IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

*Digital Object Identifier 10.1109/ACCESS.2024.0429000*

# Global Roots and Poles Finding Algorithm on Quantum Computer

**JAKUB BUCZKOWSKI[1], TOMASZ KOŹMIŃSKI[1], FILIP SZCZEPAŃSKI[1], MICHAŁ WILIŃSKI[1], and TOMASZ P. STEFAŃSKI[1], (Senior Member, IEEE)**

[1]Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Gdansk, Poland

Corresponding author: Tomasz P. Stefański (e-mail: tomasz.stefanski@pg.edu.pl).

**ABSTRACT** In this paper, the implementation of the global roots and poles finding algorithm for a complex-valued function of a complex variable on a quantum computer, which allows for solving general nonlinear algebraic equations, is presented. The considered function is sampled with the use of Delaunay's triangulation on the complex plane and a phase quadrant, in which the value of the function is located, is computed on a classical computer for all of the sampling nodes. Then, if the real and imaginary parts of the function simultaneously change signs for both ends of the same edge in the mesh, then a zero of the function is located in the region around this edge. In order to detect such edges, the mesh is transformed into a one-dimensional array and the required edges, where the sign simultaneously changes for real and imaginary parts of the function, are found with the use of quantum Grover's algorithm. If the mesh consists of $P$ edges, the computational overhead of this operation, in terms of oracle queries, is equal to $O(\sqrt{P})$ on a quantum computer, instead of $O(P)$ on a classical one. Finally, the existence of function zeros and poles is proved with the use of Cauchy's argument principle on a classical computer, and the output results are computed, based on the mesh refinement, with the assumed numerical precision of computations. Our method is implemented in Python with the use of the Qiskit software development kit and its applicability is proved by quantum emulations.

**INDEX TERMS** Quantum computing, quantum algorithm, quantum simulation, complex roots finding algorithm.

## I. INTRODUCTION

QUANTUM computing (QC) relies on using quantum mechanics for data and information processing. Although QC is in its early stages, it is believed that this technology can enable much faster computations than are now possible on classical computers. The 1980s are generally recognized as the beginning of investigations aimed at the implementation of computations with the use of quantum systems. It was then that Richard Feynman pointed out the possible advantages of computing with the use of quantum systems in 1982 [1], whilst David Deutsch presented the idea of a *universal quantum computer* in 1985 [2]. Around a decade later, first algorithms implementable on quantum computers were developed, i.e., Shor's algorithm for finding prime factors of numbers [3], and Grover's algorithm for searching databases [4]. In 1996, Seth Lloyd developed the quantum algorithm for simulations of quantum-mechanical systems [5], although, at that time, quantum computers did not exist yet. Then, Isaac Chuang, Neil Gershenfeld, and Mark Kubinec built the first quantum computer of the two-qubit size in 1998, allowing one to load data and output a solution [6]. This computer turned out to be coherent for only a few nanoseconds, and its function was trivial as far as the possibility of solving important computational problems was concerned. Despite that, it confirmed the existing theoretical predictions related to QC and stimulated fast progress of the quantum-computer technology. Currently, top IT companies (e.g., IBM [7]) focus on the development of QC, and new quantum computers are released each year. For the review of the QC development and its historical background, the reader is referred to [8]–[10].

In general, one ought to distinguish hardware (i.e., quantum computers) development from quantum-algorithm implementation. In this research, we focus on the implementation of the global roots and poles finding (GRPF) algorithm [11] based on phase analysis performed on a quantum computer. That is, we propose the implementation of the algorithm for finding zeros and poles of a complex function

of a complex variable. In this way, we can solve general nonlinear algebraic equations on a quantum computer, i.e., we can find the roots of an algebraic equation on the complex plane. In our method, Grover's algorithm [4] is employed to detect regions on the complex plane where zeros and poles are located. Afterwards, classical computations are executed which return zeros and poles with the error less than the assumed precision of the computations. In this area, QC has already been applied to solve systems of linear equations [12], [13] as well as to solve ordinary [14] and partial [15], [16] differential equations. However, the problem of finding zeros and poles of a general complex-valued function of a complex variable remains open not only in QC, but this is one of the oldest and still investigated mathematical problems of all time. We believe that the proposed application of QC for this purpose can direct further investigations towards developing new algorithms and methods in this area. In our investigations, we employ the Python programming language (version 3.10) and the Qiskit software development kit for QC (version 0.45.2). We also use Qiskit Aer library to simulate the circuits (version 0.13.2). Then, we employ the Triangle library [17] for Delaunay's triangulation [18]. Below, we report the results of numerical emulations on a personal computer which reproduce the results obtainable on a quantum computer. In order to facilitate further research, our emulation codes are publicly available on the Internet (i.e., after the acceptance of this manuscript for publication).

## II. GRPF IMPLEMENTATION ON QUANTUM COMPUTER

The aim of the proposed quantum algorithm is to find zeros and poles of the function $f : \mathbb{C} \mapsto \mathbb{C}$, i.e., $f(z) \in \mathbb{C}$ where $z \in \mathbb{C}$. The zeros $z_k$ $(k = 1, ..., K)$ of the complex function $f(z)$ are understood as the roots of the equation $f(z_k) = 0$. Analogously, the poles $p_l$ $(l = 1, ..., L)$ of the complex function $f(z)$ are understood as the roots of the equation $f^{-1}(p_l) = 0$.

In the subsequent sections, the GRPF algorithm implemented on a classical computer is presented and, then, Grover's algorithm is described briefly. Finally, we present the implementation of Grover's algorithm within GRPF in order to find zeros and poles of the complex function on a quantum computer.

### A. CLASSICAL GRPF ALGORITHM

GRPF is a numerical technique for finding zeros and poles of a wide class of complex functions. The algorithm is, in a sense, a generalization of the bisection method [19] widely applicable to real-valued functions. We consider the rectangular domain $\Omega \subset \mathbb{C}$ and search for zeros and poles of the complex function $f(z)$, where $z \in \Omega$. The accuracy of zero and pole locations is controlled by the initial mesh resolution $\Delta r$ and the final precision of computations $\epsilon$. The algorithm implemented on a classical computer is executed in the following steps:

1) The rectangular domain $\Omega$ is triangulated with the use of a regular mesh. Hence, the set of nodes $N = \{n_1, n_2, ..., n_M\}$ is created. We employ Delaunay's triangulation in the proposed algorithm in order to build a set of triangles. In other words, the set of edges $E = \{e_1, e_2, ..., e_P\}$ where $e_i = \{n_b, n_c\}$, being Delaunay's mesh, is created. It is required that the length of the longest edge in $E$ is less than or equal to the initial mesh resolution $\Delta r$.
2) Let us define the phase quadrant of the complex value of the function $f(z)$ at the point $z \in \Omega$

$$q[f(z)] = \begin{cases} 0 & 0 \le \arg[f(z)] < \pi/2 \\ 1 & \pi/2 \le \arg[f(z)] < \pi \\ 2 & \pi \le \arg[f(z)] < 3\pi/2 \\ 3 & 3\pi/2 \le \arg[f(z)] < 2\pi \end{cases} \quad (1)$$

