User's Manual for *qinf* Quantum Information and Entanglement Package v. 0.05 For The *Maxima* Computer Algebra System

G. John Lapeyre, Jr.

August 18, 2008

Contents

1 Introduction **2**
1.1 Acknowledgments . . . 3

2 Using *Maxima*. features important for *qinf* **3**

3 Representation of states and operators **5**

4 Creating instances of states **6**
4.1 **ketz**$(i_1,\ldots,i_n)$, **braz**$(i_1,\ldots,i_n)$, **ketx**(.), **brax**(.), **kety**(.), **bray**(.), multipartite eigenstates of $\sigma_x,\sigma_y,\sigma_z$ 6
4.2 **ket_n**$(j,i_1,\ldots,i_m)$, **bra_n**$(j,i_1,\ldots,i_m)$, alternate form of **ketz**, etc. . . . 7
4.3 Density matrix representation of a pure state (projection operator) . . . 7
4.4 Creating state vectors with the tensor product. **otimes** and **tensor_product** . . . 8
4.5 **schmidt_ket**(a). bipartite state in Schmidt basis . . . 8
4.6 **bell[a,b]** and **belln[i]** Bell vectors in computational basis . . . 9

5 Creating and using operators **10**
5.1 **pauli**$[i]$ . . . 10

6 Entanglement– ptrace$(\rho,i_1,\ldots)$, entropy(ρ), purity(ρ) **12**
6.1 **ptrace**$(\rho,i_1,\ldots)$, **ptracen**$(n,\rho,i_1,\ldots)$ partial trace . . . 12
6.2 **entropy** . . . 13
6.3 **purity** . . . 13
6.4 An entangled pure global state has mixed reduced states . . . 13

7 More operators and functions **15**
7.1 **pauli_product** tensor product of pauli matrices . . . 15
7.2 **pauliexp** expansion in terms of tensor products of pauli matrices . . . 16
7.3 **invpauliexp** inverse of expansion in terms of tensor products of pauli matrices . . . 16
7.4 **correlation_tensor** retrieve component of correlation tensor by index . . . 16
7.5 Using **pauliexp** and **invpauliexp**; an example . . . 16
7.6 **fidelity** . . . 18
7.7 **spinor_rotation_op**, **spinor_rotation_op_trig** . . . 18

8 Entanglement. longer examples **18**
8.1 Entanglement swapping . . . 18

1 Introduction

This quantum information package for the *Maxima* computer algebra system allows the manipulation of instances of objects— operators, vectors, tensors, *etc.* that appear in quantum information theory. More precisely these objects are typically represented in this package in a particular basis as row and column vectors and matrices, whose entries may be explicit numbers (of various classes) or algebraic expressions. This software occupies a niche distinct from high performance numerical linear algebra software as well as software such as the *Maxima* tensor packages that manipulate abstract mathematical objects. This document describes the functions and data in the package and how to use them with *Maxima* , assuming that you do not know much about *Maxima* , but do know quantum information theory. However, most of the examples are also found in introductory texts on quantum information. The package is intended for research and teaching in the theory of entanglement and quantum information and related fields.

Examples of the facilities of the package are

- Methods for constructing pure and mixed states and operators.
- Methods for executing standard operations found in computational linear algebra as well as the tensor product, partial trace, etc.
- Functions to compute commonly appearing quantities such as entropy and purity.

This document begins with a very brief introduction to *Maxima* emphasizing features that are important for *qinf*. The remainder of the manual is a series of small sections introducing functions with examples. The examples mostly consist of testing equations. First identities and textbook exercises are presented, in part becuase they are the natural calculations to include in the test suite from which they are taken. Then more complicated calculations are tested, as that is the initial application of the author— to check hand calculations and claims found in other documents. In this restricted sense, the package can give results on simple abstract statements: If 1) generic instances of objects are generated, and 2) a representation-invariant statement is formulated, and 3) the subexpressions are succesfully coerced into some canonical form, then defects in the statement can sometimes be detected if the statement is not true.

Some suggestions and things to be aware of in the following sections.

- You probably need to read at least a ten minute tutorial before or in conjunction with reading this document. There are several listed at the *Maxima* website, and others that a search engine can find. If you are too impatient there is a very brief introduction to *Maxima* below. The best source for most questions is the *Maxima manual*.
- There are several user interfaces to *Maxima* . All the examples here are generated using the *imaxima* package for the *emacs* editor/environment, but the results are similar to other graphical frontends to *Maxima* .

- Most functions currently work only with qubits, others for variable number of states.

1.1 Acknowledgments

Some of the ideas used in this package are inspired by the package *qdensity* http://www.pitt.edu/ tabakin/QDENSITY/ written for a proprietary symbolic algebra system. None of the code in qdensity has been borrowed for this project, however.

2 Using *Maxima*. features important for *qinf*

The package is loaded using *Maxima*'s load function like this

```
(%i1) load("qinf.mac");

(%o1)                                qinf.mac
```

There are several tutorials and manuals available for *Maxima*. Here is a very brief one focused on aiding the introduction to the *qinf* package. We will not give examples of matrices until later, but point out that the notation for matrix multiplication in *Maxima* is a dot, eg. `A . B`. If A is a $m \times n$ and B a $p \times q$ matrix, then the result is a $n \times p$ matrix. The inner product of quantum state vectors, the outer product of quantum state vectors, the composition of operators, and the mapping of one vector to another by an operator are all special cases of matrix multiplication and are all represented by the dot (along with conjugation in the case of the inner and outer products.) The remaining product, the tensor product, becomes the Kronecker product in the matrix representation of a finite dimensional Hilbert space. To agree with standard terminology, we introduce the infix operator **otimes** and the function **tensor_product** that eventually call the *Maxima* function **kronecker_product**. See the section on matrices in the *Maxima* manual.

