0.1 Introduction to qm

Package version: 0.5

The `qm` package provides functions and standard definitions to solve quantum mechanics problems in a finite dimensional Hilbert space. One can calculate the outcome of Stern-Gerlach experiments using the built-in definition of the Sx, Sy, and Sz for arbitrary spin, e.g. `s={1/2, 1, 3/2, ...}`. One can create ket vectors with arbitrary but finite dimension and perform standard computations.

With this package it is also possible to create tensor product states for multiparticle systems and to perform calculations on those systems.

The `qm` package was written by E. Majzoub, Univ. of Missouri. Email: majzoube-at-umsystem.edu

0.2 Functions and Variables for qm

`cvec` (a_1,a_2,...) [Function]

`cvec` creates a *column* vector of arbitrary dimension. The entries a_i can be any Maxima expression.

```
(%i6) load(qm)$
(%i7) cvec(1,2,3);
                                      [ 1 ]
                                      [   ]
(%o7)                                 [ 2 ]
                                      [   ]
                                      [ 3 ]
```

`rvec` (a_1,a_2,...) [Function]

`rvec` creates a *row* vector of arbitrary dimension. The entries a_i can be any Maxima expression.

```
(%i8) rvec(1,2,3);
(%o8)                          [ 1  2  3 ]
```

`ket` (c_1,c_2,...) [Function]

`ket` creates a *column* vector of arbitrary dimension. The entries c_i can be any Maxima expression. If the entries are simple symbols or coefficients of simple functions then they will be `declare`-ed complex. If one is having difficulty with getting the correct constants declared complex then one is suggested to use the `cvec` and `rvec` functions.

```
(%i10) ket(c1,c2);
                                     [ c1 ]
(%o10)                               [    ]
                                     [ c2 ]
(%i11) facts();
(%o11) [kind(c1, complex), kind(c2, complex)]
```

`bra` (c_1,c_2,...) [Function]

`bra` creates a *row* vector of arbitrary dimension. The entries c_i can be any Maxima expression. If the entries are simple symbols or coefficients of simple functions then

they will be `declare`-ed complex. If one is having difficulty with getting the correct constants declared complex then one is suggested to use the `cvec` and `rvec` functions.

```
(%i11) bra(c1,c2);
(%o11)                              [ c1  c2 ]
```

`ketp` (*vector*) [Function]

`ketp` is a predicate function that checks if its input is a ket, in which case it returns `true`, else it returns `false`.

`brap` (*vector*) [Function]

`brap` is a predicate function that checks if its input is a bra, in which case it returns `true`, else it returns `false`.

`dagger` (*vector*) [Function]

`dagger` returns the `conjugate transpose` of its input.

```
(%i13) dagger( bra(%i,2) );
                                       [ - %i ]
(%o13)                                 [      ]
                                       [  2   ]
```

`braket (psi,phi)` [Function]

Given two kets `psi` and `phi`, `braket` returns the quantum mechanical bracket `<psi|phi>`. The vector `psi` may be input as either a `ket` or `bra`. If it is a `ket` it will be turned into a `bra` with the `dagger` function before the inner product is taken. The vector `phi` must always be a `ket`.

```
(%i7) braket(ket(a,b,c),ket(a,b,c));
(%o7)          c conjugate(c) + b conjugate(b) + a conjugate(a)
```

`norm (psi)` [Function]

Given a `ket` or `bra` `psi`, `norm` returns the square root of the quantum mechanical bracket `<psi|psi>`. The vector `psi` must always be a `ket`, otherwise the function will return `false`.

