Quantum Operator Algebra and Commutator Algebra in *Mathematica*

by José Luis Gómez-Muñoz
http://homepage.cem.itesm.mx/lgomez/quantum/
jose.luis.gomez@itesm.mx
with contributions by Gianfranco Arroyo-Orrico and Sergio E. Martínez-Casas

Introduction

This is a tutorial on the use of Quantum *Mathematica* add-on to work with algebra of operators and commutators. The Quantum add-on modifies the behaviour of standard *Mathematica* commands Expand, ExpandAll and Simplify. Furthermore, new commands CollectFromLeft and CollectFromRight are defined for noncommutative factorization, as well as CommutatorExpand, EvaluateCommutators and EvaluateAllCommutator for generating and working with commutators and anticommutators.

Load the Package

First load the Quantum`Notation` package. Write:
Needs["Quantum`Notation`"];
then press at the same time the keys SHIFT-ENTER to evaluate. *Mathematica* will load the package:

```
Needs["Quantum`Notation`"]
```

```
Quantum`Notation` Version 2.2.0. (July 2010)
A Mathematica package for Quantum calculations in Dirac bra-ket notation
by José Luis Gómez-Muñoz

Execute SetQuantumAliases[] in order to use
  the keyboard to enter quantum objects in Dirac's notation
SetQuantumAliases[] must be executed again in each new
  notebook that is created, only one time per notebook.
```

In order to use the keyboard to enter quantum objects write:
SetQuantumAliases[ ];
then press at the same time the keys SHIFT-ENTER to evaluate. The semicolon prevents *Mathematica* from printing the help message. Remember that SetQuantumAliases[ ] must be evaluated again in each new notebook:

```
SetQuantumAliases[];
```

Noncommutative Algebraic Manipulations

We define a,b,c,d as operators:

```
Needs["Quantum`Notation`"];
Clear[a, b, c, d];
SetQuantumObject[a, b, c, d];
```

The Quantum command "CollectFromLeft" collects common left factors in noncommutative algebraic expresions

```
CollectFromLeft[a · b + a · c + b · c]
```

```
b · c + a · (b + c)
```

The Quantum command "CollectFromRight" collects common right factors in noncommutative algebraic expressions

```
CollectFromRight[a · b + a · c + b · c]
```

```
a · b + (a + b) · c
```

The command "Expand" is generalized to work in noncommutative algebraic expressions:

```
Expand[(a + b)^3]
```

$a^3 + b^3 + b^2 \cdot a + b \cdot a^2 + a^2 \cdot b + a \cdot b^2 + a \cdot b \cdot a + b \cdot a \cdot b$

Copy-paste the output from the previous calculation and use it as an argument of "CollectFromLeft"

```
CollectFromLeft[a^3 + b^3 + b^2 · a + b · a^2 + a^2 · b + a · b^2 + a · b · a + b · a · b]
```

```
a · (a · (a + b) + b · (a + b)) + b · (a · (a + b) + b · (a + b))
```

Apply "CollectFromLeft" and "CollectFromRight" to the output from the "Expand" calculation:

```
CollectFromRight[CollectFromLeft[a^3 + b^3 + b^2 · a + b · a^2 + a^2 · b + a · b^2 + a · b · a + b · a · b]]
```

$(a + b)^3$

Assignments to noncommutative products

We define e,f,g,h as operators:

```
Needs["Quantum`Notation`"];
Clear[e, f, g, h];
SetQuantumObject[e, f, g, h];
```

We assign the operator g as the result of the noncommutative product e·f

```
e · f = g
```

```
g
```

The assigned result is automatically used in simple expressions:

```
e · f + h
```

```
g + h
```

It is also automatically used in simple calculations involving Hermitian Conjugates:

```
f† · e†
```

```
g†
```

The expression $e^4 \cdot f^3$ can be written as $e^3 \cdot e \cdot f \cdot f^2$, **however** the rule for e·f is **not** automatically applied:

```
e^4 · f^3
```

```
e^4 · f^3
```

