"Accordingly, the defendant is found Not Guilty. However, this Court is of the opinion that variable-precision floating point is perilous, and its use should be restricted to qualified professionals."

— Dirk Laurie
in "Variable-precision Arithmetic Considered Perilous — A Detective Story"
Electronic Trans. on Numerical Analysis, Vol. 28, pp. 168–173, 2008.

Floating point arithmetic has numerous engineering advantages: it is well-supported... the Challenge is to demonstrate that a reliable implementation can result from the use of floating point arithmetic.

– Steve Fortune

Lecture 4
ARITHMETIC TECHNIQUES

Since numerical errors are the cause of nonrobustness, it stands to reason that if we make numerical computations more accurate, we improve the robustness of our programs. In this section, we look at some techniques for improving numerical accuracy. In particular, we look at floating point techniques to determine the correct sign of a sum, and also at robust rotations using rational angles.

§1. More Accurate Machine Arithmetic

Since numerical error is the source of non-robustness, it is natural to try to improve upon the standard floating point arithmetic. Can we improve on the conventional floating point arithmetic, which has already reached a highly optimal design under the IEEE Standard? More generally, what techniques are available in the FP mode of computation?

In conventional floating point systems, the number of bits allocated to the mantissa and the exponent are fixed. Matsui and Iri [21] proposed a system where these can vary. But let us look at a remarkable proposal of Clenshaw, Olver and Turner [5] called **level-index arithmetic** (LI).

First, any non-negative real number x is assigned a unique **level** $\ell(x) \in \mathbb{N}$ defined by the property that $\ln^{(\ell(x))}(x) \in [0,1)$, where $\ln^{(n)}(x)$ denotes n applications of the natural logarithm ln. The **index** of x defined by $i(x) := \ln^{(\ell(x))}(x)$. The function $\psi(x) = \ell(x) + i(x)$ is called the **generalized logarithm**; its inverse $\phi(x)$ is the **generalized exponential** defined by

$$\phi(x) = \begin{cases} x & \text{if } 0 \le x < 1 \\ e^{\phi(x-1)} & \text{else.} \end{cases}$$

Thus any real number x can be represented by a triple $LI(x) := (s(x), \ell(|x|), i(|x|))$ where $s(x) \in \{\pm 1\}$ is defined so that $s(x) = +1$ iff $x \ge 0$.

The symmetric version of LI is called **symmetric LI** or SLI [6]. In SLI, a non-zero number x with magnitude < 1 is represented by the level index representation of $1/x$, otherwise it basically has the same representation as before. More precisely, $x \in \mathbb{R}$ is represented by a quadruple

$$SLI(x) := (r(x), s(x), \ell(|x|^{r(x)}), i(|x|^{r(x)})) \tag{1}$$

where $r(x) = \pm 1$. We define $r(x) = -1$ iff ($|x| < 1$ and $x \neq 0$). Suppose $SLI(x) = (r', s', \ell', i')$. Then $x = 0$ iff $i' = 0$. Assuming $i' \neq 0$, the triple (s', ℓ', i') is either $LI(x)$ or $LI(1/x)$ (depending on whether $r' = +1$ or -1).

Now consider a fixed-precision version of the SLI representation: suppose we have a fixed budge t of bits to represent a number x. We use two bits for $r(x), s(x)$. It is suggested that, for all practical purposes, allocating 3 bits to $\ell(x)$ suffices. Indeed, with $\ell(x) = 5$, x can reach numbers as high as $2^{2^{5,500,000}}$. Thus $t - 5$ bits are available for representing the index, $i(x)$. There are many remarkable properties of this representation: its range is so hugh that perhaps only astronomers would fully appreciate it. The representable numbers have gaps that are much more smoothly distributed than standard floating point representation. Overflow and underflow are practically eliminated: assuming the 3-bit scheme and $t < 5,500,000$, there is no overflow or underflow in the four arithmetic operations [20]. Nevertheless, the complexity of the algorithms for basic arithmetic in LI/SLI, and their less familiar error analysis, conspire to limit its wider use. On the other hand, in the early days of modern computing, that was exactly the same prognosis for floating-point arithmetic vis-à-vis fixed-point arithmetic. Hence there is perhaps hope for SLI.

Another way to improve the accuracy of fixed precision arithmetic is to introduce some accurate primitives for critical operations. One such critical operation is the scalar product $\sum_{i=1}^{n} a_i b_i$ of two vectors. In other words, for n within a reasonable range, we would like this scalar product to yield the closest representable number. The usefulness of such extensions for problems such as line segment intersection was pointed out in [24]. In recent years, there has appeared in hardware an operation called FMA (**fused multiply and add**). This is the ternary operation of $FMA(a, b, c) = ab + c$. E.g., we can implement an accurate scalar product operation using only n FMA's.

Yet another direction is to go to **arbitrary precision arithmetic** (AP mode). In the AP mode, one can either use arbitrary precision floating point numbers, or use arbitrary precision rational numbers. The former is inherently an approximation-based computation while the latter is error-free when a computation uses only the rational operations $(\pm, \times, \div)$. The last section of this chapter explores techniques for performing rotations using arbitrary precision rational numbers.

The use of **interval arithmetic** or more generally, **validated computation**, [23, 1, 19] is also major direction for achieving robustness. Interval arithmetic is independent of the FP mode: it is just as useful in arbitrary precision computation. The interval bounds can rapidly increase, unless one counteracts this by actively reducing these intervals, using increasing precision. In this sense, arbitrary precision computation is a better setting for applications of interval arithmetic.

Software and Language Support. Ultimately any approach to robustness ought to be supported by appropriate software tools, so that it is easy to implement robust algorithms using the particular approach. For instance, to support interval methods, an extension of `Pascal` called `Pascal-SC` has been developed [2]. This language supports validated arithmetic with facilities for rounding, overloading, dynamic arrays and modules. Further extensions have been made in the languages `PASCAL-XSC`, `C-XSC` and `ACRITH-XSC`. The language **Numerical Turing** [14, 15, 16] has a concept of "precision block" whereby the computation within such a block can be repeated with increasing precision until some objective is satisfied. Despite the success of the IEEE Arithmetic in hardware, it has yet to be fully at the level of of programming languages. In particular, the proper treatment of arithmetic exceptions [8] is still lacking.

§2. Summation Problem

The previous section discuss some general proposals to use more general representations for a suitable subsets of the real numbers. These ideas suggests that, for any given bit-budget, there are extensions of the standard floating point representation that can increase the range and accuracy of the standard representations.

In this section, we consider another line of attack: we focus on certain critical computations and see if we can improve the accuracy of performing these operations in the standard model (Lecture 2, §5) of numerical computation.

We focus on the **summation problem**, which is to compute the sum

$$S = \sum_{i=1}^{n} a_i \tag{2}$$

of a sequence $a_1, a_2, \ldots, a_n$ of input numbers. Mathematically, summation is a triviality, but within FP computation, this is[1] no longer true. Such summations are central to more complex operations such as matrix-vector multiplications or the scalar product of two vectors. At any rate, it is the simplest computation with a cascaded sequence of primitive arithmetic steps which we can study. In Chapter 2, we already analyzed the error for the obvious algorithm for this problem. Higham [13, chapter 4] treats this problem in detail.

