2010 International Symposium on Nonlinear Theory and its Applications
NOLTA2010, Krakow, Poland, September 5-8, 2010

Condition Numbers of Two-Dimensional Orientation Problem

Katsuhisa Ozaki[†,**], Takeshi Ogita[‡,**] and Shin'ichi Oishi[*,**]

†Department of Mathematical Sciences, College of Systems Engineering and Science, Shibaura Institute of Technology
307 Fukasaku, Minumaku, Saitama-shi, Saitama 337-8570, Japan
‡Department of Mathematical Sciences, Tokyo Woman's Christian University
2-6-1 Zempukuji, Suginami-ku, Tokyo 167-8585, Japan
* Faculty of Science and Engineering, Waseda University
3-4-1 Okubo, Shinjyuku-ku, Tokyo 169-8555, Japan
** CREST, Japan Science and Technology Agency
Email: ozaki@sic.shibaura-it.ac.jp

Abstract—There are robustness problems in the field of computational geometry. A correct result is output by rational arithmetic. However, an inexact result is output due to rounding errors by finite precision arithmetic. A condition number is frequently used for discussions of accuracy of computed results in the area of numerical analysis. In this paper, this concept is innovated to one of the basic geometric predicates 'two-dimensional orientation problem'.

1. Introduction

This paper is concerned with basic computations in computational geometry. We focus our mind on a two-dimensional orientation problem which is frequently used in applied geometric problems, for example, a convex hull for a set of points, a point-in-polygon problem and so on. Assume that an oriented line passes from $A = (a_x, a_y)$ to $B = (b_x, b_y)$. Let $C = (c_x, c_y)$ be a point. Let $\mathbb{F}$ be the set of floating-point numbers defined by IEEE 754 [1]. Assume that $A, B, C \in \mathbb{F}^2$. The problem is to distinguish which the point C is left to, right to, or on the oriented line AB. The problem can be solved by evaluating the following sign of the determinant:

$$\mathtt{sign}(\det(G)), \quad G := \begin{pmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{pmatrix}. \tag{1}$$

If the sign of the determinant is positive, then the point is left to the oriented line. If the sign of the determinant is negative, then the point is right to the oriented line. Otherwise, the point is on the oriented line.

If the sign of the determinant (1) is evaluated by pure floating-point arithmetic, then an inexact result may be obtained due to accumulation of rounding errors. When the point is very close to the oriented line, heavy cancellation occurs in the floating-point computations of (1), so that finally an incorrect result may be obtained. It yields serious problems. See [2] for detailed examples (these problems are called robustness problems). To overcome this problem, using multiple precision arithmetic is an option. However, to apply it straightforwardly makes the computational performance slow down due to software simulation. Therefore, adaptive approaches, for example [3, 4, 5], have been investigated. Initially, a so-called floating-point filter is applied. The filter quickly checks a sufficient condition for correctness of the sign of the determinant. If the filter cannot guarantee correctness of the sign of the determinant, then computational precision is increased and we check the correctness again. These procedures are repeated until the correctness of the result can be guaranteed. Therefore, computing time of the adaptive algorithm is related to difficulty of the problem.

Various sets of three points A, B and C are necessary so as to the compare performance of adaptive algorithms. We innovate condition numbers to the two-dimensional orientation problem. The condition numbers are frequently used for discussion of the accuracy of the numerical result. There are several ways to compute $\det(G)$ in (1), for example,

$$F_1 = (a_x - c_x)(b_y - c_y) - (a_y - c_y)(b_x - c_x), \tag{2}$$
$$F_2 = a_x(b_y - c_y) + b_x(c_y - a_y) + c_x(a_y - b_y), \tag{3}$$
$$F_3 = a_x b_y - a_x c_y - c_x b_y - a_y b_x + a_y c_x + c_y b_x. \tag{4}$$

We define condition numbers for each expression. Let them be $\mathtt{cond}(F_1)$, $\mathtt{cond}(F_2)$, $\mathtt{cond}(F_3)$. We give an algorithm which randomly outputs a set of three points. The resultant set gives almost same condition numbers for $\mathtt{cond}(F_1)$, $\mathtt{cond}(F_2)$, $\mathtt{cond}(F_3)$.

