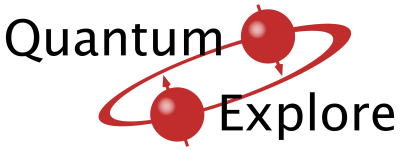
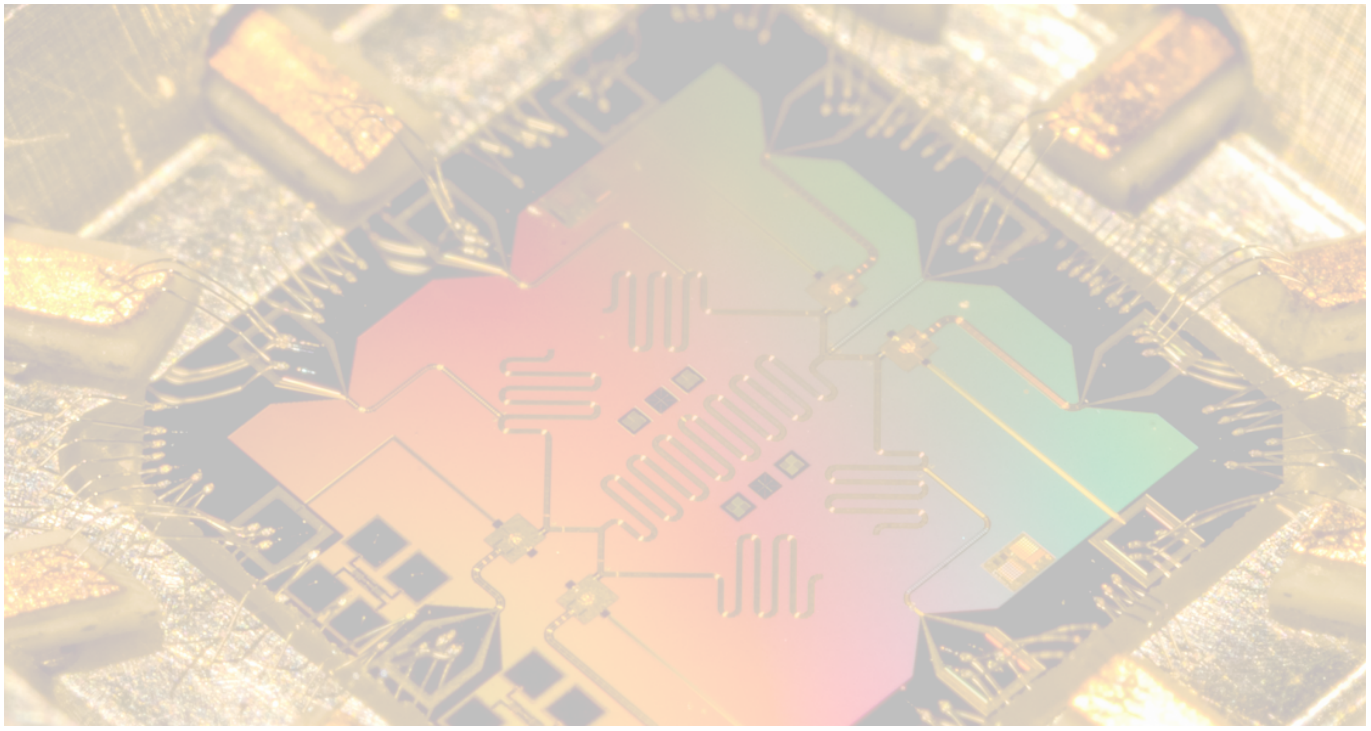


Open Quantum Software Development System



Jesús Lacalle

1. Introduction

The **Open Quantum Software Development System** (OpenQSDS) is developed in Maxima. It is a flexible system for the study, design and programming of quantum algorithms. The version presented in this manual, **OpenQSDS 00.0**, is the first and includes the basic functions of the system core.

2. Introduction to Maxima

Maxima, a Computer Algebra System, is a descendant of Macsyma, the legendary computer algebra system developed in the late 1960s at the Massachusetts Institute of Technology. It is the only system based on that effort still **publicly available and with an active user community**, thanks to its **open source nature**. Macsyma was revolutionary in its day, and many later systems, such as Maple and Mathematica, were inspired by it.

The Maxima source code can be compiled on many systems, including Windows, Linux, and MacOS X. The source code for all systems and precompiled binaries for Windows and Linux are available at the SourceForge file manager.

3. Maxima and wxMaxima versions

Maxima's ability to work symbolically provides us with a very useful tool for working in Quantum Computing. This feature together with its open source philosophy make Maxima the ideal system to develop an Open Quantum Software Development System with little work.

The latest versions of the free software are the following:

Version of Maxima	Version of wxMaxima
5.45.1	21.11.0

4. OpenQSDS 00.0 Library

The first version of the OpenQSDS system only includes the core functions of the system. Despite this, the system allows the development of quantum algorithms and the analysis of the results at any time.

