Analyzing Quantum Many-Body Systems with ITensor and PastaQ

Matthew Fishman
Center for Computational Quantum Physics (CCQ)
Flatiron Institute, NY
https://mtfishman.github.io/

February 25, 2022

FLATIRON INSTITUTE
Center for Computational Quantum Physics

Who am I?

- I received my PhD from Caltech with **John Preskill** and **Steve White** (UCI) in 2018.

mtfishman.github.io

Who am I?

- I received my PhD from Caltech with **John Preskill** and **Steve White** (UCI) in 2018.
- I visited **Frank Verstraete** and **Jutho Haegeman** during my PhD (Vienna, Ghent).

mtfishman.github.io

FLATIRON INSTITUTE
Center for Computational Quantum Physics

universität wien

Who am I?

- I received my PhD from Caltech with **John Preskill** and **Steve White** (UCI) in 2018.
- I visited **Frank Verstraete** and **Jutho Haegeman** during my PhD (Vienna, Ghent).
- Thesis on developing tensor network algorithms.

mtfishman.github.io

FLATIRON INSTITUTE
Center for Computational Quantum Physics

universität wien

Who am I?

- I received my PhD from Caltech with **John Preskill** and **Steve White** (UCI) in 2018.
- I visited **Frank Verstraete** and **Jutho Haegeman** during my PhD (Vienna, Ghent).
- Thesis on developing tensor network algorithms.
- Associate data scientist at CCQ since 2018.

mtfishman.github.io

FLATIRON INSTITUTE
Center for Computational Quantum Physics

universität wien

Who am I?

- I received my PhD from Caltech with **John Preskill** and **Steve White** (UCI) in 2018.
- I visited **Frank Verstraete** and **Jutho Haegeman** during my PhD (Vienna, Ghent).
- Thesis on developing tensor network algorithms.
- Associate data scientist at CCQ since 2018.
- Develop ITensor with **Miles Stoudenmire** (CCQ).

mtfishman.github.io

FLATIRON INSTITUTE
Center for Computational Quantum Physics

universität wien

Who am I?

- I received my PhD from Caltech with **John Preskill** and **Steve White** (UCI) in 2018.
- I visited **Frank Verstraete** and **Jutho Haegeman** during my PhD (Vienna, Ghent).
- Thesis on developing tensor network algorithms.
- Associate data scientist at CCQ since 2018.
- Develop ITensor with **Miles Stoudenmire** (CCQ).
- Develop PastaQ with **Giacomo Torlai** (AWS).

mtfishman.github.io

What is the Center for Computation Quantum Physics (CCQ)?

- Part of the Flatiron Institute (NYC), funded by the Simons Foundation.

What is the Center for Computation Quantum Physics (CCQ)?

- Part of the Flatiron Institute (NYC), funded by the Simons Foundation.
- FI has 5 research centers: math, biology, astrophysics, neuroscience, and quantum.

What is the Center for Computation Quantum Physics (CCQ)?

- Part of the Flatiron Institute (NYC), funded by the Simons Foundation.
- FI has 5 research centers: math, biology, astrophysics, neuroscience, and quantum.
- Supports computational research: computers clusters, software development, algorithm development, and scientific applications.

What is the Center for Computation Quantum Physics (CCQ)?

- Part of the Flatiron Institute (NYC), funded by the Simons Foundation.
- FI has 5 research centers: math, biology, astrophysics, neuroscience, and quantum.
- Supports computational research: computers clusters, software development, algorithm development, and scientific applications.
- Reach out to us for software needs (new features, bugs), algorithm development, high performance computing, etc.

What is the Center for Computation Quantum Physics (CCQ)?

- Part of the Flatiron Institute (NYC), funded by the Simons Foundation.
- FI has 5 research centers: math, biology, astrophysics, neuroscience, and quantum.
- Supports computational research: computers clusters, software development, algorithm development, and scientific applications.
- Reach out to us for software needs (new features, bugs), algorithm development, high performance computing, etc.
- CCQ has expertise in QMC, quantum embedding (DMFT), tensor networks, quantum dynamics, machine learning, quantum computing, quantum chemistry, etc.

What is the Center for Computation Quantum Physics (CCQ)?

- Part of the Flatiron Institute (NYC), funded by the Simons Foundation.
- FI has 5 research centers: math, biology, astrophysics, neuroscience, and quantum.
- Supports computational research: computers clusters, software development, algorithm development, and scientific applications.
- Reach out to us for software needs (new features, bugs), algorithm development, high performance computing, etc.
- CCQ has expertise in QMC, quantum embedding (DMFT), tensor networks, quantum dynamics, machine learning, quantum computing, quantum chemistry, etc.
- We are hiring postdocs, full-time scientists, part-time and full-time software developers, interns, etc.

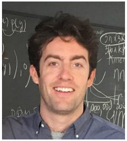
What is the Center for Computation Quantum Physics (CCQ)?

FLATIRON
INSTITUTE
Center for Computational
Quantum Physics

SIMONS FOUNDATION

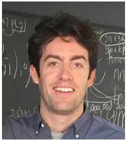
What is ITensor?

- Stands for "Intelligent Tensor".



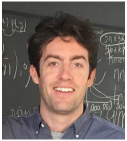
What is ITensor?

- Stands for "Intelligent Tensor".
- Started by **Steve White** of UCI (inventor of the DMRG algorithm).



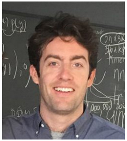
What is ITensor?

- Stands for "Intelligent Tensor".
- Started by **Steve White** of UCI (inventor of the DMRG algorithm).
- Development taken over by **Miles Stoudenmire** (CCQ) around 2011.



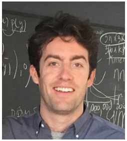
What is ITensor?

- Stands for “Intelligent Tensor”.
- Started by **Steve White** of UCI (inventor of the DMRG algorithm).
- Development taken over by **Miles Stoudenmire** (CCQ) around 2011.
- I took over development in 2018, still co-developed with Miles.



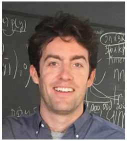
What is ITensor?

- Stands for "Intelligent Tensor".
- Started by **Steve White** of UCI (inventor of the DMRG algorithm).
- Development taken over by **Miles Stoudenmire** (CCQ) around 2011.
- I took over development in 2018, still co-developed with Miles.
- Originally written in C++. I led the port to Julia starting in 2019.



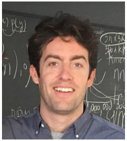
What is ITensor?