where $\arg[\cdot]$ denotes the principal argument of the complex number in the interval $[0, 2\pi)$. One can note that the quadrant values fit into a two-bit number representation from the decimal range 0–3. GRPF computes the phase quadrant at each node $n_b$, hence an array of phase quadrants is created, i.e., $Q = \{q[f(n_b)] : n_b \in N\}$. In GRPF, the complex value of the function $f(z)$ is not required but only the quadrant in which its phase is located. This makes the algorithm less sensitive to the numerical precision of floating-point function computations, and applicable to QC.
3) For each of the edges $e_i$ connecting the nodes $n_b$ and $n_c$, the phase change is computed based on the quadrants obtained in the previous step, i.e., $\Delta q(e_i) = q[f(n_b)] - q[f(n_c)]$. Hence one obtains that $\Delta q(e_i) = -2, -1, 0, 1, 2$. Any zero or pole of the function $f(z)$ is located around the edge $e_i$ such as $\Delta q(e_i) = \pm 2$. This condition means that the real and imaginary parts of the considered function simultaneously change signs for both ends of the same edge. Hence either a zero or a pole should be located in the region around $e_i$, which is called the *candidate edge*, whereas the corresponding region is called the *candidate region*.
4) All the candidate edges are collected in a single set $E_c = \{e_i \in E : \Delta q(e_i) = \pm 2\}$. For a sufficiently dense mesh, a zero/pole has to be located inside a triangle which includes the edge being the candidate edge.
5) A set of triangles $T_c$, including at least a single candidate edge from the set $E_c$, is created. Afterwards, all the edges of the triangles belonging to $T_c$ are collected in a single set denoted by $E_t$. Each boundary $C_r$ (where $r = 1, ..., R$) of the r-th candidate region consists of the edges which occur only once in the set $E_t$. This stems from the fact that internal edges are attached to two candidate triangles. Hence the boundary $C = \bigcup_{r=1}^{R} C_r$ of the candidate regions is constructed from the edges $e_i \in E_t$ such as $|\Delta q(e_i)| < 2$.
6) The set $C$ is decomposed into subsets $C_r$.
7) The potential zeros and poles are within the candidate regions $C_r$. In order to increase the accuracy of local-

ization of these points, the mesh is refined in candidate regions. Additional points are added in the centers of the edges within the candidate regions. Then, Delaunay's triangulation is executed again, and a new mesh is obtained in each candidate region. The algorithm subsequently starts off in each candidate region from the second point as long as the length of the smallest edge in the candidate region is greater than the numerical precision $\epsilon$.

8) The final verification and classification, whether the candidate region $C_r$ includes either a zero or a pole, is executed with the use of Cauchy's argument principle [20]. It requires the calculation of the integral

$$w_r = \frac{1}{2\pi i} \oint_{C_r} \frac{f'(z)}{f(z)} \mathrm{d}z. \tag{2}$$

The value of $w_r \in \mathbb{N}$ is positive for $w_r$th-order zero and negative for $w_r$th-order pole. In other cases, the candidate region $C_r$ does not include any zero/pole. In the discrete triangulated domain, the integral (2) is computed as

$$w_r = \frac{1}{4} \sum_{s=1}^{S} \Delta q(e_s) \tag{3}$$

where $e_s \in C_r$ and $S$ denotes the number of edges in the boundary $C_r$.

FIG. 1 demonstrates the operation principle of GRPF, where the quadrants of phase are presented for the function $f(z) = (z - 0.5)/(z + 0.5)^2$. This function includes the first-order zero in $z_1 = 0.5$ and the second-order pole in $z_2 = -0.5$. The first-order zero is visible as the change of phase between all the four quadrants in the counterclockwise direction around $z_1$ which reaches $2\pi$. Then, the second-order pole is visible as the change of phase between all the four quadrants in the clockwise direction around $z_2$ which reaches $4\pi$. These simple changes of quadrants (i.e., colors) are analyzed by GRPF, allowing one to find zeros and poles of a function on the complex plane. It is important to employ a sufficiently dense mesh, which means that the number of edges $P$ can be large. In FIG. 2, the mesh with visible edges and its refinement is presented. As one can note, GRPF correctly reduces the mesh size around the candidate edges.

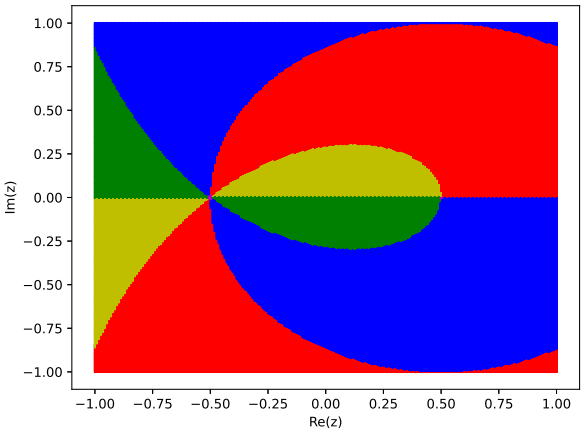


**FIGURE 1. Quadrants of phase (■ (0), ■ (1), ■ (2), ■ (3)) for function $f(z) = (z - 0.5)/(z + 0.5)^2$. Initial mesh size is set to 0.01.**

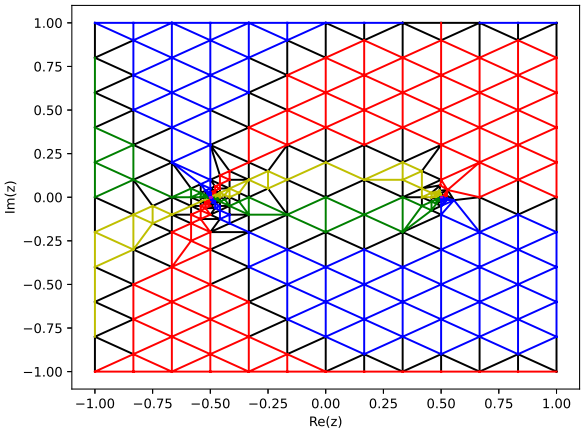


**FIGURE 2. Quadrants of phase (■ (0), ■ (1), ■ (2), ■ (3)) for function $f(z) = (z - 0.5)/(z + 0.5)^2$. Initial mesh size is set to 0.2.**

### B. GROVER'S ALGORITHM

We employ Grover's algorithm [4] in the proposed GRPF implementation, which is a quantum search algorithm. Let us define the function $F : \{0, 1, ..., P-1\} \mapsto \{0, 1\}$ which points out the element in the database we are searching for. That is, $F(x) = 1$ if $x$ is the index of the item which we are searching for, and $F(x) = 0$ otherwise. Grover's algorithm requires defining the function $Z_F$, called an oracle, which is a unitary operator taking the quantum-register input $|x\rangle$ of the size $p = \lceil \log_2 P \rceil$ and flipping it iff $x$ is the index of the item which we are searching for. That is, the action of the oracle is defined as

$$|x\rangle \xmapsto{Z_F} (-1)^{F(x)} |x\rangle . \tag{4}$$

The algorithm starts off by setting the input register to an equal superposition of all the input states with the use of the Hadamard operator $H^{\otimes p}$. That is, the following state is set at the input

$$|\psi\rangle = H^{\otimes p} |0^p\rangle = \frac{1}{\sqrt{P}} \sum_{x=0}^{P-1} |x\rangle . \tag{5}$$

Then, Grover's operator

$$G = (2 |\psi\rangle \langle\psi| - \hat{1}) Z_F \tag{6}$$

based on the oracle $Z_F$ is called $O(\sqrt{P})$ times. The measurement of a quantum-register state returns the index of the element which we are searching for. It is possible to extend the basic operation of Grover's algorithm when we search for multiple elements in a database [8].