4.1 Scalar product and tensor product

1. Scalar product with complex numbers. Example:

```
scalar_prod([1, %i, 2, -%i], [1, 1, 1, %i]);
```

$$2 - i$$

2. Tensor product. Example:

```
tensor_prod( matrix([1, 1], [1, -1]),
             matrix([1, 2], [3, 4])      );
```

$$\begin{pmatrix} 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \\ 1 & 2 & -1 & -2 \\ 3 & 4 & -3 & -4 \end{pmatrix}$$

4.2 Quantum computing module

4.2.1 Instructions for using the OpenQSDS

1. Load the OpenQSDS 00.0 by executing the following Maxima instruction:

```
batchload("OpenQSDS_000-code.wxm");
```

2. Start an OpenQSDS session.

3. Initialize the simulator by calling the **start_simulator**(n) function where n is the number of qubits we want to work with.

4. Apply one by one the quantum gates and quantum measurements of the algorithm.

   a) List of defined quantum gates:

      1) Pauli matrices:

gate_I (Identity) $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

gate_X (Pauli X) $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

gate_Y (Pauli Y) $\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$

gate_Z (Pauli Z) $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

2) Hadamard quantum gate:

$$\textbf{gate\_H} \quad \text{(Hadamard)} \quad \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

3) Useful quantum gates:

$$\begin{array}{lll} \textbf{gate\_S} & \text{(Phase)} & \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \\ \textbf{gate\_T} & (\pi/8-\text{phase}) & \begin{pmatrix} 1 & 0 \\ 0 & \frac{1+i}{\sqrt{2}} \end{pmatrix} \\ \textbf{gate\_R}(k) & (\pi/2^{k-1}-\text{phase}) & \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/2^{k-1}} \end{pmatrix} \\ \textbf{gate\_P}(\varphi) & (\varphi-\text{phase}) & \begin{pmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{pmatrix} \end{array}$$

4) One-qubit quantum gate factors:

$$\begin{array}{lll} \textbf{gate\_Ry}(\theta) & (\theta - \hat{y} \text{ rotation}) & \begin{pmatrix} \cos(\theta/2) & \sin(\theta/2) \\ -\sin(\theta/2) & \cos(\theta/2) \end{pmatrix} \\ \textbf{gate\_Rz}(\alpha) & (\alpha - \hat{z} \text{ rotation}) & \begin{pmatrix} e^{i\alpha/2} & 0 \\ 0 & e^{-i\alpha/2} \end{pmatrix} \\ \textbf{gate\_Ph}(\delta) & (\delta\text{-global phase}) & \begin{pmatrix} e^{i\delta} & 0 \\ 0 & e^{i\delta} \end{pmatrix} \end{array}$$

5) One-qubit quantum gate error $\textbf{gate\_Error}(\theta_0, \theta_1, \theta_2)$:

$$\begin{pmatrix} \cos(\theta_0) + i\sin(\theta_0)\cos(\theta_1) & -\sin(\theta_0)\sin(\theta_1)\cos(\theta_2) + i\sin(\theta_0)\sin(\theta_1)\sin(\theta_2) \\ \sin(\theta_0)\sin(\theta_1)\cos(\theta_2) + i\sin(\theta_0)\sin(\theta_1)\sin(\theta_2) & \cos(\theta_0) - i\sin(\theta_0)\cos(\theta_1) \end{pmatrix}$$

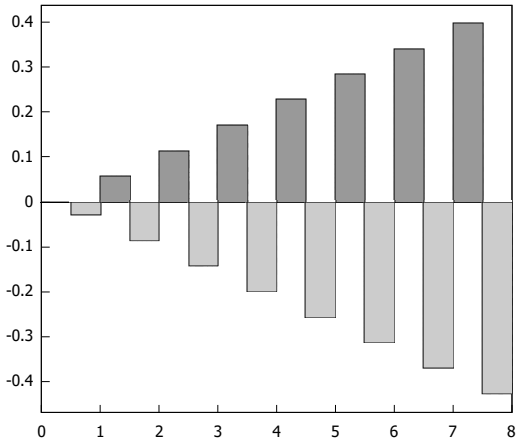
6) Two-qubit quantum gate:

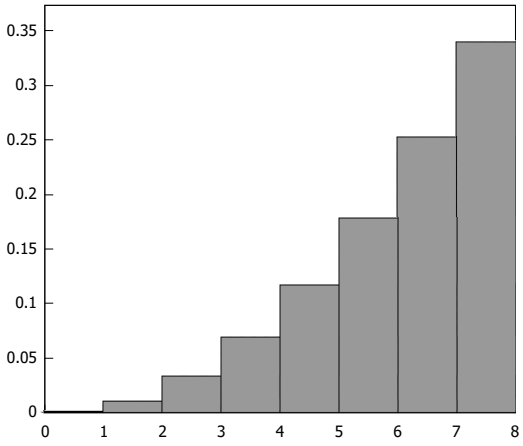
$$\textbf{gate\_swap} \quad \text{(Swap)} \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

b) List of functions for applying quantum gates and quantum measurements. The examples will apply to the following 3−qubit:

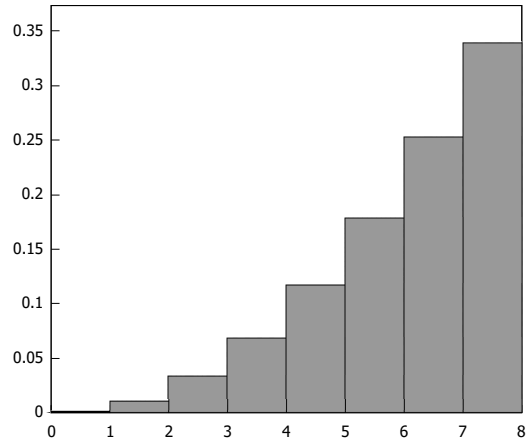
$$\frac{1}{\sqrt{1240}}\Big(- i|000\rangle + (2-3i)|001\rangle + (4-5i)|010\rangle + (6-7i)|011\rangle + (8-9i)|100\rangle + (10-11i)|101\rangle + (12-13i)|110\rangle + (14-15i)|111\rangle \Big)$$

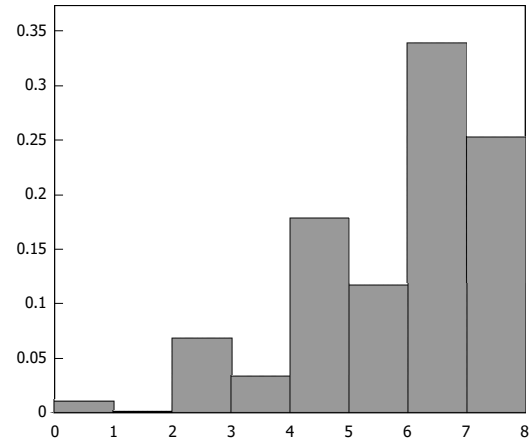
Coordinates and probabilities:



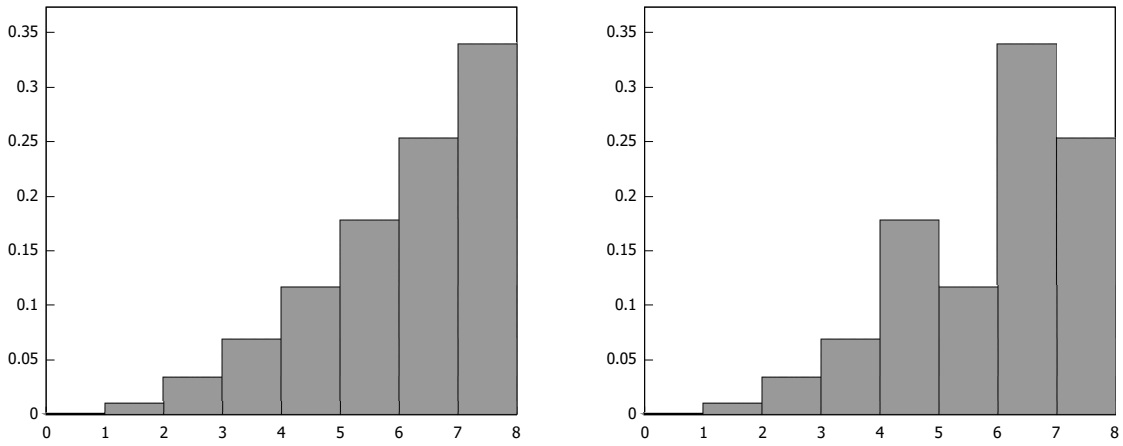


apply_1_qubit_gate(G, j). The one-qubit quantum gate G is applied to qubit j.
Example: apply_1_qubit_gate(gate_X,3)$

