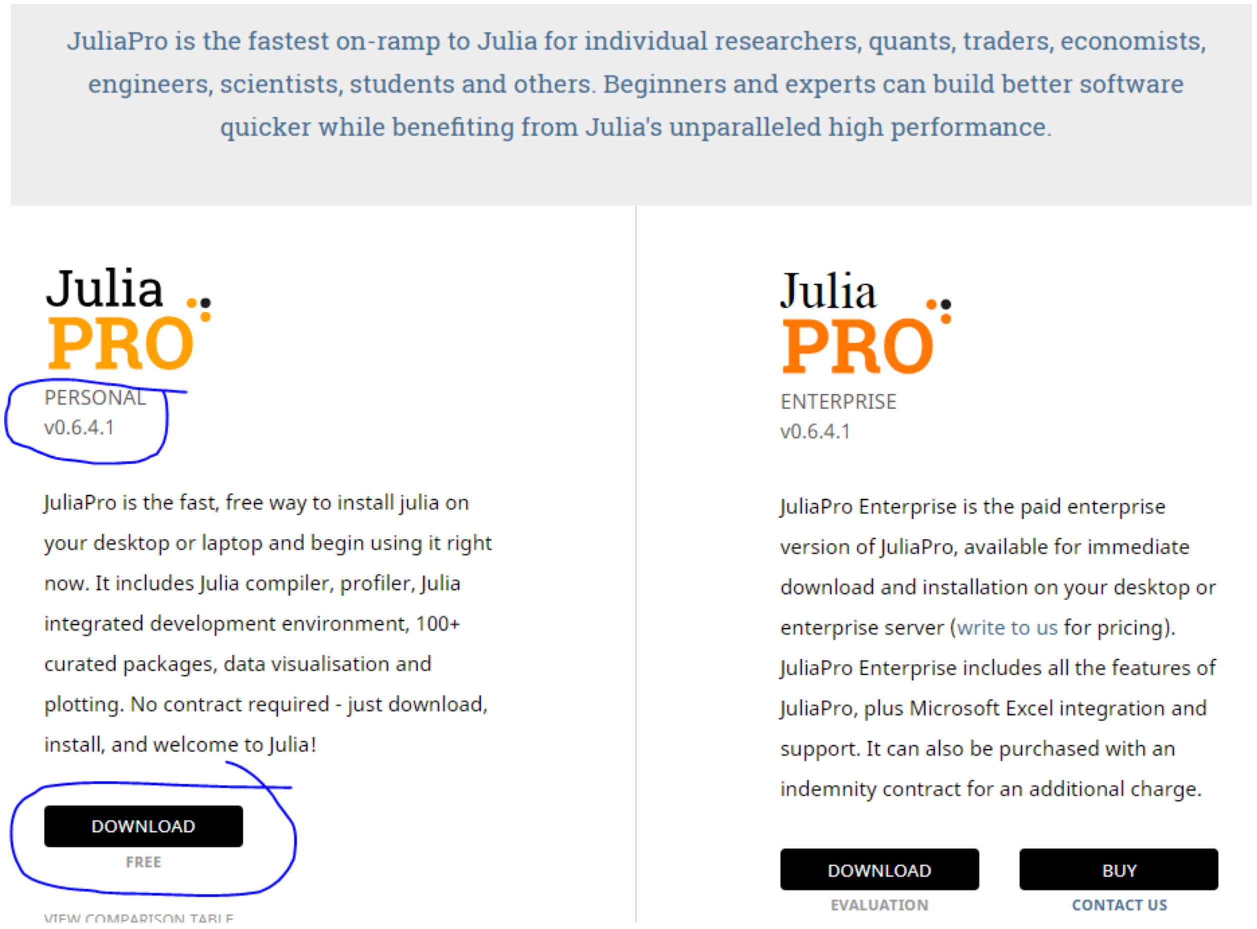
# Instructions for running the program

Estimated duration of set-up: 15 minutes

1. **Install JuliaPro**: The easiest way to run the program is to use JuliaPro which is available free for download. The download link is https://juliacomputing.com/products/juliapro.html . And install it.
    a. This contains the IDE and other packages. Although only core packages are used for processing, Gadfly is used for plots.
    b. This is self-contained and does not require admin rights to install on any system.
    c. This can be installed on any OS – Windows, Linux or Mac



JuliaPro is the fastest on-ramp to Julia for individual researchers, quants, traders, economists, engineers, scientists, students and others. Beginners and experts can build better software quicker while benefiting from Julia's unparalleled high performance.

