```
Activating project at `~/projects/QMBCTutorial/notebooks`
```

☐ Full Width Mode ☐ Present Mode

Table of Contents


```julia
TableOfContents()
```

Julia for Quantum Many-Body Computation

A Starter Kit

Yusheng Zhao, Jinguo Liu

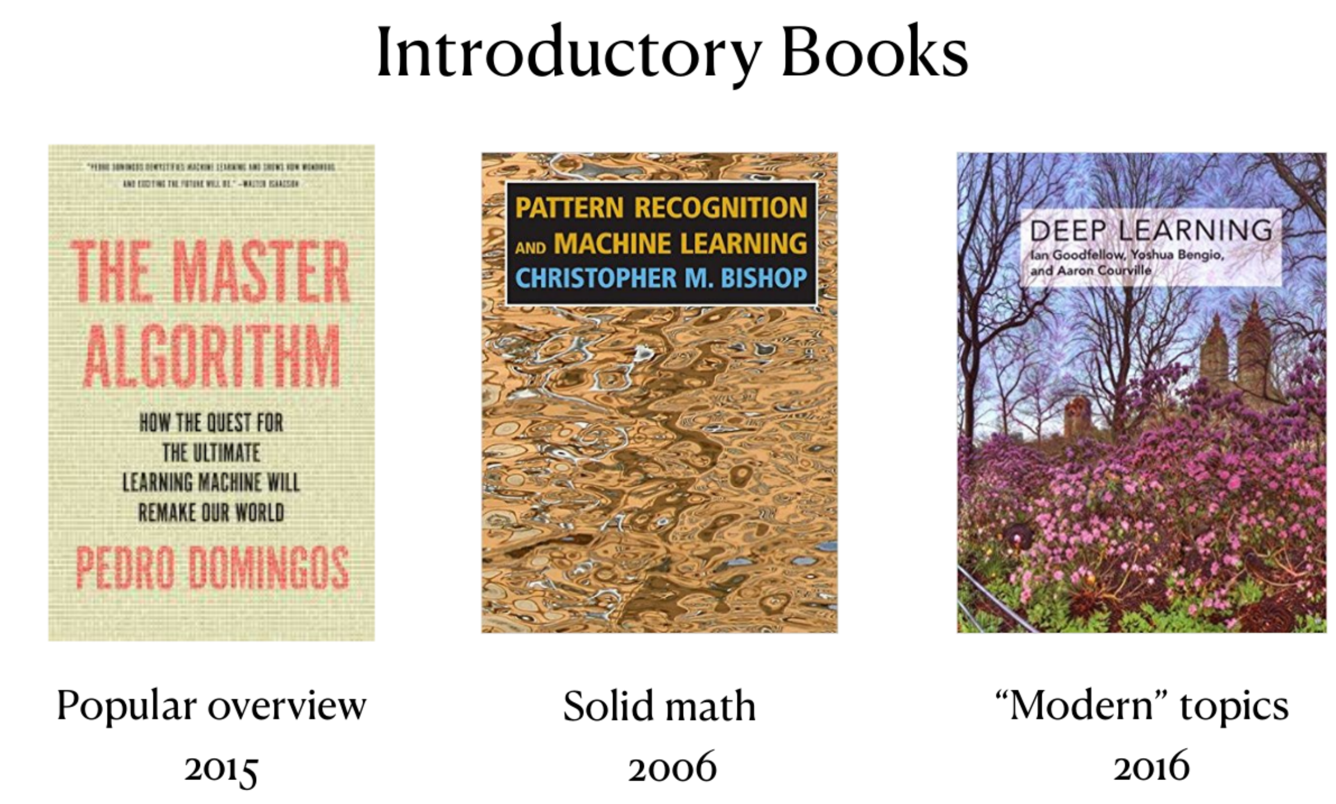
MinJiang University, Fu Zhou, 08/17/2023