The following additional examples show how to input vectors of various kinds and to do simple manipulations with them.

```
(%i1) load(qm)$
(%i2) rvec(a,b,c);
(%o2)                              [ a  b  c ]
(%i3) facts();
(%o3)        [kind(hbar, real), hbar > 0]
(%i4) bra(a,b,c);
(%o4)                              [ a  b  c ]
(%i5) facts();
(%o5) [kind(hbar, real), hbar > 0, kind(a, complex),
                  kind(b, complex), kind(c, complex)]
(%i6) braket(bra(a,b,c),ket(a,b,c));
                                   2    2    2
(%o6)                             c  + b  + a
(%i7) braket(ket(a,b,c),ket(a,b,c));
(%o7)          c conjugate(c) + b conjugate(b) + a conjugate(a)
(%i8) norm(ket(a,b,c));
(%o8)       sqrt(c conjugate(c) + b conjugate(b) + a conjugate(a))
```

0.2.1 Spin-1/2 state kets and associated operators

Spin-1/2 particles are characterized by a simple 2-dimensional Hilbert space of states. It is spanned by two vectors. In the z-basis these vectors are `{zp,zm}`, and the basis kets in the z-basis are `{xp,xm}` and `{yp,ym}` respectively.

`zp,zm,xp,xm,yp,ym` [Function]

Return the ket of the corresponding vector in the z-basis.

```
(%i6) zp;
                                   [ 1 ]
(%o6)                              [   ]
                                   [ 0 ]
(%i7) zm;
                                   [ 0 ]
(%o7)                              [   ]
                                   [ 1 ]
```

```
(%i10) yp;
                                    [    1    ]
                                    [ ------- ]
                                    [ sqrt(2) ]
(%o10)                              [         ]
                                    [   %i    ]
                                    [ ------- ]
                                    [ sqrt(2) ]
(%i11) ym;
                                   [     1     ]
                                   [  -------  ]
                                   [  sqrt(2)  ]
(%o11)                             [           ]
                                   [     %i    ]
                                   [ - ------- ]
                                   [   sqrt(2) ]
(%i6) braket(xp,zp);
                                       1
(%o6)                                -------
                                     sqrt(2)
```

Switching bases is done in the following example where a z-basis ket is constructed and the x-basis ket is computed.

```
(%i3) psi: ket(a,b);
                                     [ a ]
(%o3)                                [   ]
                                     [ b ]
(%i4) psi_x: 'xp*braket(xp,psi) + 'xm*braket(xm,psi);
                    b         a              a         b
(%o4)           (------- + -------) xp + (------- - -------) xm
                 sqrt(2)   sqrt(2)        sqrt(2)   sqrt(2)
```

0.2.2 Pauli matrices and Sz, Sx, Sy operators

`sigmax`, *sigmay*, *sigmaz* [Function]

Returns the Pauli x,y,z matrix.

`Sx`, *Sy*, *Sz* [Function]

Returns the spin-1/2 *Sx*,*Sy*,*Sz* matrix.

```
(%i3) sigmay;
                                  [ 0   - %i ]
(%o3)                             [          ]
                                  [ %i   0   ]
(%i4) Sy;
                                [            %i hbar ]
                                [    0     - ------- ]
                                [                2   ]
(%o4)                           [                    ]
                                [ %i hbar            ]
                                [ -------      0     ]
                                [    2               ]
```

0.2.3 SX, SY, SZ operators for any spin

`SX`, *SY*, *SZ* (`s`) [Function]

`SX(s)` for spin `s` returns the matrix representation of the spin operator `Sx`, and similarly for `SY(s)` and `SZ(s)`. Shortcuts for spin-1/2 are `Sx,Sy,Sz`, and for spin-1 are `Sx1,Sy1,Sz1`.

Example:

```
(%i6) SY(1/2);
                                [            %i hbar ]
                                [    0     - ------- ]
                                [                2   ]
(%o6)                           [                    ]
                                [ %i hbar            ]
                                [ -------      0     ]
                                [    2               ]
(%i7) SX(1);
                          [           hbar            ]
                          [    0     -------     0    ]
                          [          sqrt(2)           ]
                          [                            ]
                          [  hbar              hbar    ]
(%o7)                     [ -------     0      -------  ]
                          [ sqrt(2)            sqrt(2) ]
                          [                            ]
                          [           hbar             ]
                          [    0     -------     0     ]
                          [          sqrt(2)           ]
```

0.2.4 Expectation value and variance

`expect` (`O`,`psi`) [Function]

Computes the quantum mechanical expectation value of the operator `O` in state `psi`, `<psi|O|psi>`.

`qm_variance (O,psi)` [Function]

Computes the quantum mechanical variance of the operator `O` in state `psi`, `sqrt(<psi|O^2|psi> - <psi|O|psi>^2)`.

