(a) Different Pauli matrices do not commute, but "anti-commute" instead: one can verify by an explicit calculation that

$$XY = -YX, \quad YZ = -ZY, \quad ZX = -XZ.$$

Nevertheless, it turns out that $X \otimes X$, $Y \otimes Y$ and $Z \otimes Z$ pairwise commute. Prove this statement.

Hint: You can work directly with matrix representations, or combine the general identity $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$ with the anti-commuting property.

$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $Y = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$, $Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

$X \otimes X = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \hat{=} X_2$ $\quad Y \otimes Y = \begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{pmatrix} \hat{=} Y_2$ $\quad Z \otimes Z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \hat{=} Z_2$

$X_2 Y_2 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$ $\quad Y_2 X_2 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \rightarrow [X_2, Y_2] = 0$

$X_2 Z_2 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$ $\quad Z_2 X_2 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \rightarrow [X_2, Z_2] = 0$

$Y_2 Z_2 = \begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{pmatrix}$ $\quad Z_2 Y_2 = \begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{pmatrix} \rightarrow [Y_2, Z_2] = 0$

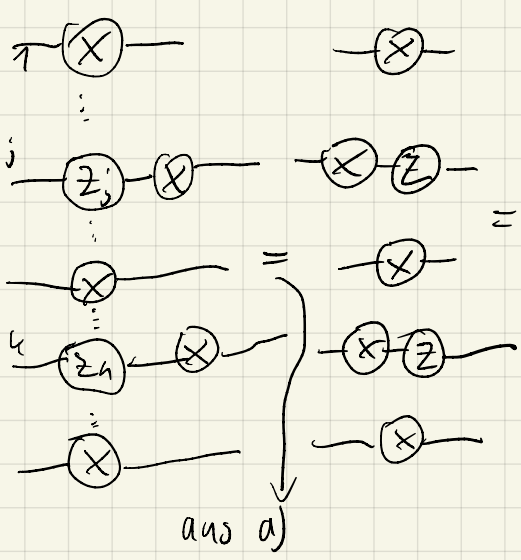
(b) A symmetry operator for the Ising model (see below) is

$$P = \prod_{j=1}^{L} X_j = X \otimes \cdots \otimes X, \qquad H = -J \sum_{\langle j,k \rangle} Z_j Z_k - g \sum_j X_j,$$

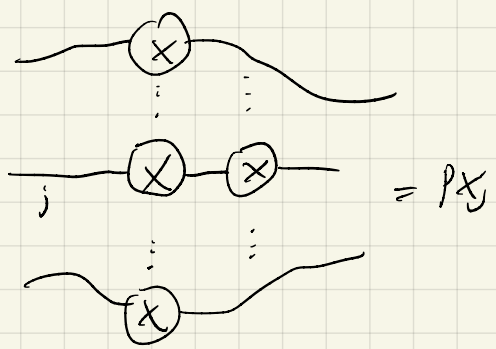
where L denotes the number of lattice sites. Show that $[H, P] = 0$.

$HP = \left(-J \sum_{\langle j,k \rangle} Z_j Z_k - g \sum_j X_j\right) \cdot P = \left(-J \sum_{\langle j,k \rangle} Z_j Z_k P - g \sum_j X_j P\right)$ ① ②

① $Z_j Z_k P =$



② $X_j P =$



$$\Rightarrow \quad = -J \sum_{\langle j,h \rangle} P z_j z_h - g \sum_j P x_j = P \cdot H \quad \Rightarrow [H, P] = 0$$

exercise9.2_template

June 23, 2022

1 Ising model

[1]:
```
import numpy as np
from scipy import sparse
import scipy.sparse.linalg as scila
import matplotlib.pyplot as plt  ##
```

1.1 Adjacency matrices for various lattices

[2]:
```
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 conditions
    should be used.
    """
    x = np.full(L - 1, 1)
    A = np.diag(x, -1) + np.diag(x, 1)

    if (pbc):
        A[0, L - 1] = 1
        A[L - 1, 0] = 1

    return A
```

[3]:
```
# should be symmetric
np.linalg.norm(adjacency_1D_lattice(7) - adjacency_1D_lattice(7).T)
```

[3]:
```
0.0
```

[4]:
```
# each site should have 2 neighbors (for periodic boundary conditions)
np.sum(adjacency_1D_lattice(7), axis=0)
```

