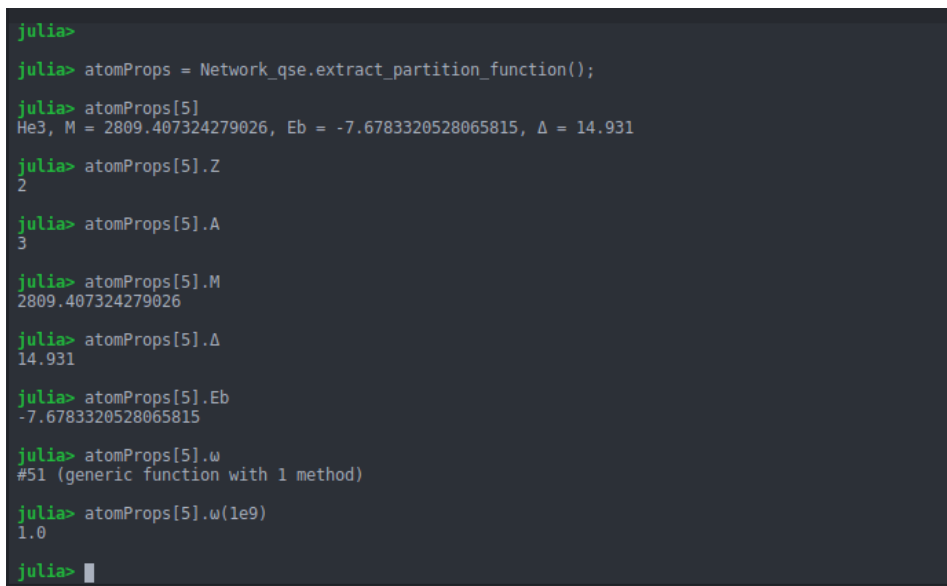


First steps:

1. cd into project directory
2. `(Network_qse) pkg> activate .`¹
3. run `include("../src/Network_qse.jl")`
4. if packages missing `pkg> add ExamplePackage`
5. now all functions can be called in Julia prompt: `Network_qse.functionName()`
6. run tests in pkg: `(Network_qse) pkg> test`

Use Julia

Example screenshot on how to call `AtomProperties` (this is a defined Datatype, see `DataTypes.jl`) in the Julia prompt



```
julia>
julia> atomProps = Network_qse.extract_partition_function();
julia> atomProps[5]
He3, M = 2809.407324279026, Eb = -7.6783320528065815, Δ = 14.931
julia> atomProps[5].Z
2
julia> atomProps[5].A
3
julia> atomProps[5].M
2809.407324279026
julia> atomProps[5].Δ
14.931
julia> atomProps[5].Eb
-7.6783320528065815
julia> atomProps[5].ω
#51 (generic function with 1 method)
julia> atomProps[5].ω(1e9)
1.0
julia> 
```

Figure 0.1: Example for calling `extract_partition_function()` in `IO.jl`. This array stores all input that is needed (except constants).

Calling `atomProps.ω` returns a T-dependent partition (type) function, with Temperature (Kelvin) as argument.

If looking for the atomic properties of a specific element, call

```
filter(i -> (atomProps[i].name == "Fe56"), 1:size(atomProps,1))
```

This returns the index or array element of a specific element, here Fe56, stored in `atomProps`, see screenshot.

Calculations for E_B , Δ , M defined in `DataTypes.jl`

¹enter package manager (pkg) with closing square bracket. Exit with Backspace

```

julia>
julia>
julia> hf(x, ap) = ap[filter(i -> (ap[i].name == x), 1:size(ap,1))]
hf (generic function with 1 method)

julia> props = Network_qse.extract_partition_function();

julia> hf("O16", props)
1-element Array{Main.Network_qse.AtomicProperties,1}:
 016, M = 14899.13672948814, Eb = -127.39798128850634, Δ = -4.737

julia> hf_index(x, ap) = filter(i -> (ap[i].name == x), 1:size(ap,1))
hf_index (generic function with 1 method)

julia> hf_index("O16", ap)
1-element Array{Int64,1}:
 7

julia> props[7]
016, M = 14899.13672948814, Eb = -127.39798128850634, Δ = -4.737

julia>

```

Figure 0.2: Inline Functions can be defined by simply `f(x) = ...`.

Boltzmann.jl

`prefactor(pf)` returns a temperature and density dependent function, all in CGS units (cococubed.asu.edu/code_pages/nse.shtml)

$$\frac{A}{N_A \rho} \cdot \omega(T) \left(\frac{2\pi k_B T M_i}{h^2} \right)^{1.5}$$

So prefactor of He3 at $T = 10^9$, $\rho = 10^7$ (cgs units) would be called with:

`Network_qse.prefactor(ap[5])(1e9, 1e7)`

`nse_condition!(res, μ, T, ρ, y, ap; precision=4000)` calculates

$$\log \left(\sum_i X_i \right) = 0,$$

$$\log \left(\frac{\sum_i \frac{Z_i}{A_i} X_i}{Y_e} \right) = 0$$

The function changes its argument `res`, similar to Fortran subroutines ².

NLsolve

NLsolve³ takes the (set to zero) function with 2 arguments. The return value of function (one wants to set zero) and the solution for chemical potential μ_p, μ_n . If `f` has more

²Convention in julia is to add "!" in the function name. The solver I use requires a function "with no return value"

³github.com/JuliaNLSolvers/NLsolve.jl

arguments, anonymous functions can resolve problem:

(output, x) -> f(output, x, T, rho, y) (in my case T, rho, y is known..).

```
-8.81
julia> nlsolve((F,x) -> Network_gse.nse_condition!(F, x, 1e9, 1e7, 0.5, a), (dres, x) -> Network_gse.df_nse_condition!(dres, x, 1e9, 1e7, 0.5, a), μ)
Results of Nonlinear Solver Algorithm
* Algorithm: Trust-region with dogleg and autoscaling
* Starting Point: [-8.8, -8.81]
* Zero: [NaN, NaN]
* Inf-norm of residuals: 0.868981
* Iterations: 1
* Convergence: false
* ||x - x*|| < 0.0e+00: false
* ||f(x)|| < 1.0e-08: false
* Function Calls (f): 2
* Jacobian Calls (df/dx): 1

julia> nlsolve((F,x) -> Network_gse.nse_condition!(F, x, 1e9, 1e7, 0.5, a), μ)
Results of Nonlinear Solver Algorithm
* Algorithm: Trust-region with dogleg and autoscaling
* Starting Point: [-8.8, -8.81]
* Zero: [-8.8, -8.81]
* Inf-norm of residuals: 1.868981
* Iterations: 1000
* Convergence: false
* ||x - x*|| < 0.0e+00: false
* ||f(x)|| < 1.0e-08: false
* Function Calls (f): 45
* Jacobian Calls (df/dx): 1

julia>
```

Figure 0.3: Example output for calling NLSolve for e.g. $T = 10^9$ and $\rho = 10^7$