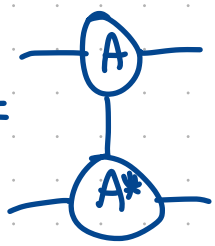
10.1 $A_{1,:,:} = \sqrt{\frac{2}{3}}\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$, $A_{0,:,:} = \sqrt{\frac{2}{3}}\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$, $A_{-1,:,:} = \sqrt{\frac{2}{3}}\begin{bmatrix} 0 & 0 \\ -1 & 0 \end{bmatrix}$

all entries of $A_{\theta,:,:}$ are real here

(a)

$-E=$ 

$$= \sum_{\theta\in\{1,0,-1\}} A_{\theta,:,:} \otimes A^*_{\theta,:,:} = \sum_{\theta\in\{0,1,-1\}} A_{\theta,:,:} \otimes A_{\theta,:,:}$$

$$- A_{1,:,:} \otimes A_{1,:,:} = \sqrt{\frac{2}{3}}\begin{bmatrix} 0 & \sqrt{\frac{2}{3}}\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \\ 0 & 0 \end{bmatrix} = \frac{2}{3}\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

8/10

$$- A_{0,:,:} \otimes A_{0,:,:} = \sqrt{\frac{1}{3}}\begin{bmatrix} -\sqrt{\frac{1}{3}}\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} & 0 \\ 0 & \sqrt{\frac{1}{3}}\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$- A_{-1,:,:} \otimes A_{-1,:,:} = \sqrt{\frac{2}{3}}\begin{bmatrix} 0 & 0 \\ \sqrt{\frac{2}{3}}\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} & 0 \end{bmatrix} = \frac{2}{3}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\Rightarrow E = \frac{2}{3}\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \frac{1}{3}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \frac{2}{3}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 2 & 0 & 0 & 1 \end{bmatrix}$$

- Spectral decomposition: λ_i s.t.

$$\det(\lambda I_4 - E) = \det\left(\begin{bmatrix} \lambda-1/3 & 0 & 0 & -2/3 \\ 0 & \lambda+1/3 & 0 & 0 \\ 0 & 0 & \lambda+1/3 & 0 \\ -2/3 & 0 & 0 & \lambda-1/3 \end{bmatrix}\right)$$

(*) $\lambda_{1/2} = \frac{\frac{2}{3} \pm \sqrt{\frac{4}{9} - 4\cdot 1\cdot(-\frac{1}{3})}}{2\cdot 1} = \frac{1}{3} \pm \frac{1}{2}\sqrt{\frac{4}{9}+\frac{12}{9}} = \frac{1}{3} \pm \frac{1}{2}\sqrt{\frac{16}{9}} = \frac{1}{3} \pm \frac{1}{2}\cdot\frac{4}{3} = \frac{1}{3} \pm \frac{2}{3} \Rightarrow \lambda_1 = -\frac{1}{3}, \lambda_2 = 1$

$$= (\lambda - \tfrac{1}{3})\det\left(\begin{bmatrix} \lambda+1/3 & 0 & 0 \\ 0 & \lambda+1/3 & 0 \\ 0 & 0 & \lambda-1/3 \end{bmatrix}\right) + (-\tfrac{2}{3})\det\left(\begin{bmatrix} 0 & 0 & -2/3 \\ \lambda+1/3 & 0 & 0 \\ 0 & \lambda+1/3 & 0 \end{bmatrix}\right)$$

$$= (\lambda-1/3)^2(\lambda+1/3)^2 - \tfrac{2}{3}\left(-\tfrac{2}{3}(\lambda+\tfrac{1}{3})^2\right) = (\lambda+\tfrac{1}{3})^2\left((\lambda-\tfrac{1}{3})^2 - \tfrac{2}{3}\right)$$

$$= (\lambda+\tfrac{1}{3})^2\left(\lambda^2 - \tfrac{2}{3}\lambda + \tfrac{1}{9} - \tfrac{4}{9}\right) = (\lambda+\tfrac{1}{3})^2(\lambda^2 - \tfrac{2}{3}\lambda - \tfrac{1}{3})$$

(*) $= (\lambda+\frac{1}{3})^3(\lambda-1)$ follows largest evalue is $\lambda = 1$

↝ argue with variat. form

— $Ev = \lambda v \Leftrightarrow Ev - \lambda v = 0 \Leftrightarrow (E - I\lambda)v = 0$

