# ANALYZING QUANTUM MANY-BODY SYSTEMS WITH ITENSOR AND PASTAQ

MATTHEW FISHMAN

CENTER FOR COMPUTATIONAL QUANTUM PHYSICS (CCQ), FLATIRON INSTITUTE, NY

mfishman@flatironinstitute.org

## *Who am I?*

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

## *Who am I?*

- I received my PhD from Caltech with John Preskill and Steve White in 2018.
- I visited Frank Verstraete and Jutho Haegeman for one year during my PhD at the University of Vienna and University of Ghent.

## *Who am I?*

- I received my PhD from Caltech with John Preskill and Steve White in 2018.
- I visited Frank Verstraete and Jutho Haegeman for one year during my PhD at the University of Vienna and University of Ghent.
- My thesis focused on developing novel tensor network algorithms for free fermions, optimizing and contracting infinite tensor networks, etc.

## *Who am I?*

- I received my PhD from Caltech with John Preskill and Steve White in 2018.
- I visited Frank Verstraete and Jutho Haegeman for one year during my PhD at the University of Vienna and University of Ghent.
- My thesis focused on developing novel tensor network algorithms for free fermions, optimizing and contracting infinite tensor networks, etc.
- Joined CCQ in the fall of 2018.

## *Who am I?*

- I received my PhD from Caltech with John Preskill and Steve White in 2018.
- I visited Frank Verstraete and Jutho Haegeman for one year during my PhD at the University of Vienna and University of Ghent.
- My thesis focused on developing novel tensor network algorithms for free fermions, optimizing and contracting infinite tensor networks, etc.
- Joined CCQ in the fall of 2018.
- Associate data scientist at CCQ, lead developer of ITensor and co-developer of PastaQ with Giacomo Torlai.

## *Who am I?*

- I received my PhD from Caltech with John Preskill and Steve White in 2018.
- I visited Frank Verstraete and Jutho Haegeman for one year during my PhD at the University of Vienna and University of Ghent.
- My thesis focused on developing novel tensor network algorithms for free fermions, optimizing and contracting infinite tensor networks, etc.
- Joined CCQ in the fall of 2018.
- Associate data scientist at CCQ, lead developer of ITensor and co-developer of PastaQ with Giacomo Torlai.
- Continuing to develop novel tensor network algorithms, with a focus on making them available as open source software.

## *Who am I?*

- I received my PhD from Caltech with John Preskill and Steve White in 2018.
- I visited Frank Verstraete and Jutho Haegeman for one year during my PhD at the University of Vienna and University of Ghent.
- My thesis focused on developing novel tensor network algorithms for free fermions, optimizing and contracting infinite tensor networks, etc.
- Joined CCQ in the fall of 2018.
- Associate data scientist at CCQ, lead developer of ITensor and co-developer of PastaQ with Giacomo Torlai.
- Continuing to develop novel tensor network algorithms, with a focus on making them available as open source software.
- Find out more here: `https://mtfishman.github.io/`.

## *What is CCQ?*

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

## *What is CCQ?*

- Part of the Flatiron Institute, funded by the Simons Foundation.
- FI is focused on supporting computational research: computers clusters, software development, algorithm development, and scientific applications.

# *What is CCQ?*

- Part of the Flatiron Institute, funded by the Simons Foundation.
- FI is focused on supporting computational research: computers clusters, software development, algorithm development, and scientific applications.
- 5 research centers: math, biology, astrophysics, neuroscience, and quantum.

## *What is CCQ?*

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

## *What is CCQ?*

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

## *What is CCQ?*

- Part of the Flatiron Institute, funded by the Simons Foundation.
- FI is focused on supporting computational research: computers clusters, software development, algorithm development, and scientific applications.
- 5 research centers: math, biology, astrophysics, neuroscience, and quantum.
- Reach out to us for software needs (new features, bugs), algorithm development, 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 ITensor?*

- Stands for “Intelligent Tensor”.

## *What is ITensor?*

- Stands for "Intelligent Tensor".
- Started by Steve White of UCI (inventor of the DMRG tensor network algorithm) [TODO: When?].

# *What is ITensor?*

- Stands for “Intelligent Tensor”.
- Started by Steve White of UCI (inventor of the DMRG tensor network algorithm) [TODO: When?].
- Development taken over by Miles Stoudenmire in 2011 [TODO: Check this!].

