The table maker's quantum search

Benjamin C. A. Morrison
Département de physique & Institut quantique
Université de Sherbrooke
Sherbrooke (QC), J1K 2R1
Canada

Stefanos Kourtis
Département de physique & Département d'informatique & Institut quantique
Université de Sherbrooke
Sherbrooke (QC), J1K 2R1
Canada

***Abstract*—We show that quantum search can be used to compute the hardness to round an elementary function, that is, to determine the minimum working precision required to compute the values of an elementary function correctly rounded to a target precision of n digits for all possible precision-n floating-point inputs in a given interval. For elementary functions f related to the exponential function, quantum search takes time $\tilde{O}(2^{n/2}\log(1/\delta))$ to return, with probability $1-\delta$, the hardness to round f over all n-bit floating-point inputs in a given binade. For periodic elementary functions in large binades, quantum search yields a modest asymptotic speedup over the best known classical algorithms and heuristics.**

1. Introduction

Quantum search algorithms based on Grover iteration [1], [2] promise to accelerate the solution of combinatorial search and optimization problems. When Grover's algorithm and its variants are used as standalone solvers, they generically achieve a quadratic speedup over exhaustive classical search. For example, hard Boolean satisfiability instances on n variables can be solved in time $\tilde{O}(2^{n/2})$, as opposed to $O(2^n)$ for exhaustive classical search. On the other hand, decades of research has yielded powerful classical algorithms that beat the scaling of standalone Grover search for most problems of practical interest by exploiting the problem structure. To obtain a quantum advantage for these problems, Grover iteration must be used as a subroutine inside suitable classical algorithms [2], [3], [4].

Are there problems of practical interest for which standalone Grover search beats the best known classical algorithm? For such a quantum advantage to occur, the search problem must be (approximately) devoid of structure that can be exploited by a classical algorithm. Relatively few such problems are known. One is symmetric-key cryptography [5], [6]. However, simply increasing the key length is sufficient to thwart attacks by standalone quantum search, due to the superpolynomial scaling of the algorithm. Another essentially structure-free problem is circuit satisfiability (CSAT) [7]. No known classical algorithm is meaningfully faster than exhaustive search in the worst case. In practice, industrial CSAT instances do have structure and it is found empirically that they can be solved with methods much faster than exhaustive search by mapping to equilogical Boolean satisfiability formulas [8], [9].

In this work, we study the problem of determining the working precision required to compute the correctly rounded value of an elementary function for finite-precision floating-point inputs. Correct rounding is of practical importance in computer arithmetic. Compounded rounding errors can give rise to large deviations between the computed and exact values of a quantity. Examples of adverse consequences of incorrect rounding include wildly erroneous index values of the Vancouver stock exchange in the early 1980s [10], [11] and the failure of a US Army Patriot array to intercept a Scud missile during the Gulf War [12], [13]. These mishaps highlight the importance of verifying that computer programs round correctly.

For transcendental functions, such verification is in general difficult: it is not known to which working precision $p > n$ we have to evaluate the function output for a given precision-n floating-point input before we can guarantee correct rounding to precision n [14], [15]. This problem is known as the table maker's dilemma [16]. The working precision required to guarantee correctly rounded evaluation of a function f for any precision-n floating-point input x in an interval I is called the hardness to round f in I, denoted $\mathrm{htr}_{f,I}(n)$. The difficulty in determining $\mathrm{htr}_{f,I}(n)$ arises from bad rounding cases, inputs for which the function, computed to working precision $p > n$, evaluates to a number that is too close to a decision boundary, so that we cannot unequivocally decide whether we should round to one or the other of two consecutive precision-n floating-point numbers. Although a $O(n^2)$ upper bound is known for the hardness to round the exponential and related functions, such as the

logarithm and trigonometric functions [15], [17], the hidden prefactors are enormous and the bound is not useful in practice. With the non-rigorous assumption that the digits of the value of a function are uniformly distributed random numbers, we can estimate that the hardness to round to n digits is roughly $2n$ [14]. Although this estimate matches empirical evidence well, it is not a worst-case bound.

No known polynomial-time algorithm computes the hardness to round a transcendental function exactly. The first nontrivial algorithm for computing the hardness to round a function f over precision-n binary floating-point inputs in a given binade scales as $\tilde{O}(2^{2n/3})$ [18]. This algorithm assumes a sufficient number of consecutive floating-point numbers in the binade to approximate f by a polynomial of degree 1. By increasing the approximating polynomial degree, the performance can be improved to $\tilde{O}(2^{n/2})$, at the expense of rendering the algorithm incomplete, i.e., a heuristic [19], [20]. The reason is that the algorithm of [19], [20] uses Coppersmith's method to find small roots of bivariate polynomials [21], a task for which the method is only known to be heuristic.

Both the aforementioned algorithms assume that a low-degree polynomial approximation is viable. For periodic elementary functions, this assumption does not always hold: consecutive floating-point numbers in large binades may even belong to different periods of the function. Ref. [22] overcomes this issue via a preprocessing step that uses the periodicity of the function to reorder inputs into an arithmetic progression. Then, the algorithm of [18] or the heuristics of [19], [20] can be applied, leading to runtimes $\tilde{O}(2^{4n/5})$ and $\tilde{O}(2^{(7-2\sqrt{10})n}) \approx \tilde{O}(2^{0.676n})$, respectively.

2. Contribution

The scalings of the best known classical algorithms and heuristics for computing the hardness to round cited above indicate that this task may be a good application of quantum search. The following theorem and algorithm formalize this intuition.