[4]:
```
array([2, 2, 2, 2, 2, 2, 2])
```

[5]:
```
# test for periodic boundary conditions (difference should be zero)
np.linalg.norm(adjacency_1D_lattice(7) - np.array(
    [[0, 1, 0, 0, 0, 0, 1],
```

```
     [1, 0, 1, 0, 0, 0, 0],
     [0, 1, 0, 1, 0, 0, 0],
     [0, 0, 1, 0, 1, 0, 0],
     [0, 0, 0, 1, 0, 1, 0],
     [0, 0, 0, 0, 1, 0, 1],
     [1, 0, 0, 0, 0, 1, 0]]))
```

[5]: 0.0

[6]:
```
# test for open boundary conditions (difference should be zero)
np.linalg.norm(adjacency_1D_lattice(7, pbc=False) - np.array(
    [[0, 1, 0, 0, 0, 0, 0],
     [1, 0, 1, 0, 0, 0, 0],
     [0, 1, 0, 1, 0, 0, 0],
     [0, 0, 1, 0, 1, 0, 0],
     [0, 0, 0, 1, 0, 1, 0],
     [0, 0, 0, 0, 1, 0, 1],
     [0, 0, 0, 0, 0, 1, 0]]))
```

[6]: 0.0

[7]:
```
def adjacency_square_lattice(Lx, Ly, pbc=True):
    """
    Construct the adjacency matrix for a 2D square lattice with `Lx x Ly` sites.
    The optional parameter `pbc` specifies whether periodic boundary conditions
    should be used.
    """
    A = np.zeros([Lx * Ly, Lx * Ly])
    for i in range(Lx * Ly):
        for j in range(Lx * Ly):
            ix = i // Ly
            iy = i % Ly
            jx = j // Ly
            jy = j % Ly

            if ix == jx:
                if iy == jy + 1 or iy + 1 == jy:
                    A[i, j] = 1
                if pbc and ((iy == 0 and jy == Ly - 1) or (iy == Ly - 1 and jy␣
 ↪== 0)):
                    A[i, j] = 1
            elif iy == jy:
                if ix == jx + 1 or ix + 1 == jx:
                    A[i, j] = 1
                if pbc and ((ix == 0 and jx == Lx - 1) or (ix == Lx - 1 and jx␣
 ↪== 0)):
                    A[i, j] = 1
```

```
    return A
```

[8]:
```
# should be symmetric
np.linalg.norm(adjacency_square_lattice(3, 4) - adjacency_square_lattice(3, 4).
 ↪T)
```

[8]: 0.0

[9]:
```
# each site should have 4 neighbors (for periodic boundary conditions)
np.sum(adjacency_square_lattice(3, 4), axis=0)
```

[9]: array([4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4.])

[10]:
```
# test for periodic boundary conditions (difference should be zero)
np.linalg.norm(adjacency_square_lattice(3, 4) - np.array(
    [[0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0],
     [1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0],
     [0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0],
     [1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1],
     [1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0],
     [0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0],
     [0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0],
     [0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1],
     [1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1],
     [0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0],
     [0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1],
     [0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0]]))
```

[10]: 0.0

[11]:
```
# test for open boundary conditions (difference should be zero)
np.linalg.norm(adjacency_square_lattice(3, 4, pbc=False) - np.array(
    [[0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
     [1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0],
     [0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0],
     [0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0],
     [1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0],
     [0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0],
     [0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0],
     [0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1],
     [0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0],
     [0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0],
     [0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1],
     [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0]]))
```

[11]: 0.0

1.2 Ising Hamiltonian construction

[12]:
```
def construct_ising_hamiltonian(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.
    """
    L = adj.shape[0]
    X = np.array([[0, 1], [1, 0]])
    Z = np.array([[1, 0], [0, -1]])
    H = np.zeros([2 ** L, 2 ** L])
    for j in range(L):
        for k in range(j + 1, L):
            if adj[j, k] > 0:
                H -= J * np.kron(np.identity(2 ** j), np.kron(Z, np.kron(np.
 ↪identity(2 ** (k - j - 1)),
                                                                          np.
 ↪kron(Z, np.identity(2 ** (L - k - 1))))))

        H -= g * np.kron(np.identity(2 ** j), np.kron(X, np.identity(2 ** (L -␣
 ↪j - 1))))

    return np.asmatrix(H)
```