We can force the application of the rule for e·f using the command "Expand"

```
Expand[e^4 · f^3]
```

```
e^3 · g · f^2
```

Assignments to Powers of Operators

The standard *Mathematica* behavior is that you **cannot** assign (with the notation used below) a value to a Power without unproctecting the command Power:

```
Clear[r];
r^2 = t
```

Set::write : Tag Power in r^2 is Protected. ≫

```
t
```

However the Quantum add-on modifies the behavior of *Mathematica* so that it is possible to assign values to Powers of operators with that notation. First we define r,t as operators:

```
Needs["Quantum`Notation`"];
Clear[r, t];
SetQuantumObject[r, t];
```

After defining r as a quantum operator, it is possible to assign a value to a power of r with this notation:

```
r^2 = t
```

```
t
```

Then the definition is used both with r^2 and $r \cdot r$

```
{r^2, r · r}
```

```
{t, t}
```

The form of the answer depends on the form of the input

```
{r^3, r · r · r}
```

```
{t · r, r · t}
```

The form of the answer depends on the form of the input

```
{r^5, r · r · r · r · r}
```

```
{t^2 · r, r · t^2}
```

Expand to Commutators

We define a,b,c,d as operators:

```
Needs["Quantum`Notation`"];
Clear[a, b, c, d];
SetQuantumObject[a, b, c, d];
```

The command "CommutatorExpand" rewrites noncommutative products to products with the variables in order (alfabetical order in this case) plus or minus the necessary commutators:

```
CommutatorExpand[b · a]
```

```
-⟦a, b⟧_ + a · b
```

The "TraditionalForm" format of the expression can be easier to read (press the keys [ESC]comm[ESC] in order to enter the commutator template, or copy-paste the output from the previous calculation):

```
TraditionalForm[-⟦a, b⟧_ + a · b]
```

$ab - [a, b]$

The expansion can be done using anticommutators:

```
CommutatorExpand[b · a, Anticommutators → True]
```

```
⟦a, b⟧₊ - a · b
```

The "TraditionalForm" format of the expression can be easier to read (press the keys [ESC]anti[ESC] in order to enter the anticommutator template, or copy-paste the output from the previous calculation):

```
TraditionalForm[⟦a, b⟧₊ - a · b]
```

$\{a, b\} - ab$

On the other hand, "CommutatorExpand" has no effect in noncommutative products where the variables are already in order (alfabetical order in this case):

```
CommutatorExpand[a · b]
```

```
a · b
```

We can force the generation of products in reverse order with the option ReverseOrdering:

```
CommutatorExpand[a · b, ReverseOrdering → True]
```

```
- ⟦b, a⟧₋ + b · a
```

Here the expansion uses anticommutators and leaves the variables in reverse order:

```
CommutatorExpand[a · b, ReverseOrdering → True, Anticommutators → True]
```

```
⟦b, a⟧₊ - b · a
```

The command "EvaluateCommutators" transforms commutators [a,b] into ab-ba. Press [ESC]comm[ESC] in order to enter the commutator template ⟦□, □⟧₋

```
EvaluateCommutators[⟦a, b⟧₋ + ⟦c, d⟧₋]
```

```
-b · a + a · b - d · c + c · d
```

The command "EvaluateCommutators" transforms anticommutators {a,b} into ab+ba. Press [ESC]anti[ESC] in order to enter the anitcommutator template ⟦□, □⟧₊

```
EvaluateCommutators[ ⟦a, b⟧₊ + ⟦c, d⟧₊ ]
```

```
b · a + a · b + d · c + c · d
```

This is the "CommutatorExpand" of the square of a binomial:

```
CommutatorExpand[(a + b · c)^2]
```

```
a^2 + (-⟦a, b⟧_ + a · b) · c + b^2 · c^2 - b · ⟦a, c⟧_ + a · b · c - b · ⟦b, c⟧_ · c
```