Theorem 1. *For any elementary function f evaluated at precision-n binary floating-point inputs $x \in I$ where $I = [2^e, 2^{e+1})$, Algorithm 1 returns $l = \min(\{\mathrm{htr}_{f,I}(n), p_{\max}\})$ with probability $1-\delta$ by making $O(2^{n/2} \log \frac{\lfloor \log p_{\max} \rfloor + 1}{\delta})$ calls to the oracles $\mathcal{O}_{n,e}^{f,\bullet}$. Furthermore, if $p_{\max} = O(\mathrm{poly}(n))$ is an upper bound for $\mathrm{htr}_{f,I}(n)$, Algorithm 1 returns $\mathrm{htr}_{f,I}(n)$ with probability $1-\delta$ in time $\tilde{O}(2^{n/2} \log(1/\delta))$.*

In Algorithm 1, the Grover operator $\mathcal{O}_{n,e}^{f,p}$ implements an arithmetic circuit that evaluates $f(x)$ up to p significant digits, coherently for all precision-n binary floating-point inputs x. A membership oracle then marks inputs x that yield bad rounding cases for $f(x)$ at precision n. We define $\mathcal{O}_{n,e}^{f,\bullet} = \{\mathcal{O}_{n,e}^{f,p}\}_{p=n+1}^{p_{\max}}$. If S is the set of bad rounding cases of a function f, we are searching for the minimum working precision p for which S is empty. Therefore, instead of searching for elements of a nonempty S – the typical use case for Grover search – we are instead asking a question about $|S|$. Whenever S is nonempty, measurement of the output state returns bad rounding cases.

The subroutine QSearch in Algorithm 1 is the quantum search algorithm [1], [23]. Given an oracle $\mathcal{O}_S$ that marks the elements of a set $S \subseteq \{0,1\}^n$ and $0 < \delta' < 1$, $\mathrm{QSearch}(n, \mathcal{O}_s, \delta')$ returns, with probability $1-\delta'$, 1 whenever $|S| > 0$ and 0 whenever $|S| = 0$. Algorithm 1 is then essentially a binary search over the working precision $p \in [n+1, p_{\max}]$ for the smallest integer $p^\star$ such that no marked item is found. If such a $p^\star$ exists in the searched interval, it is the hardness to round f. This approach works because the number of bad rounding cases is a non-increasing function of p. Finally, to achieve an overall failure rate δ, we need the failure rate of each of the $\lfloor \log p_{\max} \rfloor + 1$ runs of QSearch to scale as $\delta' = \delta/(\lfloor \log p_{\max} \rfloor + 1)$, thus obtaining the scaling cited in Theorem 1.

For functions related to the exponential function, such as the logarithm and trigonometric functions, a theorem of Nesterenko and Waldschmidt implies that we can take $p_{\max} = Cn^2$ for a sufficiently large and straightforwardly computable constant C [15], [17]. Therefore, for these functions, Algorithm 1 works in time $\tilde{O}(2^{n/2} \log(1/\delta))$ in any binade. We thus obtain a modest asymptotic speedup over the best known classical algorithms and heuristics for computing the hardness to round periodic elementary functions in large binades [22]. Moreover, by initializing the d-bit exponent of inputs x in a superposition of all possible values for a precision-n floating-point format and suitably modifying the Grover operator, we can compute the hardness to round a function over all possible 2^{n+d} floating-point inputs in the format in time $\tilde{O}(2^{(n+d)/2} \log(1/\delta))$.

Whether more efficient quantum algorithms for computing $\mathrm{htr}_{f,I}$ can be obtained by combining quantum search with suitable classical algorithms for this problem remains open. The fact that standalone quantum search outperforms the best known classical methods for a problem of some practical importance is nonetheless interesting in its own right.

3. Preliminaries and notation

Let $\mathbb{Z}, \mathbb{Q}, \mathbb{R}$, and $\mathbb{C}$ denote the sets of integer, rational, real, and complex numbers respectively. A number x is rational if $x = p/q$, with $p, q \in \mathbb{Z}$ and $q \neq 0$. A number $z \in \mathbb{C}$ is algebraic if there exists a nonzero polynomial P with integer coefficients such that $P(z) = 0$. A number $z \in \mathbb{C}$ is transcendental if it is not algebraic. Arithmetic operations are the operations of addition, subtraction, multiplication, and division.

In this work we will be dealing exclusively with univariate functions. A complex-valued function f is an algebraic function if there exists a bivariate polynomial P with integer coefficients such that for all z in the domain of f we have $P(z, f(z)) = 0$. A function is transcendental if it is not algebraic. A function is elementary if it can be defined by applying finitely many arithmetic operations and function

Algorithm 1: Quantum search for $\mathrm{htr}_{f,I}(n)$, $I = [2^e, 2^{e+1})$

Input: Precision n, oracle circuits $\mathcal{O}_{n,e}^{f,\bullet}$, upper bound $p_{\max}$, failure probability δ
Output: $\min(\{\mathrm{htr}_f(n), p_{\max}\})$
$l \leftarrow n+1$
$r \leftarrow p_{\max}$
$\delta' \leftarrow \delta/(\lfloor \log p_{\max} \rfloor + 1)$
while $l < r$ **do**
  $p \leftarrow l + \frac{\lfloor r-l \rfloor}{2}$
  $k \leftarrow \mathrm{QSearch}(n, \mathcal{O}_{n,e}^{f,p}, \delta')$
  if $k > 0$ **then**
    $l \leftarrow p+1$
  else
    $r \leftarrow p$
  end
end
Return: l

compositions to polynomial, exponential, logarithm, and constant functions. Elementary functions have derivatives of all orders that are algorithmically computable. For simplicity, below we consider exclusively real numbers and real-valued functions, but our results generalize to the complex plane.