- Stands for "Intelligent Tensor".
- Started by **Steve White** of UCI (inventor of the DMRG algorithm).
- Development taken over by **Miles Stoudenmire** (CCQ) around 2011.
- I took over development in 2018, still co-developed with Miles.
- Originally written in C++. I led the port to Julia starting in 2019.
- **Katie Hyatt** (AWS) wrote the GPU backend in Julia.



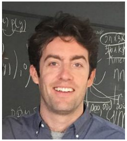
What is ITensor?

- Stands for "Intelligent Tensor".
- Started by **Steve White** of UCI (inventor of the DMRG algorithm).
- Development taken over by **Miles Stoudenmire** (CCQ) around 2011.
- I took over development in 2018, still co-developed with Miles.
- Originally written in C++. I led the port to Julia starting in 2019.
- **Katie Hyatt** (AWS) wrote the GPU backend in Julia.
- Website: itensor.org.



What is ITensor?

- Stands for "Intelligent Tensor".
- Started by **Steve White** of UCI (inventor of the DMRG algorithm).
- Development taken over by **Miles Stoudenmire** (CCQ) around 2011.
- I took over development in 2018, still co-developed with Miles.
- Originally written in C++. I led the port to Julia starting in 2019.
- **Katie Hyatt** (AWS) wrote the GPU backend in Julia.
- Website: itensor.org.
- Paper: arxiv.org/abs/2007.14822.



What is Julia?

- Started in 2009 at MIT, v0.1 released in 2013, v1 released in 2018.

What is Julia?

- Started in 2009 at MIT, v0.1 released in 2013, v1 released in 2018.
- Language is very stable, ecosystem is growing very quickly.

What is Julia?

- Started in 2009 at MIT, v0.1 released in 2013, v1 released in 2018.
- Language is very stable, ecosystem is growing very quickly.
- Fast like C/C++, high level features of Python, all in one language.

What is Julia?

- Started in 2009 at MIT, v0.1 released in 2013, v1 released in 2018.
- Language is very stable, ecosystem is growing very quickly.
- Fast like C/C++, high level features of Python, all in one language.
- Just-in-time compiled, garbage collected, package manager, plotting, great numerical libraries, etc.

What is Julia?

- Started in 2009 at MIT, v0.1 released in 2013, v1 released in 2018.
- Language is very stable, ecosystem is growing very quickly.
- Fast like C/C++, high level features of Python, all in one language.
- Just-in-time compiled, garbage collected, package manager, plotting, great numerical libraries, etc.
- Julia is great, you should use it.

What is Julia?

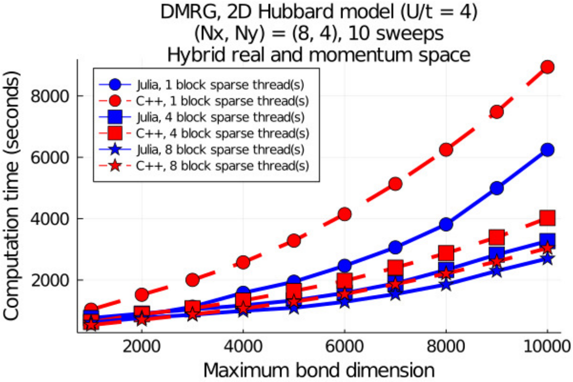
- Started in 2009 at MIT, v0.1 released in 2013, v1 released in 2018.
- Language is very stable, ecosystem is growing very quickly.
- Fast like C/C++, high level features of Python, all in one language.
- Just-in-time compiled, garbage collected, package manager, plotting, great numerical libraries, etc.
- Julia is great, you should use it.
- I could say more, ask me about it.

What is Julia?

- Started in 2009 at MIT, v0.1 released in 2013, v1 released in 2018.
- Language is very stable, ecosystem is growing very quickly.
- Fast like C/C++, high level features of Python, all in one language.
- Just-in-time compiled, garbage collected, package manager, plotting, great numerical libraries, etc.
- Julia is great, you should use it.
- I could say more, ask me about it.
- Find out more at: julialang.org.

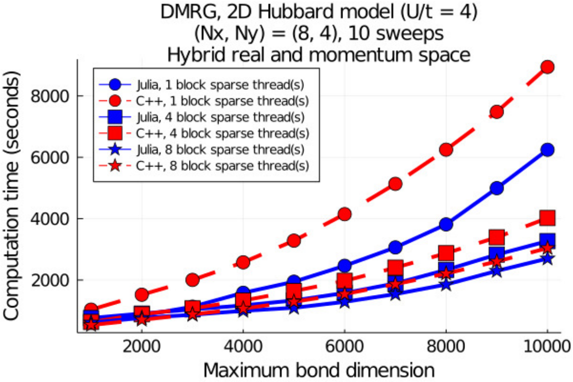
What is Julia?

- Porting the full ITensor library took about $1\frac{1}{2}$ years.



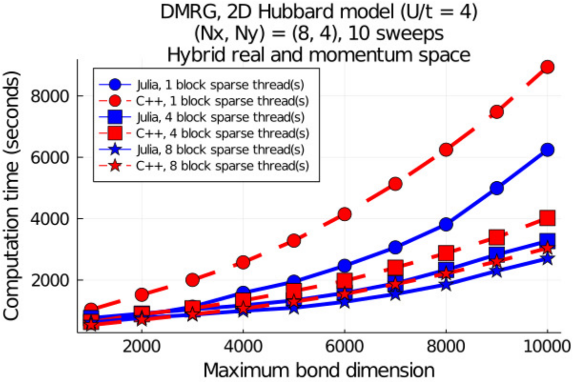
What is Julia?

- Porting the full ITensor library took about $1\frac{1}{2}$ years.
- Much less code, easier development.



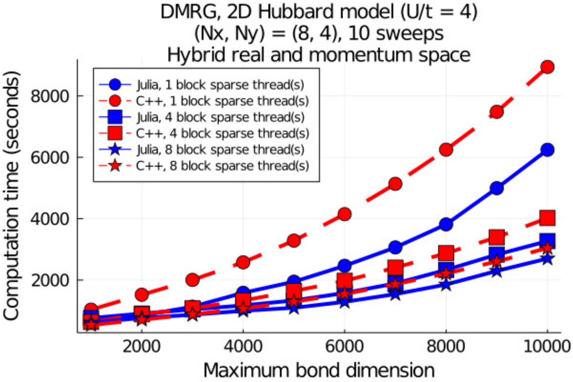
What is Julia?

- Porting the full ITensor library took about $1\frac{1}{2}$ years.
- Much less code, easier development.
- Now that it is ported, we have many more features: GPU, autodiff, infinite MPS, visualization, etc.



What is Julia?

