Analyzing Quantum Many-Body Systems with ITensor and PastaQ

Matthew Fishman
Center for Computational Quantum Physics (CCQ)
Flatiron Institute, NY
mtfishman.github.io
github.com/mtfishman/ITensorTutorials.jl

February 28, 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

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).

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

FLATIRON INSTITUTE
Center for Computational Quantum Physics

universität wien

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)?

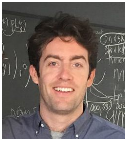
FLATIRON INSTITUTE

Center for Computational Quantum Physics

SIMONS FOUNDATION

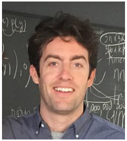
What is ITensor?

- Stands for “Intelligent Tensor”.



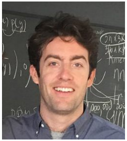
What is ITensor?

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



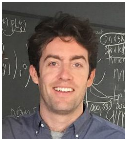
What is ITensor?

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



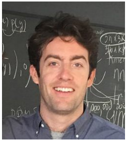
What is ITensor?

- Stands for “Intelligent Tensor”.
- Started by **Steve White** of UCI (inventor of the DMRG algorithm).
- **Miles Stoudenmire** (CCQ) started developing ITensor around 2011.
- I started working on ITensor in 2018, still co-developed with Miles.



What is ITensor?

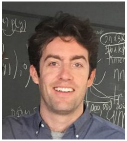
- Stands for "Intelligent Tensor".
- Started by **Steve White** of UCI (inventor of the DMRG algorithm).
- **Miles Stoudenmire** (CCQ) started developing ITensor around 2011.
- I started working on ITensor in 2018, still co-developed with Miles.
- Originally written in C++. I led the port to Julia starting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



What is ITensor?

- Stands for “Intelligent Tensor”.
- Started by **Steve White** of UCI (inventor of the DMRG algorithm).
- **Miles Stoudenmire** (CCQ) started developing ITensor around 2011.
- I started working on ITensor 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.

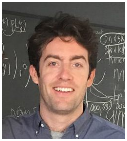
ITENSOR



julia

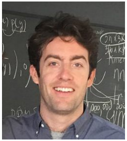
What is ITensor?

- Stands for “Intelligent Tensor”.
- Started by **Steve White** of UCI (inventor of the DMRG algorithm).
- **Miles Stoudenmire** (CCQ) started developing ITensor around 2011.
- I started working on ITensor 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.
- Used in over **400** research papers.



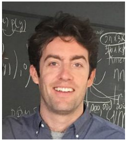
What is ITensor?

- Stands for “Intelligent Tensor”.
- Started by **Steve White** of UCI (inventor of the DMRG algorithm).
- **Miles Stoudenmire** (CCQ) started developing ITensor around 2011.
- I started working on ITensor 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.
- Used in over **400** research papers.
- Website: itensor.org.



What is ITensor?

- Stands for "Intelligent Tensor".
- Started by **Steve White** of UCI (inventor of the DMRG algorithm).
- **Miles Stoudenmire** (CCQ) started developing ITensor around 2011.
- I started working on ITensor 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.
- Used in over **400** research papers.
- 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?

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?

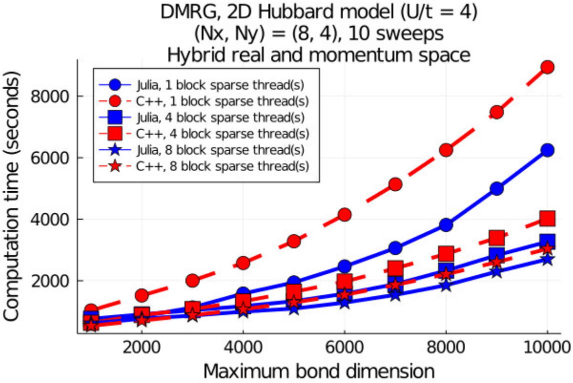
- Porting the full ITensor library took about $1\frac{1}{2}$ years.
- Much less code, easier development.

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?

- 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?

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?

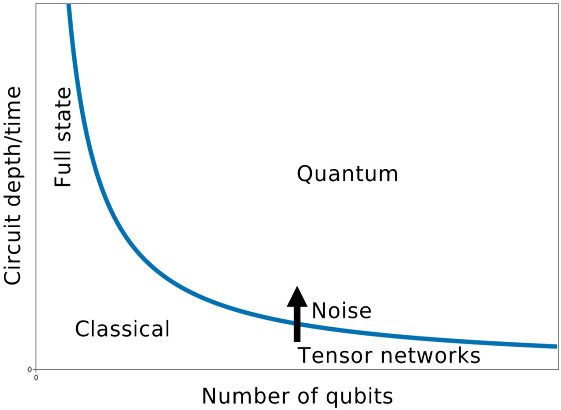
- Good for many sites or qubits but limited to shorter times/circuit depths (though better with noise).
- Locally gate structure or interactions matching the graph structure of the tensor network.
- If you need very long time evolution/many layers but few qubits, use full state simulation.
  - ITensor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

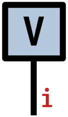
When should I use tensor networks?

- 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).

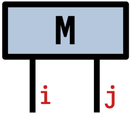


What are tensor networks?

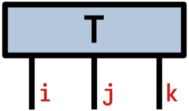
Vector V_i
Order-1 tensor



Matrix M_{ij}
Order-2 tensor

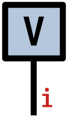


Tensor T_{ijk}
Order-3 tensor

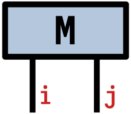


What are tensor networks?

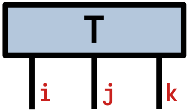
Vector V_i
Order-1 tensor



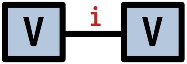
Matrix M_{ij}
Order-2 tensor



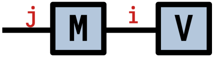
Tensor T_{ijk}
Order-3 tensor



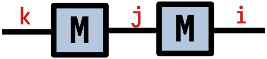
Vector inner product



Matrix-vector multiplication

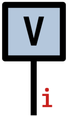


Matrix-matrix multiplication

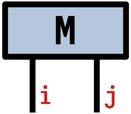


What are tensor networks?

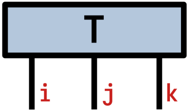
Vector V_i
Order-1 tensor



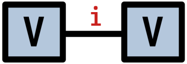
Matrix M_{ij}
Order-2 tensor



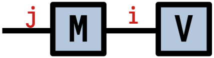
Tensor T_{ijk}
Order-3 tensor



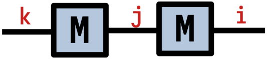
Vector inner product



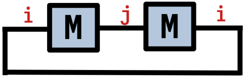
Matrix-vector multiplication



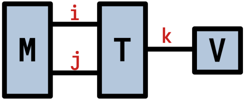
Matrix-matrix multiplication



Trace of matrix-matrix multiplication



General tensor network



How do I install ITensor/PastaQ?

