PyQudit

Release 1.0.1

Aarya Bodhankar

Jul 14, 2021

INDEX

PyQudit is a Python package for using generalised and universal versions of quantum gates, in N-dimensions. Enables building simple quantum circuit simulations on qudit logic using higher dimensional gates.

CHAPTER

ONE

GETTING STARTED

1.1 Concept

Mainstream QuantumComputing uses qubits which operate in a two dimensional Hilber space. Qudits are their higher dimensional equivalents with better informaiton density and potential for higher efficiency. PyQudit includes the qudit versions of fundamental quantum gates, useable over any dimension* as specified by the user. It can be used to understand the behaviour of qudit gates as also to build higher dimensional circuits for experimentation.

*refer gate functions

1.2 Install

The latest stable version of PyQudit is available on PyPI and can be installed with pip. It is recommeded to install in your quantum computing python environment, alongside existing packages.

```
pip install pyqudit
```

1.3 Build Locally

Alternatively, you can build the package wheel from source and then install it via pip.

```
pip3 install --upgrade pip
pip3 install --upgrade setuptools
git clone https://github.com/Ordoptimus/pyqudit.git
cd pyqudit
python3 setup.py bdist_wheel
```

Replace [version] with the latest version as seen in the wheel file in /bdist_wheel

```
pip3 install dist/pyqudit-[version]-py3-none-any.whl
```

1.4 Use

```
import pyqudit.qudit as pq
```

Use `dir(pq)` to show all package methods.

1.5 Sample

```
>>> import pyqudit.qudit as pq
>>> d = int(input('Enter Dimensions: '))
Enter Dimensions: 3
>>> print("\n---CX's Pauli Matrix---")
>>> print(pq.CXd_pauli(d))
---CX's Pauli Matrix---
[[1 0 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]
 [0 0 0 0 0 1 0 0 0]
 [0 0 0 1 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 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 0]]
```

CHAPTER

TWO

DOCUMENTATION

Refer the official documentation for detailed examples and syntax.

2.1 Gate Functions

All the gate functions available in PyQudit are documented here with examples. Since all gates are generalised implementations, the user needs to provide the dimension through the parameter d. Unless specified otherwise, all qudit parameters are in the Ket form. If number of qudits is not specified, the gates are single-qudit ones.

Table of Contents

2.1.1 CX gate

The functions for the two-qudit CX (CNOT) gate implementations. The CX gate is a fundamental gate of both qubit and qudit logics and essential for most kinds of qudit manipulations.

CXd_dis

`CXd_dis(d,qudit1,qudit2)`

Generalised CX (CNOT) Gate.

```
>>> CXd_dis(4,[0,1,0,0],[1,0,0,0])
[0,1,0,0]
```

Operates in Ket form. If working with decimal values, use `convKet(d,qt)` to convert each qudit value to Ket.

CXd

`CXd(d,state)`

Standard formulaic implementation of generalised CX, applicable for superposed states. Works using the state matrix form, which can be obtained using the `stateMat(d,qudit1,qudit2)` function.

```
>>> statematrix = stateMat(4,[0,1,0,0],[1,0,0,0])
>>> CXd(4,statematrix)
array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

CXd_cstm

`CXd_cstm(d,statematrix)`

Custom, more efficient formulaic implementation of CX with same capabilities as `CXd`. Recommended for logic involving lower dimensions. Also involves state matrix form.

```
>>> statematrix = stateMat(4,[0,1,0,0],[1,0,0,0])
>>> CXd_cstm(4,statematrix)
[0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

2.1.2 CX-Drag Gate

The functions for the two-qudit CX-Drag gate. The CX-Drag gate is an inverse of the CX gate and is required in qudit logic to make compound gates and emulate some aspects of qubit gates.