Promised: Road to mastering machine learning

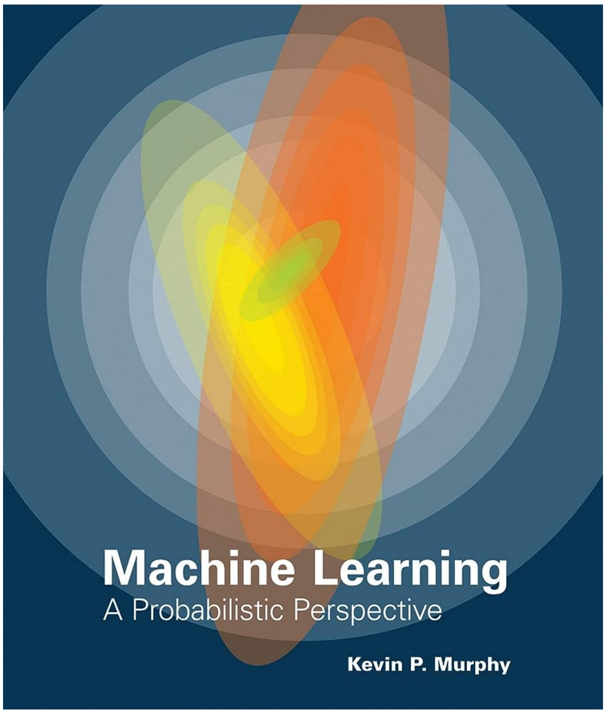
This lecture is supposed to be delivered by Lei. He wanted to lecture about machine learning. If you are interesting in learning machine learning for physicists, please check the following repo and books.

