DMRG algorithm

Team members:

Aikun Bexultanova

Alexey Fokin

Daniil Volkov

Alexandra Volkova

Alexandr Voronin

Introduction: What is DMRG

The density matrix renormalization group (DMRG) is a numerical variational technique devised to obtain the low-energy physics of quantum many-body systems with high accuracy.

Invented in 1992 by Steven White

Objectives of the project:

- Implement DMRG for Heisenberg Hamiltonian
- Share the results and discuss

Heisenberg Hamiltonian

- in quantum mechanics it describes the interactions between particles with spin, helping researchers understand phenomena such as **ferromagnetism**, **antiferromagnetism**, and **spin waves** in magnetic materials and systems with strong spin interactions.

Heisenberg Hamiltonian for a Spin Chain:

$$H = -J \sum_{\langle i,j \rangle} (\sigma_i^x \sigma_j^x + \sigma_i^y \sigma_j^y + \sigma_i^z \sigma_j^z)$$

DMRG Key Features:

- Block Hamiltonian:

$$H_{\text{super}} = H_{\text{sys, enlarged}} \otimes \mathbb{I}_{\text{env, enlarged}} + \mathbb{I}_{\text{sys, enlarged}} \otimes H_{\text{env, enlarged}} + H_{\text{int, sys-env}}$$

- Reduced Density Matrix:

$$\rho_{\text{sys}} = \mathbf{Tr}_{\text{env}}(|\Psi\rangle\langle\Psi|)$$

- Eigenvalue Decomposition of Density Matrix:

$$\rho_{\text{sys}} = \sum_i \lambda_i |\phi_i\rangle\langle\phi_i|$$

- Truncation

Choosing m largest eigenvalues λ_i and corresponding $|\phi_i\rangle$ eigenvectors