CXDrag_dis

`CXDrag_dis(d,qudit1,qudit2)`

Generalised CX Drag Gate not applicable for superposed states. Formulaic implementation.

```
>>> CXDrag_dis(4,[0,0,1,0],[1,0,0,0])
[0, 0, 0, 1]
```

CXDrag

`CXDrag(d,statematrix)`

Generalised CX Drag Gate applicable for superposed states. Uses a state matrix of the two qudits, instead of their Ket forms. Handles superposition well.

```
>>> statematrix = stateMat(4,[0,1,0,0],[1,0,0,0])
>>> CXDrag(4,statematrix)
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0])
```

2.1.3 GXOR Gate

The various functions for the two-qudit GXOR gate, which is used to achieve logic for other gates, such as SWAP or the Hermetian CX.

GXOR_dis

`GXOR_dis(d,qudit,qudit2)`

Generalised GXOR gate not applicablefor superposed states. Formulaic implementation.

```
>>> GXOR_dis(4,[0,1,0,0],[1,0,0,0])
[0,1,0,0]
```

GXOR

GXOR(d,statematrix)

Generalised implementation of the GXOR gate, applicable for superposed states. Uses the state matrix form of two qudits.

```
>>> statematrix = stateMat(4,[0,1,0,0],[1,0,0,0])
>>> GXOR(4,statematrix)
array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

2.1.4 SWAP Gate

The functions of the two-qudit SWAP gate, used to swap the states of two qudits, akin to its qubit equivalent. It is a compound gate, with CX, CXDrag, and GXOR as its constituents.

SWAPd

SWAPd(d,qudit1,qudit2)

Generalised swap gate implementation not applicable for superposition.

```
>>> SWAPd(4,[0,1,0,0],[1,0,0,0])
([1, 0, 0, 0], [0, 1, 0, 0])
```

2.1.5 Hadamard Gate

The functions of the Hadamard gate, one of the fundamental and crucial gates of quantum logic used to carry out superposition.

Hd_dis

Hd_dis(d,qudit)

Generalised implementation for all dimensions. Can't handle superposed states.

```
>>> Hd_dis(4,[0,1,0,0])
array([ 5.00000000e-01+0.00000000e+00j,  1.63397448e-07+5.00000000e-01j,
       -5.00000000e-01+3.26794897e-07j, -4.90192345e-07-5.00000000e-01j])
```

Hd

`Hd(d,qudit)`

Generalised implementation for 2^n dimensions. Handles superposed states.

```
>>> Hd(4,[0,1,0,0])
array([ 0.5, -0.5,  0.5, -0.5])
```

2.1.6 CZ Gate

The functions for the two-qudit CZ gate. This is a compound gate, with the Hadamard gate and CX gate as its constituents. CZ inherits Hadamard's superposition conditions.

CZd

`CZd(d,statematrix)`

Generalised implementation for 2^n dimensions. Handles superposed states.

```
>>> statematrix = stateMat(4,[0,1,0,0],[1,0,0,0])
>>> CZd(4,statematrix)
array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

2.1.7 X Gate

The functions for the X gate. This gate works similar to the NOT gate in classical logic, except on multiple dimensions.

Xd

`Xd(d,qudit)`

Generalised implementation of the X gate, applicable for all dimensions. Supports superposed states.

```
>>> Xd(4,[0,1,0,0])
array([0, 0, 1, 0])
```

2.1.8 Y Gate

The functions for Y gate. It is a phase-change gate like its qubit equivalent, but for multiple dimensions.

Yd

```
Yd(d,qudit)
```

Generalised implementation of the Y gate, applicable for all dimensions. Supports superposed states.

```
>>> Yd(4,[0,1,0,0])
array([ 0.+0.00000000e+00j,  0.+0.00000000e+00j, -1.+3.26794897e-07j,
        0.+0.00000000e+00j])
```

2.1.9 Z Gate

The functions for Z gate. It is also a phase-change gate like its qubit equivalent, but for multiple dimensions.