Machine learning for physicists

Github: wangleiphy/ml4p

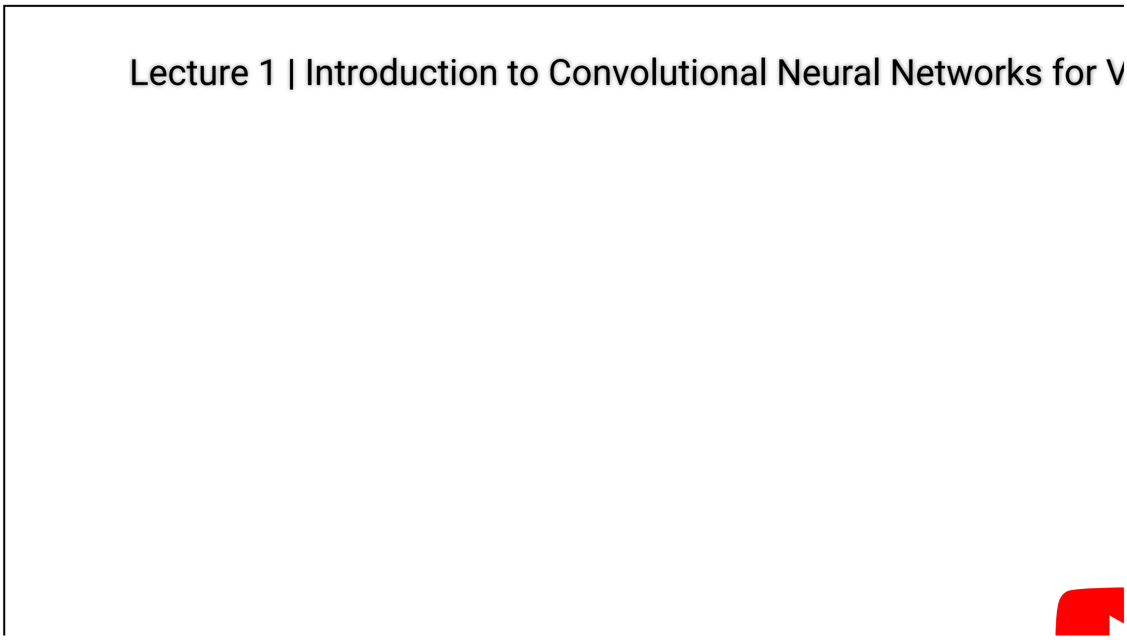


The book that I enjoyed reading



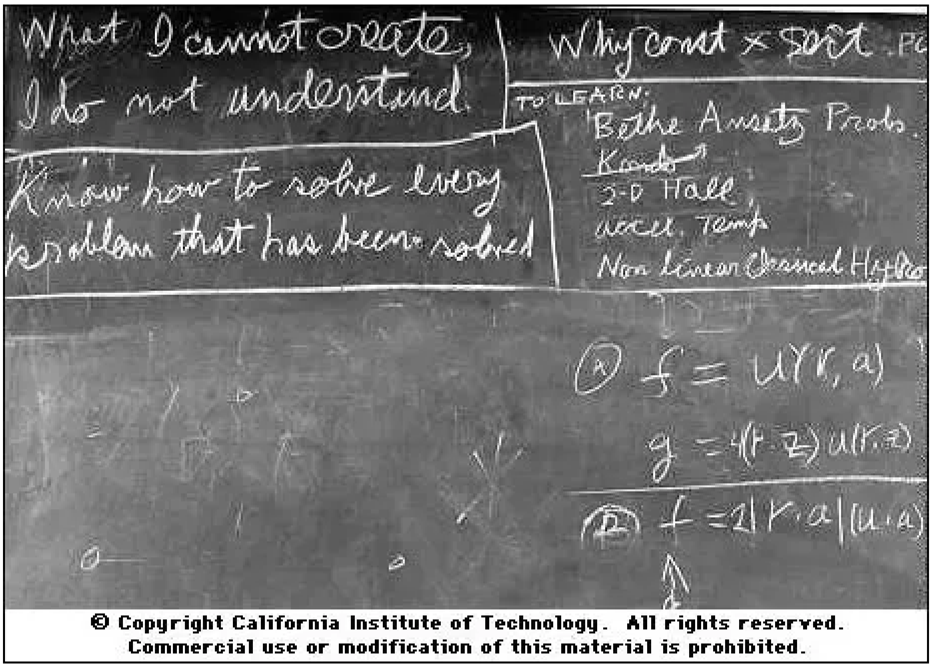
Online lecture: CS231n

Youtube Video



Motivation

"What I cannot create, I do not understand" - R.P Feynman



What language to use?


Nice paper
GitHub Link
Written in your favourite framework
Runs smoothly on your system without error or dependency issues

Anders Sandvik's (Creater of Stochastic Series Expansion Monte Carlo method) Answer

- Anders Sandvik's course page

Why Julia?

Short Answer

- **Speed**
- **Easy to use**
- **Reproducibility**

What is Julia?

Julia is an **unconventional dynamic** programming language with the goal of being **easy to be made fast**.

- Shallow learning curve & high flexibility
- ~~Low performance~~
- ~~Two language problem~~

```
sumtil (generic function with 1 method)
```

```julia
function sumtil(n)
    x = 0
    for i in 1:n
        x += i
    end
    return x
end
```

```
Process(`cat lib/demo.c`, ProcessExited(0))

  #include<stddef.h>

  int c_sumtil(size_t n) {
      int s = 0;
      for (size_t i=1; i<=n; i++) {
          s += i;
      }
      return s;
  }
```

```
Process(`gcc lib/demo.c -fPIC -O3 -shared -o lib/demo.so`, ProcessExited(0))
```

```julia
run(`gcc lib/demo.c -fPIC -O3  -shared -o lib/demo.so`)
```

```julia
using Libdl
```

```
c_sumtil (generic function with 1 method)
```

```julia
c_sumtil(x) = Libdl.@ccall "lib/demo.so".c_sumtil(x::Csize_t)::Int
```

```julia
using PyCall
```

```julia
using BenchmarkTools
```

```
2820230816
```

