LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Faculty of Physics
Tensor Networks 2022
Lecturer: Jan von Delft
Tutorial: Andreas Gleis, Jheng-Wei Li, Jeongmin Shim, Changkai Zhang

https://moodle.lmu.de → Search courses: 'Tensor Networks 2022'

Tutorial 03_jl: SVD, MPO, Fermions

Solution T03.1: Reshaping a general tensor into MPS form [6]

(a) SVD a tensor without or with truncation

(i) *Write your own SVD function.* There is a new function `svdTr` under the `Tensor` directory. Compare this with your version.
Note: `svdTr` returns $V^\dagger$ while `svd` returns V. Also, `svdTr` returns S as a vector while `svd` returns S as either a matrix or a vector depending on a setting.

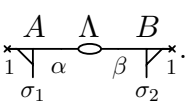
(ii) *Test your code.* We test `svdTr` for the GHZ state as follows.

```
using LinearAlgebra
using Printf

include("Tensor.jl")

## Perform an SVD on the GHZ state
GHZ3 = zeros(2,2,2); #[s1]x[s2]x[s3]
GHZ3[1,1,1] = 1/sqrt(2); # |000>
GHZ3[2,2,2] = 1/sqrt(2); # |111>
## (1)
U,S,Vd = svdTr(GHZ3,3,1,Inf,0)
## (2)
# Verify U S V^\dagger = T_GHZ
T_GHZ = contract(U,2,1,contract(diagm(S),2,1,Vd,3,1),3,1)
err = sum(GHZ3-T_GHZ)
@printf("SVD error: %.4f %%\n",err)
# Verify S = Diagonal([1, 1]) / \sqrt(2)
S
## (3)
# Set NKeep = 1
_,_,_,dw = svdTr(GHZ3,3,1,1,0)
dw # Discarded weight = 0.5
## (4)
U,S,Vd = svdTr(GHZ3,3,[1,2],Inf,0)
size(U) # rank-3 tensor
size(S) # S is a vector
size(Vd) # rank-2 tensor [matrix]
# Verify U S V^\dagger = T_GHZ
T_GHZ = contract(U,3,3,contract(diagm(S),2,1,Vd,2,1),2,1)
err = sum(GHZ3-T_GHZ)
@printf("SVD error: %.4f %%\n",err)
# Verify S = Diagonal([1, 1]) / \sqrt(2)
S
```

(b) Write a GHZN state as an MPS

[i] $N = 2$. The tensor network diagram for an SVD representation of $|\mathrm{GHZ2}\rangle$ is

$$|\mathrm{GHZ2}\rangle = \begin{pmatrix} |0\rangle_1 & |1\rangle_1 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} |0\rangle_2 \\ |1\rangle_2 \end{pmatrix} =: \mathcal{A}\Lambda\mathcal{B} =: |\sigma_1\rangle_1 A^{\sigma_1} \Lambda |\sigma_2\rangle_2 B^{\sigma_2} =$$



We thus read off the the ingredients of $\mathcal{A} = |\sigma_1\rangle_1 A^{\sigma_1}$ and $\mathcal{B} = |\sigma_2\rangle_2 B^{\sigma_2}$ as follows:

$$\mathcal{A} = |0\rangle_1 \underbrace{\begin{pmatrix}1 & 0\end{pmatrix}}_{A^{\sigma_1=0}} + |1\rangle_1 \underbrace{\begin{pmatrix}0 & 1\end{pmatrix}}_{A^{\sigma_1=1}}, \quad \mathcal{B} = |0\rangle_2 \underbrace{\begin{pmatrix}1\\0\end{pmatrix}}_{B^{\sigma_2=0}} + |1\rangle_2 \underbrace{\begin{pmatrix}0\\1\end{pmatrix}}_{B^{\sigma_2=1}}.$$

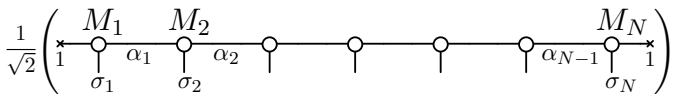
[ii] *Generalize to an arbitrary N.* Since $|\mathrm{GHZ}N\rangle$ contains terms involving only products of $|0\rangle_\ell$s or products of $|1\rangle_\ell$s,

$$|\mathrm{GHZ}N\rangle = \tfrac{1}{\sqrt{2}}\big(|0,\ldots,0\rangle + |1,\ldots,1\rangle\big) = \tfrac{1}{\sqrt{2}}\begin{pmatrix}|0\rangle_1 & |1\rangle_1\end{pmatrix}\mathcal{M}_2\ldots\mathcal{M}_{N-1}\begin{pmatrix}|0\rangle_N\\|1\rangle_N\end{pmatrix},$$

we read off the ingredients of $\mathcal{M}_\ell = |\sigma_\ell\rangle_\ell M^{\sigma_\ell}$ as follows:

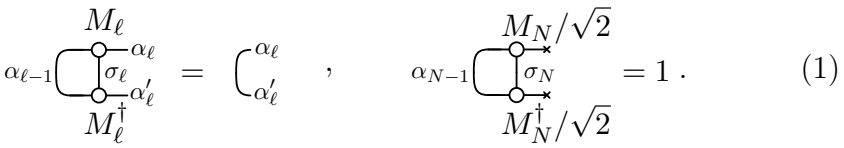
$$\mathcal{M}_\ell = \begin{pmatrix}|0\rangle_\ell & 0\\ 0 & |1\rangle_\ell\end{pmatrix}, \quad (M_\ell)^{\sigma_\ell=0} = \begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix}, \quad (M_\ell)^{\sigma_\ell=1} = \begin{pmatrix}0 & 0\\ 0 & 1\end{pmatrix}.$$

[iii] *Canonical MPS.* The MPS representation of $|\mathrm{GHZ}N\rangle$ reads:



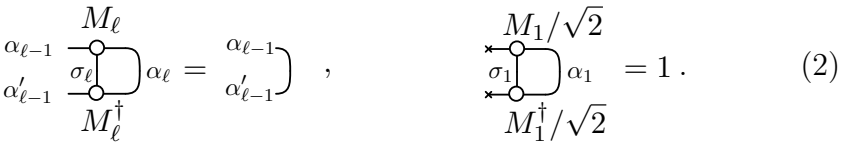
This MPS is left-canonical if the prefactor $\frac{1}{\sqrt{2}}$ is absorbed into $\mathcal{M}_N$, or right-canonical if the prefactor is absorbed in to $\mathcal{M}_1$:

left-canonical:



(1)

right-canonical:



(2)