Maxima can use exact real and complex numbers or the standard floating point approximations, or arbitrary precision floating point numbers. Numerical expressions are simplified upon entry. Each input line must be terminated by a semicolon (some interfaces do this automatically) or by a dollar sign, which suppresses the output.

```
(%i1) 1 + 1;

(%o1)                                  2
```

Assignment is denoted by a colon while function definitions are denoted by := For example, `a : b+c ;` evaluates `b+c` and assigns the result to `a`. On the other hand `a(x,y) := x^y ;` defines the function $a(x, y)$.

```
(%i2) a : 2 * 2;
```

(%o2) 4

```
(%i3) a;
```

(%o3) 4

```
(%i4) b : expand( (x+y)^4 );
```

(%o4) $y^4 + 4\,x\,y^3 + 6\,x^2\,y^2 + 4\,x^3\,y + x^4$

We suppress the output here with a dollar sign because it's big– 51 terms.

```
(%i5) b : expand( (x+y)^50 )$
(%i6) length(b);
```

(%o6) 51

Some exact numbers and floating point approximations.

```
(%i7)  1  + sqrt(2);
```

(%o7) $\sqrt{2} + 1$

```
index{floating point, conversion to}
(%i8)  1  + sqrt(2), float;
```

(%o8) 2.4142135623730949

Defining and using a function.

```
(%i9) f(x) := 3 * cos(x);
```

(%o9) $f(x) := 3\cos x$

```
(%i10) f(a);
```

(%o10) $3\cos 4$

```
(%i11) f(0);
```

(%o11) 3

Complex numbers. %i is the identifier for $i = \sqrt{-1}$.

```
(%i12) expand ( (1 + 2 * %i)^2 );
```

(%o12) $4\,i - 3$

Some special numbers are defined, such as %pi and %e.

```
(%i13) cos(%pi/2);
```

(%o13) 0

```
(%i14) %e^(%i * %pi/2);
```

(%o14) i

3 Representation of states and operators

Kets are represented by $n \times 1$ matrices, bras by $1 \times n$ matrices. The objects are represented in the z basis. Bras and kets representing the same states are related by *Maxima*'s conjugate transpose function **ctranspose**. Density operators and other operators are represented by matrices. The tensor product is represented by the Kronecker product. There is no strong typing. You are responsible for knowing that a particular vector represents a state vector in a particular space.

4 Creating instances of states

Here are some methods for creating instances of states, from scratch or from other states. Although all operators 'create' states in this sense, we omit most of them here, because they are better described as manipulating states.

4.1 ketz($i_1, \ldots, i_n$), braz($i_1, \ldots, i_n$), ketx(.), brax(.), kety(.), bray(.), multipartite eigenstates of $\sigma_x, \sigma_y, \sigma_z$

create normalized n-partite states in the z-basis. In all cases the indices are 0 or 1. The pair **ketz** and **braz** produce eigenstates of $\sigma_z^{(1)} \otimes \cdots \otimes \sigma_z^{(n)}$, with the index $i = 0$ selecting the state with eigenvalue 1 and $i = 1$ selecting the state with eigenvalue -1. In other words the ket produced represents $|i_1, i_2, \ldots, i_n\rangle$.

```
(%i3) ketz(1)
```

(%o3)
$$\begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

```
(%i4) braz(1)
```

(%o4)
$$\begin{pmatrix} 0 & 1 \end{pmatrix}$$

```
(%i5) braz(0)
```

(%o5)
$$\begin{pmatrix} 1 & 0 \end{pmatrix}$$

```
(%i6) braz(0,0)
```

(%o6)
$$\begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}$$

```
(%i7) braz(1,1)
```

(%o7)
$$\begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$$

```
(%i8) alpha[1]*braz(1,1)+alpha[0]*braz(0,0)
```

(%o8)
$$\begin{pmatrix} \alpha_0 & 0 & 0 & \alpha_1 \end{pmatrix}$$

The functions **ketx**, **brax**, **kety**, **bray** produce eigenstates of $\sigma_x^{(1)} \otimes \cdots \otimes \sigma_x^{(n)}$, or $\sigma_y^{(1)} \otimes \cdots \otimes \sigma_y^{(n)}$, with, as before, the index $i = 0$ selecting the state with eigenvalue 1 and $i = 1$ selecting the state with eigenvalue -1.

```
(%i9) brax(1)
```

(%o9) $$\begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

```
(%i10) bray(1,0,1)
```

(%o10) $$\begin{pmatrix} \frac{1}{2\sqrt{2}} & \frac{i}{2\sqrt{2}} & -\frac{i}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{i}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{i}{2\sqrt{2}} \end{pmatrix}$$

4.2 ket_n($j, i_1, \ldots, i_m$), bra_n($j, i_1, \ldots, i_m$), alternate form of ketz, etc.

These are an alternate way to call **ketx**, **kety**, etc. The index $j \in (1, 2, 3)$ is mapped to (x, y, z) and the appropriate function, eg. **ketx** is called with the remaining arguments.

4.3 Density matrix representation of a pure state (projection operator)

The projection operator (or equivalently, the density matrix) corresponding to a state vector is generated via the outer product, which is represented by the dot operator. A convenience function **toproj**(*ket*) is also provided to form a projection operator (**toproj** does not check that *ket* is normalized.) Below, we use the *Maxima* function **ctranspose** for the complex transpose. Here is $|0\rangle\langle 0|$.

```
(%i19) ketx(1) . brax(1);
```

(%o19) $$\begin{pmatrix} \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Compare this to the inner product

```
(%i20) brax(1) . ketx(1);
```

(%o20) 1

Here are different ways to make $|000\rangle\langle 000|$ and $|101\rangle\langle 101|$.

```
(%i21) is ( ketz(0,0,0) . braz(0,0,0) =  ketz(0,0,0) . ctranspose(ketz(0,0,0)) );

(%o21)                                          true

(%i22) is ( ketz(1,0,1) . braz(1,0,1) =  toproj(ketz(1,0,1)) );

(%o22)                                          true
```