## *What is ITensor?*

- Stands for “Intelligent Tensor”.
- Started by Steve White of UCI (inventor of the DMRG tensor network algorithm) [TODO: When?].
- Development taken over by Miles Stoudenmire in 2011 [TODO: Check this!].
- 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 tensor network algorithm) [TODO: When?].
- Development taken over by Miles Stoudenmire in 2011 [TODO: Check this!].
- 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 tensor network algorithm) [TODO: When?].
- Development taken over by Miles Stoudenmire in 2011 [TODO: Check this!].
- 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 tensor network algorithm) [TODO: When?].
- Development taken over by Miles Stoudenmire in 2011 [TODO: Check this!].
- 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.
- Supported by the Simons Foundation.

## *What is ITensor?*

- Stands for “Intelligent Tensor”.
- Started by Steve White of UCI (inventor of the DMRG tensor network algorithm) [TODO: When?].
- Development taken over by Miles Stoudenmire in 2011 [TODO: Check this!].
- 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.
- Supported by the Simons Foundation.
- Find out more at `https://www.itensor.org/`.

## *What is ITensor?*

- Stands for "Intelligent Tensor".
- Started by Steve White of UCI (inventor of the DMRG tensor network algorithm) [TODO: When?].
- Development taken over by Miles Stoudenmire in 2011 [TODO: Check this!].
- 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.
- Supported by the Simons Foundation.
- Find out more at `https://www.itensor.org/`.
- Paper: `https://arxiv.org/abs/2007.14822/`

## *What is Julia?*

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

[TODO: Add ITensor C++ vs. Julia benchmark plot]

# *What is Julia?*

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

[TODO: Add ITensor C++ vs. Julia benchmark plot]

# *What is Julia?*

- Started in 2009, 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.

[TODO: Add ITensor C++ vs. Julia benchmark plot]

# *What is Julia?*

- Started in 2009, 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.

[TODO: Add ITensor C++ vs. Julia benchmark plot]

# *What is Julia?*

- Started in 2009, 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.
- Julia is great, you should use it.

[TODO: Add ITensor C++ vs. Julia benchmark plot]

# *What is Julia?*

- Started in 2009, 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.
- Julia is great, you should use it.
- I could say more, ask me about it.

[TODO: Add ITensor C++ vs. Julia benchmark plot]

## *What is Julia?*

- Started in 2009, 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.
- Julia is great, you should use it.
- I could say more, ask me about it.
- Find out more at: `https://julialang.org/`

[TODO: Add ITensor C++ vs. Julia benchmark plot]

## *What is PastaQ?*

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

## *What is PastaQ?*

- Initiated by Giacomo Torlai while he was a postdoc at CCQ, co-developed by Giacomo and me.
- Quantum computing extension to ITensor.

## *What is PastaQ?*

- Initiated by Giacomo Torlai while he was a postdoc at CCQ, co-developed by Giacomo and me.
- Quantum computing extension to ITensor.
  - Tensor network-based quantum state and process tomography.

# *What is PastaQ?*

- Initiated by Giacomo Torlai while he was a postdoc at CCQ, co-developed by Giacomo and me.
- Quantum computing extension to ITensor.
  - Tensor network-based quantum state and process tomography.
  - Extensive and extendable gate definitions.

## *What is PastaQ?*

- Initiated by Giacomo Torlai while he was a postdoc at CCQ, co-developed by Giacomo and me.
- Quantum computing extension to ITensor.
  - Tensor network-based quantum state and process tomography.
  - Extensive and extendable gate definitions.
  - Built-in and easily extendable circuit definitions.

# *What is PastaQ?*

- Initiated by Giacomo Torlai while he was a postdoc at CCQ, co-developed by Giacomo and me.
- Quantum computing extension to ITensor.
  - 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 while he was a postdoc at CCQ, co-developed by Giacomo and me.
- Quantum computing extension to ITensor.
  - 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: `https://github.com/GTorlai/PastaQ.jl`

## *When should I use tensor networks?*

- Many sites or qubits in your system: linear or log scaling in the system size.

[TODO: "Quantum volume" schematic plot.]

## *When should I use tensor networks?*

- Many sites or qubits in your system: linear or log scaling in the system size.
- Short time evolution, or very noisy evolution.

[TODO: "Quantum volume" schematic plot.]