- Porting the full ITensor library took about $1\frac{1}{2}$ years.
- Much less code, easier development.
- Now that it is ported, we have many more features: GPU, autodiff, infinite MPS, visualization, etc.
- Great numerical libraries, code ended up faster.



What is PastaQ?

- Initiated by **Giacomo Torlai** (AWS) while he was a postdoc at CCQ, co-developed by Giacomo and me.

What is PastaQ?

PastaQ

- Initiated by **Giacomo Torlai** (AWS) while he was a postdoc at CCQ, co-developed by Giacomo and me.

  - Quantum computing extension to ITensor in Julia.

What is PastaQ?

PastaQ

- Initiated by **Giacomo Torlai** (AWS) while he was a postdoc at CCQ, co-developed by Giacomo and me.

  - Quantum computing extension to ITensor in Julia.
    - Tensor network-based quantum state and process tomography.

What is PastaQ?

- Initiated by **Giacomo Torlai** (AWS) while he was a postdoc at CCQ, co-developed by Giacomo and me.

- Quantum computing extension to ITensor in Julia.
  - Tensor network-based quantum state and process tomography.
  - Extensive and extendable gate definitions.

What is PastaQ?

PastaQ

- Initiated by **Giacomo Torlai** (AWS) while he was a postdoc at CCQ, co-developed by Giacomo and me.

- Quantum computing extension to ITensor in Julia.
  - Tensor network-based quantum state and process tomography.
  - Extensive and extendable gate definitions.
  - Built-in and easily extendable circuit definitions.

What is PastaQ?

PastaQ

- Initiated by **Giacomo Torlai** (AWS) while he was a postdoc at CCQ, co-developed by Giacomo and me.

  - Quantum computing extension to ITensor in Julia.
    - Tensor network-based quantum state and process tomography.
    - Extensive and extendable gate definitions.
    - Built-in and easily extendable circuit definitions.
    - Approximate circuit evolution and optimization with MPS/MPO, etc.

What is PastaQ?

- Initiated by **Giacomo Torlai** (AWS) while he was a postdoc at CCQ, co-developed by Giacomo and me.

  - Quantum computing extension to ITensor in Julia.
    - Tensor network-based quantum state and process tomography.
    - Extensive and extendable gate definitions.
    - Built-in and easily extendable circuit definitions.
    - Approximate circuit evolution and optimization with MPS/MPO, etc.
  - Find out more: github.com/GTorlai/PastaQ.jl

When should I use tensor networks?

- Good for many sites or qubits but limited to shorter times/circuit depths (though better with noise).

When should I use tensor networks?

- Good for many sites or qubits but limited to shorter times/circuit depths (though better with noise).
- Locally gate structure or interactions matching the graph structure of the tensor network.

When should I use tensor networks?

- Good for many sites or qubits but limited to shorter times/circuit depths (though better with noise).
- Locally gate structure or interactions matching the graph structure of the tensor network.
- If you need very long time evolution/many layers but few qubits, use full state simulation.

When should I use tensor networks?

- Good for many sites or qubits but limited to shorter times/circuit depths (though better with noise).
- Locally gate structure or interactions matching the graph structure of the tensor network.
- If you need very long time evolution/many layers but few qubits, use full state simulation.
  - ITensor and PastaQ handle this seamlessly.

When should I use tensor networks?

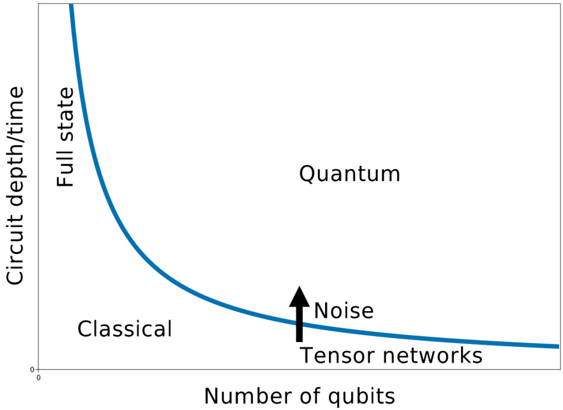
- Good for many sites or qubits but limited to shorter times/circuit depths (though better with noise).
- Locally gate structure or interactions matching the graph structure of the tensor network.
- If you need very long time evolution/many layers but few qubits, use full state simulation.
  - ITensor and PastaQ handle this seamlessly.
  - This is not the focus of ITensor and PastaQ at the moment, specialized libraries like Yao.jl may be faster.

When should I use tensor networks?

- Good for many sites or qubits but limited to shorter times/circuit depths (though better with noise).
- Locally gate structure or interactions matching the graph structure of the tensor network.
- If you need very long time evolution/many layers but few qubits, use full state simulation.
  - ITensor and PastaQ handle this seamlessly.
  - This is not the focus of ITensor and PastaQ at the moment, specialized libraries like Yao.jl may be faster.
- Tensor networks are a common, general language for reasoning about quantum many-body systems (for example, quantum circuits).

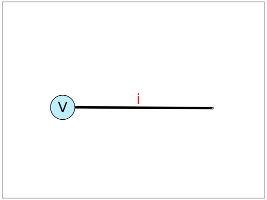
When should I use tensor networks?

- In my opinion, tensor networks are the best general purpose tool we have right now for studying quantum many-body systems (while we wait for a general purpose quantum computer).

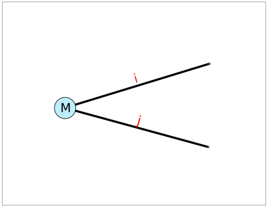


What are tensor networks?

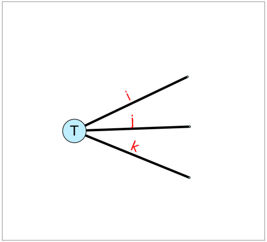
Vector V_i
Order-1 tensor



Matrix M_{ij}
Order-2 tensor



Tensor T_{ijk}
Order-3 tensor



[TODO: Add examples of matvec, matmul, trace]

How do I install ITensor/PastaQ?

1. Download Julia: julialang.org/downloads
2. Launch Julia:

```
$ julia
```

3. Type the following commands:

```
julia> using Pkg

julia> Pkg.add("ITensors")
[...]

julia> Pkg.add("PastaQ")
[...]
```

How do I install ITensor/PastaQ?

