Rule-based simplification of ladder operators with Maxima

R. Bourquin

March 6, 2010

1 Basic dirac notation

We need to define the basic objects called `bra` and `ket` as they are known in quantum mechanics. Assume there are functions which form an orthonormal basis.

$$\langle \psi_m \,|\, \psi_n \rangle = \delta_{m,n}$$

The rule we need to define in maxima is simple. First we need to declare the indices to be integers and then we can ad a rule that simplifies integrals over basis functions in the way written above.

```
(%i1) declare(m, integer);
(%o12)                                done
(%i2) declare(n, integer);
(%o12)                                done
(%i4) matchdeclare(m,lambda([t],featurep(t,integer)));
(%o4)                                 done
(%i5) matchdeclare(n,lambda([t],featurep(t,integer)));
(%o5)                                 done
(%i6) tellsimp(bra(m).ket(n), kron_delta(m,n));
(%o6)                          [.rule1, simpnct]
```

Some tests show that the rule does it's job.

```
(%i9) bra(1).ket(2);
(%o9)                                  0
(%i10) bra(1).ket(1);
(%o10)                                 1
(%i11) bra(i).ket(j);
(%o11)                          bra(i) . ket(j)
(%i12) declare(i,integer);
(%o12)                                done
(%i13) declare(j,integer);
(%o13)                                done
(%i14) bra(i).ket(j);
(%o14)                         kron_delta(i, j)
```

To get the fully simplified kronecker delta we need to tell maxima that `i` and `j` are integers.

To get further simplification we will need also to tell maxima some facts about `n` and `ket(n)`. The variables `n` and `m` are integers and therefore scalars. We need to define this for all variables we would like to use as argument of the function `ket`. The functions `bra` and `ket` are not scalar, this has to be declared too.

```
(%i75) declare(m,scalar);
(%o75)                               done
(%i76) declare(n,scalar);
(%o76)                               done
(%i77) declare(bra, nonscalar);
(%o77)                               done
(%i78) declare(ket, nonscalar);
(%o78)                               done
```

Further we need to set a maxima internal flag for simplification.

```
(%i41) dotscrules:true;
(%o41)                               true
```

The maxima help says about this flag the following.

```
(%i42) ?? dotscrules
 -- Option variable: dotscrules
     Default value: `false'

     When `dotscrules' is `true', an expression `A.SC' or `SC.A'
     simplifies to `SC*A' and `A.(SC*B)' simplifies to `SC*(A.B)'.
```

2 Ladder operators

Now let's look at the ladder operators. They transform a state $|\psi_n\rangle$ into another state $|\psi'_n\rangle$. As an exampe we use the two ladder operators from the harmonic oscillator. There a a lowering operator $\mathcal{L}$ and a raising operator $\mathcal{R}$. They act as follows:

$$\begin{aligned}\mathcal{R}\,|\Psi_n\rangle &= \sqrt{n+1}\,|\Psi_{n+1}\rangle\\ \mathcal{L}\,|\Psi_n\rangle &= \sqrt{n}\,|\Psi_{n-1}\rangle\end{aligned}$$

Now we use these definitions as simplifying rules and implement the action of both ladder operators on a `ket`. We also define the operators to be linear.

```
(%i36) tellsimp(L.ket(n), sqrt(n)*ket(n-1));
(%o36)          [.rule5, .rule4, .rule3, .rule2, .rule1, simpnct]
(%i37) tellsimp(R.ket(n), sqrt(n+1)*ket(n+1));
(%o37)      [.rule6, .rule5, .rule4, .rule3, .rule2, .rule1, simpnct]
(%i71) declare(L, linear);
(%o71)                               done
(%i72) declare(R, linear);
(%o72)                               done
```

Again some very simple tests show that these rules really work and simplifying expressions with ladder operators.

```
(%i79) L.ket(n);
(%o79)                      sqrt(n) ket(n - 1)
(%i80) L.L.ket(n);
(%o80)                sqrt(n - 1) sqrt(n) ket(n - 2)
(%i81) L.L.L.ket(n);
(%o81)          sqrt(n - 2) sqrt(n - 1) sqrt(n) ket(n - 3)
(%i82) R.ket(n);
(%o82)                    sqrt(n + 1) ket(n + 1)
(%i83) R.R.ket(n);
(%o83)              sqrt(n + 1) sqrt(n + 2) ket(n + 2)
(%i84) R.R.R.ket(n);
(%o84)        sqrt(n + 1) sqrt(n + 2) sqrt(n + 3) ket(n + 3)
```

