| \leq \delta_j \|b_j\|$, for some values of δ_j defined below. Then by the definitions

$$\begin{aligned} b_k^{(j)} &= b_k^{(j+1)} - \left(\frac{a_k}{b_j}\right) b_j + \varepsilon_j, \\ c_k^{(j)} &= c_k^{(j+1)} - \left(\frac{a_k}{c_j}\right) c_j, \end{aligned}$$

and hence we have

$$e_k^{(j)} = e_k^{(j+1)} - \left(\frac{a_k}{b_j} - \frac{a_k}{c_j}\right) c_j + \left(\frac{a_k}{b_j}\right) e_j + \varepsilon_j.$$

We can thus bound

$$\begin{aligned} \left\| e_k^{(j)} \right\| &\leq \left\| e_k^{(j+1)} \right\| + \left| \frac{a_k}{b_j} - \frac{a_k}{c_j} \right| \|c_j\| + \left| \frac{a_k}{b_j} \right| \|e_j\| + \|\varepsilon_j\| \\ &\leq \left\| e_k^{(j+1)} \right\| + \frac{\|a_k\|}{\|b_j\|\|c_j\|} \|e_j\| \|c_j\| + \|a_k\| \frac{\|e_j\|}{\|b_j\|} + \|\varepsilon_j\| \\ &\leq \left\| e_k^{(j+1)} \right\| + 2\delta_j \|a_k\| + 5(n+2)\mathbf{u}\|a_k\|. \end{aligned}$$

By unrolling the recurrence, since $e_k^{(k)} = b_k^{(k)} - c_k^{(k)} = a_k - a_k = 0$, we get

$$\|e_k\| = \left\| e_k^{(1)} \right\| \leq \left[2 \sum_{j=1}^{k-1} \delta_j + 5k(n+2)\mathbf{u} \right] \|a_k\|.$$

This motivates the definition of δ_k by the recurrence:

$$\begin{aligned} \delta_1 &= 0, \\ \delta_k &= 1.44 \left[2 \sum_{j=1}^{k-1} \delta_j + 5k(n+2)\mathbf{u} \right]. \end{aligned}$$

Finally, when the k-th loop is exited, we have $\|a_k\| \leq 1.44\|b_k\|$, which proves that $\|e_k\| \leq \delta_k \|a_k\|$. The induction is thus complete.
We recap this analysis in the following lemma.

Lemma 4.3 *Assume inductively that $\|e_j\| \leq \delta_j \|b_j\|$, for $j < k$. After the reduction of a_k by the $b_1, \ldots, b_{k-1}$, we obtain a vector b_k which is $c_k + e_k$, where $1.44\|e_k\| \leq \delta_k \|a_k\|$. When the k-th stage is over, and before the $k+1$-st stage begins we have $\|e_k\| \leq \delta_k \|b_k\|$.*

Remark. We can also bound the norm of the intermediate vectors $b_k^{(j)}$. We have $\left\|c_k^{(j)}\right\| \leq \|a_k\|$, since $c_k^{(j)}$ is obtained by reducing a_k along different orthogonal coordinates. Using the error bound on $e_k^{(j)}$ and the assumption $\delta_n \leq 0.01$, we can also show that $\left\|b_k^{(j)}\right\| \leq 1.01\|a_k\|$. (Compare with lemma 4.2.)

Because the growth of δ_k is only singly exponential in k and $\mathbf{u}$ is very small, we can assume that $\delta_n \leq 0.01$. In section 7, we will explore the implications on n and $\mathbf{u}$ of this assumption. The bound on $\|e_k\|$ is expressed as a fraction δ_k of $\|b_k\|$. In fact, for the rest of the analysis, c_k will be our reference rather than b_k so we need:

Corollary 4.4 *Assume $\delta_n \leq 0.01$. In the k-th stage, we have the following relations between b_j and c_j for all $j = 1, \ldots, k-1$:*

$$0.99\|b_j\| \leq \|c_j\| \leq 1.01\|b_j\|, \qquad 0.99\|c_j\| \leq \|b_j\| \leq 1.02\|c_j\|,$$

$$0.98{b_j}^2 \leq {c_j}^2 \leq 1.03{b_j}^2, \qquad 0.98{c_j}^2 \leq {b_j}^2 \leq 1.03{c_j}^2.$$

Proof. We compute $(1-\delta_j) \geq 0.99$, $(1-\delta_j)^2 \geq 0.98$, $(1+\delta_j) \leq 1.01$, $(1+\delta_j)^2 \leq 1.03$, $(1+\delta_j)^{-1} \geq 0.99$, $(1+\delta_j)^{-2} \geq 0.98$, $(1-\delta_j)^{-1} \leq 1.02$, $(1-\delta_j)^{-2} \leq 1.03$. □

5 Bounding the growth of the columns of A

We are now interested in what happens when the component of b_k orthogonal to $\mathrm{LC}(a_1, \ldots, a_{k-1})$ is too small, as is indicated by the test $fl\,(a_k \cdot a_k) \geq 2fl\left(b_k^{(1)} \cdot b_k^{(1)}\right)$. To make the notation precise, we let a_k denote the k-th column of the original matrix a_k. The new value of a_k that has been reduced many times in the k-th stage is denoted by a'_k. The only sloppy part in the notation is that a'_k is not always the same vector according to how many reductions of a_k have been performed in the k-th stage, but since we only analyze one reduction step at a time, it will not lead to confusions. The initial value of a'_k when entering the k-th stage is naturally a_k, and at the end of a reduction step for a_k, it is contained in the vector $a_k^{(1)}$ which is the value of a'_k

for the next reduction loop. The vector c_k always refers to the vector c_k obtained by Gram-Schmidt orthonormalization of the current a'_k with respect to $c_1, \ldots, c_{k-1}$.
We must now compute an integer s and reduce sa'_k. We first compute a weak bound on $\left\|a_k^{(j)}\right\|$ that depends on s and $\|a'_k\|$. As in the previous paragraph, we will use this bound in an error analysis to bound $\|a'_k\|$ throughout successive reduction loops in the k-th stage.

5.1 Bounding the growth of $\|a_k\|$.

We now give a weak bound on $\left\|a_k^{(j)}\right\|$ that depends on s and $\|a'_k\|$ as well as on the norms of the c_j's. For this, we first establish a lemma that bounds the growth of the vectors in the modified Gram-Schmidt reduction.

Lemma 5.1 *Assume that* $(n+1)\mathbf{u} \leq 0.01$ *and that* $\delta_n \leq 0.01$. *Then, for any vector* a, *we have*

$$\left\|a - \left(\frac{a}{b_j}\right) a_j\right\| \leq 2.46\|a\|.$$

The proof only uses the fact that b_j is close to c_j, the projection of a_j parallel to $c_1, \ldots, c_{j-1}$, and that $\|a_j\|^2 \leq 2.05\|b_j\|^2$ because the j-th stage has been exited. The proof is then routine and is given in the appendix for completeness.
Using this lemma and standard error analysis, it is not hard to show a weak bound on $\left\|a_k^{(j)}\right\|$, just as we did for $\left\|b_k^{(j)}\right\|$.