Zd

```
Zd(d,qudit)
```

Generalised implementation of the Z gate, applicable for all dimensions. Supports superposed states.

```
>>> Zd(4,[0,1,0,0])
array([0.00000000e+00+0.j, 3.26794897e-07+1.j, 0.00000000e+00+0.j,
       0.00000000e+00+0.j])
```

2.1.10 Toffoli Gate

The functions for the three-qudit Toffoli or CCNOT gate. This is another fundamental gate useful in delaing with multiple qudits.

Toffolid

```
Toffolid(d,qudit1,qudit2,qudit3)
```

Generalised implementation of the Toffoli gate, applicable for all dimensions. Supports superposed states.

```
>>> Toffolid(4,[0,1,0,0],[1,0,0,0],[0,0,0,0])
array([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, 0, 0, 0, 0, 0, 0])
```

2.2 Pauli Matrix Functions

Aside from the generalised qudit gates, PyQudit also includes functions which output the gate matrices. These matrices are useful in understanding the logic and mathematics behind a gate, as also in using them for building circuits. Parallelly, circuits and gates requiring superposition make direct use of these matrix functions. States ofthe qubit gates are represented by their Pauli Matrices, and the similarity of higher dimensionality matrices coupled with a lack of a different name meant our matrix functions are also termed as Pauli Matrices.

Table of Contents

2.2.1 CX Matrix

The functions for matrix representations of the various implementations of the CX gate.

CXd_pauli

`CXd_pauli(d)`

Matrix representation of the formulaic version of the CX gate. More efficient on smaller dimensions.

```
>>> CXd_pauli(4)CXd_cstm_pauli
array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]])
```

CXd_cstm_pauli

`CXd_cstm_pauli(d)`

Matrix representation of the custom implementation of the CX gate. More efficient on higher dimensions.

```
>>> CXd_cstm_pauli(4)
array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
```

(continues on next page)

(continued from previous page)

```
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]])
```

2.2.2 CX-Drag Matrix

The functions for the matrix representations of the CX-Drag gate.

CXDrag_pauli

```
CXDrag_pauli(d)
```

Matrix representation of the CX-Drag gate, common for both implementations.

```
>>> CXDrag_pauli(4)
array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 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, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]])
```

2.2.3 GXOR Matrix

The functions for the matrix representations of the GXOR gate.

GXOR_pauli

```
GXOR_pauli(d)
```

Matrix representation of the GXOR gate, common for both implementations.

```
>>> GXOR_pauli(4)
array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
```

(continues on next page)

(continued from previous page)

```
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]])
```

2.2.4 Hadamard Matrix

The functions for the marix representations of the Hadamard gate. These matrices are expecially useful for understanding and working with superposition.

Hd_pauli

`Hd_pauli(d)`

Matrix representation of the Hadamard gate, common for both implementations.

```
>>> Hd_pauli(4)
array([[ 0.5,  0.5,  0.5,  0.5],
       [ 0.5, -0.5,  0.5, -0.5],
       [ 0.5,  0.5, -0.5, -0.5],
       [ 0.5, -0.5, -0.5,  0.5]])
```

2.2.5 CZ Matrix

The functions for the matrix representations of the CZ gate.

CZd_pauli

`CZd_pauli(d)`

Matrix representation of the CZ gate implementation.

```
>>> pq.CZd_pauli(4)
array([[ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
```

(continues on next page)

(continued from previous page)

```
        0.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0., -1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0., -1.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., -1.,  0.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., -1.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        -1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0., -1.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  1.,  0.]])
```

2.2.6 X Matrix

The functions for the matrix representations of the X gate.

Xd_pauli

`Xd_pauli(d)`

Matrix representation of the X gate implementation.

```
>>> Xd_pauli(4)
array([[0, 0, 0, 1],
       [1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 1, 0]])
```

2.2.7 Y Matrix

The functions for the matrix representations of the Y gate.