(iv) *SVD and MPS.* We verify the observation in [iii] as follows.

```
using LinearAlgebra
using Printf

include("Tensor.jl")

N = 10
GHZ_N = zeros(2^10,1)
GHZ_N[1] = 1/sqrt(2); # |0,...,0>/sqrt(2)
GHZ_N[end] = 1/sqrt(2); # |1,...,1>/sqrt(2)

M = Array{Any}(undef,1,N);
T = reshape(GHZ_N*sqrt(2), (1,2,length(GHZ_N)/2))
for itN = (1:N-1)
    global T
    U,S,Vd = svdTr(T,3,[1,2],Inf,0)
    M[itN] = U
    T = contract(diagm(S),2,2,Vd,2,1)
    T = reshape(T, (size(T,1),2,size(T,2)/2))
end
M[end] = T
```

```
21
22 # check left canonical
23 M_N = M[N]/sqrt(2)
24 for itN in (1:N)
25     if itN < N
26         MM = contract(M[itN],3,[1,2],conj(M[itN]),3,[1,2])
27     else
28         MM = contract(M_N,3,[1,2],conj(M_N),3,[1,2])
29     end
30     Id = I(size(MM,1))
31     err = norm(MM-Id)
32     @printf("norm of diffrence between MM* and the expected identity matrix at site
           %d: %.4e\n",itN,err)
33 end
34
35 # check right canonical
36 M_1 = M[1]/sqrt(2)
37 for itN in (1:N)
38     if itN > 1
39         MM = contract(M[itN],3,[2,3],conj(M[itN]),3,[2,3])
40     else
41         MM = contract(M_1,3,[2,3],conj(M_1),3,[2,3])
42     end
43     Id = I(size(MM,1))
44     err = norm(MM-Id)
45     @printf("norm of diffrence between MM* and the expected identity matrix at site
           %d: %.4e\n",itN,err)
46 end
47
48 # Nkeep = 2
49 Nkeep = 2
50 dw_set = zeros(N-1,1)
51 T = reshape(GHZ_N*sqrt(2), (1,2,length(GHZ_N)/2))
52 for itN = (1:N-1)
53     global T
54     U,S,Vd,dw = svdTr(T,3,[1,2],Nkeep,0)
55     M[itN] = U
56     dw_set[itN] = dw
57     T = contract(diagm(S),2,2,Vd,2,1)
58     T = reshape(T, (size(T,1),2,size(T,2)/2))
59     @printf("discarded weight of the SVD on site %d: %f\n",itN,dw_set[itN])
60 end
61
62 # Nkeep = 1
63 Nkeep = 1
64 dw_set = zeros(N-1,1)
65 T = reshape(GHZ_N*sqrt(2), (1,2,length(GHZ_N)/2))
66 for itN = (1:N-1)
67     global T
68     U,S,Vd,dw = svdTr(T,3,[1,2],Nkeep,0)
69     M[itN] = U
70     dw_set[itN] = dw
71     T = contract(diagm(S),2,2,Vd,2,1)
72     T = reshape(T, (size(T,1),2,size(T,2)/2))
73     # note that the discarded weight is nonzero for itN = 1
74     @printf("discarded weight of the SVD on site %d: %f\n",itN,dw_set[itN])
75 end
```

For `Nkeep`=∞, the resulting MPS is left-canonical (or right-canoncial) if the prefactor $\frac{1}{\sqrt{2}}$ is absorbed into $\mathcal{M}_1$ (or $\mathcal{M}_N$) as found in (iii). For `Nkeep`=2, the discarded weights are all 0, because the bond dimensions of α_ℓ for $\mathcal{M}_\ell$ is $D = 2$. Since all weights are kept, the resulting MPS tensor agrees with the one found in (iii). However, for `Nkeep`=1, some discarded weight(s) are nonzero.

Solution T03.2: Matrix product operators [6]

(a) Spin Systems

[(i)] *Spin-$\frac{1}{2}$ Ising model.* [1] The MPO ingredients W_1 or $W_{\mathscr{L}}$ are the last row vector or the first column vector of W_ℓ, respectively:

$$W_1 = \begin{pmatrix} 0 & JS_1^z & \mathbb{1}_1 \end{pmatrix}, \quad W_{\mathscr{L}} = \begin{pmatrix} \mathbb{1}_{\mathscr{L}} \\ S_{\mathscr{L}}^z \\ 0 \end{pmatrix}.$$

To verify that this works, multiply out the MPO matrices for a short chain, e.g. $\mathscr{L} = 3$:

$$\begin{aligned} W_1 W_2 W_3 &= \begin{pmatrix} 0 & JS_1^z & \mathbb{1}_1 \end{pmatrix} \begin{pmatrix} \mathbb{1}_2 & 0 & 0 \\ S_2^z & 0 & 0 \\ 0 & JS_2^z & \mathbb{1}_2 \end{pmatrix} \begin{pmatrix} \mathbb{1}_3 \\ S_3^z \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & JS_1^z & \mathbb{1}_1 \end{pmatrix} \begin{pmatrix} \mathbb{1}_2\mathbb{1}_3 \\ S_2^z\mathbb{1}_3 \\ JS_2^zS_3^z \end{pmatrix} \\ &= J\Big(S_1^zS_2^z\mathbb{1}_3 + \mathbb{1}_1S_2^zS_3^z\Big). \end{aligned}$$

(2) We now explicitly construct the MPO ingredient W_ℓ, and multiply out the MPO to verify that it reproduces the Hamiltonian H_{Ising}:

```
using LinearAlgebra
using Printf

include("Tensor.jl")

J = 1
L = 4
Sz = [1 0; 0 -1]/2
W = Array{Any}(undef,1,L); # MPO W
# first site
W[1] = zeros(1,2,3,2); # leg ordering: left bottom right top
W[1][1,:,2,:] = J*Sz; # J*S_1^z
W[1][1,:,3,:] = I(2); # I
# sites between first & last sites
for itW in (2:L-1)
    W[itW] = zeros(3,2,3,2); # leg ordering: left bottom right top
    W[itW][1,:,1,:] = I(2); # I
    W[itW][2,:,1,:] = Sz; # S_itW^z
    W[itW][3,:,2,:] = J*Sz; # J*S_itW^z
    W[itW][3,:,3,:] = I(2); # I
end
# last site
W[L] = zeros(3,2,1,2); # left ordering: left bottom right top
W[L][1,:,1,:] = I(2); # I
W[L][2,:,1,:] = Sz; # S_L^z
##
# The MPOs are multiplied together & compared with the full Hamiltonian for
# $\mathcal[L]=4$.

H_MPO = reshape([1],(1,1,1,1)); # Full Hamiltonian constructed from MPO W
for itW in (1:L)
    global H_MPO
    H_MPO = contract(H_MPO,4,3,W[itW],4,1)
    H_MPO = permutedims(H_MPO,(1,2,4,5,3,6))
    # reshape H_MPO to be a rank-4 tensor
    H_MPO = reshape(H_MPO,(size(H_MPO,1),
                           size(H_MPO,2)*size(H_MPO,3),
                           size(H_MPO,4),
                           size(H_MPO,5)*size(H_MPO,6)))
end
H_MPO = reshape(H_MPO,(size(H_MPO,2),size(H_MPO,4)))

# H_dir: Full Hamiltonian constructed directly
Sz1 = kron(Sz,kron(I(2),kron(I(2),I(2))))
Sz2 = kron(I(2),kron(Sz,kron(I(2),I(2))))
Sz3 = kron(I(2),kron(I(2),kron(Sz,I(2))))
Sz4 = kron(I(2),kron(I(2),kron(I(2),Sz)))
H_dir = J*(Sz1*Sz2+Sz2*Sz3+Sz3*Sz4)
# comparing H_MPO & H_dir
diff = norm(H_dir-H_MPO)
```

```
@printf("Norm of difference between H_MPO & H_dir: %e %%",diff)
```