1. Download Julia: julialang.org/downloads

How do I install ITensor/PastaQ?

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

```
$ julia
```

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> 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

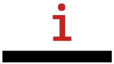
```
using ITensors

i = Index(2)
```

Load ITensor

2-dimensional labeled Hilbert space (dim=2|id=510)

Tutorial: One-site state basics

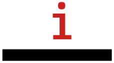
```
using ITensors

i = Index(2)
```



Tutorial: One-site state basics

```
using ITensors

i = Index(2)


```

```
Zp = ITensor(i)
Zp[i=>1] = 1

Zp = ITensor([1, 0], i)
```



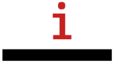
$$|Z+\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

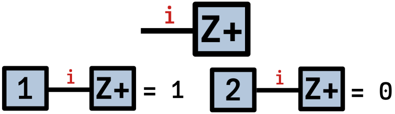
Construct from a Vector

Tutorial: One-site state basics

```julia
using ITensors

i = Index(2)


```

```julia
Zp = ITensor(i)
Zp[i=>1] = 1

Zp = ITensor([1, 0], i)
```





Tutorial: One-site state basics

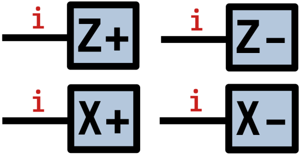
```
Zp = ITensor([1, 0], i)
Zm = ITensor([0, 1], i)
Xp = ITensor([1, 1]/√2, i)
Xm = ITensor([1, -1]/√2, i)
```

$$|Z+\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \; |Z-\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$|X+\rangle = \begin{bmatrix} 1 \\ 1 \end{bmatrix} / \sqrt{2}, \; |X-\rangle = \begin{bmatrix} 1 \\ -1 \end{bmatrix} / \sqrt{2}$$

Tutorial: One-site state basics

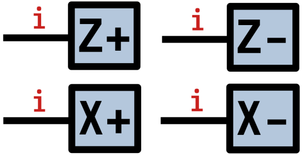
```
Zp = ITensor([1, 0], i)
Zm = ITensor([0, 1], i)
Xp = ITensor([1, 1]/√2, i)
Xm = ITensor([1, -1]/√2, i)
```



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)
```

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



$\approx$ Xp

$\approx$ ITensor($1/\sqrt{2}$)

$\approx 1/\sqrt{2}$

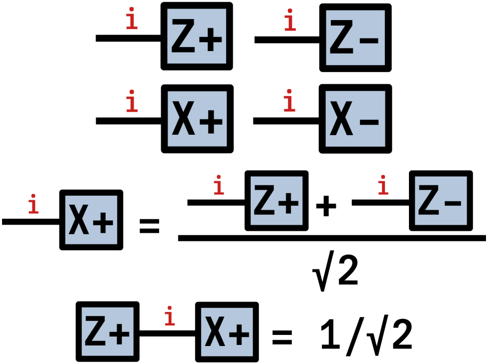
$\approx 1/\sqrt{2}$

≈ 1

Tutorial: One-site state basics

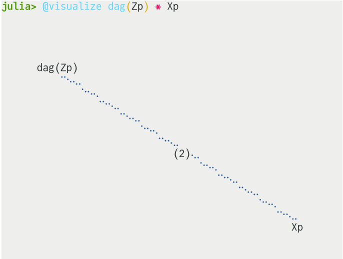
```
Zp = ITensor([1, 0], i)
Zm = ITensor([0, 1], i)
Xp = ITensor([1, 1]/√2, i)
Xm = ITensor([1, -1]/√2, i)
```

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



Tutorial: One-site state basics

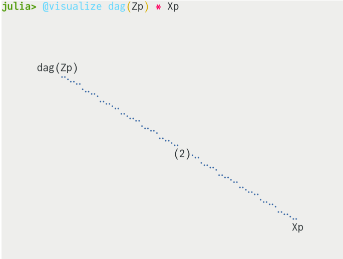
```
using ITensorUnicodePlots

@visualize dag(Zp) * Xp
```



```
julia> @visualize dag(Zp) * Xp
```

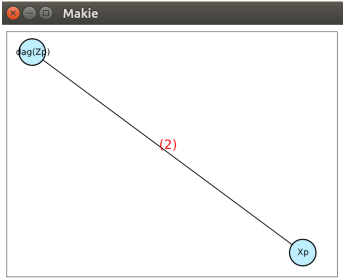
Tutorial: One-site state basics

```julia
using ITensorUnicodePlots

@visualize dag(Zp) * Xp
```



```
julia> @visualize dag(Zp) * Xp
```

```julia
using ITensorGLMakie

@visualize dag(Zp) * Xp
```



Tutorial: One-site states

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

“S=1/2” defines an operator basis

Additionally:
“Qubit”, “Qudit”,
“Electron”, . . .

Tutorial: One-site states

```
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

```
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-”

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

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

≈ im * Xm

≈ im/√2

≈ -im/√2

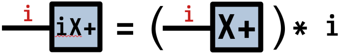
Tutorial: Custom one-site states

```
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-”

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

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



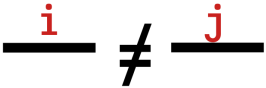
Tutorial: Priming

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

i == j
```

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

false
```

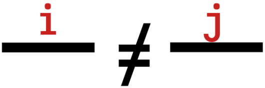
Tutorial: Priming

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

i == j
```



Tutorial: Priming

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

i == j
```



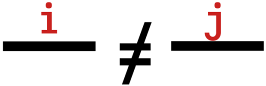
```
i = Index(2)

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

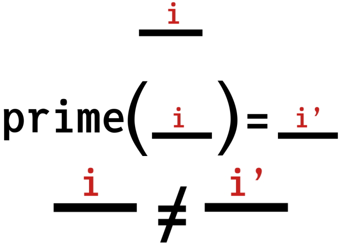
```
(dim=2|id=837)