You can "Expand" the "CommutatorExpand" of an expression:

```
Expand[CommutatorExpand[(a + b · c)^2]]
```

```
a^2 - ⟦a, b⟧_ · c + b^2 · c^2 - b · ⟦a, c⟧_ + 2 a · b · c - b · ⟦b, c⟧_ · c
```

The "TraditionalForm" format of the expression can be easier to read:

```
TraditionalForm[Expand[CommutatorExpand[(a + b · c)^2]]]
```

$-[a, b]c - b[a, c] - b[b, c]c + 2\,abc + b^2c^2 + a^2$

Nested Commutators

We define a,b,c,d as operators:

```
Needs["Quantum`Notation`"];
Clear[a, b, c, d];
SetQuantumObject[a, b, c, d];
```

Commutators are (by default) not affected by "CommutatorExpand" (press the keys [ESC]comm[ESC] in order to enter the commutator template)

```
CommutatorExpand[ ⟦a, b⟧_ · c ]
```

```
⟦a, b⟧_ · c
```

However the option "NestedCommutators → True" makes "CommutatorExpand" work with commutators as with any other expression, producing commutators of commutators (press the keys [ESC]comm[ESC] in order to enter the commutator template):

```
CommutatorExpand[ ⟦a, b⟧_ · c, NestedCommutators → True ]
```

```
-⟦c, ⟦a, b⟧_⟧_ + c · ⟦a, b⟧_
```

Copy-paste the output from the previous calcuation and use it as argument of "EvaluateCommutators". Notice that only the outermost commutator is evaluated:

```
EvaluateCommutators[-⟦c, ⟦a, b⟧_⟧_ + c · ⟦a, b⟧_]
```

```
⟦a, b⟧_ · c - c · ⟦a, b⟧_ + c · (-b · a + a · b)
```

If "Evaluate**All**Commutators" is used instead of "EvaluateCommutators", then all commutators are evaluated:

```
EvaluateAllCommutators[-⟦c, ⟦a, b⟧₋⟧₋ + c · ⟦a, b⟧₋]
```

```
(-b · a + a · b) · c
```

Copy-paste the output from the previous calculation and use it as argument of "CommutatorExpand". The original expression is recovered:

```
CommutatorExpand[ (-b · a + a · b) · c ]
```

```
⟦a, b⟧₋ · c
```

Another example with "NestedCommutators → True":

```
CommutatorExpand[ (⟦a, b⟧₋ + ⟦b, c⟧₋)^2, NestedCommutators → True ]
```

```
⟦a, b⟧₋^2 + ⟦b, c⟧₋^2 - ⟦⟦a, b⟧₋, ⟦b, c⟧₋⟧₋ + 2 ⟦a, b⟧₋ · ⟦b, c⟧₋
```

Another example with "NestedCommutators → True":

```
CommutatorExpand[ (⟦a, b⟧₋ + ⟦b, c⟧₋)^3, NestedCommutators → True ]
```

```
⟦a, b⟧₋^3 + ⟦b, c⟧₋^3 + ⟦⟦a, b⟧₋, ⟦⟦a, b⟧₋, ⟦b, c⟧₋⟧₋⟧₋ +
 2 ⟦⟦b, c⟧₋, ⟦⟦a, b⟧₋, ⟦b, c⟧₋⟧₋⟧₋ + 3 ⟦a, b⟧₋^2 · ⟦b, c⟧₋ + 3 ⟦a, b⟧₋ · ⟦b, c⟧₋^2 -
 3 ⟦a, b⟧₋ · ⟦⟦a, b⟧₋, ⟦b, c⟧₋⟧₋ - 3 ⟦b, c⟧₋ · ⟦⟦a, b⟧₋, ⟦b, c⟧₋⟧₋
```

```
TraditionalForm[CommutatorExpand[ (⟦a, b⟧₋ + ⟦b, c⟧₋)^3, NestedCommutators → True ]]
```

$$3\,[a, b]^2[b, c] + 3\,[a, b][b, c]^2 - 3\,[a, b][[a, b], [b, c]] - 3\,[b, c][[a, b], [b, c]] + [[a, b], [[a, b], [b, c]]] + 2\,[[b, c], [[a, b], [b, c]]] + [a, b]^3 + [b, c]^3$$

