XII International Workshop "Computer Algebra in Scientific Computing"
September 10, 2010, Tsakhkadzor, Armenia

Implementation of Some Quantum Algorithms with Mathematica

Vladimir P. Gerdt
Joint Institute for Nuclear Research, Dubna, Russia
E-mail: gerdt@jinr.ru
Alexander N. Prokopenya
Brest State Technical University, Belarus
E-mail: prokopenya@brest.by

Initialization

Content

- Motivation
- Circuit model for quantum computation
- Bernstein-Vazirani problem
- Quantum Fourier transformation
- The order-finding problem
- Conclusion

Motivation

Nowadays **quantum computation is a field of great interest**. It is sufficient to note that amount of publications devoted to different aspects of quantum computation grows exponentially. Main reason for this is a potential ability of a quantum computer to do certain computational task much more efficiently than it can be done by any classical computer (see: Nielsen M. and Chuang I. Quantum Computation and Quantum Information. Cambridge University Press, 2000).

Two the most famous examples of such calculations are Shor'salgorithm for efficient factorization of large integers and Grover'salgorithm of element search in an unsorted list.

There is some progress in developing hardware, as well (see, for example, D-Wave's 16-qubit quantum computer (from http://www.dwavesys.com/).

Two topical directions of investigations in quantum computation are
i) Development of the hardware to construct a realistic quantum computer that enables to test some known quantum algorithms;
ii) Searching for the problems which can be solved efficiently with a quantum computer and design the corresponding quantum algorithms.

As a realistic quantum computer is still not available it is expedient to use classical simulators of quantum computation to design new and to test known efficient quantum algorithms. One such simulator, namely, the *Mathematica* package "QuantumCircuit", we presented at the previous CASC'2009conference.
The main purpose of the present talk is to demonstrate how some quantum algorithms can be simulated with this package and to show that such simulation helps to understand ideas of quantum computation better.

Circuit model for quantum computation

Among several equivalent models of quantum computation the **quantum circuit model is the easiest to implement**, therefore this model is widely used for quantum algorithms. Quantum circuits are also **theoretically interesting as a tool for understanding the power and limitations of quantum computation.**

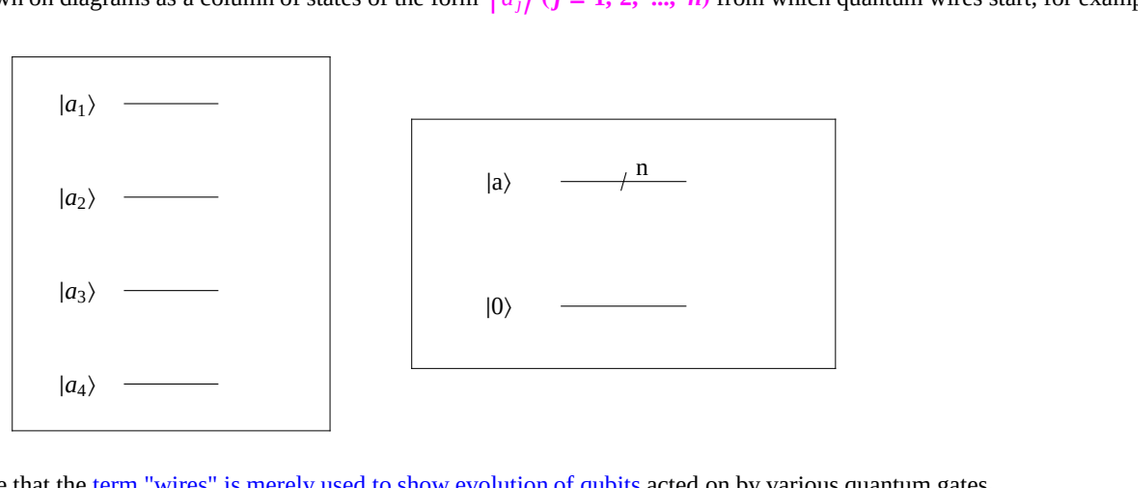
Quantum computation is composed of **three basic steps**:
(i) **preparation** of the input state of the memory register,
(ii) **implementation of the desired algorithm** (or desired unitary transformation acting on the memory register), and
(iii) **measurement of the output state**.

Elementary unit of quantum information is **a quantum bit** or **qubit** that is **a two-level quantum system**. It is assumed that a qubit can be prepared, manipulated and measured in a controlled way. The state of a qubit is denoted as $|a\rangle$ corresponding to standard Dirac notation for quantum mechanical states. Two possible states for a qubit are usually denoted as $|0\rangle$ and $|1\rangle$, which correspond to the states 0 and 1 for a classical bit. But in contrast to classical bits, qubit as a quantum system may exist not only in one of the states $|0\rangle$ or $|1\rangle$ but also in the state $|a\rangle$ being a superposition of these states

$$|a\rangle = \alpha|0\rangle + \beta|1\rangle,$$

where α and β are complex numbers constrained by the normalization condition $|\alpha|^2 + |\beta|^2 = 1$. Thus, the state of a qubit is represented by the vector $|a\rangle$ in the two-dimensional complex vector space, where the special states $|0\rangle$ and $|1\rangle$ form an orthonormal basis and are known as **computational basis states**.

A set of qubits forms a quantum memory register, where the input data and any intermediate results of computations are held. It is shown on diagrams as a column of states of the form $|a_j\rangle$ $(j = 1, 2, \ldots, n)$ from which quantum wires start, for example,



Note that the term "wires" is merely used to show evolution of qubits acted on by various quantum gates.

A system of n qubits has 2^n basis states. They are obtained as tensor product of basis states $|0\rangle$ and $|1\rangle$ associated with all n qubits