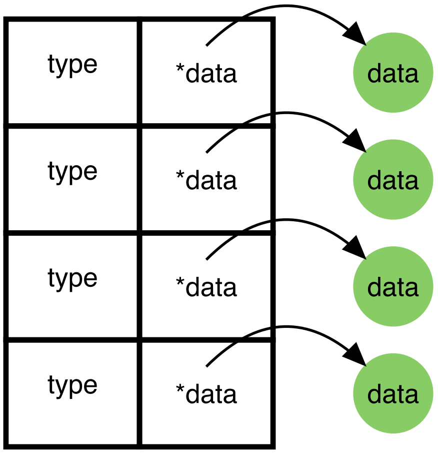
```julia
begin
    py"""
    def sumtil(n):
        x = 0
        for i in range(1, n+1):
            x += i
        return x
    """
    @btime py"sumtil"(200000);
    @btime sumtil(200000);
    @btime c_sumtil(200000);
end
```

```
6.168 ms (6 allocations: 192 bytes)
2.041 ns (0 allocations: 0 bytes)
2.875 ns (0 allocations: 0 bytes)
```

Easy to be made fast

When is a program fast?

- Computers are fast when it knows EXACTLY what to do
- Box and Pointer Model
- Slowness of a *typical* Dynamic language is related to cache miss.



Julia is different!

- Julia has a well structured type system.
- JIT(Just in time) compilation with *Type Inference*

```
CodeInfo(
1 ─ %1 = Base.add_float(x, y)::Float64
└──      return %1
)                                            ⇒ Float64
```

```julia
@code_typed 1.0 + 2.0
```

```
CodeInfo(
1 ─ %1 = Base.sitofp(Float64, x)::Float64
│   %2 = Base.add_float(%1, y)::Float64
└──      return %2
)                                               ⇒ Float64
```

```julia
@code_typed 1 + 2.0
```

```
	.section	__TEXT,__text,regular,pure_instructions
	.build_version macos, 13, 0
	.globl	"_julia_+_5654"                 ; -- Begin function julia_+_5654
	.p2align	2
"_julia_+_5654":                        ; @"julia_+_5654"
; ┌ @ float.jl:408 within `+`
	.cfi_startproc
; %bb.0:                                ; %top
	fadd	d0, d0, d1
	ret
	.cfi_endproc
; └
                                        ; -- End function
.subsections_via_symbols
```

```julia
with_terminal() do
    @code_native 1.0 + 2.0
end
```

```
CodeInfo(
1 ─ %1  = Base.sle_int(1, n)::Bool
└──       goto #3 if not %1
2 ─       goto #4
3 ─       goto #4
4 ┄ %5  = φ (#2 => n, #3 => 0)::Int64
└──       goto #5
5 ─       goto #6
6 ─ %8  = Base.slt_int(%5, 1)::Bool
└──       goto #8 if not %8
7 ─       goto #9
8 ─       goto #9
9 ┄ %12 = φ (#7 => true, #8 => false)::Bool
│   %13 = φ (#8 => 1)::Int64
│   %14 = φ (#8 => 1)::Int64
│   %15 = Base.not_int(%12)::Bool
└──       goto #15 if not %15
10 ┄ %17 = φ (#9 => %13, #14 => %26)::Int64
│   %18 = φ (#9 => %14, #14 => %27)::Int64
│   %19 = φ (#9 => 0, #14 => %20)::Int64
│   %20 = Base.add_int(%19, %17)::Int64
│   %21 = (%18 === %5)::Bool
└──       goto #12 if not %21
11 ─       goto #13
12 ─ %24 = Base.add_int(%18, 1)::Int64
└──       goto #13
13 ┄ %26 = φ (#12 => %24)::Int64
│   %27 = φ (#12 => %24)::Int64
│   %28 = φ (#11 => true, #12 => false)::Bool
│   %29 = Base.not_int(%28)::Bool
└──       goto #15 if not %29
14 ─       goto #10
15 ┄ %32 = φ (#13 => %20, #9 => 0)::Int64
└──       return %32
)                                                  => Int64
```

```julia
with_terminal() do
    @code_typed sumtil(2)
end
```

```julia
run(`gcc -S lib/add.c`);
```

```
Process(`cat lib/add.s`, ProcessExited(0))

	.section	__TEXT,__text,regular,pure_instructions
	.build_version macos, 13, 0 sdk_version 13, 3
	.globl	_main                           ; -- Begin function main
	.p2align	2
_main:                                  ; @main
	.cfi_startproc
; %bb.0:
	sub	sp, sp, #16
	.cfi_def_cfa_offset 16
	str	wzr, [sp, #12]
	fmov	s0, #3.00000000
	str	s0, [sp, #8]
	mov	w0, #1
	add	sp, sp, #16
	ret
	.cfi_endproc
                                        ; -- End function
.subsections_via_symbols
```

```
with_terminal() do
    run(`cat lib/add.s`)
end
```

