Rust Quantum Library

The quantum library in Rust contains several useful measurements of quantum entanglement. Currently the library is used in Python. The Rust library uses ndarray and the ndarray_linalg packages.

1 Complex Fibonacci function (*fibc*)

Input: Complex Number

Input Type: pub type Complex = num_complex::Complex<f64>

Input Example: n = Complex::new(2.0, 2.0)

Output: The square root of the matrix

Output Type: pub type MatrixC64 = ndarray::Array2<c64>

Output Example: -25.470811965633576 + 38.49125803729125i

This function takes a complex number and returns the Fibonacci sequence of that number using the closed form solution of the Fibonicci sequence [1]:

$$Fib(n) = \frac{\phi^n - \cos(\pi n)\phi^{-n}}{\sqrt{5}}$$

2 Create a density matrix ρ (*create_dens_matrix*)

Input: The wavefunction expressed as a column vector of coefficients

Input type: pub type VecC64 = ndarray::Array1<c64>

Input Example: For the Bell state $\phi_+ = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, let norm_const = 1./2_f64.sqrt();

let bell_phi_plus = array![c64::new(norm_const , 0.0) , c64::new(0.0 , 0.0) , c64::new(0.0 , 0.0) , c64::new(norm_const , 0.0)];

Output: The density matrix, which is square and Hermitian.

Output type: pub type MatrixC64 = ndarray::Array2<c64>

Output Example: dens matrix for phi_+ =

[[0.4999999999999999+0i, 0+0i, 0+0i, 0.4999999999999999+0i],

[0+0i, 0+0i, 0+0i, 0+0i],

[0+0i, 0+0i, 0+0i, 0+0i],

[0.4999999999999999+0i, 0+0i, 0+0i, 0.4999999999999999+0i]]

The density matrix is a more general way of representing the state of a quantum system. It is represented as $\rho = |\psi\rangle \langle\psi|$, where the wavefunction $|\psi\rangle$ is represented as a column vector.

With this same example, the function performs the operation

$$\rho = |\phi_+\rangle \langle\phi_+| = \frac{1}{2} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix}^* = \frac{1}{2} \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

3 Find ρ^2 (*find_dens_matrix_sqrd*)

Input: Square Matrix

Input Type: pub type MatrixC64 = ndarray::Array2<c64>

Input Example:

array![[c64::new(3., 1.) , c64::new(-1., 1.)],

[c64::new(2., -1.) , c64::new(-2., -1.)]]

Output: Square Matrix

Output Type: pub type MatrixC64 = ndarray::Array2<c64>

Output Example:

[[7+9i, -1+1i],

[2-1i, 2+7i]]

A simple function that squares the density matrix for various calculations.

***This is still a matrix product and not the dot product. The language is slightly misleading:

let matrix_sqrd = matrix.dot(matrix);

4 Find matrix dimension (*find_dim*)

Input: Square Matrix

Input Type: pub type MatrixC64 = ndarray::Array2<c64>

Input Example:

array![[c64::new(3., 1.) , c64::new(-1., 1.)],

[c64::new(2., -1.) , c64::new(-2., -1.)]]

Output: Integer

Output Type: i32

Output Example: 2

Another simple function that calculates the length of a square matrix. For example, if a matrix is 8 x 8, this will return 8 as a type i32 (integer 32 bit).

5 Find purity (*find_purity*)

Input: ρ^2 **(Density Matrix squared)**

Input Type: pub type MatrixC64 = ndarray::Array2<c64>

Input Example:

array![[c64::new(0.25 , 0.) , c64::new(0. , 0.)],

[c64::new(0. , 0.) , c64::new(0.25 , 0.)]]

Output: Float

Output Type: f64

Output Example: 0.5

Simply returns the trace of ρ^2 as a f64 type. For an idea on where this purity lies within the range of possible values, print a statement such as println!("The purity lies between and 1", 1./(find_dim(rho_sqrd))).

6 Find the square root of a matrix (*find_sqr_root_of_matrix*)

Input: Any square, Hermitian matrix

Input Type: pub type MatrixC64 = ndarray::Array2<c64>

Input Example:

Output: The square root of the matrix

Output Type: pub type MatrixC64 = ndarray::Array2<c64>

Output Example:

For some of the entanglement measurements, the square root of the density matrix is required. Using a LAPACK function such as matrix.ssqrt(UPLO:Lower).unwrap() is the easiest and fastest route, but LAPACK has some sort of bug and fails to compute. It's suspected that the matrix needs to be semi-positive definite (eigenvalues need to be >= 0) in order for it to work.

So, the function was written by decomposing the Hermitian matrix as $M = SDS^{-1}$, where S and S^{-1} are complex unitary matrices and D is a real diagonal matrix. S is a matrix of the eigenvectors of M, and D is has the eigenvalues of M in the diagonal.