Lemma 5.2 *Assume that* $(n+1)\mathbf{u} \leq 0.01$ *and that* $\delta_n \leq 0.01$. *After step* j *(*$j = k-1, \ldots, 1$*) of the reduction of* $a_k^{(k)}$, *we have*

$$\left\|a_k^{(j)}\right\| \leq 2.12^{k-j}\left(0.4\sqrt{S_k^c} + \left\|a_k^{(k)}\right\|\right).$$

Again, the proof is routine and is given in the appendix for completeness.
The bound given by this lemma does not really show that $a_k^{(k)}$ is being reduced. We now explain how to bootstrap this bound to obtain a tighter bound on the final $a_k^{(1)}$. But first we introduce a quantity δ'_k whose value will be motivated by the results below:

$$\delta'_k = \sum_{j=2}^{k} 2.12^{2(k-j)}\delta_j^2.$$

We now assume that $\delta'_n \leq 0.01$. This assumption implies a relation between n and $\mathbf{u}$, and its consequences will be explored in section 7.
Since $c_1, \ldots, c_k$ is an orthogonal basis, we can decompose

$$a_k^{(1)} = \sum_{j=1}^{k} \left(\frac{a_k^{(1)}}{c_j} \right) c_j.$$

But since $a_k^{(1)} = a_k^{(j)} - \mathrm{LC}(a_1, \ldots, a_{j-1}) = a_k^{(j)} - \mathrm{LC}(c_1, \ldots, c_{j-1})$, and the reductions are carried out using exact integer arithmetic (even though the coefficients are rounded), we have $\left(\frac{a_k^{(1)}}{c_j} \right) = \left(\frac{a_k^{(j)}}{c_j} \right)$. Moreover, $\left(\frac{a_k^{(1)}}{c_k} \right) = s$, and thus

$$\left\| a_k^{(1)} \right\|^2 = s^2 \|c_k\|^2 + \sum_{j=1}^{k-1} \left| \left(\frac{a_k^{(j)}}{c_j} \right) \right|^2 \|c_j\|^2.$$

We must now remember that $fl\,(a'_k \cdot a'_k) \leq 2fl\left(b_k^{(1)} \cdot b_k^{(1)} \right)$ is false, and so we find that ${a'_k}^2 \geq 1.91{c_k}^2$, or also $\|a'_k\| \geq 1.38\|c_k\|$. This implies that $\|c_k\|^2 \leq 0.53\|a'_k\|^2$ and bounds the first term. To bound the other terms, we use $\left(\frac{a_j}{c_j} \right) = 1$ to show the following bound

$$\left| \left(\frac{a_k^{(j)}}{c_j} \right) \right|^2 \|c_j\|^2 \leq 0.5\|c_j\|^2 + \delta_{j+1}^2 \left\| a_k^{(j+1)} \right\|^2. \tag{1}$$

We now use the weak bound on $a_k^{(j+1)}$ from lemma 5.2:

$$\begin{aligned}
\delta_{j+1}^2 \left\| a_k^{(j+1)} \right\|^2 &\leq 2\delta_{j+1}^2 \times 2.12^{2(k-j-1)} \left(0.4^2 S_k^c + \left\| a_k^{(k)} \right\|^2 \right), \\
\sum_{j=1}^{k-1} \delta_{j+1}^2 \left\| a_k^{(j+1)} \right\|^2 &\leq \delta'_k \left(0.32 S_k^c + 2 \left\| a_k^{(k)} \right\|^2 \right), \\
\sum_{j=1}^{k-1} \left| \left(\frac{a_k^{(j)}}{c_j} \right) \right|^2 \|c_j\|^2 &\leq (0.5 + 0.34\delta'_k) S_k^c + 2\delta'_k \left\| a_k^{(k)} \right\|^2
\end{aligned}$$

We have used the classical inequality $(x+y)^2 \leq 2(x^2+y^2)$ to obtain the first inequality, and the second follows from our choice of δ'_k.

We started the Gram-Schmidt reduction with a vector $a_k^{(k)} = sa'_k$. Summing over all the components of $\left\|a_k^{(1)}\right\|^2$, and assuming that $\delta_k \leq 0.01$, we obtain

$$\left\|a_k^{(1)}\right\|^2 \leq 0.51 S_k^c + 0.55 s^2 \|a'_k\|^2.$$

We summarize this analysis in the following lemma.

Lemma 5.3 *Assume that $\delta'_n \leq 0.01$. Then after the reduction of $a_k^{(k)} = sa'_k$, we have*

$$\left\|a_k^{(1)}\right\|^2 \leq 0.51 S_k^c + 0.55 s^2 \|a'_k\|^2.$$

Remember that S_k^c depends on $c_1, \ldots, c_{k-1}$, and not on a_k nor on a'_k. Its value is thus fixed during the k-th stage of the algorithm.

5.2 The choice of s

It is now clear that, to control the growth of a_k in the reduction loop, we must choose s carefully in terms of S_k^b and $\|a'_k\|$ before reducing the vector $a_k^{(k)}$. The value S_k^c is unknown to the algorithm, however, and only an approximation S_k^b of S_k^c is known. We must therefore obtain error bounds before we can proceed further. The proof is again routine and is given in the appendix for completeness.

Lemma 5.4 *Assume that $(n+1)\mathbf{u} \leq 0.01$ and that $\delta_n \leq 0.01$. The relative error between S_k^b and S_k^c is*

$$\left|S_k^b - S_k^c\right| \leq 0.07 S_k^c$$

In order to express the bound of lemma 5.3 in terms of S_k^c or $\|a'_k\|$ alone, we must choose s as a multiple of $\sqrt{S_k^c}/\|a'_k\|$. Remember that s has to be an integer, however. Moreover, if $s = 1$, then we must ensure that a'_k is indeed shrinking, otherwise the algorithm could loop infinitely. We leave open for now the choice of two parameters λ and μ, and we set:

$$s' := \left\lceil \sqrt{fl\left(1 + \frac{S_k^b}{\lambda \left\|a'_k\right\|^2}\right)} \right\rceil$$
$$\textbf{if } s' = 1 \textbf{ and } S_k^b \geq fl\left(\mu \|a'_k\|^2\right) \textbf{ then } s := 2 \textbf{ else } s := s'$$

This value allows to bound the final value of $\left\|a_k^{(1)}\right\|$ so that it does not depend on how many times the loop is executed. Initially, $a'_k = a_k$ so we must only examine what

happens to $a_k^{(1)}$ after reducing sa'_k with the value of s chosen above. We examine separately the cases when $s = 1$, or $s' = 1$ and $s = 2$, or $s = s' \geq 2$. After a number of reduction loops during the k-th stage, assume the algorithm reduces a'_k once more. We bound the norm of the vector $a_k^{(1)}$ obtained after the reduction.

If $s = 1$, then $s' = 1$ and the test $S_k^b \geq fl\left(\mu\|a'_k\|^2\right)$ failed. Assuming that the right-hand side can be evaluated within 0.01, we can say that $S_k^b \leq 1.01\mu\|a'_k\|^2$ and so $S_k^c \leq 1.01 \times 1.09\mu\|a'_k\|^2$. Finally, we have

$$\left\|a_k^{(1)}\right\|^2 \leq (0.51 \times 1.09\mu + 0.55)\|a'_k\|^2 \leq 0.55(1+\mu)\|a'_k\|^2.$$

If $s' = 1$ **but** $s = 2$, then the test $S_k^b \geq fl\left(\mu\|a'_k\|^2\right)$ succeeded and we have $S_k^b \geq 0.99\mu\|a'_k\|^2$. It follows that

$$\left\|a_k^{(1)}\right\|^2 \leq \left(0.51 + 2^2 \times 0.55\frac{1.07}{0.99\mu}\right) S_k^c \leq \left(0.51 + \frac{2.38}{\mu}\right) S_k^c.$$

If $s' \geq 2$ then $s = s'$. Allowing for floating point approximations, if $\delta_n \leq 0.01$, we can safely say that

$$1 + \frac{S_k^b}{\lambda\|a'_k\|^2} \geq (s - 0.51)^2$$

and so we obtain that

$$s^2\|a'_k\|^2 \leq \frac{s^2}{(s-0.51)^2 - 1}\frac{1.07}{\lambda}S_k^c.$$

Since $s \geq 2$, we can estimate $\frac{s^2}{(s-0.51)^2-1} \leq 3.28$, and

$$\left\|a_k^{(1)}\right\|^2 \leq \left(0.51 + 0.57\frac{3.28 \times 1.07}{\lambda}\right) S_k^c \leq \left(0.51 + \frac{2.01}{\lambda}\right) S_k^c.$$

We summarize these considerations in the following lemma.

Lemma 5.5 *Let a_k be the value of a_k in the original matrix A, and a'_k be its value after* any *number of reduction loops during the k-th stage of the algorithm. With the above choice for s, we have if* $0.57(1+\mu) \leq 1$,

$$\|a'_k\|^2 \leq \max\left(\|a_k\|^2, \nu S_k^c\right), \qquad \nu = 0.51 + \max\left(\frac{2.38}{\mu}, \frac{2.01}{\lambda}\right).$$

Furthermore, if $s = 1$, then $\left\|a_k^{(1)}\right\|^2 \leq 0.55(1+\mu)\|a'_k\|^2$.