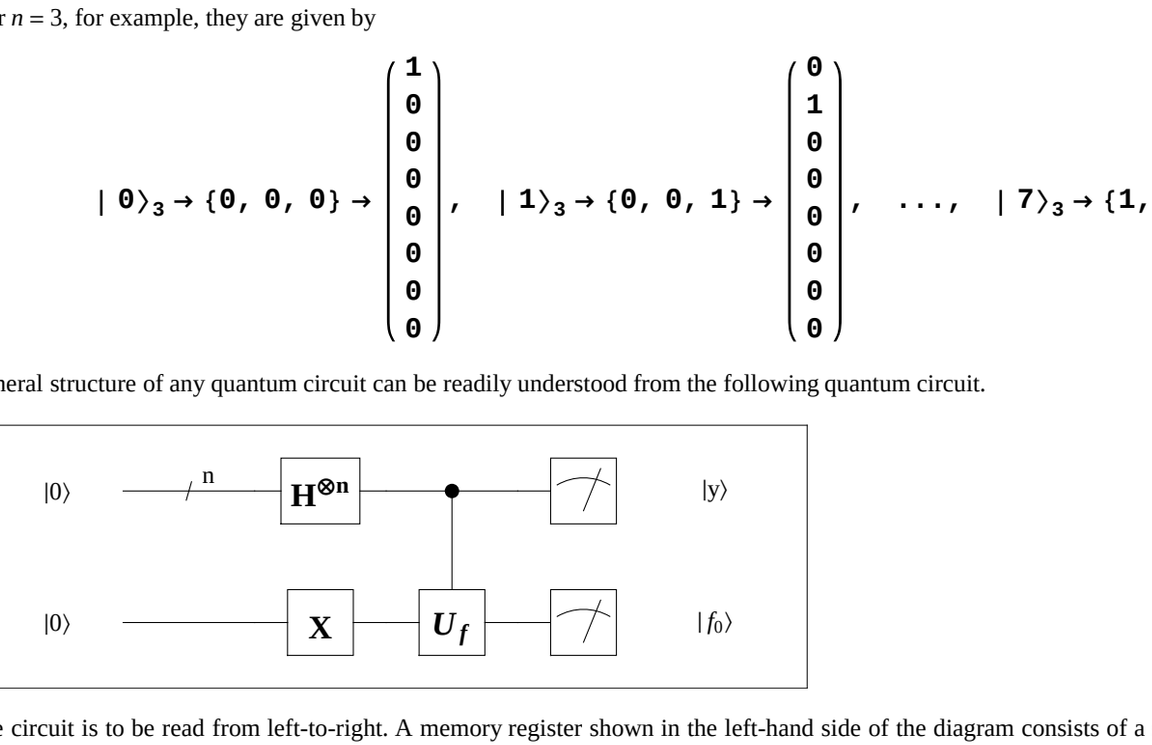
$$|a_1\rangle \otimes |a_2\rangle \otimes \ldots \otimes |a_n\rangle \equiv |a_1 a_2 \ldots a_n\rangle \equiv |a\rangle_n,$$

where the symbol $\otimes$ denotes a tensor product of vectors.

For $n = 3$, for example, they are given by

$$|0\rangle_3 \to (0,0,0) \to \begin{pmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{pmatrix},\quad |1\rangle_3 \to (0,0,1) \to \begin{pmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{pmatrix},\quad \ldots,\quad |7\rangle_3 \to (1,1,1) \to \begin{pmatrix}0\\0\\0\\0\\0\\0\\0\\1\end{pmatrix}$$

General structure of any quantum circuit can be readily understood from the following quantum circuit.



The circuit is to be read from left-to-right. A memory register shown in the left-hand side of the diagram consists of a set of n qubits and one ancillary qubit initially set in the states $|0\rangle_n$ and $|0\rangle$, respectively. Then we apply a sequence of quantum gates to different qubits and measure their final states afterwards, showing the result on the right-hand side of the diagram as the column of qubits $|y\rangle_n$ and $|f_y\rangle$.

Thus, **a quantum circuit can be understood as a device consisting of logical quantum gates that are arranged in the device according to steps in which the gates process qubits in time.**

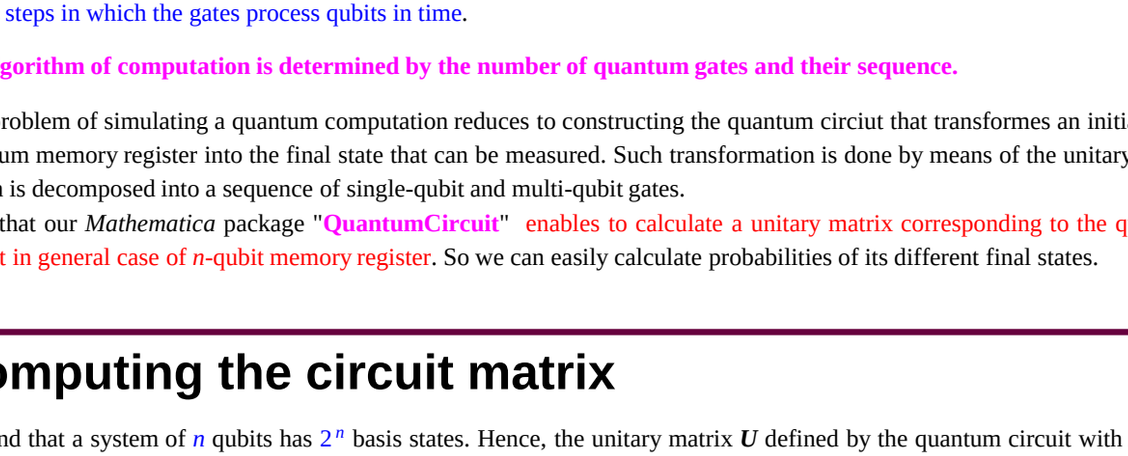
An algorithm of quantum computation is determined by the number of quantum gates and their sequence.

The problem of simulating a quantum computation reduces to constructing the quantum circuit that transforms an initial state of quantum memory register into the final state that can be measured. Such transformation is done by means of the unitary operator which is decomposed into a sequence of single-qubit and multi-qubit gates.
Note that our *Mathematica* package **"QuantumCircuit"** enables to calculate a unitary matrix corresponding to the quantum circuit in general case of n-qubit memory register. So we can easily calculate probabilities of its different final states.

Computing the circuit matrix

Remind that a system of n qubits has 2^n basis states. Then the unitary matrix U defined by the quantum circuit with n qubits may be represented as a $2^n \times 2^n$ matrix with respect to these basis states.

```
mat = (1 1 C 1 C 1
       1 C N C N 1 1
       H S 1 S' 1 S H ); circuit[mat, {a1, a2, a3}, {b1, b2, b3}]
```



As the circuit is read from left-to-right, and we use the matrix **mat** to represent the circuit, then the matrix U can be written as the following product

$$U = U_m . U_{m-1} \ldots U_1$$

where U_j $(j = 1, 2, \ldots, m)$ is the $2^n \times 2^n$ matrix defined by the quantum gates being in the jth column of the matrix **mat** and **m** is a number of columns.
(for details see: V.P. Gerdt, A.N. Prokopenya. *Some algorithms for calculating unitary matrices for quantum circuits.* Programming and Computer Software, Vol. 36, No. 2 (2010) 111 - 116)

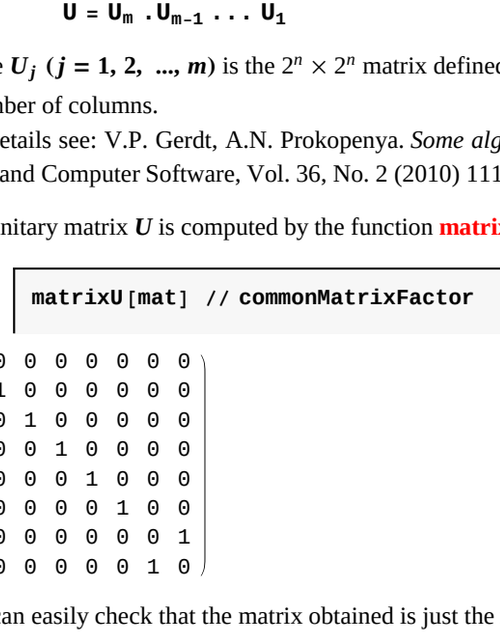
The unitary matrix U is computed by the function **matrixU[mat]**

```
matrixU[mat] // commonMatrixFactor
```

```
1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
```