(dim=2|id=837)'
(dim=2|id=837)'
true
true
```

Tutorial: Priming

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

i == j
```

```
i = Index(2)

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





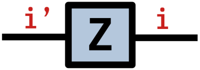
Tutorial: One-site operators

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

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

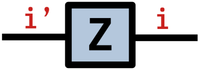
Tutorial: One-site operators

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



Tutorial: One-site operators

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



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

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

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

Matrix representation 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

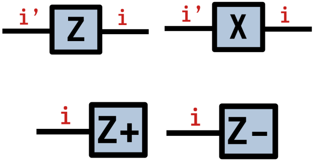
X

$|Z+\rangle$

$|Z\text{-}\rangle$

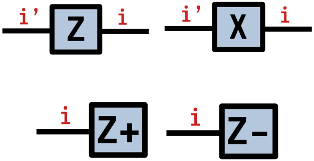
Tutorial: One-site operators

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

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



Tutorial: One-site operators

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

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



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

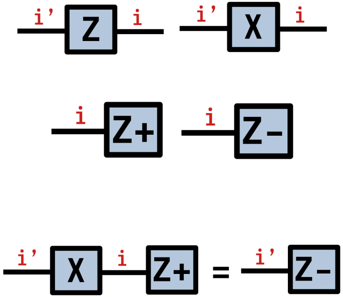
$X|Z+\rangle = |Z\text{-}\rangle$

```
false
true
true
```

Tutorial: One-site operators

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

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

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



Tutorial: One-site operators

```
XZp = X * Zp

dag(Zm) * XZp

dag(Zm') * XZp
dag(Zp') * XZp

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

$$\mathrm{X}|\mathrm{Z}+\rangle = |\mathrm{Z}\text{-}\rangle$$

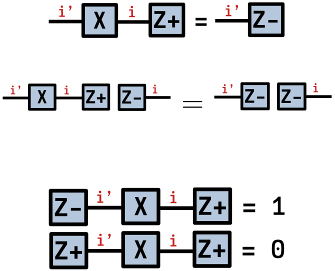
$$|\mathrm{Z}-\rangle\langle \mathrm{Z}+|X$$

$$\langle Z-|X|Z+\rangle \approx 1$$
$$\langle Z+|X|Z+\rangle \approx 0$$

$$\langle Z-|X|Z+\rangle \approx 1$$
$$\langle Z-|X|Z+\rangle \approx 1$$

Tutorial: One-site operators

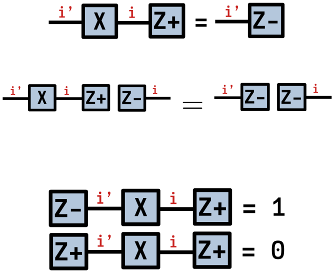
```
XZp = X * Zp

dag(Zm) * XZp

dag(Zm') * XZp
dag(Zp') * XZp

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



Tutorial: One-site operators

```
XZp = X * Zp

dag(Zm) * XZp

dag(Zm') * XZp
dag(Zp') * XZp

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

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



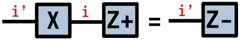
$= \text{noprime}(X * Zp)$

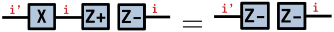
≈ 1

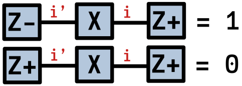
Tutorial: One-site operators

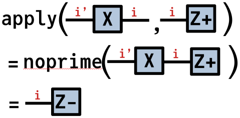
```
XZp = X ∗ Zp

dag(Zm) ∗ XZp

dag(Zm’) ∗ XZp
dag(Zp’) ∗ XZp

dag(Zm)’ ∗ X ∗ Zp
inner(Zm’, X, Zp)
```

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









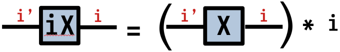
Tutorial: Custom one-site operators

```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

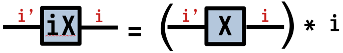
```julia
import ITensors: op

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



Tutorial: Custom one-site operators

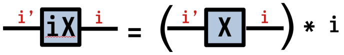
```
import ITensors: op

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

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



Tutorial: Custom one-site operators

```julia
import ITensors: op

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



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

```julia
X * im
```

Tutorial: Two-site states

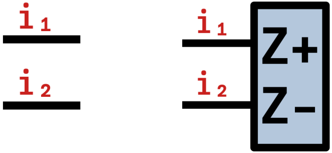
```
i1 = Index(2, "S=1/2")
i2 = Index(2, "S=1/2")

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
```

$$|\text{Z+}\rangle_1|\text{Z-}\rangle_2 = |\text{Z+Z-}\rangle$$

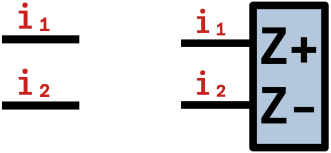
Tutorial: Two-site states

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

i1 == i2

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



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
```



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

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

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

$|\text{Z+}\rangle_1$

$|\text{Z+}\rangle_2$

$|\text{Z-}\rangle_1$

$|\text{Z-}\rangle_2$

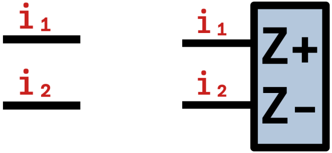
$$|\text{Z+Z-}\rangle = |\text{Z+}\rangle_1 |\text{Z-}\rangle_2$$

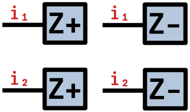
$$|\text{Z-Z+}\rangle = |\text{Z-}\rangle_1 |\text{Z+}\rangle_2$$

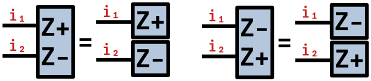
Tutorial: Two-site states

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

i1 == i2

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

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

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

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







Tutorial: Two-site states

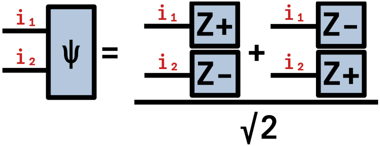
```
ψ = ITensor(i1, i2)
ψ[i1=>1, i2=>2] = 1/√2
ψ[i1=>2, i2=>1] = 1/√2

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

$$(|\text{Z+}\rangle|\text{Z-}\rangle + |\text{Z-}\rangle|\text{Z+}\rangle)/\sqrt{2}$$

From single-site states

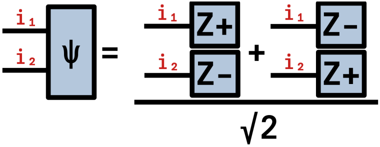
Tutorial: Two-site states

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

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



Tutorial: Two-site states

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

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



```
inner(ψ, ψ)
inner(ZpZm, ψ)

U, S, V = svd(ZmZp, i1)
s = diag(S)

U, S, V = svd(ψ, i1)
s = diag(S)
```

