Shortcuts for Variational Algorithms in Molecular Simulation

Julián Ferreiro-Vélez
PhD student

NQUIRE

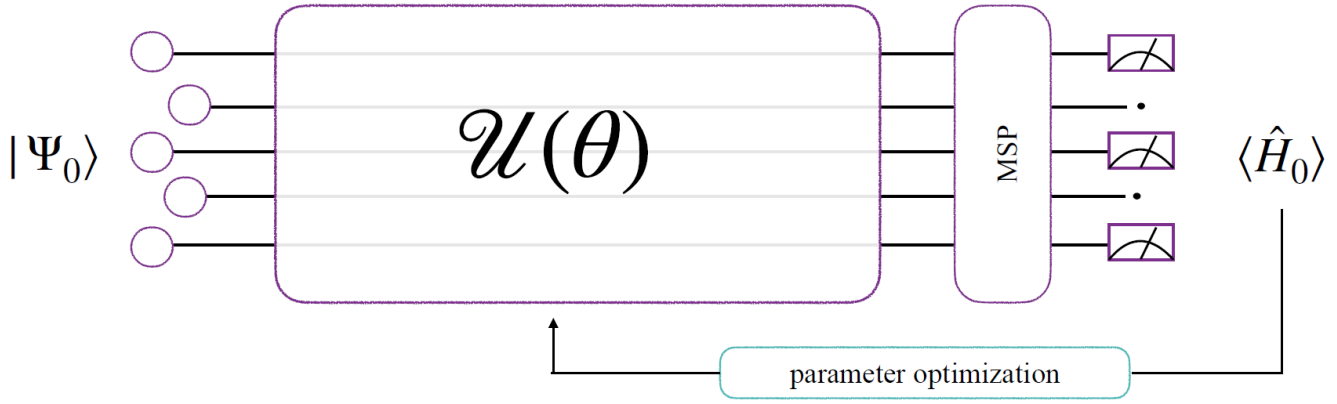
tecnalia
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE

BasQ
Basque Quantum

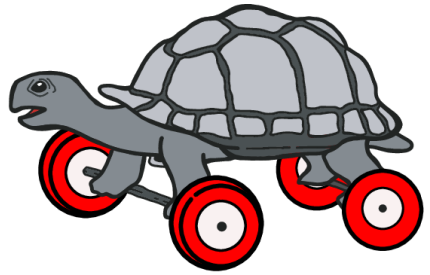
QISKIT 2024 FALL FEST 2024

Variational Quantum Eigensolver

- State Preparation
- Ansatz Implementation
- Measurement State Preparation
- Classical Optimization