A binary floating-point number x is a triple (s, m, e) such that

$$x = (-1)^s \cdot m \cdot 2^e \,, \tag{1}$$

where $s \in \{0, 1\}$ is the sign bit, $m \geq 0$ is the significand or mantissa, and $e \in \mathbb{Z}$ is the exponent. The precision of x is the number of variable significant digits of m. The possible values that a precision-n floating-point number can take depend on the semantics chosen for representing m. The binary floating-point formats of the IEEE 754 standard [24] define the significand of normal numbers as

$$m = 1.m_1 m_2 \ldots m_n = \sum_{j=0}^{n} m_j 2^{-j} \,, \tag{2}$$

that is, we fix the implicit integer bit $m_0 = 1$ and use n bits to define the fractional part of m, called the fraction, so that $m \in [1, 2)$. In the following, when we refer to "the n-bit significand of x", we mean that $m(x)$ has n variable digits after the implicit one. For fixed e and $s = 0$, there are 2^n precision-n binary floating-point numbers in the binade $[2^e, 2^{e+1})$. If e is also variable and defined as a d-bit integer, x can take 2^{n+d+1} values[1]. For convenience, we define the functions $s(x)$, $m(x)$, and $e(x)$ that return the sign bit, significand, and exponent, respectively, of a floating-point number x.

From Eqs. (1) and (2) we see that binary floating-point numbers are rational numbers whose denominator is a power of 2. For rational numbers whose denominator is not a power of 2, the expansion (2) may not terminate for any finite n (take, for example, the number 4/3). The same holds for irrational numbers. In floating-point arithmetic, for any number x whose binary floating-point representation does not terminate, we first compute sufficiently many significand digits to obtain an approximation $\tilde{x}$ and then round $\tilde{x}$ to the target precision, that is, we replace the result with a nearby floating-point number. Which nearby floating-point number we pick depends on the rounding mode chosen. Common rounding modes are rounding up/down, which returns the closest floating-point number greater/smaller than the input, and rounding to nearest, which returns the floating-point number closest to the input. In what follows, we discuss only the round-to-nearest mode for illustration purposes, but all our results translate straightforwardly to other rounding modes.

We now describe the "round to nearest, ties to even" rounding mode, which is the most commonly used in binary floating-point arithmetic. For a precision-p floating-point number x, let $\circ_n(x)$ denote its value rounded to the nearest precision-n floating-point number, with $n < p$. We call p the working precision and n the target precision. To compute $\circ_n(x)$, we examine the $p-n+1$ least significant digits of the significand $m = m(x)$, starting with the guard digit m_{n+1}:

1) If $m_{n+1} = 0$, then $\circ_n(x) = (-1)^{s(x)} \cdot 1.m_1 \ldots m_n \cdot 2^e$, that is, we simply truncate the $p-n$ least significant digits of the significand (i.e., round toward zero).
2) If $m_{n+1} = 1$, then:
   a) If $m_j = 1$ for any $j > n+1$, then $\circ_n(x) = (-1)^{s(x)} \cdot (1.m_1 \ldots m_n + 2^{-n}) \cdot 2^e$ (i.e., round away from zero).
   b) If $m_j = 0$ for all $j > n+1$, then x is exactly halfway between two precision-n floating-point numbers and we need to break the tie somehow. The "ties to even" rule is:
      i) If $m_n = 0$, we round toward zero: $\circ_n(x) = (-1)^{s(x)} \cdot 1.m_1 \ldots m_n \cdot 2^e$.
      ii) If $m_n = 1$, we round away from zero: $\circ_n(x) = (-1)^{s(x)} \cdot (1.m_1 \ldots m_n + 2^{-n}) \cdot 2^e$.

Note that if $p = n+1$, then $m_j = 0$ implicitly for all $j > n+1$.

As mentioned previously, for numbers x whose binary floating-point representation does not terminate, we instead use a computation method that returns an approximation $\tilde{x}$ with working precision p, so that $\tilde{x}$ is guaranteed to be within 2^{-p} of x. We say that this method implements correct rounding if $\circ_n(\tilde{x}) = \circ_n(x)$.

The same reasoning holds for transcendental functions f, since the exact value $f(x) \in \mathbb{R}$ is not representable as a finite-precision floating point number in general. In floating-point arithmetic, we instead evaluate a function $\tilde{f}$ whose value $\tilde{f}(x)$ is a precision-p floating-point number within $2^{e(f(x))-p}$ of $f(x)$, where $p > n$. This is done

1. In the IEEE 754 standard, some of these values are reserved for special purposes: the values $e = m = 0$ encode the signed zeros ± 0; $e = 0, m \neq 0$ the subnormal numbers; $e = 2^d - 1, m = 0$ the positive and negative infinity $\pm\infty$; and $e = 2^d - 1, m \neq 0$ the Not-a-Number value NaN.

by summing a convergent series for f, such as a Taylor or arithmetic-geometric mean series [25]. A method that evaluates f via $\tilde{f}$ implements correct rounding to precision n if $\circ_n(f(x)) = \circ_n(\tilde{f}(x))$ for all precision-n floating-point inputs x.

It is not known a priori to what precision we have to evaluate $\tilde{f}$ to ascertain correct rounding to precision n [14]. Consider the following example. Suppose that we compute the precision-p floating-point number $\tilde{f}(x)$, with $\tilde{m} = m(\tilde{f}(x))$ and $e(\tilde{f}(x)) = e(f(x)) = 0$, to obtain

$$\tilde{m} = 1.\tilde{m}_1\tilde{m}_2\ldots\tilde{m}_{n-1}110\ldots0\,. \tag{3}$$

