QCAlgebra Package

Fausto Saporito

Contents

Part 1

Description

CHAPTER 1

Package description

1. Generalities

These verbs (written in J) implements some quantum computing algebra useful to perform some quantum circuit calculation.
This new version (there's a package called qcalgebra2 somewhere hidden in my laptop) is trying to avoid the explicit calculation using matrices and components, because (as in QFT) the number of that components is exponential, so if I have a 20-qubits system, it should be quite impossible to calculate a QFT on all those qubits using matrices.

In this schema, our basic elements (kets) are:

- K0 is $|0\rangle$
- K1 is $|1\rangle$

and, for example, K00 ($|00\rangle$) is K0 TP K0. The verb TP is used to create bigger qubits (Tensor Product).
Internally, the qubits are expressed via boxed set (the first element is the complex coefficient). So, for example,

- 1;0 0 means $|00\rangle$
- 1;0 1 0 means $|010\rangle$
- 2;0 0 means $2|00\rangle$

2. Verbs

Below there are the verbs used either for the operator/gates application either for the other generic operation on qubits.

2.1. Operator verbs. The GATES, in the code, are written in CAPITAL LETTERS (they performs automatically a simplification and cleaning) and the arguments follow the J-standard:

(1) HD : Hadamard Gate acting on qubit x of quantum state y

0 HD K00

it means Hadamard gate acting on the first qubit (qubit 0) of $|00\rangle$, so $\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)|0\rangle = \frac{1}{\sqrt{2}}(|00\rangle+|10\rangle)$. This gate is useful to generate a superposition of states, starting from a simple one ($|0\rangle$ or $|1\rangle$).
(2) XG, YG, ZG: Pauli Gates. They use the same standard as described for HD gate
(3) CNOT : Controlled NOT (XG) gate. Use: (c,t) CNOT qreg, where c is control qubit and t is target qubit
(4) CY, CZ : Controlled Y and Controlled Z. As above.
(5) RPHI : Phase shift gate (rotation) R_ϕ

(6) RK : Phase shift gate modified for QFT (internal use) (rotation) R_k
(7) CRK : Controlled version of R_k
(8) SW: Swap GATE
(9) DTHETA : Deutch (universal) gate $D(\theta)$.

2.2. Other verbs. There are some other interesting verbs, used mainly inside the other verbs to perform very specific activities. They are:

- simpl : it performs simplification of qubit expression (summation and cleaning of small complex amplitudes)
- PROB : it extracts the probability related to measurement of bits in the qreg, i.e. (0;1 1) PROB 0 HD K00 - it returns the probability to have as result of measurement the first two qubit to 1, so all the states: $|11?\rangle$ after applying hadamard gate to the first qubit of $|00\rangle$. The first element in the boxed list is the offset (0-based) for the qubit pattern.
- QFT : performs the QFT (quantum fourier transform) on qubits.

  3 QFT K000

  computes the QFT on the three first qubits of K000, in terms of qubits we have:

  $$\begin{gathered}0.35355339 + 0.00000000i|000\rangle+\\ 0.35355339 + 0.00000000i|001\rangle+\\ 0.35355339 + 0.00000000i|010\rangle+\\ 0.35355339 + 0.00000000i|011\rangle+\\ 0.35355339 + 0.00000000i|100\rangle+\\ 0.35355339 + 0.00000000i|101\rangle+\\ 0.35355339 + 0.00000000i|110\rangle+\\ 0.35355339 + 0.00000000i|111\rangle\end{gathered}$$

  the probability associated to each state is 12.5%, if we use PROB verb for the first state $|000\rangle$

  (0;0 0 0) PROB 3 QFT K000

  it returns 0.125 i.e. 12.5%

Last but not least, there's a starting implementation of QEC protocols. At the moment there's an encoding developed by Shor, using 9 qubits.