It now becomes clear how to choose μ and λ. The greater λ and μ, the more extra bits are needed by the algorithm. The smaller s, however, the slower the algorithm. A good implementation should take the smallest λ and μ allowed by the number of extra bits available for the specific entries. It should be clear that the best choice is to take $\lambda = \frac{2.011}{2.38}\mu \approx 0.844\mu$, since it allows the smallest ν for the greatest s.
In any case, we must choose $\mu \leq 0.82$. If we demand that the norm of $a_k^{(1)}$ decrease by 0.9 for each iteration such that $s = 1$, then we may take $\mu = 0.472$, leading to $\lambda = 0.399$ and $\nu = 5.543$. This will be our choice of parameters thereon. The reader should keep in mind that it is possible to trade off the number of extra bits required with the speed of convergence of the algorithm.

5.3 Controlling the growth of S_k^c

After the reduction of a_k, we know that $\|c_k\|^2 \leq \|a'_k\|^2 \leq \max\left(\|a_k\|^2, \nu S_k^c\right)$. Let M be the maximum norm of any column a_j of A. Then $S_2^c = \|c_1\|^2 = \|a_1\|^2 \leq M^2$, and so we can prove inductively that for all $k = 1, \ldots, n$, $S_k^c \leq (1+\nu)^{k-2}M^2$. Indeed, we have

$$S_{k+1}^c = \|c_k\|^2 + S_k^c \leq \max\left(M^2, \nu \times (1+\nu)^{k-2}M^2\right) + (1+\nu)^{k-2}M^2 = (1+\nu)^{k-1}M^2.$$

With our choice of $\nu = 5.543$, we obtain for $k \geq 2$:

$$S_k^c \leq 6.55^{k-2}M^2.$$

6 Computing the sign of the determinant

Knowing a matrix B that is close to C allows to approximately compute the determinant of C, which is the same as that A. We must quantify how close B is to C in terms of determinants. In particular, we must show that the signs of $\det C$ and $\det B$ are the same. Then we show that a Gauss pivot correctly computes the sign of the determinant of B.
We introduce several auxiliary matrices for the analysis. Starred matrices are normalized, and overlined matrices are their floating point approximations (only for B). In mathematical notation:

$$\bar{C} = C^* = \left(\frac{c_j}{\|c_j\|}\right)_{j=1,\ldots,n}, \; \bar{B} = \left(fl\left(\frac{b_j}{\|b_j\|}\right)\right)_{j=1,\ldots,n}, \; B^* = \left(\frac{b_j}{\|b_j\|}\right)_{j=1,\ldots,n}.$$

There are several possible norms for matrices (all equivalent). For the purposes of this paper, we consider

$$\|B\| = \max_{x \neq 0} \frac{\|Bx\|}{\|x\|}.$$

With this norm, it is possible to show that

$$\left\| {}^t\bar{B} - {}^t\bar{C} \right\| \leq 10n\mathbf{u} + \sqrt{\delta'_n}.$$

Indeed, lemma 4.3 and corollary 4.4 can be used to show that $\bar{B}$ differs from B^* by at most $10n\mathbf{u}$ for the matrix norm. Also, the norms of a matrix and of its transpose are the same, and for any vector x, we have [4]

$$\left\| {}^t\bar{B}x - {}^t\bar{C}x \right\|^2 \leq \sum_{j=1}^{n} \delta_j^2 \|x\|^2.$$

Let τ_i be the singular values of $\bar{C}$, with $\tau_i = \pm 1$. The singular values σ_i of $\bar{B}$ verify

$$|\sigma_i - \tau_i| \leq 10n\mathbf{u} + \sqrt{\delta'_n} = \varepsilon_n.$$

It is straightforward to show that $\det \bar{B}$ and $\det \bar{C} = \pm 1$ differ by at most $(1+\varepsilon_n)^n - 1$. It is also well known [7] that a Gaussian elimination with partial pivoting produces a matrix LU, where L is lower triangular and U is upper triangular, such that

$$\left\| LU - \bar{B} \right\| = \|\Delta\| \leq 2n^{2.5} 2^{n-1} \mathbf{u}.$$

The computed determinant of LU, which is the floating point product of the diagonal elements of L and U, is an approximation of $\det(LU)$ within $2n\mathbf{u}$, and likewise it can be shown [7] that this approximates $\det \bar{B}$ within $2n\mathbf{u} + (1 + \|\Delta\|)^n - 1$.
All in all, the computed determinant of LU approximates the determinant of C within $2n\mathbf{u} + (1 + \|\Delta\|)^n - 1 + (1 + \varepsilon_n)^n - 1$, which is smaller than 1 with the usual assumptions. Therefore the computed sign of $\det A$ is correct.

7 How much accuracy is needed?

All the logarithms in this section are taken in base 2 unless otherwise mentioned.

7.1 Assumptions on n and u.

The restrictions on n and $\mathbf{u}$ come from the assumptions that $(n+2)\mathbf{u} \leq 0.01$, $2^n\mathbf{u} \leq 1$, $\delta_n \leq 0.01$, and $\delta'_n \leq 0.01$. We first derive a simple bound on δ_k.

Lemma 7.1 *We have*

$$\delta_k \leq 12k(n+2)4^k\mathbf{u},$$

$$\delta'_k \leq 5.25k^2(n+2)^2 8.48^{2k}\mathbf{u}.$$

The proof is routine and given in the appendix for completeness. Its consequences are a lower bound on the floating point precision needed to perform the computations correctly.

Corollary 7.2 *Assume that $n \geq 2$. Then the assumptions that $(n+2)\mathbf{u} \leq 0.01$, $2^n\mathbf{u} \leq 1$, $\delta_n \leq 0.01$, and $\delta'_n \leq 0.01$ are all implied by*

$$\log \frac{1}{\mathbf{u}} \geq 6.17n + 4\log(n+2) + 9.1\,.$$

For double precision in the IEEE 754 standard, we have $\mathbf{u} = 2^{-53}$ and exact computation of the $\delta'_i s$ show that all the conditions above are satisfied for $n \leq 31$, and this also includes the computation of the Gaussian elimination.

7.2 Assumptions on n and b.

Now we must check that we have no problems in computing the auxiliary vectors $a_k^{(j)}$ in the reductions of a_k, we must show that all the vectors $a_k^{(j)}$ have components smaller than N, the greatest integer representable. Typically, if we use floating point arithmetic to represent integers, we have $N = 2/\mathbf{u} - 1$.

Lemma 7.3 *Let N be the greatest integer representable in exact integer arithmetic, and suppose that the coefficients in the original matrix A are given with b bits. All the vectors $a_k^{(j)}$ that occur in the reduction of a'_k during any stage k can be computed if*

$$\log N \geq b + 2.45n + 0.5\log n - 0.66\,.$$

Equivalently, if M if the maximum norm of the columns of the original matrix A, then the algorithm reduces A correctly if one of the two following conditions is satisfied:

$$\log N \geq \log M + 2.45n - 0.66.$$

Proof. Let a_k be the k-th column of A, prior to the k-th stage of the algorithm. Let a'_k be this vector after many reductions. We can obtain a bound on the norms of any vector $a_k^{(j)}$ appearing in the reduction of sa'_k as follows. In the proof of lemma 5.5, we have seen that if $s' \geq 2$, then $s^2\|a'_k\|^2 \leq 7.46S_k^c$. Otherwise, if $s = 2$ and $s' = 1$, then we have seen that $0.99\mu\|a'_k\|^2 \leq S_k^b$, hence $\left\|a_k^{(k)}\right\|^2 \leq \frac{4\times 1.07}{0.99\mu} S_k^c \leq 9.16S_k^c$. If $s = 1$, then $\left\|a_k^{(k)}\right\|^2 = \|a'_k\|^2$. In any case:

$$\left\|a_k^{(k)}\right\|^2 = s^2\|a'_k\|^2 \leq \max\left(\|a_k\|^2, 9.16S_k^c\right).$$

From this and lemma 5.2, we obtain

$$\begin{aligned}
\left\|a_k^{(j)}\right\| &\leq 2.12^{k-j}\left(0.33\sqrt{S_k^c} + \left\|a_k^{(k)}\right\|\right) \\
&\leq 2.12^{k-j}\max\left(\|a_k\| + 0.33\sqrt{S_k^c}, (0.4+\sqrt{7.46})\sqrt{S_k^c}\right) \\
&\leq 2.12^{k-j}\max\left(\|a_k\| + 0.33\sqrt{S_k^c}, 3.43\sqrt{S_k^c}\right) \\
&\leq 2.12^{k-j}\max\left(M + 0.33\sqrt{6.55}^{k-2}M, 3.43\sqrt{6.55}^{k-2}M\right) \\
&\leq 3.43\times 2.12^{k-2}\sqrt{6.55}^{k-2}M \leq 3.43\times 5.43^{k-1}M
\end{aligned}$$

It thus suffices that this last expression be less than N, which is indeed implied by $\log N \geq \log M + 2.45n - 0.66$. In terms of b, the number of bits of the coefficients of A, we have $M \leq \sqrt{n}2^b$, so that a sufficient condition becomes $\log N \geq b + 2.45n + 0.5\log n - 0.66$. □

Computing with the IEEE 754 standard for double precision. For $\mathbf{u} = 2^{-53}$, an exact computation of δ_k and δ'_k shows that $n = 21$ is the maximum order of the matrix allowed by these conditions. Clarkson obtained $n = 32$ as the maximum size of the matrix. Both conditions seem practical enough.