We even can derive the so called *number operator* $\mathcal{N}$. The mathematical definition is:

$$\mathcal{N} \left|\Psi_n\right\rangle = \mathcal{R}\mathcal{L} \left|\Psi_n\right\rangle = n \left|\Psi_n\right\rangle$$

With all rules we told maxima until now we can simply write:

```
(%i49) R.L.ket(n);
(%o49)                            n ket(n)
```

and get the correct result.

There is one more relation which deserves our attention: the *commutator* of these operators. It is defined as:

$$[\mathcal{L}, \mathcal{R}] = \mathcal{I}$$

where $\mathcal{I}$ denotes the identity operator. Therefore we first need to implement this operator before we can make use of this relation.

```
(%i64) tellsimp(I.ket(n), ket(n));
(%o64)            [.rule4, .rule3, .rule2, .rule1, simpnct]
(%i73) declare(I, linear);
(%o73)                              done
```

Now we can implement a simplifictaion rule which makes use of the commutation relation.

```
(%i74) tellsimp(R.L, L.R-I);
(%o74)        [.rule5, .rule4, .rule3, .rule2, .rule1, simpnct]
```

Some simple tests showing this rule.

```
(%i75) R.L;
(%o75)                            L . R - I
(%i76) L.R-R.L;
(%o76)                                I
(%i77) (L.R-R.L).ket(i);
(%o77)                             ket(i)
```

Coming back to the number operator we have now some more simplifications. But some of them are not done automatically.

```
(%i90) N : R.L;
(%o90)                           L . R - I
(%i91) N.ket(i);
(%o91)                     (L . R - I) . ket(i)
(%i92) expand(N.ket(i));
(%o92)                           i ket(i)
```

3 More complicated operator expressions

Simplifications of expressions like $(\mathcal{R}+\mathcal{L})^k$ are not trivial and require some more rules. First we will need rules to reduce the power of single operators. This can be done like this.

```
(%i1) declare(k, integer);
(%o1)                                done
(%i2) matchdeclare(k,lambda([t],featurep(t,integer)));
(%o2)                                done
(%i8) tellsimp(I^^k, I);
(%o8)                       [^^rule1, simpncexpt]
(%i6) tellsimp((L^^k).ket(n), (L^^(k-1)).(L.ket(n)));
(%o6)                         [.rule1, simpnct]
(%i7) tellsimp((R^^k).ket(n), (R^^(k-1)).(R.ket(n)));
(%o7)                     [.rule2, .rule1, simpnct]
```

Further we also need some rules to reduce mixed prodcuts.

```
(%i18) tellsimp(((L.R)^^k).ket(n), ((L.R)^^(k-1)).(L.R.ket(n)));
(%o18)                [.rule3, .rule2, .rule1, simpnct]
(%i19) tellsimp(((R.L)^^k).ket(n), ((R.L)^^(k-1)).(R.L.ket(n)));
(%o19)            [.rule4, .rule3, .rule2, .rule1, simpnct]
```

As usual some examples that show the rules in action.

```
(%i14) L.L.R.R.ket(n);
(%o14)                     (n + 1) (n + 2) ket(n)
(%i17) (L^^2).(R^^2).ket(n);
(%o17)                     (n + 1) (n + 2) ket(n)
(%i15) R.R.L.L.ket(n);
(%o15)                        (n - 1) n ket(n)
(%i16) (R^^2).(L^^2).ket(n);
(%o16)                        (n - 1) n ket(n)
```

But these rules do not suffice for any case. Therefore some more complicated expressions do not work yet. In some cases additional `expand` calls are necessary.

```
(%i35) expand(expand(((L.R.L.R)^^2).ket(j)));
           4              3              2
(%o35)    j  ket(j) + 4 j  ket(j) + 6 j  ket(j) + 4 j ket(j) + ket(j)
(%i34) expand(expand(((R.R.L.R)^^2).ket(j)));
                               <2>          <2>
(%o34)                       (R    . L . R)    . ket(j)
```

4 Matrix representation

The ladder operators can be represented as *infinite* matrices. We can construct these matrices quite easily but of course only a finite part.