Shortcuts to adiabaticity

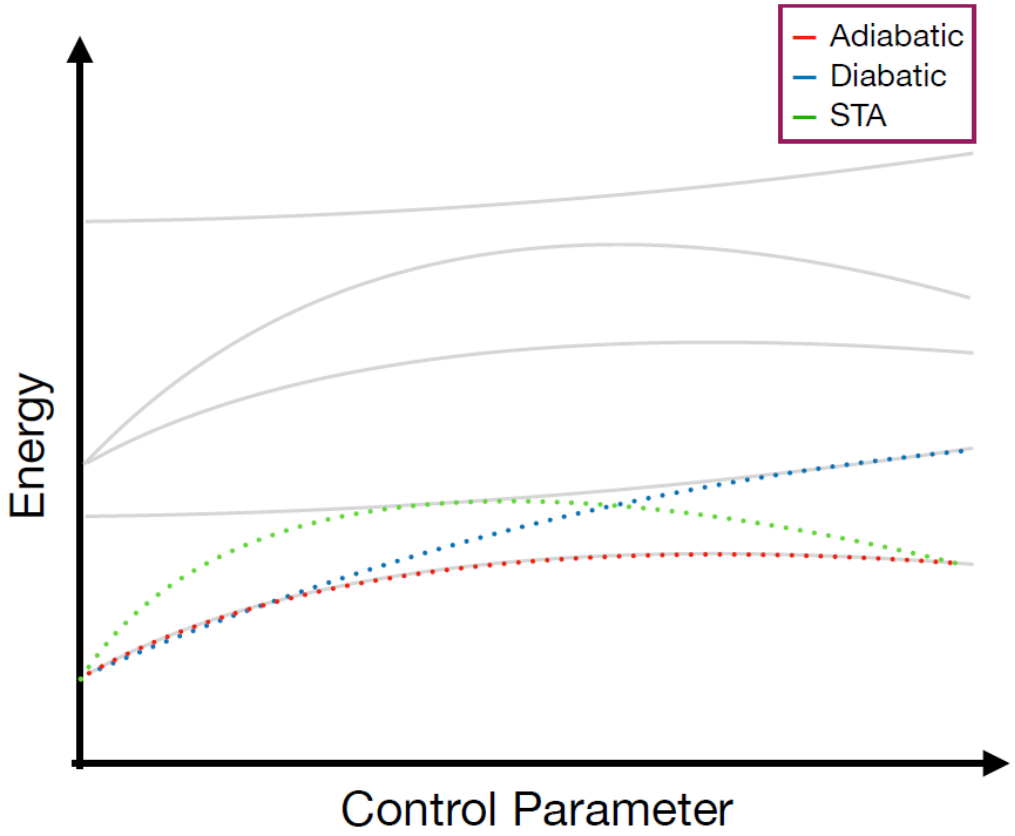


- Counter-Diabatic Driving

$$\hat{H}(t) = \hat{H}_\lambda(t) + \hat{H}_{CD}(t),$$

- Nested Commutator Expansion

$$\hat{H}_{CD}(t) = \dot{\lambda}(t)\, i \sum_{k=1}^{l} \alpha_k \underbrace{[\hat{H}_\lambda, [\hat{H}_\lambda, \ldots [\hat{H}_\lambda}_{2k-1}, \partial_\lambda \hat{H}_\lambda]]],$$



Adiabatic Gauge Ansatz and Reduced Adiabatic Gauge Ansatz

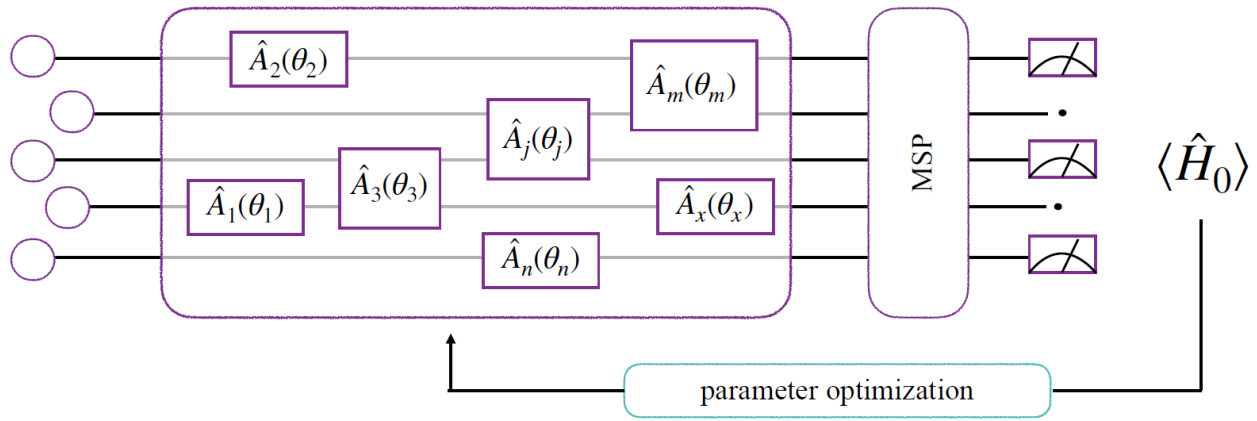
Molecule Nested Expansion

$$\hat{H}_\lambda(t) = \sum_{p,q} h_{pq}\hat{a}_p^\dagger \hat{a}_q + \frac{\lambda(t)}{2}\sum_{p,q,r,s} h_{pqrs}\hat{a}_p^\dagger \hat{a}_q^\dagger \hat{a}_r \hat{a}_s.$$

$$\hat{H}_{CD}(t) = \dot{\lambda}(t)\, i \sum_{k=1}^{l} \alpha_k \underbrace{[\hat{H}_\lambda, [\hat{H}_\lambda, \ldots [\hat{H}_\lambda}_{2k-1}, \partial_\lambda \hat{H}_\lambda]]],$$