We see that the digit $\tilde{m}_n = 1$ is followed by the guard digit $\tilde{m}_{n+1} = 1$, followed by $p-n-1$ zeros. For $\tilde{f}(x)$ to be correctly rounded to precision n, we must ascertain that $\tilde{m}_j = m_j$ for $j = 0,\ldots,n$, where $m = m(f(x))$ is the infinitely precise significand of $f(x)$. This is not necessarily the case: $\tilde{m}$ and $\tilde{m}' = 1.\tilde{m}_1\tilde{m}_2\ldots\tilde{m}_{n-1}101\ldots1$ define floating-point numbers that are both within 2^{-p} of $f(x)$. When rounding to nearest with ties to even, $\tilde{m}_n$ would be incremented to obtain $\tilde{m}_n = 0$, while $\tilde{m}'_n$ would be left as $\tilde{m}'_n = 1$. Therefore, we cannot decide whether $\tilde{m}_n = m_n$, which, in turn, means that we cannot ascertain whether $\circ_n(f(x)) = \circ_n(\tilde{f}(x))$. Now, suppose we compute $\tilde{f}(x)$ up to higher precision of $p+1$ bits, that is, we approximate $f(x)$ to within $2^{-(p+1)}$ by summing more terms in the series expansion of f. If we obtain

$$\tilde{m} = 1.\tilde{m}_1\tilde{m}_2\ldots\tilde{m}_{n-1}110\ldots01\,, \tag{4}$$

then the only precision-$(p+1)$ floating-point numbers within $2^{-(p+1)}$ of $f(x)$ have significands that agree with the infinitely precise significand of $f(x)$ on up to n significant digits. Therefore, we can ascertain that $\circ_n(f(x)) = \circ_n(\tilde{f}(x))$.

For bad rounding cases x, the last $p-n-1$ digits of $m(\tilde{f}(x))$ evaluate to $00\ldots0$ or $11\ldots1$. Such long runs of 0s or 1s can arise in two cases. The first (trivial) case arises when $f(x)$ evaluates exactly to a floating-point number of precision smaller than p, such as the function $\log_2$ evaluated at integer powers of 2. For most common transcendental functions, these exceptional inputs are well-known and can be efficiently specified and excluded [14]. The second case is the one illustrated in the example above, when there are more than one precision-p floating-point numbers within 2^{-p} of $f(x)$. A concrete example is the single-precision (binary32 format of IEEE 754) input

$$x = 1.00111011101100100011010_2 \cdot 2^{-1} \tag{5}$$

with 24-bit significand, for which the function $f = 2\sin$ evaluates to

$$\begin{aligned} 2\sin(x) = 1.00011011101000110110010 \\ 11111111111111111111000\ldots_2\,, \end{aligned} \tag{6}$$

that is, the 24th digit of $m(f(x))$ is 1, followed by the guard digit 0, followed by 21 consecutive 1s. Therefore, to guarantee correct rounding for this input, we have to approximate $f(x)$ up to a precision of at least $24+21 = 45$ bits.

Given a function f and target precision n, up to what precision p do we need to evaluate $\tilde{f}(x)$ in order to be sure that $\circ_n(f(x)) = \circ_n(\tilde{f}(x))$? This question is known as the table maker's dilemma [16]. The smallest p for which $\circ_n(f(x)) = \circ_n(\tilde{f}(x))$ for all precision-n floating-point inputs $x \in I$, where I is some interval, is called the hardness to round f in I and denoted $\mathrm{htr}_{f,I}(n)$. We denote the hardness to round f over all precision-n floating-point inputs $\mathrm{htr}_f(n)$.

4. Grover search and membership oracle

For completeness, we give an overview of Grover search [1] (see [26] for a complete introduction). Given a search problem over n Boolean variables, the Grover algorithm in its simplest form is the repeated application of the unitary operator $\mathcal{G} = \mathcal{D}\mathcal{O}$ to the n-qubit quantum state $|+\rangle^{\otimes n} = 2^{-n/2}\sum_{k=0}^{2^n-1}|k\rangle$, where k runs over all n-bit strings. The membership oracle $\mathcal{O}$ acts as $\mathcal{O}|k\rangle = -|k\rangle$ when k is a solution and as $\mathcal{O}|k\rangle = |k\rangle$ otherwise. We describe the circuit construction of a membership oracle for bad rounding cases below. The diffusion operator $\mathcal{D} = 2(|+\rangle\langle+|)^{\otimes n} - I$, where I is the identity operator, is constructed with $O(n)$ Hadamard and classical controlled gates. After r iterations, the algorithm produces the state

$$\mathcal{G}^r|+\rangle^{\otimes n} = \cos\left(\frac{2r+1}{2}\theta\right)|\bar{S}\rangle + \sin\left(\frac{2r+1}{2}\theta\right)|S\rangle\,, \tag{7}$$

where $|S\rangle = |S|^{-1/2}\sum_{k\in S}|k\rangle$ is the equal superposition of all states in the solution set S, $|\bar{S}\rangle = |2^n - S|^{-1/2}\sum_{k\notin S}|k\rangle$ the equal superposition of non-solutions, and $\cos\theta/2 = \sqrt{(2^n-|S|)/2^n}$. Measurement of the resulting state yields a solution to the search problem with probability $\sin^2\left(\frac{2r+1}{2}\theta\right)$. When $\mathcal{O}$ can be expressed as a poly-sized quantum circuit, the worst-case runtime for either returning a solution, if there is one, or asserting that there are no solutions with high probability is $\tilde{O}(2^{n/2})$ [23].