From the theory we know the results for the operators $\mathcal{R}$, $\mathcal{L}$ and $\mathcal{N}$. An entry in the matrix M representing an operator A can be written as follows:

$$M_{i,j} := \langle i \,|\, A \,|\, j \rangle$$

Therefore we can derive the following matrices:

$$\begin{aligned} M^L_{i.j} &:= \langle i \,|\, \mathcal{L} \,|\, j \rangle \\ M^R_{i.j} &:= \langle i \,|\, \mathcal{R} \,|\, j \rangle \\ M^N_{i.j} &:= \langle i \,|\, \mathcal{N} \,|\, j \rangle \end{aligned}$$

The actual matrices look like this:

$$M^L = \begin{pmatrix} 0 & \sqrt{1} & 0 & 0 & \dots & 0 & \dots \\ 0 & 0 & \sqrt{2} & 0 & \dots & 0 & \dots \\ 0 & 0 & 0 & \sqrt{3} & \dots & 0 & \dots \\ 0 & 0 & 0 & 0 & \ddots & \vdots & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \sqrt{n} & \dots \\ 0 & 0 & 0 & 0 & \dots & 0 & \ddots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$M^R = \begin{pmatrix} 0 & 0 & 0 & \dots & \dots \\ \sqrt{1} & 0 & 0 & \dots & \dots \\ 0 & \sqrt{2} & 0 & \dots & \dots \\ 0 & 0 & \sqrt{3} & \dots & \dots \\ \vdots & \vdots & \vdots & & \\ 0 & 0 & 0 & \sqrt{n+1} & 0\dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

Now let's calculate these matrices in maxima. The code is just one line, we use the `genmatrix` function provided by maxima.

```
(%i61) ML: genmatrix(lambda([i,j], bra(i-1).L.ket(j-1)), 6, 6);
                      [ 0  1     0         0     0     0    ]
                      [                                     ]
                      [ 0  0  sqrt(2)      0     0     0    ]
                      [                                     ]
                      [ 0  0     0      sqrt(3)  0     0    ]
(%o61)                [                                     ]
                      [ 0  0     0         0     2     0    ]
                      [                                     ]
```

```
                     [ 0  0     0         0     0  sqrt(5) ]
                     [                                     ]
                     [ 0  0     0         0     0     0    ]
```

and in analogy we get for the other one

```
(%i62) MR: genmatrix(lambda([i,j], bra(i-1).R.ket(j-1)), 6, 6);
                     [ 0     0         0     0     0     0 ]
                     [                                   ]
                     [ 1     0         0     0     0     0 ]
                     [                                   ]
                     [ 0  sqrt(2)      0     0     0     0 ]
(%o62)               [                                   ]
                     [ 0     0      sqrt(3)  0     0     0 ]
                     [                                   ]
                     [ 0     0         0     2     0     0 ]
                     [                                   ]
                     [ 0     0         0     0  sqrt(5)  0 ]
```

It's is possible to calculate the matrix of the number operator with matrix algebra only. It holds that $M^N = M^R M^L$ and this yields:

```
(%i63) MR.ML;
                               [ 0  0  0  0  0  0 ]
                               [                  ]
                               [ 0  1  0  0  0  0 ]
                               [                  ]
                               [ 0  0  2  0  0  0 ]
(%o63)                         [                  ]
                               [ 0  0  0  3  0  0 ]
                               [                  ]
                               [ 0  0  0  0  4  0 ]
                               [                  ]
                               [ 0  0  0  0  0  5 ]
```