One can easily check that the matrix obtained is just the matrix corresponding to the Toffoli gate.

```
mat1 = {{C}, {C}, {N}}; circuit[mat1, {a1, a2, a3}, {b1, b2, b3}]
```



```
{matrixU[mat] // MatrixForm, matrixU[mat1] // MatrixForm}
```

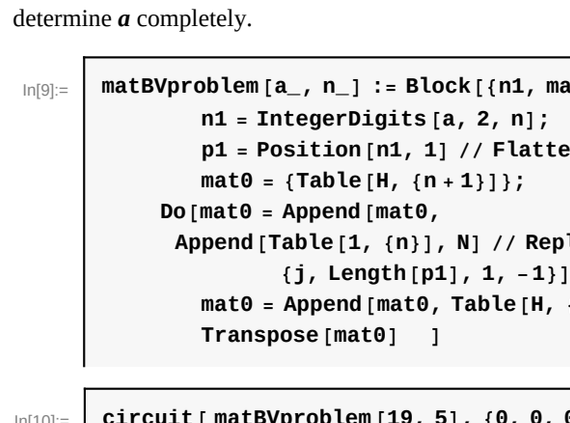
```
{ 1 0 0 0 0 0 0 0     1 0 0 0 0 0 0 0
  0 1 0 0 0 0 0 0     0 1 0 0 0 0 0 0
  0 0 1 0 0 0 0 0     0 0 1 0 0 0 0 0
  0 0 0 1 0 0 0 0  ,  0 0 0 1 0 0 0 0
  0 0 0 0 1 0 0 0     0 0 0 0 1 0 0 0
  0 0 0 0 0 1 0 0     0 0 0 0 0 1 0 0
  0 0 0 0 0 0 0 1     0 0 0 0 0 0 0 1
  0 0 0 0 0 0 1 0     0 0 0 0 0 0 1 0 }
```

The Bernstein-Vazirani problem

Let a be an unknown integer, $0 \le a < 2^n$. Let $f(x)$ take any other such integer x into the modulo-2 sum of the products of corresponding bits a and x

$$f(x) = a.x = a_1 x_1 \oplus a_2 x_2 \oplus \ldots \oplus a_n x_n$$

Suppose we have a subroutine that evaluates $f(x) = a.x$. The circuit implementing such unitary subroutine can be represented as a collection of the **controlled-Not** gates. Actually, action of the **CNOT** gate can be represented as

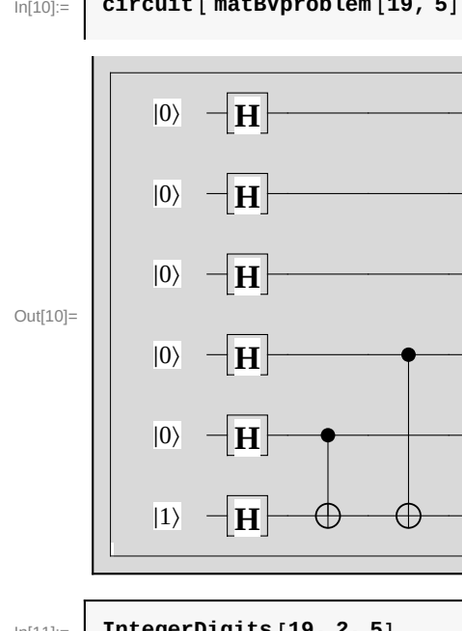


Hence, if $a = 0$ nothing happens and **CNOT** gate flips the state of the second qubit $|x\rangle$ if $a = 1$.

Considering a binary representation of the number a, for example, {1,1,0,0,1} corresponds to $a = 25$, one can conclude that the function

$$f(x) = a.x = x_1 \oplus x_2 \oplus x_5$$

can be implemented with the following quantum circuit.

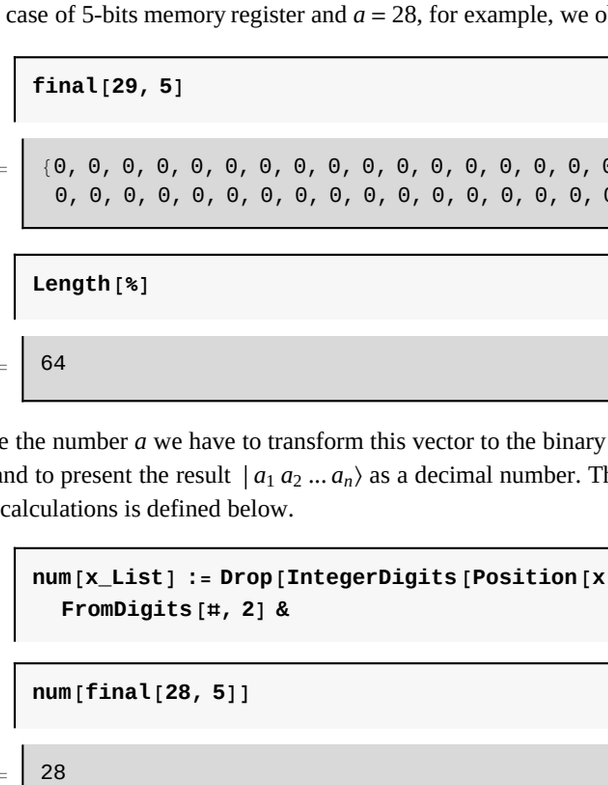


Suppose that the function is hidden in the black box and the question is how many times do we have to call the function to determine the value of a?

The mth bit of a is $a.2^m$, since the binary expansion of 2^m has 1 in position m and 0 in all the other positions. So with a classical computer we can learn the n bits of a by applying f to the n values $x = 2^m$, $0 \le m < n$. This, or any other classical method one can think of, requires n different invocations of the subroutine. But with a quantum computer a *single invocation* is enough to determine a completely.

```
matBVproblem[a_, n_] := Block[{n1, mat0, p1},
   n1 = IntegerDigits[a, 2, n];
   p1 = Position[n1, 1] // Flatten;
   mat0 = Table[H, {n + 1}];
  Do[mat0 = Append[mat0,
     Append[Table[1, {n}], N] // ReplacePart[#, p1[[j]] → C] &],
      {j, Length[p1], 1, -1}];
   mat0 = Append[mat0, Table[H, {n + 1}]];
   Transpose[mat0]  ]
```

```
circuit[matBVproblem[19, 5], {0, 0, 0, 0, 0, 1}, {x1, x2, x3, x4, x5, f1}]
```



```
IntegerDigits[19, 2, 5]
```

```
{1, 0, 0, 1, 1}
```

Initial state of the memory register $|00 \ldots 01\rangle$ is given by

```
initial[n_] := SparseArray[2 → 1, 2^(n+1)];
```

Its final state is obtained as a result of unitary transformation determined by the quantum circuit (function **matrixU**)

```
final[a_, n_] := matrixU[matBVproblem[a, n]].initial[n] // Normal
```

In the case of 5-bits memory register and $a = 28$, for example, we obtain

```
final[28, 5]
```

```
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0}
```

```
Length[%]
```

```
64
```