[13]:
```
# example
adj = adjacency_1D_lattice(3, pbc=False)
H = construct_ising_hamiltonian(1.1, 0.7, adj)
H
```

[13]:
```
matrix([[-2.2, -0.7, -0.7,  0. , -0.7,  0. ,  0. ,  0. ],
        [-0.7,  0. ,  0. , -0.7,  0. , -0.7,  0. ,  0. ],
        [-0.7,  0. ,  2.2, -0.7,  0. ,  0. , -0.7,  0. ],
        [ 0. , -0.7, -0.7,  0. ,  0. ,  0. ,  0. , -0.7],
        [-0.7,  0. ,  0. ,  0. ,  0. , -0.7, -0.7,  0. ],
        [ 0. , -0.7,  0. ,  0. , -0.7,  2.2,  0. , -0.7],
        [ 0. ,  0. , -0.7,  0. , -0.7,  0. ,  0. , -0.7],
        [ 0. ,  0. ,  0. , -0.7,  0. , -0.7, -0.7, -2.2]])
```

[14]:
```
# convert to NumPy array to show entries
np.asarray(H)
```

[14]:
```
array([[-2.2, -0.7, -0.7,  0. , -0.7,  0. ,  0. ,  0. ],
       [-0.7,  0. ,  0. , -0.7,  0. , -0.7,  0. ,  0. ],
       [-0.7,  0. ,  2.2, -0.7,  0. ,  0. , -0.7,  0. ],
       [ 0. , -0.7, -0.7,  0. ,  0. ,  0. ,  0. , -0.7],
       [-0.7,  0. ,  0. ,  0. ,  0. , -0.7, -0.7,  0. ],
```

```
       [ 0. , -0.7,  0. ,  0. , -0.7,  2.2,  0. , -0.7],
       [ 0. ,  0. , -0.7,  0. , -0.7,  0. ,  0. , -0.7],
       [ 0. ,  0. ,  0. , -0.7,  0. , -0.7, -0.7, -2.2]])
```

[15]:
```
# test: this should give zero
np.linalg.norm(H - np.array(
    [[-2.2, -0.7, -0.7, 0., -0.7, 0., 0., 0.],
     [-0.7, 0., 0., -0.7, 0., -0.7, 0., 0.],
     [-0.7, 0., 2.2, -0.7, 0., 0., -0.7, 0.],
     [0., -0.7, -0.7, 0., 0., 0., 0., -0.7],
     [-0.7, 0., 0., 0., 0., -0.7, -0.7, 0.],
     [0., -0.7, 0., 0., -0.7, 2.2, 0., -0.7],
     [0., 0., -0.7, 0., -0.7, 0., 0., -0.7],
     [0., 0., 0., -0.7, 0., -0.7, -0.7, -2.2]]))
```

[15]:
```
0.0
```

1.3 Exemplary eigenvalues and -vectors

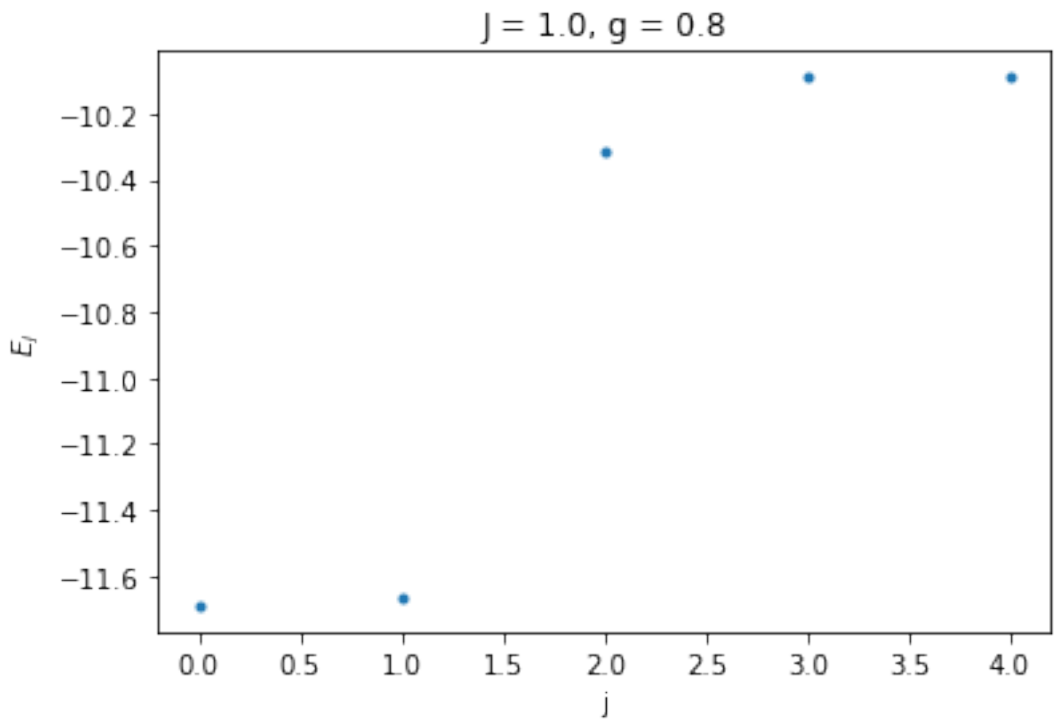
[16]:
```
L = 10
J = 1.0
g = 0.8
adj = adjacency_1D_lattice(L)
H = construct_ising_hamiltonian(J, g, adj)
H
```