In this section, we want to introduce a remarkable algorithm called "compensated summation". By way of motivation, consider the sum $\widetilde{s} = [a+b]$ where a, b are floating point numbers with $|a| \ge |b|$. Here $[a+b]$ uses the bracket notation to denote floating point arithmetic in the standard model discussed in Chapter 2.

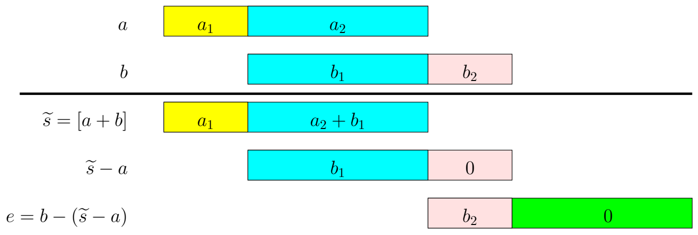

Figure 1: Estimated error, $e = (b - ((a+b) - a))$.

In Figure 1, we have illustrated a sequence of three computational steps starting from a pair of floating point numbers a, b:

$$\begin{aligned} \widetilde{s} &:= a+b \\ \widetilde{t} &:= \widetilde{s} - a \\ e &:= b - \widetilde{t} \end{aligned}$$

The results of these operations are aligned to give intuition to an error analysis of these steps. From studying this figure, a reader might find it quite plausible that the computed quantity

$$e = (b - ((a+b) - a)),$$

has the property that $|e|$ is the error in $\widetilde{s}$ as an approximation to $a+b$. Indeed, this is a result of Dekker:

Lemma 1 (Dekker (1971)). *Let $|a| \ge |b|$ and compute the expression*

$$\widetilde{e} = [b - [\widetilde{s} - a]] = [[a - [a+b]] + b].$$

If the floating point arithmetic is in base 2, then

$$a + b = \widetilde{s} + \widetilde{e}. \tag{3}$$

In other words, $|\widetilde{e}|$ is precisely the error in $\widetilde{s}$.

This gives rise to the following summation algorithm.

```
Compensated Sum Algorithm
Input: Floating point numbers $a_1, \ldots, a_n$
Output: Approximate sum $s = \sum_{i=1}^n a_i$
    $s \leftarrow 0$; $e \leftarrow 0$;
    for $i = 1$ to $n$
        $temp \leftarrow s$;
        $a \leftarrow [a_i + e]$;   ◁ Compensated summand
        $s \leftarrow [temp + a]$;   ◁ Cumulative sum
        $e \leftarrow [[temp - s] + a]$;   ◁ Error estimate
```

[1] This is true of many issues in FP computation: mathematical trivialities that aren't.

The idea is to estimate the error of the addition at each step, and to add this error back to the partial sum. Note that this error estimate is not exact because Dekker's analysis depends on the inequality $|a| \geq |b|$ which we do not guarantee. Nevertheless, it can be shown (see [18, pp. 572–573] and [11]) that the computed value is equal to

$$\widetilde{s} = \sum_{i=1}^{n} (1+\delta_i)a_i, \qquad |\delta_i| \leq 2u + O(nu^2). \tag{4}$$

So the relative backwards error in each argument a_i is about $2u$. This is much better than the corresponding error in our analysis in Lecture 2: we showed that the relative backwards error is $\gamma_{n-i+1} \approx (n-i+1)u$ for each a_i $(i = 1, \ldots, n)$.

Exercises

Exercise 2.1:
(i) Implement the Compensated Sum Algorithm in same standard programming language.
(ii) Randomly generate sequences of floating point numbers in the range $[-1, 1]$ and compute their sum using three methods:
(A) the straightforward method in machine double precision,
(B) your compensated sum algorithm from (a), and
(C) an exact algorithm using BigFloats.

Use sequences of length $5, 10, 50, 100, 1000$, and compute some statistics about their relative accuracy and speeds. ♢

Exercise 2.2: Prove Dekker's lemma. ♢

Exercise 2.3: Prove the error bound (4). ♢

Exercise 2.4: Let $x_1, \ldots, x_n$ be a sequence of numbers. Consider the following five strategies for forming their sum:

- (O) Random reordering (if your input is randomly generated, you do not need to do anything).
- (A) Sort the numbers in non-decreasing order and add them in a simple loop.
- (B) As in (A), but first sort the numbers in non-increasing order.
- (C) Put the numbers in a min-priority queue, and at each iteration, extract two numbers, and insert their sum back into the queue. We halt when there is only one number in the queue. [This method is called "Psum" in [13, p. 90].
- (D) As in (C), except we use a max-priority queue.

(i) Experimentally, show that when the numbers are positive, (A) and (C) are superior to (B) and (D), respectively.
(ii) Give examples in which (B) and (D) are better than (A) and (C).
(iii) Compare the performance of (A) and (C) for positive numbers. ♢

End Exercises

§3. Sign of Sum Problem

A simplification of the Summation Problem is the following **Sign of Sum Problem** (SoS Problem): *Given a sequence $a_1, a_2, \ldots, a_n$ of numbers, determine the sign of the sum $S = \sum_{i=1}^{n} a_i$.* An exact solution for SoS can be used to solve the non-robustness of problems such as convex hulls. To see this, observe that all convex hull algorithms ultimately depend on the predicate $\mathsf{Orientation}(p, q, r)$. According to the principles of exact geometric computation (EGC), the convex hull would be computed exactly provided all such predicates are evaluated without error. But the Orientation predicate can be reduced to the sign of a

2×2 determinant, which is an expression of the form $ad - bc$; this is the SoS Problem with $n = 2$. For a more accurate analysis, we should view it as a special 3×3 determinant, which expands to

$$\Delta = (b-a)(c'-a') - (c-a)(b'-a') = bc' - ac' - ac' - cb' + ab' + ca'. \tag{5}$$

This is SoS with $n = 6$.

Ratschek and Rokne [25] introduced an exact algorithm for SoS, called[2] ESSA. But before going into the details of ESSA, let us develop some intuitions. Suppose you are given a sequence of numbers, and you want to determine the sign of its sum. How can you organize the arithmetic to improve its reliability over the naive summation? The simplest idea is to break the sum into two parts, comprising the positive and negative numbers. Assume the positive part is $A = \sum_{i=1}^{k} a_i$ (with $a_i > 0$) and the negative part is $B = -\sum_{j=1}^{n-1} b_j$ (where $b_j > 0$). Then we just have to compare A and B.

We now focus on improving the summation process. We only need to focus on the first sum A. Suppose the a_i's are sorted in increasing order and we form the cumulative sum from smallest to largest. The intuition reason for this is clear – we are trying to minimize the truncation error by adding comparable size numbers (recall that in floating point addition, we first align the two numbers at their binary point, truncate the least significant bits of the smaller number, and then add). But this sorting is not optimal, and does not always add two numbers that are as comparable as possible. To fix this, we next put all the a_i's into a min-priority queue and at each step, pull two elements from the queue, add them, and insert the sum back into the queue. Call this the **priority queue SoS Algorithm**.