To see the number a we have to transform this vector to the binary representation $\{a_1, a_2, \ldots, a_n, a_{n+1}\}$, then to remove the last digit a_{n+1} and to present the result $\{a_1, a_2, \ldots, a_n\}$ as a decimal number. This is done with the function **num[a,n]** defined as follows

```
num[a_List] := Drop[IntegerDigits[Position[a, 1][[1, 1]] - 1, 2, Log[2, Length[a]]], -1] //
   FromDigits[#, 2] &
```

```
num[final[28, 5]]
```

```
28
```

Quantum Fourier Transform

Quantum Fourier Transform (QFT) plays a principal role in the development of efficient quantum algorithms. It is a unitary transformation whose action on the computational basis is given by

$$U_F|x\rangle_n = \frac{1}{2^{n/2}} \sum_{y=0}^{2^n-1} \exp(2\pi i \frac{x y}{2^n}) |y\rangle_n.$$

To generate a matrix modeling a quantum circuit implementing the Fourier transform for n-qubit memory register we define the following function

```
modelFourier[n_] := Module[{model, nn, n1},
   model = Array[mm, {n, n (n + 1) / 2 + Floor[n / 2]}] /. mm[i_, j_] → 1;
         n1 = 0;
   Do[    Do[If[k = 1, model[[j, n1 + 1]] = H,
             (model[[j, n1 + k]] = R_k; model[[j + k - 1, n1 + k]] = C)
             ], {k, n - j + 1}];
         n1 = n1 + n - j + 1,    {j, n}];
   Do[model[[j, n1 + 1]] = SW; model[[n - j + 1, n1 + 1]] = SW , {j, Floor[n / 2]}];
   model  ]
```

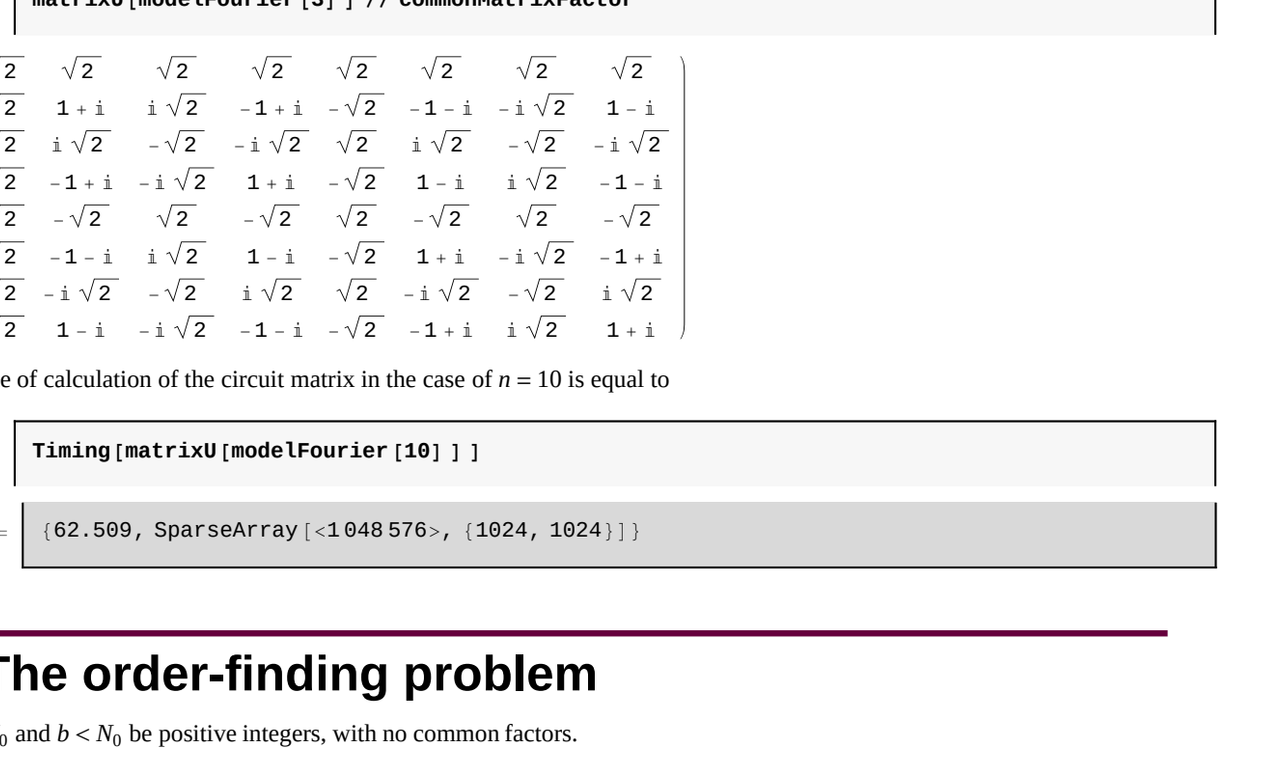
In the case of $n = 5$, for example, the corresponding matrix is given by

```
modelFourier[5] // MatrixForm
```

```
H R2 R3 R4 R5 1 1 1 1 1 1 1 1 1 1 SW 1
1 C  1  1  1  H R2 R3 R4 1 1 1 1 1 1 1 SW
1 1  C  1  1  1 C  1  1  H R2 R3 1 1 1 1 1
1 1  1  C  1  1 1  C  1  1 C  1  H R2 1 1 SW
1 1  1  1  C  1 1  1  C  1 1  C  1 C  H 1 SW
```

Of course, we can easily visualize the corresponding quantum circuit with function **circuit**.

```
circuit[modelFourier[4], {a1, a2, a3, a4}, {b1, b2, b3, b4}]
```



```
matrixU[modelFourier[3]] // commonMatrixFactor
```

$$\frac{1}{4}\begin{pmatrix}
\sqrt{2} & \sqrt{2} & \sqrt{2} & \sqrt{2} & \sqrt{2} & \sqrt{2} & \sqrt{2} & \sqrt{2} \\
\sqrt{2} & 1+i & i\sqrt{2} & -1+i & -\sqrt{2} & -1-i & -i\sqrt{2} & 1-i \\
\sqrt{2} & i\sqrt{2} & -\sqrt{2} & -i\sqrt{2} & \sqrt{2} & i\sqrt{2} & -\sqrt{2} & -i\sqrt{2} \\
\sqrt{2} & -1+i & -i\sqrt{2} & 1+i & -\sqrt{2} & 1-i & i\sqrt{2} & -1-i \\
\sqrt{2} & -\sqrt{2} & \sqrt{2} & -\sqrt{2} & \sqrt{2} & -\sqrt{2} & \sqrt{2} & -\sqrt{2} \\
\sqrt{2} & -1-i & i\sqrt{2} & 1-i & -\sqrt{2} & 1+i & -i\sqrt{2} & -1+i \\
\sqrt{2} & -i\sqrt{2} & -\sqrt{2} & i\sqrt{2} & \sqrt{2} & -i\sqrt{2} & -\sqrt{2} & i\sqrt{2} \\
\sqrt{2} & 1-i & -i\sqrt{2} & -1-i & -\sqrt{2} & -1+i & i\sqrt{2} & 1+i
\end{pmatrix}$$