Installation and Setup

Installation Guide: CodingThrust/CodingClub

How to program in Julia

Grammars: Tutorial

- Almost Python like, but
  - Index starts from 1
  - Column Major

Multiple Dispatch

- Programming Paradigm
- Expressiveness
- Contrasted with Single Dispatch Polymorphism

```
Enter cell code...
```

```
abstract type Pet end
```

```julia
struct Dog <: Pet
        name::String
    end
```

```julia
struct Cat <: Pet
        name::String
    end
```

encounter (generic function with 1 method)

```julia
function encounter(a::Pet, b::Pet)
        verb = meets(a,b)
        println("$(a.name) meets $(b.name) and $(verb)")
    end
```

meets (generic function with 1 method)

```julia
meets(a::Pet, b::Pet) = "FALLBACK"
```

meets (generic function with 2 methods)

```julia
meets(a::Dog, b::Dog) = "sniffs"
```

meets (generic function with 3 methods)

```julia
meets(a::Dog, b::Cat) = "chases"
```

meets (generic function with 4 methods)

```julia
meets(a::Cat, b::Dog) = "hisses"
```

meets (generic function with 5 methods)

```julia
meets(a::Cat, b::Cat) = "slinks"
```

```julia
sam = Dog("Sam");
```

```julia
bob = Dog("Bob");
```

```julia
erwin = Cat("Erwin");
```

```julia
tom = Cat("Tom");
```

```julia
encounter(sam, bob)
```

```
Sam meets Bob and sniffs
```

```julia
encounter(sam, erwin)
```

```
Sam meets Erwin and chases
```

```julia
encounter(erwin, bob)
```

```
Erwin meets Bob and hisses
```

```
encounter(erwin, tom)
```

```
Erwin meets Tom and slinks
```

```
Process(`g++ lib/multidispatch.cpp -o pets`, ProcessExited(0))
```

```
run(`g++ lib/multidispatch.cpp -o pets`)
```

```
Process(`cat lib/multidispatch.cpp`, ProcessExited(0))
```

```
#include <iostream>
#include <string>

using namespace std;

class Pet {
    public:
        string name;
};

string meets(Pet a, Pet b) {
    return "FallBACK";
}

void encounter(Pet a, Pet b) {
    string verb = meets(a, b);
    cout << a.name << " meets " << b.name << " " << verb << endl;
}
```

```
with_terminal() do
    run(`cat lib/multidispatch.cpp`)
end
```

```
Process(`./pets`, ProcessExited(0))
```

```
run(`./pets`)
```

```
Sam meets Bob FallBACK
Sam meets Erwin FallBACK
Erwin meets Bob FallBACK
Erwin meets Tom FallBACK
```

Benefit of MD

- Easy to define `new types` where old operations apply
- Easy to implement `new operations` on old types

```
struct Snake <: Pet
    name::String
end
```

```
begin
    Nagini = Snake("Nagini")
    encounter(Nagini, erwin)
end
```

```
Nagini meets Erwin and FALLBACK
```

hears (generic function with 1 method)

```
function hears(a::Cat, b::Union{Dog,Snake})
    println("$(a.name) hears $(b.name) and has a piloerection")
end
```

```
hears(erwin,Nagini)
```

```
Erwin hears Nagini and has a piloerection
```