0.2.5 Angular momentum and ladder operators

`SP (s)` [function]

`SP` is the raising ladder operator S_+ for spin `s`.

`SM (s)` [function]

`SM` is the raising ladder operator S_- for spin `s`.

Examples of the ladder operators:

```
(%i4) SP(1);
                        [ 0  sqrt(2) hbar       0       ]
                        [                               ]
(%o4)                   [ 0       0        sqrt(2) hbar ]
                        [                               ]
                        [ 0       0             0       ]
(%i5) SM(1);
                        [      0             0        0 ]
                        [                             ]
(%o5)                   [ sqrt(2) hbar       0        0 ]
                        [                             ]
                        [      0        sqrt(2) hbar  0 ]
```

0.3 Rotation operators

`RX`, *RY*, *RZ* `(s,t)` [Function]

`RX(s)` for spin `s` returns the matrix representation of the rotation operator `Rx` for rotation through angle `t`, and similarly for `RY(s,t)` and `RZ(s,t)`.

```
(%i10) RZ(1/2,t);
                              [     %i t         ]
                              [   - ----         ]
                              [      2           ]
                              [ %e          0    ]
(%o10)                        [                  ]
                              [             %i t ]
                              [             ---- ]
                              [              2   ]
                              [    0      %e     ]
```

0.4 Time-evolution operator

`UU (H,t)` [Function]

`UU(H,t)` is the time evolution operator for Hamiltonian `H`. It is defined as the matrix exponential `matrixexp(-%i*H*t/hbar)`.

```
(%i12) UU(w*Sy,t),demoivre,trigreduce;
                              [     t w          t w  ]
                              [ cos(---)  - sin(---) ]
                              [      2            2   ]
(%o12)                        [                       ]
                              [     t w        t w    ]
                              [ sin(---)   cos(---)   ]
                              [      2            2   ]
```

0.5 Tensor products

Tensor products are represented as lists in Maxima. The ket tensor product `|z+,z+>` is represented as `[tpket,zp,zp]`, and the bra tensor product `<a,b|` is represented as `[tpbra,a,b]` for kets `a` and `b`. The list labels `tpket` and `tpbra` ensure calculations are performed with the correct kind of objects.

`ketprod` (k_1, k_2, ...) [Function]

`ketprod` produces a tensor product of kets k_i. All of the elements must pass the `ketp` predicate test to be accepted.

`braprod` (b_1, b_2, ...) [Function]

`braprod` produces a tensor product of bras b_i. All of the elements must pass the `brap` predicate test to be accepted.

`braketprod (B,K)` [Function]

`braketprod` takes the inner product of the tensor products `B` and `K`. The tensor products must be of the same length (number of kets must equal the number of bras).

Examples below show how to create tensor products and take the bracket of tensor products.

```
(%i3) ketprod(zp,zm);
                                      [ 1 ]  [ 0 ]
(%o3)                      [tpket, [[   ], [   ]]]
                                      [ 0 ]  [ 1 ]
(%i4) ketprod('zp,'zm);
(%o4)                         [tpket, [zp, zm]]
(%i5) braprod(bra(a,b),bra(c,d));
(%o5)                   [tpbra, [[ a  b ], [ c  d ]]]
(%i6) braprod(dagger(zp),bra(c,d));
(%o6)                   [tpbra, [[ 1  0 ], [ c  d ]]]
```

```
(%i7) zpb: dagger(zp);
(%o7)                              [ 1  0 ]
(%i8) zmb: dagger(zm);
(%o8)                              [ 0  1 ]
(%i9) K: ketprod('zp,'zm);
(%o9)                           [tpket, [zp, zm]]
(%i10) B: braprod(zpb,zmb);
(%o10)                   [tpbra, [[ 1  0 ], [ 0  1 ]]]
(%i11) B: braprod('zpb,'zmb);
(%o11)                          [tpbra, [zpb, zmb]]
(%i12) braketprod(K,B);
(%o12)                                false
(%i13) braketprod(B,K);
(%o13)                        (zmb . zm) (zpb . zp)
```