### C. GRPF ON QUANTUM COMPUTER

Quantum implementation of GRPF consists of data preparation on a classical computer, quantum computations for candidate-edge detection and, again, classical computations

for the subsequent mesh refinement. It is assumed that the computational overhead of data preparation on a classical computer is not significant compared to the overhead of searching for candidate edges. Otherwise, the use of quantum computations in GRPF will not provide important advantages in comparison to classical computations. The domain analyzed by GRPF can be arbitrarily large, hence the number of edges after triangulation can be substantial. However, with the use of QC, one can reduce the computational overhead of the candidate-edge detection in GRPF from $O(P)$ to $O(\sqrt{P})$.

If the time of calling the function and computing the function value in a single point is significant, it is crucial to minimize the number of function calls. Then, one can reduce the number of function calls compared to the original GRPF algorithm with the use of a self-adaptive mesh generator [21]. However, we do not investigate this GRPF extension in order to concentrate on quantum implementation of this algorithm.

We focus further on the first iteration (i.e., executed on a quantum computer) of the quantum GRPF algorithm, which is presented in FIG. 3. QC is based on random properties of the reality, hence this iteration includes tasks which are executed multiple times in a loop (i.e., the number of these executions is chosen randomly). In the subsequent sections, we describe the elements of this quantum iteration of GRPF.

### 1) Input data

Based on the input data prepared on a classical computer, quantum circuits are generated. These data consist of quadrants of function values for nodes of a triangular mesh. It is checked, inside the quantum circuits, if the difference between the quadrants is equal to two for the nodes in a single edge. Hence the application of Grover's algorithm to GRPF requires its implementation which processes the nodes and edges of the mesh. That is, the edges in the triangulated domain are the search space, whereas the function $F$ returns 1 when the argument $x$ is the candidate edge. The quantum circuits based on Grover's algorithm find the candidate edges in the first iteration of the GRPF algorithm. In our Python implementation, there are three such quantum circuits and each one computes a set of candidate edges in one out of three directions $\delta \in \{\alpha, \beta, \gamma\}$ on the regular triangular mesh (see FIG. 4). As one can note, such a mesh geometry allows for covering the rectangular domain with edges (i.e., $E = E_\alpha \cup E_\beta \cup E_\gamma$), which can easily be decomposed into the sets $E_\alpha$, $E_\beta$, $E_\gamma$ associated with the search directions. In general, the three quantum circuits, each processing one of the directions $\alpha$, $\beta$, $\gamma$, are similar and only differ in the direction of the edges they process. Then, when the candidate edges are detected, the mesh is refined on a classical computer and the complex values of zeros and poles are computed with the assumed numerical precision $\epsilon$. Of course, one can employ other methods of decomposing the mesh into directions, even without covering the rectangular boundary.

### 2) Query model of computations

The quantum circuit for each direction consists of four quantum registers: **a**, **b**, **c** and **d**. The registers **a**, **b**, and **c** are used together as the search space for Grover's algorithm. The size of the register **a** is set to fit the largest index of nodes in the generated triangular mesh, so its size is $m = \lceil \log_2(M) \rceil$, where $M$ is the size of the set of all the nodes. The registers **b** and **c** have a constant size of two qubits to fit four possible values of a quadrant index $(0, 1, 2, 3)$. The register **d** of a single-qubit size is used for the *phase kickback* in the oracle (which is explained below), and it is initialized to the minus state $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$. In our implementation of Grover's algorithm, the query model of computations [22] is used. This means that the function $F$ is accessed by the query gate defined as the unitary operation

$$U_F(|y\rangle\,|x\rangle) = |y \oplus F(x)\rangle\,|x\rangle\,. \tag{7}$$

The gate operates on two quantum states, i.e., the function-argument state $|x\rangle$ and the function-value state $|y\rangle$. The exclusive-alternation (XOR) operation is realized for the state $|y\rangle$ and the value of the function $F(x)$, hence the function-argument state remains the same. One can note that XOR of two numbers $a$ and $b$, represented by quantum states, is equivalent to the operation of negation

$$\mathrm{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{8}$$

executed $a$ times on the state $|b\rangle$. That is, one can write $|a \oplus b\rangle = \mathrm{X}^a\,|b\rangle = \mathrm{X}^b\,|a\rangle$. This can easily be proved by checking equalities for the states of the computational basis
$|0 \oplus 0\rangle = \mathrm{X}^0\,|0\rangle = |0\rangle$
$|0 \oplus 1\rangle = \mathrm{X}^1\,|0\rangle = |1\rangle$
$|1 \oplus 0\rangle = \mathrm{X}^0\,|1\rangle = |1\rangle$
$|1 \oplus 1\rangle = \mathrm{X}^1\,|1\rangle = |0\rangle$
and then taking into account the fact that any state is a linear combination of the computational-basis states $|0\rangle$ and $|1\rangle$. Knowing this, we can rewrite the unitary operation of the query gate as $U_F(|y\rangle\,|x\rangle) = |y \oplus F(x)\rangle\,|x\rangle = (\mathrm{X}^{F(x)}\,|y\rangle)\,|x\rangle$.

The operator $Z_F$ in Grover's operator $G$ is a query gate that flips the sign of $|x\rangle$ iff $F(x) = 1$. To achieve this, the state $|y\rangle$ is set to the minus state $|-\rangle$. Then, the operation of our query gate is $(\mathrm{X}^{F(x)}\,|-\rangle)\,|x\rangle$. However, one can note that $\mathrm{X}\,|-\rangle = -\,|-\rangle$, so the operation of the query gate becomes $((-1)^{F(x)}\,|-\rangle)\,|x\rangle$. This can be rewritten as $|-\rangle\,((-1)^{F(x)}\,|x\rangle)$, so now, in a sense, the argument state has changed, but the value state has not. This phenomenon is called the phase kickback in our implementation of the oracle $Z_F$.

### 3) Oracle

Although the oracle is usually represented by a single quantum gate, here we develop a quantum circuit in its place. Its operation can be understood by analyzing a classical version of such a circuit, where the bits store classical states. Its quantum version concurrently computes all the candidate edges by the superposition of states. The quantum circuit of