2.8722813232690143

```
@btime sqrt(sum(abs2, (2.0, 3.0, 4.0, 5.5) .- (1.0, 3.0, 5.0, 3.0)))
```

```
  1.208 ns (0 allocations: 0 bytes)
```

Demo

MyPoint

```
begin
    struct MyPoint{N, T<:Real}
        coo::NTuple{N, T}
    end
    MyPoint(args::Real...) = MyPoint((args...,))
end
```

```
function Base.:-(p1::MyPoint{N, T}, p2::MyPoint{N,T}) where {N,T}
    MyPoint(p1.coo .- p2.coo)
end
```

p1 = MyPoint((2.0, 3.0))

```
p1 = MyPoint(2.0, 3.0)
```

p2 = MyPoint((3.0, 4.0))

```
p2 = MyPoint(3.0, 4.0)
```

MyPoint((-1.0, -1.0))

```
@btime p1 - p2
```

```
  32.193 ns (1 allocation: 32 bytes)
```

```
p3 =  MyPoint((3.0, 4.0, 5.0))
```

```julia
p3 = MyPoint(3.0, 4.0, 5.0)
```

```
p4 =  MyPoint((3.0, 4.0, 6.0))
```

```julia
p4 = MyPoint(3.0, 4.0, 6.0)
```

```
MyPoint((0.0, 0.0, 1.0))
```

```julia
p4 - p3
```

Ecosystem

Exact Diagonalization

- We are all experts in ED
- Important algorithm for benchmarking
- Demonstrate ED of a 1D Heisenberg XXZ model: $H = \sum_{<i,j>} J\sigma_i^x\sigma_j^x + J\sigma_i^y\sigma_j^y + J_z\sigma_i^z\sigma_j^z$
- Construct Hamiltonian using `Yao.jl`
- More professionally ExactDiagonalization.jl

Yao

- Yao is an Extensible, Efficient Quantum Algorithm Design library For Humans written and maintained by Xiuzhe (Roger) Luo and Jin-Guo Liu
- arXiv:1912.10877
-  
- Construction of Hamiltonian as Sparse Matrices

```
make_XXZhamiltonian (generic function with 1 method)
```

```julia
function make_XXZhamiltonian(L::Int, J::Real, Jz::Real; periodic::Bool=false)
    # construct the Hamiltonian
    # L: number of sites
    # J: coupling strength between pauli X and Y
    # Jz: coupling strength between pauli Z
    # return: Hamiltonian
    offset = periodic ? 0 : 1
    hamiltonian = sum([Jz * kron(L, i=>Z, mod1(i+1,L)=>Z) for i in 1:L-offset])
    for pauli in [X,Y]
        hamiltonian += sum([J * kron(L, i=>pauli , mod1(i+1, L)=>pauli) for i in 1:L-offset])
    end

    return hamiltonian
end
```

Number of sites: 10

```
ham = nqubits: 10
+
├─ +
│  ├─ +
│  │  ├─ [+] kron
│  │  │     ├─ 1=>Z
│  │  │     └─ 2=>Z
│  │  ├─ [+] kron
│  │  │     ├─ 2=>Z
│  │  │     └─ 3=>Z
│  │  ├─ [+] kron
│  │  │     ├─ 3=>Z
│  │  │     └─ 4=>Z
│  │  ├─ [+] kron
│  │  │     ├─ 4=>Z
│  │  │     └─ 5=>Z
│  │  ├─ [+] kron
│  │  │     ├─ 5=>Z
│  │  │     └─ 6=>Z
│  │  ├─ [+] kron
│  │  │     ├─ 6=>Z
│  │  │     └─ 7=>Z
│  │  ├─ [+] kron
│  │  │     ├─ 7=>Z
│  │  │     └─ 8=>Z
│  │  ├─ [+] kron
│  │  │     ├─ 8=>Z
│  │  │     └─ 9=>Z
│  │  └─ [+] kron
│  │        ├─ 9=>Z
│  │        └─ 10=>Z
│  └─ +
│     ├─ [+] kron
```

```julia
ham = make_XXZhamiltonian(nq, 1.0, 1.0; periodic=false)
```

```
1024×1024 SparseMatrixCSC{ComplexF64, Int64} with 5632 stored entries:
```

```
# I can fit 22 qubits, good!
mat(ham)
```