Now you can use ITensors and PastaQ:

```julia
julia> using ITensors

julia> i = Index(2);

julia> A = ITensor(i);

julia> using PastaQ

julia> gates = [("X", 1), ("CX", (1, 3))];

julia> psi = runcircuit(gates);
```

Tutorial: One-site state basics

```julia
using ITensors

i = Index(2)


```

```
# Load ITensor

# 2−dimensional labeled
# Hilbert space
# (dim=2|id=510)
```

Tutorial: One-site state basics

```julia
using ITensors          # Load ITensor

i = Index(2)            # 2-dimensional labeled
                        # Hilbert space
                        # (dim=2|id=510)
```

```julia
Zp = ITensor(i)         # Z|Z+⟩ = |Z+⟩
Zp[i=>1] = 1

Zp = ITensor([1, 0], i) # Construct from a Vector
```

Tutorial: One-site state basics

```
Zp = ITensor([1, 0], i)
Zm = ITensor([0, 1], i)
Xp = ITensor([1, 1]/√2, i)
Xm = ITensor([1, -1]/√2, i)
```

```
# Z|Z+⟩ = |Z+⟩
# Z|Z−⟩ = −|Z−⟩
# X|X+⟩ = |X+⟩
# X|X−⟩ = −|X−⟩
```

Tutorial: One-site state basics

```
Zp = ITensor([1, 0], i)
Zm = ITensor([0, 1], i)
Xp = ITensor([1, 1]/√2, i)
Xm = ITensor([1, -1]/√2, i)
```

```
# Z|Z+⟩ = |Z+⟩
# Z|Z−⟩ = −|Z−⟩
# X|X+⟩ = |X+⟩
# X|X−⟩ = −|X−⟩
```

```
(Zp + Zm)/√2
(dag(Zp) * Xp)
(dag(Zp) * Xp)[]
inner(Zp, Xp)
norm(Xp)
```

```
# ≈ Xp
# ≈ ITensor(1/√2)
# ≈ 1/√2
# ≈ 1/√2
# ≈ 1
```

Tutorial: One-site state basics

```julia
using ITensorVisualizationBase: set_backend!
```

Tutorial: One-site state basics

```julia
using ITensorVisualizationBase: set_backend!
```

```julia
back = "UnicodePlots"
set_backend!(back)

@visualize dag(Zp) * Xp
```

```
# TODO: Add
    UnicodePlots
    visualization.
```

Tutorial: One-site state basics

```julia
using ITensorVisualizationBase: set_backend!
```

```julia
back = "UnicodePlots"
set_backend!(back)

@visualize dag(Zp) * Xp
```

```julia
# TODO: Add
    UnicodePlots
    visualization.
```

```julia
back = "Makie"
set_backend!(back)

@visualize dag(Zp) * Xp
```

[TODO: Add GLMakie visualization.]

Tutorial: One-site state basics

```
i = Index(2, "S=1/2")
```

```
# “S=1/2” defines an
# operator basis
#
# Additionally:
# “Qubit”, “Qudit”,
# “Electron”, ...
```

Tutorial: One-site state basics

```
i = Index(2, "S=1/2")
```

```
# "S=1/2" defines an
# operator basis
#
# Additionally:
# "Qubit", "Qudit",
# "Electron", ...
```

```
Zp = state("Z+", i)
Zm = state("Z-", i)
Xp = state("X+", i)
Xm = state("X-", i)
```

```
# ITensor([1 0], i)
# ITensor([0 1], i)
# (Zp + Zm)/√2
# (Zp − Zm)/√2
```

Tutorial: Custom one-site states

```julia
import ITensors: state


function state(
  ::StateName"iX-",
  ::SiteType"S=1/2"
)
  return [im -im]/√2
end
```

```
# Overload ITensors.jl
# behavior

# Define a state with the
# name “iX−”
```

Tutorial: Custom one-site states

```julia
import ITensors: state


function state(
  ::StateName"iX-",
  ::SiteType"S=1/2"
)
  return [im -im]/√2
end
```

```julia
# Overload ITensors.jl
# behavior

# Define a state with the
# name "iX−"
```

```julia
iXm = state("iX-", i)

inner(Zp, iXm)
inner(Zm, iXm)
```

```julia
# ≈ im ∗ Xm

# ≈ im/√2
# ≈ −im/√2
```

Tutorial: Priming

```
i = Index(2)
j = Index(2)

i == j
```

```
# (dim=2|id=837)
# (dim=2|id=899)

# false
```

Tutorial: Priming

```
i = Index(2)
j = Index(2)

i == j
```

```
# (dim=2|id=837)
# (dim=2|id=899)

# false
```

```
i

prime(i)
i'
i == i'
noprime(i')
```

```
# (dim=2|id=837)

# (dim=2|id=837)'
# (dim=2|id=837)'
# false
# (dim=2|id=837)
```

Tutorial: One-site operators

```
Z = ITensor(i', i)
Z[i'=>1, i=>1] = 1
Z[i'=>2, i=>2] = -1
```

```
# TODO: Diagram
# Set elements
```

Tutorial: One-site operators

```
Z = ITensor(i', i)
Z[i'=>1, i=>1] = 1
Z[i'=>2, i=>2] = -1
```

```
# TODO: Diagram
# Set elements
```

```
z = [
  1 0
  0 -1
]

Z = ITensor(z, i', dag(i))

Z = op("Z", i)
```

```
# Matrix representation
# of Z


# Convert to ITensor

# Use predefined definition
```

Tutorial: One-site operators

```
Z = op("Z", i)
X = op("X", i)

Zp = state("Z+", i)
Zm = state("Z-", i)
```

```
# Z
# X

# |Z+⟩
# |Z−⟩
```

Tutorial: One-site operators

```
Z = op("Z", i)
X = op("X", i)

Zp = state("Z+", i)
Zm = state("Z-", i)
```

```
# Z
# X

# |Z+⟩
# |Z−⟩
```

```
XZp = X * Zp
XZp == Zm
XZp == Zm'
noprime(XZp) == Zm
```

```
# X|Z+⟩ = |Z−⟩
# false
# true
# true
```

Tutorial: One-site operators

[TODO: Add visualization of inner product]

```
XZp = X * Zp

inner(Zm, XZp)

inner(Zm', XZp)
inner(Zp', XZp)
```

```
# X|Z+⟩ = |Z−⟩

# error: not a scalar value

# ≈ 1
# ≈ 0
```

Tutorial: One-site operators

[TODO: Add visualization of inner product]

```
XZp = X * Zp

inner(Zm, XZp)

inner(Zm', XZp)
inner(Zp', XZp)
```

```
# X|Z+⟩ = |Z−⟩

# error: not a scalar value

# ≈ 1
# ≈ 0
```

```
(dag(Zm)' * X * Zp)[]
inner(Zm', X, Zp)

XZp = apply(X, Zp)
inner(Zm, XZp)
```

```
# ≈ 1
# ≈ 1

# = noprime(X * Zp)
# ≈ 1
```