≈ 1

$\approx 1/\sqrt{2}$

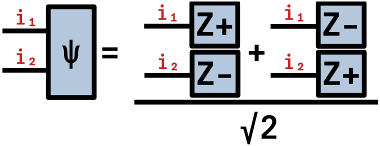
$\approx [1,\ 0]$

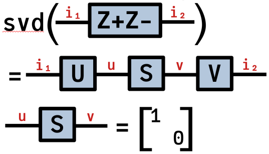
$\approx [1/\sqrt{2},\ 1/\sqrt{2}]$

Tutorial: Two-site states

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

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

```
inner(ψ, ψ)
inner(ZpZm, ψ)

U, S, V = svd(ZmZp, i1)
s = diag(S)

U, S, V = svd(ψ, i1)
s = diag(S)
```

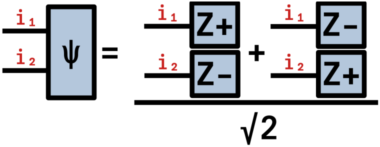


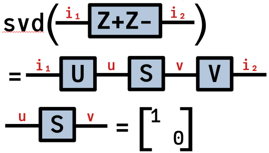


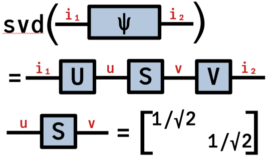
Tutorial: Two-site states

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

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

```
inner(ψ, ψ)
inner(ZpZm, ψ)

U, S, V = svd(ZmZp, i1)
s = diag(S)

U, S, V = svd(ψ, i1)
s = diag(S)
```







Tutorial: Two-site operators

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

Make a Hamiltonian:
Transverse field Ising ($n = 2$)

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

Tutorial: Two-site operators

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

Make a Hamiltonian:
Transverse field Ising ($n = 2$)

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

```
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.)

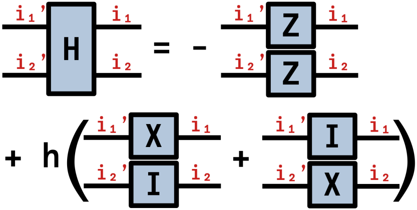
Tutorial: Two-site operators

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

```
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)
```

Make a Hamiltonian:
Transverse field Ising ($n = 2$)

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



Tutorial: Two-site operators

```
ZpZp = Zp1 * Zp2



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

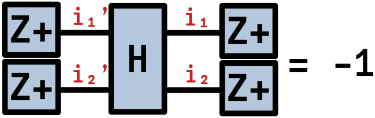
Expectation value:

$\langle \mathrm{H} \rangle = \langle \mathrm{Z{+}Z{+}} | \mathrm{H} | \mathrm{Z{+}Z{+}} \rangle$

$\approx$ -1

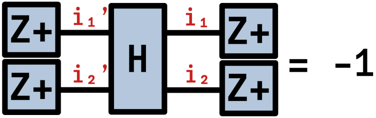
Tutorial: Two-site operators

```
ZpZp = Zp1 * Zp2



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



Tutorial: Two-site operators

```
ZpZp = Zp1 * Zp2



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



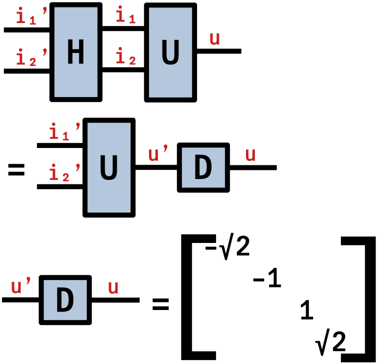
```
D, U = eigen(H)
diag(D)
```

$\approx [-\sqrt{2}, -1, 1, \sqrt{2}]$

Tutorial: Two-site operators

```
ZpZp = Zp1 * Zp2



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

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



Tutorial: Custom two-site operators

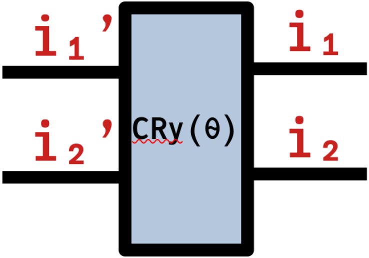
```
import ITensors: op

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

```
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
```



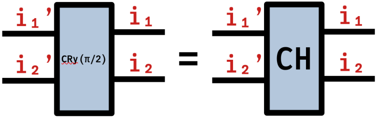
Tutorial: Custom two-site operators

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

Controlled-Hadamard gate
CH = CRy(θ=$\pi/2$)

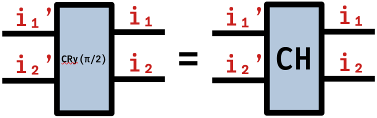
Tutorial: Custom two-site operators

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



Tutorial: Custom two-site operators

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



```
ZpZm = Zp1 * Zm2

CH_Xm = apply(CH, ZpZm)

CH_Xm ≈ Zp1 * Xm2
```

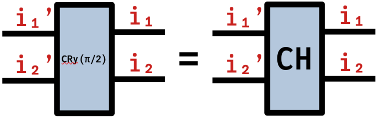
$$|\text{Z+Z-}\rangle = |\text{Z+}\rangle_1 |\text{Z-}\rangle_2$$

$$\text{CH}|\text{Z+Z-}\rangle = |\text{Z+X-}\rangle$$

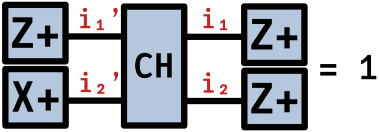
true

Tutorial: Custom two-site operators

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



```
ZpZm = Zp1 * Zm2

CH_Xm = apply(CH, ZpZm)

CH_Xm ≈ Zp1 * Xm2
```



Tutorial: Two-site state optimization

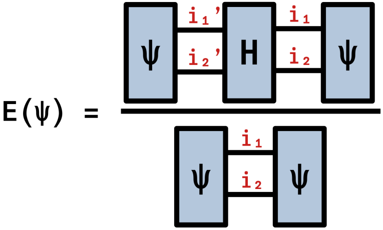
```
function E(ψ)
  ψHψ = inner(ψ', H, ψ)
  ψψ = inner(ψ, ψ)
  return ψHψ / ψψ
end
```

Function to minimize: Expectation value of the energy.

$$\mathrm{E}(\psi) = \langle\psi|\mathrm{H}|\psi\rangle/\langle\psi|\psi\rangle$$

Tutorial: Two-site state optimization

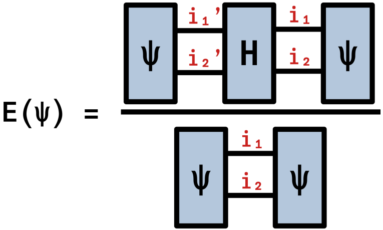
```
function E(ψ)
  ψHψ = inner(ψ’, H, ψ)
  ψψ = inner(ψ, ψ)
  return ψHψ / ψψ
end
```



Tutorial: Two-site state optimization

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

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

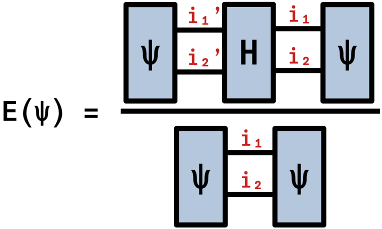


Simple gradient descent. Must provide function f(x) to minimize and ∂f(x), the gradient of f at x.
 γ is the gradient descent step size.