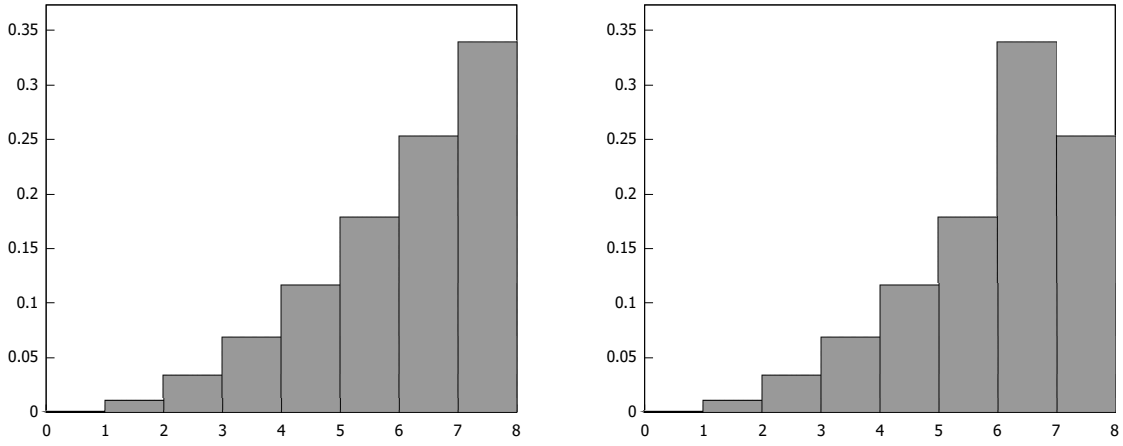




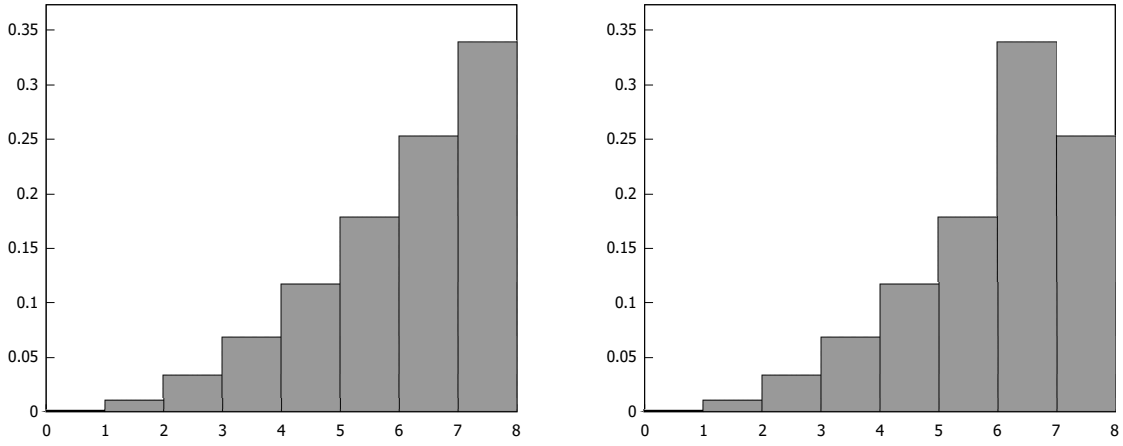
apply_1_controlled_1_qubit_gate$(G, j0, j)$. The one-qubit quantum gate G is applied to qubit j if qubit $j0$ is in state $|1\rangle$.
Example: apply_1_controlled_1_qubit_gate(gate_X,1,3)$



apply_2_controlled_1_qubit_gate$(G, j0, j1, j)$. The one-qubit quantum gate G is applied to qubit j if qubits $j0$ and $j1$ are in state $|1\rangle$.
Example: apply_2_controlled_1_qubit_gate(gate_X,1,2,3)$

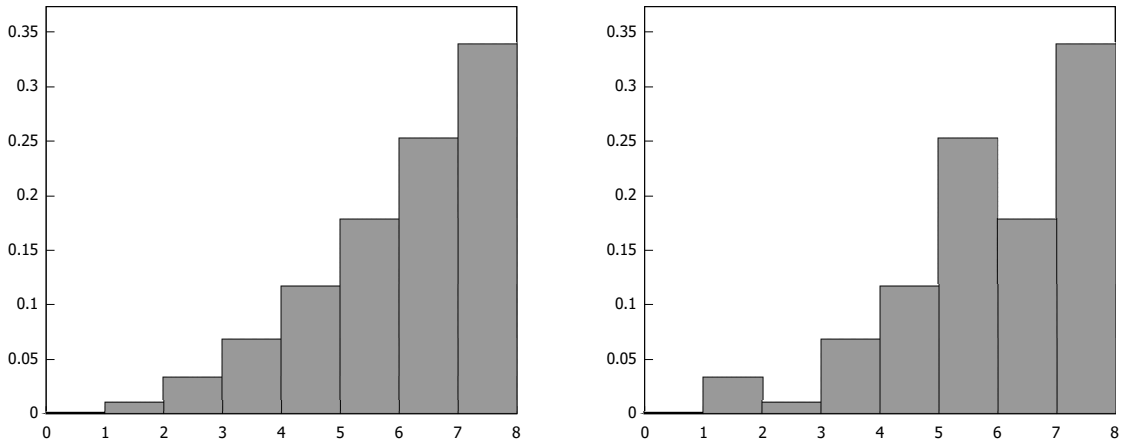


apply_k_controlled_1_qubit_gate(G, L, j). The one-qubit quantum gate G is applied to qubit j if qubits $L[1]$, $L[2]$, ... are in state $|1\rangle$.
Example: apply_k_controlled_1_qubit_gate(gate_X,[2,1],3)$



apply_2_qubit_gate$(G, j1, j2)$. The two-qubit quantum gate G is applied to qubits $j1$ and $j2$.