There is also a function **tostate** (needs a better name) that is the inverse (up to a phase) of **toproj**— it returns the ket corresponding to a projection operator. If the input matrix is not a projection operator, the result is undefined.

```
(%i17) is ( tostate( toproj(schmidt_ket(alpha))) = schmidt_ket(alpha));

(%o17)                                          true
```

4.4 Creating state vectors with the tensor product. otimes and tensor_product

The function **tensor_product**$(v_1, \ldots, v_n)$, returns $v_1 \otimes v_2 \cdots \otimes v_n$, where v_i are vectors or matrices. The **otimes** operator is an 'infix' operator that is equivalent to the function **tensor_product**. The following uses *Maxima*'s **is**(*expr*) function which tries to determine if the predicate *expr* is true. Keep in mind that, in this example, the expressions are not analyzed abstractly, but rather vectors with integer elements are generated and compared elementwise.

```
(%i12) is(ketz(0,1) = ketz(0) otimes ketz(1))

(%o12)                                          true

(%i13) is(ketz(0,1) = tensor_product(ketz(0),ketz(1)))

(%o13)                                          true

(%i14) is(ketx(0,1,0) otimes kety(1,0,1)
           = tensor_product(ketz(0),ketx(1),ketx(0),kety(1),kety(0),kety(1)))

(%o14)                                          true
```

4.5 schmidt_ket(a). bipartite state in Schmidt basis

creates a ket in the schmidt form. This is equivalent to `sqrt(a)*ket(0,0)+ sqrt(1-a)*ket(1,1)`. This only works for qubits ($d = 2$). Note that you may need to enter *Maxima*'s `assume(a>0,1-a>0)` when manipulating this state.

4.6 bell[a,b] and belln[i] Bell vectors in computational basis

create vector bell states. **bell[a,b]** creates the state

$$|\Psi_{a,b}\rangle = \frac{1}{\sqrt{2}}|0,b\rangle + (-1)^a|1,\bar{b}\rangle, \tag{1}$$

where $a, b \in \{0,1\}$. The array **belln[i]** creates the same states where i is the decimal representation of the binary numeration (a,b). That is, $(0,1,2,3)$ corresponds to $((0,0),(0,1),(1,0),(1,1))$.

As an exercise, we will check our definitions of the Bell states by testing for orthonormality. We first define an array function that returns the inner product of two Bell states. An array function **f[x,y]** is like an ordinary function **f(x,y)** except that it can be used where an array is expected.

```
(%i2) f[x,y] := belln[x] . belln[y];
```

(%o2) $$f_{x,y} := \mathrm{belln}_x \cdot \mathrm{belln}_y$$

Create a 4×4 matrix with *Maxima*'s **genmatrix** which maps the two dimension array **f** over the indices of the matrix with the given range.

```
(%i3) genmatrix( f , 3,3,0,0);
```

(%o3) $$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

But instead of the named function **f** we could have used just a function body with *Maxima*'s **lambda** function, which returns a function that is not bound to a symbol.

```
(%i4) genmatrix( lambda( [x,y], belln[x] . belln[y]) , 3,3,0,0);
```

(%o4) $$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

It is obviously the 4×4 identity matrix. The function **identitymatrixp**(*mat*) is a predicate defined in the quantum information package in analogy to the *Maxima* function **zeromatrixp**. It returns **true** only if its argument is an identity matrix. (The symbol % refers to the previous output.

```
(%i5) identitymatrixp(%);

(%o5)                                    true
```

In the following sections, we often perform these comparisons in a single line. This is how the test appears in the regression test suite.

```
(%i6) identitymatrixp(genmatrix( lambda( [x,y], belln[x] . belln[y]) , 3,3,0,0));

(%o6)                                    true
```

We see that these four vectors are orthonormal and thus form a basis in $\mathbb{C}^2 \otimes \mathbb{C}^2$. We can also check that

$$|\Psi_{00}\rangle\langle\Psi_{00}| + |\Psi_{01}\rangle\langle\Psi_{01}| + |\Psi_{10}\rangle\langle\Psi_{10}| + |\Psi_{11}\rangle\langle\Psi_{11}| \ = \mathbb{1}_4. \tag{2}$$

```
(%i2) identitymatrixp(apply("+",map(lambda([i],toproj(belln[i])),[0,1,2,3])));

(%o2)                                    true
```

5 Creating and using operators

5.1 pauli[i]

creates the pauli matrices $(\sigma_0, \sigma_1, \sigma_2, \sigma_3) = (\mathbb{1}_2, \sigma_x, \sigma_y, \sigma_z)$.

```
(%i12) [ pauli[0], pauli[1], pauli[2], pauli[3] ];

(%o12)  [ (1 0)  (0 1)  (0 -i)  (1  0) ]
        [ (0 1), (1 0), (i  0), (0 -1) ]
```

Show that the ket $|1\rangle_x$ is an eigenvector of σ_x with eigenvalue -1.

```
(%i8) is (  pauli[1] . ket_n(1,1) = -1 * ket_n(1,1) );

(%o8)                                    true
```

Here are we check that all our definitions of the pauli matrices and kets are consistent in this sense.

```
(%i9) mapapply( lambda([i,j], is(pauli[i] . ket_n(i,j) = (-1)^j * ket_n(i,j))),
         [[1,0],[1,1],[2,0],[2,1],[3,0],[3,1]  ]);
```

(%o9) [**true**, **true**, **true**, **true**, **true**, **true**]

Here we use **anticommutator**(op_1, op_2) to test the anticommutation relations among the pauli matrices: $\{\sigma_i, \sigma_j\} = 2\delta_{i,j}$ for $i, j \in \{1, 2, 3\}$.

```
(%i3) genmatrix(lambda([i,j],  anticommutator(pauli[i],pauli[j])/2 ), 3,3,1,1);
```

(%o3)
$$\begin{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{pmatrix}$$

The *Maxima* function **mat_unblocker**, flattens the blocks in the above expression, so we can write

```
(%i4) identitymatrixp( mat_unblocker (genmatrix(lambda([i,j],
         anticommutator(pauli[i],pauli[j])/2 ), 3,3,1,1)));
```