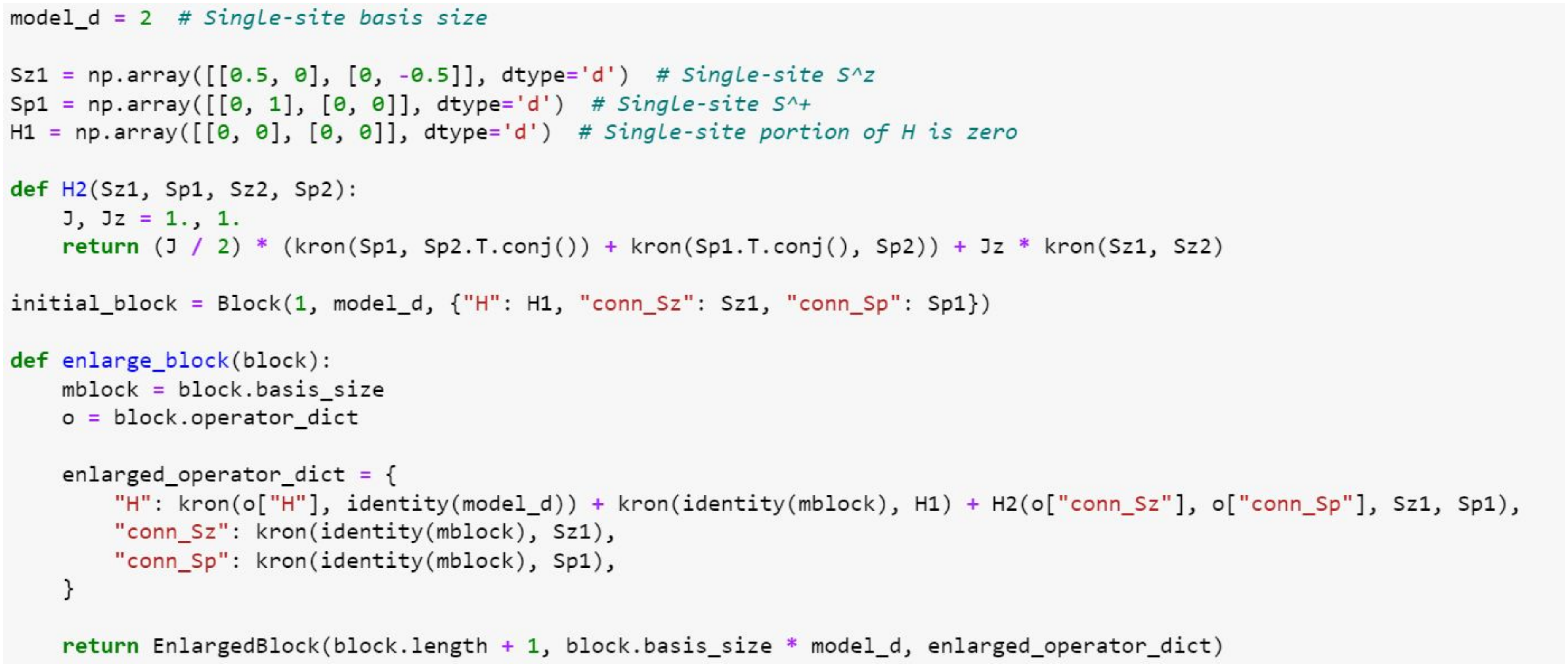
- Transformation of Operators

$$\tilde{O}_{\text{sys}} = U^\dagger O_{\text{sys}} U$$

Here, U is the matrix of eigenvectors $\{|\phi_i\rangle\}_{i=1}^{m}$

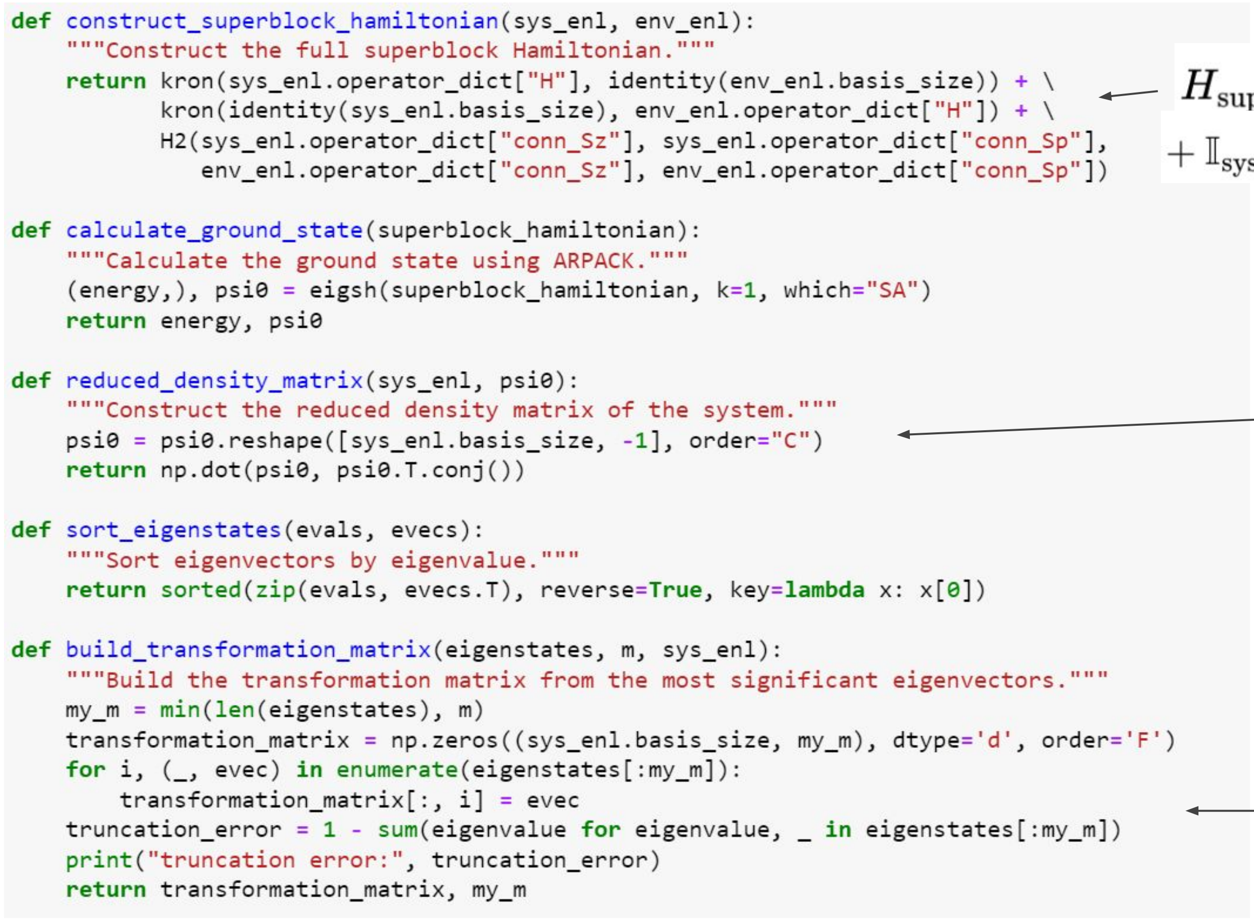
- Iteration and Sweeping

Implementation Details:



```python
model_d = 2  # Single-site basis size

Sz1 = np.array([[0.5, 0], [0, -0.5]], dtype='d')  # Single-site S^z
Sp1 = np.array([[0, 1], [0, 0]], dtype='d')  # Single-site S^+
H1 = np.array([[0, 0], [0, 0]], dtype='d')  # Single-site portion of H is zero

def H2(Sz1, Sp1, Sz2, Sp2):
    J, Jz = 1., 1.
    return (J / 2) * (kron(Sp1, Sp2.T.conj()) + kron(Sp1.T.conj(), Sp2)) + Jz * kron(Sz1, Sz2)

initial_block = Block(1, model_d, {"H": H1, "conn_Sz": Sz1, "conn_Sp": Sp1})

def enlarge_block(block):
    mblock = block.basis_size
    o = block.operator_dict

    enlarged_operator_dict = {
        "H": kron(o["H"], identity(model_d)) + kron(identity(mblock), H1) + H2(o["conn_Sz"], o["conn_Sp"], Sz1, Sp1),
        "conn_Sz": kron(identity(mblock), Sz1),
        "conn_Sp": kron(identity(mblock), Sp1),
    }

    return EnlargedBlock(block.length + 1, block.basis_size * model_d, enlarged_operator_dict)
```

Implementation Details:



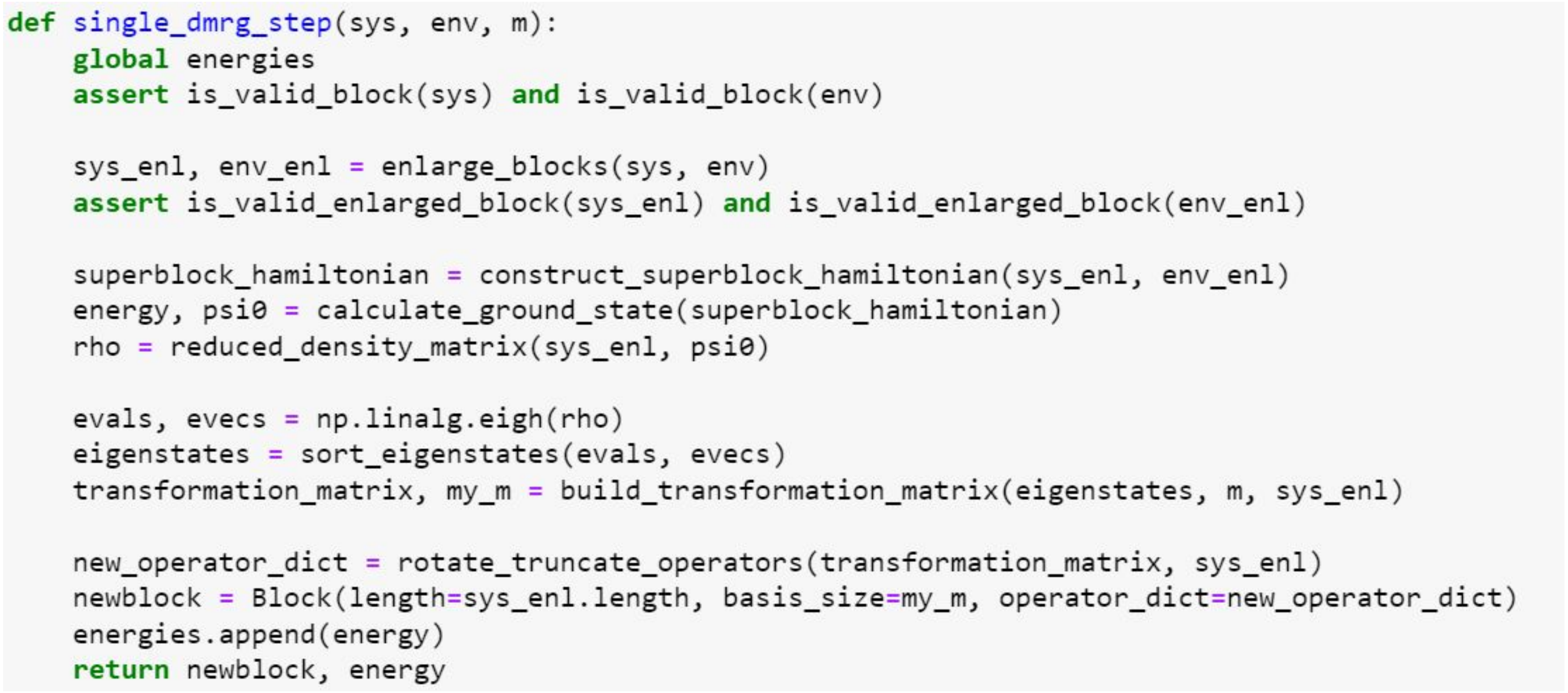
```python
def construct_superblock_hamiltonian(sys_enl, env_enl):
    """Construct the full superblock Hamiltonian."""
    return kron(sys_enl.operator_dict["H"], identity(env_enl.basis_size)) + \
           kron(identity(sys_enl.basis_size), env_enl.operator_dict["H"]) + \
           H2(sys_enl.operator_dict["conn_Sz"], sys_enl.operator_dict["conn_Sp"],
              env_enl.operator_dict["conn_Sz"], env_enl.operator_dict["conn_Sp"])

def calculate_ground_state(superblock_hamiltonian):
    """Calculate the ground state using ARPACK."""
    (energy,), psi0 = eigsh(superblock_hamiltonian, k=1, which="SA")
    return energy, psi0

def reduced_density_matrix(sys_enl, psi0):
    """Construct the reduced density matrix of the system."""
    psi0 = psi0.reshape([sys_enl.basis_size, -1], order="C")
    return np.dot(psi0, psi0.T.conj())

def sort_eigenstates(evals, evecs):
    """Sort eigenvectors by eigenvalue."""
    return sorted(zip(evals, evecs.T), reverse=True, key=lambda x: x[0])

def build_transformation_matrix(eigenstates, m, sys_enl):
    """Build the transformation matrix from the most significant eigenvectors."""
    my_m = min(len(eigenstates), m)
    transformation_matrix = np.zeros((sys_enl.basis_size, my_m), dtype='d', order='F')
    for i, (_, evec) in enumerate(eigenstates[:my_m]):
        transformation_matrix[:, i] = evec
    truncation_error = 1 - sum(eigenvalue for eigenvalue, _ in eigenstates[:my_m])
    print("truncation error:", truncation_error)
    return transformation_matrix, my_m
```

$$H_{\text{super}} = H_{\text{sys, enlarged}} \otimes \mathbb{I}_{\text{env, enlarged}} + \\ + \mathbb{I}_{\text{sys, enlarged}} \otimes H_{\text{env, enlarged}} + H_{\text{int, sys-env}}$$