Tutorial: Custom one-site operators

```julia
import ITensors: op

function op(
  ::OpName"iX",
  ::SiteType"S=1/2"
)
  return [
    0 im
    im 0
  ]
end
```

```
# Overload ITensors.jl
# behavior
```

Tutorial: Custom one-site operators

```julia
import ITensors: op

function op(
  ::OpName"iX",
  ::SiteType"S=1/2"
)
  return [
    0 im
    im 0
  ]
end
```

```
# Overload ITensors.jl
# behavior
```

```julia
op("iX", i)
```

```
# im * X
```

Tutorial: Two-site states

```
i1 = Index(2, "S=1/2")
i2 = Index(2, "S=1/2")

i1 == i2

ZpZm = ITensor(i1, i2)
ZpZm[i1=>1, i2=>2] = 1
```

```
# (dim=2|id=505|“S=1/2”)
# (dim=2|id=576|“S=1/2”)

# false

# |Z+⟩₁|Z−⟩₂ = |Z+Z−⟩
```

Tutorial: Two-site states

```
i1 = Index(2, "S=1/2")
i2 = Index(2, "S=1/2")

i1 == i2

ZpZm = ITensor(i1, i2)
ZpZm[i1=>1, i2=>2] = 1
```

```
# (dim=2|id=505|“S=1/2”)
# (dim=2|id=576|“S=1/2”)

# false

# |Z+⟩₁|Z−⟩₂ = |Z+Z−⟩
```

```
Zp1 = state("Z+", i1)
Zp2 = state("Z+", i2)

Zm1 = state("Z-", i1)
Zm2 = state("Z-", i2)

ZpZm = Zp1 * Zm2
ZmZp = Zm1 * Zp2
```

```
# |Z+⟩₁
# |Z+⟩₂

# |Z−⟩₁
# |Z−⟩₂

# |Z+Z−⟩ = |Z+⟩₁|Z−⟩₂
# |Z−Z+⟩ = |Z−⟩₁|Z+⟩₂
```

Tutorial: Two-site states

[TODO: Add visualization of inner(Cat, Cat), SVD]

```
Cat = ITensor(i1, i2)
Cat[i1=>1, i2=>2] = 1/√2
Cat[i1=>2, i2=>1] = 1/√2

Cat = (Zp1 * Zm2 +
            Zm1 * Zp2)/√2
```

```
# (|Z+⟩|Z−⟩ + |Z−⟩|Z+⟩)/√2




# From single−site states
```

Tutorial: Two-site states

[TODO: Add visualization of inner(Cat, Cat), SVD]

```
Cat = ITensor(i1, i2)                 # (|Z+⟩|Z−⟩ + |Z−⟩|Z+⟩)/√2
Cat[i1=>1, i2=>2] = 1/√2
Cat[i1=>2, i2=>1] = 1/√2

Cat = (Zp1 * Zm2 +                    # From single−site states
       Zm1 * Zp2)/√2
```

```
inner(Cat, Cat)                       # ≈ 1
inner(ZpZm, Cat)                      # ≈ 1/√2

U, S, V = svd(ZmZp, i1)
s = diag(S)                           # ≈ [1, 0]

U, S, V = svd(Cat, i1)
s = diag(S)                           # ≈ [1/√2, 1/√2]
```

Tutorial: Two-site operators

[TODO: Add visualization of H]

```
H = ITensor(i1', i2', i1, i2)
H[i1'=>2, i2'=>1,
  i1=>2, i2=>1] = -1
# ...
```

```
# Make a Hamiltonian:
# Transverse field Ising
# n=2 sites
# H = -\sum_j^{n-1} Z_j Z_{j+1} + h \sum_j^n X_j
```

Tutorial: Two-site operators

[TODO: Add visualization of H]

```
H = ITensor(i1', i2', i1, i2)
H[i1'=>2, i2'=>1,
  i1=>2, i2=>1] = -1
# ...
```

```
# Make a Hamiltonian:
# Transverse field Ising
# n=2 sites
# $H = -\sum_j^{n-1} Z_j Z_{j+1} + h \sum_j^n X_j$
```

```
Id1 = op("Id", i1)
Z1 = op("Z", i1)
X1 = op("X", i1)
# ...

ZZ = Z1 * Z2
XI = X1 * Id2
IX = Id1 * X2

h = 0.5
H = -ZZ + h * (XI + IX)
```

```
# Alternative:
# Build from single−site
# operators.
# Less error−prone.
```

Tutorial: Two-site operators

[TODO: Add visualization of <H>]

```
ZpZp = Zp1 * Zp2



(dag(ZpZp)' * H * ZpZp)[]
inner(ZpZp', H, ZpZp)
inner(ZpZp, apply(H, ZpZp))
```

```
# Expectation value:
# ⟨H⟩ = ⟨Z+Z+|H|Z+Z+⟩



# ≈ −1
```

Tutorial: Two-site operators

[TODO: Add visualization of <H>]

```
ZpZp = Zp1 * Zp2
(dag(ZpZp)' * H * ZpZp)[]
inner(ZpZp', H, ZpZp)
inner(ZpZp, apply(H, ZpZp))
```

```
# Expectation value:
# ⟨H⟩ = ⟨Z+Z+|H|Z+Z+⟩
# ≈ −1
```

```
D, U = eigen(H)
diag(D)
```

```
# ≈ [−√2, −1, 1, √2]
```

Tutorial: Custom two-site operators

```julia
import ITensors: op

function op(
  ::OpName"CRy",
  ::SiteType"S=1/2";
  θ
)
  c = cos(θ/2)
  s = sin(θ/2)
  return [
    1 0 0  0
    0 1 0  0
    0 0 c -s
    0 0 s  c
  ]
end
```

```
# Controlled−Ry (CRy)
# rotation gate

# CRy(θ)
```

Tutorial: Custom two-site operators

[TODO: Add visualization of CH|ZpZm>]

```
CH = op("CRy", i1, i2; θ=π/2)
```

```
# Controlled−Hadamard gate
# CH = CRy(θ=π/2)
```

Tutorial: Custom two-site operators

[TODO: Add visualization of CH|ZpZm>]

```
CH = op("CRy", i1, i2;
             θ=π/2)
```

```
# Controlled−Hadamard gate
# CH = CRy(θ=π/2)
```

```
ZpZm = Zp1 * Zm2

CH_Xm = apply(CH, ZpZm)

CH_Xm ≈ Zp1 * Xm2
```

```
# |Z+Z−⟩ = |Z+⟩₁|Z−⟩₂

# CH|Z+Z−⟩ = |Z+X−⟩

# true
```