[ii] *Spin-$\frac{1}{2}$ XY model.* The Hamiltonian for the XY model can be phrased as an MPO with bond dimension $D_W = 4$, using the ingredients

$$W_1 = \begin{pmatrix} 0 & JS_1^x & JS_1^y & \mathbb{1}_1 \end{pmatrix}, \quad W_\ell = \begin{pmatrix} \mathbb{1}_\ell & 0 & 0 & 0 \\ S_\ell^x & 0 & 0 & 0 \\ S_\ell^y & 0 & 0 & 0 \\ 0 & JS_\ell^x & JS_\ell^y & \mathbb{1}_\ell \end{pmatrix}, \quad W_\mathscr{L} = \begin{pmatrix} \mathbb{1}_\mathscr{L} \\ S_\mathscr{L}^x \\ S_\mathscr{L}^y \\ 0 \end{pmatrix}.$$

To verify that this works, multiply out the MPO matrices for a short chain, e.g. $\mathscr{L} = 3$:

$$\begin{aligned} W_1 W_2 W_3 &= \begin{pmatrix} 0 & JS_1^x & JS_1^y & \mathbb{1}_1 \end{pmatrix} \begin{pmatrix} \mathbb{1}_2 & 0 & 0 & 0 \\ S_2^x & 0 & 0 & 0 \\ S_2^y & 0 & 0 & 0 \\ 0 & JS_2^x & JS_2^y & \mathbb{1}_2 \end{pmatrix} \begin{pmatrix} \mathbb{1}_3 \\ S_3^x \\ S_3^y \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & JS_1^x & JS_1^y & \mathbb{1}_1 \end{pmatrix} \begin{pmatrix} \mathbb{1}_2\mathbb{1}_3 \\ S_2^x\mathbb{1}_3 \\ S_2^y\mathbb{1}_3 \\ J\Big(S_2^xS_3^x + S_2^yS_3^y\Big) \end{pmatrix} \\ &= J\Big(S_1^xS_2^x\mathbb{1}_3 + S_1^yS_2^y\mathbb{1}_3 + \mathbb{1}_1S_2^xS_3^x + \mathbb{1}_1S_2^yS_3^y\Big). \end{aligned}$$

The term $h_\ell S_\ell^z$ can be included in W_ℓ as

$$W_1 = \begin{pmatrix} h_1S_1^z & JS_1^x & JS_1^y & \mathbb{1}_1 \end{pmatrix}, \quad W_\ell = \begin{pmatrix} \mathbb{1}_\ell & 0 & 0 & 0 \\ S_\ell^x & 0 & 0 & 0 \\ S_\ell^y & 0 & 0 & 0 \\ h_\ell S_\ell^z & JS_\ell^x & JS_\ell^y & \mathbb{1}_\ell \end{pmatrix}, \quad W_\mathscr{L} = \begin{pmatrix} \mathbb{1}_\mathscr{L} \\ S_\mathscr{L}^x \\ S_\mathscr{L}^y \\ h_\mathscr{L} S_\mathscr{L}^z \end{pmatrix}.$$

To verify that this works, multiply out the MPO matrices for a short chain, e.g. $\mathscr{L} = 3$:

$$\begin{aligned} W_1 W_2 W_3 &= \begin{pmatrix} h_1S_1^z & JS_1^x & JS_1^y & \mathbb{1}_1 \end{pmatrix} \begin{pmatrix} \mathbb{1}_2 & 0 & 0 & 0 \\ S_2^x & 0 & 0 & 0 \\ S_2^y & 0 & 0 & 0 \\ h_2S_2^z & JS_2^x & JS_2^y & \mathbb{1}_2 \end{pmatrix} \begin{pmatrix} \mathbb{1}_3 \\ S_3^x \\ S_3^y \\ h_3S_3^z \end{pmatrix} \\ &= \begin{pmatrix} h_1S_1^z & JS_1^x & JS_1^y & \mathbb{1}_1 \end{pmatrix} \begin{pmatrix} \mathbb{1}_2\mathbb{1}_3 \\ S_2^x\mathbb{1}_3 \\ S_2^y\mathbb{1}_3 \\ h_2S_2^z\mathbb{1}_3 + h_3\mathbb{1}_2S_3^z + J\Big(S_2^xS_3^x + S_2^yS_3^y\Big) \end{pmatrix} \\ &= h_1S_1^z\mathbb{1}_2\mathbb{1}_3 + h_3\mathbb{1}_1S_2^z\mathbb{1}_3 + h_3\mathbb{1}_1\mathbb{1}_2S_3^z + J\Big(S_1^xS_2^x\mathbb{1}_3 + S_1^yS_2^y\mathbb{1}_3 + \mathbb{1}_1S_2^xS_3^x + \mathbb{1}_1S_2^yS_3^y\Big). \end{aligned}$$

[(iii)] *Exponential long-range interactions.* [1] These can be encoded in W_ℓ with $D_W = 3$ as

$$W_1 = \begin{pmatrix} 0 & e^{-\lambda}S_1^z & \mathbb{1}_1 \end{pmatrix}, \quad W_\ell = \begin{pmatrix} \mathbb{1}_\ell & 0 & 0 \\ S_\ell^z & e^{-\lambda}\mathbb{1}_\ell & 0 \\ 0 & e^{-\lambda}S_\ell^z & \mathbb{1}_\ell \end{pmatrix}, \quad W_\ell = \begin{pmatrix} \mathbb{1}_\ell \\ S_\mathscr{L}^z \\ 0 \end{pmatrix}$$

To verify that this works, multiply out the MPO matrices for a short chain, e.g. $\mathscr{L} = 3$:

$$W_1 W_2 W_3 = \begin{pmatrix} 0 & e^{-\lambda} S_1^z & \mathbb{1}_1 \end{pmatrix} \begin{pmatrix} \mathbb{1}_2 & 0 & 0 \\ S_2^z & e^{-\lambda}\mathbb{1}_2 & 0 \\ 0 & e^{-\lambda} S_2^z & \mathbb{1}_2 \end{pmatrix} \begin{pmatrix} \mathbb{1}_3 \\ S_3^z \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & e^{-\lambda} S_1^z & \mathbb{1}_1 \end{pmatrix} \begin{pmatrix} \mathbb{1}_2\mathbb{1}_3 \\ S_2^z\mathbb{1}_3 + e^{-\lambda}\mathbb{1}_2 S_3^z \\ e^{-\lambda} S_2^z S_3^z \end{pmatrix}$$
$$= e^{-\lambda}\left(S_1^z S_2^z \mathbb{1}_3 + \mathbb{1}_1 S_2^z S_3^z\right) + e^{-2\lambda} S_1^z \mathbb{1}_2 S_3^z .$$