Tutorial: Two-site state optimization

```
function E($\psi$)
  $\psi H \psi$ = inner($\psi$', H, $\psi$)
  $\psi\psi$ = inner($\psi$, $\psi$)
  return $\psi H \psi$ / $\psi\psi$
end
```

```
function minimize(f, $\partial$f, x;
  nsteps, $\gamma$)
  for n in 1:nsteps
    x = x - $\gamma * \partial$f(x)
  end
  return x
end
```



$$\min_{\psi} E(\psi) \qquad \partial_{\psi} E(\psi)$$

Tutorial: Two-site state optimization

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

E(ψ0)

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

norm(∂E(ψ0))
```

$|\psi_0\rangle = (|\mathrm{Z{+}Z{+}}\rangle +$
$|\mathrm{Z{-}Z{-}}\rangle)/\sqrt{2}$

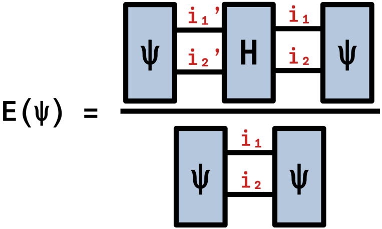
$\approx$ -1

Using Zygote for automatic differentation of the energy.

≈ 2

Tutorial: Two-site state optimization

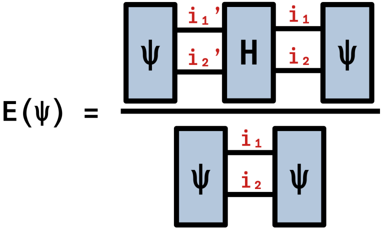
```
ψ₀ = (Zp1 * Zm2 +
        Zm1 * Zp2)/√2

E(ψ₀)

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

norm(∂E(ψ₀))
```



Tutorial: Two-site state optimization

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

E(ψ₀)

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

norm(∂E(ψ₀))
```

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

E(ψ₀), norm(∂E(ψ₀))
E(ψ), norm(∂E(ψ))

```



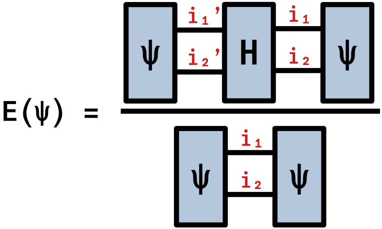
Minimize over ψ:
$E(\psi) = \langle\psi|H|\psi\rangle/\langle\psi|\psi\rangle$

(-1, 2)
(-1.4142131, 0.0010865277)
$\approx$ (-$\sqrt{2}$, 0)

Tutorial: Two-site state optimization

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

E(ψ0)

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

norm(∂E(ψ0))
```

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

E(ψ0), norm(∂E(ψ0))
E(ψ), norm(∂E(ψ))

```



$$\min_{\psi} E(\psi) \qquad \partial_{\psi} E(\psi)$$

Tutorial: Two-site circuit optimization

```
# 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]),
]
```

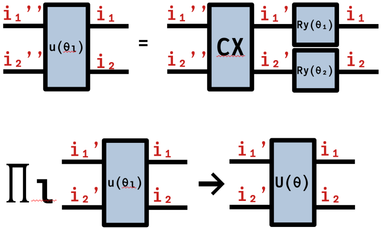
Tutorial: Two-site circuit optimization

```
# 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

```
# 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]),
]
```



Tutorial: Two-site circuit optimization

```
$\psi_0$ = Zp1 * Zp2


function E($\theta$)
  $\psi_\theta$ = apply(U($\theta$), $\psi_0$)
  return inner($\psi_\theta$', H, $\psi_\theta$)
end
```

References state:

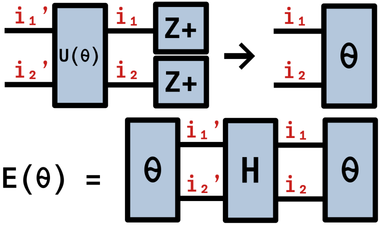
$$|0\rangle = |\mathrm{Z{+}Z{+}}\rangle$$

Find θ that minimizes:

$$\begin{aligned} \mathrm{E}(\theta) &= \langle 0|\mathrm{U}(\theta)^\dagger\ \mathrm{H}\ \mathrm{U}(\theta)|0\rangle \\ &= \langle \theta|\mathrm{H}|\theta\rangle \end{aligned}$$

Tutorial: Two-site circuit optimization

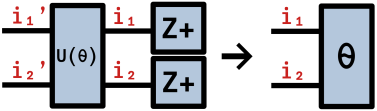
```
ψ0 = Zp1 * Zp2


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



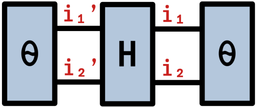
Tutorial: Two-site circuit optimization

```
ψ0 = Zp1 * Zp2


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

```
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))

```



E(θ) =



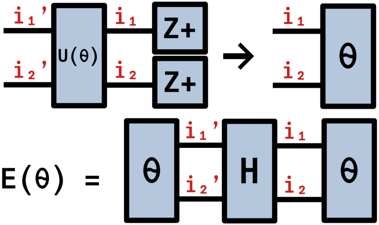
$$\min_{\theta} E(\theta)$$

$$\partial_{\theta} E(\theta)$$

Tutorial: Two-site circuit optimization

```
ψ0 = Zp1 * Zp2


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

```
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))

```



$(\text{-}1,\ \sqrt{3}/2)$

$(\text{-}1.4142077,\ 0.0017584116)$

$\approx (\text{-}\sqrt{2},\ 0)$

Tutorial: Two-site fidelity optimization

```
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:

$|0\rangle = |\mathrm{Z{+}Z{+}}\rangle$

Target state:

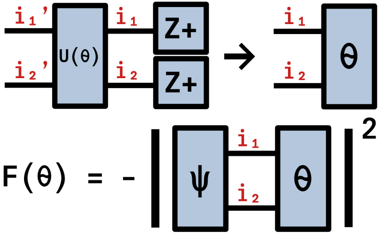
$|\psi\rangle = (|\mathrm{Z{+}Z{+}}\rangle + |\mathrm{Z{-}Z{-}}\rangle)/\sqrt{2}$