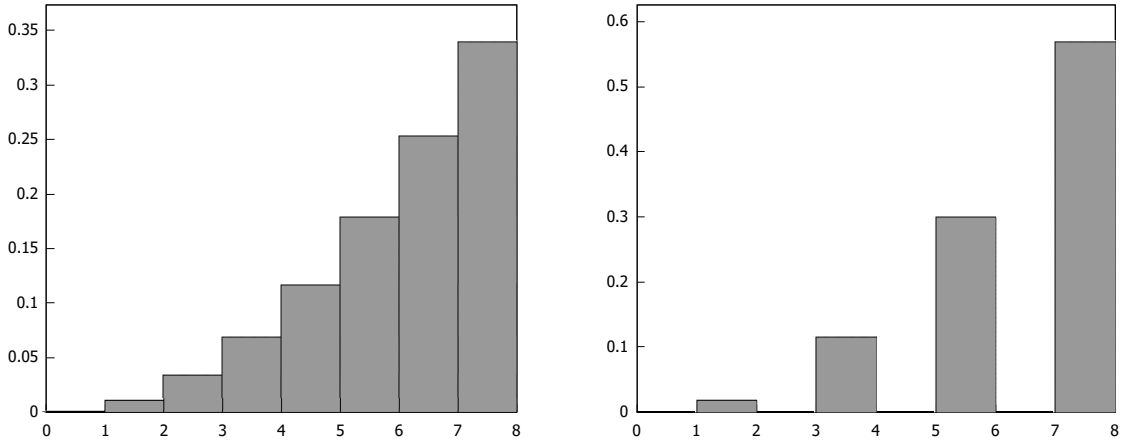
Example: apply_2_qubit_gate(gate_swap,2,3)$



apply_1_qubit_measure(j). The one-qubit measure is applied to qubit j. Returns the list $[p0, p1, res]$ where $p0$ is the probability of 0, $p1$ the probability of 1 and res the result.

Example: apply_1_qubit_measure(3);

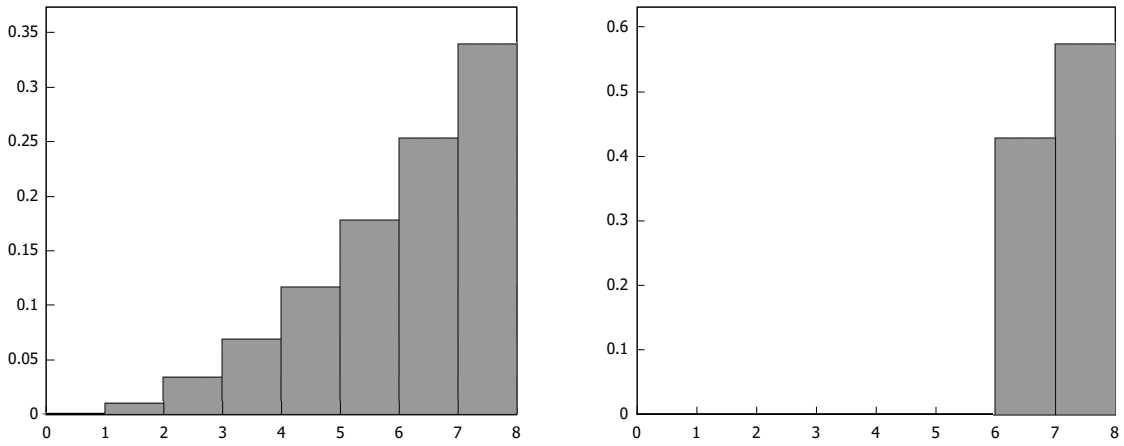
[0.4032258064516129,0.596774193548387,1]



apply_reg_measure($j0, j1$). All qubits of the register $j0...j1$ are measured. Returns the list $[[b\_1, b\_2, \ldots, b\_k], res]$ where $k = j1 - j0 + 1$, the first element is the list of bits resulting from the measurements and res is the integer with binary representation $b\_1 b\_2 \ldots b\_k$.

Example: apply_reg_measure(1,2);

[[1,1],3]

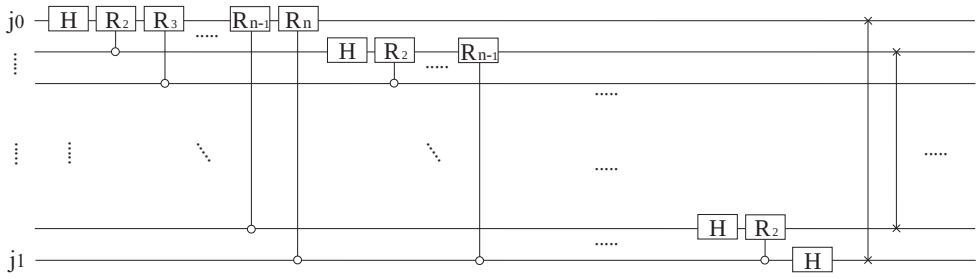


c) List of general transformations:

WH_transformation$(j0, j1)$. Apply the Walsh-Hadamard transformation to register $j0...j1$:

$$H_{j0} \otimes \cdots \otimes H_{j1}$$

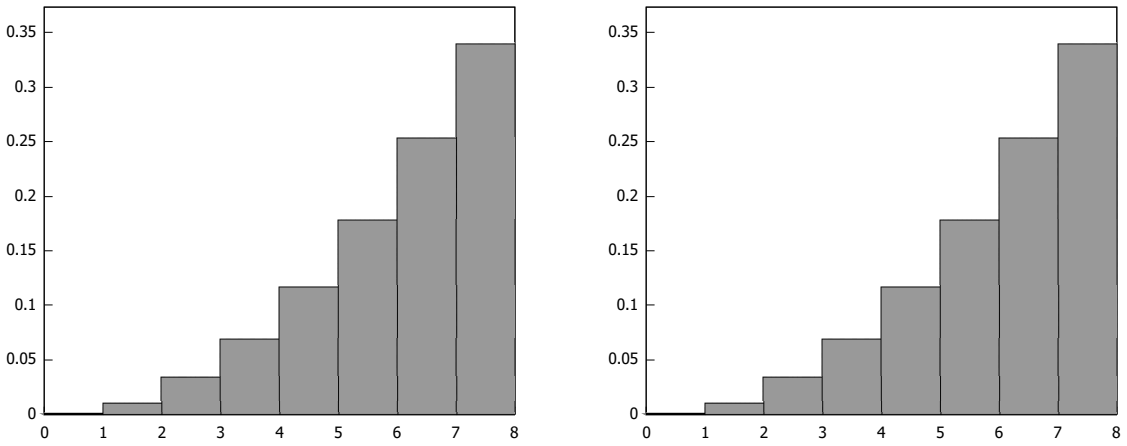
QF_transformation$(j0, j1)$. Apply the Quantum Fourier transformation to register $j0...j1$.



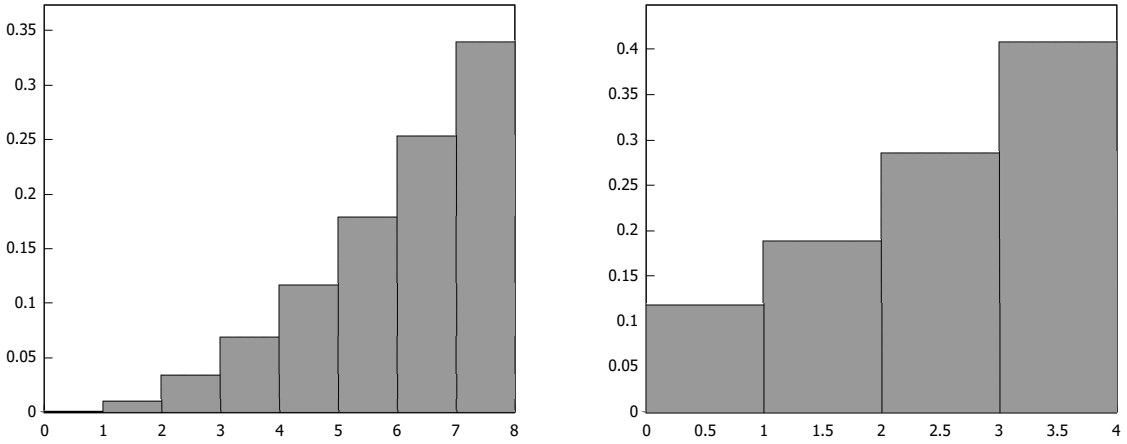
5. At any moment we can visualize the $n-$qubit. List of functions to display the $n-$qubit (the *name* variable is a character string or a number and gives a name to the graphical representation). The examples will apply to the following 3−qubit:

$$\frac{1}{\sqrt{1240}}\Big(- i|000\rangle + (2-3i)|001\rangle + (4-5i)|010\rangle + (6-7i)|011\rangle + (8-9i)|100\rangle + (10-11i)|101\rangle + (12-13i)|110\rangle + (14-15i)|111\rangle \Big)$$

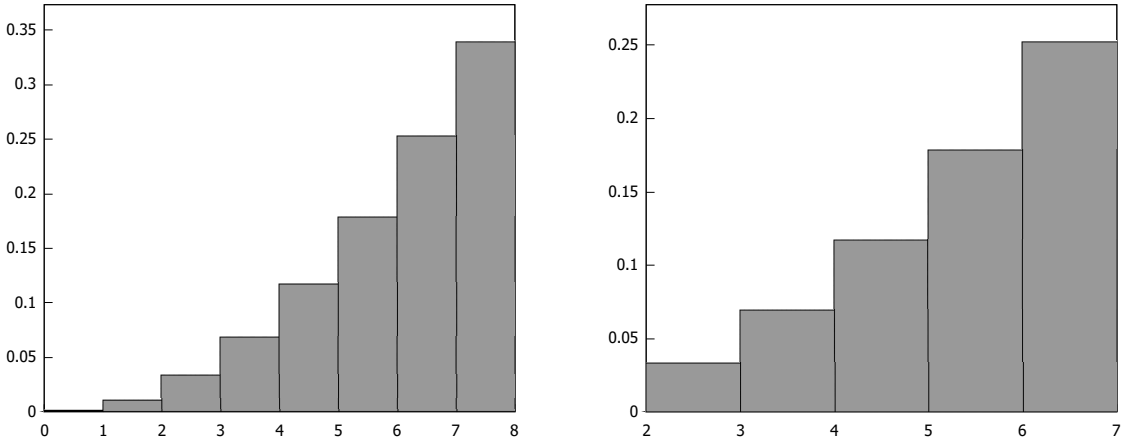
draw_prob$(name)$. Bar diagram of the probability distribution of the $n-$qubit (maximum 8 qubits − 256 bars).
Example: draw_prob("probability bar diagram")$