The Priorty Queue SoS Algorithm seems pretty good. Of course, it is not exact. We now address the ESSA Algorithm which is based on a similar idea, but it achieves exactness by tracking any truncation error. We assume the arithmetic is in the floating point system $F(2, t)$ (for some t) and

$$n \leq 2^{t-1}. \tag{6}$$

Recall from Chapter 2 that $F(2, t)$ comprise all numbers of the form $b_1.b_2 \ldots b_t \times 2^e$ where $b_i \in \{0, 1\}$ and $e \in \mathbb{Z}$. The exponent does not overflow or underflow in $F(2, t)$. By sorting the inputs in a preprocessing step, we may further assume

$$a_1 \geq a_2 \geq \cdots a_k > 0 > a_{k+1} \geq \cdots \geq a_n.$$

By renaming the negative inputs a_{k+i} as $-b_i$ ($i = 1, \ldots, \ell = n-k$) the problem is now reduced to determining the sum

$$S = A - B := \sum_{i=1}^{k} a_i - \sum_{j=1}^{\ell} b_j.$$

Let E_i be the exponent part of a_i and F_j the exponent part of b_j. Thus, $2^{E_i} \leq |a_i| < 2^{E_i+1}$. The algorithm goes as follows:

[2]ESSA stands for Exact Sign of Sum Algorithm. Beware that the other acronym "SoS" also refers to "Simulation of Simplicity", a technique for treating data degeneracies from Edelsbrunner and Muecke.

```
ESSA
Input: $a_i, b_j$ $(i = 1, \ldots, k, j = 1, \ldots, \ell)$ as above.
Output: The sign of $S = \sum_i a_i - \sum_j b_j$
1.   (BASIS) Terminate with the correct output in the following cases:
     1.1   If $k = \ell = 0$ then $S = 0$.
     1.2   If $k > \ell = 0$ then $S > 0$.
     1.3   If $\ell > k = 0$ then $S < 0$.
     1.4   If $a_1 \ge \ell 2^{F_1+1}$ then $S > 0$.
     1.5   If $b_1 \ge k 2^{E_1+1}$ then $S < 0$.
2.   (AUXILIARY VARIABLES)
         $a' = a'' = b' = b'' = 0$;
3.   (PROCESSING THE LEADING SUMMANDS)
         CASE $E_1 = F_1$:
             If $a_1 \ge b_1$ then $a' \leftarrow a_1 - b_1$
             Else $b' \leftarrow b_1 - a_1$;
         CASE $E_1 > F_1$:
             $u \leftarrow 2^{F_1+1}$;
             $a' \leftarrow a_1 - u$, $a'' \leftarrow u - b_1$;
         CASE $E_1 < F_1$:
             $v \leftarrow 2^{E_1+1}$;
             $b' \leftarrow b_1 - u$, $b'' \leftarrow u - a_1$;
4.   (UPDATE BOTH LISTS)
         Discard $a_1, b_1$ from list.
         Insert $a', a''$ into the $a$-list (only non-zero values need to be inserted).
         Insert $b', b''$ into the $b$-list (only non-zero values need to be inserted).
         Update the values of $k, \ell$ and return to Step 1.
```

To see the correctness of this algorithm, we note the following properties:

- *Basis.* The correctness of Step 1 is easy. For instance, Step 1.4 is true because $\sum_{j=1}^{\ell} b_j < \ell 2^{F_1+1}$.
- *Each arithmetic operation is error free.* This concerns only Step 3: In case $E_1 = F_1$, the value $a' \leftarrow a_1 - b_1$ is computed exactly. In case $E_1 > F_1$, then $a'' \leftarrow u - b_1$ is exact because the exponent of u is exactly one more than the exponent of b_1 and so (because of the guard bit) the scaling of b_1 before the operation incurs no loss in precision. To see that $a' \leftarrow a_1 - u$ is also exact, we use the fact that the non-execution of Step (1.4) implies $a_1 < \ell 2^{F_1+1}$ and hence by assumption (6), we have $a_1 < 2^{F_1+t}$. This implies $E_1 \le F_1 + t - 1$. In the operation $a_1 - 2^{F_1+1}$, we need to scale the operand 2^{F_1+1} by right shifting $E_1 - F_1 - 1$ positions. Since $E_1 - F_1 - 1 \le t$, and again because of the guard bit, no truncation error occurs.
- *Invariance of sum S.* The main computation is represented by Step 3. At the end of Step 3, we verify that the following invariant holds

  $$a_1 - b_1 = (a' + a'') - (b' + b'').$$

  This implies that updating the a- and b-lists in Step 4 preserves the value $S = A - B$ where A, B are the sum of the two lists.
- *Partial Correctness.* From the invariance of the sign, and the correctness of Step 1, this proves that the algorithm is correct if it halts.
- *Termination.* This follows by observing that the largest element in either the a-list or the b-list is reduced in each iteration.

Although termination is guaranteed but it may be slow in the worst case. In practice, this worst case behaviour may be an issue. If we apply ESSA to our 2×2 determinant $ad - bc$, there are two places where the numbers may overflow or underflow in a finite precision number system (not $F(2,t)$ but $F(2,t,e_{\min},e_{\max})$). First the values a,b,c,d really arise as the difference of two floating point numbers So this may cause an overflow or underflow in the values a,b,c,d. The possibility of overflow or underflow is reduced if we view the problem as an ESSA problem with $n=6$ as in (5). Each of the 6 terms is a produce of two floating point numbers, and a potential overflow still exists; but there are well-known ways to handle this [25].

Exercises

Exercise 3.1:
(i) Complete the proof that the ESSA algorithm is correct.
(ii) Describe modifications to the algorithm when exponents are limited in the range $[e_{\min}, e_{\max}]$. $\diamondsuit$

Exercise 3.2: Consider the Priority Queue SoS Algorithm.
(i) Give examples where this algorithm is inexact.
(ii) Improve this algorithm by working on the priority queues for A and B at the same time. $\diamondsuit$

Exercise 3.3: Implement ESSA. Experimentally compare its performance against the following alternative methods:
(E) Using ESSA
(N) Naive Method (no reordering), just add and compare.
(S) Separate the input numbers into positive and negative parts, and add them separately. Then determine their sign.
(C) As in (S), but use the Compensated Sum method to add the separate positive and negative parts.

We want to use two classes of inputs:
(i) One class of inputs should be a random input sequences of numbers taken from the range $[-1,1]$. Use sequences of length $5, 10, 50, 100, 1000$.
(ii) The second class of inputs are those that sum to zero. For this, general sequences of numbers from the range $[0,1]$, and then append the negation of the input. Then randomly permute the numbers before feeding to your algorithms.