(%o4) **true**

Now we load the **itensor** package, which provides the levi-civita tensor, and make use of the *Maxima* functions **permutations** (which returns the set of all permutations of a list) and **listify** (which turns a set into an ordered list). The *qinf* package provides **mapapply**$(func, [list1, list2, \ldots])$, which **apply**s *func* to each of the *list*s and returns a list of the results. (see the *Maxima* documentation for **apply**.) With all these, we can test the commutation relations of the pauli matrices. (In reality, the matrix definitions are not complicated, we are actually testing the other functions.): $[\sigma_i, \sigma_j] = 2i\epsilon_{i,j,k}\sigma_k$.

```
(%i5) load("itensor");
```

(%o5) /usr/share/maxima/5.15.0/share/tensor/itensor.lisp

```
(%i6) mapapply(lambda([i,j,k],zeromatrixp(commutator(pauli[i],pauli[j])
     - 2*%i*levi_civita([i,j,k])*pauli[k])), listify(permutations([1,2,3])));
```

(%o6) [**true**, **true**, **true**, **true**, **true**, **true**]

6 Entanglement– ptrace($\rho, i_1, \ldots$), entropy(ρ), purity(ρ)

Now we are ready to introduce features that are more specific to the study quantum entanglement, namely the partial trace **ptrace**, the von Neumann entropy **entropy** and **purity**.

6.1 ptrace($\rho, i_1, \ldots$), ptracen($n, \rho, i_1, \ldots$) partial trace

computes the partial trace of the density operator ρ over the component spaces given by the indices. For **ptrace** the density matrix ρ is assumed to be in $(\mathbb{C}^n)^{\otimes m}$, with $n = 2$, that is a tensor product of qubit spaces. For **ptracen** the component spaces are all n-state qudits.

In this example we create three arbitrary 3×3 matrices, and check that

$$\mathrm{Tr}_{1,2}(m_1 \otimes m_2 \otimes m_3) \ = (\mathrm{Tr}(m_1)\mathrm{Tr}(m_2))\, m_3$$

and that

$$\mathrm{Tr}_3(m_1 \otimes m_2 \otimes m_3) \ = (\mathrm{Tr}(m_3)\, (m_1 \otimes m_2).$$

Notice that we use the *Maxima* function **ratsimp** to put both sides of the equation in the same form because = checks only that two expressions are lexically identical (well, modulo some details of the simplification process.) We also make use of *Maxima*'s **mat_trace** function (so named to avoid conflicting with the code-execution trace function.)

```
(%i1) m1 : matrix([a1,b1,c1],[d1,e1,f1],[g1,h1,i1])$

(%i2) m2 : matrix([a2,b2,c2],[d2,e2,f2],[g2,h2,i2])$

(%i3) m3 : matrix([a3,b3,c3],[d3,e3,f3],[g3,h3,i3])$

(%i4) is ( ratsimp( ptracen(3, m1 otimes m2 otimes m3 , 1,2) =
            mat_trace(m1)*mat_trace(m2)*m3));

(%o4)                                          true

(%i5) is ( ratsimp( ptracen(3, m1 otimes m2 otimes m3 ,3) = mat_trace(m3)* m1 otimes m2));

(%o5)                                          true
```

Here we trace over one component repeatedly and check that the result is equal to the full trace. Note that, each time, we are tracing over the new first component.

```
(%i10) factor( ptracen(3,ptracen(3,ptracen(3,m1 otimes m2 otimes m3,1),1),1));

(%o10)                          ( (i1 + e1 + a1) (i2 + e2 + a2) (i3 + e3 + a3) )
```

6.2 entropy

entropy(ρ) returns the von Neumann **entropy** of the density matrix ρ defined by

$$S(\rho) = -\mathrm{Tr}\left(\rho \log_2 \rho\right). \tag{3}$$

6.3 purity

purity(ρ) returns the purity of the density matrix ρ defined by $\mathrm{Tr}(\rho^2)$. The purity is 1 for a pure state and is less than 1 for a mixed state.

6.4 An entangled pure global state has mixed reduced states

We examine a textbook example of entanglement— the joint state of two qubits. The state of the whole system is pure, but the local states are mixed. We begin by creating a joint state of two qubits in Schmidt basis $|\alpha\rangle = \sqrt{\alpha}|00\rangle + \sqrt{1-\alpha}|11\rangle$. In order to see the mixed character of the local states, we need to express the full state as a density operator (or equivalently as a projection operator.) Let's try to make $|\alpha\rangle\langle\alpha|$.

```
(%i2)  pr : toproj(schmidt_ket(alpha));
```

(%o2)
$$\begin{pmatrix} \sqrt{\alpha}^{\star}\sqrt{\alpha} & 0 & 0 & \sqrt{1-\alpha}^{\star}\sqrt{\alpha} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \sqrt{\alpha}^{\star}\sqrt{1-\alpha} & 0 & 0 & \sqrt{1-\alpha}^{\star}\sqrt{1-\alpha} \end{pmatrix}$$

We see that *Maxima* is allowing that the quantities under the radicals may be negative. So we set some rules, and try again.

```
(%i3) assume(alpha>0, 1-alpha>0);
```

(%o4)
$$[\alpha > 0, \alpha < 1]$$

```
(%i5)  pr : toproj(schmidt_ket(alpha));
```

(%o5)
$$\begin{pmatrix} \alpha & 0 & 0 & \sqrt{1-\alpha}\sqrt{\alpha} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \sqrt{1-\alpha}\sqrt{\alpha} & 0 & 0 & 1-\alpha \end{pmatrix}$$

The entropy vanishes for a pure state, so that $S(|\alpha\rangle\langle\alpha|)$ is

```
(%i6)  entropy(pr);
```

(%o7) 0

The **purity** is equal to 1 if and only if ρ is a pure state.

```
(%i8) purity(pr);
```