Find θ that minimizes:

$\mathrm{F}(\theta) = -|\langle\psi|\mathrm{U}(\theta)|0\rangle|^2$

Tutorial: Two-site fidelity optimization

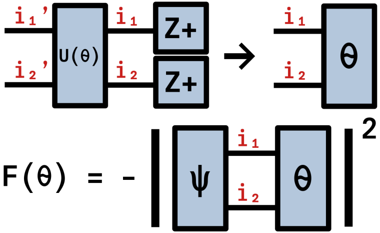
```
ψ0 = Zp1 * Zp2
ψ = (ZpZp + ZmZm) / √2



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



Tutorial: Two-site fidelity optimization

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



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

```
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))

```



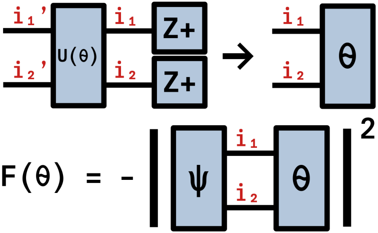
$\min_{\theta} F(\theta)$

$\partial_\theta F(\theta)$

Tutorial: Two-site fidelity optimization

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



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

```
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))

```



$\approx$ (-0.5, 0.5)

$\approx$ (-0.9938992, 0.07786879)

$\approx$ (-1, 0)

Tutorial: n-site states with MPS

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

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

maxlinkdim(Zp)
```

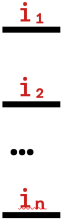
n-site state

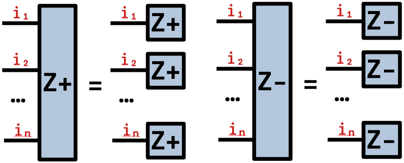
$|Z+Z+\dots Z+\rangle$
$|Z-Z-\dots Z-\rangle$

1 (product state)

Tutorial: n-site states with MPS

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

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

maxlinkdim(Zp)
```





Tutorial: n-site states with MPS

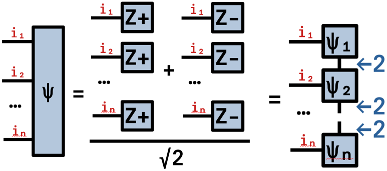
```
ψ = (Zp + Zm)/√2


maxlinkdim(ψ)
```

$$(|Z+Z+\ldots Z+\rangle + \\ |Z-Z-\ldots Z-\rangle)/\sqrt{2}$$

2 (entangled state)

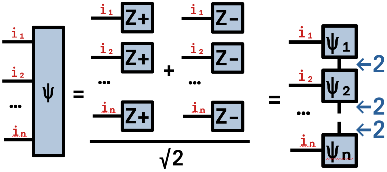
Tutorial: n-site states with MPS

```
ψ = (Zp + Zm)/√2


maxlinkdim(ψ)
```



Tutorial: n-site states with MPS

```
ψ = (Zp + Zm)/√2


maxlinkdim(ψ)
```

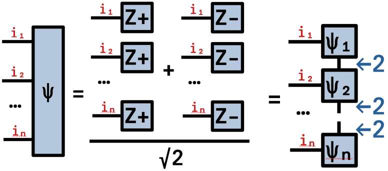


```
   inner(Zp, Zp)
   inner(Zm, Zp)
   norm(ψ)
   inner(Zp, ψ)
```

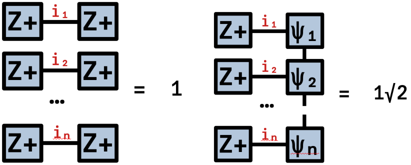
$$\langle Z+|Z+\rangle \approx 1$$
$$\langle Z-|Z+\rangle \approx 0$$
$$||\psi\rangle| \approx 1$$
$$\langle Z+|\psi\rangle \approx 1/\sqrt{2}$$

Tutorial: n-site states with MPS

```
ψ = (Zp + Zm)/√2


maxlinkdim(ψ)
```



```
inner(Zp, Zp)
inner(Zm, Zp)
norm(ψ)
inner(Zp, ψ)
```



Tutorial: n-site states with MPS

```
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)
```

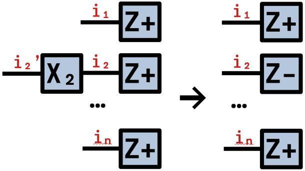
$j = n/2$
X_j

$X_j|Z+Z+\dots Z+\rangle =$
$|Z+Z+\dots Z-$
$\dots Z+\rangle$

$|Z+Z+\dots Z-$
$\dots Z+\rangle$

Tutorial: n-site states with MPS

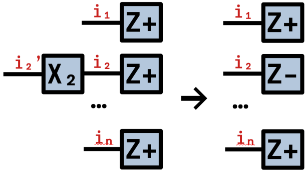
```
j = n ÷ 2
Xⱼ = op("X", i[j])

XⱼZp = apply(Xⱼ, Zp)


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



Tutorial: n-site states with MPS

```
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)
```

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



1 (product state)
≈ 0
≈ 1

Tutorial: n-site operators with MPO

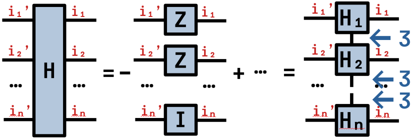
```julia
function ising(n; h)
  H = OpSum()
  for j in 1:(n - 1)
    H -= "Z", j, "Z", j + 1
  end
  for j in 1:n
    H += h, "X", j
  end
  return H
end
```

n sites

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

Tutorial: n-site operators with MPO

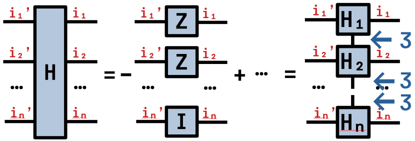
```julia
function ising(n; h)
  H = OpSum()
  for j in 1:(n - 1)
    H -= "Z", j, "Z", j + 1
  end
  for j in 1:n
    H += h, "X", j
  end
  return H
end
```



Tutorial: n-site operators with MPO

```
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
```

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

```

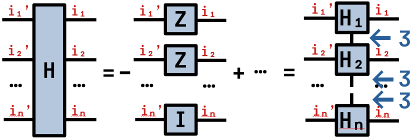


= 3 (local Hamiltonian)

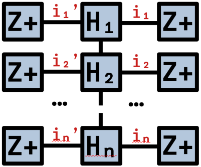
$\langle$Z+Z+...Z+$|$H$|$Z+Z+...Z+$\rangle$
$\approx$ -(n - 1) = -29

Tutorial: n-site operators with MPO

```
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
```



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

