QuantumUtils
User Manual

A. Martin

Version 1.0.1
September 1, 2025

Contents

1 Introduction

QuantumUtils is an easy-to-use command-line tool designed for quantum computing calculations and bra-ket notation operations. This tool is a lightweight alternative to bulky computing suites, aiming to providing researchers, students, and hobbyists with an efficient way to perform quantum computations directly in the terminal. QuantumUtils is quantum-specific but also contains general linear algebra operations useful in quantum computing.

1.1 Features

QuantumUtils is an efficient way to perform:

- Matrix operations with complex numbers
- Bra-ket notation calculations
- Tensor products
- Expectation value calculations
- Applying quantum gates to states

and more in an interactive command-line interface.

1.2 System Requirements

- The latest version of Python (currently 3.13.7)
- The NumPy library
- Any terminal or command prompt environment

2 Installation

2.1 Installing Dependencies

```
# Install Python 3
sudo apt-get install python3

# Install pip (if not already installed)
sudo apt-get install python3-pip

# Install NumPy
pip3 install numpy
```

2.2 Downloading and Running QuantumUtils

```
# Clone the repository, which contains the script
git clone https://github.com/a-mart0/QuantumUtils.git

# Run the tool
python3 quantum_utils.py
```

3 Getting Started

3.1 Launching the Tool

After installing the dependencies, run the tool with:

```
python3 quantum_utils.py
```

You will be greeted with the welcome banner and the prompt head

```
quantum>
```

3.2 First Steps

Let's start with a simple calculation. Type the following command to see the Pauli-X matrix

```
quantum> pauli X
```

You should see the output:

```
X =
    0.000     1.000
    1.000     0.000
```

4 Command Reference

4.1 General Commands

4.1.1 help

Displays the help menu with all available commands.

```
quantum> help
```

4.1.2 clear

Clears the terminal screen.

```
quantum> clear
```

4.2 Matrix Operations

4.2.1 create

Creates a new matrix with the specified name and dimensions i, j:

```
quantum> create my_matrix 2 2
```

You will be prompted to enter the matrix elements row by row.

4.2.2 show

Displays the contents of a variable.

```
quantum> show X
```

4.2.3 add

Adds two matrices and stores the result.

```
quantum> add X Y result
```

4.2.4 mult

Multiplies two matrices and stores the result.

```
quantum> mult X Y result
```

4.2.5 tensor

Computes the tensor product of two matrices.

```
quantum> tensor X Z result
```

4.2.6 det

Calculates the determinant of a matrix.

```
quantum> det H
```

4.2.7 eigen

Calculates eigenvalues and eigenvectors of a matrix.

```
quantum> eigen H
```

4.2.8 conjugate

Computes the complex conjugate of a matrix.

```
quantum> conjugate A result
```

4.2.9 transpose

Computes the transpose of a matrix.

```
quantum> transpose A result
```

4.2.10 dagger

Computes the conjugate transpose, known as the dagger of a matrix.

```
quantum> dagger A result
```

4.2.11 inverse

Computes the inverse of a matrix.

```
quantum> inverse A result
```

4.3 Quantum Operations

4.3.1 inner

Calculates the inner product of two quantum states.

```
quantum> inner <0| |1>
```

4.3.2 outer

Calculates the outer product of two quantum states.

```
quantum> outer |0> <1|
```

4.3.3 normalize

Normalizes a quantum state.

```
quantum> normalize |psi> result
```

4.3.4 expectation

Calculates the expectation value of an operator for a given state.

```
quantum> expectation X |+>
```

4.4 Special Functions

4.4.1 pauli

Displays one of the Pauli matrices (X, Y, Z), Identity (I), or the Hadamard matrix (H).

```
quantum> pauli X
```

4.4.2 bell

Creates a Bell state.

```
quantum> bell
```

5 Examples

5.1 Example 1: Basic Quantum State Operations

Let's create and perform some operations on a quantum state.

1. Create a custom state:

   ```
   quantum> create my_state 2 1
   Enter 2x1 matrix (complex numbers: a+bj):
   Row 1: 1
   Row 2: 1j
   ```

   which will output

   ```
       Matrix my_state created successfully!
   ```

2. Normalize the state:

   ```
   quantum> normalize my_state normalized_state
   ```

   which produces the following

   ```
   Normalized state stored in normalized_state:
       0.707
       0.707j
   ```

5.2 Example 2: Quantum Gate Operations

Let's experiment with quantum gates.