As for the number of extra bits demanded by the exact integer arithmetic, the number of bits b_n with which the entries can be given is represented in the following table for different values of n. This is the bound given by the above analysis. (In fact, experimental evidence suggest that $b_n = 50$ up to $n = 6$. See also section 9.)

$n \times n$ matrix	2	3	4	5	6	7	8	9	10	11	12	13	14
b_n bits	48	45	42	40	37	35	32	30	27	24	22	19	17

Note that according to his paper, Clarkson can compute 10×10 determinants with 32 bits. Our analysis does not fall too much behind. It is also very pessimistic, so that in practise one should test for overflow in the code itself rather than strictly enforce the number of bits in the entries.

8 Running time analysis

We must now show that the algorithm terminates. In fact, the algorithm given by Clarkson does *not* terminate if the determinant is null, so we must safeguard against such situations by introducing an additional control into the loop. Two approaches are possible:

1. compute the maximum number of iterations for a non-null determinant, and conclude that the determinant is null when this number of iterations is achieved, or
2. dynamically compute an upper bound on the floating point approximation to the determinant, and conclude that the determinant is null when this bound is smaller than 1.

We examine the first question in the first subsection, by bounding the number of iterations when the determinant is not zero. In the second subsection, we take a close look at what happens when the determinant is null. We first introduce a quantity inversely related to the *orthogonality defect* introduced by Clarkson, which we call an orthogonality criterion:

$$\mathcal{OD}_k(a_1, \ldots, a_k) = \frac{\mathrm{Vol}_k(a_1, \ldots, a_k)}{\prod_{j=1}^{k} \|a_j\|} = \frac{\prod_{j=1}^{k} \|c_j\|}{\prod_{j=1}^{k} \|a_j\|}.$$

Here, $\mathrm{Vol}_k(a_1, \ldots, a_k)$ is the k-dimensional volume of the fundamental region of the lattice generated by $a_1, \ldots, a_k$. It is clear that $\mathcal{OD}_k$ is contained between 0 and 1 and is 0 if and only if the first k vectors are linearly dependent. Moreover, if A has integer coefficients, $\mathrm{Vol}_k(a_1, \ldots, a_k)$ is an integer for all k. Thus:

Lemma 8.1 *If A is an integral non-singular matrix, then for all k*

$$\frac{1}{\prod_{j=1}^{k} \|a_j\|} \leq \mathcal{OD}_k(a_1, \ldots, a_k) \leq 1.$$

The total number of iterations

Each iteration either multiplies a vector c_k by $s \geq 2$, or it keeps c_k constant but it decreases the norm of a_k by 0.9 for our choice of $\mu = 0.579$. Therefore, in either case, $\mathcal{OD}_k$ is multiplied by a factor at least 1.1. Since $\mathcal{OD}_k$ is at least $\frac{1}{M^k}$ to start

with (if A is non-singular) and must remain smaller than 1, the number of iterations in the first k stages cannot outgrow $k \log_{1.1} M$, which is $O(k \log M)$.
For the entire algorithm, the number of iterations is bounded by $\log 1/\mathcal{OD}_n(A)$, which depends on how orthogonal A is. Note that this is unbounded if A is singular, and the algorithm presented in [4] does not terminate if the matrix is singular.
In the case of a null determinant, we can stop after enough iterations. In the table below, we give the maximum number $m_k^{(n)}$ of iterations for the first k stages of a $n \times n$ determinant with $b = 53$, b_n as above, and $M = \sqrt{n}2^{b_n}$. Note that a much better way of counting it is to separately count the iterations when $s \geq 2$ or when $s = 1$ since only in the latter ones does $\mathcal{OD}$ increase by a factor of only 1.1. Therefore $m_k^{(n)}$ in the table below only counts the maximum number of iterations for which $s' \geq 2$.

k	2	3	4	5	6	7	8	9	10
$n = 2$	97								
$n = 3$	91	137							
$n = 4$	86	129	172						
$n = 5$	82	123	164	205					
$n = 6$	76	114	153	191	229				
$n = 7$	72	109	145	182	218	254			
$n = 8$	67	100	134	167	201	234	268		
$n = 9$	63	94	126	157	189	221	252	284	
$n = 10$	57	85	114	143	171	200	229	257	286

The case of a null determinant

Just as we introduce the sum of the square norms for bounding the growth of the vectors, we now introduce their product to maintain an upper bound on the determinant. Let

$$P_k^c = \prod_{1 \leq j < k} c_j \cdot c_j, \qquad P_k^b = fl\left(\prod_{1 \leq j < k} (1+\delta_j)^2 b_j \cdot b_j\right), \qquad P_k^a = \prod_{1 \leq j < k} a_j \cdot a_j.$$

Clearly, without the floating point roundoffs, P_k^b is an upper bound on P_k^c. Moreover, $\mathcal{OD}_k$ is the quotient of P_k^c and P_k^a. Thus P_k^b/P_k^a is an good approximation of $\mathcal{OD}_k$. In view of the lemma above, if the original P_k^c is smaller than 1, then this means than A is singular. Note that multiplying a_k by s also multiplies the minimum value of P_k^a by s^2. Therefore we actually test whether P_k^c is smaller than $\prod s^2$, the product of all the (squared) s factors throughout the entire algorithm. We can test this condition

using the approximate value P_k^b. For this, we give a standard error analysis in the appendix for completeness.

Lemma 8.2 *Assume that* $(n+2)\mathbf{u} \leq 0.01$. *Then we have* $P_k^c \leq 1.05 P_k^b$.

In the k-th stage, the factor for $(1+\delta_k)^2 b_k \cdot b_k$ in the above product is not known to be an upper bound on $c_k \cdot c_k$. It is thus replaced by $\|b_k\|^2 + \delta_k^2 \|a_k\|^2$, an upper bound on $\|c_k\|^2$ according to theorem 4.3. Thus in the k-th stage, if we assume that the value of P_k^b is maintained correctly and that $(\prod s)$ stores the product of the successive factors s by which the determinant has been multiplied in all the previous stages, we can insert the following test to detect when the determinant is null:

$$\textbf{if } fl\left(P_k^b \times \left(\|b_k\|^2 + \delta_k^2 \|a_k\|^2\right)\right) \leq fl\left(0.95 \times (\textstyle\prod s)^2\right) \textbf{ then return } 0.$$

Correctness of this test is ensured by the previous lemma and the fact that $P_k^c < (\prod s)^2$ implies that $P_k^c = 0$. This test takes advantage on the fact that the floating point estimation of the determinant is not too bad, as opposed to the method of counting the number of iterations. There is no guarantee, however, that it will succeed after enough iterations. In our implementation, we use both this test (for efficiency) and a counter for the maximum number of iterations (for correctness).