Yd_pauli

```
Yd_pauli(d)
```

Matrix representation of the Y gate implementation.

```
>>> Yd_pauli(4)
array([[ 0.00000000e+00+0.00000000e+00j,  0.00000000e+00+0.00000000e+00j,
         0.00000000e+00+0.00000000e+00j,  1.00000000e+00-9.80384690e-07j],
       [ 0.00000000e+00+1.00000000e+00j,  0.00000000e+00+0.00000000e+00j,
         0.00000000e+00+0.00000000e+00j,  0.00000000e+00+0.00000000e+00j],
       [ 0.00000000e+00+0.00000000e+00j, -1.00000000e+00+3.26794897e-07j,
         0.00000000e+00+0.00000000e+00j,  0.00000000e+00+0.00000000e+00j],
       [ 0.00000000e+00+0.00000000e+00j,  0.00000000e+00+0.00000000e+00j,
        -6.53589793e-07-1.00000000e+00j,  0.00000000e+00+0.00000000e+00j]])
```

2.2.8 Z Matrix

The functions for the matrix representations of the Z gate.

Zd_pauli

```
Zd_pauli(d)
```

Matrix representation of the Z gate implementation.

```
>>> Zd_pauli(4)
array([[ 1.00000000e+00+0.00000000e+00j,  0.00000000e+00+0.00000000e+00j,
         0.00000000e+00+0.00000000e+00j,  0.00000000e+00+0.00000000e+00j],
       [ 0.00000000e+00+0.00000000e+00j,  3.26794897e-07+1.00000000e+00j,
         0.00000000e+00+0.00000000e+00j,  0.00000000e+00+0.00000000e+00j],
       [ 0.00000000e+00+0.00000000e+00j,  0.00000000e+00+0.00000000e+00j,
        -1.00000000e+00+6.53589793e-07j,  0.00000000e+00+0.00000000e+00j],
       [ 0.00000000e+00+0.00000000e+00j,  0.00000000e+00+0.00000000e+00j,
         0.00000000e+00+0.00000000e+00j, -9.80384690e-07-1.00000000e+00j]])
```

2.2.9 Toffoli Matrix

The functions for the matrix representations of the Toffoli gate.

Toffolid_pauli

`Toffolid_pauli(d)`

Matrix representations of the Toffoli gate implementation.

```
>>> Toffolid_pauli(4)
array([[1, 0, 0, ..., 0, 0, 0],
       [0, 1, 0, ..., 0, 0, 0],
       [0, 0, 1, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 1, 0, 0],
       [0, 0, 0, ..., 0, 1, 0]])
```

2.3 Other Functions

These functions serve as utilities that help the user with activities such as conversions and tensor products. They are used within and in tandem with the gate and matrix functions.

Table of Contents

2.3.1 Convert2Ket

`convKet(d,decimal_qudit)`

A function which enables qudit conversion from decimal form to ket form.

```
>>> convKet(4,1)
[0, 1, 0, 0]
```

2.3.2 Convert2Decimal

`convDec(d,ket_qudit)`

A function which enables qudit conversion from ket form to decimal form.

```
>>> pq.convDec(4,[0,1,0,0])
1
```

2.3.3 StateMatrix

`stateMat(d,qudit1,qudit2)`

A function which facilitates conversion of the states of two qudits in Ket form to a state matrix.

```
>>> stateMat(4,[0,1,0,0],[1,0,0,0])
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

2.3.4 IdentityTensorProduct

`tensorIGt(d,gate_symbol)`

A function to output the tensor product of an identity matrix with either of Hadamard (H), X, or Z gates.

```
>>> tensorIGt(2,'H')
array([[ 0.70710678,  0.70710678,  0.        ,  0.        ],
       [ 0.70710678, -0.70710678,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.70710678,  0.70710678],
       [ 0.        ,  0.        ,  0.70710678, -0.70710678]])
```