2. Condition Numbers for Orientation Problem

In this section, we innovate condition numbers to the two-dimensional orientation problem. $fl(\cdots)$ means that an expression in parentheses is evaluated by pure floating-point arithmetic. First, we define $\mathtt{cond}(F_1)$. For $a, b \in \mathbb{F}$, there is an algorithm which transforms $a + b$ into $x + y (x = fl(a + b))$ without rounding errors when overflow does not occur in $fl(a + b)$. Denote this algorithm as

$$[x, y] = \mathtt{TwoSum}(a, b).$$

See [8] for the detail of this algorithm. There is an algorithm which transforms $a \times b$ into $x + y (x = fl(a \times b))$ without rounding errors when neither overflow nor underflow occurs in the computation of the algorithm. Let this algorithm denote

$$[x, y] = \texttt{TwoProduct}(a, b).$$

See [7] for the detail of this algorithm. Applying `TwoSum` to each subtraction in (2) as follows:

$$\begin{array}{rcl} [t_1, e_1] & = & \mathrm{TwoSum}(a_x, -c_x), \\ [t_2, e_2] & = & \mathrm{TwoSum}(b_y, -c_y), \\ [t_3, e_3] & = & \mathrm{TwoSum}(a_y, -c_y), \\ [t_4, e_4] & = & \mathrm{TwoSum}(b_x, -c_x), \end{array} \tag{5}$$

(2) can be transformed into

$$\det(G) = (t_1 + e_1)(t_2 + e_2) - (t_3 + e_3)(t_4 + e_4). \tag{6}$$

Expanding (6) straightforwardly yields

$$t_1 t_2 + t_1 e_2 + t_2 e_1 + e_1 e_2 - t_3 t_4 - t_3 e_4 - t_4 e_3 - e_3 e_4. \tag{7}$$

Applying `TwoProduct` for each product in (7) as follows:

$$\begin{array}{rcl} [p_1, p_2] & = & \mathrm{TwoProduct}(t_1, t_2), \\ [p_3, p_4] & = & \mathrm{TwoProduct}(-t_3, t_4), \\ [p_5, p_6] & = & \mathrm{TwoProduct}(t_1, e_2), \\ [p_7, p_8] & = & \mathrm{TwoProduct}(t_2, e_1), \\ [p_9, p_{10}] & = & \mathrm{TwoProduct}(-t_3, e_4), \\ [p_{11}, p_{12}] & = & \mathrm{TwoProduct}(-t_4, e_3), \\ [p_{13}, p_{14}] & = & \mathrm{TwoProduct}(e_1, e_2), \\ [p_{15}, p_{16}] & = & \mathrm{TwoProduct}(-e_3, e_4), \end{array}$$

we finally obtain

$$\det(G) = \sum_{i=1}^{16} p_i.$$

This way is used in robust algorithms [4, 5]. We now introduce the condition number of summation of floating-point numbers. Let $v \in \mathbb{F}^n$, the condition number of summation is introduced in [6] as follows:

$$\mathrm{cond}(\sum_{i=1}^{n} v_i) = \frac{\sum_{i=1}^{n} |v_i|}{|\sum_{i=1}^{n} v_i|}.$$

Therefore, it is natural that the condition number for (2) is defined as

$$\mathrm{cond}(F_1) = \frac{\sum_{i=1}^{16} |p_i|}{|\sum_{i=1}^{16} p_i|}.$$

Next, we define $\mathrm{cond}(F_2)$. Applying `TwoSum` for the each subtraction in (3), we obtain

$$\begin{array}{rcl} [t_5, e_5] & = & \mathrm{TwoSum}(b_y, -c_y), \\ [t_6, e_6] & = & \mathrm{TwoSum}(c_y, -a_y), \\ [t_7, e_7] & = & \mathrm{TwoSum}(a_y, -b_y). \end{array} \tag{8}$$

After that, we have

$$\begin{array}{rcl} & & a_x(t_5 + e_5) + b_x(t_6 + e_6) + c_x(t_7 + e_7) \\ & = & a_x t_5 + a_x e_5 + b_x t_6 + b_x e_6 + c_x t_7 + c_x e_7 \end{array}$$

`TwoProduct` can be applied for each product in the abovementioned expression as follows:

$$\begin{array}{rcl} [q_1, q_2] & = & \mathrm{TwoProduct}(a_x, t_5), \\ [q_3, q_4] & = & \mathrm{TwoProduct}(a_x, e_5), \\ [q_5, q_6] & = & \mathrm{TwoProduct}(b_x, t_6), \\ [q_7, q_8] & = & \mathrm{TwoProduct}(b_x, e_6), \\ [q_9, q_{10}] & = & \mathrm{TwoProduct}(c_x, t_7), \\ [q_{11}, q_{12}] & = & \mathrm{TwoProduct}(c_x, e_7). \end{array}$$