A time calculation of the circuit matrix in the case of $n = 10$ is equal to

```
Timing[matrixU[modelFourier[10]]]
```

```
{62.509, SparseArray[<1048576>, {1024, 1024}]}
```

The order-finding problem

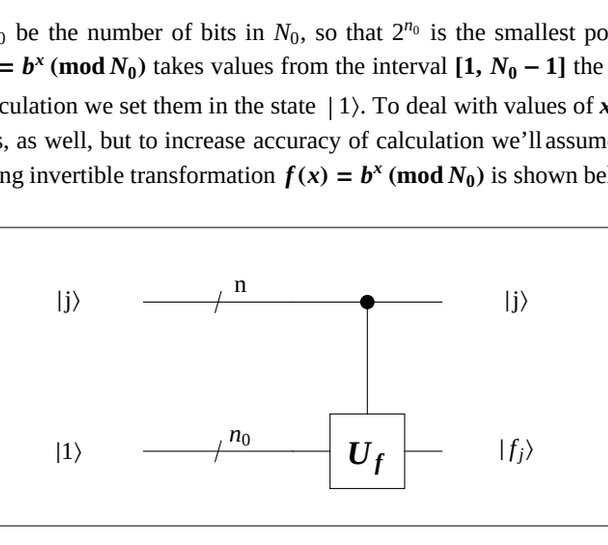
Let N_0 and $b < N_0$ be positive integers, with no common factors.
The order of b modulo N_0 is defined to be the **smallest positive integer**, r, such that

$$b^r \pmod{N_0} = 1.$$

The **order-finding problem is to determine the order for some specified b and N_0**. This problem is believed to be a hard problem on a classical computer, in the sense that no algorithm is known to solve the problem using resources polynomial in the $O(n_0)$ bits needed to specify the problem, where n_0 is the number of bits needed to specify N_0.

Note that the function $f(x) = b^x \pmod{N_0}$ is periodic and the order r is just its period. And one might think that it should not be very difficult to find the period of such a periodic function. But the problem is that this function is defined on the integers and its values within a period r are virtually random from one integer to the next, and therefore give no hint of the value of r.
As an example, let us consider two coprime integers $N_0 = 1987$, $b = 709$ and visualize the corresponding function $f(x) = 709^x \pmod{1987}$.

```
data1a = Table[Mod[709^k, 1987], {k, 0, 1000}];
ListPlot[data1a, Filling → Axis, BaseStyle → {FontSize → 12}]
```



Obviously, it is quite difficult to find a period on this graph although it is equal to 993.

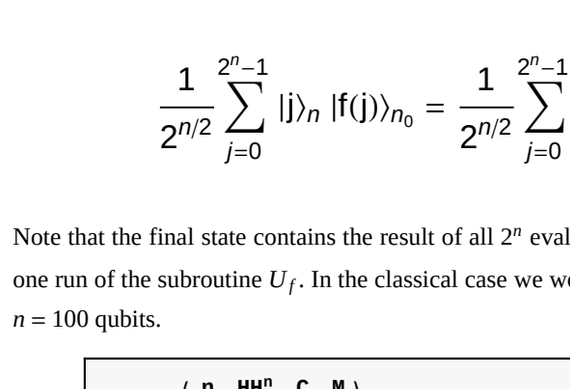
```
Position[data1a, 1]
```

```
{{1}, {994}}
```

The best known classical algorithms for finding the period of such a function take a time that grows exponentially with $n_0^{1/3}$. But in 1994 Peter Shor discovered a quantum algorithm to find the period r, in a time that grows only a little bit faster than n_0^3. Note that this discovery is of considerable practical interest because the ability to find periods efficiently, combined with some number-theoretic tricks, enables one to factor efficiently the product of two large prime numbers.
Let us consider the main steps in implementation of the corresponding quantum algorithm.

Step 1.

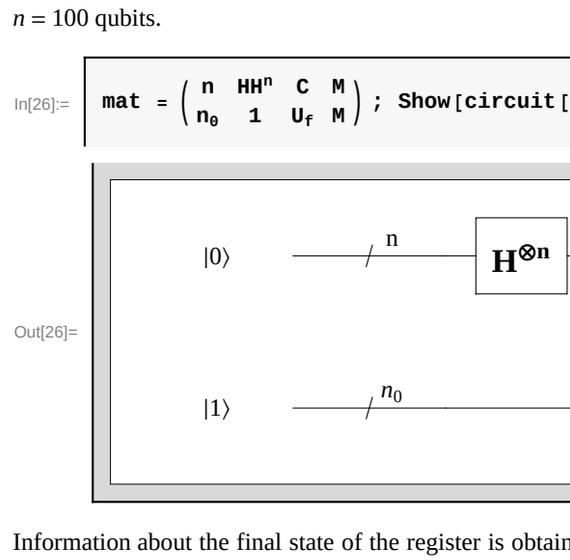
Let n_0 be the number of bits in N_0, so that 2^{n_0} is the smallest power of 2 that exceeds N_0 $(N_0 < 2^{n_0})$. In order to calculate $f(x) = b^x \pmod{N_0}$ values from the interval $[1, N_0 - 1]$ the **output register** must **contain n_0 qubits** and a number of qubits, n, in input register, to increase the accuracy of calculation we'll assume that it **contains $n > n_0$ qubits**. The quantum circuit implementing invertible transformation $f(x) = b^x \pmod{N_0}$ is shown below



$$|j\rangle_n |1\rangle_{n_0} \to |j\rangle_n |b^j \bmod N_0\rangle_{n_0}$$

Note that a subroutine U_f for calculation of $b^x \pmod{N_0}$ can be realized efficiently with a quantum computer.

An advantage of a quantum computer over its classical counterpart is that we can transform the input register to the state being an equally weighted superposition of all possible basis states. To do this it is sufficient to apply the Hadamard gate to every qubit of the input register, initialized initially in the standard state $|0\rangle$.



$$|0\rangle_n |1\rangle_{n_0} \to \frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} |j\rangle_n |1\rangle_{n_0}$$

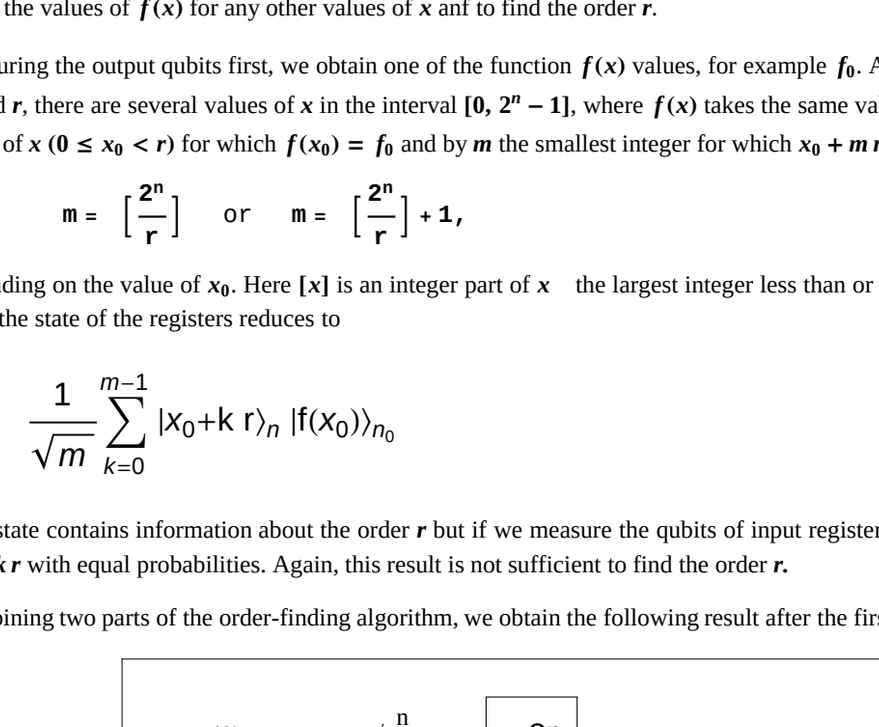
Note that here we need only n Hadamard gates.

