JuliaQuantum : Julia libraries for Quantum Science and Technology

Quantum Physics library support using Julia

Amit Jamadagni

Birla Institute of Technology and Science, Pilani, India.

bitsjamadagni@gmail.com

Abstract

JuliaQuantum is an umbrella organization aiming to support libraries of quantum science using Julia language. Currently, the following libraries are being supported by the organization

- QuBase.jl : A base library which provides the basic constructs used in Quantum Mechanics.
- QuDirac.jl : Library for using Dirac notation to perform quantum mechanics computations.
- QuDynamics.jl : Library for solving dynamical equations in quantum mechanics.

The poster aims at presenting the current capabilities of these packages as well present the ongoing efforts to integrate packages which focus on Quantum Information, Symbolic Quantum Mechanics among others.

JuliaQuantum and libraries

QuBase.jl - Foundational library for quantum mechanics in Julia

Current Functionality

The current version is based on v0.4 of Julia. The base library provides a container **QuArray** constructs for state vectors, operators which are Julia arrays. QuArray is the base layer and every other construct is supported on the top of this layer. Most of the features like norm, scale, dot among many others are imported from the base and have been defined for QuArray. This package also provides support to some special operators like creation, annihilation, squeezing operators, the pauli matrices among others.

Various constructions using QuBase

```
julia> using QuBase

julia> σx
2x2 QuMatrix in QuBase.FiniteBasis{QuBase.Orthonormal}:
...coefficients: Array{Float64,2}
[0.0 1.0
 1.0 0.0]

julia> statevec(1, FiniteBasis(2))
2-element QuVector in QuBase.FiniteBasis{QuBase.Orthonormal}:
...coefficients: Array{Float64,1}
[1.0,0.0]

julia> normalize!(QuArray([0.5+0.1im, 0.2+0.2im]))
2-element QuVector in QuBase.FiniteBasis{QuBase.Orthonormal}:
...coefficients: Array{Complex{Float64},1}
Complex{Float64}[0.8574929257125441 + 0.17149858514250882im,0.34299717028501764 +
  0.34299717028501764im]
```

```
julia> coeffs(σx)
2x2 Array{Float64,2}:
 0.0  1.0
 1.0  0.0

 julia> lowerop(2), raiseop(2)
(2x2 QuMatrix in QuBase.FiniteBasis{QuBase.Orthonormal}:
...coefficients: SparseMatrixCSC{Float64,Int64}

        [1, 2]  =  1.0,2x2 QuMatrix in QuBase.FiniteBasis{QuBase.Orthonormal}:
...coefficients: SparseMatrixCSC{Float64,Int64}

        [2, 1]  =  1.0)

julia> coherentstatevec(2,2)
2-element QuVector in QuBase.FiniteBasis{QuBase.Orthonormal}:
...coefficients: Array{Float64,1}
[-0.41614683654714235,0.9092974268256816]
```

Features to be integrated

In the recent past, there has been a lot of discussion on improving the **QuArray** construct to support a more general construct that would accommodate the information of basis. This would not only be helpful to the JuliaQuantum community but may also serve as a common base package to other packages which use Linear Algebra. Efforts to integrate in-place operations, parallelization at various levels of abstraction, sparse support for QuArray are currently being discussed as these would help increase the time and memory efficiency of the supported operations.

QuDynamics.jl - Library for solving dynamical equations in quantum mechanics

QuDynamics is a Julia package which provides a framework for solving dynamical equations arising in Quantum Mechanics. The current release of the master includes the support for solving Schrondinger Equation, Liouville von Neumann Equation, Lindblad Master Equation using various solvers which have been integrated from various other Julia packages like ODE.jl, ExpmV.jl, Expokit.jl. The package also provides support for Quantum Monte-Carlo Wave Function method.

Current Functionality

Properties of the system with QuDynamics equivalent

To evolve a system, the following properties are a minimal requirement :

System Construction

- Hamiltonian (or) Liouvillian, Lindblad operator with collapse operators
- Initial state vector / density matrix
- Time step array / range
- Solver used to evolve the system, the following solvers are currently supported :
  - Euler, Crank-Nicholson, Krylov
  - ode45, ode23s, ode78 from ODE.jl
  - expmv from ExpmV.jl, Expokit.jl
  - Monte Carlo Wave Function Method (MCWF)