As usual, compute some statistics about their relative accuracy and speeds. $\diamondsuit$

Exercise 3.4: (i) Analyze the worst case behavior of ESSA.
(ii) Give numerical examples with this behavior. $\diamondsuit$

Exercise 3.5: Show that for all $p \ge 1$, if $2^{-p}b < a < 2^p b$ then the error in computing $a-b$ is at most $p-1$ bits. $\diamondsuit$

End Exercises

§4. Evaluation Strategies for Orientation Predicate

In the previous two sections, we studied two numerical problems (Summation and Sign of Sum). In this section, we focus on a more specifically geometric problem.

The orientation predicate is a critical one in many elementary problems of computational geometry. In this section, we explore some heuristics for the accuracy of floating point evaluation of this predicate. Note that even if we are ultimately interested in the *exact* sign of sum, it is useful to have heuristic techniques which may not be always correct, but are usually correct. Such techniques, if they are efficient, can be used as filters in exact computation.

There are essentially 3 evaluation strategies for the orientation predicate $\mathsf{Orientation}(p,q,r)$ (see Lecture 3,§4). We consider the strategies for choosing one of p, q or r as the **pivot** for forming the 2×2 determinant $ad-bc$. For instance, if p is the pivot then the entries a,b,c,d are the coordinates of $q-p$ and $r-p$:

$$a = q.x - p.x, \quad b = q.y - p.y, \quad c = r.x - p.x, \quad d = r.y - p.y.$$

Which is the best choice of a pivot?

Fortune [10] suggests to choose the pivot which minimizes

$$|ad| + |bc|. \tag{7}$$

If we interpret $|ad|$ and $|bc|$ as the area of suitable boxes involving the coordinates of p, q, r, than this amounts to choosing the pivot whose corresponding boxes have least total area.

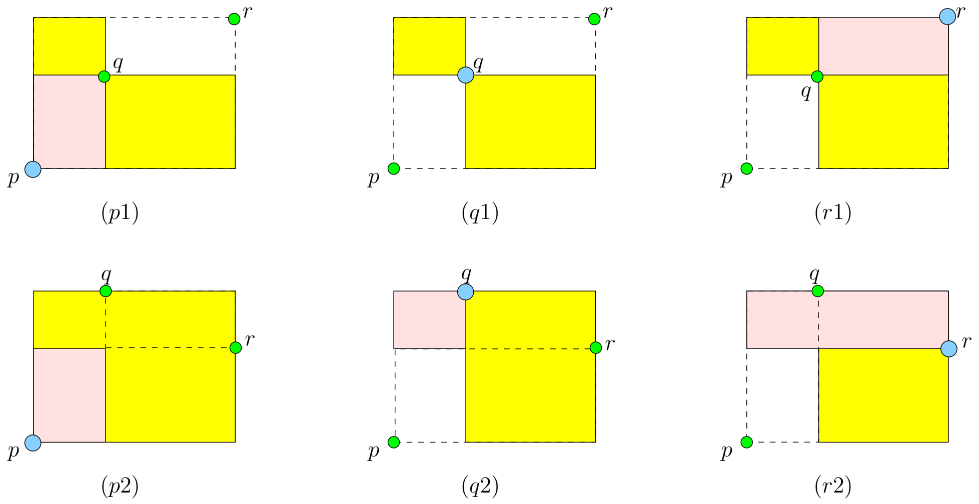

Figure 2: Evaluation Strategies: pivots are indicated by the larger (blue) circles.

Let B be the smallest axes-parallel box containing p, q, r. In terms of pivot choices that depends on $|ad|$ and $|bc|$, there are basically two distinct configurations of p, q, r. These correspond to the two rows in Figure 2: the top row corresponds to the case where two points (say p, r) are corners of B. The bottom row corresponds to the case where only one point (say p) is a corner of B. In each row, we show the three possible choice of pivots: p, q or r. The two boxes corresponding to $|ad|$ and $|bc|$ are drawn, and any overlap region is shown in darker shade. It is clear from these pictures that in the top row, pivoting at q will minimize the area (7); in the bottom row, the minimum is achieved by pivoting at q or r. Summarizing all these cases: the minimum is achieved if we pivot at the points whose x-coordinate is the median among the x-coordinates of p, q, r. Naturally, we could also use the y-coordinates.

How can we justify this heuristic? Recall (Chapter 2) that in the standard floating point model, each machine operation $\circ \in \{+, -, \times, \div, \sqrt{\cdot}\}$ satisfy the following property:

$$[x \circ y] = (x \circ y)(1 + \varepsilon)$$

where $|\varepsilon| \le \mathbf{u}$ (the unit roundoff), and the bracket notation $[\cdots \circ \cdots]$ indicates the machine operation corresponding to $\circ$. The numbers a, b, c, d in the orientation predicate are not exact, but obtained as the difference of two input numbers. Let the floating point results of these differences be a', b', c', d'. Thus

$$\begin{array}{rcl}
a' & = & a(1+\varepsilon_a), \qquad b' = b(1+\varepsilon_b), \qquad c' = c(1+\varepsilon_c), \qquad d' = d(1+\varepsilon_d) \\
{[a'd']} & = & ad(1+\varepsilon_a)(1+\varepsilon_d)(1+\varepsilon_{ad}) \\
{[b'c']} & = & bc(1+\varepsilon_b)(1+\varepsilon_c)(1+\varepsilon_{bc)} \\
{[[a'd'] - [b'c']]} & = & \big(ad(1+\varepsilon_a)(1+\varepsilon_d)(1+\varepsilon_{ad}) - bc(1+\varepsilon_b)(1+\varepsilon_c)(1+\varepsilon_{bc)}\big)\,(1+\varepsilon_{abcd}), \\
 & = & (ad)(1+\theta_0) - (bc)(1+\theta_1)
\end{array}$$

where each $|\varepsilon_i| \le \mathbf{u}$ and hence $|\theta_j| \le \gamma_4$ (where γ_4 is defined in Chapter 2.4). *Assuming that the ε_i's are independent*, the *worst case* error absolute error is $(|ad| + |bc|)\gamma_4$. This error is minimized when we minimize $|ad| + |bc|$.

Let $\mathsf{Orientation}(p, q, r, k)$ be the evaluation of $\mathsf{Orientation}(p, q, r)$ using the k-th point as pivot. That is, $k = 1$ means pivot at p; $k = 2$ means pivot at q, and $k = 3$ means pivot at r. The code goes like this:

```
Fortune's Pivoting Strategy
    Input: point $p, q, r$
    Output: approximate Orientation$(p, q, r)$
    If $(p.x < q.x)$ then
        If $(q.x < r.x)$ then $k = 2$,
        Else if $(p.x < r.x)$ then $k = 3$,
        Else $k = 1$.
    Else If $(p.x < r.x)$ then $k = 1$,
        Else $(q.x < r.x)$ then $k = 3$,
        Else $k = 2$.
    Return(Orientation$(p, q, r, k)$).
```

This is only a heuristic is because the errors (the various ε_i's) are not really independent. So it is possible that a deeper analysis will give a better worst case bound. The Exercises ask you to explore other heuristics for choosing the pivot.

The above implementation was discussed and used in an stable algorithm of Fortune for maintaining Delaunay Triangulations.