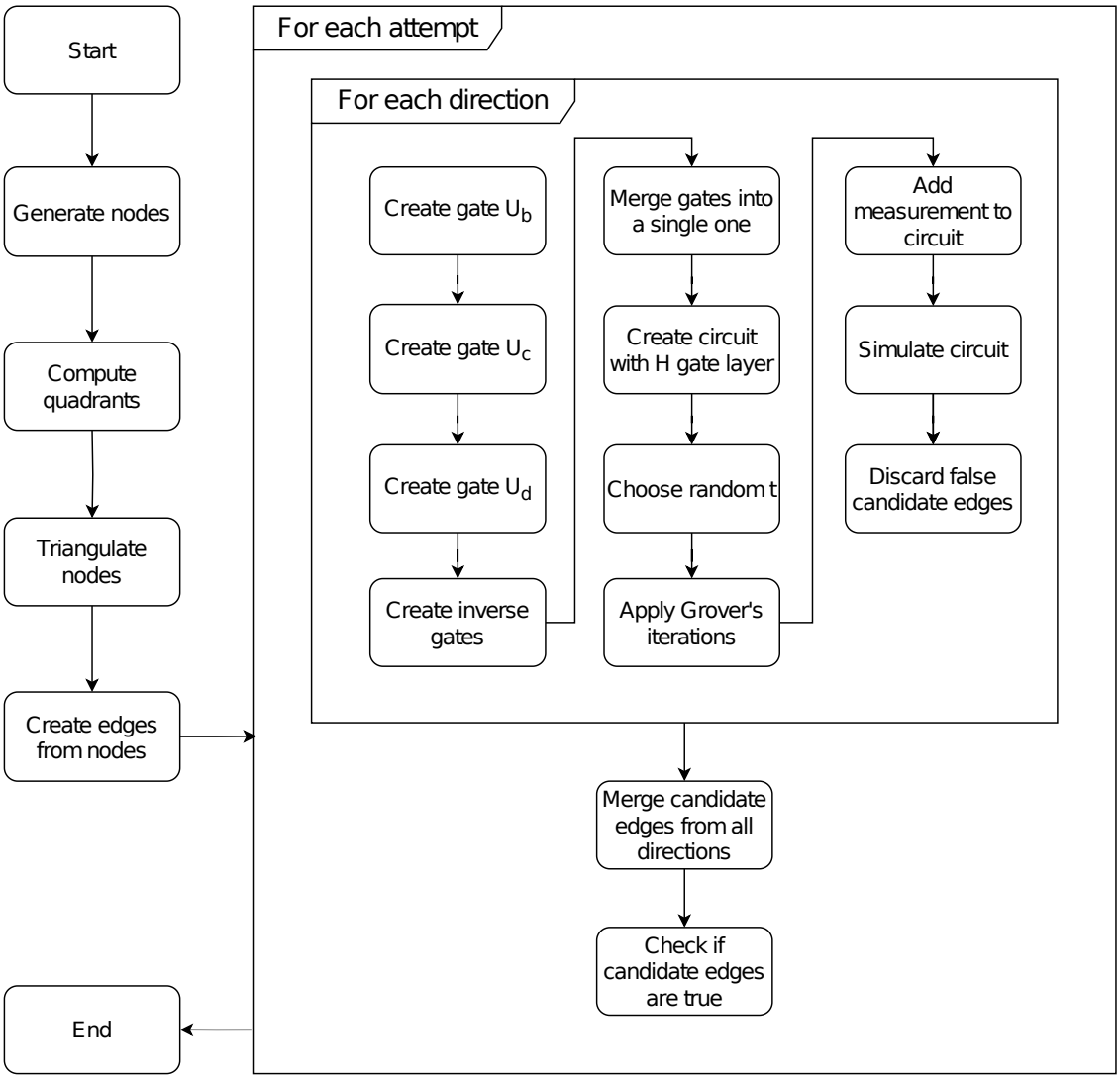


**FIGURE 3.** **Flowchart of the first (quantum) iteration of proposed GRPF algorithm.**

the oracle is presented in FIG. 5. It takes the node index from one side of the edge which is stored in the register **a**. The first gate $U_b$ sets the state of the register **b** to the quadrant of the node represented by $|a\rangle$. The second gate $U_c$ sets the state of the register **c** to the quadrant of the node on the other side of the edge connected to the node represented by $|a\rangle$. The third gate $U_d$ performs the XOR operation on the register **d** resulting in the function $F$. Therefore, if **d** is 0 and the edge is the candidate edge, meaning $F$ returned 1, then **d** will be $0 \oplus 1 = 1$. This output means that the analyzed edge is a candidate edge.

The gates $U_b$ and $U_c$ are the unitary operators defined as follows:

$$U_b |b\rangle |a\rangle = |q[f(n_b)]\rangle |a\rangle \tag{9}$$

$$U_c |c\rangle |a\rangle = |q[f(n_c)]\rangle |a\rangle . \tag{10}$$

In (9)–(10), $n_b$ and $n_c$ denote the nodes, i.e., complex values representing their coordinates on the complex plane. Let us define the edge $x = \{n_b, n_c\}$. Knowing that $|d\rangle$ is initialized with $|-\rangle$, we can describe $U_d$ as follows:

$$U_d|d\rangle|c\rangle|b\rangle = |-\rangle((-1)^{F(x)}|c\rangle|b\rangle) \tag{11}$$

where

$$F(x) = \begin{cases} 0 & \text{if } |q[f(n_b)] - q[f(n_c)]| \neq 2 \\ 1 & \text{if } |q[f(n_b)] - q[f(n_c)]| = 2 \end{cases} . \tag{12}$$

The gate $U_b$ consists of multi-controlled X (MCX) gates, which are gates with multiple control qubits that determine

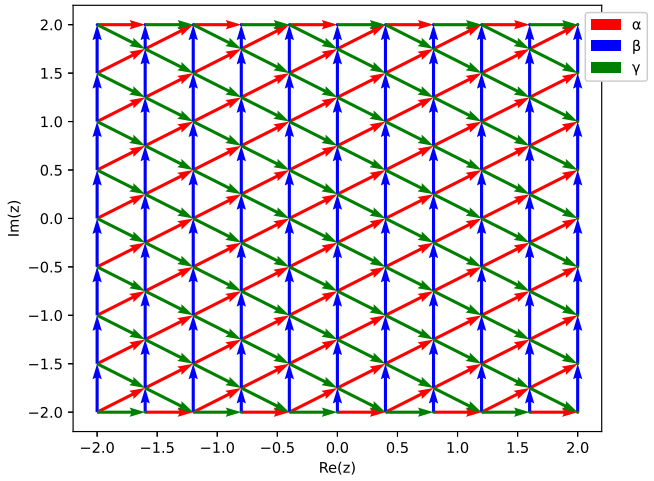


**FIGURE 4.** Edge directions processed by quantum circuits on a small mesh.

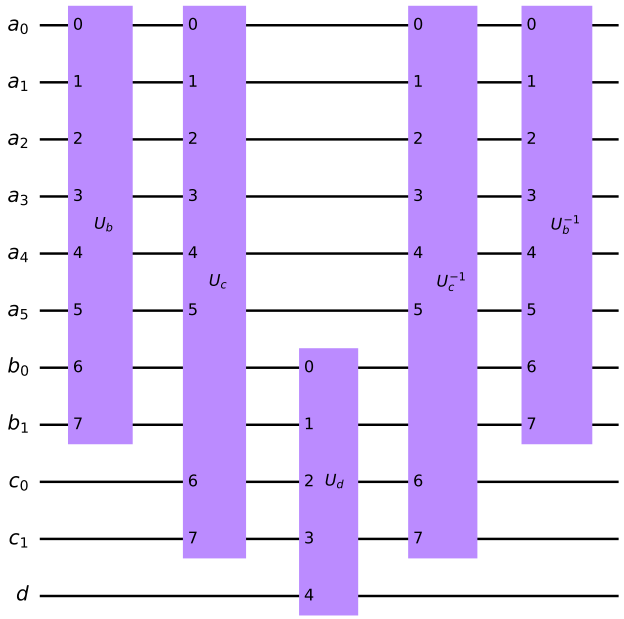


**FIGURE 5.** Quantum circuit representing oracle for $m = 6$.

whether a given gate should be enabled. In our case, the register **a** with node indices represents the control for these MCX gates. That is, we add the X gate before the control to qubits, which we want to set in the register **b**, if the corresponding bit of the node index is 0. Therefore, the control lines enable the gates which set the quadrants into the register **b**. As one can note, the X gates are always applied in pairs in order to restore the input states of qubits on control lines. The targets of the MCX gates are one or both qubits, depending on the quadrant values which should be written into the register **b**. Let us consider the following example of the gate $U_b$, which is presented in FIG. 6. Suppose that $[1, 1, 3, 2]$ is the array of quadrants for four subsequent nodes in the mesh. Hence we can assume that the size of the register **a** is set at two qubits. The index of the first quadrant (i.e., 1) in the array is 0. Our aim is to activate the least significant bit $b_0$ when the node index in the register **a** is equal to 00 (binary). Therefore, the X gates are initially placed at the control lines from the register **a**, which gives the enable signal activating the line $b_0$. Hence the value 01 (binary) is set at the register **b**. Analogously, other binary values of quadrants are set by the following gates, according to the binary indices of the array elements.