[16]:
```
matrix([[-10. ,  -0.8,  -0.8, …,   0. ,   0. ,   0. ],
        [ -0.8,  -6. ,   0. , …,   0. ,   0. ,   0. ],
        [ -0.8,   0. ,  -6. , …,   0. ,   0. ,   0. ],
        …,
        [  0. ,   0. ,   0. , …,  -6. ,   0. ,  -0.8],
        [  0. ,   0. ,   0. , …,   0. ,  -6. ,  -0.8],
        [  0. ,   0. ,   0. , …,  -0.8,  -0.8, -10. ]])
```

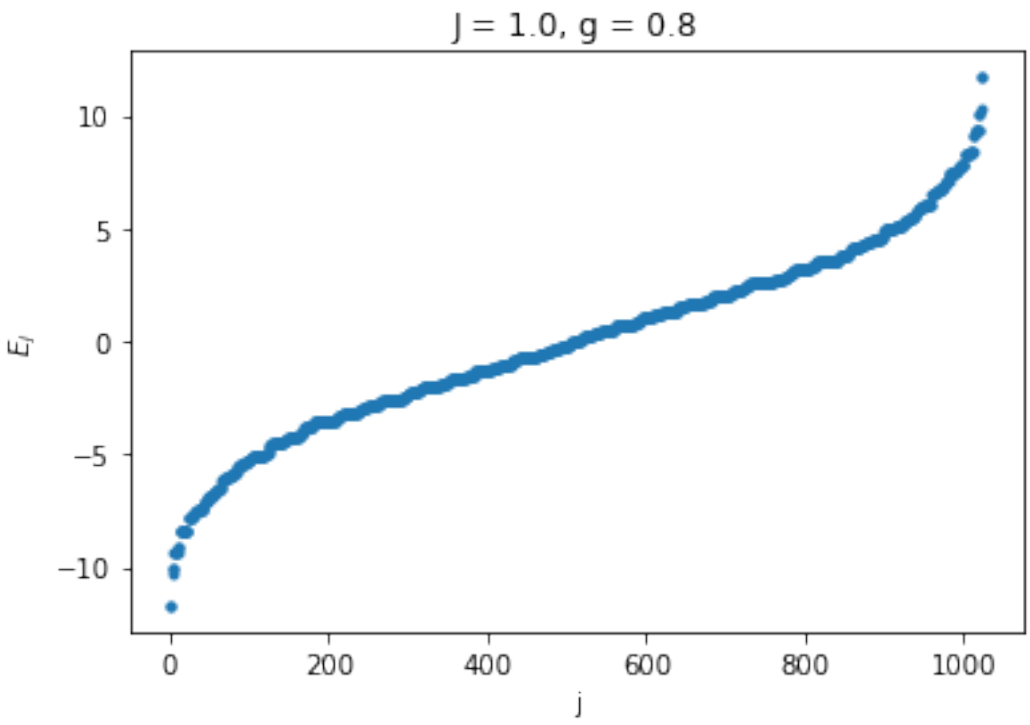
[17]:
```
# compute algebraically smallest few eigenvalues and corresponding eigenvectors
en,  = scila.eigsh(H, 5, which='SA')
print(en)
```

```
[-11.69093572 -11.66430728 -10.31508937 -10.08859846 -10.08859846]
```