The membership oracles for Algorithm 1 are composed of two circuits. The first one is a reversible arithmetic circuit that, on input x a precision-n floating-point number with exponent e, computes a precision-p approximation $\tilde{f}(x)$ to $f(x)$, correct up to additive error 2^{-p}, with $p > n$. This is a standard task in arbitrary-precision arithmetic and can be implemented with multiple methods in time polynomial in p [25], with the (asymptotically) fastest known being the arithmetic-geometric mean method that runs in time $O(M(p)\log p)$, where $M(p)$ is the cost of multiplying two n-bit integers. Turning any of these methods into a reversible circuit incurs at most polynomial overhead in time and ancilla bits [27], [28]. After computation, any ancilla bits used are returned to their initial state by uncomputation. Here we are interested in comparisons with classical methods whose scaling is typically evaluated in a fixed binade. We therefore choose to hard-code the input exponent e into the arithmetic circuits $C^{f,p}_{n,e}$ and give only the n-bit significand $m(x)$ as input. Similarly, although it implicitly evaluates $e(\tilde{f}(x))$, the circuit outputs only the n-bit significand $m(\tilde{f}(x))$, since

that is all we need to detect bad rounding cases. To further lighten the notation, we also omit sign bits and the implicit integer bit of the significands. With this convention, the initial state for the quantum search algorithm is the equal superposition of all 2^n precision-n significands, prepared as $|+\rangle^{\otimes n}$.

The second circuit $C^{\text{bad}}_{n,p}$ tests if the last bits of $\tilde{f}(x)$ form a long string of 0s or 1s, corresponding to a bad rounding case. Specifically, the strings of interest are $110\ldots0$ or $101\ldots1$ in the case of rounding to nearest with ties to even[2]. A circuit for a given rounding mode tests whether the last $p-n+1$ bits of $\tilde{f}(x)$ are in the corresponding state and, in the affirmative, flags the state x by conditionally flipping a register, which in this case is a single qubit prepared in the state $|-\rangle$. The conditional operation can be implemented with $O(n)$ Toffoli gates and $O(n)$ ancilla qubits and is uncomputed after the state has been flagged, returning all ancillas to their initial state.

The action of the circuits $C^{f,p}_{n,e}$ and $C^{\text{bad}}_{n,p}$ is then

$$C^{f,p}_{n,e}\,|m(x)\rangle\,|0\rangle^{\otimes p}\,|0\rangle^{\otimes w} = |m(x)\rangle\,|m(\tilde{f}(x))\rangle\,|0\rangle^{\otimes w}\;, \quad (8)$$

$$C^{\text{bad}}_{n,p}\,|m(x)\rangle\,|m(\tilde{f}(x))\rangle\,|0\rangle^{\otimes w}\,|-\rangle = \\ = (-1)^{b_n(\tilde{f}(x))}\,|m(x)\rangle\,|m(\tilde{f}(x))\rangle\,|0\rangle^{\otimes w}\,|-\rangle\;, \quad (9)$$

where w is the number of ancilla bits required by $C^{f,p}_{n,e}$ and $C^{\text{bad}}_{n,p}$, and $b_n(y)$ is an indicator function that evaluates to 1 whenever the floating-point number y is a bad case for rounding to precision n.

With the above, we can define the membership oracle for bad rounding cases as

$$\mathcal{O}^{f,p}_{n,e} = C^{f,p\mathsf{T}}_{n,e}\,C^{\text{bad}}_{n,p}\,C^{f,p}_{n,e}\,. \quad (10)$$

The action of the oracle is thus

$$\mathcal{O}^{f,p}_{n,e}\,|m(x)\rangle\,|0\rangle^{\otimes p}\,|0\rangle^{\otimes w}\,|-\rangle = \\ = (-1)^{b_n(\tilde{f}(x))}\,|m(x)\rangle\,|0\rangle^{\otimes p}\,|0\rangle^{\otimes w}\,|-\rangle\;, \quad (11)$$

as required for the Grover iteration.

There are some subtleties to address. First, some functions may evaluate exactly to a precision-n floating-point number for some inputs. In such cases, the significand $m(\tilde{f}(x))$ may terminate in $110\ldots0$ (or a string relevant for detecting bad cases for another rounding mode), even though x is not a bad rounding case. Such numbers are exceptional and can be dealt with efficiently. For example, for trigonometric functions $\sin$ and $\cos$, Niven's theorem [29], [30] implies that the only finite-precision floating-point x for which these functions evaluate to a rational number is $x = 0$. Similar results hold for other elementary functions, such as $\ln$ and $\exp$. Some elementary functions, such as power, root, and logarithm functions, evaluate exactly to floating-point numbers with fewer than n significant digits for many inputs (e.g., integers), but these are known and can be efficiently detected and excluded from oracle marking with additional circuit logic [14]. We assume that this logic is included in $C^{f,p}_{n,e}$. Second, we may have $e(x) \neq e(\tilde{f}(x))$, although both exponents have the same length d and format. An evaluation of $f(x)$ that yields $e(\tilde{f}(x))$ outside the range representable by a d-bit integer leads to an under- or overflow. Such cases can be efficiently detected by exception handling in $C^{f,p}_{n,e}$, as in the IEEE floating-point standard. Because of this, and since underflows and overflows do not constitute bad rounding cases, we ignore them in our discussion. For the same reason, we also ignore inputs corresponding to subnormal numbers, infinities, and NaN.

We conclude that, even taking into account the subtleties of the preceding paragraph, the circuit $C^{f,p}_{n,e}$ has size polynomial in n and p, and can be constructed in time polynomial in n and p.