(2) We implement the Hamiltonian $-H_{\lambda=1}$ and show that the state $|\downarrow \ldots \downarrow\rangle$ is an eigenstate:

```
using LinearAlgebra
using Printf

include("Tensor.jl")

L = 6
lambda = 1;
Sz = [1 0; 0 -1]/2
W = Array{Any}(undef,1,L); # MPO W
# first site
W[1] = zeros(1,2,3,2); # leg ordering: left bottom right top
W[1][1,:,2,:] = -exp(-lambda)*Sz; # J*S_1^z
W[1][1,:,3,:] = I(2); # I
# sites between first & last sites
for itW in (2:L-1)
    W[itW] = zeros(3,2,3,2); # leg ordering: left bottom right top
    W[itW][1,:,1,:] = I(2); # I
    W[itW][2,:,1,:] = Sz; # S_itW^z
    W[itW][2,:,2,:] = exp(-lambda)*I(2); # e(^\lambda)*I
    W[itW][3,:,2,:] = -exp(-lambda)*Sz; # J*S_itW^z
    W[itW][3,:,3,:] = I(2); # I
end
# last site
W[L] = zeros(3,2,1,2); # left ordering: left bottom right top
W[L][1,:,1,:] = I(2); # I
W[L][2,:,1,:] = Sz; # S_L^z
##
# The MPOs are multiplied together & compared with the full Hamiltonian for
# $\mathcal[L]=4$.

Psi = zeros(2^L,1);
Psi[end] = 1; # |Psi> = |down_arrow ... down_arrow>

H_MPO = reshape([1],(1,1,1,1)); # Full Hamiltonian constructed from MPO W
for itW in (1:L)
    global H_MPO
    H_MPO = contract(H_MPO,4,3,W[itW],4,1)
    H_MPO = permutedims(H_MPO,(1,2,4,5,3,6))
    # reshape H_MPO to be a rank-4 tensor
    H_MPO = reshape(H_MPO,(size(H_MPO,1),
                           size(H_MPO,2)*size(H_MPO,3),
                           size(H_MPO,4),
                           size(H_MPO,5)*size(H_MPO,6)))
end
H_MPO = reshape(H_MPO,(size(H_MPO,2),size(H_MPO,4)))

HPsi = H_MPO*Psi; # H|Psi>
HPsi = HPsi/norm(HPsi); # normalize H|Psi>
P_Psi = Psi'.*Psi; # Proejction on the Psi
diff = norm(P_Psi*HPsi - HPsi); # if P(H|Psi>) = H|Psi>, then H|Psi> = |Psi>
@printf("Norm of difference between PH|Psi> and H|Psi>: %e %%",diff)
```

(b) Fermionic systems

[i] *Spinless tight-binding free fermions.* To keep track of fermion signs, we introduce z factors:

$$\hat{c}_\ell^\dagger \hat{c}_{\ell+1} = (c_\ell^\dagger z_{\ell+1} z_{\ell+2} \ldots z_{\mathscr{L}})(c_{\ell+1} z_{\ell+2} \ldots \times z_{\mathscr{L}}) = c_\ell^\dagger z_{\ell+1} c_{\ell+1} = c_\ell^\dagger c_{\ell+1} \,, \tag{3}$$
$$\hat{c}_{\ell+1}^\dagger \hat{c}_\ell = (c_{\ell+1}^\dagger z_{\ell+2} \ldots z_{\mathscr{L}})(c_\ell z_{\ell+1} z_{\ell+2} \ldots \times z_{\mathscr{L}}) = c_{\ell+1}^\dagger z_{\ell+1} c_\ell = c_{\ell+1}^\dagger c_\ell \,. \tag{4}$$

For the last steps on the right we exploited the fact that $c_\ell^\dagger z_{\ell+1} = c_\ell^\dagger$ and $z_{\ell+1} c_\ell = c_\ell$, since these operators are nonzero only when acting on $|0\rangle_\ell$ or $|1\rangle_\ell$, respectively. Hence, all z factors drop out, and we choose

$$W_1 = \begin{pmatrix} 0 & tc_1^\dagger & tc_1 & \mathbb{1}_1 \end{pmatrix}, \quad W_\ell = \begin{pmatrix} \mathbb{1}_\ell & 0 & 0 & 0 \\ c_\ell & 0 & 0 & 0 \\ c_\ell^\dagger & 0 & 0 & 0 \\ 0 & tc_\ell^\dagger & tc_\ell & \mathbb{1}_\ell \end{pmatrix}, \quad W_{\mathscr{L}} = \begin{pmatrix} \mathbb{1}_{\mathscr{L}} \\ c_{\mathscr{L}} \\ c_{\mathscr{L}}^\dagger \\ 0 \end{pmatrix}.$$

[ii] *Jordan–Wigner transformation.* Under the Jordan-Wigner transformation, $\hat{c}_\ell^\dagger \hat{c}_{\ell+1}$ and $\hat{c}_{\ell+1}^\dagger \hat{c}_\ell$ become as follows:

$$\hat{c}_\ell^\dagger \hat{c}_{\ell+1} = (\sigma_1^z \dots \sigma_{\ell-1}^z \sigma_\ell^+)(\sigma_1^z \dots \sigma_{\ell-1}^z \sigma_\ell^z \sigma_{\ell+1}^-) = \sigma_\ell^+ \sigma_\ell^z \sigma_{\ell+1}^- = -\sigma_\ell^+ \sigma_{\ell+1}^- , \tag{5}$$

$$\hat{c}_{\ell+1}^\dagger \hat{c}_\ell = (\sigma_1^z \dots \sigma_{\ell-1}^z \sigma_\ell^z \sigma_{\ell+1}^+)(\sigma_1^z \dots \sigma_{\ell-1}^z \sigma_\ell^-) = \sigma_{\ell+1}^+ \sigma_\ell^z \sigma_\ell^- = -\sigma_{\ell+1}^+ \sigma_\ell^- . \tag{6}$$