Exercises

Exercise 4.1: Conduct some experimental studies to investigate the effectiveness of Fortune's pivoting strategy. ♢

Exercise 4.2: Consider implementations of Fortune's pivoting strategy, trying to minimize the number of arithmetic operations:
(i) What is the maximum and minimum number of comparisons on any input p, q, r?
(ii) The goal is to determine the median value in $p.x, q.x, r.x$. An alternative strategy here is to begin by computing the differences $q.x - p.x$ and $r.x - p.x$. What does one do next?
(iii) Compare the approach in (ii) with the one in the text. ♢

Exercise 4.3: Extend the pivot-choice heuristic of Fortune to the 3-dimensional orientation predicate. Conduct some experiments to see the efficacy of your strategy. ♢

Exercise 4.4: Explore other pivot choice heuristics for Orientation(p, q, r). Experimentally compare Fortune's heuristic with the pivot corresponding to the smallest (or largest) angle in the triangle $\Delta(p, q, r)$. ♢

Exercise 4.5: Analyze more carefully the optimal choice of a pivot for the 2-dimensional orientation predicate. In particular, do not assume that the round-off errors of the operations are independent. ♢

Exercise 4.6: Apply the same heuristic to the evaluation of the InCircle predicate. ♢

End Exercises

§5. Exact Rotation

So far, we considered arithmetic techniques based on fixed precision floating point arithmetic. This section considers arbitrary precision arithmetic. One approach to nonrobustness is to require that all arithmetic operations are performed exactly. The simplest class of problems where such **exact arithmetic approach** is applicable is when our computation require only rational operations $(\pm, \times, \div)$, and the input numbers are rational. For instance, computing the convex hull of a set of points can be treated by this approach.

But what if irrational operations are called for? One class of operations that is frequently desired in geometric modeling is rigid transformations, defined to be transformations that preserve distance and orientation. In the plane, rigid transformations can be decomposed into a translation $(x, y) \mapsto (x + a, y + b)$ where $a, b \in \mathbb{R}$ composed with a rotation, $(x, y) \mapsto (x, y) \cdot R(\theta)$ where

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

is a 2×2 matrix representing rotation by angle θ. The entries in $R(\theta)$ involve the transcendental functions $\sin(\theta), \cos(\theta)$. Such functions are a problem for exact computation. In this section, we look at some special classes of angles for which $\sin(\theta), \cos(\theta)$ can be computed exactly.

A number $z \in \mathbb{C}$ is algebraic if it is the root of a polynomial $P(X) \in \mathbb{Z}[X]$, $P(z) = 0$. For instance, $\mathbf{i} = \sqrt{-1}$ is algebraic as it is the root of $P(X) = X^2 + 1$. A basic fact is that the algebraic numbers are closed under the four arithmetic operations. If $z \in \mathbb{C}$ is not algebraic, it is **transcendental**. For instance, $\pi = 3.1415926\cdots$ and $e = 2.718281828\cdots$ are transcendental. We recall a basic result from transcendental number theory. Lindemann's theorem says that if $\theta \in \mathbb{C}$ is non-zero and algebraic, then e^θ is transcendental.

Lemma 2. *If $\theta \in \mathbb{R}$ is non-zero then either θ or $\cos\theta$ must be transcendental.*

Proof. Clearly, for any θ, either both $\cos\theta, \sin\theta$ are algebraic or both are transcendental. If θ is transcendental, we are done. Otherwise, Lindemann's theorem implies that $e^{\mathbf{i}\theta}$ is transcendental. Then the identity $e^{\mathbf{i}\theta} = \cos\theta + \mathbf{i}\sin\theta$ implies that $\cos\theta$ and $\sin\theta$ must be transcendental. **Q.E.D.**

We also recall the Gelfond-Schneider theorem: if $a, b \in \mathbb{C}$ such that b is algebraic, $b \notin \{0, 1\}$, and a is algebraic and irrational, then b^a is transcendental. Most of the "well-known angles" such at 45° or 60° have the property that their sine or cosines are rational. Of course, these well-known angles are commensurable with π. Two quantities a, b are commensurable means that a/b are rational. How general is this observation? The following is from [4]:

Lemma 3. *If $\cos\theta$ and θ/π are algebraic, the θ and π are commensurable.*

Proof. Write $\theta = \alpha\pi$ for some α. By assumption, α is algebraic. But we must prove that it is, in fact, rational. Since we assume $\cos\theta$ is algebraic, so is $e^{\mathbf{i}\theta} = \cos\theta + \mathbf{i}\sin\theta$. Hence

$$e^{\mathbf{i}\theta} = e^{\mathbf{i}\pi\alpha} = \left(e^{\mathbf{i}\pi}\right)^\alpha = (-1)^\alpha.$$

By Gelfond-Schneider, if α is irrational, then $(-1)^\alpha$ is transcendental. But since $(-1)^\alpha$ is algebraic, we conclude that α is rational. **Q.E.D.**

We could carry out exact comparisons and the basic arithmetic operations exactly (without error) on algebraic numbers; the same cannot be said for transcendental numbers. These ideas will be pursued in later chapters. To ensure robustness in rigid transformations, we must therefore be more selective in the set of angles for which we allow for rotation. Informally, call these the **admissible angles**. We note some candidates for admissible angles:

(A) Restrict θ to be **Pythagorean**, i.e., both $\sin\theta$ and $\cos\theta$ are rational numbers.

(B) Restrict θ to be **geodetic**, i.e., the square of $\sin\theta$ (equivalently $\cos\theta$) is rational.

(C) Restrict θ to be **commensurable with** π i.e., θ/π is a rational number. For short, we just say "commensurable".

(D) Restrict θ so that $\cos\theta$ is algebraic.

Canny, Donald and Ressler [3] introduced the computational problem of approximating arbitrary angles by Pythagorean angles. They did not explicitly define Pythagorean angles, but defined a **rational sine** to be[3] a rational solution s to the equation $s^2 + c^2 = 1$. In any case, the definition of Pythagorean angles

[3]They did not explicitly say whether c ought to be rational or not; but it is clear from their development that c is intended to be rational. But if c is not required to be rational, this implicitly defines a class of angles that is between Pythagorean and geodetic.

deserves comment. For instance, a less restrictive notion of admissibility is to require only one of $\sin\theta, \cos\theta$ to be rational. Thus non-Pythagorean angles such as $30°$ and $60°$ would become admissible. But $45°$ would still not be admissible. So we may further liberalize our notion of admissible to any angle θ where at least one of $\sin\theta, \cos\theta, \tan\theta$ is rational. This last class of admissible angles is clearly a subset of the geodetic angles. It is even a proper subset because $\theta = \arcsin(\sqrt{2}/\sqrt{5})$ is geodetic but $\sin\theta, \cos\theta, \tan\theta$ are not rational.