[18]:
```
# visualize these eigenvalues
plt.plot(en, '.')
plt.xlabel("j")
plt.ylabel(r"$E_j$")
plt.title("J = {}, g = {}".format(J, g))
plt.show()
```



[19]:
```
# for comparison: plot full spectrum
plt.plot(np.linalg.eigvalsh(np.asarray(H)), '.')
plt.xlabel("j")
plt.ylabel(r"$E_j$")
plt.title("J = {}, g = {}".format(J, g))
plt.show()
```



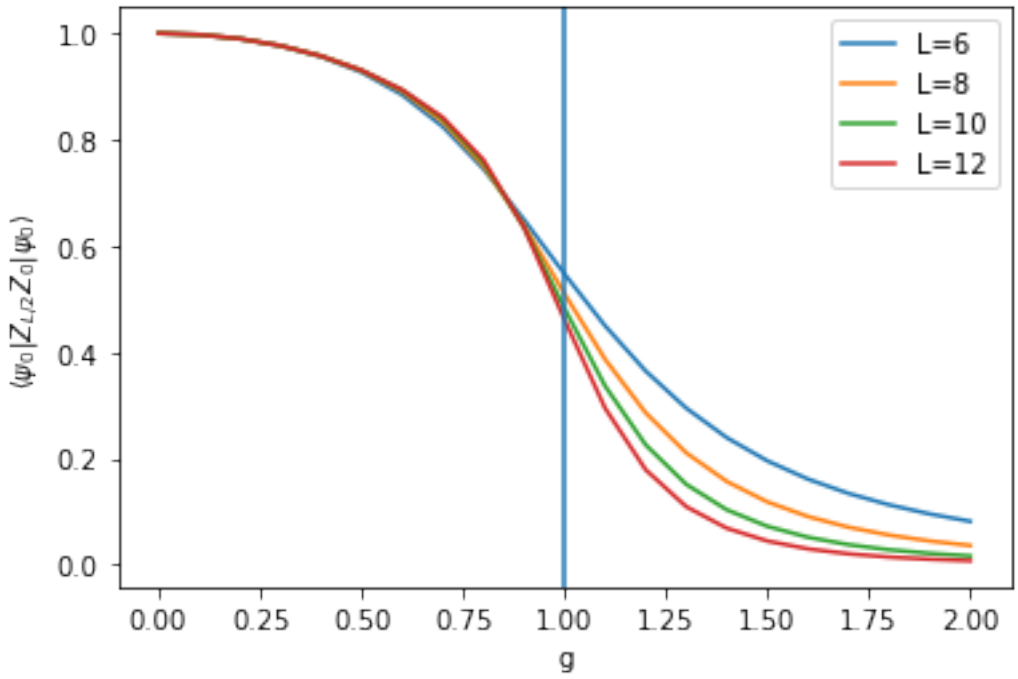
1.4 Spin-spin correlation function

[20]:
```
J = 1.0

# Pauli-Z matrix
Z = sparse.csr_matrix([[1., 0.], [0., -1.]])

Llist = np.array([6, 8, 10, 12])
glist = np.linspace(0, 2, 21)

C = np.zeros((len(Llist), len(glist)))

# iterate over various lattice sizes
for i, L in enumerate(Llist):
    print("L =", L)
    # spin operators (Pauli-Z) on site 0 and center site L/2
    # TODO: construct Z_0 and Z_c as sparse CSR matrices here, with c = L//2
    Z = np.array([[1, 0], [0, -1]])
    Z0 = np.kron(Z, np.identity(2 ** (L - 1)))
    Zc = np.kron(np.identity(2 ** (L // 2)), np.kron(Z, np.identity(2 ** (L -␣
 ↪(L // 2) - 1))))
    adj = adjacency_1D_lattice(L)
    # parameter sweep over `g`
```

```
    for j, g in enumerate(glist):
        H = construct_ising_hamiltonian(J, g, adj)
        en,   = scila.eigsh(H, 5, which='SA')
        # ground state
         0 =  [:, 0]
        # spin-spin correlation function
        C[i, j] = np.vdot( 0, Z0 @ (Zc @  0))
```

```
L = 6
L = 8
L = 10
L = 12
```