(a) View the Pauli matrices:

```
quantum> pauli X
quantum> pauli Y
quantum> pauli Z
```

which will print the pauli matrices.

(b) Apply the X gate to $|0\rangle$:

```
quantum> mult X |0> result
```

which outputs

```
Result stored in result:
     0.000
     1.000
```

(c) Create a two-qubit system:

```
quantum> tensor |0> |1> |01>
```

which outputs

```
Result stored in |01>:
     0.000
     1.000
     0.000
     0.000
```

(d) Apply a Hadamard to the first qubit (identity to the second):

```
quantum> tensor H I HI
Result stored in HI:
     0.707     0.000     0.707     0.000
     0.000     0.707     0.000     0.707
     0.707     0.000    -0.707    -0.000
     0.000     0.707    -0.000    -0.707
quantum> tensor |0> |1> |01>
Result stored in |01>:
     0.000
     1.000
     0.000
     0.000
quantum> mult HI |01> HI|01>
Result stored in HI|01>:
     0.000
     0.707
     0.000
     0.707
```

which is equivalent to

```
quantum> mult H |0> H|0>
Result stored in H|0>:
     0.707
     0.707
quantum> tensor H|0> |1> HI|01>
Result stored in H|01>:
     0.000
     0.707
     0.000
     0.707
```

5.3 Example 3: Entanglement and Measurement

Let's create and analyze a bell state.

(a) Create Bell states:

```
quantum> bell
```

which produces

```
Bell states created:
  |PHI+> = (|00> + |11>)/sqrt{2}
  |PHI-> = (|00> - |11>)/sqrt{2}
  |PSI+> = (|01> + |10>)/sqrt{2}
  |PSI-> = (|01> - |10>)/sqrt{2}
```

(b) Display the $|\Phi_+\rangle$ Bell state:

```
quantum> show |PHI+>
```

with output

```
|PHI+> =
    0.707
    0.000
    0.000
    0.707
```

(c) Calculate the expectation value of $X \otimes X$:

```
quantum> tensor X X XX
quantum> expectation XX |PHI+>
```

which produces:

```
quantum> tensor X X XX
Result stored in XX:
     0.000     0.000     0.000     1.000
     0.000     0.000     1.000     0.000
     0.000     1.000     0.000     0.000
     1.000     0.000     0.000     0.000
quantum> expectation XX |PHI+>
<PHI+|XX|PHI+> = (0.9999999999999998+0j)
```

as expected.

6 Miscellaneous

6.1 Complex Number Input

When entering complex numbers, use the format `a+bj`, rather than the format $a + ib$:

- Real numbers can be written as: `1` or `1.0`
- Imaginary numbers can be written as: `n*j` (where n is any real number), but not $1i$.
- A complex number is written as: `1+2j`, for example.

7 Conclusion

I will continue to develop and improve QuantumUtils. For the latest updates and additional resources, visit my github page at `https://github.com/a-mart0`.

A Mathematical Background

A.1 Dirac Notation

In quantum mechanics, Dirac notation (bra-ket notation) is used to conveniently represent quantum states.

- Kets $|\psi\rangle$ represent column vectors
- Bras $\langle\psi|$ represent row vectors

A.2 Quantum Gates

Common single-qubit gates include the Pauli matrices (X, Y, Z), the Identity matrix (I), Hadamard (H), and S. In matrix form, these operators are

$$\begin{aligned} I &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & X &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\ Y &= \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & Z &= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ H &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & S &= \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}. \end{aligned}$$

B License Information

QuantumUtils is released under the GNU General Public License v3.0. This means you are free to run, study, share and modify the software, provided that any distributed copies or derivative works are also licensed under the GPL v3.0 and accompanied by the corresponding source code, in accordance with the terms of that license. For more information, see `https://www.gnu.org/licenses/gpl-3.0.en.html#license-text`.

C Version History

- Version 1.0.1 (Current): Fixed an issue where the expectation command was not displaying bras correctly.
- Version 1.0.0 (Previous): Initial release with basic functionality.
- Future versions: Planned features include partial trace, norm of an operator, and more algebraic properties like commutators and anti-commutators. The ability to generate a GHZ state will be implemented. Better documentation is coming.