Then, we have

$$\det(G) = \sum_{i=1}^{12} q_i.$$

Therefore, the condition number for (3) is defined by

$$\mathrm{cond}(F_2) = \frac{\sum_{i=1}^{12} |q_i|}{|\sum_{i=1}^{12} q_i|}.$$

From similar discussion, $\mathrm{cond}(F_3)$ can be defined. Applying `TwoProduct` for each product in (4) as follows:

$$\begin{array}{rcl} [r_1, r_2] & = & \mathrm{TwoProduct}(a_x, b_y), \\ [r_3, r_4] & = & \mathrm{TwoProduct}(a_y, c_x), \\ [r_5, r_6] & = & \mathrm{TwoProduct}(b_x, x_y), \\ [r_7, r_8] & = & \mathrm{TwoProduct}(a_x, -c_y), \\ [r_9, r_{10}] & = & \mathrm{TwoProduct}(a_y, b_x), \\ [r_{11}, r_{12}] & = & \mathrm{TwoProduct}(b_y, c_x), \end{array}$$

the condition number for (4) is defined by

$$\mathrm{cond}(F_3) = \frac{\sum_{i=1}^{12} |r_i|}{|\sum_{i=1}^{12} r_i|}.$$

This form is used in Shewchuk's algorithm 'Orient2dExact'.

Remark 1 *There is a set of three points which yields big difference of the condition numbers. For example, let three points be*

$$A = (2, 1), \quad B = (-1, -1), \quad C = (2^{100}, 2^{100}). \tag{9}$$

Then, $\mathrm{cond}(F_1) = 2.53e + 30$, $\mathrm{cond}(F_2) = 5$, $\mathrm{cond}(F_3) = 5$. *If one library uses the form (2) and the other library uses the form (3), comparing the performance of two libraries by (9) yields an unfair result.*

It is possible to give different orders of the expression (2), for example,

$$\begin{array}{rcll} F_1' & = & (c_x - b_x)(a_y - b_y) - (c_y - b_y)(a_x - b_x), & (10) \\ F_1\text{"} & = & (b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x). & (11) \end{array}$$

Condition numbers for F_1' *and* F_1*" can be defined from similar discussion. Let them be* $\mathrm{cond}(F_1')$ *and* $\mathrm{cond}(F\text{"}_1)$,

respectively. There is a case that the condition numbers are much different. For example, recalling (9), the condition numbers are $\mathrm{cond}(F_1) = 2.53e+30$, $\mathrm{cond}(F'_1) = 5$, $\mathrm{cond}(F"_1) = 5$.

We can derive the following approximation of the condition numbers

$$\begin{aligned}
\mathrm{cond}(F_1) &\approx \frac{|(a_x - c_x)(b_y - c_y)| + |(b_x - c_x)(a_y - c_y)|}{|(a_x - c_x)(b_y - c_y) + (b_x - c_x)(a_y - c_y)|}, \\
\mathrm{cond}(F_2) &\approx \frac{|a_x(b_y - c_y)| + |b_x(a_y - c_y)| + |c_x(a_y - b_y)|}{|a_x(b_y - c_y) + b_x(a_y - c_y) + c_x(a_y - b_y)|}, \\
\mathrm{cond}(F_3) &\approx \frac{|a_x b_y| + |a_x c_y| + |c_x b_y| + |a_y b_x| + |a_y c_x| + |c_y b_x|}{|a_x b_y - a_x c_y - c_x b_y + a_y b_x + a_y c_x - c_y b_x|}.
\end{aligned}$$

3. How to Generate Set

Let X be a required condition number. We propose an algorithm which randomly outputs a set of three points. Moreover, the set gives almost same condition numbers: $\mathrm{cond}(F_1)$, $\mathrm{cond}(F_2)$ and $\mathrm{cond}(F_3)$. Our algorithm consists on the following three parts.

3.1. Condition Number up to 1e+16

If a required condition number is less than 10^{16}, an algorithm developed in this subsection is used. First, we generate a_x, a_y, b_x, b_y, c_x at random[1]. Let S be

$$S = fl(|a_x b_y| + |a_y c_x| + |a_y b_x| + |b_y c_x|).$$

A vector w is generated as follows:

$$\begin{aligned}
[w_1, w_2] &= \mathrm{TwoProduct}(a_x, b_y), \\
[w_3, w_4] &= \mathrm{TwoProduct}(b_x, -a_y), \\
[w_5, w_6] &= \mathrm{TwoProduct}(c_x, a_y), \\
[w_7, w_8] &= \mathrm{TwoProduct}(b_y, -c_x).
\end{aligned}$$