Tutorial: Two-site state optimization

[TODO: Add visualization of minimizing <v|H|v>]

```
function E(ψ)
  ψHψ = inner(ψ', H, ψ)
  ψψ = inner(ψ, ψ)
  return ψHψ / ψψ
end
```

```
# Function to minimize:
# Expectation value of the
# energy.
#
# E(ψ) = ⟨ψ|H|ψ⟩/⟨ψ|ψ⟩
```

Tutorial: Two-site state optimization

[TODO: Add visualization of minimizing <v|H|v>]

```
function E(ψ)
  ψHψ = inner(ψ', H, ψ)
  ψψ = inner(ψ, ψ)
  return ψHψ / ψψ
end
```

```
# Function to minimize:
# Expectation value of the
# energy.
#
# E(ψ) = ⟨ψ|H|ψ⟩/⟨ψ|ψ⟩
```

```
function minimize(f, ∂f, x;
  nsteps, γ)
  for n in 1:nsteps
    x = x - γ * ∂f(x)
  end
  return x
end
```

```
# Simple gradient descent.
# Must provide function f(x)
# to minimize and ∂f(x),
# the gradient of f at x.

# γ is the gradient
# descent step size.
```

Tutorial: Two-site state optimization

[TODO: Add visualization of minimizing <v|H|v>]

```
ψ0 = (Zp1 * Zm2 +                          # |ψ0⟩ = (|Z+Z+⟩ +
        Zm1 * Zp2)/√2                      #            |Z−Z−⟩)/√2

E(ψ0)                                      # ≈ −1

using Zygote: gradient                     # Using Zygote for automatic
∂E(ψ) = gradient(E, ψ)[1]                  # differentation of the energy.

norm(∂E(ψ0))                               # ≈ 2
```

Tutorial: Two-site state optimization

[TODO: Add visualization of minimizing <v|H|v>]

```julia
ψ0 = (Zp1 * Zm2 +
      Zm1 * Zp2)/√2

E(ψ0)

using Zygote: gradient
∂E(ψ) = gradient(E, ψ)[1]

norm(∂E(ψ0))
```

```julia
# |ψ0⟩ = (|Z+Z+⟩ +
#          |Z−Z−⟩)/√2

# ≈ −1

# Using Zygote for automatic
# differentation of the energy.

# ≈ 2
```

```julia
ψ = minimize(E, ∂E, ψ0;
       nsteps=10, γ=0.1)

E(ψ0), norm(∂E(ψ0))
E(ψ), norm(∂E(ψ))

```

```julia
# Minimize over ψ:
# E(ψ) = ⟨ψ|H|ψ⟩/⟨ψ|ψ⟩

# (−1, 2)
# (−1.4142131, 0.0010865277)
#         ≈ (−√2, 0)
```

Tutorial: Two-site circuit optimization

[TODO: Add visualization of U]

```
# Circuit as a vector of gates:
U(θ, i1, i2) = [
  op("Ry", i1; θ=θ[1]),
  op("Ry", i2; θ=θ[2]),
  op("CX", i1, i2),
  op("Ry", i1; θ=θ[3]),
  op("Ry", i2; θ=θ[4]),
]
```

```
# PastaQ notation:
u(θ, j1, j2) = [
  ("Ry", j1, (; θ=θ[1])),
  ("Ry", j2, (; θ=θ[2])),
  ("CNOT", j1, j2),
  ("Ry", j1, (; θ=θ[3])),
  ("Ry", j2, (; θ=θ[4])),
]
U(θ, i1, i2) =
  buildcircuit(u(θ, 1, 2), [i1, i2])
```

Tutorial: Two-site circuit optimization

[TODO: Add visualization of minimizing <0|UHU|0>]

```julia
ψ₀ = Zp1 * Zp2


function E(θ)
  ψθ = apply(U(θ), ψ₀)
  return inner(ψθ', H, ψθ)
end
```

```
# References state:
# |0⟩ = |Z+Z+⟩

# Find θ that minimizes:
# E(θ) = ⟨0|U(θ)† H U(θ)|0⟩
#      = ⟨θ|H|θ⟩
```

Tutorial: Two-site circuit optimization

[TODO: Add visualization of minimizing <0|UHU|0>]

```
ψ₀ = Zp1 * Zp2


function E(θ)
  ψθ = apply(U(θ), ψ₀)
  return inner(ψθ', H, ψθ)
end
```

```
# References state:
# |0⟩ = |Z+Z+⟩

# Find θ that minimizes:
# E(θ) = ⟨0|U(θ)† H U(θ)|0⟩
#      = ⟨θ|H|θ⟩
```

```
θ₀ = [0, 0, 0, 0]
θ = minimize(E, ∂E, θ₀;
        nsteps=40, γ=0.5)

E(θ₀), norm(∂E(θ₀))
E(θ), norm(∂E(θ))
```

```
# (−1, √3/2)
# (−1.4142077, 0.0017584116)
#       ≈ (−√2, 0)
```

Tutorial: Two-site fidelity optimization

[TODO: Add visualization of minimizing <v|U|v0>]

```
ψ0 = Zp1 * Zp2
ψ = (ZpZp + ZmZm) / √2



function F(θ)
  ψθ = apply(U(θ, i1, i2), ψ0)
  return -abs(inner(ψ, ψθ))^2
end
```

```
# Reference state:
# |0⟩ = |Z+Z+⟩

# Target state:
# |ψ⟩ = (|Z+Z+⟩ +
#        |Z−Z−⟩)/√2

# Find θ that minimizes:
# F(θ) = −|⟨ψ|U(θ)|0⟩|^2
```

Tutorial: Two-site fidelity optimization

[TODO: Add visualization of minimizing <v|U|v0>]

```
ψ₀ = Zp1 ∗ Zp2
ψ = (ZpZp + ZmZm) / √2



function F(θ)
  ψθ = apply(U(θ, i1, i2), ψ₀)
  return -abs(inner(ψ, ψθ))^2
end
```

```
# Reference state:
# |0⟩ = |Z+Z+⟩

# Target state:
# |ψ⟩ = (|Z+Z+⟩ +
#           |Z−Z−⟩)/√2

# Find θ that minimizes:
# F(θ) = −|⟨ψ|U(θ)|0⟩|^2
```

```
θ₀ = [0, 0, 0, 0]
θ = minimize(F, ∂F, θ₀;
        nsteps=50, γ=0.1)

F(θ₀), norm(∂F(θ₀))
F(θ), norm(∂F(θ))
```

```
# ≈ (−0.5, 0.5)
# ≈ (−0.9938992, 0.07786879)
```

Tutorial: n-site states with MPS

[TODO: Add visualization of <Zp|Cat>]

```
n = 30
i = [Index(2, "S=1/2")
        for j in 1:n]

Zp = MPS(i, "Z+")
Zm = MPS(i, "Z-")

Cat = (Zp + Zm)/√2