• $\lambda = -\frac{1}{3}$: ✓

$$\underbrace{\frac{1}{3}\begin{bmatrix}1&0&0&2\\0&-1&0&0\\0&0&-1&0\\2&0&0&1\end{bmatrix}}_{E} + \underbrace{\frac{1}{3}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}}_{-\lambda I} = \frac{1}{3}\begin{bmatrix}2&0&0&2\\0&0&0&0\\0&0&0&0\\2&0&0&2\end{bmatrix}$$

solve LSE: $\begin{bmatrix}2/3&0&0&2/3\\0&0&0&0\\0&0&0&0\\2/3&0&0&2/3\end{bmatrix} \overset{(-1)}{\rightsquigarrow} \begin{bmatrix}2/3&0&0&2/3\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix} \Leftrightarrow \frac{2}{3}x_1 + \frac{2}{3}x_4 = 0$

$\Leftrightarrow \frac{2}{3}x_1 = -\frac{2}{3}x_4 \Leftrightarrow x_1 = -x_4 \Rightarrow \mathbb{L} = \langle [1\ 0\ 0\ -1]^T, e_2, e_3 \rangle$

$\Rightarrow$ Take $v_1 = [1\ 0\ 0\ -1]^T$, $v_2 = [1\ 1\ 0\ -1]^T$, $v_3 = [1\ 0\ 1\ -1]^T$

• $\lambda = 1$: ✓

$$\frac{1}{3}\begin{bmatrix}1&0&0&2\\0&-1&0&0\\0&0&-1&0\\2&0&0&1\end{bmatrix} - \begin{bmatrix}1&&&\\&1&&\\&&1&\\&&&1\end{bmatrix} = \frac{1}{3}\begin{bmatrix}-2&&&-1\\&-4&&\\&&-4&\\-1&&&-2\end{bmatrix}$$

solve LSE: $\begin{bmatrix}-2/3&0&0&2/3\\0&-4/3&0&0\\0&0&-4/3&0\\2/3&0&0&-4/3\end{bmatrix} \overset{1}{\rightsquigarrow} \begin{bmatrix}-2/3&0&0&2/3\\0&-4/3&0&0\\0&0&-4/3&0\\0&0&0&0\end{bmatrix} \Leftrightarrow$ $-\frac{2}{3}x_1 = -\frac{2}{3}x_4$, $x_2 = x_3 = 0$

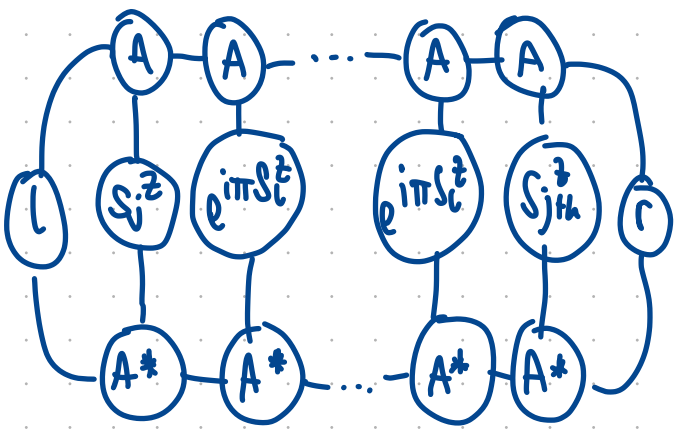
$\Rightarrow v_4 = [1\ 0\ 0\ 1]^T$

$\Rightarrow$

$$Q = \begin{matrix}v_1\ v_2\ v_3\ v_4\end{matrix}\begin{bmatrix}1&1&1&1\\0&1&0&0\\0&0&1&0\\-1&-1&-1&1\end{bmatrix}, \quad \Lambda = \begin{bmatrix}-1/3&0&0&0\\0&-1/3&0&0\\0&0&-1/3&0\\0&0&0&1\end{bmatrix}$$

in spectral Decomposition $Q\Lambda Q^{-1}$

(b)



✓

4/5

Matrix product operators (MPOs) for representing quantum Hamiltonians

In [1]:
```
import numpy as np
from scipy import sparse
```

Construct MPOs

In [2]:
```
X = np.array([[0, 1], [1, 0]])  # Pauli-X matrix
Z = np.array([[1, 0], [0, -1]])  # Pauli-Z matrix
I2 = np.eye(2)  # 2D identity
zero = np.zeros((2, 2))  # 2x2 zeros

def construct_ising_hamiltonian_mpo(J, g, L, pbc=False):
    """
    Construct Ising Hamiltonian on a 1D lattice with `L` sites as MPO,
    for interaction parameter `J` and external field parameter `g`.
    """
    # TODO: implement this function (you can ignore the case pbc=True for n
    As = (
        [np.transpose(np.array([[-g*X, -J*Z, I2]]), (2, 3, 0, 1))] +
        [np.transpose(np.array([
            [I2, zero, zero],
            [Z, zero, zero],
            [-g*X, -J*Z, I2]
        ]), (2, 3, 0, 1)) for _ in range(1, L-1)] +
        [np.transpose(np.array([[I2, Z, -g*X]]), (2, 3, 1, 0))]
    )
    return As
```