## *When should I use tensor networks?*

- Many sites or qubits in your system: linear or log scaling in the system size.
- Short time evolution, or very noisy evolution.
- If you need very long time evolution/many layers and few qubits, use full state simultion.

[TODO: "Quantum volume" schematic plot.]

## *When should I use tensor networks?*

- Many sites or qubits in your system: linear or log scaling in the system size.
- Short time evolution, or very noisy evolution.
- If you need very long time evolution/many layers and few qubits, use full state simultion.
  - ITensor and PastaQ handle this seemlessly.

[TODO: "Quantum volume" schematic plot.]

## *When should I use tensor networks?*

- Many sites or qubits in your system: linear or log scaling in the system size.
- Short time evolution, or very noisy evolution.
- If you need very long time evolution/many layers and few qubits, use full state simultion.
  - ITensor and PastaQ handle this seemlessly.
  - This is not the focus of ITensor and PastaQ at the moment, specialized libraries like Yao.jl may be faster.

[TODO: “Quantum volume” schematic plot.]

## *When should I use tensor networks?*

- Many sites or qubits in your system: linear or log scaling in the system size.
- Short time evolution, or very noisy evolution.
- If you need very long time evolution/many layers and few qubits, use full state simultion.
  - ITensor and PastaQ handle this seemlessly.
  - This is not the focus of ITensor and PastaQ at the moment, specialized libraries like Yao.jl may be faster.
- If TNs could do everything, we would not need a quantum computer! But in my opinion, it is the best general purpose tool we have right now.

[TODO: "Quantum volume" schematic plot.]

## *When should I use tensor networks?*

- Many sites or qubits in your system: linear or log scaling in the system size.
- Short time evolution, or very noisy evolution.
- If you need very long time evolution/many layers and few qubits, use full state simultion.
  - ITensor and PastaQ handle this seemlessly.
  - This is not the focus of ITensor and PastaQ at the moment, specialized libraries like Yao.jl may be faster.
- If TNs could do everything, we would not need a quantum computer! But in my opinion, it is the best general purpose tool we have right now.
- Perhaps most importantly, tensor networks are a common, general language for reasoning about quantum many-body systems (for example, quantum circuits).

[TODO: "Quantum volume" schematic plot.]

## *What are tensor networks?*

[TODO: Show drawings of tensor networks.]

## *How do I install ITensor/PastaQ?*

1. Download Julia.

[TODO: Add links, show code]

## *How do I install ITensor/PastaQ?*

1. Download Julia.
2. Add ITensors.jl.

[TODO: Add links, show code]

## *How do I install ITensor/PastaQ?*

1. Download Julia.
2. Add ITensors.jl.
3. Add PastaQ.jl.

[TODO: Add links, show code]

## *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

i = Index(2)
```

```
# Load ITensor

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

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

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

```
# Z|Z+⟩ = |Z+⟩


# 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*

```
using ITensorVisualizationBase: set_backend!
```

# *Tutorial: One-site state basics*

```
using ITensorVisualizationBase: set_backend!
```

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

@visualize dag(Zp) * Xp
```

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

# *Tutorial: One-site state basics*

```
using ITensorVisualizationBase: set_backend!
```

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

@visualize dag(Zp) * Xp
```

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

```
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*

```
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
```

```
# Overload ITensors.jl
# behavior

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

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

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

```
# ≈ 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*

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

Zp = state("Z+", i)           # |Z+⟩
Zm = state("Z−", i)           # |Z−⟩
```

```
XZp = X * Zp                  # X|Z+⟩ = |Z−⟩
XZp == Zm                     # false
XZp == Zm'                    # true
noprime(XZp) == Zm            # 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
```

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

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

```
# false

# ≈ 1
# ≈ 1
```

# *Tutorial: Custom one-site operators*

```julia
import ITensors: op

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

```julia
# 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")          # (dim=2|id=505|"S=1/2")
i2 = Index(2, "S=1/2")          # (dim=2|id=576|"S=1/2")

i1 == i2                        # false

ZpZm = ITensor(i1, i2)          # |Z+⟩_1|Z−⟩_2 = |Z+Z−⟩
ZpZm[i1=>1, i2=>2] = 1
```

```
Zp1 = state("Z+", i1)           # |Z+⟩_1
Zp2 = state("Z+", i2)           # |Z+⟩_2

Zm1 = state("Z−", i1)           # |Z−⟩_1
Zm2 = state("Z−", i2)           # |Z−⟩_2

ZpZm = Zp1 * Zm2                # |Z+Z−⟩ = |Z+⟩_1|Z−⟩_2
ZmZp = Zm1 * Zp2                # |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
# $H = -\sum_i X_i X_{i+1} + h \sum_i Z_i$
```

