Using a user-defined eigendecomposition function

Huo Chen

November 11, 2020

0.1 Initialize Hamiltonian with a custom eigendecomposition function

When defining a Hamiltonian object, a keyword argument `EIGS` can be supplied to construct a user defined eigendecomposition routine. All the eigendecomposition calls inside the solver will call this function instead of the default one.

For example:

```
using QuantumAnnealingTools

# Define a function to construct an eigendecomposition routine for the
# Hamiltonian. The routine should have the signature: (H, t, lvl) -> (w, v).
# The argument of this function is the cache used by the Hamiltonian object.
function build_user_eigen(u_cache)
    EIGS = function(H, t, lvl)
        println("I am the user defined eigendecomposition routine.")
        w, v = eigen(Hermitian(H(t)))
        w[1:lvl], v[:, 1:lvl]
    end
end

H = DenseHamiltonian([(s)->1-s, (s)->s], [σx, σz], EIGS=build_user_eigen)

eigen_decomp(H, 0.5, lvl=2)
```

```
I am the user defined eigendecomposition routine.
([-0.7071067811865476, 0.7071067811865476], Complex{Float64}[-0.38268343236
508984 + 0.0im 0.9238795325112866 - 0.0im; 0.9238795325112868 - 0.0im 0.382
6834323650897 - 0.0im])
```

0.1.1 Constant Hamiltonian

There are two applications for this functionality. First, if the Hamiltonian is constant, one can precalculates the eigensystem of that Hamiltonian and build a function that returns those precalculated values. This is particular helpful for adiabatic master equation solver.

```
function build_user_eigen(u_cache)
    # note that to keep the unit consistent, the unit of any value inside the routine
should be 1/h
    w, v = eigen(Hermitian(2*π*(σx+σz)))
    EIGS = function(H, t, lvl)
```

```
        w[1:lvl], v[:, 1:lvl]
    end
end

H = DenseHamiltonian([(s)->1.0], [σx+σz], EIGS=build_user_eigen)

print(eigen_decomp(H, 0.5, lvl=2))
print(eigen_decomp(H, 0.0, lvl=2))
```

```
([-1.4142135623730951, 1.4142135623730951], Complex{Float64}[0.382683432365
0897 + 0.0im -0.9238795325112867 + 0.0im; -0.9238795325112867 + 0.0im -0.38
26834323650897 + 0.0im])([-1.4142135623730951, 1.4142135623730951], Complex
{Float64}[0.3826834323650897 + 0.0im -0.9238795325112867 + 0.0im; -0.923879
5325112867 + 0.0im -0.3826834323650897 + 0.0im])
```

0.1.2 Sparse Hamiltonian

Another application is to supply speical eigendecomposition algorithms for sparse matrices in order to take advantage of the sparsity.

For example, the default eigendecomposition algorithm for sparse Hamiltonian is to first convert it into dense matrices and then perform the decomposition.

```
Hd = standard_driver(4, sp=true);
Hp = two_local_term(rand(3), [[1,2],[2,3],[3,4]], 4, sp=true)
H = SparseHamiltonian([(s)->1-s, (s)->s], [Hd, Hp], unit=:ħ)

# the default eigen_decomposition using dense matrices algorithm
w, v = eigen_decomp(H, 0.1, lvl=4)
```

```
([-0.5735316712156144, -0.30476343162186026, -0.28906825965875976, -0.28442
55393119809], Complex{Float64}[0.23775345210658458 + 0.0im 0.04834242388403
459 + 0.0im 0.16140856607158252 + 0.0im 0.12766070878640218 + 0.0im; -0.240
50532890906157 + 0.0im -0.09657525354907054 + 0.0im 0.19052560511367683 + 0
.0im 0.19000782996828985 + 0.0im; ...@*( ; -0.24050532890906146 + 0.0im
0.09657525354907057 + 0.0im -0.19052560511367703 + 0.0im -0.1900078299682898 + 0.0im;
0.2377534521065846 + 0.0im -0.04834242388403451 + 0.0im -0.16140856607158244 + 0.0im
-0.12766070878640207 + 0.0im])
```

If the size of the Hamiltonian becomes very large, we can use sparse algorithms provided by Arpack instead. Let's first install `Arpack.jl` by running:

```
using Pkg
Pkg.add("Arpack")
using Arpack
```

Next, we can use `Arpack` function to replace the default eigendecomposition routine:

```
function build_user_eigen(u_cache)
    function (H, t, lvl)
        hmat = H(t)
        println("Using sparse matrix algorithm")
        # define all the Arpack routine parameters here
        eigs(hmat, nev = lvl, which=:SR, tol=0.0, maxiter=300)
    end
end

Hd = standard_driver(4, sp=true);
```

```
Hp = two_local_term(rand(3), [[1,2],[2,3],[3,4]], 4, sp=true)
H = SparseHamiltonian([(s)->1-s, (s)->s], [Hd, Hp], unit=:ħ, EIGS =build_user_eigen)

eigen_decomp(H, 0.1, lvl=4)

Using sparse matrix algorithm
([-0.5735516727811818, -0.3046485707262054, -0.2915537146247742, -0.2818230
8632865133], Complex{Float64}[-0.2311252347467679 - 0.05054643771420568im 0
.013475305653670437 - 0.009600287077920877im -0.0049270347186456485 + 0.139
3776605541147im 0.006812658644544562 - 0.05546172624860093im; 0.23742845187
818473 + 0.05192493354345323im -0.11503836315932262 + 0.08195742194557974im
 -0.006326159230190435 + 0.1789565781748491im 0.031877083677014736 - 0.2595
1074033539673im; ...@*( ; 0.23742845187818473 + 0.0519249335434535im 0.1150383631593228 -
0.08195742194557963im 0.006326159230190432 - 0.1789565781748491im-0.03187708367701511 +
0.2595107403353964im; -0.23112523474676808 - 0.05054643771420557im -0.01347530565367078 +
0.009600287077920737im 0.0049270347186458445 - 0.13937766055411466im
-0.006812658644544527 + 0.05546172624860083im])
```