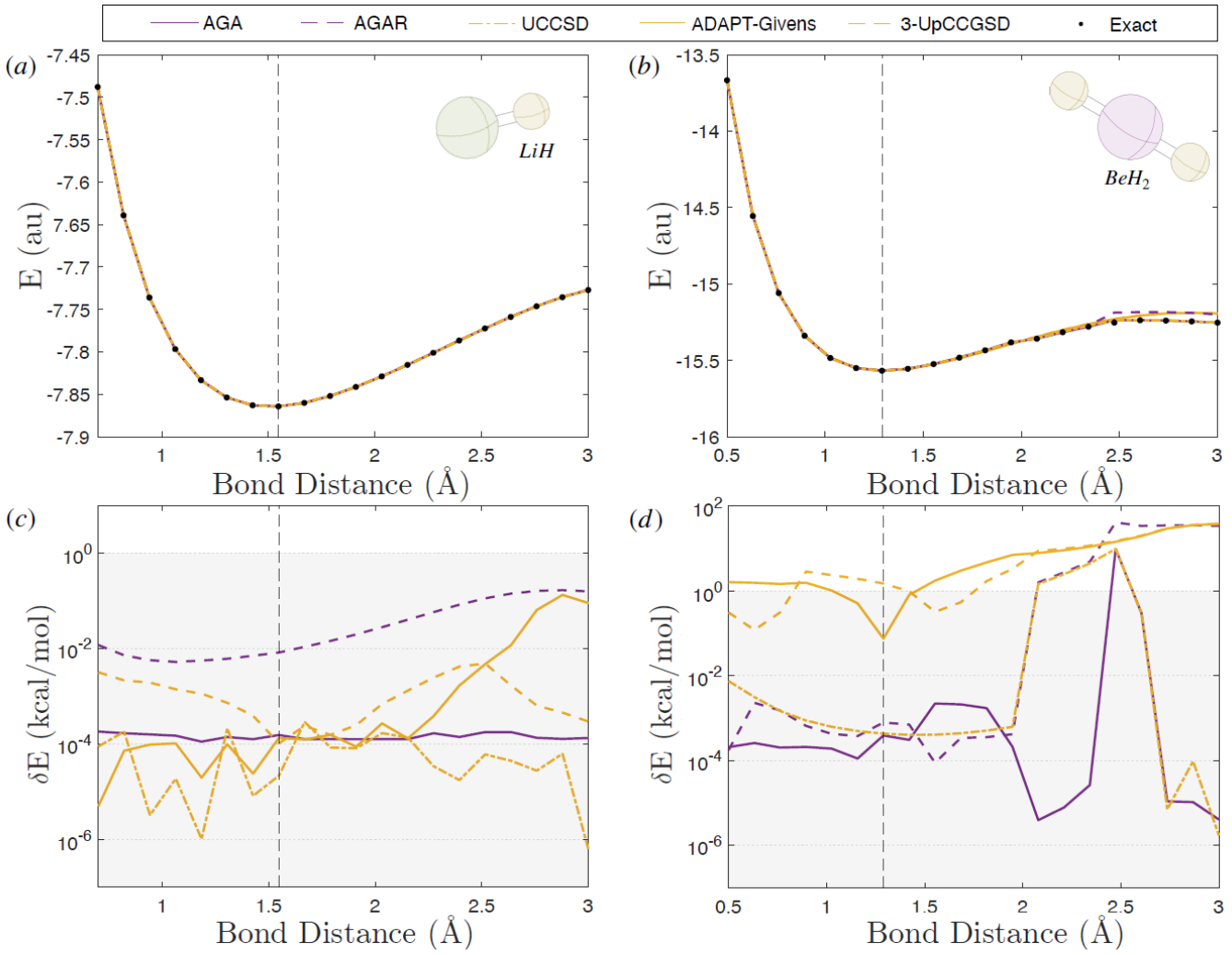
AGA Engineering

$$AGA^{(l)}(\vec{\theta}) = \sum_{k=1}^{l} \{\theta_1 \hat{A}_1 + \ldots + \theta_j \hat{A}_j + \ldots + \theta_m \hat{A}_m\}_k$$



Simulation Results

	UCCSD	AGA	AGAR	3-UpCCGSD	ADAPT-Givens
H_2	4 (14)	2	2	18	2
LiH	20(130)	56	9	54	13*
BeH_2	77(574)	30*	10*	108	10*



Implementation in IBM

```python
#----------------------------------
# Define the problem Hamiltonian
#----------------------------------
hamiltonian = SparsePauliOp.from_list(
    [("YZ", 0.3980), ("ZI", -0.3980), ("ZZ", -0.0113), ("XX", 0.1810)]
)

#----------------------------------
# Define the ansatz
#----------------------------------
ansatz = EfficientSU2(hamiltonian.num_qubits)

#----------------------------------
# Transpile the hamiltonian and the ansatz to the device lenguage
#----------------------------------
real_backend = service.backend("ibm_*****")
backend = AerSimulator.from_backend(real_backend)

target = backend.target
pm = generate_preset_pass_manager(target=target, optimization_level=3)

ansatz_isa = pm.run(ansatz)

#----------------------------------
# Minimize the cost function
#----------------------------------
with Session(backend=backend) as session:
    estimator = Estimator(session=session)
    estimator.options.default_shots = 10000

    res = minimize(
        cost_func,
        x0,
        args=(ansatz_isa, hamiltonian_isa, estimator),
        method="cobyla",
    )