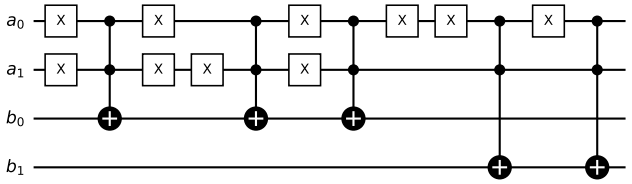


**FIGURE 6.** Exemplary gate $U_b$ for array of quadrants [1, 1, 3, 2].

The gate $U_c$ is constructed similarly to $U_b$, except that for each node, instead of its own quadrant value, the quadrant value of its neighbor is assigned in a given direction (see again FIG. 4). Using this scheme, the quadrants of neighbors for each node in an array are assigned, and passed on to the function constructing $U_b$.

The gate $U_d$ is represented by a matrix of the size $2^5 \times 2^5$, because $size(\mathbf{b}) + size(\mathbf{c}) + size(\mathbf{d}) = 5$. The behavior of any unitary-matrix transformation can be described by analyzing the columns of the matrix, each having an index that is represented in Dirac's notation. Therefore, for each column having the index $dc_1c_0b_1b_0$ (binary) representing the quantum state $|d\rangle\,|c\rangle\,|b\rangle$, we insert the output which indicates if the input state denotes the candidate edge. That is, $U_d$ is a unitary transformation represented, to some extent, by a lookup table. For example, having the quantum state $|0\rangle|00\rangle|11\rangle$ (the candidate edge) as the input, we expect the quantum state $|1\rangle|00\rangle|11\rangle$ at the output. Therefore, we insert a column vector representing the expected state as the matrix column with the index represented by the input state. The output bit $d$ is computed by XOR of the input $d$ and the expected output $d$, because it is done in the query model of computations. This means that the two nodes (having $b$ and $c$ quadrant values) are a candidate edge. The matrix $U_d$ can be written as

$$U_d = \begin{bmatrix} | & | & & | \\ |y\rangle_{00000} & |y\rangle_{00001} & \cdots & |y\rangle_{11111} \\ | & | & & | \end{bmatrix}$$

where $|y\rangle_{dc_1c_0b_1b_0}$ is a basis state which is the expected value for the input basis state $|dc_1c_0b_1b_0\rangle$, i.e., $U_d\,|dc_1c_0b_1b_0\rangle = |y\rangle_{dc_1c_0b_1b_0}$. Because we are dealing with quantum states, the gate $U_d$ is not the last one in the circuit of the oracle. There are two quantum gates afterwards being the inverse of the gate $U_c$ and the inverse of the gate $U_b$. This is implemented in order to retrieve the previous states of the registers **a**, **b** and **c**. Therefore, the whole quantum circuit can be treated as the oracle $Z_F$.

It is the fact that some of QC algorithms are still far from being implementable for meaningful applications on today's quantum hardware [23], [24]. Therefore, Grover's algorithm is usually not considered in terms of implementability, but the polynomial reduction in a query complexity (i.e., how many times the oracle is queried in a similar way to a classical database). Although Grover's algorithm may not provide a practical quantum advantage in searches in the near future, it is a fundamentally important quantum algorithm, as well as a representative model for a more general technique with many applications in QC. In our case, the transfer of classical data to the quantum oracle is a bottleneck for the efficiency of the algorithm (with respect to classical solutions), because the number of the gates X and MCX in $U_b$ and $U_c$ depends, as $O(P)$, on the size of the database (i.e. the size of the mesh consisting of $P$ edges). However, the gate $U_d$ has a size independent of the mesh size, which is an advantageous feature. To sum up, our GRPF implementation should provide advantages in comparison to classical approaches if an efficient method of data transfer between classical and quantum computing systems is developed.

#### 4) Quantum computations

Three quantum circuits implementing the oracle, each for a single direction $\alpha, \beta, \gamma$ in the mesh, run sequentially in a loop. In general, a single run of the quantum algorithm is usually insufficient to obtain results in QC. Therefore, we simulate each quantum circuit 1024 times to obtain a histogram of the outcomes, which is a default number of simulations of a quantum circuit in Qiskit. Then, we choose the outcomes which occur more often than the others. Each outcome is the index of a node, which together with the mesh direction of the quantum circuit, represents a candidate edge. After running the quantum circuit for a single direction $\delta \in \{\alpha, \beta, \gamma\}$, one obtains the set of candidate edges

$$E_c = \left\{ x \in E : \frac{o_x}{o_{max}} > DT \right\} \tag{13}$$

where $x = \{n_b, n_c\}$ is the edge uniquely identified by the pair $(n_b, \delta)$, $o_x$ is the number of times that this edge occurred as the outcome of the simulation along the direction $\delta$, and $o_{max}$ is the maximal number of times that any edge occurred as the outcome of the same simulation along the direction $\delta$. In (13), $DT$ denotes the detection threshold of candidate edges which we set experimentally at the level of 50%.

Based on (5)–(6), the whole quantum circuit of the GRPF algorithm for a single direction, operating on $|a\rangle$, can finally be described as follows:

$$[H^{\otimes m}(2\,|0^m\rangle\,\langle 0^m| - \hat{1}^m)H^{\otimes m}Z_F|_{\mathbf{a}}]^t H^{\otimes m} \tag{14}$$

where $Z_F|_{\mathbf{a}}$ denotes the oracle operating on the register **a**, and $t$ is the number of iterations of Grover's algorithm. If we knew how many solutions (candidate edges) exist along a single direction, then $t$ could be calculated so that it would yield the highest probability of finding them all. Unfortunately we do not know the number of solutions beforehand, so we need to calculate $t$ in a different way. This can be done by randomly choosing a number from the set $\left\{1, \ldots, \left\lfloor \frac{\pi\sqrt{\bar{M}}}{4} \right\rfloor\right\}$ where $\bar{M} = 2^m$ denotes the number of possible outcomes and $m$ is the size of the register **a**. This method gives us a chance of finding a single solution (assuming one exists) in a single simulation greater than 40% [25]. By repeating this procedure and checking the outcome in the same way as described before, the probability of finding a solution can be made very close to 1. This procedure is proposed in Qiskit as a default option, but other methods of estimating the number of solutions can be applied, e.g., based on quantum counting [26].

If the results of a simulation along a single direction are ambiguous, then all the solutions along that direction are discarded. This happens when there are no true candidate edges in a given direction. The ambiguity $A$ is measured by the ratio of the number of candidate edges proposed by the algorithm $M_c$ to the size of the search space of Grover's algorithm:

$$A = \frac{M_c}{\bar{M}}. \tag{15}$$

We discard the edges $E_c$ if the ambiguity is greater than or equal to the level of 33%, which we found with the trial and error method.

## III. NUMERICAL RESULTS

We employ 7 qubits for the register **a** in order to run emulations of the quantum GRPF algorithm in reasonable time. The exact number of the required qubits depends on the initial distance between nodes $\Delta r$. However, if the size of the register **a** is greater than 7, then there might be a need to increase the number of simulations to over 1024 in order to decrease statistical fluctuations of the results.