5. Proof of Theorem 1

Let $\mathcal{O}^{f,p}_{n,e}$ be constructed as described in Section 4. As is shown in [23], a single run of Grover search with at most $O(2^{n/2})$ iterations is sufficient to determine, with success probability $O(1)$, whether the solution set is empty. By making use of the Chernoff-Hoeffding bound, one can show that the success probability can be boosted to $1-\delta'$ by repeating the Grover search $O(\log \delta'^{-1})$ times and taking a majority vote over successful runs. That a run is successful can be determined in polynomial time by checking whether the measurement outcome is indeed a bad rounding case. We denote this repeated Grover search as QSearch in Algorithm 1. Each invocation of QSearch therefore makes at most $O(2^{n/2}\log \delta'^{-1})$ calls to an operator in $\mathcal{O}^{f,\bullet}_{n,e} = \{\mathcal{O}^{f,p}_{n,e}\}_{p=n+1}^{p_{\max}}$. Since Algorithm 1 is a binary search, it makes at most $\lfloor \log p_{\max} \rfloor + 1$ iterations. For the algorithm to succeed with probability at least $1-\delta$, we must set a sufficiently small δ' for each iteration. Setting $\delta' \leq \delta/(\lfloor \log p_{\max} \rfloor + 1)$ suffices:

$$\delta \geq \delta'(\lfloor \log p_{\max} \rfloor + 1) \;\Rightarrow \quad (12)$$

$$\Rightarrow 1-\delta \leq 1-\delta'(\lfloor \log p_{\max} \rfloor + 1) \leq (1-\delta')^{\lfloor \log p_{\max} \rfloor + 1}\,, \quad (13)$$

where the last step is due to the Bernoulli inequality. Finally, if $p_{\max} = O(\text{poly}(n))$ is an upper bound for $\text{htr}_{f,I}(n)$, every operator in $\mathcal{O}^{f,\bullet}_{n,e}$ is implemented by a circuit of size $O(\text{poly}(n))$ and the overall runtime is $\tilde{O}(2^{n/2}\log(1/\delta))$. □

6. Oracle Resource Estimation

To translate this asymptotic advantage into a demonstration of practical quantum utility, one promising target would be the rounding of $\sin(x)$ and $\cos(x)$ in the IEEE 754 standard binary64 format. The bad cases for these functions for $x > 2^{11}$ were until this spring [31] out of reach for classical methods [15]. Accordingly, we will consider the cost of implementing the required arithmetic circuits for $C^{\sin,106}_{53,e}$ and the resulting cost of implementing Algorithm 1.

The binary splitting algorithm [25], [32] can be used to calculate both $\sin$ and $\cos$ at once. Its asymptotic complexity is less than the algebraic-geometric mean algorithm,

2. When rounding down or up, the bad cases terminate in $00\ldots0$ or $11\ldots1$.

Circuit	$M(128)$	$M(64)$	$M(32)$	$M(16)$	$C_{53,e}^{\sin,106}$
Schoolbook (gate count)	65152	16192	4000	976	4.3×10^6
Karatsuba (gate count)	31272	9912	3048	888	2.3×10^6
Parallel Karatsuba (depth)	3112	1576	808	424	2.8×10^4

TABLE 1. TOFFOLI GATE COUNTS AND PARALLELIZED TOFFOLI DEPTHS OF 2^k-BIT MULTIPLICATION CIRCUITS AND THE $C_{53,e}^{\sin,106}$ ORACLE.

but below the 10^6 bit range its smaller constants make it superior. Achieving 106-bit precision requires partitioning the angle $x \mod 2\pi$ into eight substrings y_j with mantissas of length $1, 2, ..., 64, 128$. The binary splitting algorithm can be applied separately (and in parallel) to calculate the sine and cosine of each substring, and the results combined.

Evaluating $\sin y_j$ and $\cos y_j$ to the desired precision requires taking no more than $\sqrt{256} = 16$ terms from the power series. Summing these terms via binary splitting's divide-and-conquer method requires 45 multiplications and 15 additions, but only one division. However, only three of these multiplications and one of these additions is at the highest bit width. In total, the cost is $3M(128) + 6M(64) + 12M(32) + 24M(16)$ plus a negligible addition cost. Almost all of these operations can be implemented in parallel, so the circuit depth required for one evaluation is only $M(128) + M(64) + M(32) + M(16)$.

Combining the results to obtain the final $\sin(x)$ and $\cos(x)$ requires an additional $21M(128)$ plus again a negligible addition cost. The total Toffoli cost is $45M(128) + 48M(64) + 96M(32) + 192M(16)$; again, this can be parallelized heavily to $8M(128) + M(64) + M(32) + M(16)$.

With the schoolbook method of multiplication by shifted additions, $M(n) = 4n^2 - 3n$, and we find the gate counts given in the first row of Tab. 1, with the $C_{53,e}^{\sin,106}$ oracle subcircuit requiring a total of approximately 4.3×10^6 Toffoli gates.

Especially for the larger multiplication circuits, the schoolbook method is not the fastest available. Using reversible Karatsuba multiplication [33], we have $M(2^k) = 3M(2^{k-1}) + 12 \cdot 2^k$. If we implement 8-bit multiplication using the schoolbook method, $M(8) = 232$, yielding the gate counts in the second row of Tab. 1. Thus the total number of Toffoli gates required by $C_{53,e}^{\sin,106}$ is approximately 2.3×10^6. However, by running each recursive Karatsuba multiplication subcircuit in parallel, the circuit depth required is only $M(2^k) = M(2^{k-1}) + 12 \cdot 2^k$. Using 8-bit schoolbook multiplication again yields the Toffoli depths in the third row of Tab. 1. The depth could be reduced further by using schoolbook multiplication only at the 4-bit level, but this would increase the Toffoli count. The total Toffoli depth of the parallel $C_{53,e}^{\sin,106}$ is then approximately 2.8×10^4.

Assuming an at least partially fault-tolerant quantum computer—as will likely be necessary for gate counts even in the millions—the main limit on wallclock time will be the rate at which syndromes can be extracted and decoded. With the very generous assumption of a single-shot error correcting code [34] and factories directly distilling magic Toffoli states from T-states [35], the number of measurement cycles per Toffoli gate can be as low as 1. On superconducting hardware, a measurement and repreparation can require as little as hundreds of nanoseconds [36]; thus the wallclock time per oracle call could be as low as tens of milliseconds.