[21]:
```
# visualize correlation function
for i, L in enumerate(Llist):
    plt.plot(glist, C[i, :], label="L={}".format(L))
# Ising model exhibits a phase transition at g = 1 in the thermodynamic limit L␣
 ↪-> ∞
plt.axvline(x=1)
plt.legend()
plt.xlabel("g")
plt.ylabel(r"$\langle\psi_0|Z_{L/2} Z_0|\psi_0\rangle$")
plt.show()
```

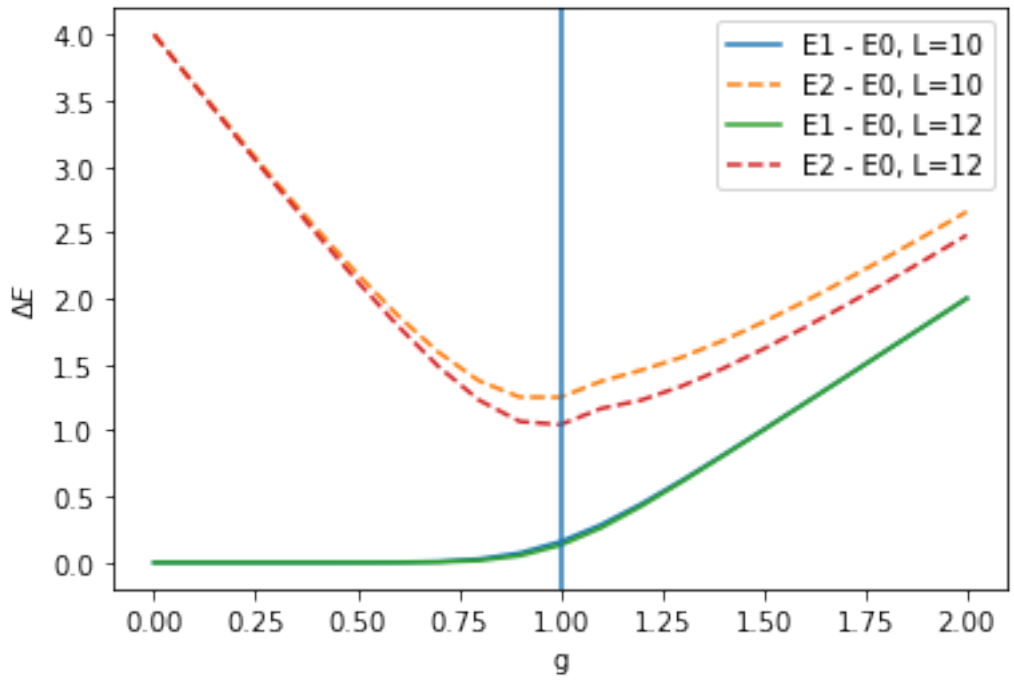


1.5 Excitation energies

[22]:
```
J = 1.0

Llist = np.array([10, 12])
glist = np.linspace(0, 2, 21)

# excitation energies
exc1 = np.zeros((len(Llist), len(glist)))
exc2 = np.zeros((len(Llist), len(glist)))

# iterate over various lattice sizes
for i, L in enumerate(Llist):
    print("L =", L)
    adj = adjacency_1D_lattice(L)
    # parameter sweep over `g`
    for j, g in enumerate(glist):
        H = construct_ising_hamiltonian(J, g, adj)
        en,  = scila.eigsh(H, 5, which='SA')
        en = np.sort(en)
        exc1[i, j] = en[1] - en[0]
        exc2[i, j] = en[2] - en[0]
```

```
L = 10
L = 12
```

[23]:
```
# visualize excitation energies
for i, L in enumerate(Llist):
    plt.plot(glist, exc1[i, :], label="E1 - E0, L={}".format(L))
    plt.plot(glist, exc2[i, :], '--', label="E2 - E0, L={}".format(L))
# Ising model exhibits a phase transition at g = 1 in the thermodynamic limit L␣
 ↪-> ∞
# Here we see that a "gap" opens at g = 1, i.e., E1 - E0 remains strictly␣
 ↪positive for g > 1.
plt.axvline(x=1)
plt.legend()
plt.xlabel("g")
plt.ylabel(r"$\Delta E$")
plt.show()
```


E1 - E0, L=10
E2 - E0, L=10
E1 - E0, L=12
E2 - E0, L=12
ΔE
g
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00