Commutator Properties

We define a,b,c,d as operators:

```
Needs["Quantum`Notation`"];
Clear[a, b, c, d];
SetQuantumObject[a, b, c, d];
```

Commutators automatically evolve to have their arguments in order, which is alphabetical order in this case, and a general "canonical" *Mathematica* order in general (see the documentation for the standard *Mathematica* command Sort[]). Press the keys [ESC]comm[ESC] in order to enter the commutator template:

$$[\![b, a]\!]_-$$

$$-[\![a, b]\!]_-$$

A noncommutative product inside a commutator is automatically expanded:

$$[\![a \cdot b, c]\!]_-$$

$$[\![a, c]\!]_- \cdot b + a \cdot [\![b, c]\!]_-$$

Additions inside a commutator are automatically expanded:

$$[\![a + b, c + d]\!]_-$$

$$[\![a, c]\!]_- + [\![a, d]\!]_- + [\![b, c]\!]_- + [\![b, d]\!]_-$$

Powers inside a commutator are automatically expanded:

$$[\![a^3, b]\!]_-$$

$$[\![a, b]\!]_- \cdot a^2 + a^2 \cdot [\![a, b]\!]_- + a \cdot [\![a, b]\!]_- \cdot a$$

Powers inside a commutator are automatically expanded:

$$[\![a^3, b^3]\!]_-$$

$$\left([\![a, b]\!]_- \cdot b^2 + b^2 \cdot [\![a, b]\!]_- + b \cdot [\![a, b]\!]_- \cdot b\right) \cdot a^2 + a^2 \cdot \left([\![a, b]\!]_- \cdot b^2 + b^2 \cdot [\![a, b]\!]_- + b \cdot [\![a, b]\!]_- \cdot b\right) + a \cdot \left([\![a, b]\!]_- \cdot b^2 + b^2 \cdot [\![a, b]\!]_- + b \cdot [\![a, b]\!]_- \cdot b\right) \cdot a$$

As usual, it can be better to (further) "Expand" expressions

$$\text{Expand}\left[[\![a^3, b^3]\!]_-\right]$$

$$[\![a, b]\!]_- \cdot b^2 \cdot a^2 + b^2 \cdot [\![a, b]\!]_- \cdot a^2 + a^2 \cdot [\![a, b]\!]_- \cdot b^2 + a^2 \cdot b^2 \cdot [\![a, b]\!]_- + a \cdot [\![a, b]\!]_- \cdot b^2 \cdot a + a \cdot b^2 \cdot [\![a, b]\!]_- \cdot a + b \cdot [\![a, b]\!]_- \cdot b \cdot a^2 + a^2 \cdot b \cdot [\![a, b]\!]_- \cdot b + a \cdot b \cdot [\![a, b]\!]_- \cdot b \cdot a$$

As usual, it can be easier to read the expression in "TraditionalForm":

$$\text{TraditionalForm}\left[\text{Expand}\left[[\![a^3, b^3]\!]_-\right]\right]$$

$$[a, b]b^2a^2 + b^2[a, b]a^2 + a^2[a, b]b^2 + a^2b^2[a, b] + b[a, b]ba^2 + a^2b[a, b]b + a[a, b]b^2a + ab^2[a, b]a + ab[a, b]ba$$

Some identities are not immediately recognized by *Mathematica*

Mathematica automatically identifies simple arithmetic identities. Notice that tests like this one have double-equal == between the left-hand side and the right-hand side:

```
2 + 2 == 3 + 1
```

```
True
```