$$\rho_{\text{sys}} = \mathrm{Tr}_{\text{env}}(|\Psi\rangle\langle\Psi|)$$

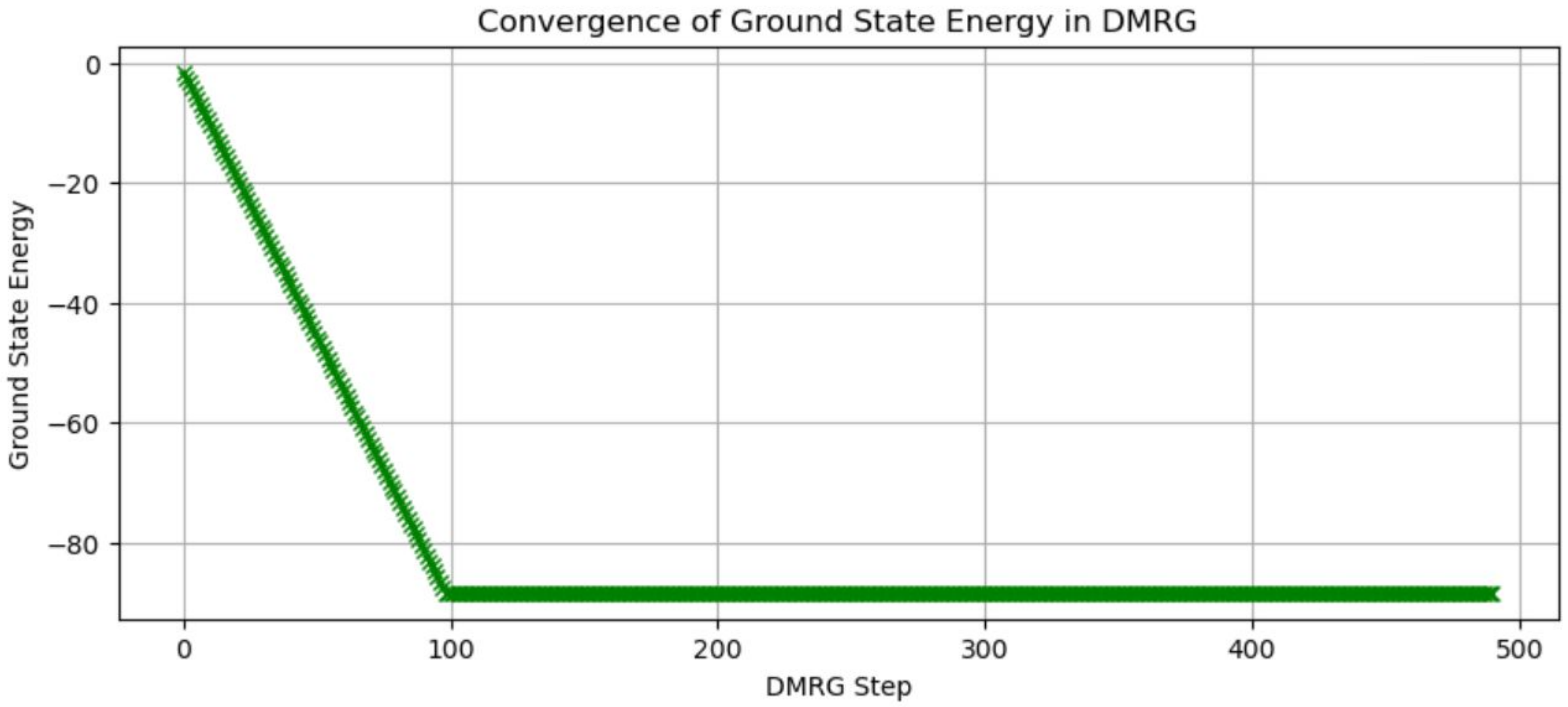
$$\tilde{O}_{\text{sys}} = U^{\dagger} O_{\text{sys}} U$$

Implementation Details:

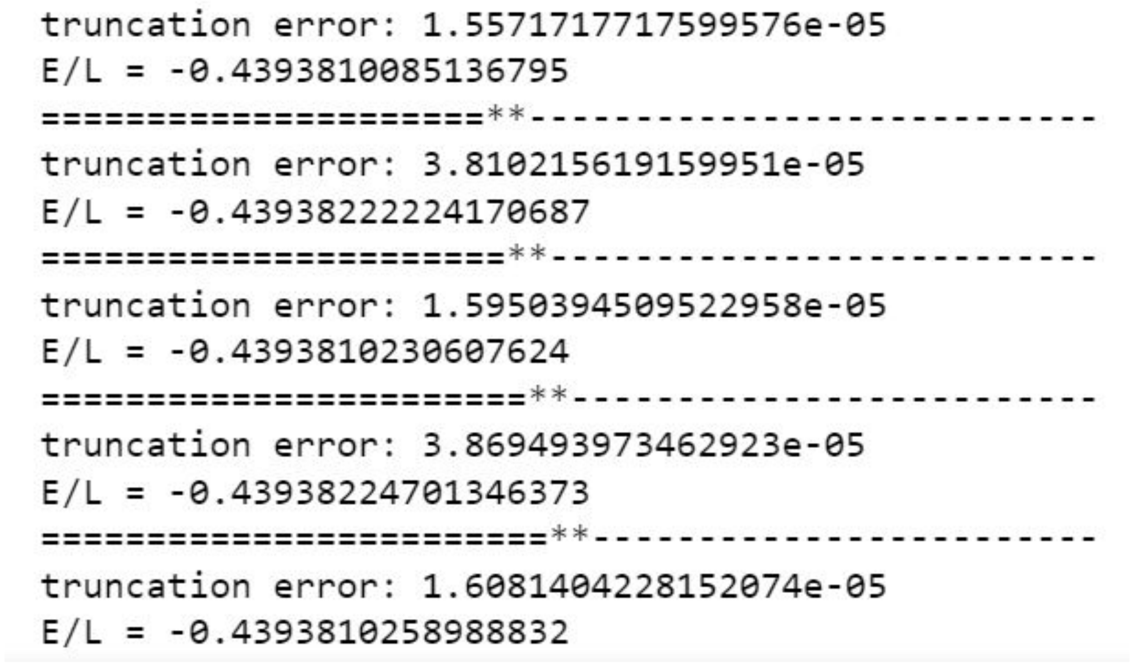


```python
def single_dmrg_step(sys, env, m):
    global energies
    assert is_valid_block(sys) and is_valid_block(env)

    sys_enl, env_enl = enlarge_blocks(sys, env)
    assert is_valid_enlarged_block(sys_enl) and is_valid_enlarged_block(env_enl)

    superblock_hamiltonian = construct_superblock_hamiltonian(sys_enl, env_enl)
    energy, psi0 = calculate_ground_state(superblock_hamiltonian)
    rho = reduced_density_matrix(sys_enl, psi0)

    evals, evecs = np.linalg.eigh(rho)
    eigenstates = sort_eigenstates(evals, evecs)
    transformation_matrix, my_m = build_transformation_matrix(eigenstates, m, sys_enl)

    new_operator_dict = rotate_truncate_operators(transformation_matrix, sys_enl)
    newblock = Block(length=sys_enl.length, basis_size=my_m, operator_dict=new_operator_dict)
    energies.append(energy)
    return newblock, energy
```

Results: Spin Chain of L=200 spins in 2 seconds



Results: Coincides with Theoretical



```
truncation error: 1.5571717717599576e-05
E/L = -0.4393810085136795
=====================**---------------------------
truncation error: 3.810215619159951e-05
E/L = -0.43938222224170687
======================**--------------------------
truncation error: 1.5950394509522958e-05
E/L = -0.4393810230607624
=======================**-------------------------
truncation error: 3.869493973462923e-05
E/L = -0.43938224701346373
========================**------------------------
truncation error: 1.6081404228152074e-05
E/L = -0.439381025898832
```

$$\frac{E_0}{L} = -\frac{J}{4} - \frac{\ln(2)J}{\pi}$$

DMRG method intersects with the following methods and ideas presented during the NLA course:

- Eigenvalue Problems
- Variational Methods
- High-Dimensional Problems
- Matrix Product States (MPS) as an instance of Tensor Trains

Contribution

Alexey, Alexander and Daniil were involved in the technical part of the project and code development.

Aikun and Alexandra did the theoretical research and rationale, as well as the graphic presentation.

References

- https://arxiv.org/pdf/1312.6542.pdf DMRG and AMEn
- https://github.com/simple-dmrg/simple-dmrg/ MIT Tutorial