```

```
# n−site state



# |Z+Z+...Z+⟩
# |Z−Z−...Z−⟩

# (|Z+Z+...Z+⟩ +
#  |Z−Z−...Z−⟩)/√2
```

Tutorial: n-site states with MPS

[TODO: Add visualization of <Zp|Cat>]

```julia
n = 30                         # n−site state
i = [Index(2, "S=1/2")
      for j in 1:n]

Zp = MPS(i, "Z+")              # |Z+Z+...Z+⟩
Zm = MPS(i, "Z-")              # |Z−Z−...Z−⟩

Cat = (Zp + Zm)/√2             # (|Z+Z+...Z+⟩ +
                               #  |Z−Z−...Z−⟩)/√2
```

```julia
maxlinkdim(Zp)                 # 1 (product state)
maxlinkdim(Cat)                # 2 (entangled state)
inner(Zp, Zp)                  # ⟨Z+|Z+⟩ ≈ 1
inner(Zm, Zp)                  # ⟨Z−|Z+⟩ ≈ 0
norm(Cat)                      # ((⟨Z+| + ⟨Z−|)(|Z+⟩ + |Z+⟩))/2
                               #       ≈ 1
```

Tutorial: n-site states with MPS

[TODO: Add visualization of minimizing <psi|H|psi>]

```
j = n ÷ 2
X_j = op("X", i[j])

X_jZp = apply(X_j, Zp)


state = [k == j ? "Z-" : "Z+"
         for k in 1:n]
X_jZp = MPS(i, state)
```

```
# j = n ÷ 2
# X_j

# X_j|Z+Z+...Z+⟩ =
#     |Z+Z+...Z−...Z+⟩

# |Z+Z+...Z−...Z+⟩
```

Tutorial: n-site states with MPS

[TODO: Add visualization of minimizing <psi|H|psi>]

```julia
j = n ÷ 2
X_j = op("X", i[j])

X_jZp = apply(X_j, Zp)


state = [k == j ? "Z-" : "Z+"
         for k in 1:n]
X_jZp = MPS(i, state)
```

```
# j = n ÷ 2
# X_j

# X_j|Z+Z+...Z+⟩ =
#     |Z+Z+...Z−...Z+⟩

# |Z+Z+...Z−...Z+⟩
```

```julia
maxlinkdim(X_jZp)
inner(Zp, X_jZp)
inner(X_jZp, apply(X_j, Zp))
```

```
# 1 (product state)
# ≈ 0
# ≈ 1
```

Tutorial: n-site states with MPS

[TODO: Add visualization of H]

```julia
function ising(n; h)
  H = OpSum()
  for j in 1:(n - 1)
    H -= "Z", j, "Z", j + 1
  end
  for j in 1:n
    H += h, "X", j
  end
  return H
end
```

n sites

$H = -\sum_j^{n-1} Z_j Z_{j+1} + h \sum_j^n X_j$

Tutorial: n-site states with MPS

[TODO: Add visualization of H]

```julia
function ising(n; h)
  H = OpSum()
  for j in 1:(n - 1)
    H -= "Z", j, "Z", j + 1
  end
  for j in 1:n
    H += h, "X", j
  end
  return H
end
```

n sites

$H = -\sum_j^{n-1} Z_j Z_{j+1} + h \sum_j^n X_j$

```julia
h = 0.5
H = MPO(ising(n; h=h), i)
maxlinkdim(H)
Zp = MPS(i, "Z+")
inner(Zp', H, Zp)
```

= 3 (local Hamiltonian)

⟨Z+Z+...Z+|H|Z+Z+...Z+⟩

Tutorial: n-site states with MPS

[TODO: Add visualization of H]

```
ψ0 = MPS(i, "Z+")                          # |0⟩ = |Z+Z+...Z+⟩

ψ = minimize(E, ∂E, ψ0;                    # Minimize over ψ:
        nsteps=50, γ=0.1,                  # ⟨ψ|H|ψ⟩/⟨ψ|ψ⟩
        maxdim=10, cutoff=1e-5)

E_dmrg, ψ_dmrg = dmrg(H, ψ0;               # DMRG solves:
        nsweeps=10,                        # H|ψ⟩ ≈ |ψ⟩
        maxdim=10, cutoff=1e-5)
```

Tutorial: n-site states with MPS

[TODO: Add visualization of H]

```
ψ0 = MPS(i, "Z+")                              # |0⟩ = |Z+Z+...Z+⟩

ψ = minimize(E, ∂E, ψ0;                        # Minimize over ψ:
        nsteps=50, γ=0.1,                      # ⟨ψ|H|ψ⟩/⟨ψ|ψ⟩
        maxdim=10, cutoff=1e-5)

E_dmrg, ψ_dmrg = dmrg(H, ψ0;                   # DMRG solves:
        nsweeps=10,                            # H|ψ⟩ ≈ |ψ⟩
        maxdim=10, cutoff=1e-5)
```

```
maxlinkdim(ψ0)                                 # = 1 (product state)
maxlinkdim(ψ)                                  # = 3 (entangled state)
maxlinkdim(ψ_dmrg)                             # = 2 (entangled state)
E(ψ0), norm( ∂(ψ0))                            # ≈ (−29, 5.4772256)
E(ψ), norm(∂E(ψ))                              # ≈ (−31.0317917, 0.06673780)
E(ψ_dmrg), norm(∂E(ψ_dmrg))                    # ≈ (−31.0356110, 0.02413237)
```

Tutorial: n-site states with MPS

[TODO: Add visualization of H]

```
Ry_layer(θ, i) = [op("Ry", i[j]; θ=θ[j]) for j in 1:n]
CX_layer(i) = [op("CX", i[j], i[j+1]) for j in 1:2:(n-1)]
```

Tutorial: n-site states with MPS

[TODO: Add visualization of H]

```
Ry_layer(θ, i) = [op("Ry", i[j]; θ=θ[j]) for j in 1:n]
CX_layer(i) = [op("CX", i[j], i[j+1]) for j in 1:2:(n-1)]
```

```
function U(θ, i; nlayers)
  n = length(i)
  U_θ = Ry_layer(θ[1:n], i)
  for l in 1:(nlayers - 1)
    θ_l = θ[(1:n) .+ l * n]
    U_θ = [U_θ; CX_layer(i)]
    U_θ = [U_θ; Ry_layer(θ_l, i)]
  end
  return U_θ
end
```

```
# U(θ) = Ry_1(θ_1) ... Ry_n(θ_n)
#        CX_{1,2} ... CX_{n-1,n}
#        Ry_1(θ_{n+1}) ... Ry_n(θ_{2n})
#        ...
```