# *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
# H = −∑_i X_i X_{i+1} + h ∑_i Z_i
```

$H = -\sum_i X_i X_{i+1} + h \sum_i Z_i$

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

XX = X1 * X2
ZI = Z1 * Id2
IZ = Id1 * Z2

h = 1.0
H = -XX + h * (ZI + IZ)
```

```
# 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+⟩
# ≈ 2
```

# *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+⟩
# ≈ 2
```

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

```
# ≈ [−√5 −1 1 √5]
```

# *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>]

```julia
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>]

```
using Zygote

∂E(ψ) = gradient(E, ψ)[1]

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

```

```
# Using Zygote for automatic
# differentation of the energy.
#
# Starting state:
#
# |ψ₀⟩ = (|Z+Z+⟩ +
#          |Z−Z−⟩)/√2
```

## *Tutorial: Two-site state optimization*

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

```julia
using Zygote

∂E(ψ) = gradient(E, ψ)[1]

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

```

```
# Using Zygote for automatic
# differentation of the energy.
#
# Starting state:
#
# |ψ0⟩ = (|Z+Z+⟩ +
#           |Z−Z−⟩)/√2
```

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


E(ψ0), norm(∂E(ψ0))
E(ψ), norm(∂E(ψ))

```

```
# Minimize:
#
# E(ψ) = ⟨ψ|H|ψ⟩/⟨ψ|ψ⟩
#
# (−1, 4)
# (−2.236068, 4.2766287e−6)
# ≈ (−√5, 0)
```

# *Tutorial: Two-site circuit optimization*

```julia
function op(
  ::OpName"U",
  ::SiteType"S=1/2";
  θ1, θ2
)
  c1 = cos(θ1/2)
  s1 = sin(θ1/2)
  c2 = cos(θ2/2)
  s2 = sin(θ2/2)
  return [
    c1  0   0 -s1
    0  c2 -s2   0
    0  s2  c2   0
    s1  0   0  c1
  ]
end
```

```julia
# Circuit optimization.
#
# Note: This is a Matchgate.
# We could do this
# with free fermions.
```

# *Tutorial: Two-site circuit optimization*

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

```julia
ψ0 = Zp1 * Zp2

function E(θ)
  θ1, θ2 = θ
  Uθ = op("U", i1, i2;
              θ1=θ1, θ2=θ2)
  ψθ = apply(Uθ, ψ0)
  return inner(ψθ', H, ψθ)
end
```

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

# Find U(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)
  θ1, θ2 = θ
  U_θ = op("U", i1, i2;
            θ1=θ1, θ2=θ2)
  ψ_θ = apply(U_θ, ψ₀)
  return inner(ψ_θ', H, ψ_θ)
end
```

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

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

```
θ₀ = [π/2, 0]
θ = minimize(E, ∂E, θ₀;
        nsteps=30, γ=0.1)

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

```
# E(θ₀), norm(∂E(θ₀)) =
#   (−1, 2)

# E(θ), norm(∂E(θ)) =
#   (−2.2360675, 0.0014603)
```

# *Tutorial: Two-site fidelity optimization*

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

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

function F(θ)
  θ1, θ2 = θ
  Uθ = op("U", i1, i2;
              θ1=θ1, θ2=θ2)
  ψθ = apply(Uθ, ψ0)
  return -abs(inner(ψ, ψθ))
end
```

```
# Reference state:
# |0⟩ = |Z+Z+⟩

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

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

# *Tutorial: Two-site fidelity optimization*

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

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

function F(θ)
  θ1, θ2 = θ
  Uθ = op("U", i1, i2;
              θ1=θ1, θ2=θ2)
  ψθ = apply(Uθ, ψ0)
  return -abs(inner(ψ, ψθ))
end
```

```
# Reference state:
# |0⟩ = |Z+Z+⟩

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

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

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

F(θ0), norm(∂F(θ0))
```

```
# F(θ0), norm(∂F(θ0)) =
#   (−0.5, 0.5)

# F(θ), norm(∂F(θ)) =
#   (−0.9938992, 0.07786879)
```

## *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.