Applying then the subroutine U_f to this superposition and taking into account a linearity of U_f, we obtain

$$\frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} |j\rangle_n |f(j)\rangle_{n_0} = \frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} |j\rangle_n |b^j \bmod N_0\rangle_{n_0}$$

Note that the final state contains the result of all 2^n evaluations of the function $f(x) = b^x \pmod{N_0}$ and it is obtained after only one run of the subroutine U_f. In the classical case we would have to calculate this function 2^n times, what is about $2^{100} \approx 10^{30}$ for $n = 100$ qubits.

```
mat = ( n  H⊗n  C  M
        n0  1   Uf M ); Show[circuit[mat, {0, 1}, {y, f0}], Background → White]
```



Information about the final state of the register is obtained by means of measuring it. If we apply the measurement gates to the qubits of input register we obtain with equal probability one of the values of x from the interval $[0, 2^n - 1]$ because all states in the superposition above have the same coefficients. Measuring qubits of the output register will give the corresponding value of $f(x)$. As a result of measurement, the state of the registers reduces to $|x\rangle_n |f(x)\rangle_{n_0}$ and we are no longer able to learn anything about the values of $f(x)$ for any other values of x and to find the order r.

Measuring the output qubits first, we obtain one of the function $f(x)$ values, for example f_0. As this function is periodic with the period r, there are several values of x in the interval $[0, 2^n - 1]$, where $f(x)$ takes the same value f_0. Denoting by x_0 the smallest value of x $(0 \le x_0 < r)$ for which $f(x_0) = f_0$ and by m the smallest integer for which $x_0 + m r \ge 2^n$, we find that

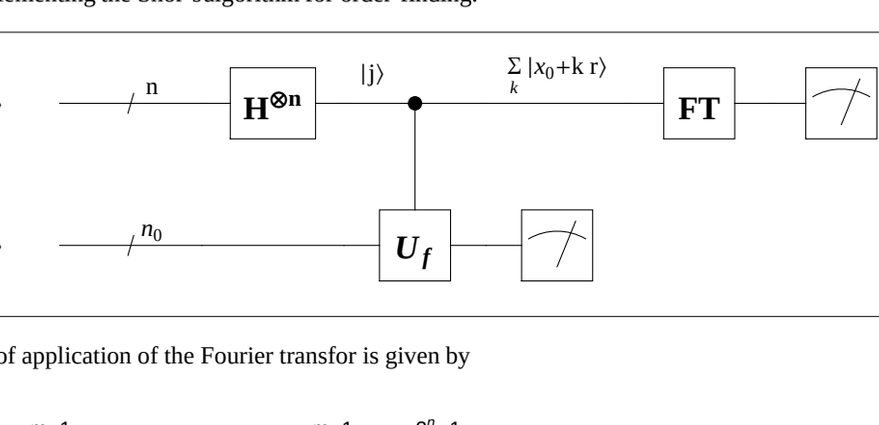
$$m = \left[\frac{2^n}{r}\right] \quad \text{or} \quad m = \left[\frac{2^n}{r}\right] + 1,$$

depending on the value of x_0. Here $[x]$ is an integer part of x — the largest integer less than or equal to x. As a result of measurement the state of the registers reduces to

$$\frac{1}{\sqrt{m}} \sum_{k=0}^{m-1} |x_0 + k\, r\rangle_n |f(x_0)\rangle_{n_0}$$

This state contains information about the order r but if we measure the qubits of input register we obtain only one of the values $x_0 + k r$ with equal probabilities. Again, this result is not sufficient to find the order r.

Combining two parts of the order-finding algorithm, we obtain the following result after the first step.

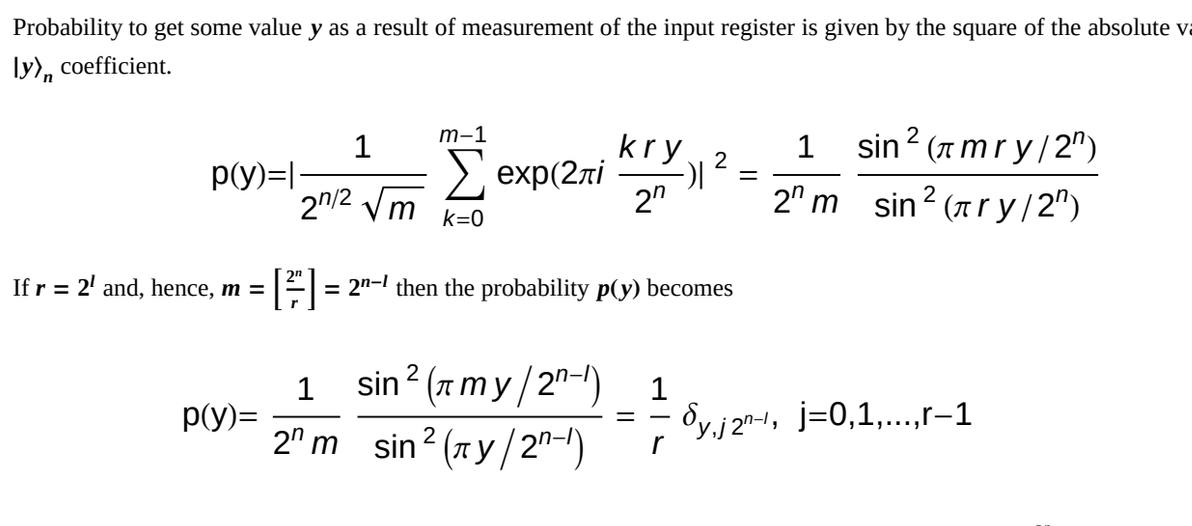


$$|0\rangle_n |1\rangle_{n_0} \to \frac{1}{\sqrt{m}} \sum_{k=0}^{m-1} |x_0 + k\, r\rangle_n |f(x_0)\rangle_{n_0}$$

Note that repeating the calculation determined by the circuit above we obtain similar final state of the registers but with different x_0. So we have to construct such unitary transformation which gives us a possibility to avoid the x_0 dependence of the result of measurement. This is accomplished with the ***quantum Fourier transform***.