In [3]:
```
Alist_ising = construct_ising_hamiltonian_mpo(1.1, 0.7, 5)
Alist_ising[0].shape
```

Out[3]:
```
(2, 2, 1, 3)
```

In [4]:
```
# check dimensions (should return True)
Alist_ising[0].shape == (2, 2, 1, 3)
```

Out[4]:
```
True
```

In [5]:
```
# check dimensions (should return True)
Alist_ising[1].shape == (2, 2, 3, 3)
```

Out[5]:
```
True
```

In [6]:
```
# check dimensions (should return True)
Alist_ising[-1].shape == (2, 2, 3, 1)
```

Out[6]:
```
True
```

In [7]:
```
# example
Alist_ising[1]
```

Out[7]:
```
array([[[[ 1. ,  0. ,  0. ],
         [ 1. ,  0. ,  0. ],
         [-0. , -1.1,  1. ]],

        [[ 0. ,  0. ,  0. ],
         [ 0. ,  0. ,  0. ],
         [-0.7, -0. ,  0. ]]],


       [[[ 0. ,  0. ,  0. ],
         [ 0. ,  0. ,  0. ],
         [-0.7, -0. ,  0. ]],

        [[ 1. ,  0. ,  0. ],
         [-1. ,  0. ,  0. ],
         [-0. ,  1.1,  1. ]]]])
```

In [8]:
```
def construct_cluster_hamiltonian_mpo(J, L):
    """
    Construct the cluster state Hamiltonian as MPO.
    """
    # TODO: implement this function
    As = (
        [np.transpose(np.array([[zero, zero, -J*Z, I2]]), (2, 3, 0, 1))] +
        [np.transpose(np.array([
            [I2, zero, zero, zero],
            [Z, zero, zero, zero],
            [zero, X, zero, zero],
            [zero, zero, -J*Z, I2],
        ]), (2, 3, 0, 1)) for _ in range(1, L-1)] +
        [np.transpose(np.array([[I2, Z, zero, zero]]), (2, 3, 1, 0))]
    )
    assert len(As) == L
    return As
```

In [9]:
```
Alist_cluster = construct_cluster_hamiltonian_mpo(0.9, 5)
```

In [10]:
```
# example: show dimensions
Alist_cluster[0].shape
```

Out[10]:
```
(2, 2, 1, 4)
```

In [11]:
```
# example: show dimensions
Alist_cluster[1].shape
```

Out[11]:
```
(2, 2, 4, 4)
```

Utility functions

In [12]:
```
def mpo_to_full_tensor(Alist):
    """
    Construct the full tensor corresponding to the MPO tensors `Alist`.

    The i-th MPO tensor Alist[i] is expected to have dimensions (m[i], n[i]
    with `m` and `n` the list of logical dimensions and `D` the list of vir

    The returned tensor has dimensions m[0] x ... x m[L-1] x n[0] x ... x n

    Note: Should only be used for debugging and testing.
    """
    # consistency check
```

```
    assert Alist[0].ndim == 4
    # use leftmost virtual bond as first dimension
    T = np.transpose(Alist[0], (2, 0, 1, 3))
    # contract virtual bonds
    for i in range(1, len(Alist)):
        T = np.tensordot(T, Alist[i], axes=(-1, 2))
    # contract leftmost and rightmost virtual bond (has no influence if the
    assert T.shape[0] == T.shape[-1]
    T = np.trace(T, axis1=0, axis2=-1)
    # now T has dimensions m[0] x n[0] x m[1] x n[1] ... m[d-1] x n[d-1];
    # as last step, we group the `m` dimensions together, and likewise the
    T = np.transpose(T, list(range(0, T.ndim, 2)) + list(range(1, T.ndim, 2
    return T
```

In [13]:
```
# example
mpo_to_full_tensor([np.random.randn(3, 4, 1, 5), np.random.randn(7, 2, 5, 3
```

Out[13]:
```
(3, 7, 6, 4, 2, 5)
```

Construct quantum Hamiltonian as sparse matrix (as reference)

Transverse-field Ising Hamiltonian

In [14]:
```
def adjacency_1D_lattice(L, pbc=True):
    """
    Construct the adjacency matrix for a 1D lattice with `L` sites.
    The optional parameter `pbc` specifies whether periodic boundary condit
    should be used.
    """
    assert L > 1
    # special case
    if L == 2:
        return np.array([[0, 1], [1, 0]])
    if pbc:
        # periodic boundary conditions
        return np.roll(np.identity(L, dtype=int), -1, axis=0) + np.roll(np.
    else:
        # open boundary conditions
        return np.diag(np.ones(L - 1, dtype=int), k=-1) + np.diag(np.ones(L
```