However *Mathematica* does **not** automatically answer True in this case:

```
Cos[t]^2 + Sin[t]^2 == 1
```

```
Cos[t]^2 + Sin[t]^2 == 1
```

Simplify gives the expected answer:

```
Simplify[ Cos[t]^2 + Sin[t]^2 == 1 ]
```

```
True
```

We define a,b,c,d as operators:

```
Needs["Quantum`Notation`"];
Clear[a, b, c, d];
SetQuantumObject[a, b, c, d];
```

Similar to the trigonometric identities, the "Jacobi Identity" is **not** automatically recognized by *Mathematica*:

```
⟦a, ⟦b, c⟧_⟧_ + ⟦b, ⟦c, a⟧_⟧_ + ⟦c, ⟦a, b⟧_⟧_ == 0
```

```
⟦a, ⟦b, c⟧_⟧_ - ⟦b, ⟦a, c⟧_⟧_ + ⟦c, ⟦a, b⟧_⟧_ == 0
```

Simplify gives the expected answer (the Jacobi Identity):

```
Simplify[ ⟦a, ⟦b, c⟧_⟧_ + ⟦b, ⟦c, a⟧_⟧_ + ⟦c, ⟦a, b⟧_⟧_ == 0]
```

```
True
```

Simplify of noncommutative expresions

We define a,b,c,d as operators:

```
Needs["Quantum`Notation`"];
Clear[a, b, c, d];
SetQuantumObject[a, b, c, d];
```

This is an noncommutative expansion whose result will be used below:

```
Expand[(a + b) · (c + d)^2]
```

```
a · c^2 + b · c^2 + a · d^2 + b · d^2 + a · d · c + b · d · c + a · c · d + b · c · d
```

Copy-paste the output from the previous calculation and use it as argument of "Simplify". In this case *Mathematica* gets an equivalent expression that is very close to the original one, using an anticommutator:

```
Simplify[a · c^2 + b · c^2 + a · d^2 + b · d^2 + a · d · c + b · d · c + a · c · d + b · c · d]
```

```
(a + b) · (c^2 + d^2 + ⟦c, d⟧_+)
```

As usual, it can be easier to read the expression in "TraditionalForm":

```
TraditionalForm[(a + b) · (c^2 + d^2 + ⟦c, d⟧_+)]
```

$(a+b)(\{c, d\} + c^2 + d^2)$

Assignments to Commutators (Commutation Relations)

We define m,n,o,q as operators:

```
Needs["Quantum`Notation`"];
Clear[m, n, p, q];
SetQuantumObject[m, n, p, q];
```

Here is the "CommutatorExpand" of a power using nested commutators:

```
CommutatorExpand[(m + n)^3, NestedCommutators → True]
```

```
m^3 + n^3 + ⟦m, ⟦m, n⟧_-⟧_- + 2 ⟦n, ⟦m, n⟧_-⟧_- + 3 m^2 · n + 3 m · n^2 - 3 m · ⟦m, n⟧_- - 3 n · ⟦m, n⟧_-
```

Now we assign "m" as the result of the commutator of p and q:

```
⟦m, n⟧_- = p
```

```
p
```

This time the commutator expand of the power uses the fact that the commutator is equal to m. However the commutators of m with p and with q were not defined, therefore they appear in the answer. Compare this answer with the answer that was obtained before assigning a value to the commutator and using the NestedCommutators→True option:

```
CommutatorExpand[(m + n)^3]
```

```
m^3 + n^3 + ⟦m, p⟧_- + 2 ⟦n, p⟧_- + 3 m^2 · n + 3 m · n^2 - 3 m · p - 3 n · p
```

by José Luis Gómez-Muñoz
http://homepage.cem.itesm.mx/lgomez/quantum/
jose.luis.gomez@itesm.mx
with contributions by Gianfranco Arroyo-Orrico and Sergio E. Martínez-Casas