The definition of geodetic angles is from Conway et al [7]. They also call an angle **mixed geodetic** if it is a rational linear combination of geodetic angles. For instance, $45°$ is geodetic but $75° = (30+45)°$ and $1° = (1/30)30°$ are mixed geodetic. The set of mixed geodetic angles forms a vector space over $\mathbb{Q}$ They [7] constructed a basis for this vector space. From this, we achieve a classification of all rational linear relations among mixed geodetic angles. An elegant notation comes out of this theory: for any real number $x \in \mathbb{R}$, define the **angle** $\angle x$ as follows:

$$\angle x = \begin{cases} n\pi/2 = n90° & \text{if } x = n \in \mathbb{Z} \\ \angle n + \arcsin(\sqrt{x}) & \text{if } x = n + r, 0 < r < 1, n \in \mathbb{Z}. \end{cases} \tag{8}$$

Observe that if x is rational, then $\angle x$ is a geodetic angle. For instance, $\angle 0 = 0°, \angle(1/4) = 30°, \angle(1/2) = 45°, \angle(3/4) = 60°, \angle 1 = 90°$.

Commensurable Angles. Let us first show that *commensurable angles have algebraic cosines.* I.e., angles that are admissible under (B) is admissible under (D). To see this, we use **Chebyshev polynomials** of the first kind. These are integer polynomials $T_n(X)$ of degree n ($n \geq 0$) which are defined recursively by

$$T_{n+1}(X) = 2XT_n(X) - T_{n-1}(X),$$

starting from $T_0(X) = 1$ and $T_1(X) = X$. The next few polynomials are

$$\begin{aligned} &T_2(X) = 2X^2 - 1, \quad T_3(X) = X(4X^2 - 3), \quad T_4(X) = 8X^4 - 8X^2 + 1, \\ &T_5(X) = X(16X^4 - 20X^2 + 5). \end{aligned}$$

For any θ, we have

$$T_n(\cos\theta) = \cos n\theta. \tag{9}$$

If θ is commensurable, then $n\theta$ is a multiple of π for some n. From (9), we conclude that $T_n(\cos\theta) = \cos n\theta = \pm 1$. This relation proves that $\cos\theta$ is algebraic, as we claimed. But in fact, Jahnel [17] show the stronger statement that $2\cos\theta$ is an algebraic integer (Exercise).

Commensurable angles are clearly closed under addition, subtraction and multiplication by rationals. We claim that angles with algebraic cosines have the same closure properties. Closure under addition and subtraction from the usual formulas for $\cos(\theta + \theta')$ and $\cos(\theta - \theta')$. If $p, q \in \mathbb{Z}$, $q \neq 0$, and $\cos\theta$ is algebraic, then the equation $T_q(\cos(\theta/q)) = \cos\theta$ shows that $\cos(\theta/q)$ is algebraic; also $T_p(\cos(\theta/q)) = \cos((p/q)\theta)$ shows that $\cos((p/q)\theta)$ is algebraic.

Lemma 4 (Jahnel). *The following are equivalent:*
(i) The angle θ is commensurable and $\cos\theta$ is rational.
(ii) $2\cos\theta \in \{0, \pm 1, \pm 2\}$.

Proof. Clearly, (ii) implies (i). To show that (i) implies (ii), note that

$$2\cos 2\theta = (2\cos\theta)^2 - 2 \tag{10}$$

is just the identity $\cos 2\theta = 2\cos^2\theta - 1$ in disguise. Assume $2\cos\theta = m/n$ where $m, n \in \mathbb{Z}$, $\texttt{GCD}(m,n) = 1$ and $n \neq 0$. Substituting into (10), we get $2\cos 2\theta = (m^2 - 2n^2)/n^2$. Note that $\texttt{GCD}(m^2 - 2n^2, n^2) = 1$ for, if $p > 1$ and $p|(m^2 - 2n^2)$ and $p|n^2$, then $p|n$ and hence $p|m$, contradiction. Repeated application of the transformation $2\cos\theta \mapsto 2\cos 2\theta$ gives the sequence

$$2\cos 2\theta, 2\cos 4\theta, 2\cos 8\theta, \ldots, 2\cos 2k\theta, \ldots, \qquad (k \geq 1).$$

Moreover, each $2\cos 2k\theta = f_k(m,n)/n^{2^k}$, for some integer function $f_k(m,n)$ that is relatively prime to n. But, by the commensurablity of θ, there is some ℓ, j when $2\ell\theta = 2j\pi$ and hence $\cos 2\ell\theta = \cos 2j\pi = 1$. If

$n \neq \pm 1$, then the denominator of $\cos 2k\theta$ is growing arbitrarily large as $k \to \infty$. This contradicts the fact that $\cos 2\ell\theta = 1$ for some ℓ. Hence $n = \pm 1$. Therefore $2\cos\theta = \pm m$. This implies $m = 0, \pm 1, \pm 2$. Hence $\cos\theta \in \{0, \pm\frac{1}{2}, \pm 1\}$. **Q.E.D.**

Corollary 5. *Let θ be Pythogorean. Then θ is commensurable iff $\theta \notin \{90°, 0°, 180°, 60°, 120°\}$.*

Proof. Just note that the condition Lemma 4(ii) is equivalent to $\theta \notin \{90°, 0°, 180°, 60°, 120°\}$. **Q.E.D.**

For instance, this shows that arcsin(3/5) and arcsin(5/13) are incommensurable.

Theorem 6. *Let θ be incommensurable. Then*

$$\mathbb{Z}\theta := \{(n\theta) \,\mathbf{mod}\, 2\pi : n \in \mathbb{Z}\}$$

is a dense subset of $[0, 2\pi)$.

Proof. Given $\phi \in [0, 2\pi)$ and $N \in \mathbb{N}$, we must find $n \in \mathbb{Z}$ such that

$$|\phi - (n\theta) \,\mathbf{mod}\, 2\pi| < 2\pi/N. \tag{11}$$

By the pigeonhole principle, one of the following N intervals

$$[0, 2\pi/N), [2\pi/N, 4\pi/N), [4\pi/N, 6\pi/N), \ldots, [2(N-1)\pi/N, 2\pi)$$

must contain two members $a < b$ from the set $\{(i\theta) \,\mathbf{mod}\, 2\pi : i = 0, \ldots, N\}$ which has $N+1$ members. Then $0 < b - a < 2\pi/N$. We claim that $b - a \in \mathbb{Z}\theta$. To see this, let $a = i_a\theta - 2j_a\pi$ and $b = i_b\theta - 2j_b\pi$ where $i_a, i_b = 0, \ldots, N$ and $j_a, j_b \in \mathbb{Z}$. Then $b - a = (i_b - i_a)\theta - 2(j_b - j_a)\pi \in \mathbb{Z}\theta$. Let $M = \lceil 2\pi/(b-a) \rceil$. Clearly, $M > N$. So there is a unique $k \in \{0, 1, \ldots, M-1\}$ such that $k(b-a) \le \phi < (k+1)(b-a)$. Therefore, we may choose $n = k(i_b - i_a)$ to achieve the conclusion in (11). **Q.E.D.**

Thus, we can use the set $\mathbb{Z}\theta$ to arbitrarily approximate any desired angle ϕ to within any desired error bound.

Pythagorean Equation. In the following, we will study Pythagorean angles. Note that rotation by Pythagorean angles preserves the rationality of input points. This problem was studied by Canny, Donald and Ressler [3].

