

## Exercise 4: Exact diagonalization

In this exercise, we consider the transverse field Ising model in 1D, given by the Hamiltonian

$$H = -J \sum_{j=0}^{L-1} \sigma_j^x \sigma_{j+1}^x - g \sum_{j=0}^{L-1} \sigma_j^z, \quad (1)$$

where we used periodic boundary conditions,  $\sigma_L^x \equiv \sigma_0^x$ . As discussed in the lecture, this model shows (for  $L \rightarrow \infty$ ) a quantum phase transition at  $g = 1$ . To see this transition, we need to find the groundstate while tuning  $g$  (keeping  $J \equiv 1$  as the unit of energy). The goal of this exercise is to represent eq. (1) as a sparse matrix, `scipy.sparse.csr_matrix` and diagonalize it with (a variant of) the Lanczos algorithm provided in `scipy`.

- a) Define the  $2 \times 2$  matrices

$$\text{Id} = \mathbb{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \text{Sx} = \sigma^x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \text{Sz} = \sigma^z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad (2)$$

as `scipy.sparse.csr_matrix`

- b) When we write  $\sigma_j^z$  in eq. (1), what we mean is

$$\sigma_j^z \equiv \mathbb{1} \otimes \cdots \otimes \mathbb{1} \otimes \sigma^z \otimes \mathbb{1} \otimes \cdots \otimes \mathbb{1} \quad (3)$$

where the  $\sigma^z$  is at the  $j^{\text{th}}$  position, and similar for  $\sigma_j^x$ . The operator  $\sigma_j^z$  corresponds thus to a  $2^L \times 2^L$  matrix. To implement the tensor product, you can use successive calls to `scipy.sparse.kron()`, e.g. to generate  $\mathbb{1} \otimes \mathbb{1} \otimes \sigma^z \otimes \mathbb{1}$ :

```
full = scipy.sparse.kron(Id, Id, format='csr')    # 1 1
full = scipy.sparse.kron(full, Sz, format='csr')  # 1 1 Sz
full = scipy.sparse.kron(full, Id, format='csr')  # 1 1 Sz 1
```

Write a function which returns (for given  $L$ ) a list, which contains a representation of  $\sigma_j^z$  (in the form of a `csr_matrix`) as  $j^{\text{th}}$  entry of the list.

- c) Write a similar function returning the  $\sigma_j^x$  operators.  
 d) Write a function `gen_hamiltonian(sx_list, sz_list, g, J)` generating the Hamiltonian as a `csr_matrix` of shape  $2^L \times 2^L$ , where the arguments `sx_list`, `sz_list` should be the lists generated by the functions defined in b) and c).

*Hint:* Addition and scalar multiplication of sparse matrices work as expected with `+` and `*`. For two `csr_matrix`, the code `A*B` gives the **matrix** product of `A` and `B`. (This is different from `numpy` arrays where `A*B` gives the element-wise product!) Also note  $A \otimes B = (A \otimes \mathbb{1})(\mathbb{1} \otimes B)$ , hence  $\sigma_j^x \sigma_{j+1}^x$  can be obtained by a matrix product of  $\sigma_j^x$  (in the sense of eq. (3)) with  $\sigma_{j+1}^x$ .

Check that the function works as expected, e.g., for  $L = 2$  and  $g = 0.1$ , you should obtain

$$H(L = 2, g = 0.1, J = 1) = \begin{pmatrix} -0.2 & 0 & 0 & -2 \\ 0 & 0 & -2 & 0 \\ 0 & -2 & 0 & 0 \\ -2 & 0 & 0 & 0.2 \end{pmatrix}. \quad (4)$$

*Hint:* You can convert the sparse  $H$  to a non-sparse numpy array with `H.toarray()` for the comparison. Remember to use periodic boundary conditions.

- e) When you have  $H$  as a `csr_matrix`, you can use the function `scipy.sparse.linalg.eigsh` to obtain the ground state. This function uses (an improved version of) the Lanczos method discussed in the lecture. Read the documentation to find out how to obtain the 3 eigenstates (and eigenvalues) with smallest (algebraic) eigenvalues.

*Optionally:* Compare the run time of `scipy.sparse.linalg.eigsh` with the full diagonalization performed by `np.linalg.eigh` (acting on a non-sparse numpy array) for  $L = 12$ .

- f) For system sizes  $L \in \{6, 8, 10, 12\}$ , calculate the ground state for  $\approx 20$  values of  $g \in [0, 2]$ . Calculate and plot the largest-distance spin-spin correlation  $C = \langle \psi_0 | \sigma_0^x \sigma_{L/2}^x | \psi_0 \rangle$  against  $g$  for the various system sizes.

*Hint:* To calculate  $C$ , apply the operators (= sparse matrices)  $\sigma_0^x$  and  $\sigma_{L/2}^x$  to the ground state using matrix-vector products (implemented as `op*v0`) and calculate the overlap of the result with the the ground state (e.g., using `np.inner`).

- g) Plot the excitation energies ( $E_i - E_0$ ,  $i = 1, 2$ ) for the first two excited states versus  $g$ . Does the result coincide with your expectations?