Julia PRO
PERSONAL
v0.6.4.1

JuliaPro is the fast, free way to install julia on your desktop or laptop and begin using it right now. It includes Julia compiler, profiler, Julia integrated development environment, 100+ curated packages, data visualisation and plotting. No contract required - just download, install, and welcome to Julia!

DOWNLOAD
FREE

VIEW COMPARISON TABLE

Julia PRO
ENTERPRISE
v0.6.4.1

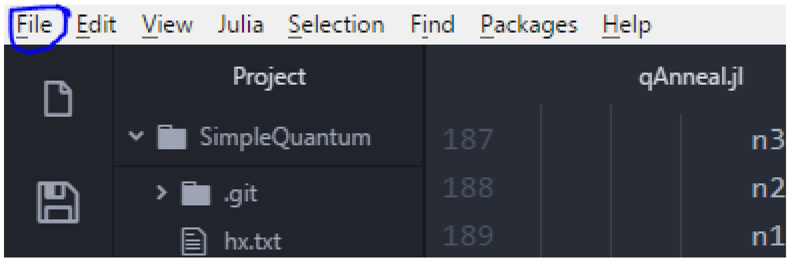
JuliaPro Enterprise is the paid enterprise version of JuliaPro, available for immediate download and installation on your desktop or enterprise server (write to us for pricing). JuliaPro Enterprise includes all the features of JuliaPro, plus Microsoft Excel integration and support. It can also be purchased with an indemnity contract for an additional charge.

DOWNLOAD
EVALUATION

BUY
CONTACT US

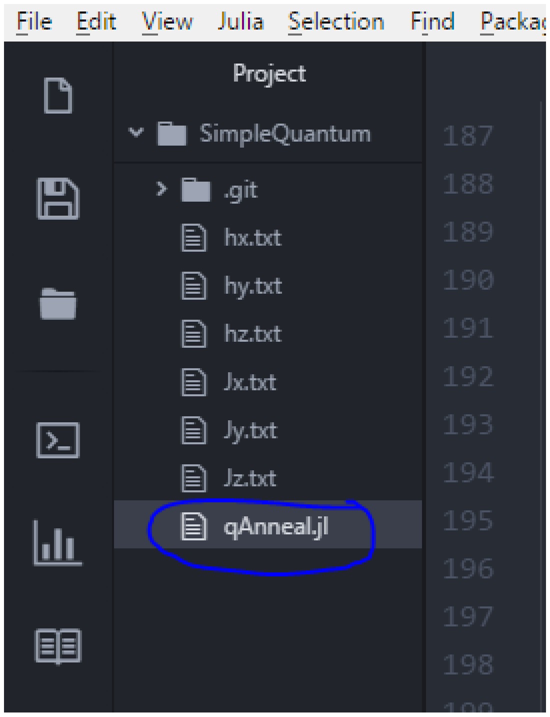
2. **Clone** the github project on linux command line as "git clone https://github.com/Code1983/qAnneal" or download the project folder directly from the link.

3. **Open Juno**, the IDE for Julia that was installed as part of JuliaPro. The installation would have created a desktop shortcut in windows. In linux, a  script “Juno” might have been created in installation folder.