In [15]:
```
# should be symmetric
np.linalg.norm(adjacency_1D_lattice(7) - adjacency_1D_lattice(7).T)
```

Out[15]:
```
0.0
```

In [16]:
```
# each site should have 2 neighbors (for periodic boundary conditions)
np.sum(adjacency_1D_lattice(7), axis=0)
```

Out[16]:
```
array([2, 2, 2, 2, 2, 2, 2])
```

In [17]:
```
# example
adjacency_1D_lattice(5, pbc=False)
```

Out[17]:
```
array([[0, 1, 0, 0, 0],
       [1, 0, 1, 0, 0],
       [0, 1, 0, 1, 0],
       [0, 0, 1, 0, 1],
       [0, 0, 0, 1, 0]])
```

In [18]:
```
# Note: this is a solution of Exercise 9.2 (b)
def construct_ising_hamiltonian_sparse(J, g, adj):
    """
    Construct Ising Hamiltonian as sparse matrix,
    for interaction parameter `J` and external field parameter `g`.
    `adj` is the adjacency matrix of the underlying lattice.
    """
    # Pauli-X and Z matrices
    X = sparse.csr_matrix([[0., 1.], [1.,  0.]])
    Z = sparse.csr_matrix([[1., 0.], [0., -1.]])
    # overall number of lattice sites
    L = adj.shape[0]
    H = sparse.csr_matrix((2**L, 2**L), dtype=float)
    for j in range(L):
        for k in range(j+1, L):
            if adj[j, k] > 0:
                H -= J * sparse.kron(sparse.eye(2**j),
                           sparse.kron(Z,
                           sparse.kron(sparse.eye(2**(k-j-1)),
                           sparse.kron(Z,
                                       sparse.eye(2**(L-k-1))))))
    # external field
    for j in range(L):
        H -= g * sparse.kron(sparse.eye(2**j), sparse.kron(X, sparse.eye(2*
    return H
```

In [19]:
```
adj = adjacency_1D_lattice(5, pbc=False)
Hising = construct_ising_hamiltonian_sparse(1.1, 0.7, adj)
Hising
```

Out[19]:
```
<32x32 sparse matrix of type '<class 'numpy.float64'>'
        with 180 stored elements in Compressed Sparse Row format>
```

In [20]:
```
# convert to NumPy array to show entries
Hising.toarray()
```

Out[20]:
```
array([[-4.4, -0.7, -0.7, ...,  0. ,  0. ,  0. ],
       [-0.7, -2.2,  0. , ...,  0. ,  0. ,  0. ],
       [-0.7,  0. ,  0. , ...,  0. ,  0. ,  0. ],
       ...,
       [ 0. ,  0. ,  0. , ...,  0. ,  0. , -0.7],
       [ 0. ,  0. ,  0. , ...,  0. , -2.2, -0.7],
       [ 0. ,  0. ,  0. , ..., -0.7, -0.7, -4.4]])
```

In [21]:
```
# compare (difference should be zero)
np.linalg.norm(Hising.toarray() - np.reshape(mpo_to_full_tensor(Alist_ising
```

Out[21]:
```
0.0
```

In [22]:
```
# periodic boundary conditions
adj = adjacency_1D_lattice(5, pbc=True)
Hising_per = construct_ising_hamiltonian_sparse(1.1, 0.7, adj)
Hising_per
```

Out[22]:
```
<32x32 sparse matrix of type '<class 'numpy.float64'>'
        with 192 stored elements in Compressed Sparse Row format>
```

In [23]:
```
# compare (difference should be zero) - this is only relevant for part (c)
np.linalg.norm(Hising_per.toarray() - np.reshape(mpo_to_full_tensor(constru
```

Out[23]:
```
6.222539674441618
```

Cluster state Hamiltonian

In [24]:
```
def construct_cluster_hamiltonian_sparse(J, L):
    """
    Construct the cluster state Hamiltonian as sparse matrix
    on a one-dimensional lattice with open boundary conditions.
    """
    # Pauli-X and Z matrices
    X = sparse.csr_matrix([[0., 1.], [1.,  0.]])
    Z = sparse.csr_matrix([[1., 0.], [0., -1.]])
    H = sparse.csr_matrix((2**L, 2**L), dtype=float)
    h = sparse.kron(sparse.kron(Z, X), Z)
    for j in range(L-2):
        H -= sparse.kron(sparse.eye(2**j),
             sparse.kron(h,
                         sparse.eye(2**(L-j-3))))
    return J*H
```

In [25]:
```
Hcluster = construct_cluster_hamiltonian_sparse(0.9, 5)
```

In [26]:
```
# compare (difference should be zero)
np.linalg.norm(Hcluster.toarray() - np.reshape(mpo_to_full_tensor(Alist_clu
```

Out[26]:
```
0.0
```

In [ ]:
```

```