draw_reg_prob($name, j0, j1$). Bar diagram of the register $j0...j1$ probability distribution (maximum 8 qubits − 256 bars).
Example: draw_reg_prob("register probability bar diagram",2,3)$

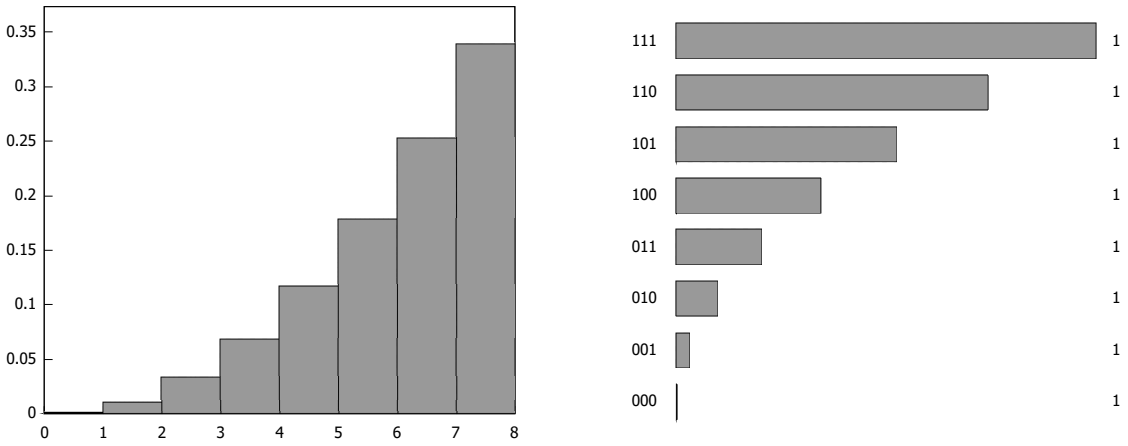


draw_list_prob($name, j0, j1$). Bar diagram of the sublist $|j0\rangle...|j1\rangle$ ($j1 - j0 \leq 256$ bars).
Example: draw_list_prob("list probability bar diagram",2,6)$

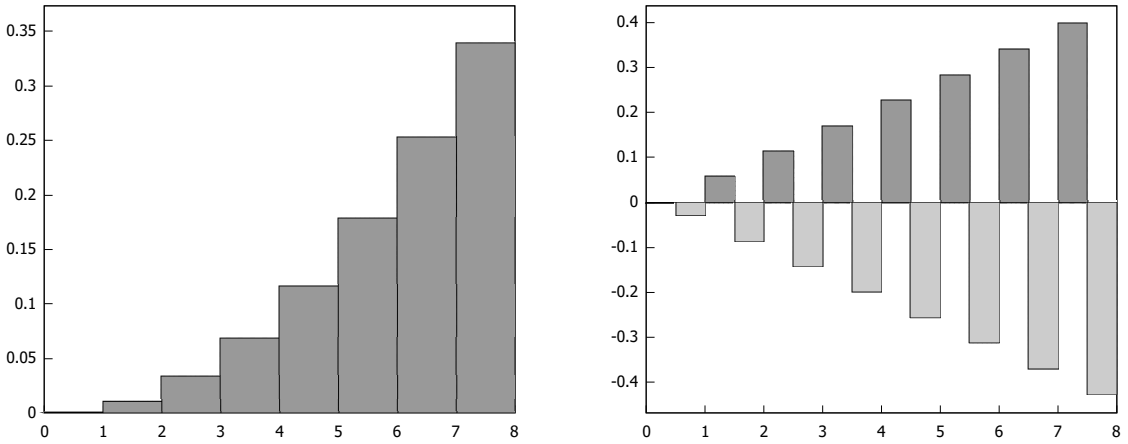


draw_biggest_prob($name, j$). Bar diagram of the j biggest probabilities ($j \leq 32$). For each of the j largest probabilities indicates the multiplicity and an element of the computation base.