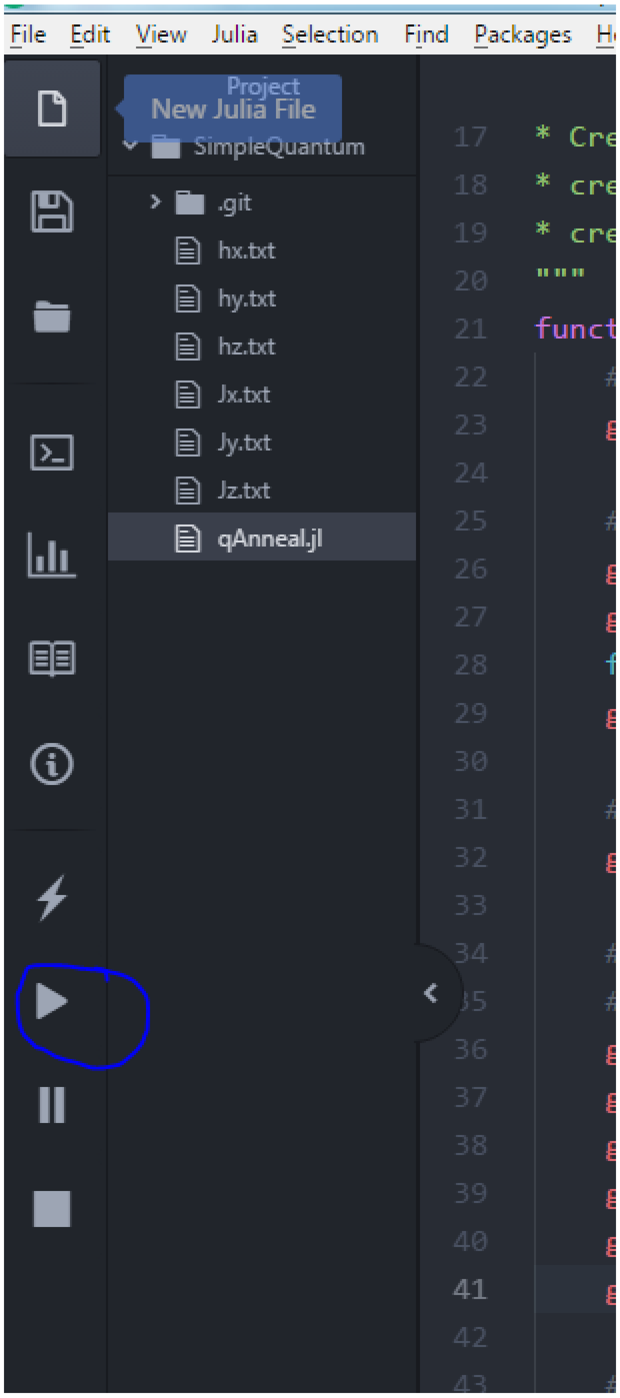


4.  **Open the project** on Juno, open File → Open Folder → provide the location of “src” folder within github cloned/downloaded project.

5. **Open qAnneal.jl** file

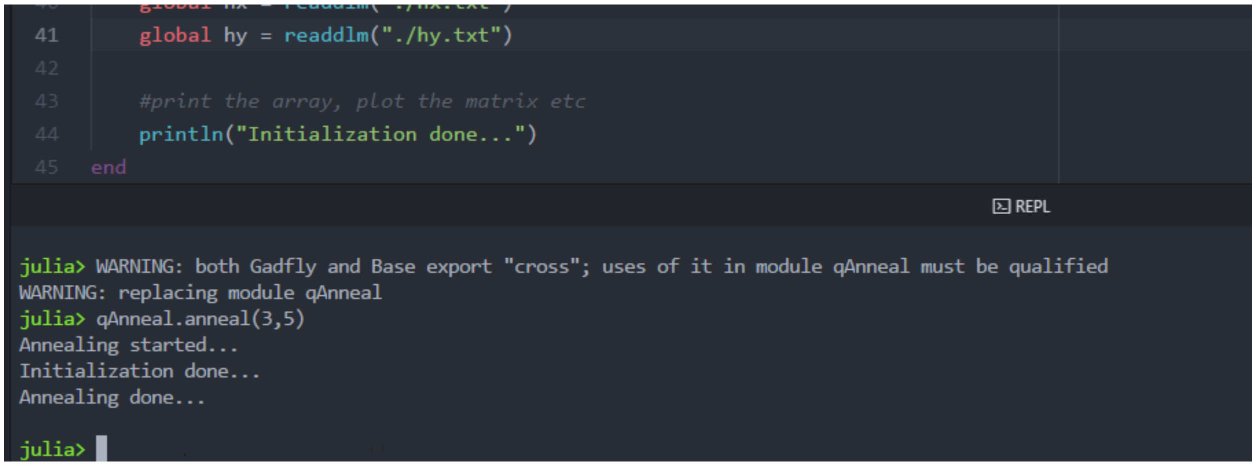


6. **Run the program** using the run button on left side



7. **Update the hx, hy, hz and Jx, Jy, Jz** files within the project with appropriate values.