Part 2

Source code

```
NB. Quantum Computing Algebra Package
NB. F. Saporito, 2015
K0=:1;1$0
K1=:1;1$1

sq=:%%:2
pi2=:(%2)*1p1
pi4=:(%4)*1p1
pi8=:(%8)*1p1

tp=:dyad define
NB. tensor product between
cx=.>0{x
cy=.>0{y
stx=.>1{x
sty=.>1{y
cf=.cx*cy
stf=.stx,sty
cf;stf
)

sum=:dyad define
NB. sum of two 1-qubit
NB. i.e. |0> + |1>
cx=.>0{x
cy=.>0{y
stx=.>1{x
sty=.>1{y
x,:y
)

SUM=:sum"1

diff=:dyad define
x sum _1 mul y
)

DIFF=:diff"1

mul=:dyad define
NB. multiplication with a scalar
coef=.>0{y
coef=.x * coef
stat=.>1{y
coef;stat
)

MUL=:mul"1
```

```
bmul=:dyad define
NB. multiplication between bra and ket
NB. with simplification rules embedded
NB. x is BRA ---- y is KET
stx=.>1{ x
sty=.>1{ y
coe=.(>0{ x) * >0{ y
lstx=.#stx
lsty=.#sty
if. lstx = 1 *. lsty = 1 do.
 if. stx=sty do.
  coe;stx
 else.
  0;stx
 end.
else.
 if. lsty > lstx do.
  status=.*/(lstx {. sty) = stx
  if. status=0 do. NB. orthogonal states so nullify multiplication
   0;sty
  else.
   coe;(lstx }. sty)
  end.
 else.
  status=.*/(lsty {. stx) = sty
  if. status=0 do. NB. orthogonal states so nullify multiplication
   0;sty
  else.
   coe;(lsty }. stx)
  end.
 end.
end.
)

BMUL=:([: simpl bmul"1)

checksmall=:monad define
NB. Check for small amplitudes
NB. either in the real or imag part
re=.9&o.
im=.11&o.
if. (|re y) < 1e_10 do.
rey=.0
else.
rey=.re y
end.
if. (|im y) < 1e_10 do.
```

```
imy=.0
else.
imy=.im y
end.
rey + _11 o. imy
)

clean=:monad define
NB. clean qubits with 0 amplitude from a sum
amp0=.<0
f0=.-.amp0 E. 0{"1 y
tqb=.f0#y
NB. now remove amplitude too much little
coef=.>0{"1 tqb
coef=.checksmall"0 coef
coeffilt=.(|coef) > 1e_10
tqb=.(<"0 coef),.1{"1 tqb NB. using the new coeffs
coeffilt#tqb
)

binbox=:monad define
NB. convert to decimal, binary values in a boxed list
len=.#,y
stpos=.(2|i.len)#i.len
for_j. stpos do.
  y=.(<#.>j{y) j }y
end.
y
)

boxbin=:dyad define
NB. convert to binary, a boxed list of states
len=.#>y
l2=.len%2
twostr=.x$2
stpos=.(2|i.len)#i.len
for_j. stpos do.
  yval=.twostr#:>j{y
  y=.(<yval) j }y
end.
(l2,2)$y
)


simpl=:monad define
NB. simplify multi qubit summation
stlen=.#>1{,y
y=.binbox ,y
```

```
len=.#,y
l2=.len%2
stpos=.(2|i.len)#i.len
cpos=.1-stpos          NB. positions for the coeffs
y=.(l2,2)$y
states=.stpos { >,y    NB. extract the states
coeffs=.cpos { >,y
stateq=.(-.~:states)#states  NB. duplicated states
nstateq=.#stateq             NB. how many ?
stateqpos=.stateq I.@:E."0 1 [ states   NB. positions of duplicated states
if. nstateq=0 do.
  clean stlen boxbin ,y
else.
  tt=.(+/(0{stateqpos){coeffs);0{stateq
  for_j. 1+i.nstateq-1 do.
    tt=.tt,((+/(j{stateqpos){coeffs);j{stateq)
  end.
  stlen boxbin ,clean (nstateq,2)$tt
end.
)

TP=:tp"1 1/

K00=:K0 TP K0
K01=:K0 TP K1

CMUL=:dyad define
NB. multiplication qubit by a constant
cf=.x*>0{y
st=.>1{y
cf;st
)

hd=:monad define
NB. Hadamard gate acting on 1 qubit
st=.>1{y
cf=.>0{y
h0=.sq;0
h1=.sq;1
hm1=.(-sq);1
if. st=0 do.
cf CMUL"0 1 h0 sum h1
else.