Under the hood

- Pauli Matrices are constructed as sparse matrices

`(:Y, PermMatrix([2, 1], ConstGateDefaultType[-im, im]))` code

- Kronecker products and summations are supported on sparse matrices

```
2×2 LuxurySparse.SDPermMatrix{ComplexF64, Int64, Vector{ComplexF64}, Vector{Int64}}:
 0.0+0.0im  0.0-1.0im
 0.0+1.0im  0.0+0.0im
```

```
mat(Y)
```

```
4×4 LuxurySparse.SDPermMatrix{ComplexF64, Int64, Vector{ComplexF64}, Vector{Int64}}:
 0.0+0.0im  0.0+0.0im  0.0+0.0im  1.0+0.0im
 0.0+0.0im  0.0+0.0im  1.0+0.0im  0.0+0.0im
 0.0+0.0im  1.0+0.0im  0.0+0.0im  0.0+0.0im
 1.0+0.0im  0.0+0.0im  0.0+0.0im  0.0+0.0im
```

```
begin
    yy = mat(kron(2,Y,Y));
    println(yy.perm)
    println(yy.vals)
    xx = mat(kron(2,X,X));
end
```

```
[4, 3, 2, 1]
ComplexF64[-1.0 - 0.0im, 1.0 + 0.0im, 1.0 + 0.0im, -1.0 + 0.0im]
```

```
4×4 SparseMatrixCSC{ComplexF64, Int64} with 2 stored entries:
     ⋅          ⋅          ⋅          ⋅    
     ⋅          ⋅      2.0+0.0im      ⋅    
     ⋅      2.0+0.0im      ⋅          ⋅    
     ⋅          ⋅          ⋅          ⋅    
```

```
begin
    xxpyy = sum([kron(2,X,X),kron(2,Y,Y)])
    mat(xxpyy)
end
```

Eigenvalue solving

- We will use `KrylovKit.jl`
- It is a Julia package collecting a number of Krylov-based algorithms for linear problems, singular value and eigenvalue problems and the application of functions of linear maps or operators to vectors.

Show following doc: ☐

```
if show_te_doc1
    @doc eigsolve
end
```

```
([-17.0321, -15.7227, -15.7227], [[7.87077e-21+5.00346e-19im, 4.73347e-18+1.31873e-18im,
```

```
eval_eds, vecs_ed, info_ed =  eigsolve(mat(ham), 3, :SR,ishermitian=true)
```

Time Evolution

API for doing real time evolution and imaginary time evolution

Show following doc: ☐

```
if show_te_doc
    @doc time_evolve
end
```

Do Imaginary time evolution: ☐

```
begin
    if do_eval
        evo_op = time_evolve(ham,-im*0.01;tol=1e-10,check_hermicity=false)
        ψ = rand_state(nq)
        for _ in 1:300
            apply!(ψ,evo_op)
            ψ = Yao.normalize!(ψ)
        end
        e_it = real(Yao.expect(ham,ψ));
    end
end
```

DMRG

- If you want to be a happy API caller, just use `ITensors.jl`
- `ITensors.jl` is a library for rapidly creating correct and efficient tensor network algorithms