(%o8) $$\alpha^2 + 2\ (1-\alpha)\ \alpha + (1-\alpha)^2$$

The above line should be simplified by writing it as a canonical rational expression (CRE)

```
(%i9) ratsimp(%);
```

(%o9) $$1$$

Now we compute the reduced density matrix of the second qubit by tracing over the first— $\rho_2 = \mathrm{Tr}_1 |\alpha\rangle\langle\alpha|$.

```
(%i10)  pr2 : ptrace(pr,1);
```

(%o10) $$\begin{pmatrix} \alpha & 0 \\ 0 & 1-\alpha \end{pmatrix}$$

Tracing over the second qubit instead gives the same result

```
(%i11) ptrace(pr,2);
```

(%o11) $$\begin{pmatrix} \alpha & 0 \\ 0 & 1-\alpha \end{pmatrix}$$

Computing the entropy of a local state shows that this state is, in general, mixed

```
(%i12) entropy(pr2);
```

(%o12) $$-\alpha \log 2\,(\alpha) - \log 2\,(1-\alpha)\ (1-\alpha)$$

Each eigenvalue λ satisfies $0 \leq \lambda < 1$, so that the sum of their squares is less than one

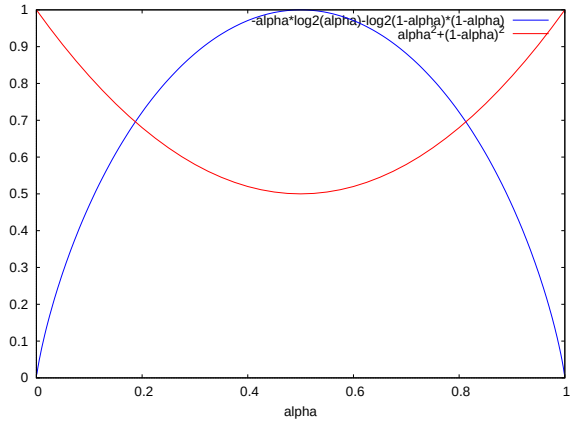
```
(%i13) purity(pr2);
```

(%o13) $$\alpha^2 + (1-\alpha)^2$$

We can plot the results (the plot command **plot2d** is more common, depending on your user interface. **wxplot2d** has the same calling syntax, but inlines the resulting plot.) We see that the maximum entanglement occurs at $\alpha = 1/2$ and decreases monotonically from there in both directions, with $\alpha = 0, 1$ giving pure joint states.

```
(%i14)  wxplot2d([entropy(pr2), purity(pr2)],[alpha,0,1]);
```

(%o14)



7 More operators and functions

The next four functions **pauli_product**$(i_1, \ldots, i_n)$, **pauliexp**(ρ), **invpauliexp**(c), and **correlation_tensor**$(c, i_1, \ldots, i_n)$ are related. An example using them follows their definitions.

7.1 pauli_product tensor product of pauli matrices

pauli_product$(i_1, \ldots, i_n)$ returns the tensor product $\sigma_{i_1} \otimes \cdots \otimes \sigma_{i_n}$, where the indices i_j are in $\{0, 1, 2, 3\}$.

7.2 pauliexp expansion in terms of tensor products of pauli matrices

pauliexp(ρ) returns the correlation tensor, that is, the coefficients in the expansion of the matrix ρ in tensor products of pauli matrices. Explicitly, **pauliexp** returns the coefficents $c_{i_1,\ldots,i_n}$ in

$$\rho \;=\; \sum_{i_1,\ldots,i_n=0}^{1} c_{i_1,\ldots,i_n}\; \sigma_{i_1} \otimes \cdots \otimes \sigma_{i_n}. \tag{4}$$

ρ must be a $2^n \times 2^n$ matrix. The coefficients are returned as a list of $2^n \times 2^n$ elements. The place of $c_{i_1,\ldots,i_n}$ in the returned list is determined by taking $i_1, \ldots, i_n$ to be the binary representation of an integer. For convenience, the coefficient can be retrieved by index with the function **correlation_tensor**.

7.3 invpauliexp inverse of expansion in terms of tensor products of pauli matrices

invpauliexp(c) is the inverse of **pauiexp**. Given a list c representing the correlation tensor (*i.e.* expansion coefficients), **invpauliexp** returns the matrix ρ given by (4).

7.4 correlation_tensor retrieve component of correlation tensor by index

correlation_tensor($c, i_1, \ldots, i_n$) returns the expansion coefficient for the term $\sigma_{i_1} \otimes \cdots \otimes \sigma_{i_n}$ in the expansion of ρ, where c is the list of coefficients in the expansion of ρ as given, for intance, by **pauliexp**.

7.5 Using pauliexp and invpauliexp; an example

Here is an example using the four functions defined above. First we create three generic 2×2 (complex) matrices.

```
(%i2) m1 : matrix([a1,b1],[c1,d1]);
```

(%o2)
$$\begin{pmatrix} a1 & b1 \\ c1 & d1 \end{pmatrix}$$

```
(%i3) m2 : matrix([a2,b2],[c2,d2]) $
(%i4) m3 : matrix([a3,b3],[c3,d3]) $
```

Here is the tensor product of the three matrices. This is *not* a generic element in the three qubit Hilbert space represented by $M(\mathbb{C}, 8)$. For instance, the three matrices have 12 complex parameters while a generic matrix in the tensor product space has 64 complex parameters.

```
(%i5) mp : m1 otimes m2 otimes  m3 ;
```