Here, we introduce the accurate summation algorithm by Rump, Ogita and Oishi [11]. It is guaranteed that accuracy of the result by their algorithm is within 1 ulp (it is called faithful rounding). This function denotes $\mathtt{AccSum}(p)$ for a vector $p \in \mathbb{F}^n$ in INTLAB [9]. Let d be a sum of all elements in w. After that, c_y is obtained as follows:

$$d = \mathtt{AccSum}(w), \quad c_y = fl\left(\frac{S/X - d}{b_x - a_x}\right).$$

3.2. Condition Number up to 1e+32

We generate a_x, a_y, b_x, b_y at random. Let S be

$$S = fl(|a_x b_y| + |a_y b_x|).$$

The following vector w is generated as follows:

$$\begin{aligned}
[w_1, w_2] &= \mathrm{TwoProduct}(a_x, b_y), \\
[w_3, w_4] &= \mathrm{TwoProduct}(b_x, -a_y).
\end{aligned}$$

[1] If $c_x \approx a_x$ or $c_x \approx b_x$, then we regenerate c_x. Same discussion is held for c_y.

Let d be a sum of all elements in w. Then c_x is obtained by

$$d = \mathtt{AccSum}(w), \quad c_x = fl\left(\frac{S/X - d}{a_y - b_y}\right).$$

Next, we add four elements to the vector w as follows:

$$\begin{aligned}
[w_5, w_6] &= \mathrm{TwoProduct}(c_x, -b_y), \\
[w_7, w_8] &= \mathrm{TwoProduct}(c_x, a_y).
\end{aligned}$$

After that, d is updated and c_y is obtained as follows:

$$d = \mathtt{AccSum}(w), \quad c_y = fl\left(\frac{S/X - d}{b_x - a_x}\right).$$

3.3. Condition Number over 1e+32

If a_x, a_y, b_x and b_y are given at random, then it is almost impossible to get a set of points which gives condition numbers up to 10^{32}. First, we apply an ill-conditional matrix generator by Rump [10].

$$A = \mathtt{randmat}(2, X/1e+32);$$

This function is supported by INTLAB [9]. Then, let a_x, a_y, b_x and b_y be

$$A = \begin{pmatrix} a_x & a_y \\ b_x & b_y \end{pmatrix}.$$

Next, we generate a vector w as

$$\begin{aligned}
[w_1, w_2] &= \mathrm{TwoProduct}(a_x, b_y), \\
[w_3, w_4] &= \mathrm{TwoProduct}(b_x, -a_y).
\end{aligned}$$

Let d and c_x be defined as follows:

$$d = \mathtt{AccSum}(w), \quad c_x = fl\left(\frac{-d}{a_y - b_y}\right)$$

We extend a vector by adding four terms as follows:

$$\begin{aligned}
[w_5, w_6] &= \mathrm{TwoProduct}(c_x, -b_y), \\
[w_7, w_8] &= \mathrm{TwoProduct}(c_x, a_y).
\end{aligned}$$

Then, d is updated and c_y are computed by

$$d = \mathtt{AccSum}(w), \quad c_y = fl\left(\frac{-d}{b_x - a_x}\right).$$

This algorithm is able to output a set of three points which gives the condition numbers up to $cnd = 10^{64}$.

4. Numerical Examples

We show numerical examples by our algorithm. We used MATLAB's built-in function `randn` for generating points [12]. In Table 1, the item 'cnd' shows a required condition number. The items '$\mathrm{cond}(F_1)$', '$\mathrm{cond}(F_2)$' and '$\mathrm{cond}(F_3)$' show the condition numbers defined in Section 2. The results are averages of 100 examples. It is confirmed from Table 1 that our algorithm can output a set of points

Table 1: Comparison of the condition numbers.

cnd	$cond(F_1)$	$cond(F_2)$	$cond(F_3)$
10^5	2.39e+05	1.62e+05	2.47e+05
10^{10}	3.07e+10	2.22e+10	2.93e+10
10^{15}	2.47e+15	1.68e+15	2.49e+15
10^{20}	1.75e+20	1.76e+20	3.05e+20
10^{25}	1.71e+25	1.78e+25	2.91e+25
10^{30}	1.51e+30	1.76e+30	2.78e+30
10^{35}	8.82e+36	8.73e+36	8.82e+36
10^{40}	4.99e+40	4.99e+40	4.99e+40
10^{45}	1.35e+46	1.35e+46	1.35e+46
10^{50}	3.51e+50	3.51e+50	3.51e+50

Table 2: Difference of condition numbers.