9 Experimental results

The algorithm was implemented in C. The implementation and some measurements are available on the WWW at

`http://www.inria.fr/prisme/personnel/bronnimann/clarkson/english.html`

We compared the output with an exact determinant computation. The sign always agreed when there was no overflow and these overflows were always detected by the algorithm.
To gain some insight on the behavior of the algorithm, we counted the number of iterations for different kinds of matrices. A random number on b bits means an integer drawn uniformly at random in the range $[-2^b + 1, 2^b - 1]$. We tried three kinds of inputs:

random : the coefficients are random on b bits.

null : the columns are of the form $k_i U_i$, except for the last one which is of the form $\sum l_i U_i$. The coefficients of the vectors U_i $(i = 1 \ldots n-1)$ are random over $\lceil b/2 \rceil$ bits, the coefficients k_i and l_i's are random over $\lfloor b/2 \rfloor$ bits.

perturbed : the entries of a null determinant are perturbed by a random number over 2 bits.

To complicate the combinatorics, we have two choices for s, the one in Clarkson's original paper, which we call *Clarkson's choice*, and the one introduced in section 5.2, which we call *our choice* (or *custom* choice).
Some justification is needed here. Random determinants occur in practise, of course, and null determinants provide the worst case for the algorithm, which makes them noteworthy objects of study. Null $n \times n$ determinants considered here are of rank $n-1$, because this is the hardest case over null determinants. The values of perturbed determinants are typically of magnitude $\mathbf{u}2^{bd}$, a fraction $\mathbf{u}$ of the maximum value 2^{bd}. Such determinants are the ones not caught by an arithmetic filter [5], and it has been shown experimentally [6] that determinants like these are usually between $\mathbf{u}2^{bd}$ and $\mathbf{u}^2 2^{bd}$. Therefore they are also desirable objects of study. Determinants smaller than $\mathbf{u}^2 2^{bd}$ very rarely occur [5], except in some applications where the input data are highly dependent.

For each kind of determinant and each choice for s, we look at the *average (minimum, maximum) number of iterations* as a function of the dimension (over 50 bits in dimensions 2 to 5, over 49 bits in dimensions 6 to 9, and over 48 bits in dimension 10 to 15). We also look at the *influence of the number of bits of the entries* on the number of iterations. Finally, we look at *the failure rate* depending on the number of bits (the percentage of determinants for which the algorithm fails).
The algorithm can exit by giving the exact sign, or by failing in the case that some Gram-Schmidt reduction cannot be computed exactly. The above analysis guarantees that the algorithm will not fail if the entries are smaller than 2^{b_n} for the values of b_n given at the end of section 7, but in practice we find that it rarely fails for entries smaller than 2^{50}. We observe a failure rate of zero for 50 bits in dimensions 2 to 5, for 49 bits in dimensions 6 to 9, and for 48 bits in dimension 10 to 15.

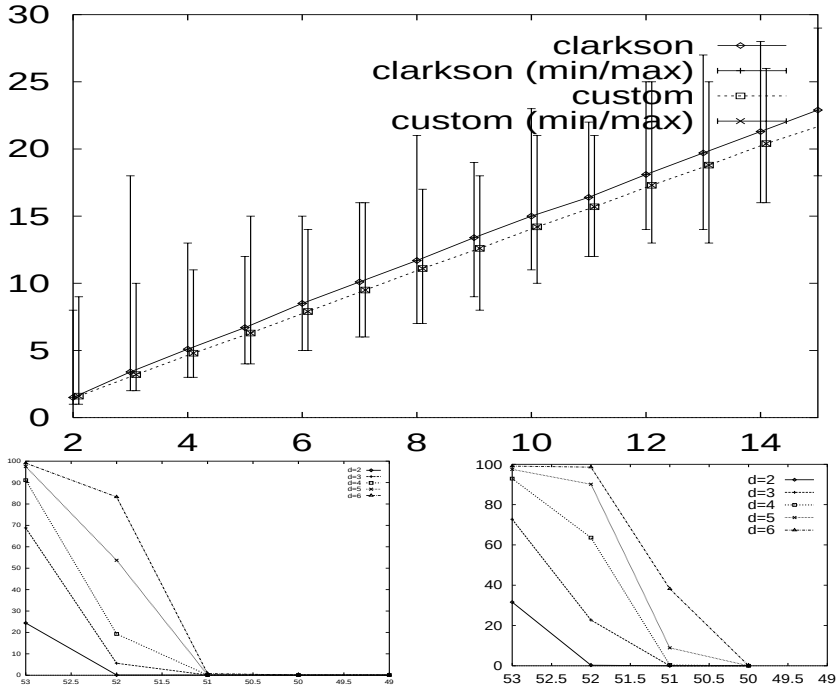


Figure 4: The number of iterations for a random determinant. The bottom row shows the failure rate for Clarkson's choice (left) and our choice (right) as a function of the number of bits.

9.1 Random determinants

The numbers of iterations and failure rates are shown on figure 4. One notices that the average number of iterations grows almost linearly as a function of n, typically as $1.5n$. The choice for s does not make a difference of more than one iteration on the average, although the maximum is typically higher for Clarkson's choice. The average number of iterations does not seem to greatly depend on the number of bits of the input in either case. The choices for s also show similarities in the way that the number of bits of the input influences the number of iterations and failure rate: for both choices, the failure rate is zero over 50 bits up to the dimension 6. Clarkson's choice seems to lead to an algorithm with a faster-decreasing failure rate, though.

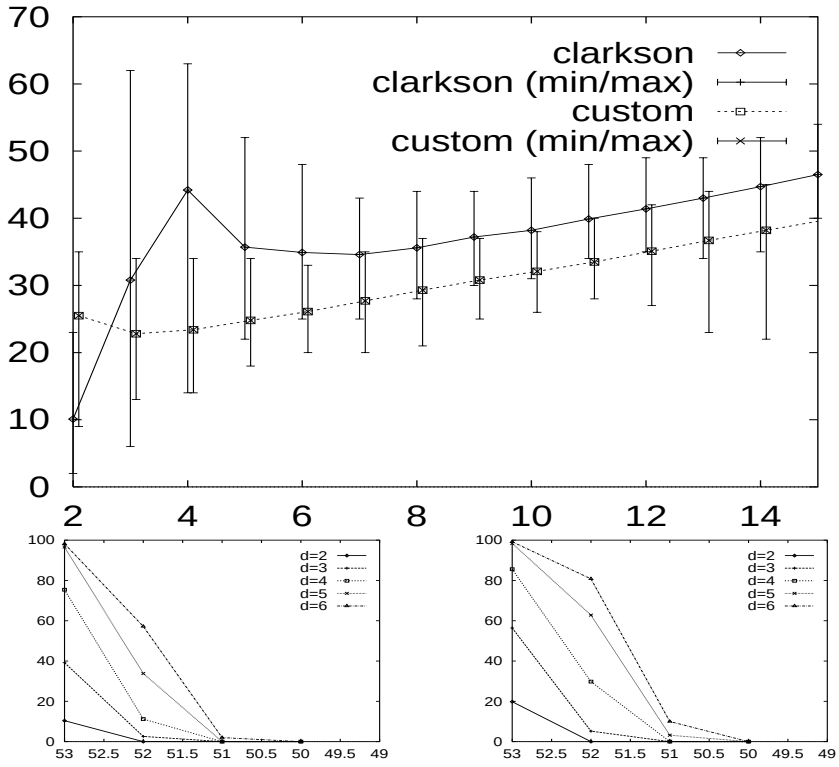


Figure 5: The number of iterations for a perturbed determinant. The bottom row shows the failure rate of Clarkson's choice (left) and our choice (right) as a function of the number of bits.

9.2 Perturbed determinants

The numbers of iterations and failure rates are shown on figure 5. The average number of iterations is much higher, which is normal since the determinant is almost null. Still, it is about $23 + 1.5n$ for Clarkson's choice, and about $19.5 + 1.5n - 0.5b'$ for our choice. Here $b' = 53 - b$ is the number of extra bits available for the integer computations. The difference in the number of iterations is of about 10 in favor of our choice (average and extremes). Again, however, we note that the failure rate decreases with fewer bits faster for Clarkson's choice than for our choice, and even more markedly for higher dimensions. For perturbed determinants, Clarkson's choice leads to a slower but more robust algorithm. Still, the failure rate is zero for both choices and entries over 50 bits up to the dimension 6.

