Using a user-defined eigendecomposition function

Huo Chen

November 25, 2020

0.1 Initialize a Hamiltonian with a custom eigendecomposition function

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

For example:

```julia
using OpenQuantumTools

# 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 precalculate that Hamiltonian's eigensystem and build a function that returns those precalculated values. This is particularly helpful for the adiabatic master equation solver.

```julia
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)
```

```julia
        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 special eigendecomposition algorithms for sparse matrices to take advantage of the sparsity.

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

```julia
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 the dense matrices algorithm
w, v = eigen_decomp(H, 0.1, lvl=4)
```

```
([-0.573158213493025, -0.29722068915454863, -0.28751113923440075, -0.285721
66599114707], Complex{Float64}[0.24306368695026373 + 0.0im 0.01793984049002
0342 + 0.0im -0.23331220659368423 + 0.0im -0.028894435063400392 + 0.0im; -0
.24510136534686483 + 0.0im -0.09243319864704294 + 0.0im -0.1087343136421368
2 + 0.0im -0.3042258322136764 + 0.0im; ...@*( ; -0.24510136534686444 + 0.0im
0.09243319864704286 + 0.0im 0.10873431364213687 + 0.0im 0.3042258322136764 + 0.0im;
0.24306368695026367 + 0.0im -0.01793984049002037 + 0.0im 0.23331220659368412 + 0.0im
0.02889443506340017 + 0.0im])
```

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

```julia
using Arpack
```

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

```julia
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
([-0.5733051799040485, -0.3003771871130221, -0.28945094836234425, -0.284119
50958797505], Complex{Float64}[-0.23564220236670824 - 0.05374317789247806im
 -0.025486113209488615 + 0.003223181669690833im 0.0019530356849578514 - 0.0
3308116177349319im 0.1426995029346503 - 0.26184025362558205im; 0.2469994389
710251 + 0.056333435414527976im -0.31228021902101716 + 0.039493502578560685
im 0.002454120701199303 - 0.041568704839317884im -0.185561252076381 + 0.340
4875581732886im; ...@*( ; 0.24699943897102514 + 0.056333435414528295im 0.3122802190210173
- 0.039493502578560255im -0.0024541207011992213 + 0.041568704839317676im
0.18556125207638086 - 0.3404875581732885im; -0.23564220236670846 -0.05374317789247819im
0.025486113209488594 - 0.003223181669691298im -0.0019530356849581903 +
0.033081161773493545im -0.1426995029346503 + 0.2618402536255819im])
```