The next two subsections present the results of the quantum GRPF algorithm applied to two complex functions $f_A$ and $f_B$. For each of them, we set the final precision of computations at $\epsilon = 10^{-9}$. The initial mesh resolution $\Delta r$, which defines the distance between the nodes in the first regular meshing, is set individually for each considered function.

### A. FUNCTION $F_A$

This function is defined as follows:

$$f_A(z) = (z - (4 + 3i))^2(z + 3)(z + i)^3(z - 2)^2(z - (2 + i)).$$

Its analytical zeros and their orders are provided in TABLE 1.

| Zero | Multiplicity |
|---|---|
| $4 + 3i$ | 2 |
| $-3$ | 1 |
| $-i$ | 3 |
| $2$ | 2 |
| $2 + i$ | 1 |

**TABLE 1. Zeros with multiplicities for $f_A$ (theoretically).**

The function is analyzed for

$$z \in \Omega = \{a + bi \ : \ a \in [-8, 8] \wedge b \in [-8, 8]\}.$$

The initial mesh is generated with $\Delta r = 3$. In the first iteration, the quantum circuits find 30 candidate edges that are colored purple in FIG. 7. In this simulation, the edges from the direction $\beta$ are discarded because of too high ambiguity of the results. The numbers of occurrences of a given edge

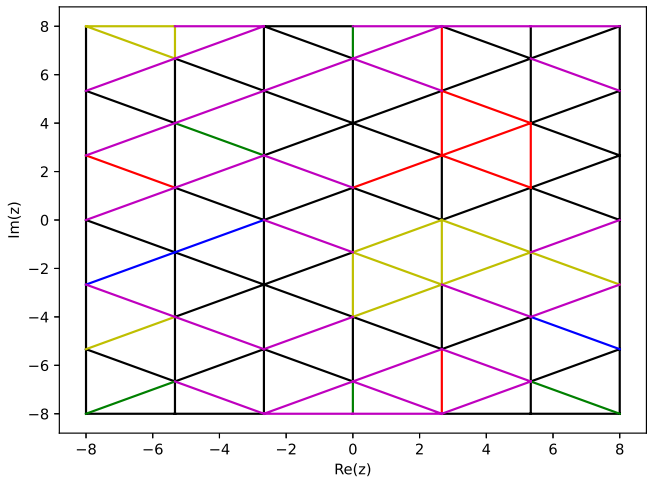


**FIGURE 7. Candidate edges (■) in the first iteration for $f_A$.**

as a candidate for each direction are presented in FIG. 8. As a result of the algorithm run, one obtains all the zeros of the function $f_A$ computed with some numerical error, as given in TABLE 2. One can note that obtained values are within the range defined by the numerical precision of computations $\epsilon$. Furthermore, quantum GRPF detects all the zeros within the

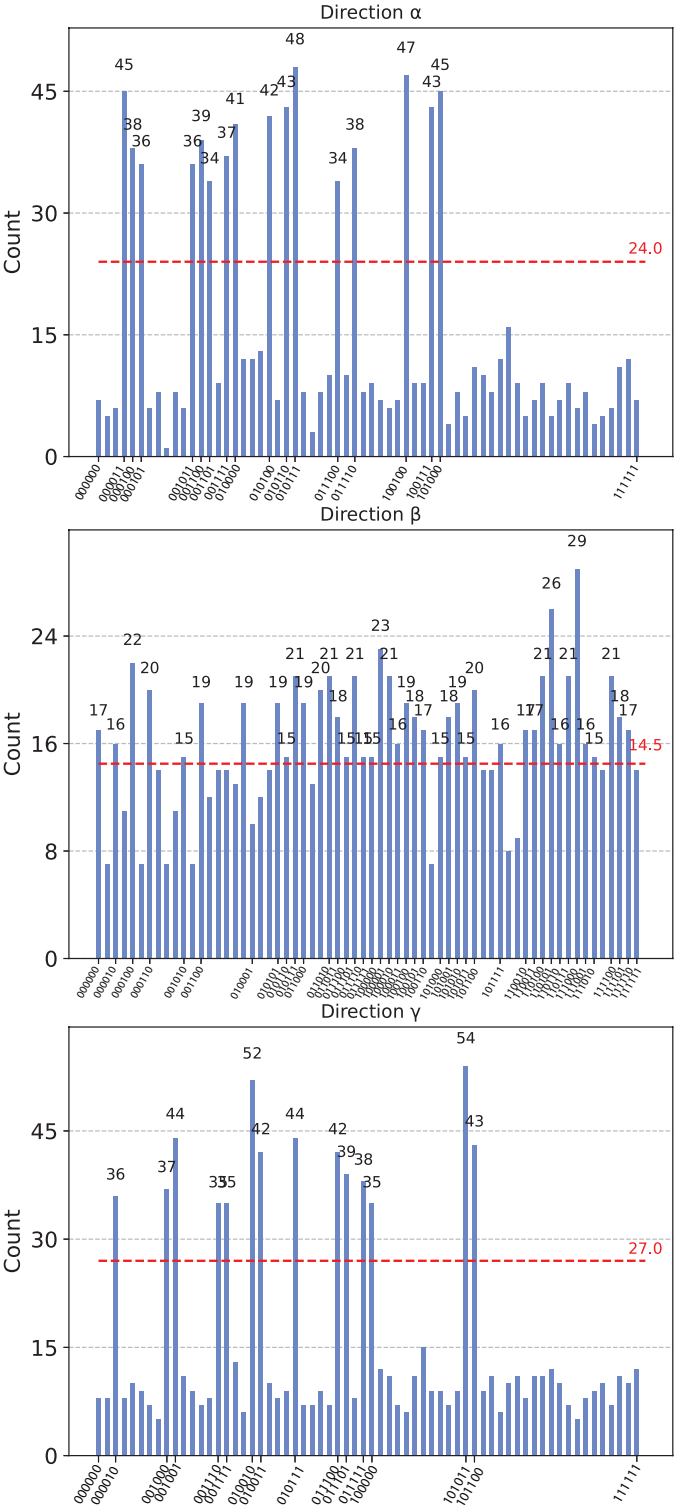


**FIGURE 8. Function A: Number of occurrences as candidate edge for direction $\alpha$, $\beta$, $\gamma$. Horizontal axis contains binary indices of edges that occur more often than detection threshold.**

assumed searching space. One can note that the multiplicities of function zeros are correctly computed classically. This confirms the correctness of our implementation of quantum GRPF in a code. The final mesh, i.e., after running the algo-

| Zero | Multiplicity |
|---|---|
| $3.999999999943556 + 3.0000000000282214i$ | 2 |
| $-3.000000000214626 + 1.8971383003646632e - 10i$ | 1 |
| $3.880510727564485e - 11 - 0.9999999999805979i$ | 3 |
| $1.9999999999435558 - 2.82219517116203e - 11i$ | 2 |
| $2.0000000003725287 + 0.9999999998137353i$ | 1 |

**TABLE 2. Zeros with multiplicities for $f_A$ (numerically).**

rithm, is presented in FIG. 9 for reference.