8. **Anneal** by hitting enter on the REPL Window and enter the following command qAnneal.anneal(3,5). The first parameter is the number of qubits and second parameter is the number of steps in which it will transition from initial state (all hx=1) to final state. The first execution will take longer because compilation happens at this time.



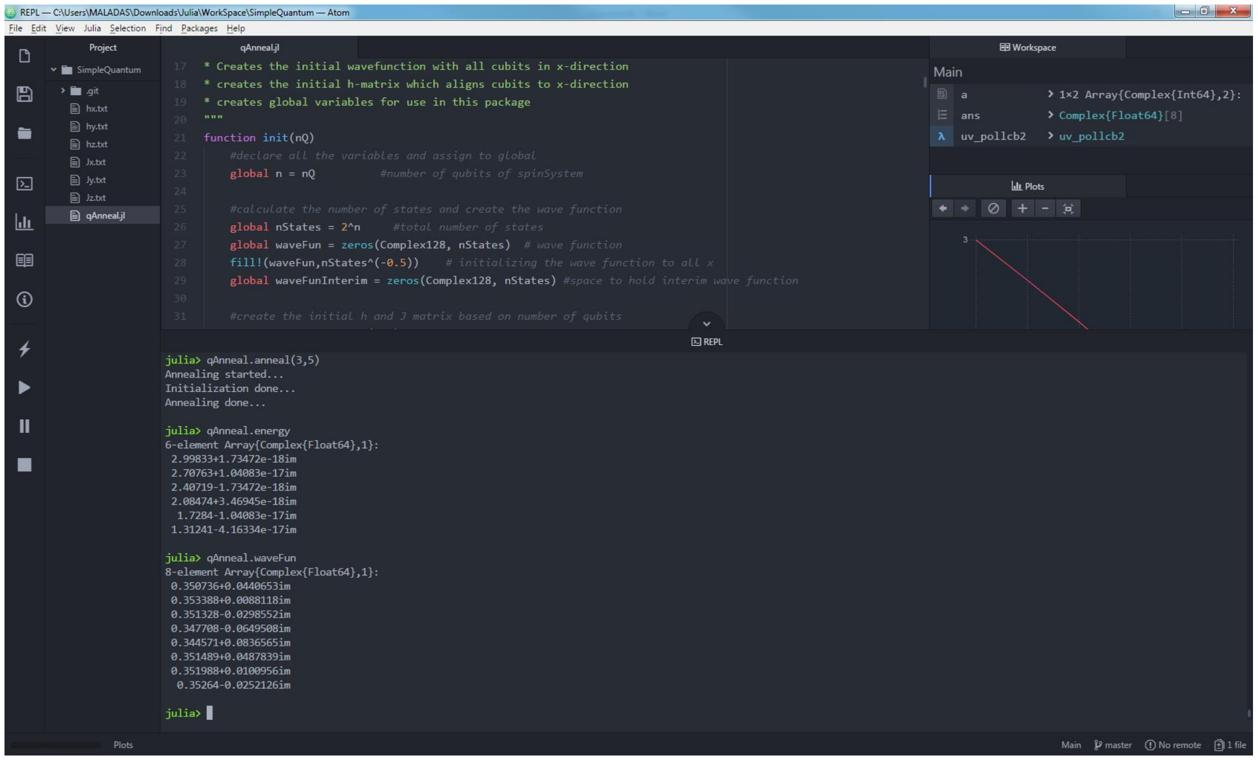
```
    global hy = readdlm("./hy.txt")

    #print the array, plot the matrix etc
    println("Initialization done...")
end
```

```
julia> WARNING: both Gadfly and Base export "cross"; uses of it in module qAnneal must be qualified
WARNING: replacing module qAnneal
julia> qAnneal.anneal(3,5)
Annealing started...
Initialization done...
Annealing done...

julia> 
```