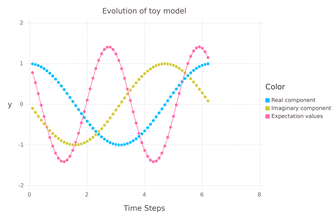
QuDynamics Equivalent Construction

- QuSchrodingerEq, QuLiouvillianEq, QuLindbladEq
- QuArray type
- Float range
- Solver used to evolve the system, the following solvers are currently supported :
  - QuEuler, QuCrankNicholson, QuKrylov
  - QuODE45, QuODE23s, QuODE78 using ODE.jl
  - QuExpmV from ExpmV.jl, QuExpokit using Expokit.jl
  - QuMCWF, Monte Carlo Wave Function Method (MCWF)

Evolving various systems using QuDynamics

```
julia> using QuBase

julia> using QuDynamics

julia> using Gadfly

julia> H = σx;

julia> ψ = statevec(1, FiniteBasis(2));

julia> t_n = 0.:0.01:2*π;

julia> prop_expmV = QuPropagator(H, ψ, t_n, QuExpmV())
Summarizing the system :
Equation type : QuDynamics.QuSchrodingerEq{QuBase.QuArray{QuBase.FiniteBasis{QuBase.
  Orthonormal},Float64,2,Array{Float64,2}}}
Size of the Hamiltonian of the system : (2,2)
Size of the Initial state : (2,)
Time steps used : 0.0:0.01:6.28
Solver used : QuDynamics.QuExpmV(Dict{Symbol,Any}())

julia> evolved_states_1 = Array(Real, length(t)-1);

julia> evolved_states_2 = Array(Real, length(t)-1);

julia> expectation = Array(Real, length(t)-1)
```

```
julia> for (t, psi) in prop_expmV
          evolved_states_1[i] = real(psi[1])
          evolved_states_2[i] = imag(psi[2])
          expectation[i] = real(expectationvalue(psi, sigmax + sigmay + sigmaz))
          i = i + 1
       end

# Plot using Gadfly
```



Features to be integrated

Currenty the package supports time independent equations, but in the recent past efforts have been made to provide support for time dependent equations. Parallelization of Monte Carlo Wave Function method has been achieved but the feature is yet to be integrated into the package. In addition, once parallelization is achieved in QuBase, efforts in the direction of making QuDynamics compatible is one of the tasks for the future. Some of the other open tasks involve improving the memory and timing efficieny which can be achieved by equipping the library with in place operators and other features. Equipping the package with additional solvers will always remain the central aim of the package and efforts to communicate this to the developers has been made.

QuDirac.jl - Library for performing quantum mechanics using Dirac notation in Julia

Current Functionality

QuDirac.jl is a Julia library for using Dirac notation to perform quantum mechanics computations. Some of the features are as follows :

```
julia> using QuDirac

julia> ket(0)
Ket{KroneckerDelta,1,Int64} with 1 state(s):
  1  |0⟩

julia> d"""
       ψ = 1/√2 * (|0,0⟩ + |1,1⟩)
       a = purity(ptrace(ψ * ψ', 2))
       ϕ = normalize!(1/5 * ⟨0| + 4/5 * ⟨1|)
       result = normalize!(a * act_on(ϕ, ψ, 2))
       """

julia> result
Ket{KroneckerDelta,1,Float64} with 2 state(s):
  0.2425356250363329 7 | 0 ⟩
  0.9701425001453319 | 1 ⟩

julia> d"normalize!(sum(i -> (i^2)|i⟩, 0:3) * sum(i -> (i/2)|i⟩, -3:3))"
```

```
Ket{KroneckerDelta,2,Float64} with 18 state(s):
  -0.5154323951168185 | 3,-3 ⟩
  0.3436215967445456 | 3,2 ⟩
  -0.019090088708030313 | 1,-1 ⟩ ...

julia> ψ = d"normalize!(|0,1⟩ + 2.0|1,0⟩)"
Ket{KroneckerDelta,2,Float64} with 2 state(s):
  0.4472135954999579 | 0,1 ⟩
  0.8944271909999159 | 1,0 ⟩

julia> ϕ = d"1/√2 * (⟨0| + ⟨1|)"
Bra{KroneckerDelta,1,Float64} with 2 state(s):
  0.7071067811865475 ⟨ 0 |
  0.7071067811865475 ⟨ 1 |

julia> act_on(ϕ, ψ, 2)
Ket{KroneckerDelta,1,Float64} with 2 state(s):
  0.3162277660168379 | 0 ⟩
  0.6324555320336758 | 1 ⟩
```

Features to be integrated

The current version is supported on version 0.3 of Julia. The next release is expected to use the features of Julia 0.5.