(%o5)
$$\begin{pmatrix}
\mathrm{a1\,a2\,a3} & \mathrm{a1\,a2\,b3} & \mathrm{a1\,a3\,b2} & \mathrm{a1\,b2\,b3} & \mathrm{a2\,a3\,b1} & \mathrm{a2\,b1\,b3} & \mathrm{a3\,b1\,b2} & \mathrm{b1\,b2\,b3} \\
\mathrm{a1\,a2\,c3} & \mathrm{a1\,a2\,d3} & \mathrm{a1\,b2\,c3} & \mathrm{a1\,b2\,d3} & \mathrm{a2\,b1\,c3} & \mathrm{a2\,b1\,d3} & \mathrm{b1\,b2\,c3} & \mathrm{b1\,b2\,d3} \\
\mathrm{a1\,a3\,c2} & \mathrm{a1\,b3\,c2} & \mathrm{a1\,a3\,d2} & \mathrm{a1\,b3\,d2} & \mathrm{a3\,b1\,c2} & \mathrm{b1\,b3\,c2} & \mathrm{a3\,b1\,d2} & \mathrm{b1\,b3\,d2} \\
\mathrm{a1\,c2\,c3} & \mathrm{a1\,c2\,d3} & \mathrm{a1\,c3\,d2} & \mathrm{a1\,d2\,d3} & \mathrm{b1\,c2\,c3} & \mathrm{b1\,c2\,d3} & \mathrm{b1\,c3\,d2} & \mathrm{b1\,d2\,d3} \\
\mathrm{a2\,a3\,c1} & \mathrm{a2\,b3\,c1} & \mathrm{a3\,b2\,c1} & \mathrm{b2\,b3\,c1} & \mathrm{a2\,a3\,d1} & \mathrm{a2\,b3\,d1} & \mathrm{a3\,b2\,d1} & \mathrm{b2\,b3\,d1} \\
\mathrm{a2\,c1\,c3} & \mathrm{a2\,c1\,d3} & \mathrm{b2\,c1\,c3} & \mathrm{b2\,c1\,d3} & \mathrm{a2\,c3\,d1} & \mathrm{a2\,d1\,d3} & \mathrm{b2\,c3\,d1} & \mathrm{b2\,d1\,d3} \\
\mathrm{a3\,c1\,c2} & \mathrm{b3\,c1\,c2} & \mathrm{a3\,c1\,d2} & \mathrm{b3\,c1\,d2} & \mathrm{a3\,c2\,d1} & \mathrm{b3\,c2\,d1} & \mathrm{a3\,d1\,d2} & \mathrm{b3\,d1\,d2} \\
\mathrm{c1\,c2\,c3} & \mathrm{c1\,c2\,d3} & \mathrm{c1\,c3\,d2} & \mathrm{c1\,d2\,d3} & \mathrm{c2\,c3\,d1} & \mathrm{c2\,d1\,d3} & \mathrm{c3\,d1\,d2} & \mathrm{d1\,d2\,d3}
\end{pmatrix}$$

We compute the correlation tensor of `mp`

```
(%i6) pe : pauliexp(mp) $
```

Check that the tensor has 64 elements and see what a coefficient looks like.

```
(%i7) length(pe);
```

(%o7) 64

```
(%i8) part(pe,10);
```

(%o8)
$$\frac{-i\,\mathrm{c1\,c2\,d3} - i\,\mathrm{b1\,c2\,d3} + i\,\mathrm{b2\,c1\,d3} + i\,\mathrm{b1\,b2\,d3} - i\,\mathrm{a3\,c1\,c2} - i\,\mathrm{a3\,b1\,c2} + i\,\mathrm{a3\,b2\,c1} + i\,\mathrm{a3\,b1\,b2}}{8}$$

Check that the inverse of the expansion gives the original matrix back

```
(%i9) is ( ratsimp( invpauliexp( pauliexp(mp) )) = mp);
```

(%o9) **true**

Here is the convenience function to return an element of the correlation tensor by index

```
(%i10) correlation_tensor(pe,1,2,3);
```

(%o10)
$$\frac{i\,\mathrm{c1\,c2\,d3} + i\,\mathrm{b1\,c2\,d3} - i\,\mathrm{b2\,c1\,d3} - i\,\mathrm{b1\,b2\,d3} - i\,\mathrm{a3\,c1\,c2} - i\,\mathrm{a3\,b1\,c2} + i\,\mathrm{a3\,b2\,c1} + i\,\mathrm{a3\,b1\,b2}}{8}$$

7.6 fidelity

fidelity(ρ_1, ρ_2) returns the scalar valued fidelity of the density matrices ρ_1 and ρ_2 defined by

$$\mathrm{Tr}\left(\sqrt{\sqrt{\rho_2}\rho_1\sqrt{\rho_2}}\right).$$

7.7 spinor_rotation_op, spinor_rotation_op_trig

Stub

8 Entanglement. longer examples

8.1 Entanglement swapping

Consider a pair of entangled qubits A and B, and another entangled pair C and D. By performing a joint measurement on, say B and C, we can put A and B in an entangled state although they may be widely separated. We begin by considering the most general projective measure on B and C, and calculate the reduced density matrix for a single qubit and the probability of outcome. In this example we calculate these quantities two ways— one, directly from the density matrix formalism, and two, via formulas taking advantage of the particulars of this problem. To do the first calculation by hand would be extremely upleasant, as it involves multiplying 16×16 matrices with several factors in a single element. Carrying it out below with *Maxima* is a concise exercise. This may appear to be an empty exercise only because all the calculations agree. In fact, in taking this example from a journal article, the author easily found a typographical error leading to an inconsistent result in one of the formulas below that would have been difficult to track down by hand. At present this example does not continue by discussing the measurements that maximize the resulting entanglement of A and D.