```



Tutorial: n-site state optimization

```
$\psi_0$ = MPS(i, "Z+")

$\psi$ = minimize(E, $\partial$E, $\psi_0$;
        nsteps=50, $\gamma$=0.1,
        maxdim=10, cutoff=1e-5)

$E_{dmrg}$, $\psi_{dmrg}$ = dmrg(H, $\psi_0$;
        nsweeps=10,
        maxdim=10, cutoff=1e-5)
```

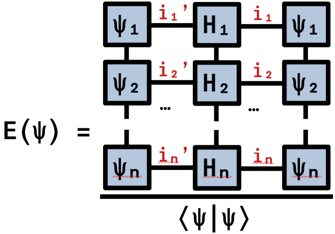
$|0\rangle = |\mathrm{Z+Z+...Z+}\rangle$

Minimize over ψ:
$\langle\psi|\mathrm{H}|\psi\rangle/\langle\psi|\psi\rangle$

DMRG solves:
$\mathrm{H}|\psi\rangle \approx |\psi\rangle$

Tutorial: n-site state optimization

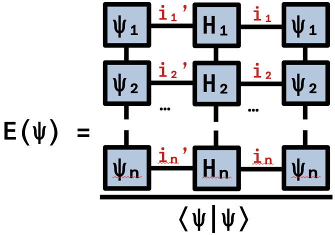
```
ψ0 = MPS(i, "Z+")

ψ = minimize(E, ∂E, ψ0;
        nsteps=50, γ=0.1,
        maxdim=10, cutoff=1e-5)

Edmrg, ψdmrg = dmrg(H, ψ0;
        nsweeps=10,
        maxdim=10, cutoff=1e-5)
```



$$\min_{\psi} E(\psi) \qquad \partial_{\psi} E(\psi)$$

Tutorial: n-site state optimization

```
ψ₀ = MPS(i, "Z+")

ψ = minimize(E, ∂E, ψ₀;
        nsteps=50, γ=0.1,
        maxdim=10, cutoff=1e-5)

E_dmrg, ψ_dmrg = dmrg(H, ψ₀;
        nsweeps=10,
        maxdim=10, cutoff=1e-5)
```



$$\min_{\psi} E(\psi) \qquad \partial_{\psi} E(\psi)$$

```
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 circuit optimization

```
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 circuit optimization

```
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)]
```

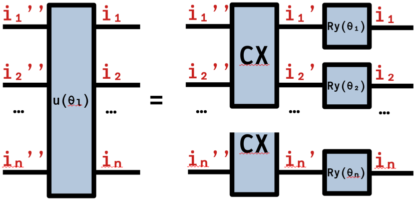
```
function U(θ, i; nlayers)
  n = length(i)
  Uθ = Ry_layer(θ[1:n], i)
  for l in 1:(nlayers - 1)
    θl = θ[(1:n) .+ l * n]
    Uθ = [Uθ; CX_layer(i)]
    Uθ = [Uθ; Ry_layer(θl, i)]
  end
  return Uθ
end
```

$$
\begin{aligned}
U(\theta) = {} & \mathrm{Ry}_1(\theta_1) \ \dots \ \mathrm{Ry}_n(\theta_n) \\
& * \ \mathrm{CX}_{1,2} \ \dots \ \mathrm{CX}_{n-1,n} \\
& * \ \mathrm{Ry}_1(\theta_{n+1}) \ \dots \ \mathrm{Ry}_n(\theta_{2n}) \\
& * \ \dots
\end{aligned}
$$

Tutorial: n-site circuit optimization

```
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
```



$$\prod_l u(\theta_l) \rightarrow U(\theta)$$

Tutorial: n-site states with MPS

```
$\psi_0$ = MPS(i, "Z+")
nlayers = 6
function E($\theta$)
  $\psi_\theta$ = apply(U($\theta$, i; nlayers), $\psi_0$)
  return inner($\psi_\theta$', H, $\psi_\theta$)
end

$\theta_0$ = zeros(nlayers $*$ n)
$\theta$ = minimize(E, $\partial$E, $\theta_0$;
        nsteps=20, $\gamma$=0.1)
```

$$|0\rangle = |\mathrm{Z+Z+...Z+}\rangle$$

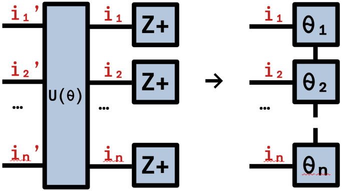
Minimize over θ:

$$|\theta\rangle = \mathrm{U}(\theta)|0\rangle$$

$$\mathrm{E}(\theta) = \langle\theta|\mathrm{H}|\theta\rangle$$

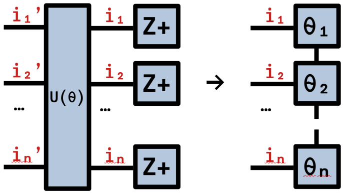
Tutorial: n-site states with MPS

```
$\psi_0$ = MPS(i, "Z+")
nlayers = 6
function E($\theta$)
  $\psi_\theta$ = apply(U($\theta$, i; nlayers), $\psi_0$)
  return inner($\psi_\theta$', H, $\psi_\theta$)
end

$\theta_0$ = zeros(nlayers * n)
$\theta$ = minimize(E, $\partial$E, $\theta_0$;
        nsteps=20, $\gamma$=0.1)
```



Tutorial: n-site states with MPS

```
$\psi_0$ = MPS(i, "Z+")
nlayers = 6
function E($\theta$)
  $\psi_\theta$ = apply(U($\theta$, i; nlayers), $\psi_0$)
  return inner($\psi_\theta$', H, $\psi_\theta$)
end

$\theta_0$ = zeros(nlayers * n)
$\theta$ = minimize(E, ∂E, $\theta_0$;
        nsteps=20, $\gamma$=0.1)
```



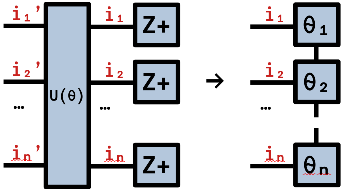
```
maxlinkdim($\psi_0$)
maxlinkdim($\psi_\theta$)
E($\theta_0$), norm(∂E($\theta_0$))
E($\theta$), norm(∂E($\theta$))
```

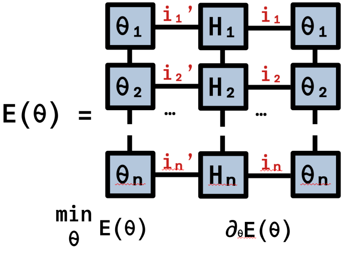
```
1 (product state)
2 (entangled state)
(-29, 5.773335)
(-31.017062, 0.000759)
```

Tutorial: n-site states with MPS

```
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)
```

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





Other features not mentioned

- Noisy state and process simulation (PastaQ). AD/optimization support in progress.