cnd	max	mean	min
10^5	11.5	3.03	1.01
10^{10}	24.5	3.01	1.04
10^{15}	9.68	2.70	1.00
10^{20}	30.0	3.91	1.03
10^{25}	75.4	5.29	1.00
10^{30}	43.7	9.63	1.00
10^{35}	1.91	1.05	1.00
10^{40}	1.00	1.00	1.00
10^{45}	1.00	1.00	1.00
10^{50}	1.00	1.00	1.00

with almost required condition number for each $\mathrm{cond}(F_1)$, $\mathrm{cond}(F_2)$ and $\mathrm{cond}(F_3)$. Next, we check the difference among $\mathrm{cond}(F_1)$, $\mathrm{cond}(F_2)$ and $\mathrm{cond}(F_3)$. Table 2 shows the maximum, mean and minimum of

$$\frac{\max\left(\mathrm{cond}(F_1), \mathrm{cond}(F_2), \mathrm{cond}(F_3)\right)}{\min\left(\mathrm{cond}(F_1), \mathrm{cond}(F_2), \mathrm{cond}(F_3)\right)}$$

for 100 examples. From Table 2, the difference among condition numbers of F_1, F_2 and F_3 is not a lot.

Next, we checked difference of $\mathrm{cond}(F_1)$, $\mathrm{cond}(F_1')$ and $\mathrm{cond}(F"_1)$. Table 2 shows the maximum, mean and minimum of

$$\frac{\max\left(\mathrm{cond}(F_1), \mathrm{cond}(F_1'), \mathrm{cond}(F_1")\right)}{\min\left(\mathrm{cond}(F_1), \mathrm{cond}(F_1'), \mathrm{cond}(F_1")\right)}$$

for 100 examples. It is confirmed from Table 3 that there is not so much difference among condition numbers except $cnd = 10^{50}$. When $cnd = 10^{50}$, we found that b_x and b_y are relatively large. Then $\mathrm{cond}(F_1')$ is approximately $|b_x b_y|$. There is no $b_x b_y$ in F_1 and F_1". Therefore, it yields big difference.

Table 3: Difference of condition numbers.

cnd	max	mean	min
10^5	7.83	3.15	2.03
10^{10}	9.09	3.23	2.01
10^{15}	11.5	3.32	2.00
10^{20}	12.0	3.57	2.01
10^{25}	23.1	3.39	2.01
10^{30}	17.8	3.30	2.00
10^{35}	3.74	2.32	2.00
10^{40}	3.82	2.39	2.00
10^{45}	3.48	2.42	2.00
10^{50}	161	112	77.0

References

[1] *IEEE Standard for Floating-Point Arithmetic*, Std 754–2008, 2008.

[2] L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, C. Yap: Classroom Examples of Robustness Problems in Geometric Computations, Computational Geometry 40, 1 (2008), 61-78.

[3] G. Melquiond and S. Pion, Formally certified floating-point filters for homogenous geometric predicates, Theoretical Informatics and Applications, Special issue on Real Numbers, vol. 41 (2007), 57-69.

[4] J. R. Shewchuk: Adaptive precision floating-point arithmetic and fast robust geometric predicates, Discrete & Computational Geometry, 18 (1997), 305–363.

[5] K. Ozaki, T. Ogita, S. M. Rump, S. Oishi: Adaptive and Efficient Algorithm for 2D Orientation Problem, accepted for Japan Journal of Industrial and Applied Mathematics (JJIAM), 26 (2009), 215-231.

[6] T. Ogita, S.M. Rump, S. Oishi, Accurate sum and dot product, SIAM J. Sci. Comput., 26(6): 1955-1988, 2005.

[7] T.J. Dekker, A floating-point technique for extending the available precision, Numer. Math., 18: 224–242, 1971.

[8] D.E. Knuth, The Art of Computer Programming: Seminumerical Algorithms, volume 2, Addison-Wesley, Reading, Massachusetts, 1969.

[9] S.M. Rump: INTLAB - INTerval LABoratory. In Tibor Csendes, editor, Developments in Reliable Computing, Kluwer Academic Publishers, Dordrecht, 1999, 77-104.

[10] S.M. Rump: A Class of Arbitrarily Ill-conditioned Floating-Point Matrices, SIAM Journal on Matrix Analysis and Applications, 12(4), 1991, 645-653.

[11] S.M. Rump, T. Ogita, and S. Oishi: Accurate floating-point summation part I: Faithful rounding. SIAM J. Sci. Comput., 31(1), 2008, 189-224.

[12] MATLAB User's Guide, Version 7. The MathWorks Inc., 2004.