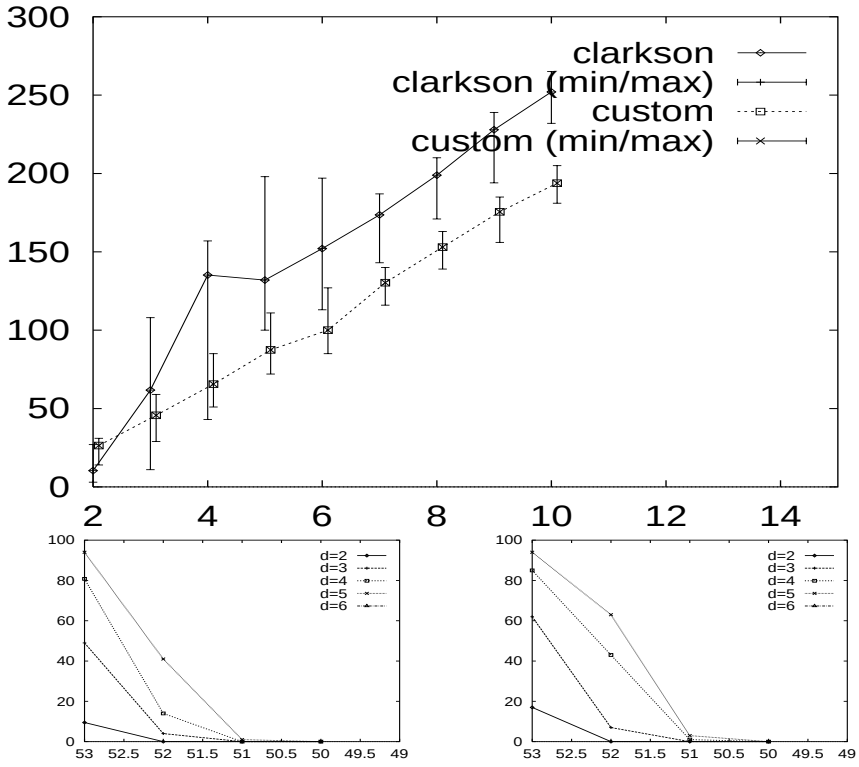


Figure 6: The number of iterations on a null determinant. The bottom row shows the failure rate of Clarkson's choice (left) and our choice (right) as a function of the number of bits.

9.3 Null determinants

The numbers of iterations and failure rates are shown on figure 6. As expected, the number of iterations is much higher in this case, about $25n$ for Clarkson's choice, and $20n - 2b'$ for our choice, where $b' = 53 - b$ is the number of extra bits available for the integer computations. Our choice clearly stands out as a faster method in this case. Again, the failure rate decreases faster for Clarkson's choice, but both rates are zero for both choices and entries over 50 bits up to the dimension 6.

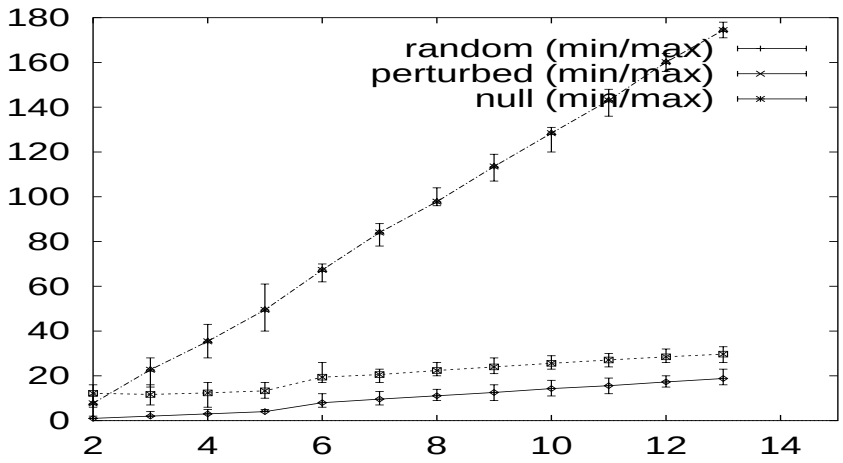


Figure 7: The number of iterations for determinants with small entries (32 bits out of 53 available bits).

9.4 Small entries

Sometimes one wishes to use the code with maximum bit size much smaller than the maximum available. We report on the number of iterations of the method for our choice of s in the algorithm, on matrices generated as above but with entries on 32 bits (and 53 bits available for the integer computations). Clarkson's choice for s is not reported here, as it is always worse. One notes that if no determinant is null, one can expect the algorithm to perform at most 20 iterations.

10 Conclusions

Clarkson [4] gives an algorithm that computes the exact sign of a determinant with integer coefficients of any size. The original analysis contains some small gaps and, from his paper, it is not clear what the practical efficiency of the method is.
We provide a complete analysis of Clarkson's algorithm. We show the method practical and competitive, and also examine its expected behavior depending on the condition of the matrix. In particular, although the analysis only guarantees that the sign is correctly computed for $n \times n$ matrices over roughly $53 - 2.5n$ bits using double precision, we find experimentally that the algorithm computes the sign of the determinant without overflowing for 50 bits in dimensions 2 to 5, for 49 bits in dimensions 6 to 9, and for 48 bits in dimension 10 to 15.

We introduce a different choice of the parameter s in the algorithm, which simplifies the analysis and experimentally seems to lead to a smaller number of iterations. We report on the number of iterations performed by the algorithm on several kinds of determinants (random, almost null, null) and several bit-length for the entries.
The method is limited to integer matrices, and it would be very valuable to extend it to a broader range of exponent for the input. The problem is that no upper bound on the number of iterations is known in this case, and it is also not clear how to perform the integral combinations over the a_k's exactly. If this is needed, however, Shewchuk [13] gives a method that achieves this and is also practical at least in small dimensions.

Acknowledgments

The authors would like to thank Ken Clarkson for several discussions about his paper. Jean-Pierre Merlet is acknowledged for supplying them with his interactive drawing preparation system JPdraw .

References

[1] F. Avnaim, J-D. Boissonnat, O. Devillers, F. Preparata, and M. Yvinec. Evaluating signs of determinants using single-precision arithmetic. *Algorithmica*, 1997. to appear.

[2] C. Burnikel. *Exact Computation of Voronoi Diagrams and Line Segment Intersections*. Ph.D thesis, Universität des Saarlandes, March 1996.

[3] C. Burnikel, J. Könnemann, K. Mehlhorn, S. Näher, S. Schirra, and C. Uhrig. Exact geometric computation in LEDA. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages C18–C19, 1995.

[4] K. L. Clarkson. Safe and effective determinant evaluation. In *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 387–395, 1992.

[5] O. Devillers and F. Preparata. A probabilistic analysis of the power of arithmetic filters. Rapport de recherche 2971, INRIA, 1996. Also Report CS96-27, Brown University.

[6] I. Z. Emiris. A Complete Implementation for Computing General Dimensional Convex Hull. Research Report 2551, INRIA, BP93, 06902 Sophia-Antipolis, France, 1995.

[7] G. Forsythe and C. Moler. *Computer solutions of linear algebraic systems*. Prentice Hall, 1967.

[8] S. Fortune. Numerical stability of algorithms for 2-d Delaunay triangulations and Voronoi diagrams. *Int. J. Comput. Geom. Applic.*, 5(1&2):193–213, 1995.

[9] S. Fortune and C. J. Van Wyk. Efficient exact arithmetic for computational geometry. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 163–172, 1993.

[10] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 32(1):5–48, March 1991.

[11] C. M. Hoffmann. The problems of accuracy and robustness in geometric computation. *IEEE Computer*, 22(3):31–41, March 1989.

[12] V. Milenkovic. Verifiable implementations of geometric algorithms using finite precision arithmetic. *Artif. Intell.*, 37:377–401, 1988.

[13] Jonathan R. Shewchuk. Robust adaptive floating-point geometric predicates. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 141–150, 1996.

[14] K. Sugihara and M. Iri. A robust topology-oriented incremental algorithm for Voronoi diagrams. *Internat. J. Comput. Geom. Appl.*, 4:179–228, 1994.

[15] C. K. Yap. Towards exact geometric computation. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 405–419, 1993. To appear in *Comput. Geom. Theory Appl.*

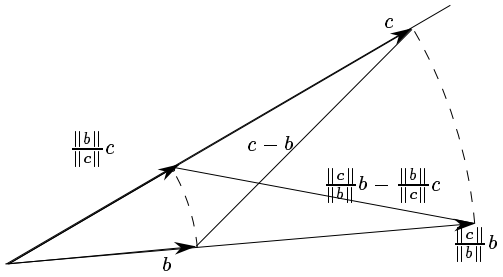


Figure 8: For the proof of (2).