cf CMUL"0 1 h0 sum hm1
end.
)

Hd=:dyad define
```

```
NB. hadamard gate for multi-qubit
NB. x is the bit where is acting
nqb=.#>1{y NB. how many qubits
tst0=.x{>1{y NB. select the state of target qubit
tqb0=.hd 1;tst0 NB. apply hadamard on it
NB. now evaluate the position inside the qubit lists
NB. we have to handle the coefficients with ": verb (default format)
ex=.''
for_j. i.nqb do.
 if. (j=x) *. (j=nqb-1) do. NB. target qubit is last
 ex=.ex,'tqb0 '
 elseif. (j=x) *. (j~:nqb-1) do.
  ex=.ex,'tqb0 TP '
 elseif. (j~:x) *. (j~:nqb-1) do.
  if. (j{>1{y)=0 do.
  ex=.ex,'K0 TP '
  else.
  ex=.ex,'K1 TP '
  end.
 elseif. (j~:x) *. (j=nqb-1) do.
  if. (j{>1{y)=0 do.
  ex=.ex,'K0'
  else.
  ex=.ex,'K1'
  end.
 end.
end.
". (":>0{y),' CMUL"0 1 ',ex
)

HD=:dyad define
tt=.,simpl x Hd"1 y
ttlen=.(#tt)%2
(ttlen,2)$tt
)

xg=:monad define
NB. Pauli-X-gate 1-qubit
cf=.>0{y
cf;-.>1{y
)

yg=:monad define
NB. Pauli-Y-gate 1-qubit
cf=.(_11 o. 1)*>0{y
cf;-.>1{y
)
```

```
zg=:monad define
NB. Pauli-Z-gate 1-qubit
st=.>1{y
if. st=1 do.
cf=.->0{y
cf;st
else.
y
end.
)

rphi=:dyad define
NB. Phase shift gate with angle phi (x)
NB. working on 1-qubit
phi=._12 o. x
st=.>1{y
if. st=1 do.
cf=.phi*>0{y
cf;st
else.
y
end.
)

rk=:dyad define
NB. a variant of RPHI using 2^x
NB. as argument
phi=.2p1 % 2^x
phi rphi y
)

Rk=:dyad define
NB. Rk gate for multiqubit
tqb=.0{x
angle=.1{x
st=.tqb{>1{y
stlen=.#>1{y
cf=.>0{y
qbf=.angle rk cf;st
cf=.>0{qbf
if. tqb=stlen-1 do.
stf=.(i.stlen-1){>1{y
stf=.stf,>1{qbf
elseif. tqb=0 do.
stf=.,>1{qbf
stf=.stf,((stlen-tqb+2)+i.stlen-tqb+1){>1{y
elseif. tqb~:0 do.
stf=.(i.(stlen-1)-tqb){>1{y
```

```
stf=.stf,>1{qbf
stf=.stf,((stlen-tqb+1)+i.stlen-tqb+1){>1{y
end.
cf;stf
)

RK=:Rk"1

CRk=:dyad define
NB. Generic Controlled-Rk gate for 1-qubit
NB. x = list of
NB.     - controller qubit
NB.     - target qubit
NB.     - k rotation parameter
NB. y = qubit register
cst=.(0{x){>1{y
if. cst=1 do.
((1{x),2{x) RK y
else.
y
end.
)

CRK=:CRk"1

Xg=:dyad define
NB. Pauli-X gate for multiqubit
st=.x{>1{ y
stlen=.#>1{y
cf=.>0{ y
qbf=.xg cf;st
cf=.>0{qbf
if. x=stlen-1 do.
stf=.(i.stlen-1){>1{ y
stf=.stf,>1{qbf
elseif. x=0 do.
stf=.>1{qbf
stf=.stf,((stlen-x+2)+i.stlen-x+1){>1{ y
elseif. x~:0 do.
stf=.(i.(stlen-1)-x){>1{ y
stf=.stf,>1{qbf
stf=.stf,((x+1)+i.(stlen-1)-x){>1{ y
end.
cf;stf
)

XG=:([: simpl Xg"1)
```

```
Yg=:dyad define
NB. Pauli-Y gate for multiqubit
st=.x{>1{y
stlen=.#>1{y
cf=.>0{y
qbf=.yg cf;st
cf=.>0{qbf
if. x=stlen-1 do.
stf=.(i.stlen-1){>1{y
stf=.stf,>1{qbf
elseif. x=0 do.
stf=.>1{qbf
stf=.stf,((stlen-x+2)+i.stlen-x+1){>1{y
elseif. x~:0 do.
stf=.(i.(stlen-1)-x){>1{y
stf=.stf,>1{qbf
stf=.stf,((x+1)+i.(stlen-1)-x){>1{y
end.
cf;stf
)

YG=:([: simpl Yg"1)

Zg=:dyad define
NB. Pauli-Z gate for multiqubit
st=.x{>1{y
stlen=.#>1{y
cf=.>0{y
qbf=.zg cf;st
cf=.>0{qbf
if. x=stlen-1 do.
stf=.(i.stlen-1){>1{y
stf=.stf,>1{qbf
elseif. x=0 do.
stf=.>1{qbf
stf=.stf,((stlen-x+2)+i.stlen-x+1){>1{y
elseif. x~:0 do.
stf=.(i.(stlen-1)-x){>1{y