Qubits A and B are in the state

$$|\alpha\rangle \ = \sqrt{\alpha}|00\rangle + \sqrt{1-\alpha}|11\rangle,$$

C and D are in the state

$$|\beta\rangle \ = \sqrt{\beta}|00\rangle + \sqrt{1-\beta}|11\rangle,$$

with the Schmidt coefficients satisfying $\alpha, \beta > 1/2$. For now, we only want to tell *Maxima* that the coefficients of the kets are real.

```
(%i2) assume(alpha>0,1-alpha>0,beta>0,1-beta>0);
```

(%o2) $$[\alpha > 0, \alpha < 1, \beta > 0, \beta < 1]$$

```
(%i3) a : schmidt_ket(alpha);
```

(%o3) $$\begin{pmatrix} \sqrt{\alpha} \\ 0 \\ 0 \\ \sqrt{1-\alpha} \end{pmatrix}$$

```
(%i4) b : schmidt_ket(beta);
```

(%o4) $$\begin{pmatrix} \sqrt{\beta} \\ 0 \\ 0 \\ \sqrt{1-\beta} \end{pmatrix}$$

We consider the projective measurement $\{E_m\}$, that is $E_m = |u_m\rangle\langle u_m|$ and $\sum_m E_m = \mathbb{1}_4$. We consider only a single basis vector here, so we don't use the subscript m for *Maxima* vector name. We need to use *Maxima*'s **declare** to declare that the components are complex. The state $|u_m\rangle$ is normalized, but we don't need to impose that condition in *Maxima* at this point.

```
(%i5)  declare([u00,u01,u10,u11], complex);
```

(%o5) **done**

```
(%i6) u : ket(u00,u01,u10,u11);
```

(%o6) $$\begin{pmatrix} \text{u00} \\ \text{u01} \\ \text{u10} \\ \text{u11} \end{pmatrix}$$

The inital joint state $|\alpha\beta\rangle\langle\alpha\beta|$ is pure and remains so after the measurment applying $|u_m\rangle\langle u_m|$ to qubits B and C. But we write the density operator because we will examine the reduced states, which are mixed.

```
(%i7) rho : conjsimp((ident(2) otimes toproj(u) otimes ident(2)) . toproj(a otimes b))$,
```

where **conjsimp** (supplied via the *Maxima* listserv by Barton Willis) replaces xx^* with $|x|^2$. The output was suppressed with the trailing dollar sign because the ρ is a 16×16 matrix with large expressions for entries. The reduced state of qubits A and D is obtained by tracing out components 2 and 3 corresponding to qubits B and C, ie $\rho_{AD} = \text{Tr}_{BC}\rho$.

```
(%i8) rho_14 : ptrace(rho,2,3);
```

(%o8)
$$\begin{pmatrix} \alpha\,\beta\,|\mathrm{u00}|^2 & \alpha\sqrt{1-\beta}\sqrt{\beta}\,\mathrm{u00}\,\mathrm{u01}^\star & \sqrt{1-\alpha}\sqrt{\alpha}\,\beta\,\mathrm{u00}\,\mathrm{u10}^\star & \sqrt{1-\alpha}\sqrt{\alpha}\sqrt{1-\beta}\sqrt{\beta}\,\mathrm{u00}\,\mathrm{u11}^\star \\ \alpha\sqrt{1-\beta}\sqrt{\beta}\,\mathrm{u00}\,\mathrm{u01}^\star & (\alpha-\alpha\,\beta)\,|\mathrm{u01}|^2 & \sqrt{1-\alpha}\sqrt{\alpha}\sqrt{1-\beta}\sqrt{\beta}\,\mathrm{u01}\,\mathrm{u10}^\star & \left(\sqrt{1-\alpha}\sqrt{\alpha}-\sqrt{1-\alpha}\sqrt{\alpha}\,\beta\right)\mathrm{u01}\,\mathrm{u11}^\star \\ \sqrt{1-\alpha}\sqrt{\alpha}\,\beta\,\mathrm{u00}\,\mathrm{u10}^\star & \sqrt{1-\alpha}\sqrt{\alpha}\sqrt{1-\beta}\sqrt{\beta}\,\mathrm{u01}\,\mathrm{u10}^\star & (1-\alpha)\,\beta\,|\mathrm{u10}|^2 & (1-\alpha)\sqrt{1-\beta}\sqrt{\beta}\,\mathrm{u10}^\star\,\mathrm{u11} \\ \sqrt{1-\alpha}\sqrt{\alpha}\sqrt{1-\beta}\sqrt{\beta}\,\mathrm{u00}\,\mathrm{u11}^\star & \left(\sqrt{1-\alpha}\sqrt{\alpha}-\sqrt{1-\alpha}\sqrt{\alpha}\,\beta\right)\mathrm{u01}\,\mathrm{u11}^\star & (1-\alpha)\sqrt{1-\beta}\sqrt{\beta}\,\mathrm{u10}\,\mathrm{u11}^\star & ((\alpha-1)\,\beta-\alpha+1)\,|\mathrm{u11}|^2 \end{pmatrix}$$

Likewise, the reduced state of just qubit D is $\rho_D = \mathrm{Tr}_{ABC}\rho$.

```
(%i9) rho_4 : ptrace(rho,1,2,3);
```

(%o9)
$$\begin{pmatrix} (1-\alpha)\,\beta\,|\mathrm{u10}|^2+\alpha\,\beta\,|\mathrm{u00}|^2 & (1-\alpha)\sqrt{1-\beta}\sqrt{\beta}\,\mathrm{u10}^\star\,\mathrm{u11}+\alpha\sqrt{1-\beta}\sqrt{\beta}\,\mathrm{u00}^\star\,\mathrm{u01} \\ (1-\alpha)\sqrt{1-\beta}\sqrt{\beta}\,\mathrm{u10}\,\mathrm{u11}^\star+\alpha\sqrt{1-\beta}\sqrt{\beta}\,\mathrm{u00}\,\mathrm{u01}^\star & ((\alpha-1)\,\beta-\alpha+1)\,|\mathrm{u11}|^2+(\alpha-\alpha\,\beta)\,|\mathrm{u01}|^2 \end{pmatrix}$$

The second method of calculating ρ_D is as follows. Considering the following map from $\mathbb{C}^2\otimes\mathbb{C}^2$ to $M(\mathbb{C},2)$:

$$|a\rangle = \sum_{i,j=0}^{1} a_{ij}|ij\rangle \quad \mapsto \quad \widehat{a} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix}, \tag{5}$$

one can show that ρ_D is equal to $X_m X_m^\dagger$, with $X_m = \widehat{\alpha}\,\widehat{u}_m\,\widehat{\beta}$. The *Maxima* function implementing the mapping (5) is

```
(%i10) ket_to_mat(iket) := matrix([iket[1,1],iket[2,1]],[iket[3,1],iket[4,1]])$
```