Appendix

Often in the algorithm, we will need to bound the difference of two reductions having the same numerator. We will use the following inequality

$$\left|\frac{a}{b}-\frac{a}{c}\right| \leq \frac{\|a\|}{\|b\|\|c\|}\|b-c\|. \tag{2}$$

Proof. Indeed, we can compute

$$\left|\frac{a}{b}-\frac{a}{c}\right| \quad \leq \quad \|a\|\left\|\frac{b}{\|b\|^2}-\frac{c}{\|c\|^2}\right\| \leq \frac{\|a\|}{\|b\|\|c\|}\left\|\frac{\|c\|}{\|b\|}b-\frac{\|b\|}{\|c\|}c\right\| = \frac{\|a\|}{\|b\|\|c\|}\|b-c\|.$$

The last equality can be proven simply by looking at figure 8. The first vector $\frac{\|c\|}{\|b\|}b-\frac{\|b\|}{\|c\|}c$ is one diagonal, and the second vector $b-c$ is the other diagonal of the figure. Both diagonals have equal length by symmetry. □

Lemma 4.2. *Assume that* $(n+2)\mathbf{u} \leq 0.01$*, and that* $2^n\mathbf{u} \leq 1$*. At step* j *of the reduction of* b_k*, we have*

$$\|\varepsilon_j\| \leq 5(n+2)\mathbf{u}\|a_k\|,$$
$$\left\|b_k^{(j)}\right\| \leq 1.03(k-j+2)\|a_k\|.$$

Proof. We know that, if $j \leq k-1$,

$$\begin{aligned} b_k^{(j)} &= fl\left(b_k^{(j+1)} - fl\left(fl\left(\frac{a_k}{b_j}\right)b_j\right)\right) \\ &= fl\left(b_k^{(j+1)} - fl\left(\left(\frac{a_k}{b_j}+\varepsilon_j^{(1)}\right)b_j\right)\right) \end{aligned}$$

$$\begin{aligned} &= fl\left(b_k^{(j+1)} - \left(\frac{a_k}{b_j}\right)b_j + \varepsilon_j^{(2)}\right) \\ &= b_k^{(j+1)} - \left(\frac{a_k}{b_j}\right)b_j + \varepsilon_j. \end{aligned}$$

We can bound the errors using floating point error bounds and lemmas 3.1 and 3.2. This yields successively:

$$\begin{aligned} \left|\varepsilon_j^{(1)}\right| &\leq \frac{\|a_k\|}{\|b_j\|}1.01(2n+2)\mathbf{u} \\ \left\|\varepsilon_j^{(2)}\right\| &\leq \|a_k\|\mathbf{u} + \left|\varepsilon_j^{(1)}\right|\|b_j\|(1+\mathbf{u}) \leq (3n+4)\mathbf{u}\|a_k\| \\ \|\varepsilon_j\| &\leq \left\|b_k^{(j+1)} - \left(\frac{a_k}{b_j}\right)b_j\right\|\mathbf{u} + \left\|\varepsilon_j^{(2)}\right\|(1+\mathbf{u}) \leq \left\|b_k^{(j+1)}\right\|\mathbf{u} + 3(n+2)\mathbf{u}\|a_k\|. \end{aligned}$$

Bounding trivially $\left\|b_k^{(j)}\right\|$ by its exact value plus the error $\|\varepsilon_j\|$ yields the recurrence

$$\begin{aligned} \left\|b_k^{(j)}\right\| &\leq \left\|b_k^{(j+1)} - \left(\frac{a_k}{b_j}\right)bj\right\| + \|\varepsilon_j\| \\ &\leq (1+\mathbf{u})\left\|b_k^{(j+1)}\right\| + (1+3(n+2)\mathbf{u})\,\|a_k\| \\ &\leq (1+\mathbf{u})\left\|b_k^{(j+1)}\right\| + 1.03\|a_k\| \\ &\leq 1.03\left((1+\mathbf{u})^{k-j} + \cdots + (1+\mathbf{u}) + 1\right)\|a_k\| \\ &\leq 1.03\frac{(1+\mathbf{u})^{k-j+1}-1}{\mathbf{u}}\|a_k\|. \end{aligned}$$

We used the assumption that $(n+2)\mathbf{u} \leq 0.01$ and that $2^n\mathbf{u} \leq 1$. We can bound the term $(1+\mathbf{u})^{k-j+1} - 1$ by $(k-j+1)\mathbf{u} + 2^{(k-j+1)}\mathbf{u}^2$, by grouping all the powers of $\mathbf{u}$ higher than one. Using $2^n\mathbf{u} \leq 1$ shows that

$$\left\|b_k^{(j)}\right\| \leq 1.03(k-j+2)\|a_k\| \leq 2(n+2)\|a_k\|.$$

Plugging the last bound into the bound on $\|\varepsilon_j\|$ yields the other part of the lemma. □

Lemma 5.1. *Assume that* $(n+1)\mathbf{u} \leq 0.01$ *and that* $\delta_n \leq 0.01$. *Then, for any vector* a, *we have*

$$\left\|a - \left(\frac{a}{b_j}\right)a_j\right\| \leq 2.46\|a\|.$$

Proof. Recall from lemma 4.1 that $\|a_j\|^2 \leq 2.05\|b_j\|^2$, $\|a_j\| \leq 1.44\|b_j\|$. But b_j is not any vector, it is obtained by orthogonalization of a_j. Specifically, $a_j \cdot c_j = {c_j}^2 \geq 0.98{b_j}^2$ and

$\|c_j - b_j\| \leq \delta_j \|b_j\|$. With $\delta_j \leq 0.01$, it is easily shown that

$$a_j \cdot b_j \geq a_j \cdot c_j - \|a_j\|\|b_j - c_j\| \geq 0.98{b_j}^2 - 1.44\delta_j {b_j}^2 \geq 0.96{b_j}^2,$$

and hence

$$\|b_j - a_j\|^2 \leq \|b_j\|^2 + \|a_j\|^2 - 2a_j \cdot b_j \leq (1 + 2.05 - 2 \times 0.96)\|b_j\|^2 \leq 1.13\|b_j\|^2.$$

Finally,

$$\begin{aligned} \left\| a - \left(\frac{a}{b_j}\right) a_j \right\| &\leq \left\| a - \left(\frac{a}{b_j}\right) b_j \right\| + \left\| \left(\frac{a}{b_j}\right)(b_j - a_j) \right\| \\ &\leq \|a\| + \frac{\|a\|}{\|b_j\|}\|b_j - a_j\| \\ &\leq (1 + \sqrt{1.13})\|a\| \leq 2.09\|a\|. \end{aligned}$$

□

Lemma 5.2. *Assume that* $(n+1)\mathbf{u} \leq 0.01$ *and that* $\delta_n \leq 0.01$. *After step* j *(*$j = k-1, \ldots, 1$*) of the reduction of* $a_k^{(k)}$*, we have*

$$\left\| a_k^{(j)} \right\| \leq 2.12^{k-j} \left(0.4\sqrt{S_k^c} + \left\| a_k^{(k)} \right\| \right).$$

Proof. First recall that

$$a_k^{(j)} = a_k^{(j+1)} - \left\lceil fl\left(\frac{a_k^{(j+1)}}{b_j}\right) \right\rfloor a_j.$$

Using that $(n+1)\mathbf{u} \leq 0.01$ as usual, we know from lemmas 5.1 and 3.2 that

$$\begin{aligned} \left\| a_k^{(j+1)} - \left(\frac{a_k^{(j+1)}}{b_j}\right) a_j \right\| &\leq 2.09 \left\| a_k^{(j+1)} \right\|, \\ \left| \left(\frac{a_k^{(j+1)}}{b_j}\right) a_j - fl\left(\frac{a_k^{(j+1)}}{b_j}\right) a_j \right| &\leq 1.01(2n+2)\mathbf{u} \frac{\left\| a_k^{(j+1)} \right\|}{\|b_j\|}\|a_j\| \leq 0.03 \left\| a_k^{(j+1)} \right\|, \\ \left| fl\left(\frac{a_k^{(j+1)}}{b_j}\right) - \left\lceil fl\left(\frac{a_k^{(j+1)}}{b_j}\right) \right\rfloor \right| &\leq \frac{1}{2}. \end{aligned}$$

by the definition of $\lceil \cdot \rfloor$. Recall from lemma 4.1 and corollary 4.4 that $\|a_j\| \leq 1.44\|b_j\| \leq 1.47\|c_j\|$, so that we have $\frac{1}{2}\|a_j\| \leq 0.74\|c_j\|$. Summing up, we finally obtain

$$\left\| a_k^{(j)} \right\| \leq 2.12 \left\| a_k^{(j+1)} \right\| + 0.74\|c_j\|.$$