5 Harmonic oscillator

With all the tools from above we can now solve the *harmonic oscillator* with the help of ladder operators. The hamiltonian of the harmonic oscillator is known to be

$$\mathcal{H} = \frac{p^2}{2m} + \frac{1}{2}\omega^2 x^2$$

where x is the *position operator* and p the *momentum operator* We now define the ladder operators in terms of x and p and get

$$\begin{aligned}
\mathcal{L} &= \sqrt{\frac{m\omega}{2\hbar}}\left(x + \frac{i}{m\omega}p\right) \\
\mathcal{R} &= \sqrt{\frac{m\omega}{2\hbar}}\left(x - \frac{i}{m\omega}p\right)
\end{aligned}$$

They still act on any $|\Psi_n\rangle$ as described above. These system of linear equations can be solved for x and p and yields

$$\begin{aligned} x &= \sqrt{\frac{\hbar}{2m\omega}}\,(\mathcal{R}+\mathcal{L}) \\ p &= i\sqrt{\frac{\hbar m\omega}{2}}\,(\mathcal{R}-\mathcal{L}) \end{aligned}$$

Now plug this into the hamiltonian and get $\mathcal{H}$ in terms of $\mathcal{R}$ and $\mathcal{L}$

$$\mathcal{H} = \frac{\hbar\omega}{2}\,(\mathcal{R}\mathcal{L}+\mathcal{I})$$

This can be done in maxima in pretty the same way.

```
(%i161) declare(alpha, scalar);
(%o161)                               done
(%i162) declare(beta, scalar);
(%o162)                               done

(%i157) x : alpha*(R+L);
(%o157)                           alpha (R + L)
(%i159) p : beta*(R-L);
(%o159)                           beta (R - L)

(%i163) p.p/(2*u)+1/2*w^2*x.x;
                         2  2        <2>       2        <2>
                    alpha  w  (R + L)      beta  (R - L)
(%o163)             -------------------- + ----------------
                             2                   2 u


                <2>           <2>
        hbar w R      hbar w R      hbar w (L . R)   hbar w (L . R)
(%o182) ----------- - ----------- + -------------- + --------------
            4 u           4              2 u               2
                                                <2>           <2>
                                       hbar w L      hbar w L      hbar w I   hbar w I
                                     + ----------- - ----------- - -------- - --------
                                           4 u           4           4 u         4
```

Whats about simplification? (Need to elaborate this point) Assume a successfull simplification now. The hamiltonian is finnaly simplified to

```
(%i143) HAM:h*w/2*(R.L+I);
                                   h w (L . R)
(%o143)                            -----------
                                        2
```

Therefore `HAM` acts on a `ket` like

```
(%i151) HAM.ket(n);
                                h (n + 1) ket(n) w
(%o151)                         ------------------
                                        2
```

Now we can construct the matrix of `HAM` and calculate the eigenvalues to get the energy levels of the harmonic oscillator.

```
(%i153) HAMIL: genmatrix(lambda([i,j], expand(bra(i-1).HAM.ket(j-1))), 6, 6);
                    [ h w                                   ]
                    [ ---    0     0      0      0      0   ]
                    [  2                                    ]
                    [                                       ]
                    [  0    h w    0      0      0      0   ]
                    [                                       ]
                    [            3 h w                      ]
                    [  0     0   -----    0      0      0   ]
(%o153)             [              2                        ]
                    [                                       ]
                    [  0     0     0    2 h w    0      0   ]
                    [                                       ]
                    [                          5 h w        ]
                    [  0     0     0      0    -----    0   ]
                    [                            2          ]
                    [                                       ]
                    [  0     0     0      0      0    3 h w ]
```

For the first 6 energy levels we get the following expressions.

```
(%i155) eigenvalues(HAMIL);
           h w  3 h w  5 h w
(%o155)  [[---, -----, -----, h w, 2 h w, 3 h w], [1, 1, 1, 1, 1, 1]]
            2     2      2
```

And we are done.

6 Angular momentum operators

It's possible to generalize this maxima approach and use it for the angular momentum operators too.

$$\begin{aligned} J_+ \left|j,m\right\rangle &= \hbar\sqrt{(j-m)(j+m+1)}\left|j,m+1\right\rangle \\ J_- \left|j,m\right\rangle &= \hbar\sqrt{(j+m)(j-m+1)}\left|j,m-1\right\rangle \end{aligned}$$