Algorithm 1 requires a number of oracle calls approximately equal to $\frac{\pi}{4} 2^{n/2} \approx 7.45 \times 10^7$. To match the wallclock time of 4 hours per binade [31], the cost per oracle call would need to be reduced by two orders of magnitude, to the hundreds of microseconds regime. On the other hand, searching all 1024 binades simultaneously would require only $\frac{\pi}{4} 2^{(n+10)/2} \approx 2.39 \times 10^9$ oracle calls; to match the total wallclock time of two weeks [31], the oracle would only need to run in hundreds of milliseconds, which would be possible for the architecture sketched above. This of course does not actually imply equal practical utility to the classical algorithm. First, the simultaneous search solves only the decision-problem version of hardness to round, which is equally difficult but far less useful. Second, the implementation described above would require a hypothetical quantum computer significantly larger and more advanced than any currently in existence, whereas the wallclock times stated for Lefèvre, Ly, and Zimmerman's algorithm are based on execution on extant classical computers [31]. Still, it is valuable to consider what capabilities a quantum computer would require to make the table maker's quantum search a candidate for quantum advantage experiments.

7. Discussion

Algorithm 1 with its $\tilde{O}(2^{n/2} \log(1/\delta))$ runtime achieves a modest asymptotic speedup compared to the best known classical algorithm or heuristic for computing the hardness to round periodic elementary functions in large binades, whose runtimes scale as $\tilde{O}(2^{4n/5})$ and $\tilde{O}(2^{(7-2\sqrt{10})n}) \approx \tilde{O}(2^{0.676n})$, respectively [22]. While there has been a long line of work on quantum arithmetic [37], to our knowledge, this is the first work that demonstrates an asymptotic quantum speedup in a computer arithmetic task. What is more interesting is that the speedup comes from standalone quantum search, which, as discussed in the introduction, is rather rare.

Algorithm 1 has a number of shortcomings that can perhaps be overcome. First, it is exponential in n. The table maker's dilemma is related to factoring [19]. Could Shor's [38] or Regev's [39] algorithms be adapted to this problem?

Second, the $\tilde{O}$ notation hides potentially large prefactors related to the complexity of the arithmetic circuits used. Any implementation would need to optimize the circuits considerably to fit them in the short coherence window of foreseeable quantum computers. On the other hand, for a

given function, binade, and target precision, the hardness to round needs to be computed only once, thus amortizing the implementation cost. Given that, to date, we do not know the hardness to round for trigonometric functions in binades higher than 2^{11} for the binary64 format of the IEEE 754 standard [15], it is conceivable that Algorithm 1 may be of some use eventually. In a practical scenario, Algorithm 1 can be sped up by focusing the search in the range $[n+1, 2n]$ first, since the hardness to round is very likely in that range. Results in quantum arithmetic [37] could also be exploited to reduce the number of ancilla qubits required.

Third, while the algorithm can be generalized to determine the hardness to round over all binades simultaneously (instead of fixing the exponent, initialize a superposition of all possible exponent values), the arithmetic circuits for function evaluation will be more complex in this case, since different argument reductions are favorable in large and small binades [25] and hence the suitable logic will have to be built into the circuits. Still, the arithmetic circuits will remain poly-sized.

Finally, whether combining quantum search with classical algorithms could yield practical methods for estimating the hardness to round is an open question.

References

[1] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 212–219. [Online]. Available: https://doi.org/10.1145/237814.237866

[2] G. Brassard, P. Høyer, M. Mosca, and A. Tapp, "Quantum amplitude amplification and estimation," p. 53–74, 2002. [Online]. Available: http://dx.doi.org/10.1090/conm/305/05215

[3] A. Ambainis, "Quantum search algorithms," *SIGACT News*, vol. 35, no. 2, p. 22–35, Jun. 2004. [Online]. Available: https://doi.org/10.1145/992287.992296

[4] M. Fürer, "Solving np-complete problems with quantum search," in *LATIN 2008: Theoretical Informatics*, E. S. Laber, C. Bornstein, L. T. Nogueira, and L. Faria, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 784–792.

[5] D. J. Bernstein, *Introduction to post-quantum cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–14. [Online]. Available: https://doi.org/10.1007/978-3-540-88702-7_1

[6] M. Grassl, B. Langenberg, M. Roetteler, and R. Steinwandt, "Applying grover's algorithm to aes: Quantum resource estimates," in *Post-Quantum Cryptography*, T. Takagi, Ed. Cham: Springer International Publishing, 2016, pp. 29–43.

[7] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[8] N. Amla, X. Du, A. Kuehlmann, R. P. Kurshan, and K. L. McMillan, "An analysis of sat-based model checking techniques in an industrial environment," in *Proceedings of the 13 IFIP WG 10.5 International Conference on Correct Hardware Design and Verification Methods*, ser. CHARME'05. Berlin, Heidelberg: Springer-Verlag, 2005, p. 254–268. [Online]. Available: https://doi.org/10.1007/11560548_20

[9] Y. Vizel, G. Weissenbacher, and S. Malik, "Boolean satisfiability solvers and their applications in model checking," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 2021–2035, 2015.

[10] B. D. McCullough and H. D. Vinod, "The numerical reliability of econometric software," *Journal of Economic Literature*, vol. 37, no. 2, p. 633–665, June 1999. [Online]. Available: https://www.aeaweb.org/articles?id=10.1257/jel.37.2.633

[11] Y. Nievergelt, "Rounding errors to knock your stocks off," *Mathematics Magazine*, vol. 73, no. 1, pp. 47–48, 2000.

[12] US General Accounting Office, "Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia," *Report Number: GAO/IMTEC-92-26*, 1992.

[13] E. Schmitt, "After the war; army is blaming patriot's computer for failure to stop the dhahran scud." *The New York Times*, pp. NA–NA, 1991.

