```
function Energy = iDMRG(model, chain_length, m, conserved_QNum)
    %initialize the system block and the environment block where we use
    %reflection symmetry by assume system block equals environment block
    sys_block = model.InitSingleSiteBlock();
    env_block = sys_block;

    iteration_count = (chain_length-2)/2;
    for iteration=1: iteration_count
        %enlarge the sys and env block and find the corresponding superblock
        sys_block = model.EnlargeBlock(sys_block);
        env_block = model.EnlargeBlock(env_block);
        superBlock_H = model.ConstructSuperBlock_H(sys_block, env_block);

        %basis_QNum is an arary that contains the value of all possible
        %quantum number for that block. The following KeyValueMap maps
        %the quantum numbers to the indices that they were found. For
        %example, consider the sys_QNum_Indices_Map. If the quantum number
        %3 is found at indices 3,7,2 then sys_QNum_Indices_Map(3)
        %will give us {[3,7,2]}, note that this is in a cell array.
        sys_QNum_Indices_Map = KeyValueMap(sys_block.basis_QNum);
        env_QNum_Indices_Map = KeyValueMap(env_block.basis_QNum);

        restricted_basis_indices = [];

        %Now we consider the restriction of conserved quantum numbers
        sys_possible_QNum_List = keys(sys_QNum_Indices_Map);
        sys_QNum_Count = sys_QNum_Indices_Map.Count;
        new_sys_QNum_Indices_Map = containers.Map(sys_possible_QNum_List, cell(1,↙
sys_QNum_Count));
        for i=1: sys_QNum_Count
            sys_QNum = sys_possible_QNum_List{i};
            %here we calculate env_QNum by assuming the QNum on the sys_block
            env_QNum = conserved_QNum - sys_QNum;
            %checks whether the calculated env_QNum is in the possible
            %QNum of the env_block.
            if isKey(env_QNum_Indices_Map, env_QNum)
                %gets the indices(basis) that matches whatever sys_QNum is
                sys_QNum_Indices = sys_QNum_Indices_Map(sys_QNum);
                sys_QNum_Indices = sys_QNum_Indices{1};
                for j=1: length(sys_QNum_Indices)
                    index_offset = sys_block.basis_size*(sys_QNum_Indices(j)-1);
                    %gets the indices(basis) that matches whatever env_QNum is
                    env_QNum_Indices = env_QNum_Indices_Map(env_QNum);
                    env_QNum_Indices = env_QNum_Indices{1};
                    for k=1: length(env_QNum_Indices)
                        restricted_basis_indices(end+1) = index_offset +↙
env_QNum_Indices(k);
                        new_index = length(restricted_basis_indices);
                        tmp_arr = new_sys_QNum_Indices_Map(sys_QNum);
                        new_sys_QNum_Indices_Map(sys_QNum) = [tmp_arr new_index];
```

```
                    end
                end
            end
        end
        restricted_superBlock_H = superBlock_H(restricted_basis_indices,↙
restricted_basis_indices);
        [restricted_psi0, Energy] = eigs(restricted_superBlock_H,1,'smallestreal');


        new_sys_QNum = keys(new_sys_QNum_Indices_Map);
        new_sys_Indices = values(new_sys_QNum_Indices_Map);

        %we now calculate the reduce density matrix. Here, we make them in
        %block matrix form where each block of the matrix represents the
        %corresponding quantum numbers
        rho_e_values = [];
        rho_e_vects = zeros(sys_block.basis_size,1);
        new_basis_QNum = [];
        current_evalue_index = 1;
        for i=1:new_sys_QNum_Indices_Map.Count
            if ~isempty(new_sys_Indices{i})
                psi0_QNum = restricted_psi0(new_sys_Indices{i});
                sys_QNum_Indices = sys_QNum_Indices_Map(new_sys_QNum{i}).';
                psi0_QNum = reshape(psi0_QNum, [],length(sys_QNum_Indices{1})).';
                rho_block = psi0_QNum*psi0_QNum';
                [e_vects, diag_mat] = eig(rho_block);
                e_values = diag(diag_mat);
                current_QNum_Indices = sys_QNum_Indices_Map(new_sys_QNum{i});
                for j = 1: length(e_values)
                    rho_e_values(current_evalue_index) = e_values(j);
                    rho_e_vects(current_QNum_Indices{1},current_evalue_index) = e_vects↙
(:,j);
                    new_basis_QNum(current_evalue_index) = new_sys_QNum{i};
                    current_evalue_index = current_evalue_index + 1;
                end
            end
        end

        %reorder the eigenvalues, eigen vectors, the corresponding quantum
        %number in descending eigenvalue order.
        [rho_e_values, Index] = sort(rho_e_values, 'descend');
        rho_e_vects = rho_e_vects(:,Index);
        new_basis_QNum = new_basis_QNum(Index);

        %truncation
        NKeep = min(length(rho_e_values), m);
        T = rho_e_vects(:, 1:NKeep);
        new_basis_QNum = new_basis_QNum(:, 1:NKeep);
        trunc_err = 1 - sum(rho_e_values(1:NKeep));
```

```
        %update our curent block
        newBlock.basis_size = NKeep;
        newBlock.length = sys_block.length;
        newBlock.op_list = containers.Map;
        newBlock.basis_QNum = new_basis_QNum;
        sys_op_name = keys(sys_block.op_list);
        for i=1: length(sys_op_name)
           op_name = sys_op_name(i);
           op_mat = sys_block.op_list(op_name{1});
           newBlock.op_list(op_name{1}) = T'*op_mat*T;
        end
        sys_block = newBlock;
        env_block = newBlock;

        fprintf("L=%i, E=%d, err=%d\n", iteration*2+2, Energy, trunc_err);
    end
end

function map = KeyValueMap(array)
    map = containers.Map('KeyType','double','ValueType','any');
    possible_keys = unique(array);
    for i=1: length(possible_keys)
        possible_key = possible_keys(i);
        map(possible_key) = {find(array == possible_key)};
    end
end
```