Step 2.

Adding the quantum Fourier transform of the input register and its measurement afterwards, we obtain the following quantum circuit implementing the Shor'salgorithm for order-finding.



The result of application of the Fourier tranform is given by

$$\frac{1}{\sqrt{m}} \sum_{k=0}^{m-1} |x_0 + k\, r\rangle_n \xrightarrow{U_F} \frac{1}{\sqrt{m}} \sum_{k=0}^{m-1} \frac{1}{2^{n/2}} \sum_{y=0}^{2^n-1} \exp(2\pi i \frac{(x_0 + k\, r) y}{2^n}) |y\rangle_n \to$$

$$\to \sum_{y=0}^{2^n-1} \exp(2\pi i \frac{x_0\, y}{2^n}) \frac{1}{2^{n/2}\sqrt{m}} \sum_{k=0}^{m-1} \exp(2\pi i \frac{k\, r\, y}{2^n}) |y\rangle_n$$

We see that a random value x_0 appears only in the phase factor and it doesn'tinfluence on the result of measurement.

Probability to get some value y as a result of measurement of the input register is given by the square of the absolute value of the $|y\rangle_n$ coefficient.

$$p(y) = \left|\frac{1}{2^{n/2}\sqrt{m}} \sum_{k=0}^{m-1} \exp(2\pi i \frac{k\, r\, y}{2^n})\right|^2 = \frac{1}{2^n m} \frac{\sin^2(\pi m r y / 2^n)}{\sin^2(\pi r y / 2^n)}$$

If $r = 2^l$ and, hence, $m = \left[\frac{2^n}{r}\right] = 2^{n-l}$ then the probability $p(y)$ becomes

$$p(y) = \frac{1}{2^n m} \frac{\sin^2(\pi m y / 2^{n-l})}{\sin^2(\pi y / 2^{n-l})} = \frac{1}{r} \delta_{y, j\, 2^{n-l}}, \quad j = 0, 1, \ldots, r-1$$

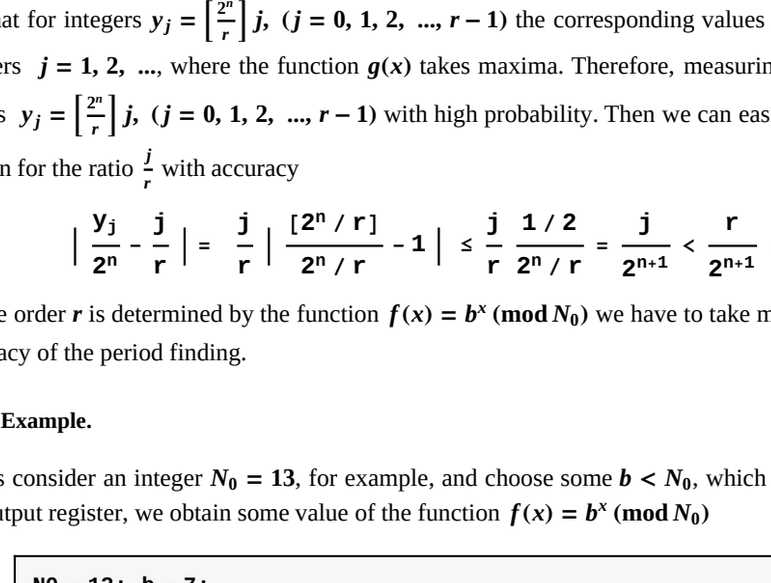
It means that measuring the final state of the input register we can obtain only some multiple of $2^{n-l} = \frac{2^n}{r}$ with equal probabilities $\frac{1}{r}$. Then we can easily calculate $r = \frac{2^n}{y} j$, $(j = 0, 1, \ldots, r-1)$ with a classical computer and choose such value of the factor j for which $b^r \pmod{N_0} = 1$.
But probability for the period r to be a power of 2 is extremely small and usually the ratio $2^n/r$ turns out to be a rational number.

To estimate probabilities of different final states of the input register let us consider the function

$$g(x) = \frac{1}{m} \frac{\sin^2(\pi m x)}{\sin^2(\pi x)},$$

where $x = r\, y / 2^n$. The corresponding graph is shown below.

```
Manipulate[Plot[Sin[π m x]^2 / (m Sin[π x]^2), {x, 0, 1.2}, PlotRange → All, PlotStyle → {Thick}],
 {m, 1, 20, 1}, ControlPlacement → Top]
```



Obviously, the function is periodic with the period 1. It takes maximum values $g = m$ for $x = 0, 1, 2, \ldots$ and has $m - 1$ zero between each pair of the neighbouring maxima in the points $x = \frac{1}{m}, \frac{2}{m}, \ldots, \frac{m-1}{m}, \frac{m+1}{m}, \ldots$. Assuming $r \ll 2^n$, one can readily see that for integers $y_j = \left[\frac{2^n}{r} j\right]$, $(j = 0, 1, 2, \ldots, r-1)$ the corresponding values $x_j = \frac{r\, y_j}{2^n} = \frac{[2^n j/r]}{2^n/r}$ will be very close to the integers $j = 1, 2, \ldots$, where the function $g(x)$ takes maxima. Therefore, measuring the input register will give us one of the values $y_j = \left[\frac{2^n}{r} j\right]$, $(j = 0, 1, 2, \ldots, r-1)$ with high probability. Then we can easily calculate the ratio $\frac{y_j}{2^n}$ which gives approximation for the ratio $\frac{j}{r}$ with accuracy

$$\left|\frac{y_j}{2^n} - \frac{j}{r}\right| = \frac{1}{r}\left|\frac{[2^n j / r]}{2^n / r} - j\right| \le \frac{1}{r}\frac{1/2}{2^n/r} = \frac{1}{2^{n+1}} < \frac{1}{r^2},$$

As the order r is determined by the function $f(x) = b^x \pmod{N_0}$ we have to take more qubits in the input register to increase the accuracy of the period finding.

Example.

Let us consider an integer $N_0 = 13$, for example, and choose some $b < N_0$, which has no common factors with N_0. Measuring the output register, we obtain some value of the function $f(x) = b^x \pmod{N_0}$

```
N0 = 13; b = 7;
data1a = Table[Mod[b^k, N0], {k, 0, 2^5 - 1}];
f0a = data1a[[RandomInteger[2^5 - 1]]]
```

```
11
```

Then we define the state of the input register after measurement and construct the corresponding normalized vector in the 2^n dimensional space.

```
pos1 = Position[data1a, f0a];
initial1 = SparseArray[Table[pos1[[j, 1]] → 1/Sqrt[Length[pos1]], {j, Length[pos1]}], 2^5];
```

```
initial1 // Normal
```

$$\{0, 0, 0, 0, 0, \frac{1}{\sqrt{3}}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \frac{1}{\sqrt{3}}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \frac{1}{\sqrt{3}}, 0, 0\}$$

The next step is to apply the Fourier transform to this vector.

```
FTransform1 = matrixU[modelFourier[5]];
finalFourier1 = FTransform1.initial1;
```