9. **Plots** - The execution will generate the energy plot as shown below


REPL — C:\Users\MALADAS\Downloads\Julia\WorkSpace\SimpleQuantum — Atom
File Edit View Julia Selection Find Packages Help
Project
SimpleQuantum
.git
hx.txt
hy.txt
hz.txt
Jx.txt
Jy.txt
Jz.txt
qAnneal.jl
Workspace
Main
Plots
3
REPL
Main
master
No remote
1 file

```
* Creates the initial wavefunction with all cubits in x-direction
* creates the initial h-matrix which aligns cubits to x-direction
* creates global variables for use in this package
"""
function init(nQ)
    #declare all the variables and assign to global
    global n = nQ          #number of qubits of spinSystem

    #calculate the number of states and create the wave function
    global nStates = 2^n     #total number of states
    global waveFun = zeros(Complex128, nStates)  # wave function
    fill!(waveFun,nStates^(-0.5))    # initializing the wave function to all x
    global waveFunInterim = zeros(Complex128, nStates) #space to hold interim wave function

    #create the initial h and J matrix based on number of qubits
```

```
a            1×2 Array{Complex{Int64},2}:
ans          Complex{Float64}[8]
uv_pollcb2   uv_pollcb2
```

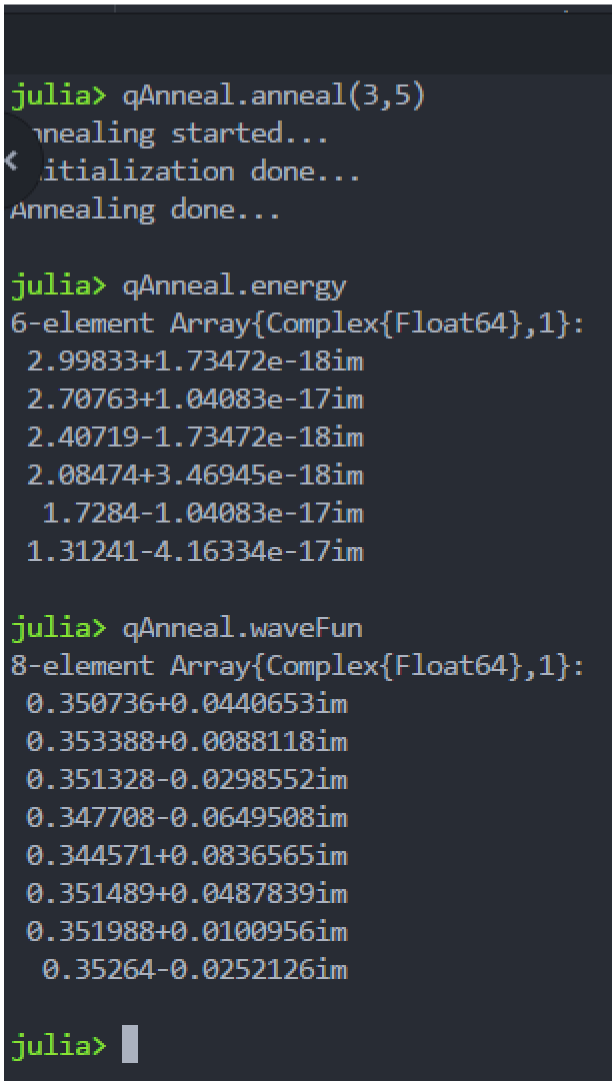
```
julia> qAnneal.anneal(3,5)
Annealing started...
Initialization done...
Annealing done...

julia> qAnneal.energy
6-element Array{Complex{Float64},1}:
 2.99833+1.73472e-18im
 2.70763+1.04083e-17im
 2.40719-1.73472e-18im
 2.08474+3.46945e-18im
  1.7284-1.04083e-17im
 1.31241-4.16334e-17im

julia> qAnneal.waveFun
8-element Array{Complex{Float64},1}:
 0.350736+0.0440653im
 0.353388+0.0088118im
 0.351328-0.0298552im
 0.347708-0.0649508im
 0.344571+0.0836565im
 0.351489+0.0487839im
 0.351988+0.0100956im
  0.35264-0.0252126im

julia> 
```

10. **Investigating variables** - The energy (at each timestep in incremental order) can be viewed as qAnneal.energy and wavefunction (the states are in order |000> |001> |010>…) can be viewed as qAnneal.waveFun