Tutorial: n-site states with MPS

[TODO: Add visualization of H]

```julia
ψ_0 = MPS(i, "Z+")
nlayers = 6
function E(θ)
  ψ_θ = apply(U(θ, i; nlayers), ψ_0)
  return inner(ψ_θ', H, ψ_θ)
end

θ_0 = zeros(nlayers * n)
θ = minimize(E, ∂E, θ_0;
        nsteps=20, γ=0.1)
```

```
# |0⟩ = |Z+Z+...Z+⟩

# Minimize over θ:
# |θ⟩ = U(θ)|0⟩
# E(θ) = ⟨θ|H|θ⟩
```

Tutorial: n-site states with MPS

[TODO: Add visualization of H]

```
ψ₀ = MPS(i, "Z+")
nlayers = 6
function E(θ)
  ψθ = apply(U(θ, i; nlayers), ψ₀)
  return inner(ψθ', H, ψθ)
end

θ₀ = zeros(nlayers * n)
θ = minimize(E, ∂E, θ₀;
        nsteps=20, γ=0.1)
```

```
# |0⟩ = |Z+Z+...Z+⟩

# Minimize over θ:
# |θ⟩ = U(θ)|0⟩
# E(θ) = ⟨θ|H|θ⟩
```

```
maxlinkdim(ψ₀)
maxlinkdim(ψθ)
E(θ₀), norm(∂E(θ₀))
E(θ), norm(∂E(θ))
```

```
# 1 (product state)
# 2 (entangled state)
# (−29, 5.773335)
# (−31.017062, 0.000759)
```

Tutorial: n-site states with MPS

[TODO: Add visualization of H]

```julia
ψ_0 = MPS(i, "Z+")
nlayers = 6
function F(θ)
  ψ_θ = apply(U(θ, i; nlayers), ψ_0)
  return -abs(inner(ψ', ψ_θ))^2
end

θ_0 = zeros(nlayers * n)
θ = minimize(F, ∂F, θ_0;
        nsteps=20, γ=0.1)
```

```
# |0⟩ = |Z+Z+...Z+⟩

# Minimize over θ:
# F(θ) = −|⟨ψ|U(θ)|0⟩|^2
#      = −|⟨ψ|θ⟩|2
```

Tutorial: n-site states with MPS

[TODO: Add visualization of H]

```
ψ0 = MPS(i, "Z+")
nlayers = 6
function F(θ)
  ψθ = apply(U(θ, i; nlayers), ψ0)
  return -abs(inner(ψ', ψθ))^2
end

θ0 = zeros(nlayers * n)
θ = minimize(F, ∂F, θ0;
        nsteps=20, γ=0.1)
```

```
# |0⟩ = |Z+Z+...Z+⟩

# Minimize over θ:
# F(θ) = −|⟨ψ|U(θ)|0⟩|^2
#      = −|⟨ψ|θ⟩|2
```

```
maxlinkdim(ψ0)
maxlinkdim(ψθ)
F(θ0), norm(∂F(θ0))
F(θ), norm(∂F(θ))
```

```
# 1 (product state)
# 2 (entangled state)
# (−0.556066, 0.895717)
# (−0.995230, 0.048939)
```

Future directions

- More AD, make ITensor fully differentiable (have some work to do, like tensor decompositions and general network contractions, more MPS/MPO functions. You will find bugs!).

Future directions

- More AD, make ITensor fully differentiable (have some work to do, like tensor decompositions and general network contractions, more MPS/MPO functions. You will find bugs!).
- More general built-in tensor networks beyond MPS/MPO

Future directions

- More AD, make ITensor fully differentiable (have some work to do, like tensor decompositions and general network contractions, more MPS/MPO functions. You will find bugs!).
- More general built-in tensor networks beyond MPS/MPO
- More HPC with multithreaded and multiprocessor parallelism and GPUs

Future directions

- More AD, make ITensor fully differentiable (have some work to do, like tensor decompositions and general network contractions, more MPS/MPO functions. You will find bugs!).
- More general built-in tensor networks beyond MPS/MPO
- More HPC with multithreaded and multiprocessor parallelism and GPUs
- Many ongoing projects and directions: quantum chemistry (for example UCC), real space parallel DMRG, TDVP, and TEBD, MPO compression tools, general approximate contraction techniques for unstructured networks, contracting and optimizing general tensor networks with AD, infinite MPS and tensor network tools like VUMPS and TDVP, trying out different network topologies for noisy circuit tomography, simulation and optimization.

Future directions

- More AD, make ITensor fully differentiable (have some work to do, like tensor decompositions and general network contractions, more MPS/MPO functions. You will find bugs!).
- More general built-in tensor networks beyond MPS/MPO
- More HPC with multithreaded and multiprocessor parallelism and GPUs
- Many ongoing projects and directions: quantum chemistry (for example UCC), real space parallel DMRG, TDVP, and TEBD, MPO compression tools, general approximate contraction techniques for unstructured networks, contracting and optimizing general tensor networks with AD, infinite MPS and tensor network tools like VUMPS and TDVP, trying out different network topologies for noisy circuit tomography, simulation and optimization.
- Building general tools, looking for people with problems and code to contribute.

Future directions

- More AD, make ITensor fully differentiable (have some work to do, like tensor decompositions and general network contractions, more MPS/MPO functions. You will find bugs!).
- More general built-in tensor networks beyond MPS/MPO
- More HPC with multithreaded and multiprocessor parallelism and GPUs
- Many ongoing projects and directions: quantum chemistry (for example UCC), real space parallel DMRG, TDVP, and TEBD, MPO compression tools, general approximate contraction techniques for unstructured networks, contracting and optimizing general tensor networks with AD, infinite MPS and tensor network tools like VUMPS and TDVP, trying out different network topologies for noisy circuit tomography, simulation and optimization.
- Building general tools, looking for people with problems and code to contribute.

Future directions

- More AD, make ITensor fully differentiable (have some work to do, like tensor decompositions and general network contractions, more MPS/MPO functions. You will find bugs!).
- More general built-in tensor networks beyond MPS/MPO
- More HPC with multithreaded and multiprocessor parallelism and GPUs
- Many ongoing projects and directions: quantum chemistry (for example UCC), real space parallel DMRG, TDVP, and TEBD, MPO compression tools, general approximate contraction techniques for unstructured networks, contracting and optimizing general tensor networks with AD, infinite MPS and tensor network tools like VUMPS and TDVP, trying out different network topologies for noisy circuit tomography, simulation and optimization.
- Building general tools, looking for people with problems and code to contribute.