For the last steps on the right we exploited the fact that $\sigma_\ell^+ \sigma_\ell^z = -\sigma_\ell^+$ and $\sigma_\ell^z \sigma_\ell^- = -\sigma_\ell^-$ since these operators are nonzero only when acting $|\downarrow\rangle_\ell$ or $|\uparrow\rangle_\ell$, respectively. Hence, all σ_ℓ^z drop out, and we choose

$$W_1 = \begin{pmatrix} 0 & -t\sigma_1^+ & -t\sigma_1^- & \mathbb{1}_1 \end{pmatrix}, \quad W_\ell = \begin{pmatrix} \mathbb{1}_\ell & 0 & 0 & 0 \\ \sigma_\ell^- & 0 & 0 & 0 \\ \sigma_\ell^+ & 0 & 0 & 0 \\ 0 & -t\sigma_\ell^+ & -t\sigma_\ell^- & \mathbb{1}_\ell \end{pmatrix}, \quad W_{\mathscr{L}} = \begin{pmatrix} \mathbb{1}_{\mathscr{L}} \\ \sigma_{\mathscr{L}}^- \\ \sigma_{\mathscr{L}}^+ \\ 0 \end{pmatrix}.$$

The W_ℓ matrices from [i] and [ii] are identical, up to $t \leftrightarrow -t$. This reflect the fact that for spinless fermions, $H_{\text{ff}} = t \sum_{\langle \ell\bar{\ell} \rangle} (c_\ell^\dagger c_{\bar{\ell}} + c_{\bar{\ell}}^\dagger c_\ell)$ is equivalent to the spinflip Hamiltonian $-t \sum_{\langle \ell\bar{\ell} \rangle} (\sigma_\ell^+ \sigma_{\bar{\ell}}^- + \sigma_\ell^- \sigma_{\bar{\ell}}^+)$.

[iii] *Fourier transform.* The operator $c_k^\dagger$ can be expressed as the MPO of $D_W = 2$ as follows.

$$W_1 = \begin{pmatrix} e^{ik} c_1^\dagger & \mathbb{1}_1 \end{pmatrix}, \quad W_\ell = \begin{pmatrix} z_\ell & 0 \\ e^{ik\ell} c_\ell^\dagger & \mathbb{1}_\ell \end{pmatrix}, \quad W_{\mathscr{L}} = \begin{pmatrix} z_{\mathscr{L}} \\ e^{ik\mathscr{L}} c_{\mathscr{L}}^\dagger \end{pmatrix}.$$

To verify that this works, multiply out the MPO matrices for a short chain, e.g., $\mathscr{L} = 3$:

$$\begin{aligned} W_1 W_2 W_3 &= \begin{pmatrix} e^{ik} c_1^\dagger & \mathbb{1}_1 \end{pmatrix} \begin{pmatrix} z_2 & 0 \\ e^{2ik} c_2^\dagger & \mathbb{1}_2 \end{pmatrix} \begin{pmatrix} z_3 \\ e^{3ik} c_3^\dagger \end{pmatrix} = \begin{pmatrix} e^{ik} c_1^\dagger & \mathbb{1}_1 \end{pmatrix} \begin{pmatrix} z_2 z_3 \\ e^{2ik} c_2^\dagger z_3 + e^{3ik} \mathbb{1}_2 c_3^\dagger \end{pmatrix} \\ &= e^{ik} c_1^\dagger z_2 z_3 + e^{2ik} \mathbb{1}_1 c_2^\dagger z_3 + e^{3ik} \mathbb{1}_1 \mathbb{1}_2 c_3^\dagger = \sum_\ell e^{ik\ell} \hat{c}_\ell^\dagger. \end{aligned}$$

Solution T03.3: Iterative diagonalization and MPOs [5]

(a) Spin systems

(i) *Verify your MPO implementation.* (1) We compute the ground state $|\Psi_{\text{MPS}}\rangle$ and its energy E_{MPS}. (2) We construct the MPO for the XY Hamiltonian. (3) We verify that the energy E_{MPO} agrees with E_{MPS} within machine precision.

```
using LinearAlgebra
using Printf
using JLD2

include("Tensor.jl");

J = 1;
L = 50; # maximum chain length
Nkeep = 300; # maximal number of states to keep
S,Id = getLocalSpace("Spin",1/2)

@printf("Iterative Diagonalization\n")

H0 = Id*0; # Hamiltonian for only the 1st site
A0 = getIdentity(1,2,Id,2); # 1st leg is dummy leg (vacuum)

# MPS of the GS
MPS = Array{Any}(undef,1,L);

# initialization
Hnow = H0
D,V = eigen((Hnow+Hnow')/2)
MPS[1] = contract(A0,3,3,V,2,1)
Hprev = diagm(D)

for itL in (2:L)

    global A, Hprev, Eg_Iter;

    # step [i-ii]
    # spin operator at the current site; to be used for generating
    # the coupling term at the next iteration
    Sprev = updateLeft([],[],MPS[itL-1],S,3,MPS[itL-1])

    # step [iii]
    # # add new site
    Anow = getIdentity(Hprev,2,Id,2)
    Hnow = updateLeft(Hprev,2,Anow,[],[],Anow)
    # update the Hamiltonian up to the last sites
    # to the enlarged Hilbert space

    # step [iv]
    # # spin-spin interaction
    Sprev_x = (Sprev[:,1,:]+Sprev[:,2,:])/sqrt(2); Rx = size(Sprev_x);
    Sprev_y = (Sprev[:,1,:]-Sprev[:,2,:])/sqrt(2)/1im; Ry = size(Sprev_y);
    if length(Rx) == 2; Sprev_x = reshape(Sprev_x,(Rx[1],1,Rx[2])); end;
    if length(Ry) == 2; Sprev_y = reshape(Sprev_y,(Ry[1],1,Ry[2])); end;
    Sx = reshape((S[:,1,:]+S[:,2,:])/sqrt(2),(2,1,2));
    Sy = reshape((S[:,1,:]-S[:,2,:])/sqrt(2)/1im,(2,1,2));
    Hxy = J * updateLeft(Sprev_x,3,Anow,Sx,3,Anow) + J * updateLeft(Sprev_y,3,Anow,
        Sy,3,Anow)

    # step [v]
    Hnow = Hnow+Hxy

    # diagonalize the current Hamiltonian
    D,V = eigen((Hnow+Hnow')/2)
    # sort eigenvalues & eigenvectors in the order of increasing
    # eigenvalues
    ids = sortperm(D)
    D = D[ids]
    V = V[:,ids]

    if itL == L; Eg_Iter = minimum(D); end

    # truncation threshold for energy
    Etr = D[min(length(D),Nkeep)]
    oks = (D .< Etr)
    # true: to keep; false: not to keep
    # keep all degenerate states up to tolerance

    if itL < L
        MPS[itL] = contract(Anow,3,3,V[:,oks],2,1)
    else
        MPS[itL] = contract(Anow,3,3,reshape(V[:,1],(size(V[:,1])[1],1)),2,1) #
            select GS at the last step
```

```
    end
    Hprev = diagm(D[oks])

end


@printf("MPS/MPO Contraction\n")

Sx = reshape((S[:,1,:]+S[:,2,:])/sqrt(2),(2,1,2));
Sy = reshape((S[:,1,:]-S[:,2,:])/sqrt(2)/1im,(2,1,2));

W = Array{Any}(undef,1,L); # MPO W
# first site
W[1] = zeros(Complex{Float64},1,2,4,2); # ordering: left bottom right top
W[1][1,:,2,:] = J*Sx; # J*S_1^x
W[1][1,:,3,:] = J*Sy; # J*S_1^y
W[1][1,:,4,:] = I(2); # I
# sites between first & last sites
for itW in (2:L-1)
    W[itW] = zeros(Complex{Float64},4,2,4,2); # ordering: left bottom right top
    W[itW][1,:,1,:] = I(2); # I
    W[itW][2,:,1,:] = Sx; # S_itW^x
    W[itW][3,:,1,:] = Sy; # S_itW^y
    W[itW][4,:,2,:] = J*Sx; # J*S_itW^x
    W[itW][4,:,3,:] = J*Sy; # J*S_itW^y
    W[itW][4,:,4,:] = I(2); # I
end
# last site
W[L] = zeros(Complex{Float64},4,2,1,2); # ordering: left bottom right top
W[L][1,:,1,:] = I(2); # I
W[L][2,:,1,:] = Sx; # S_L^x
W[L][3,:,1,:] = Sy; # S_L^y


Eg_MPO = reshape([1],(1,1,1));
for itL in (1:L)
    global Eg_MPO
    Eg_MPO = updateLeft(Eg_MPO,3,MPS[itL],W[itL],4,MPS[itL]);
end

@save "GSEnergies.jld2" Eg_Iter Eg_MPO

diff = Eg_Iter - Eg_MPO[1];
@printf("Difference between Eg_Iter and Eg_MPO: %.4f\n", diff)
```