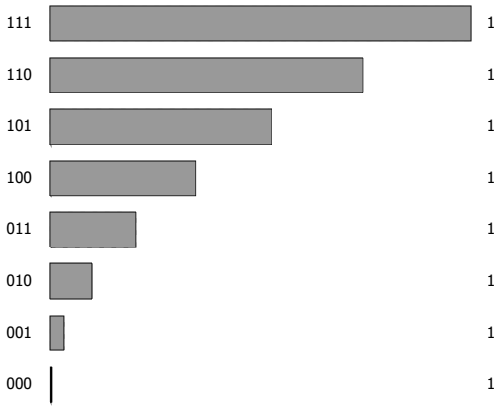
Example: draw_biggest_prob("biggest probability bar diagram",10)$



draw_coor(*name*). $N-$qubit coordinate bar diagram (maximum 8 qubits $-$ 256 bars).
Example: draw_coor("coordinate bar diagram")$



draw_graphics(*name*). Function to display graphical representation identified by the *name* variable.
Example: draw_graphics("biggest probability bar diagram")$



6. To start a simulation again with the same number of qubits use the function **initialize_n_qubit**().

4.2.2 More features of OpenQSDS

1. Error monitoring:

   a) Maximum deviation of the $n-$qubit from norm 1: global variable **max_error_n_qubit**. This global variable is updated every time a quantum gate or quantum measurement is applied.

   b) Maximum error in the unitarity of quantum gates: global variable **max_error_gates**. This global variable is updated each time a parameterized quantum gate is defined.

   c) Maximum error of probability: global variable **max_error_prob**. This global variable is updated each time a quantum measurement or a function to display $n-$qubit probabilities is applied.

   d) Checking the unitarity of the $n-$qubit: function **check_n_qubit**(). This function is used to update the global variable **max_error_n_qubit**.

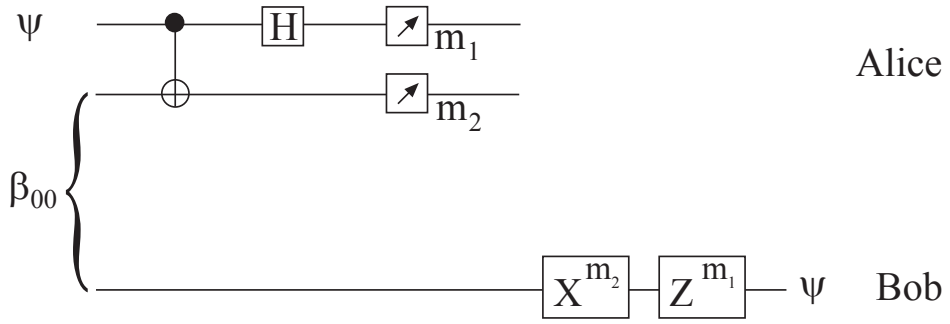
   e) Checking the unitarity of a $1-$qubit quantum gate: function **check_1_qubit_gate**(G). It is used to update the global variable **max_error_gates**.

   f) Checking the unitarity of a two-qubit quantum gate: function **check_2_qubit_gate**(G). It is also used to update the global variable **max_error_gates**.

2. Session summary: It is obtained with the function **summary**(). Reports the system variables **num_qubits**, **dim_n_qubit**, **max_error_n_qubit**, **max_error_prob** and **max_error_gates**.

3. Simulator extension. The simulator can be expanded by incorporating new quantum gates, both one-qubit and two-qubit quantum gates, and algorithms.

   a) Definition of new algorithms. They are entered as Maxima functions.
   Example. Quantum teleportation:



$$\Psi = a_0|0\rangle + a_1|1\rangle \qquad \text{and} \qquad \beta_{00} = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right)$$

Function in Maxima:

```
Q_teleportation(a0, a1) := block([m1, m2],
    /* Start the simulator */
    start_simulator(3),
    /* Preparation of the initial state */
    n_qubit[0, 0] : realpart(a0),
    n_qubit[0, 1] : imagpart(a0),
    n_qubit[4, 0] : realpart(a1),
    n_qubit[4, 1] : imagpart(a1),
    /* Representation of the initial state */
    draw_coor("initial state"),
    /* Preparation of the entangled pair */
    apply_1_qubit_gate(gate_H, 2),
    apply_1_controlled_1_qubit_gate(gate_X, 2, 3),
    /* Start of the teleportation algorithm */
    apply_1_controlled_1_qubit_gate(gate_X, 1, 2),
    apply_1_qubit_gate(gate_H, 1),
    m1 : apply_1_qubit_measure(1)[3],
    m2 : apply_1_qubit_measure(2)[3],
    if m2 = 1 then apply_1_qubit_gate(gate_X, 3),
    if m1 = 1 then apply_1_qubit_gate(gate_Z, 3),
    /* End of algorithm */
    [m1, m2]
)$
```

Example: $\Psi = \left(\frac{1}{\sqrt{30}} + \frac{2}{\sqrt{30}}i\right)|0\rangle + \left(\frac{3}{\sqrt{30}} - \frac{4}{\sqrt{30}}i\right)|1\rangle$.

```
Q_teleportation(float(1/sqrt(30)) + %i * float(2/sqrt(30)),
                float(3/sqrt(30)) - %i * float(4/sqrt(30)));
```

[1,1]