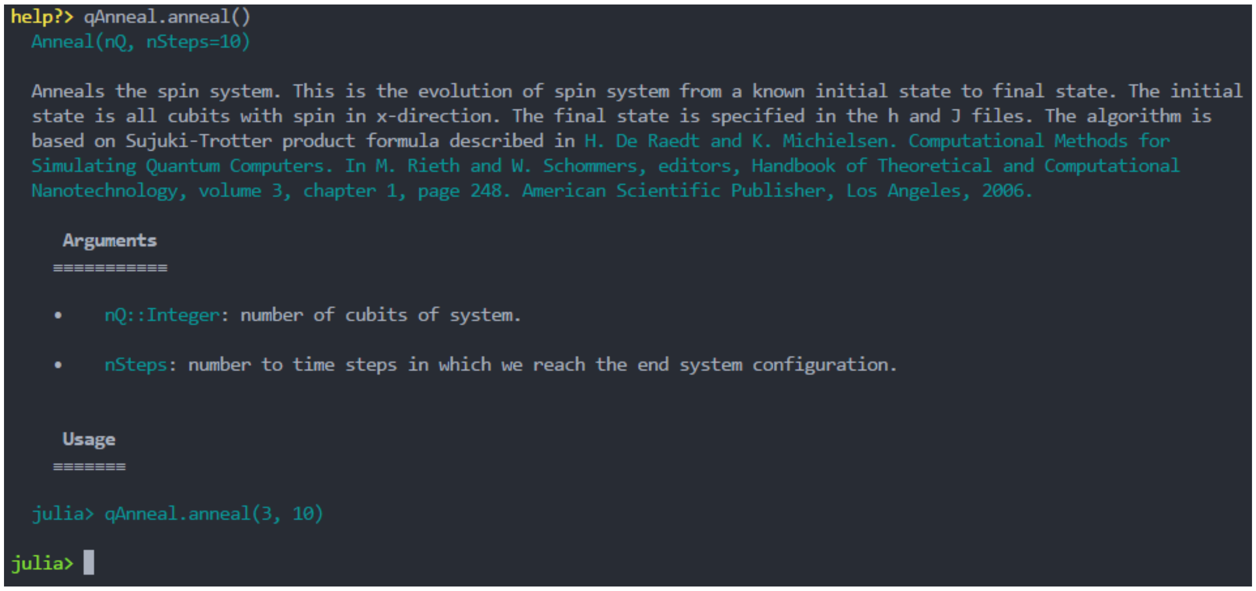
```
julia> qAnneal.anneal(3,5)
 nnealing started...
 itialization done...
Annealing done...

julia> qAnneal.energy
6-element Array{Complex{Float64},1}:
 2.99833+1.73472e-18im
 2.70763+1.04083e-17im
 2.40719-1.73472e-18im
 2.08474+3.46945e-18im
  1.7284-1.04083e-17im
 1.31241-4.16334e-17im

julia> qAnneal.waveFun
8-element Array{Complex{Float64},1}:
 0.350736+0.0440653im
 0.353388+0.0088118im
 0.351328-0.0298552im
 0.347708-0.0649508im
 0.344571+0.0836565im
 0.351489+0.0487839im
 0.351988+0.0100956im
  0.35264-0.0252126im

julia>
```

11. **Help** - Documentation of each function can be viewed by writing “?” followed by function name.



```
help?> qAnneal.anneal()
  Anneal(nQ, nSteps=10)

  Anneals the spin system. This is the evolution of spin system from a known initial state to final state. The initial
  state is all cubits with spin in x-direction. The final state is specified in the h and J files. The algorithm is
  based on Sujuki-Trotter product formula described in H. De Raedt and K. Michielsen. Computational Methods for
  Simulating Quantum Computers. In M. Rieth and W. Schommers, editors, Handbook of Theoretical and Computational
  Nanotechnology, volume 3, chapter 1, page 248. American Scientific Publisher, Los Angeles, 2006.

     Arguments
    ===========

    •    nQ::Integer: number of cubits of system.

    •    nSteps: number to time steps in which we reach the end system configuration.


     Usage
    =======

  julia> qAnneal.anneal(3, 10)

julia> 
```