(b) Fermionic systems

(i) *Iterative diagonalization.* Under the Jordan-Wigner transformation, the Hamiltonian H_{ff} is mapped to $H_{\mathrm{ff}} = -t\sum_{\ell}(\sigma_{\ell}^{+}\sigma_{\ell+1}^{-} + \text{h.c.})$. The diagonalization of this Hamiltonian is described below. The ground state energy of H_{ff} is found to agree with that of H_{XY} from (a).

```
using LinearAlgebra
using Printf
using JLD2

include("Tensor.jl");

@load "GSEnergies.jld2"

t = 0.5;
L = 50; # maximum chain length()
Nkeep = 300; # maximal number of states to keep
S,Id = getLocalSpace("Spin",1/2)

@printf("Iterative Diagonalization\n")

H0 = Id*0; # Hamiltonian for only the 1st site
A0 = getIdentity(1,2,Id,2); # 1st leg is dummy leg (vacuum)

# MPS of the GS
MPS = Array{Any}(undef,1,L);

# initialization
Hnow = H0
```

```
D,V = eigen((Hnow+Hnow')/2)
MPS[1] = contract(A0,3,3,V,2,1)
Hprev = diagm(D)

# simga_+ sigma_z
SpSz = contract(reshape(sqrt(2)*S[:,1,:],(2,1,2)),3,3,
    reshape(2*S[:,3,:],(2,2)),2,1);
# sigma_z sigma_-
SzSm = contract(reshape(2*S[:,3,:],(2,2)),2,2,
    reshape(sqrt(2)*S[:,2,:],(2,1,2)),3,1);

for itL in (2:L)

    global A, Hprev, SpSz, SzSm, Eg_IterF;

    # step [i-ii]
    # spin operator at the current site; to be used for generating
    # the coupling term at the next iteration
    SpSz_prev = updateLeft([],[],MPS[itL-1],SpSz,3,MPS[itL-1]);
    SzSm_prev = updateLeft([],[],MPS[itL-1],SzSm,3,MPS[itL-1]);

    # step [iii]
    # # add new site
    Anow = getIdentity(Hprev,2,Id,2)
    Hnow = updateLeft(Hprev,2,Anow,[],[],Anow)
    # update the Hamiltonian up to the last sites
    # to the enlarged Hilbert space

    # step [iv]
    # # spin-spin interaction
    Hxy = t * updateLeft(SpSz_prev,3,Anow,reshape(sqrt(2)*S[:,2,:],(2,1,2)),3,Anow)
         + t * updateLeft(SzSm_prev,3,Anow,reshape(sqrt(2)*S[:,1,:],(2,1,2)),3,Anow)
        ;

    # step [v]
    Hnow = Hnow+Hxy

    # diagonalize the current Hamiltonian
    D,V = eigen((Hnow+Hnow')/2)
    # sort eigenvalues & eigenvectors in the order of increasing
    # eigenvalues
    ids = sortperm(D)
    D = D[ids]
    V = V[:,ids]

    if itL == L; Eg_IterF = minimum(D); end

    # truncation threshold for energy
    Etr = D[min(length(D),Nkeep)]
    oks = (D .< Etr)
    # true: to keep; false: not to keep
    # keep all degenerate states up to tolerance

    if itL < L
        MPS[itL] = contract(Anow,3,3,V[:,oks],2,1)
    else
        MPS[itL] = contract(Anow,3,3,reshape(V[:,1],(size(V[:,1])[1],1)),2,1) #
            select GS at the last step
    end
    Hprev = diagm(D[oks])

end

@save "MPS.jld2" MPS

@printf("Ground state energy of Fermionic chain: %.4f\n", Eg_IterF)

diff = Eg_Iter - Eg_IterF[1];
@printf("Difference between Eg_Iter and Eg_IterF: %.4f\n", diff)
```

(ii) *MPO for Hamiltonian.* The construction of the MPO for H_{ff} is described below. The energy E_{MPS} from (i) is reproduced.

```
using LinearAlgebra
using Printf
```

```
using JLD2

include("Tensor.jl");

@load "GSEnergies.jld2"
@load "MPS.jld2"

t = 0.5;
L = 50; # maximum chain length
S,Id = getLocalSpace("Spin",1/2)

W = Array{Any}(undef,1,L); # MPO W
# first site
W[1] = zeros(Complex{Float64},1,2,4,2); # ordering: left bottom right top
W[1][1,:,2,:] = -t*sqrt(2)*S[:,1,:]; # t*S_1^x
W[1][1,:,3,:] = -t*sqrt(2)*S[:,2,:]; # t*S_1^y
W[1][1,:,4,:] = Id; # I
# sites between first & last sites
for itW in (2:L-1)
    W[itW] = zeros(Complex{Float64},4,2,4,2); # ordering: left bottom right top
    W[itW][1,:,1,:] = Id; # I
    W[itW][2,:,1,:] = sqrt(2)*S[:,2,:]; # S_itW^x
    W[itW][3,:,1,:] = sqrt(2)*S[:,1,:]; # S_itW^y
    W[itW][4,:,2,:] = -t*sqrt(2)*S[:,1,:]; # t*S_itW^x
    W[itW][4,:,3,:] = -t*sqrt(2)*S[:,2,:]; # t*S_itW^y
    W[itW][4,:,4,:] = Id; # I
end
# last site
W[L] = zeros(Complex{Float64},4,2,1,2); # ordering: left bottom right top
W[L][1,:,1,:] = Id; # I
W[L][2,:,1,:] = sqrt(2)*S[:,2,:]; # S_L^x
W[L][3,:,1,:] = sqrt(2)*S[:,1,:]; # S_L^y

Eg_MPOF = reshape([1],(1,1,1));
for itL in (1:L)
    global Eg_MPOF
    Eg_MPOF = updateLeft(Eg_MPOF,3,MPS[itL],W[itL],4,MPS[itL]);
end

@printf("Ground state energy of Fermionic chain: %.4f\n", Eg_MPOF[1])

diff = Eg_MPO[1] - Eg_MPOF[1];
@printf("Difference between Eg_MPO and Eg_MPOF: %.4f\n", diff)
```