Other features not mentioned

- Noisy state and process simulation (PastaQ).
  AD/optimization support in progress.
- Time evolution with trotterization (ITensor/PastaQ).
  AD/optimization and TDVP support in progress.

Other features not mentioned

- Noisy state and process simulation (PastaQ).
  AD/optimization support in progress.
- Time evolution with trotterization (ITensor/PastaQ).
  AD/optimization and TDVP support in progress.
- Quantum state and process tomograph (PastaQ).

Other features not mentioned

- Noisy state and process simulation (PastaQ).
  AD/optimization support in progress.
- Time evolution with trotterization (ITensor/PastaQ).
  AD/optimization and TDVP support in progress.
- Quantum state and process tomograph (PastaQ).
- Predefined differentiable circuits (PastaQ).

Other features not mentioned

- Noisy state and process simulation (PastaQ). AD/optimization support in progress.
- Time evolution with trotterization (ITensor/PastaQ). AD/optimization and TDVP support in progress.
- Quantum state and process tomograph (PastaQ).
- Predefined differentiable circuits (PastaQ).
- Monitered quantum circuits (PastaQ).

Other features not mentioned

- Noisy state and process simulation (PastaQ). AD/optimization support in progress.
- Time evolution with trotterization (ITensor/PastaQ). AD/optimization and TDVP support in progress.
- Quantum state and process tomograph (PastaQ).
- Predefined differentiable circuits (PastaQ).
- Monitered quantum circuits (PastaQ).
- GPU support (ITensor/PastaQ). Dense only for now, will support block sparse.

Other features not mentioned

- Noisy state and process simulation (PastaQ). AD/optimization support in progress.
- Time evolution with trotterization (ITensor/PastaQ). AD/optimization and TDVP support in progress.
- Quantum state and process tomograph (PastaQ).
- Predefined differentiable circuits (PastaQ).
- Monitered quantum circuits (PastaQ).
- GPU support (ITensor/PastaQ). Dense only for now, will support block sparse.
- Conserved quantities with multithreaded block sparse tensors (ITensor/PastaQ).

Other features not mentioned

- Noisy state and process simulation (PastaQ). AD/optimization support in progress.
- Time evolution with trotterization (ITensor/PastaQ). AD/optimization and TDVP support in progress.
- Quantum state and process tomograph (PastaQ).
- Predefined differentiable circuits (PastaQ).
- Monitered quantum circuits (PastaQ).
- GPU support (ITensor/PastaQ). Dense only for now, will support block sparse.
- Conserved quantities with multithreaded block sparse tensors (ITensor/PastaQ).
- More that I am probably forgetting... Ask Giacomo and me for more details.

Future directions/in progress

- 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, please contribute gradient definitions!).

Future directions/in progress

- 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, please contribute gradient definitions!).
- More general built-in tensor networks beyond MPS/MPO (tree tensor networks and PEPS, general contraction and optimization, use in circuit evolution and tomography, etc.).

Future directions/in progress

- 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, please contribute gradient definitions!).
- More general built-in tensor networks beyond MPS/MPO (tree tensor networks and PEPS, general contraction and optimization, use in circuit evolution and tomography, etc.).
- More HPC with multithreaded and multiprocessor parallelism and GPUs.

Future directions/in progress

- 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, please contribute gradient definitions!).
- More general built-in tensor networks beyond MPS/MPO (tree tensor networks and PEPS, general contraction and optimization, use in circuit evolution and tomography, etc.).
- More HPC with multithreaded and multiprocessor parallelism and GPUs.
- Improved gradient optimization: higher order derivatives, preconditioners, isometrically constrained gradient optimization, etc.

Future directions/in progress

- Free fermions/matchgates and conversion to tensor networks.

Future directions/in progress

- Free fermions/matchgates and conversion to tensor networks.
- Automatic fermions (**Miles Stoudenmire** (CCQ), **Jing Chen** (CCQ, Zapata)).

Future directions/in progress

- Free fermions/matchgates and conversion to tensor networks.
- Automatic fermions (**Miles Stoudenmire** (CCQ), **Jing Chen** (CCQ, Zapata)).
- Non-abelian symmetries (**Miles Stoudenmire**).

Future directions/in progress

- Free fermions/matchgates and conversion to tensor networks.
- Automatic fermions (**Miles Stoudenmire** (CCQ), **Jing Chen** (CCQ, Zapata)).
- Non-abelian symmetries (**Miles Stoudenmire**).
- Quantum chemistry (for example UCC).

Future directions/in progress

- Free fermions/matchgates and conversion to tensor networks.
- Automatic fermions (**Miles Stoudenmire** (CCQ), **Jing Chen** (CCQ, Zapata)).
- Non-abelian symmetries (**Miles Stoudenmire**).
- Quantum chemistry (for example UCC).
- Distributed computing: real space parallel circuit evolution, TDVP, and DMRG, distributed tensors, etc.

Future directions/in progress

- Free fermions/matchgates and conversion to tensor networks.
- Automatic fermions (**Miles Stoudenmire** (CCQ), **Jing Chen** (CCQ, Zapata)).
- Non-abelian symmetries (**Miles Stoudenmire**).
- Quantum chemistry (for example UCC).
- Distributed computing: real space parallel circuit evolution, TDVP, and DMRG, distributed tensors, etc.
- Infinite MPS and tensor network tools like VUMPS and TDVP.

Future directions/in progress

- Free fermions/matchgates and conversion to tensor networks.
- Automatic fermions (**Miles Stoudenmire** (CCQ), **Jing Chen** (CCQ, Zapata)).
- Non-abelian symmetries (**Miles Stoudenmire**).
- Quantum chemistry (for example UCC).
- Distributed computing: real space parallel circuit evolution, TDVP, and DMRG, distributed tensors, etc.
- Infinite MPS and tensor network tools like VUMPS and TDVP.
- Improved MPO compression for time evolving operators.

Future directions/in progress

- Free fermions/matchgates and conversion to tensor networks.
- Automatic fermions (**Miles Stoudenmire** (CCQ), **Jing Chen** (CCQ, Zapata)).
- Non-abelian symmetries (**Miles Stoudenmire**).
- Quantum chemistry (for example UCC).
- Distributed computing: real space parallel circuit evolution, TDVP, and DMRG, distributed tensors, etc.
- Infinite MPS and tensor network tools like VUMPS and TDVP.
- Improved MPO compression for time evolving operators.
- I'm interested in building general tools and algorithms, and I'm looking for people with problems to solve. Also, we need help, code to contributions are welcome!