Then the second calculation of ρ_D, which we call `rho_4a` is given by the following two lines.

```
(%i11)  X : ket_to_mat(a) . ket_to_mat(u) . ket_to_mat(b);
```

(%o11)
$$\begin{pmatrix} \sqrt{\alpha}\sqrt{\beta}\,\mathrm{u00} & \sqrt{\alpha}\sqrt{1-\beta}\,\mathrm{u01} \\ \sqrt{1-\alpha}\sqrt{\beta}\,\mathrm{u10} & \sqrt{1-\alpha}\sqrt{1-\beta}\,\mathrm{u11} \end{pmatrix}$$

```
(%i12) rho_4a : conjsimp(ctranspose(X) . X);
```

(%o12)
$$\begin{pmatrix} (1-\alpha)\,\beta\,|\mathrm{u10}|^2+\alpha\,\beta\,|\mathrm{u00}|^2 & (1-\alpha)\sqrt{1-\beta}\sqrt{\beta}\,\mathrm{u10}^\star\,\mathrm{u11}+\alpha\sqrt{1-\beta}\sqrt{\beta}\,\mathrm{u00}^\star\,\mathrm{u01} \\ (1-\alpha)\sqrt{1-\beta}\sqrt{\beta}\,\mathrm{u10}\,\mathrm{u11}^\star+\alpha\sqrt{1-\beta}\sqrt{\beta}\,\mathrm{u00}\,\mathrm{u01}^\star & ((\alpha-1)\,\beta-\alpha+1)\,|\mathrm{u11}|^2+(\alpha-\alpha\,\beta)\,|\mathrm{u01}|^2 \end{pmatrix}$$

Comparing `(%o9)` and `(%o12)`, we see that the two methods of calculating the reduced state for qubit D after the

measurement give the same result, showing that the second method is correct. Now we compute the probability $p_m = \mathrm{Tr}(\rho) = \mathrm{Tr}(\rho_D)$ that the state is in fact projected onto $|u_m\rangle$.

```
(%i13)  res1 : conjsimp(mat_trace(rho));
```

(%o13) $$((\alpha-1)\,\beta-\alpha+1)\,|\mathrm{u11}|^2+(1-\alpha)\,\beta\,|\mathrm{u10}|^2+(\alpha-\alpha\,\beta)\,|\mathrm{u01}|^2+\alpha\,\beta\,|\mathrm{u00}|^2$$

```
(%i14) res2 : conjsimp(mat_trace(rho_4a));
```

(%o14) $$((\alpha-1)\,\beta-\alpha+1)\,|\mathrm{u11}|^2+(1-\alpha)\,\beta\,|\mathrm{u10}|^2+(\alpha-\alpha\,\beta)\,|\mathrm{u01}|^2+\alpha\,\beta\,|\mathrm{u00}|^2$$

Finally, we compare this to the trace computed by hand from the expression following (5), which is given by

$$p_m \;=\sum_{i,j=0}^{1}\widehat{\alpha}_{i,j}^2\widehat{\beta}_{i,j}^2|\widehat{u}_{m,ij}|^2 \tag{6}$$

```
(%i15) res3 : ratsimp(braz(0,0).a^2 * braz(0,0).b^2 *abs(u00)^2+ braz(0,0).a^2 * braz(1,1).b^2 * abs(u01)^2
    +  braz(1,1).a^2 * braz(0,0).b^2 * abs(u10)^2 +  braz(1,1).a^2 * braz(1,1).b^2 * abs(u11)^2);
```

(%o15) $$((\alpha-1)\,\beta-\alpha+1)\,|\mathrm{u11}|^2+(1-\alpha)\,\beta\,|\mathrm{u10}|^2+(\alpha-\alpha\,\beta)\,|\mathrm{u01}|^2+\alpha\,\beta\,|\mathrm{u00}|^2$$

and see that p_m is the same as above— verifying this last formula was well.

Index

e base of natural log, 5
=, 12
load, 3

anticommutator, 11
apply, 11
arrays
    array functions, 9
assignment, 3
assume, 8, 13, 18

Bell states, 9
bell[a,b], 9
belln[i], 9
brax, 7
bray, 7
braz, 6

canonical rational expression, 14
commutator, 11
complex numbers, 5
    declaring variable complex, 18
conjsimp, 19, 20
ctranspose, 5, 7, 20

declare, 18
density operator
    of a pure state, 7
dot product, 3

emacs, 2
entanglement swapping, 17
entropy, 13, 14
expand, 4

fidelity, 17
functions
    user defined, 3, 4

genmatrix, 9, 11

identitymatrixp(*mat*), 9
imaginary unit *i*, 5
imaxima, 2
inner product, 3, 7
invpauliexp, 15
itensor, 11

ket_n, 7
kets
    creating a ket, *see* states representation, 5
ketx, 7
kety, 7
ketz, 6
kronecker_product, 3

lambda, 9
levi-civita tensor, 11
listify, 11
load, 21
loading *qinf* , 3

mapapply, 11
mat_trace, 12
mat_unblocker, 11
matrix multiplication, 3

orthonormality, 9
otimes, 3, 8
outer product, 3

pauli[*i*], 10
pauli_product, 15
pauliexp, 15
permutations, 11
pi π, 5
plot2d, 14
plotting, 14
projection operator, 7
projective measure, 18
ptrace, 12, 19
ptracen, 12
purity, 13, 14

qdens, 3

ratsimp, 12, 14

reduced state, 19

Schmidt basis, 8
Schmidt coefficients, 18
schmidt_ket(a), 8
spinor_rotation_op, 17
spinor_rotation_op_trig, 17
states
    creating a ket, 6
suppressing output, 4

tensor product, 3, 8
tensor_product, 3, 8
toproj, 7, 8
tostate, 8

wxplot2d, 14

zeromatrixp, 9