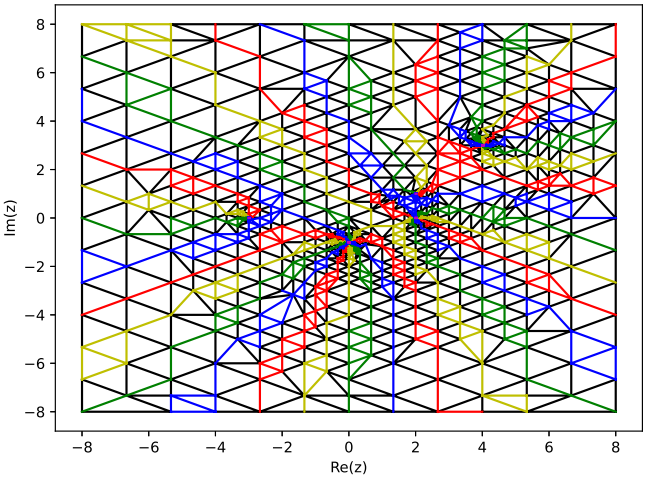


**FIGURE 9. Mesh on the last iteration of quantum GRPF algorithm on $f_A$. Quadrants of phase: ■ (0), ■ (1), ■ (2), ■ (3).**

### B. FUNCTION $F_B$

This function is defined as follows:

$$f_B(z) = (z-1)(z-i)^2\frac{(z+1)^3}{z+i}.$$

Its analytical zeros and poles, with their orders, are provided in TABLE 3.

| Zero | Multiplicity |
|---|---|
| 1 | 1 |
| $i$ | 2 |
| $-1$ | 3 |
| Pole | Multiplicity |
| $-i$ | 1 |

**TABLE 3. Zeros and poles with multiplicities for $f_B$ (theoretically).**

The function is analyzed for

$$z \in \Omega = \{a + bi \ : \ a \in [-2, 2] \wedge b \in [-2, 2]\}.$$

The initial mesh is generated with $\Delta r = 0.5$. In the first iteration, the quantum circuits find 17 candidate edges that are colored purple in FIG. 10. The numbers of occurrences of a given edge as a candidate for each direction are presented in FIG. 11. As a result of the algorithm run, one obtains all the zeros and poles of the function $f_B$ computed with some numerical error, as given in TABLE 4. One can note

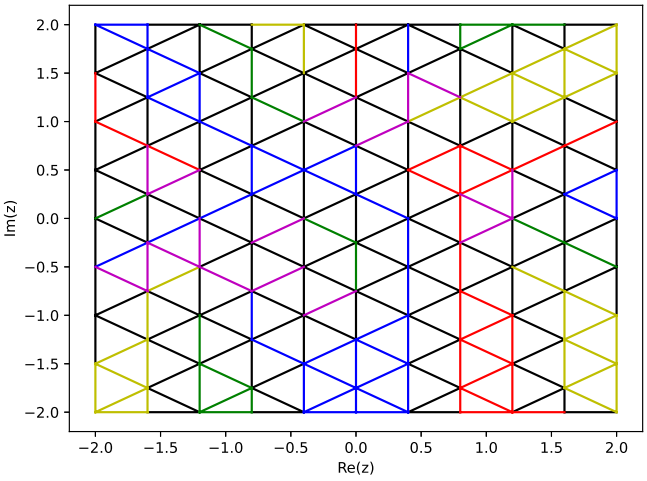


**FIGURE 10. Candidate edges (■) in the first iteration for $f_B$.**

that obtained values are within the range defined by the numerical precision of computations $\epsilon$. Again, quantum GRPF detects all the zeros and poles within the assumed searching space. Then, the multiplicities of function zeros and poles are correctly computed. This confirms the correctness of our implementations of quantum GRPF in a code.

| Zero | Multiplicity |
|---|---|
| $0.999999999627471 + 7.761021455128987e - 11i$ | 1 |
| $1.6556845770941875e - 10 + 0.9999999998965197i$ | 2 |
| $-1.000000000057312 - 1.0746029707101676e - 10i$ | 3 |
| Pole | Multiplicity |
| $3.101927297073854e - 25 - 1.0000000000582077i$ | 1 |

**TABLE 4. Zeros and poles with multiplicities for $f_B$ (numerically).**

The final mesh is presented in FIG. 12.

## IV. CONCLUSION

The implementation of the GRPF algorithm, which allows for solving general nonlinear algebraic equations on a quantum computer, is developed. The considered function is sampled with the use of Delaunay's triangulation on the complex plane, and the phase quadrants, where the function values are located, are computed on a classical computer. Then, quantum circuits are employed in the first iteration of the algorithm to detect the edges (so called candidate edges) in the mesh for which function values belong to the opposite quadrants of the complex plane. Zeros and poles of a complex function are located around such edges, which are afterwards precisely computed with the use of mesh refinement on a classical computer. In order to effectively detect candidate edges on a quantum computer, the mesh is transformed into a one-dimensional array and the candidate edges are found with the use of Grover's algorithm. If the mesh consists of $P$ edges, the computational overhead of this operation, in terms of oracle queries, is equal to $O(\sqrt{P})$ on a quantum computer,

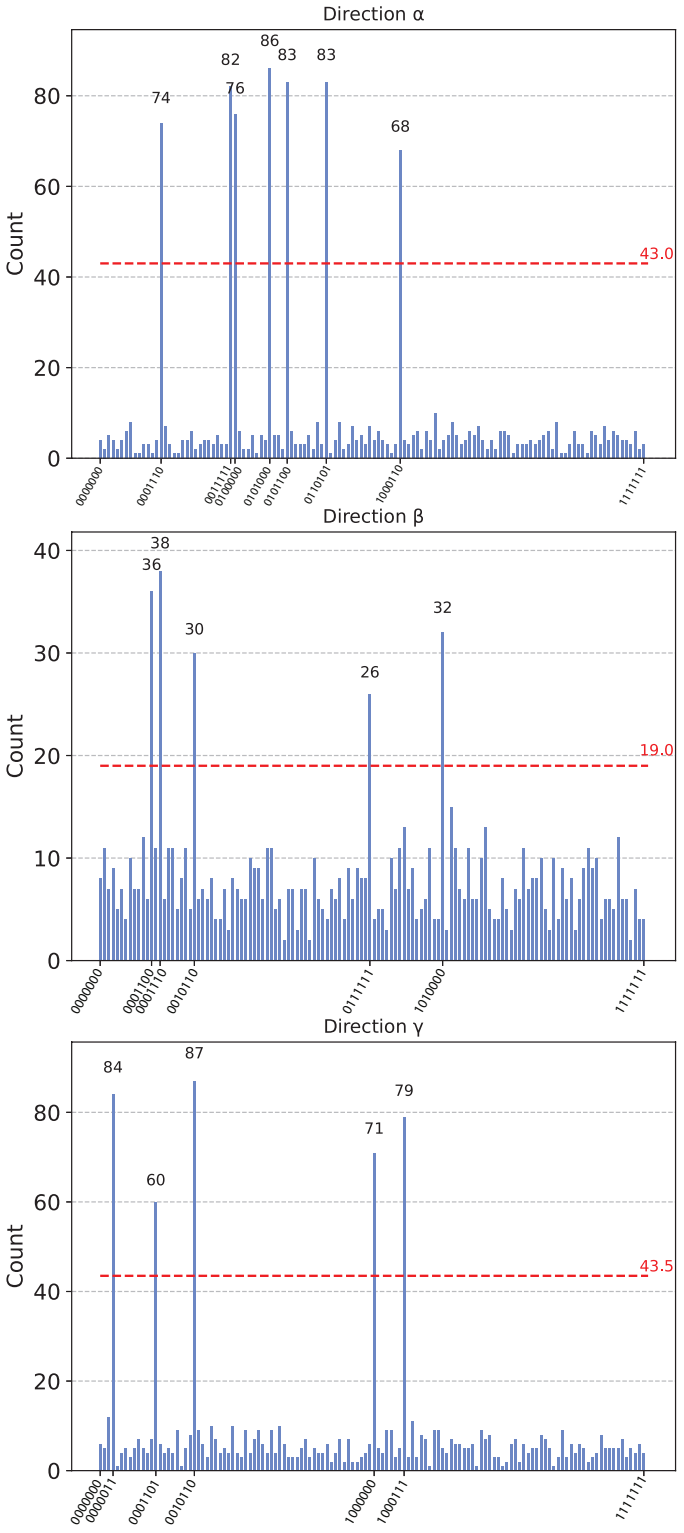


**FIGURE 11. Function B: Number of occurrences as candidate edge for direction $\alpha$, $\beta$, $\gamma$. Horizontal axis contains binary indices of edges that occur more often than detection threshold.**

instead of $O(P)$ on a classical one. The proposed algorithm is implemented in Python using the Qiskit library and it is open sourced. Using the emulation of QC, we are able to