Unrolling the recurrence eventually proves that

$$\left\|a_k^{(j)}\right\| \leq 0.74 \sum_{i=j}^{k-1} 2.12^{i-j}\|c_j\| + 2.12^{k-j}\left\|a_k^{(k)}\right\|.$$

We can now use the Cauchy-Schwartz inequality which yields

$$\begin{aligned}\left(\sum_{i=j}^{k-1} 2.12^{i-j}\|c_j\|\right)^2 &\leq \left(\sum_{i=j}^{k-1} 2.12^{2(i-j)}\right)\left(\sum_{i=j}^{k-1}\|c_j\|^2\right) \\ &\leq \frac{2.12^{2(k-j)}}{2.12^2-1} S_k^c.\end{aligned}$$

The lemma follows from the fact that $0.74(2.12^2-1)^{-\frac{1}{2}} \leq 0.4$. □

Proof of (1). Remembering that $\left(\frac{a_j}{c_j}\right) = 1$, we have by the definition of $a_k^{(j)}$

$$\begin{aligned}\left|\left(\frac{a_k^{(j)}}{c_j}\right)\right|\|c_j\| &\leq \left|\left(\frac{a_k^{(j+1)}}{c_j}\right) - \left\lceil fl\left(\frac{a_k^{(j+1)}}{b_j}\right)\right\rfloor\right|\|c_j\|, \\ &\leq \left|\frac{a_k^{(j+1)}}{c_j} - \frac{a_k^{(j+1)}}{b_j}\right|\|c_j\| + \left|\frac{a_k^{(j+1)}}{b_j} - fl\left(\frac{a_k^{(j+1)}}{b_j}\right)\right|\|c_j\| + \frac{1}{2}\|c_j\| \\ &\leq \frac{\left\|a_k^{(j+1)}\right\|}{\|b_j\|}\|e_j\| + \frac{\|c_j\|}{\|b_j\|}\left\|a_k^{(j+1)}\right\|1.01(2n+2)\mathbf{u} + \frac{1}{2}\|c_j\| \\ &\leq 0.5\|c_j\| + (\delta_j + 1.04(2n+2)\mathbf{u})\left\|a_k^{(j+1)}\right\| \\ &\leq 0.5\|c_j\| + \delta_{j+1}\left\|a_k^{(j+1)}\right\|.\end{aligned}$$

In the last inequalities, we have used the facts that $\delta_j \leq \delta_{j+1}$ and that $\delta_j + 1.04(2n+2)\mathbf{u} \leq \frac{1}{1.44}\delta_{j+1} \leq 0.7\delta_{j+1}$ as is immediate from the definition of the δ_j's. We can use the classical inequality $(x+y)^2 \leq 2(x^2+y^2)$ to get

$$\left|\left(\frac{a_k^{(j)}}{c_j}\right)\right|^2 \|c_j\|^2 \leq 0.5\|c_j\|^2 + \delta_{j+1}^2\left\|a_k^{(j+1)}\right\|^2.$$

□

Lemma 5.4. *Assume that* $(n+1)\mathbf{u} \leq 0.01$ *and that* $\delta_n \leq 0.01$. *The relative error between* S_k^b *and* S_k^c *is*

$$\left|S_k^b - S_k^c\right| \leq 0.07 S_k^c$$

Proof. Summing corollary 4.4 over j, we get

$$0.98 \sum_{j=1}^{k-1} \|b_j\|^2 \leq S_k^c = \sum_{j=1}^{k-1} \|c_j\|^2 \leq 1.03 \sum_{j=1}^{k-1} \|b_j\|^2.$$

But we must now compute the floating point accuracy of S_k^b with respect to the sum above. Since the dot products create an error at most $1.01(2n+2)\mathbf{u}\|b_j\|^2$, and each of the $k-2$ additions creates an error of at most $\mathbf{u}\sum_{j=1}^{k-1}\|b_j\|^2$, we get

$$\left| S_k^b - \sum_{j=1}^{k-1} \|b_j\|^2 \right| \leq (2.02n + k + 0.02)\mathbf{u} \sum_{j=1}^{k-1} \|b_j\|^2.$$

Finally, since $k \leq n$, we can bound $2.02n + k + 0.02$ by $3.1(n+1)$, and the assumption $(n+1)\mathbf{u} \leq 0.01$ shows that the quantity above is smaller than $0.031 \sum_{j=1}^{k-1}\|b_j\|^2$. Hence,

$$0.93 S_k^c \leq 0.98(1-0.031) S_k^c \leq S_k^b \leq 1.03(1+0.031) S_k^c \leq 1.07 S_k^c.$$

□

Lemma 7.1. *We have*

$$\delta_k \leq 12k(n+2)4^k\mathbf{u},$$
$$\delta_k' \leq 5.25k^2(n+2)^2 8.48^{2k}\mathbf{u}.$$

Proof. Indeed, $\delta_1 = 0$, and using the inductive definition of δ_k, we find that for $k \geq 2$,

$$\begin{aligned}
\delta_k &\leq 2.88 \sum_{j=2}^{k-1} \delta_j + 7.2k(n+2)\mathbf{u} \\
&\leq 2.88 \times 12k(n+2)\frac{4^k}{4-1}\mathbf{u} + 7.2k(n+2)\mathbf{u} \\
&\leq 12k(n+2)4^k\mathbf{u}.
\end{aligned}$$

As for δ_k', we can use the bound just derived to show that

$$\delta_k' \leq \sum_{j=2}^{k} 2.12^{2(k-j)}\delta_j^2\mathbf{u}$$

$$
\begin{aligned}
&\leq 144k^2(n+2)^2 \sum_{j=2}^{k} 4^{2j} 2.12^{2(k-j)}\mathbf{u} \\
&\leq 144k^2(n+2)^2 \sqrt{\sum_{j=0}^{k-2} 4^{4(j+2)}} \sqrt{\sum_{j=0}^{k-2} 2.12^{4j}}\mathbf{u} \\
&\leq \frac{144 \times 4^4}{\sqrt{2.49^4-1}\sqrt{4^4-1}} k^2(n+2)^2(4 \times 2.12)^{2(k-1)}\mathbf{u} \leq 5.25k^2(n+2)^2 8.48^{2k}\mathbf{u}.
\end{aligned}
$$

□

Lemma 8.2. *Assume that* $(n+2)\mathbf{u} \leq 0.01$. *Then we have* $P_k^c \leq 1.05 P_k^b$.

Proof. First we notice that during the k-th loop, we have $\|c_j\| \leq (1+\delta_j)\|b_j\|$ for all $j < k$, so that it immediately follows that

$$P_k^c \leq \prod_{1 \leq j < k} (1+\delta_j)^2 b_j \cdot b_j.$$

Computing the quantities $(1+\delta_j) b_j \cdot b_j$ yields a P_k^b which approximates the product above. The error in computing the j-th factor is an additional $1.01(2n+4)\mathbf{u}(1+\delta_j)^2\|b_j\|^2$, and computing the entire chain of $k-2$ products results in an additional error term of at most $(k-2)\mathbf{u}\prod_{1\leq j<k}(1+\delta_j)^2\|b_j\|^2$. It follows that

$$
\begin{aligned}
P_k^b &\geq \left(\prod_{1 \leq j < k} (1+\delta_j)^2 b_j \cdot b_j \right) (1 - 1.01k(2n+4)\mathbf{u})\,(1-(k-2)\mathbf{u}) \\
&\geq 0.96 \left(\prod_{1 \leq j < k} (1+\delta_j)^2 b_j \cdot b_j \right),
\end{aligned}
$$

which proves the lemma, since $1/0.96 \leq 1.05$. □

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399