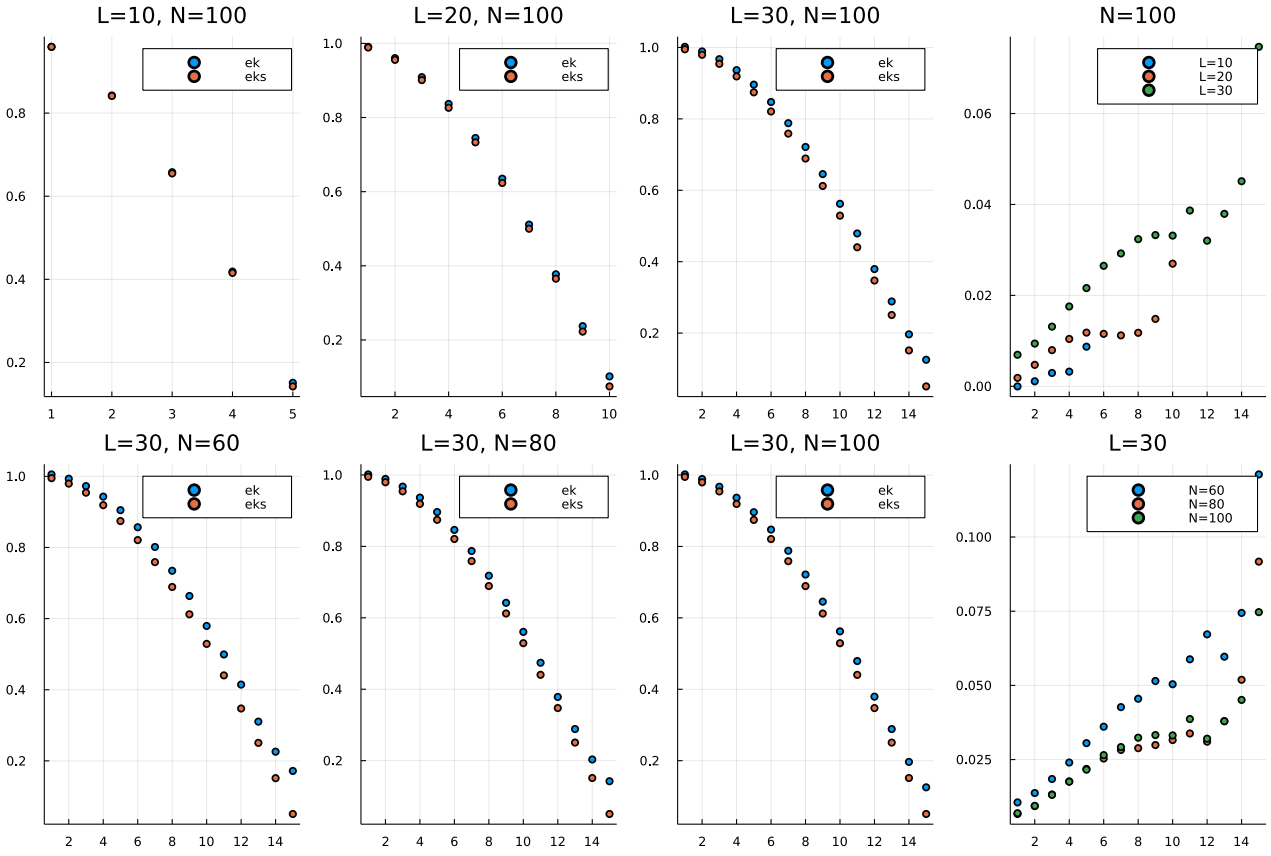
(iii) *MPO for one-particle eigenstates.* We obtain the ground state $|\Psi_{\rm MPS}\rangle$ of $H_{\rm ff}$ and construct the MPO for c_k as follows. The energies e_k, $\varepsilon_k = \cos\left[\ell k\pi/(\mathscr{L}+1)\right]$ and their difference $(e_k - \varepsilon_k)$ are plotted for $k = 1, \ldots, 25 (= \mathscr{L}/2)$. We observe that the difference increases for increasing $\mathscr{L}$ and fixing $N_{\rm keep}$ and decreases for increasing $N_{\rm keep}$ and fixing $\mathscr{L}$.

```
using LinearAlgebra
using Printf
using JLD2
using Plots

include("Tensor.jl");


# Iterative Diagonalization of Free Fermion
function diagFreeFermion(L,Nkeep)

global Hprev, Eg_Iter, MPS;

t = 0.5;
F,Z,Id = getLocalSpace("Fermion");

# MPS of the GS
MPS = Array{Any}(undef,1,L);

# initialization
H0 = Id*0; # Hamiltonian for only the 1st site
A0 = getIdentity(1,2,Id,2); # 1st leg is dummy leg (vacuum)
Hnow = H0
D,V = eigen((Hnow+Hnow')/2)
```

```
MPS[1] = contract(A0,3,3,V,2,1)
Hprev = diagm(D)
Eg_Iter = nothing

for itL in (2:L)

    # fermion aniihilation operator at the current site
    Fprev = updateLeft([],[],MPS[itL-1],F,3,MPS[itL-1])
    Anow = getIdentity(Hprev,2,Id,2)
    Hnow = updateLeft(Hprev,2,Anow,[],[],Anow)
    # free fermion hopping
    Hhop = t * updateLeft(Fprev,3,Anow,permutedims(F,(3,2,1)),3,Anow);
    Hhop = Hhop + Hhop';
    Hnow = Hnow + Hhop;
    # diagonalize the current Hamiltonian
    D,V = eigen((Hnow+Hnow')/2)
    # sort eigenvalues & eigenvectors in the order of increasing
    # eigenvalues
    ids = sortperm(D)
    D = D[ids]
    V = V[:,ids]

    if itL == L; Eg_Iter = minimum(D); end

    # truncation threshold for energy
    Etr = D[min(length(D),Nkeep)]
    oks = (D .< Etr)
    # true: to keep; false: not to keep
    # keep all degenerate states up to tolerance

    if itL < L
        MPS[itL] = contract(Anow,3,3,V[:,oks],2,1)
    else
        MPS[itL] = contract(Anow,3,3,reshape(V[:,1],(size(V[:,1])[1],1)),2,1) #
            select GS at the last step
    end
    Hprev = diagm(D[oks])

end

return Eg_Iter

end



# MPO Method for Free Fermion
function mpoFreeFermion(L)

global Eg_MPOF, W

t = 0.5;
F,Z,Id = getLocalSpace("Fermion");

W = Array{Any}(undef,1,L); # MPO W
# first site
W[1] = zeros(1,2,4,2); # ordering: left bottom right top
W[1][1,:,2,:] = t*F[:,1,:]'; # t*F'
W[1][1,:,3,:] = t*F[:,1,:]; # t*F
W[1][1,:,4,:] = Id; # I
# sites between first & last sites
for itW in (2:L-1)
    W[itW] = zeros(4,2,4,2); # ordering: left bottom right top
    W[itW][1,:,1,:] = Id; # I
    W[itW][2,:,1,:] = F[:,1,:]; # F
    W[itW][3,:,1,:] = F[:,1,:]'; # F'
    W[itW][4,:,2,:] = t*F[:,1,:]'; # t*F'
    W[itW][4,:,3,:] = t*F[:,1,:]; # t*F
    W[itW][4,:,4,:] = Id; # I
end
# last site
W[L] = zeros(4,2,1,2); # ordering: left bottom right top
W[L][1,:,1,:] = Id; # I
W[L][2,:,1,:] = F[:,1,:]; # F
W[L][3,:,1,:] = F[:,1,:]'; # F'