[14] J.-M. Muller, N. Brunie, F. de Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefèvre, G. Melquiond, N. Revol, and S. Torres, *Evaluating Floating-Point Elementary Functions*. Cham: Springer International Publishing, 2018, pp. 375–433. [Online]. Available: https://doi.org/10.1007/978-3-319-76526-6_10

[15] P. Brisebarre, Nicolas, G. Hanrot, J.-M. Muller, and P. Zimmermann, "Correctly-rounded evaluation of a function: Why, how, and at what cost?" *ACM Comput. Surv.*, vol. 58, no. 1, Sep. 2025. [Online]. Available: https://doi.org/10.1145/3747840

[16] W. Kahan, "A logarithm too clever by half," 2004. [Online]. Available: https://people.eecs.berkeley.edu/~wkahan/LOG10HAF.TXT

[17] Y. Nesterenko and M. Waldschmidt, "On the approximation of the values of exponential function and logarithm by algebraic numbers," 2000. [Online]. Available: https://arxiv.org/abs/math/0002047

[18] V. Lefevre, "New results on the distance between a segment and z ² . application to the exact rounding," in *17th IEEE Symposium on Computer Arithmetic (ARITH'05)*, 2005, pp. 68–75.

[19] D. Stehlé, "On the randomness of bits generated by sufficiently smooth functions," in *Algorithmic Number Theory*, F. Hess, S. Pauli, and M. Pohst, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 257–274.

[20] D. Stehle, V. Lefevre, and P. Zimmermann, "Searching worst cases of a one-variable function using lattice reduction," *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 340–346, 2005.

[21] D. Coppersmith, "Finding a small root of a bivariate integer equation; factoring with high bits known," in *Advances in Cryptology — EUROCRYPT '96*, U. Maurer, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 178–189.

[22] G. Hanrot, V. Lefevre, D. Stehle, and P. Zimmermann, "Worst cases of a periodic function for large arguments," in *18th IEEE Symposium on Computer Arithmetic (ARITH '07)*, 2007, pp. 133–140.

[23] M. Boyer, G. Brassard, P. Høyer, and A. Tapp, "Tight bounds on quantum searching," *Fortschritte der Physik*, vol. 46, no. 4–5, p. 493–505, Jun. 1998. [Online]. Available: http://dx.doi.org/10.1002/(SICI)1521-3978(199806)46:4/5⟨493::AID-PROP493⟩3.0.CO;2-P

[24] M. S. Committee, "IEEE standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

[25] R. P. Brent and P. Zimmermann, *Elementary and special function evaluation*, ser. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2010, p. 125–184.

[26] M. A. Nielsen and I. L. Chuang, *Quantum search algorithms*. Cambridge University Press, 2010, p. 248–276.

[27] C. H. Bennett, "Time/space trade-offs for reversible computation," *SIAM Journal on Computing*, vol. 18, no. 4, pp. 766–776, 1989. [Online]. Available: https://doi.org/10.1137/0218053

[28] R. Y. Levine and A. T. Sherman, "A note on bennett's time-space tradeoff for reversible computation," *SIAM Journal on Computing*, vol. 19, no. 4, pp. 673–677, 1990. [Online]. Available: https://doi.org/10.1137/0219046

[29] D. H. Lehmer, "A note on trigonometric algebraic numbers," *The American Mathematical Monthly*, vol. 40, no. 3, pp. 165–166, 1933. [Online]. Available: http://www.jstor.org/stable/2301023

[30] I. Niven, *Irrational numbers*, ser. The Carus Mathematical Monographs. Mathematical Association of America, ; distributed by John Wiley & Sons, Inc., New York, 1956, vol. No. 11.

[31] V. Lefèvre, T. Ly, and P. Zimmermann, "Computing hard-to-round cases of sin, cos, tan in double precision," in *ARITH 2026 - 33rd IEEE International Symposium on Computer Arithmetic*, Fulda, Germany, Jun. 2026. [Online]. Available: https://inria.hal.science/hal-05593313

[32] R. P. Brent, "The complexity of multiple-precision arithmetic," in *The Complexity of Computational Problem Solving*, R. S. Anderssen and R. P. Brent, Eds. Brisbane, Australia: University of Queensland Press, 1976, pp. 126–165. [Online]. Available: https://arxiv.org/abs/1004.3412

[33] A. Parent, M. Roetteler, and M. Mosca, "Improved reversible and quantum circuits for Karatsuba-based integer multiplication," in *12th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2017)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), M. M. Wilde, Ed., vol. 73. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018, pp. 7:1–7:15. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.TQC.2017.7

[34] H. Bombín, "Single-shot fault-tolerant quantum error correction," *Phys. Rev. X*, vol. 5, p. 031043, Sep 2015. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevX.5.031043

[35] B. Eastin, "Distilling one-qubit magic states into toffoli states," *Phys. Rev. A*, vol. 87, p. 032321, Mar 2013. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.87.032321

[36] P. A. Spring, L. Milanovic, Y. Sunada, S. Wang, A. F. van Loo, S. Tamate, and Y. Nakamura, "Fast multiplexed superconducting-qubit readout with intrinsic purcell filtering using a multiconductor transmission line," *PRX Quantum*, vol. 6, p. 020345, Jun 2025. [Online]. Available: https://link.aps.org/doi/10.1103/PRXQuantum.6.020345

[37] S. Wang, X. Li, W. J. B. Lee, S. Deb, E. Lim, and A. Chattopadhyay, "A comprehensive study of quantum arithmetic circuits," *Philosophical Transactions A*, vol. 383, no. 2288, p. 20230392, 2025.

[38] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134.

[39] O. Regev, "An efficient quantum factoring algorithm," *J. ACM*, vol. 72, no. 1, Jan. 2025. [Online]. Available: https://doi.org/10.1145/3708471