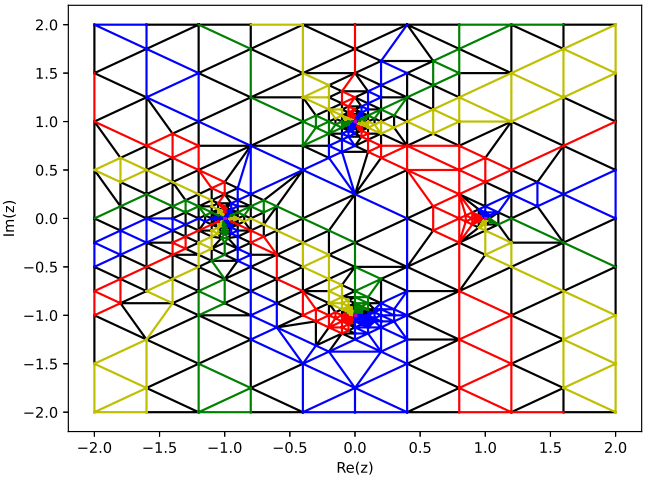


**FIGURE 12. Mesh on the last iteration of quantum GRPF algorithm on $f_B$. Quadrants of phase: (0), (1), (2), (3).**

demonstrate the correct operation of the developed algorithm based on two exemplary complex functions.

## SOURCE CODE

The source code for the QC implementation of the global complex roots and poles finding algorithm based on phase analysis will be released on: https://github.com/stafan26, and will be licensed under the MIT License (i.e., after the acceptance of this manuscript for publication).

## REFERENCES

[1] Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6/7):467–488, 1982.

[2] David Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[3] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[4] Lov K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, 79:325–328, Jul 1997.

[5] Seth Lloyd. Universal quantum simulators. *Science*, 273(5278):1073–1078, 1996.

[6] Isaac L. Chuang, Neil Gershenfeld, and Mark Kubinec. Experimental implementation of fast quantum searching. *Phys. Rev. Lett.*, 80:3408–3411, Apr 1998.

[7] Davide Castelvecchi. Ibm releases first-ever 1,000-qubit quantum chip. *Nature*, 624, Dec 2023.

[8] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[9] T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. L. O'Brien. Quantum computers. *Nature*, 464(7285):45–53, Mar 2010.

[10] Sandeep Kumar Sood and Pooja. Quantum computing review: A decade of research. *IEEE Transactions on Engineering Management*, pages 1–15, 2023.

[11] Piotr Kowalczyk. Global complex roots and poles finding algorithm based on phase analysis for propagation and radiation problems. *IEEE Transactions on Antennas and Propagation*, 66(12):7198–7205, 2018.

[12] Stefanie Barz, Ivan Kassal, Martin Ringbauer, Yannick Ole Lipp, Borivoje Dakić, Alán Aspuru-Guzik, and Philip Walther. A two-qubit photonic quantum processor and its application to solving systems of linear equations. *Scientific Reports*, 4(1):6115, Aug 2014.

[13] Li Xu, Xiao qi Liu, Jin min Liang, Jing Wang, Ming Li, and Shu qian Shen. Quantum algorithm for solving matrix equations of the form ax = b. *Laser Physics Letters*, 19(5):055202, mar 2022.
[14] Benjamin Zanger, Christian B. Mendl, Martin Schulz, and Martin Schreiber. Quantum Algorithms for Solving Ordinary Differential Equations via Classical Integration Methods. *Quantum*, 5:502, July 2021.
[15] Albert J. Pool, Alejandro D. Somoza, Michael Lubasch, and Birger Horstmann. Solving partial differential equations using a quantum computer. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 864–866, 2022.
[16] Furkan Oz, Omer San, and Kursat Kara. An efficient quantum partial differential equation solver with chebyshev points. *Scientific Reports*, 13(1):7767, May 2023.
[17] Jonathan Richard Shewchuk. Triangle library, 2005.
[18] E. W. Weisstein. Delaunay triangulation, 2024.
[19] E. W. Weisstein. Bisection, 2024.
[20] E. W. Weisstein. Argument principle, 2024.
[21] Sebastian Dziedziewicz, Malgorzata Warecka, Rafal Lech, and Piotr Kowalczyk. Self-adaptive mesh generator for global complex roots and poles finding algorithm. *IEEE Transactions on Microwave Theory and Techniques*, 71(7):2854–2863, 2023.
[22] John Watrous. IBM Quantum Learning: Quantum query algorithms, 2024.
[23] G.F. Viamontes, I.L. Markov, and J.P. Hayes. Is quantum search practical? *Computing in Science & Engineering*, 7(3):62–70, 2005.
[24] Raphael Seidel, Colin Kai-Uwe Becker, Sebastian Bock, Nikolay Tcholtchev, Ilie-Daniel Gheorghe-Pop, and Manfred Hauswirth. Automatic generation of grover quantum oracles for arbitrary data structures. *Quantum Science and Technology*, 8(2):025003, jan 2023.
[25] John Watrous. IBM Quantum Learning: Grover's algorithm, 2024.
[26] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tappa. *Tight Bounds on Quantum Searching*, chapter 10, pages 187–199. John Wiley & Sons, Ltd, 1999.

**JAKUB BUCZKOWSKI** is a bachelor's student of computer science at the Faculty of Electronics, Telecommunications and Informatics of the Gdansk University of Technology since 2021.

**TOMASZ KOŹMIŃSKI** is a bachelor's student of computer science at the Faculty of Electronics, Telecommunications and Informatics of the Gdansk University of Technology since 2021.

**FILIP SZCZEPAŃSKI** is a bachelor's student of computer science at the Faculty of Electronics, Telecommunications and Informatics of the Gdansk University of Technology since 2021.

**MICHAŁ WILIŃSKI** is a bachelor's student of computer science at the Faculty of Electronics, Telecommunications and Informatics of the Gdansk University of Technology since 2021.

**TOMASZ P. STEFAŃSKI** (M'13, SM'22) received the M.Sc. degree in telecommunications and the Ph.D. degree in electronics engineering from the Gdansk University of Technology (GUT), Gdansk, Poland, in 2002 and 2007, respectively. Between 2009 and 2011, he was with ETH Zürich, Zürich, Switzerland, conducting research on parallelization of electromagnetic solvers on modern computing architectures using OpenCL programming language. Between 2006 and 2009, he was with the University of Glasgow, Glasgow, U.K., developing parallel alternating direction implicit finite-difference time-domain full-wave solvers for general purpose high-performance computers and graphics processing units. He is currently an Associate Professor with the Faculty of Electronics, Telecommunications, and Informatics, GUT. His current research interests include circuit theory, electromagnetics, mathematical modeling, control engineering and quantum computing.