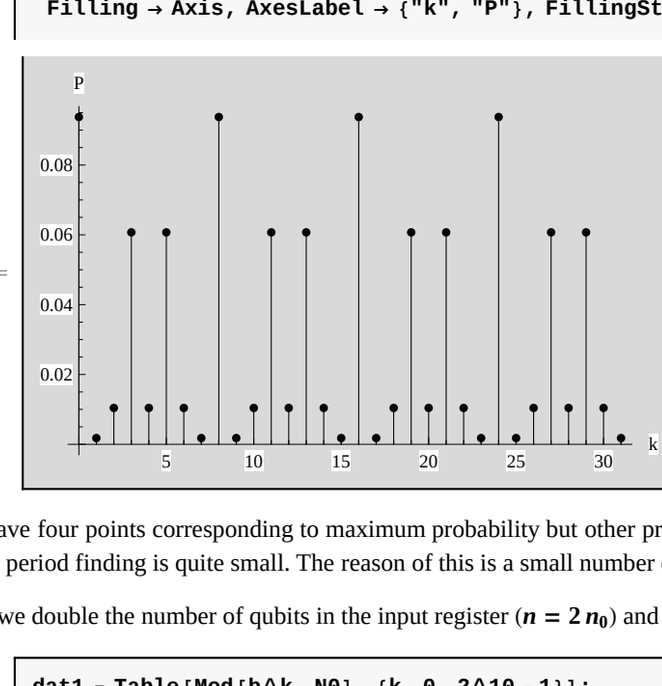
Square of the absolute value of each component of the vector obtained after the Fourier transform gives a probability to get the corresponding state as a result of measurement of the input register. One can easily check that total probability is equal to 1.

```
finalFourier1.Conjugate[finalFourier1] // N[#, 25] &
```

```
1.000000000000000000000000 + 0. × 10^-25 i
```

Now we can visualize the probability distribution of different final states of the input register.

```
data2a = Table[{j - 1, finalFourier1[[j]] Conjugate[finalFourier1[[j]]]},
   {j, Length[finalFourier1]}];
ListPlot[data2a, PlotStyle → {Black, PointSize[0.015]}, PlotRange → All,
 Filling → Axis, AxesLabel → {"k", "P"}, FillingStyle → Black]
```



We have four points corresponding to maximum probability but other probabilities are also quite high what means that accuracy of the period finding is quite small. The reason of this is a small number of qubits in the input register $(n = n_0)$.

Now we double the number of qubits in the input register $(n = 2 n_0)$ and repeat the calculation for the same numbers N_0 and b.

```
dat1 = Table[Mod[b^k, N0], {k, 0, 2^10 - 1}];
f0 = dat1[[RandomInteger[2^10 - 1]]]
```

```
3
```

Again, the result of measurement of the output register is chosen randomly and the input register jumps to the state ***initial***.

```
pos = Position[dat1, f0];
initial = SparseArray[Table[pos[[j, 1]] → 1/Sqrt[Length[pos]], {j, Length[pos]}], 2^10];
```

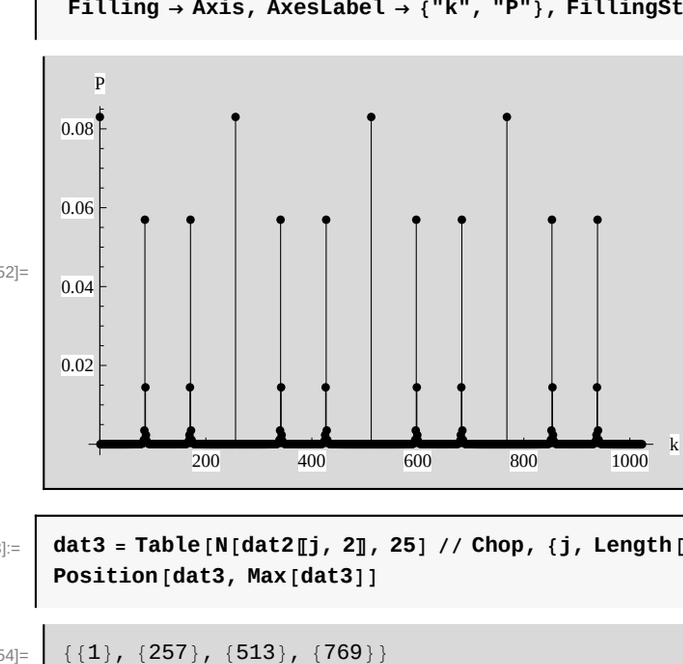
Then we apply Fourier transform to the input qubits and measure the result.

```
FTransform = matrixU[modelFourier[10]];
finalFourier = FTransform.initial;
```

```
finalFourier.Conjugate[finalFourier] // N[#, 25] &
```

```
1.000000000000000000000000 + 0. × 10^-25 i
```

```
dat2 = Table[
   {j - 1, finalFourier[[j]] Conjugate[finalFourier[[j]]]}, {j, Length[finalFourier]}];
ListPlot[dat2, PlotStyle → {Black, PointSize[0.015]}, PlotRange → All,
 Filling → Axis, AxesLabel → {"k", "P"}, FillingStyle → Black]
```



```
dat3 = Table[N[dat2[[j, 2]], 25] // Chop, {j, Length[finalFourier]}];
Position[dat3, Max[dat3]]
```

```
{{1}, {257}, {513}, {769}}
```

```
Rationalize[256. / 2^10, 0.01]
```

$$\frac{1}{4}$$

```
Table[{4 k, Mod[b^(4 k), N0]}, {k, 0, 10}]
```

```
{{0, 1}, {4, 9}, {8, 3}, {12, 1}, {16, 9},
 {20, 3}, {24, 1}, {28, 9}, {32, 3}, {36, 1}, {40, 9}}
```

Now we see that period is equal to 12. It is correct number what can be seen from the set of the function values.

```
Position[dat1, 1]
```

```
{{1}, {13}, {25}, {37}, {49}, {61}, {73}, {85}, {97}, {109}, {121}, {133}, {145}, {157},
 {169}, {181}, {193}, {205}, {217}, {229}, {241}, {253}, {265}, {277}, {289}, {301},
 {313}, {325}, {337}, {349}, {361}, {373}, {385}, {397}, {409}, {421}, {433}, {445},
 {457}, {469}, {481}, {493}, {505}, {517}, {529}, {541}, {553}, {565}, {577}, {589},
 {601}, {613}, {625}, {637}, {649}, {661}, {673}, {685}, {697}, {709}, {721}, {733},
 {745}, {757}, {769}, {781}, {793}, {805}, {817}, {829}, {841}, {853}, {865}, {877},
 {889}, {901}, {913}, {925}, {937}, {949}, {961}, {973}, {985}, {997}, {1009}, {1021}}
```

Conclusion

In the present talk we have demonstrated application of our *Mathematica* package to simulation of quantum circuits.
Note that the package provides a user-friendly graphical interface to specify a quantum circuit, to draw it, and to construct the corresponding unitary matrix for quantum computation defined by the circuit. The matrix is computed by means of the linear algebra tools built into *Mathematica*.
As a result we can evaluate probability of different results after measurement and to check different quantum algorithms.