```

```
Eg_MPOF = reshape([1],(1,1,1));
for itL in (1:L)
    Eg_MPOF = updateLeft(Eg_MPOF,3,MPS[itL],W[itL],4,MPS[itL]);
end

return Eg_MPOF[1]

end


# ek for Free Fermion
function ekFreeFermion(k,L)

global ckMPS, Egk

F,Z,Id = getLocalSpace("Fermion");

Wk = Array{Any}(undef,1,L); # MPO W
# first site
Wk[1] = zeros(1,2,2,2); # ordering: left bottom right top
Wk[1][1,:,1,:] = sqrt(2/(L+1))*sin(1*k*pi/(L+1))*F[:,1,:]';
Wk[1][1,:,2,:] = Id;
# sites between first & last sites
for itW in (2:L-1)
    Wk[itW] = zeros(2,2,2,2); # ordering: left bottom right top
    Wk[itW][1,:,1,:] = Z; # I
    Wk[itW][2,:,1,:] = sqrt(2/(L+1))*sin(itW*k*pi/(L+1))*F[:,1,:]';
    Wk[itW][2,:,2,:] = Id;
end
# last site
Wk[L] = zeros(2,2,1,2); # ordering: left bottom right top
Wk[L][1,:,1,:] = Z; # I
Wk[L][2,:,1,:] = sqrt(2/(L+1))*sin(L*k*pi/(L+1))*F[:,1,:]';

ckMPS = Array{Any}(undef,1,L);
Egk = reshape([1],(1,1,1));
for itL in (1:L)
    ckMPS[itL] = contract(Wk[itL],4,4,MPS[itL],3,2,[1 4 2 3 5]);
    ckMPS[itL] = reshape(ckMPS[itL],
        (size(Wk[itL],1)*size(MPS[itL],1),
        size(Wk[itL],2),
        size(Wk[itL],3)*size(MPS[itL],3)));
    Egk = updateLeft(Egk,3,ckMPS[itL],W[itL],4,ckMPS[itL]);
end

return Egk[1] - Eg_Iter

end


# make comparisons
L = 30
Nkeep = 100;
rangeL = 10:10:30;
rangeN = 60:20:100;

ekL = Array{Any}(undef,1,length(rangeL));
ekN = Array{Any}(undef,1,length(rangeN));

for itL in 1:length(rangeL)
    Eg_Iter = diagFreeFermion(rangeL[itL],Nkeep);
    Eg_MPOF = mpoFreeFermion(rangeL[itL]);
    @printf("Eg_Iter - Eg_MPOF = %.4f\n", Eg_Iter - Eg_MPOF);
    ekL[itL] = zeros(rangeL[itL]/2)
    for itk in 1:rangeL[itL]/2
        ekL[itL][itk] = ekFreeFermion(itk,rangeL[itL]);
        @printf("  k = %i, ek - epsk = %.4f\n", itk,
            ekL[itL][itk] - cos(itk*pi/(rangeL[itL]+1)))
    end
end


for itN in 1:length(rangeN)
    Eg_Iter = diagFreeFermion(L,rangeN[itN]);
    Eg_MPOF = mpoFreeFermion(L);
    @printf("Eg_Iter - Eg_MPOF = %.4f\n", Eg_Iter - Eg_MPOF);
```

```
    ekN[itN] = zeros(L/2)
    for itk in 1:L/2
        ekN[itN][itk] = ekFreeFermion(itk,L);
        @printf("  k = %i, ek - epsk = %.4f\n", itk,
            ekN[itN][itk] - cos(itk*pi/(L+1)))
    end
end


# plot the results
pL = Array{Any}(undef,1,length(ekL)+1);
pN = Array{Any}(undef,1,length(ekN)+1);
diffL = Array{Any}(undef,1,length(ekL)+1);
diffN = Array{Any}(undef,1,length(ekN)+1);

for itL = 1:length(ekL)
    L = length(ekL[itL])*2; N = 100;
    exact = cos.(collect(1:length(ekL[itL])).*pi ./(length(ekL[itL])*2+1));
    pL[itL] = scatter([1:length(ekL[itL]),1:length(ekL[itL])],
        [ekL[itL],exact], labels=["ek" "eks"], title="L=$L, N=$N")
    diffL[itL] = ekL[itL] - exact;
end
L = length(ekL[1])*2; N = 100;
pL[end] = scatter(1:length(ekL[1]),diffL[1],label="L=$L",title="N=$N")
for itL = 2:length(ekL)
    L = length(ekL[itL])*2; N = 100;
    scatter!(1:length(ekL[itL]),diffL[itL],label="L=$L",title="N=$N")
end

for itN = 1:length(ekN)
    L = 30; N = rangeN[itN];
    exact = cos.(collect(1:length(ekN[itN])).*pi ./(length(ekN[itN])*2+1));
    pN[itN] = scatter([1:length(ekN[itN]),1:length(ekN[itN])],
        [ekN[itN],exact], labels=["ek" "eks"], title="L=$L, N=$N")
    diffN[itN] = ekN[itN] - exact;
end
L = 30; N = rangeN[1];
pN[end] = scatter(1:length(ekN[1]),diffN[1],title="L=$L",label="N=$N")
for itN = 2:length(ekN)
    L = 30; N = rangeN[itN];
    scatter!(1:length(ekN[itN]),diffN[itN],title="L=$L",label="N=$N")
end

output = plot(pL..., pN..., layout = (2,4), size = (1200,800))
savefig(output,"output.pdf")
```



[Total number of subtasks for T03: 17]