The Pythagorean equation is the equation

$$x^2 + y^2 = z^2 \tag{12}$$

to be solved in integers. Our goal is to characterize all solutions (x, y, z) to this equation. Since the signs of x, y, z do not matter, let us assume they are non-negative: $x \ge 0, y \ge 0, z \ge 0$. As the solutions (x, y, z) where $xyz = 0$ are easily characterized, we further assume positivity: $x \ge 1, y \ge 1, z \ge 1$. A solution (x, y, z) such that $\mathtt{GCD}(x, y, z) = 1$ is said to be **primitive**. We may restrict attention to primitive solutions.

It is now easy to see that we have a bijection between positive primitive solutions (x, y, z) to (12) and Pythagorean angles θ $(0 < \theta < \pi/2)$, as given by

$$(\sin\theta, \cos\theta) = (x/z, y/z). \tag{13}$$

Note that if d divides any two values in x, y, z then it divides the third. Hence primitivity is the same as saying that *any* two components in (x, y, z) are co-prime. In particular, there is at most one even number among x, y, z.

Claim: *Exactly one of x or y is even.* Note that the square of an odd number is congruent to 1 modulo 8, and the square of an even number is congruent to 0 modulo 4. If both x and y are odd, then $x^2 + y^2$ is congruent to 2 modulo 8. But this cannot represent the square of any number. This proves our claim.

By symmetry, we assume solutions (x, y, z) in which y is even but x, z are odd. Since $z + x$ and $z - x$ are even, let $m' := (z+x)/2, n' := (z-x)/2$. Note that $\mathtt{GCD}(m', n') = 1$ because if k divides m' and n' then k divides $m' + n' = z$ and k divides $m' - n' = x$, we get a contradiction. If $y = 2y'$ then $4y'^2 = (z+x)(z-x)$

and hence $y'^2 = m'n'$. This implies $m' = m^2$ and $n' = n^2$ for some m, n and $\texttt{GCD}(m, n) = 1$. We conclude the solution (x, y, z) we seek must have the form $(x, y, z) = (m' - n', 2y', m' + n')$, or

$$(x, y, z) = (m^2 - n^2, 2mn, m^2 + n^2) \tag{14}$$

for some co-prime $m, n \in \mathbb{N}$ with $m \geq n$. Conversely, it is easy to see that every co-prime $m, n \in \mathbb{N}$ gives rise to a primitive solution of the desired form. Thus all positive, primitive solutions to the Pythagorean equation is completely characterized by (14).

Pythagorean Angle Problem. Define the **length** of a rational number p/q in reduced form to be $\lg |q|$ (roughly the bit length of q in binary notation). Canny, Donald and Ressler [3] posed and solved the following problem: *on input θ, ε, a rational number $0 < s \leq 1$ such that $|\arcsin(s) - \theta| < \varepsilon$ and the length of s is as short as possible.* In practice, we might be happy if the length of s is within $O(1)$ bits of the shortest.

Since the angle θ may be hard to specify exactly in the input, we prefer the alternative specification: *given an input interval $I \subseteq [0, 1]$, to find a rational number $0 < s \leq 1$ such that s lies in the interval I, and the length of s is shortest.* We expect that $I = [\underline{s}, \overline{s}]$ will have rational endpoints. Note that the set of angles with rational sine does not include all angles with rational cosines. So an alternative formulation is: given $I \subseteq [0, 1]$, to to compute a rational number $0 < r < 1$ such that $r \in I$ or $\sqrt{1 - r^2} \in I$, and the length of r is shortest.

We now present one of their solutions: ...

In 3-dimensions, the generalization of the rotation matrix $R(\theta)$ is an orthonormal 3×3 matrix. Milenkovic and Milenkovic [22] considered the 3-dimensional analogue of Pythagorean angles, i.e., orthonormal matrices whose entries are rational numbers. They also show that such matrices are dense among orthonormal ones.

Exercises

Exercise 5.1: Given θ, suppose we computed some approximations $s \approx \sin\theta$ and $c \approx \cos\theta$. In particular, assume $s^2 + c^2 \neq 1$. The rotation of a point p by angle θ is given by $R \cdot p$ where $R = \begin{bmatrix} c & -s \\ s & c \end{bmatrix}$, and treating p as a vector and $R \cdot p$ as matrix-vector multiplication. Clearly, $\det R$ may not be 1 in general. Discuss real applications where this is a problem, and where this is not a problem. HINT: problems arise when you need to compute intersections of transformed and untransformed objects. $\diamondsuit$

Exercise 5.2: Let $\theta = \arcsin(3/5)$. We want to use the set $\mathbb{Z}\theta$ for approximation of angles. Given any real interval $I = [a, b] \subseteq [0, 2\pi)$, give an algorithm to find the smallest $|n|$ such $(n\theta) \,\mathbf{mod}\, 2\pi$ lies in I. $\diamondsuit$

Exercise 5.3: Improve Theorem 6.
(i) Can the $|n|$ in (11) be upper bounded as a function of N?
(ii) Can you show that $\mathbb{N}\theta := \{(n\theta) \,\mathbf{mod}\, 2\pi : n \in \mathbb{N}\}$ is dense? $\diamondsuit$

Exercise 5.4: Show that (B) and (C) are not comparable: there is a geodetic angle that is incommensurable, and there is a commensurable angle that is not geodetic. $\diamondsuit$

Exercise 5.5: [12, p.237] Show that $\arccos(1/3)$ is incommensurable. NOTE: This angle is about $70°31'44''$, the dihedral angle of a regular tetrahedron. $\diamondsuit$

Exercise 5.6: (Jörg Jahnel) Let θ be commensurable.
(i) $2\cos\theta$ is an algebraic integer, and all its conjugates have absolute value ≤ 2. HINT: This statement is a generalization of Lemma 4. Indeed, the proof imitates the proof of Lemma 4, except that we must decompose $2\cos\alpha$ into a product of distinct ideals, and argue that none of the ideals has negative exponent.
(ii) If x is a quadratic integer and $|x| < 2$ and $|\overline{x}| < 2$ then $x \in \{\pm\sqrt{2}, \pm\sqrt{3}, \pm\frac{1}{2} \pm \sqrt{5}\}$.
(iii) If $\cos\theta$ is a quadratic irrationality, then $2\cos\theta \in \{\pm\sqrt{2}, \pm\sqrt{3}, \pm\frac{1}{2} \pm \sqrt{5}\}$. Morover, all these values are achievable:

$$\begin{aligned}(\theta, 2\cos\theta) = {} & (45°, \sqrt{2}), (135°, -\sqrt{2}), (30°, \sqrt{3}), \\ & (150°, \sqrt{3}), (36°, \frac{1}{2} + \frac{1}{2}\sqrt{5}), (144°, -\frac{1}{2} - \frac{1}{2}\sqrt{5}), \\ & (72°, -\frac{1}{2} + \frac{1}{2}\sqrt{5}), (108°, \frac{1}{2} - \frac{1}{2}\sqrt{5}).\end{aligned}$$

(iv) If $2\cos\theta$ is a cubic irrationality then its minimal polynomial $x^3 + ax^2 + bx + c$ satisfies $|a| < 6, |b| < 12, |c| < 8$. Among these, there are exactly 4 irreducible polynomials: $p_1 : x^3 - x^2 - 2x + 1$, $p_2 : x^3 + x^2 - 2x - 1$, $p_3 : x^3 - 3x + 1$, $p_4 : x^3 - 3x - 1$.
(v) For each $i = 1, \ldots, 4$, determine the angles $\alpha_i, \beta_i, \gamma_i$ such that $2\cos\alpha_i, 2\cos\beta_i, 2\cos\gamma_i$ are the zeros of p_i. ♢

End Exercises

§6. Additional Notes

A clear expression of the goal of achieving robustness using fixed-precision arithmetic is stated in Fortune's paper [10].