Since D is diagonal, one can take the square of the elements in order to find $\sqrt{D}$. So, $\sqrt{M} = S\sqrt{D}S^{-1}$.

But, one problem is that again, when doing this decomposition, the matrix M needs to be semi-positive definite. It usually is, but LAPACK will sometimes find that one eigenvalue is a very small negative number, such as 2.8×10^{-15}. A rescaling function is called in order to fix this issue. The rescaling function takes matrix M as an input and finds its eigenvalues and eigenvectors, and sets any negative eigenvalues to 0. Then, matrix M can be decomposed into $M = SDS^{-1}$. The rescaling function returns (D, S) as a tuple. Then, the square root function returns $\sqrt{M} = S\sqrt{D}S^{-1}$.

***Unfortunately after rescaling the function and calling LAPACK with matrix.ssqrt(UPLO:Lower).unwrap() doesn't work.

7 Find fidelity (*find_fidelity*)

Input:two density matrices ρ and σ

Input Type: pub type MatrixC64 = ndarray::Array2<c64>

Input Example:

array![

$$[c64 :: new(0.25, 0.0), c64 :: new(0.0, 0.0), c64 :: new(0.0, 0.0), c64 :: new(0.0, 0.0)]$$

,
    c64::new(0.0 , 0.0) , c64::new(0.25 , 0.0) , c64::new(0.0 , 0.0) , c64::new(0.0 , 0.0)

,
    c64::new(0.0 , 0.0) , c64::new(0.0 , 0.0) , c64::new(0.25 , 0.0) , c64::new(0.0 , 0.0)

,
    c64::new(0.0 , 0.0) , c64::new(0.0 , 0.0) , c64::new(0.0 , 0.0) , c64::new(0.25 , 0.0)

];

Output:A number between 0 and 1

Output Type: f64

Output Example: 1

This function is a distance measurement of two quantum states ρ and σ. It is expressed as $F = tr\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}$. This function calls *find_sqr_root_of_matrix* to calculate $\sqrt{\rho}$ and the overall square root of $\sqrt{\rho}\sigma\sqrt{\rho}$.

8 Find concurrence (*find_concurrence*)

Input: The density matrix ρ

Input Type: pub type MatrixC64 = ndarray::Array2<c64>

Input Example:

Output:A number between 0 and 1

Output Type: f64

Output Example:

9 Find trace norm (*find_trace_norm*)

Input: The density matrix ρ

Input Type: pub type MatrixC64 = ndarray::Array2<c64>

Input Example:

Output:A number between 0 and 1

Output Type: f64

Output Example:

The trace norm is expressed as $||\rho||_1 = tr\sqrt{\rho\rho^\dagger}$. $\rho^\dagger$ is the complex conjugate and transpose of the matrix ρ. The function calls *find_sqr_root_of_matrix*$(\rho\rho^\dagger)$.

10 Find negativity (*find_negativity*)

Input: The density matrix ρ

Input Type: pub type MatrixC64 = ndarray::Array2<c64>

Input Example:

Output:A number between 0 and 1

Output Type: f64

Output Example:

Th negativity can be expressed as $N = \frac{||\rho^{\Gamma_A}||_1 - 1}{2}$ where ρ^{Γ_A} is the partial transpose of a substate of ρ. The Peres-Horodecki criterion separates ρ into the tensor product of two states A and B. Practically speaking, ρ^{Γ_A} is called with the function *find_partial_transpose*(ρ).

11 Find log negativity (*find_log_negativity*)

Input: The density matrix ρ

Input Type: pub type MatrixC64 = ndarray::Array2<c64>

Input Example:

Output:A number between 0 and 1

Output Type: f64

Output Example:

This function lets $N = find\_log\_negativity(\rho)$. Then it takes $log_2(2N + 1)$.

12 Find Schmidt number (*find_trace_norm*)

Input: The joint spectral intensity (JSI) ρ

Input Type: pub type MatrixF64 = ndarray::Array2<f64>

Input Example:

Output: An entanglement measurement of the JSI.

Output Type: f64

Output Example:

Takes the square root of the elements of the JSI to find the JSA (joint spectral amplitude). The joint spectral amplitude is a matrix of floats from SPDCalc. Then, it finds the singular value decomposition (SVD) of the JSA.

13 Find partial transpose (*find_partial_transpose*)

Input: Any square matrix

Input Type: pub type MatrixC64 = ndarray::Array2<c64>

Input Example:

Output:A square matrix

Output Type: pub type MatrixC64 = ndarray::Array2<c64>

Output Example:

This computes the partial transpose of a matrix, where for any square density matrix, it splits the matrix into 4 symmetric square blocks transposes the upper right and lower left blocks of the matrix.

References

[1] "generalizations of fibonacci numbers". https://en.wikipedia.org/wiki/Generalizations_of_Fibonacci_numbers.