Hamiltonian

- We create Matrix Product Operator using ITensor interface
- The same XXZ Model as in ED section

```
begin
    using ITensors
end
```

make_xxzmpo (generic function with 1 method)

```
function make_xxzmpo(L::Int, J::Real, Jz::Real; periodic::Bool=false)
    sites = siteinds("S=1/2", L)
    ham = OpSum()
    offset = periodic ? 0 : 1
    for i in 1:L-offset
        ham += 4*J , "Sx",i,"Sx",mod1(i+1,L)
        ham += 4*J , "Sy",i,"Sy",mod1(i+1,L)
        ham += 4*Jz, "Sz", i, "Sz", mod1(i+1,L)
    end
    return MPO(ham, sites), sites
end
```

```
(ITensors.MPO
 [1] ((dim=5|id=415|"Link,l=1"), (dim=2|id=226|"S=1/2,Site,n=1")', (dim=2|id=226|"S=1/2
```

```
xxzmpo, xxzsites = make_xxzmpo(nq,1.0,1.0;periodic=false)
```

do_dmrg (generic function with 1 method)

```
function do_dmrg(H,sites,psi0_i,sweeps::Int, maxdims::Vector{Int},cutoff::Float64)
    # Do 10 sweeps of DMRG, gradually
    # increasing the maximum MPS
    # bond dimension
    sweeps = Sweeps(sweeps)
    setmaxdim!(sweeps,maxdims...)
    setcutoff!(sweeps,cutoff) # Run the DMRG algorithm
    energy,psi0 = dmrg(H,psi0_i,sweeps)
end
```

```
(-17.0321, ITensors.MPS
           [1] ((dim=2|id=283|"Link,l=1"), (dim=2|id=226|"S=1/2,Site,n=1"))
```

```
begin
    psi0 = randomMPS(xxzsites;linkdims=10)
    energy, ψ₀ = do_dmrg(xxzmpo,xxzsites,psi0,10,
    [10,20,50,60,80,100,120,140,160,180,200],1e-11)
end
```

```
After sweep 1 energy=-17.029858236314574  maxlinkdim=10 maxerr=4.28E-04 time=0.008
After sweep 2 energy=-17.032140820419407  maxlinkdim=20 maxerr=2.81E-10 time=0.010
After sweep 3 energy=-17.03214082892069  maxlinkdim=25 maxerr=9.38E-12 time=0.021
After sweep 4 energy=-17.03214082892078  maxlinkdim=25 maxerr=9.38E-12 time=0.011
After sweep 5 energy=-17.032140828920802  maxlinkdim=25 maxerr=9.38E-12 time=0.018
After sweep 6 energy=-17.03214082892078  maxlinkdim=25 maxerr=9.38E-12 time=0.018
After sweep 7 energy=-17.032140828920785  maxlinkdim=25 maxerr=9.38E-12 time=0.012
After sweep 8 energy=-17.032140828920813  maxlinkdim=25 maxerr=9.38E-12 time=0.014
After sweep 9 energy=-17.032140828920813  maxlinkdim=25 maxerr=9.38E-12 time=0.011
After sweep 10 energy=-17.032140828920777  maxlinkdim=25 maxerr=9.38E-12 time=0.013
```

Answer Comparison

Information

- Should you be interested in Julia, you can join the following communities to obtain more information
  1. Slack
  2. Zulip
  3. HKUST(GZ) Zulip
  4. Julia Discourse
  5. JuliaCN Discourse
- There are many opportunities for you to contribute to the Julia community
  1. OSPP
  2. JSoC
  3. GSoC

Acknowledgements

We appreciate the help of Gui-Xin Liu, Shi-Gang Ou, Rui-Si Wang, and Zhongyi Ni during the preparation of this presentation.

Where to find us

You can find Yusheng at his Github page. You can find Jin-Guo at his Github page.

References

- Why is Julia faster than Python?
- Why is Julia so fast
- TUM Course on Machine Learning using Julia
- Setting Up Julia PkgServer
- Is Julia Static or Dynamic
- Steven Johnson Lecture
- The ITensor Software Library for Tensor Network Calculations