"The challenge is to develop floating-point analyses of a wide variety of geometric algorithms. Such analyses should lead to efficient, reliable, easily-implemented geometric algorithms.

A general theory, easily applicable to an arbitrary algorithm, appears to be a distant goal. An important component of a general theory would be a way of transforming exact predicates into approximate predicates. Such a transformation must have two properties: first, the truth of approximate predicates has to be testable using floating-point arithmetic; second reasoning usng approximate predicates has to adequately mimic reasoning using exact predicates. An example is the transformation of the 2-d incircle predicate into the D predicate. This transformation satisfies the first property, since incircle can be used to test the D predicate; it does not entirely satisfy the second property.

Many computational geometers attacking robust geometric computation in the early 1990's assumed that the solution must come from fixed-precision floating point computational models. This view is well-summed up in Fortune's introduction: "Floating point arithmetic has numerous engineering advantages: it is well-supported... the Challenge is to demonstrate that a reliable implementation can result from the use of f.p. arithmetic."

Lindemann's theorem is more general than stated in the text. It says that the following expression can never hold:

$$c_1 e^{a_1} + c_2 e^{a_2} + \cdots + c_n e^{a_n} = 0 \tag{15}$$

where $n \geq 1$, the c_i's and a_i's are algebraic, the a_i's distinct and the c_i's are non-zero. This implies Hermite's theorem that e is transcendental: if e is algebraic, then there is an integer polynomial $A(X)$ such that $A(e) = 0$. But $A(e)$ has the form (15), contradiction. This also implies that π is transcendental because of Euler's identity $e^{\mathbf{i}\pi} + 1 = 0$ has the form (15). Finally, this result about π is proof that the ancient quest for squaring the circle is impossible. The quest ask for a ruler-and-compass construction of a square whose area is that of a circle. Without loss of generality, let the circle have unit radius and hence area π. If we can square the circle, we would have constructed a square whose side length is $\sqrt{\pi}$. But it is well-know that all such lengths arising from ruler-and-compass constructions are "constructible reals" (and hence algebraic). Thus, as a length of a square, $\sqrt{\pi}$ must be algebraic. Thus π is algebraic, contradiction.

For a collection of papers on implementation issues, we refer to a special 2000 issue of Algorithmica [9].

References

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, New York, 1983. Translated from German by Jon Rokne.

[2] G. Bohlender, C. Ullrich, J. W. von Gudenberg, and L. B. Rall. *Pascal-SC*, volume 17 of *Perspectives in Computing.* Academic Press, Boston-San Diego-New York, 1990.

[3] J. F. Canny, B. Donald, and E. Ressler. A rational rotation method for robust geometric algorithms. *Proc. 8th ACM Symp. on Computational Geometry*, pages 251–160, 1992. Berlin.

[4] E.-C. Chang, S. W. Choi, D. Kwon, H. Park, and C. Yap. Shortest paths for disc obstacles is computable. *Int'l. J. Comput. Geometry and Appl.*, 16(5-6):567–590, 2006. Special Issue of IJCGA on Geometric Constraints. (Eds. X.S. Gao and D. Michelucci).

[5] C. Clenshaw, F. Olver, and P. Turner. Level-index arithmetic: an introductory survey. In P. Turner, editor, *Numerical Analysis and Parallel Processing*, pages 95–168. Springer-Verlag, 1987. Lecture Notes in Mathematics, No.1397.

[6] C. Clenshaw and P. Turner. The symmetric level-index system. *IMA J. Num. Anal.*, 8:517–526, 1988.

[7] J. H. Conway, C. Radin, and L. Sadun. On angles whose squared trigonometric functions are rational. *Discrete and Computational Geometry*, 22:321–332, 1999.

[8] S. Figueroa. *A Rigorous Framework for Fully Supporting the IEEE Standard for Floating-Point Arithmetic in High-Level Programming Languages*. Ph.D. thesis, New York University, 1999.

[9] S. Fortune. Editorial: Special issue on implementation of geometric algorithms, 2000.

[10] S. J. Fortune. Numerical stability of algorithms for 2-d Delaunay triangulations. *Internat. J. Comput. Geom. Appl.*, 5(1):193–213, 1995.

[11] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[12] R. Hartshorne. *Geometry: Euclid and Beyond*. Springer, 2000.

[13] N. J. Higham. *Accuracy and stability of numerical algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, 1996.

[14] Holt, Matthews, Rosselet, and Cordy. *The Turing Programming Language*. Prentice-Hall, Englewood Cliffs, NJ, 1988.

[15] T. Hull, A. Abraham, M. Cohen, A. Curley, C. Hall, D. Penny, and J. Sawchuk. Numerical Turing. *ACM SIGNUM newsletter*, 20(3):26–34, July, 1985.

[16] T. Hull, M. Cohen, J. Sawchuk, and D. Wortman. Exception handling in scientific computing. *ACM Trans. on Math. Software*, 14(3):201–217, 1988.

[17] J. Jahnel. When does the (co)sine of a rational angle give a rational number?, 2004. From `http://http://www.uni-math.gwdg.de/jahnel/linkstopaperse.html`.

[18] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, Boston, 2nd edition edition, 1981.

[19] U. Kulisch, R. Lohner, and A. Facius, editors. *Perspectives on Enclosure Methods*. Springer-Verlag, Vienna, 2001.

[20] D. Lozier and F. Olver. Closure and precision in level-index arithmetic. *SIAM J. Numer. Anal.*, 29:1295–1302, 1990.

[21] S. Matsui and M. Iri. An overflow/underflow-free floating-point representation of numbers. *J. Inform. Process*, 4(3):123–133, 1981.

[22] V. Milenkovic and V. Milenkovic. Rational orthogonal approximations to orthogonal matrices. *Proc. 5th Canadian Conference on Computational Geometry*, pages 485–490, 1993. Waterloo.

[23] R. E. Moore. *Interval Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1966.

[24] T. Ottmann, G. Thiemt, and C. Ullrich. Numerical stability of geometric algorithms. In *Proc. 3rd ACM Sympos. Comput. Geom.*, pages 119–125, 1987.

[25] H. Ratschek and J. Rokne. Exact and optimal convex hulls in 2d. *Int'l. J. Comput. Geometry and Appl.*, 10(2):109–130, 2000.