stf=.stf,>1{qbf
stf=.stf,((x+1)+i.(stlen-1)-x){>1{y
end.
cf;stf
)

ZG=:([: simpl Zg"1)

RPhi=:dyad define
NB. Phase gate for multiqubit
```

```
NB. (targqubit,phase) RPHI qreg
xx=.0{x
phi=.1{x
st=.xx{>1{y
stlen=.#>1{y
cf=.>0{y
qbf=.phi rphi cf;st
cf=.>0{qbf
if. xx=stlen-1 do.
stf=.(i.stlen-1){>1{y
stf=.stf,>1{qbf
elseif. xx=0 do.
stf=.>1{qbf
stf=.stf,((stlen-xx+2)+i.stlen-xx+1){>1{y
elseif. x~:0 do.
stf=.(i.(stlen-1)-xx){>1{y
stf=.stf,>1{qbf
stf=.stf,((xx+1)+i.(stlen-1)-xx){>1{y
end.
cf;stf
)

RPHI=:([: simpl RPhi"1)

cnot=:monad define
NB. CNOT gate for 2-qubit
NB. 1st qubit is the controller
NB. 2nd qubit is the target
cst=.0{>1{y NB. controller qubit state
if. cst=1 do.
1 XG y
else.
y
end.
)

Cnot=:dyad define
NB. Generic CNOT gate for multi qubit
NB. x = list of controller qubit and target qubit
NB. y = qubit register
cst=.0{x{>1{y
if. cst=1 do.
(1{x) XG y
else.
y
end.
)
```

```
CNOT=:([: simpl Cnot"1)

CYg=:dyad define
NB. Generic Controlled-Y gate for multi qubit
NB. x = list of controller qubit and target qubit
NB. y = qubit register
cst=.0{x{>1{y
if. cst=1 do.
(1{x) YG y
else.
y
end.
)

CYG=:([: simpl CYg"1)

CZg=:dyad define
NB. Generic Controlled-Z gate for multi qubit
NB. x = list of controller qubit and target qubit
NB. y = qubit register
cst=.0{x{>1{y
if. cst=1 do.
(1{x) ZG y
else.
y
end.
)

CZG=:([: simpl CZg"1)

Sw=:dyad define
NB. SWAP gate for 2qubit
sou=.0{x
des=.1{x
cf=.>0{y
st=.>1{y
vals=.sou{st
st=.(des{st) sou } st
st=.vals des } st
cf;st
)

SW=:([: simpl Sw"1)

DTheta=:dyad define
NB. Deutsch gate 3-qubit
st=.>1{y
st1=.0{st
```

```
st2=.1{st
st3=.2{st
cf=.>0{y
cf1=.cf*_11 o. 2 o. x
cf2=.cf*1 o. x
if. (st1=1) *. (st2=1) do.
(cf1;(st1,st2,st3)) sum (cf2;(st1,st2,1-st3))
else.
y
end.
)

DTHETA=:([: simpl DTheta"1)

prob=:dyad define
NB. return the probability to have the specified state
NB. after a measurement
st=.,>1{"1 y
startpos=.>0{x
targpos=.(>1{x) I.@:E. st
tarfilt=.startpos = targpos
if. (+/tarfilt)=1 do.
NB. cpos=.tarfilt#i.2
(|,>0{"1 y)^2
else.
0
end.
)

PROB=:dyad define
tt=.,x prob"1 y
+/tt
)

NRM=:monad define
NB. returns norm of the vector
norm=.,>0{"1 y
norm=.+/norm^2
%:norm
)

NRMZ=:monad define
NB. normalize the vector to unitary
coef=.,>0{"1 y
nn=.NRM y
coef=.coef%nn
st=.>1{"1 y
coef;"0 1 st
```

```
)

QFT=:dyad define
NB. Computes the QFT recursively.
NB. 3 QFT K000 computes the QFT on the three first qubits of K000
tt=.y
for_j. i.x-1 do.
 tt=.j HD tt
 for_k. i.j+1 do.
  arg=.2^k+1
  tt=.(j+1,k,arg) CRK tt
 end.
end.
tt=.(x-1) HD tt
simpl tt
)

CRQB=:dyad define
NB. script to create large qubits in equal states
NB. i.e. |0000...0> or |11111...1>
NB. so, 0 CRQB 9 it means : |000000000>
if. x=0 do.
  K0 TP^:(y-1) K0
else.
  K1 TP^:(y-1) K1
end.
)

K000=:0 CRQB 3
K111=:1 CRQB 3

NB. internal usage
B000=:sq MUL K000 SUM K111
B111=:sq MUL K000 DIFF K111

NB. redundant coding (steane - QEC)
L0=:B000 TP B000 TP B000 NB. Logical |0>
L1=:B111 TP B111 TP B111 NB. Logical |1>
```