```

```python
LiH_cd_energy = []

for r in range(0,len(lih_ham__dict_simp)):
    print(r)

    #----------------------------------
    # Define the molecule Hamiltonian for r-bond distance
    #----------------------------------
    lih_ham = []
    for h_k in list(lih_ham__dict_simp[r].keys()): lih_ham.append((h_k, lih_ham__dict_simp[r][h_k]))
    lih_ham_qiskit = SparsePauliOp.from_list(lih_ham)

    #----------------------------------
    # Construct the circuit ansatz
    #----------------------------------
    circuit = QuantumCircuit(nqb)
    circuit = hf_creation(circuit,HF_state_simp) #HF state

    for ansatz_k in ansatz_cd: circuit = ansatz_creator(circuit,ansatz_k[0],ansatz_k[1])

    #----------------------------------
    # Transpile the hamiltonian and the ansatz to the device lenguage
    #----------------------------------
    pm = TranspilerService(backend_name="ibm_*****",ai="auto",optimization_level=3)

    circuit_isa = pm.run(circuit)
    isa_depth = pm.run(circuit).depth()

    n_tr = 20
    for _ in range(0,n_tr):
        circuit_isa_aux = pm.run(circuit)
        if circuit_isa_aux.depth() < isa_depth:
            circuit_isa = circuit_isa_aux
            isa_depth   = circuit_isa.depth()

    hamiltonian_isa = lih_ham_qiskit.apply_layout(layout=circuit_isa.layout)

    #Parameters
    x0 = 0 * np.random.random(len(cd_keys))

    cost_history_dict = {
        "prev_vector": None,
        "iters": 0,
        "cost_history": [],
    }

    #----------------------------------
    # Minimize the cost function
    #----------------------------------
    with Session(backend=backend) as session:
        estimator = Estimator(mode=session, options={"resilience_level": 2})
        estimator.options.default_shots = 50000

        res = minimize(
            cost_func,
            x0,
            args=(circuit_isa, hamiltonian_isa, estimator),
            method="cobyla",
        )

    LiH_cd_energy.append(min(cost_history_dict['cost_history']))
```

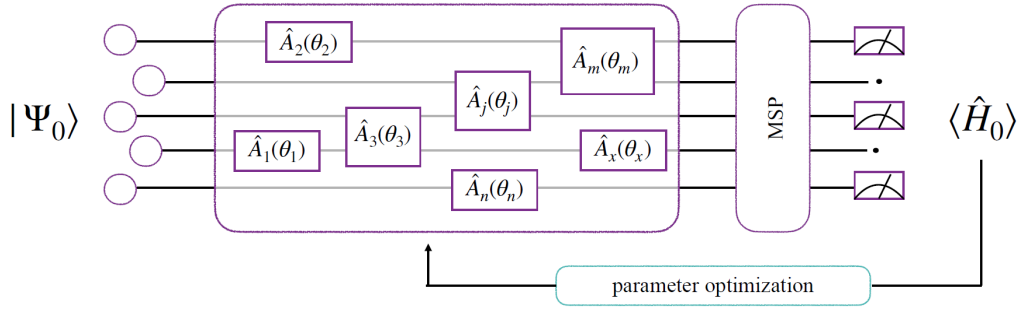

NQUIRE

tecnalia
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE

BasQ
Basque Quantum

QISKIT 2024 2024 FALL FEST

Ansatz Design




```python
#-----------------------------------
# Define the molecule Hamiltonian for r-bond distance
#-----------------------------------
lih_ham = []
for h_k in list(lih_ham__dict_simp[r].keys()): lih_ham.append((h_k, lih_ham__dict_simp[r][h_k]))
lih_ham_qiskit = SparsePauliOp.from_list(lih_ham)

#-----------------------------------
# Construct the circuit ansatz
#-----------------------------------
circuit = QuantumCircuit(nqb)
circuit = hf_creation(circuit,HF_state_simp) #HF state

for ansatz_k in ansatz_cd: circuit = ansatz_creator(circuit,ansatz_k[0],ansatz_k[1])

```


NQUIRE

tecnalia
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE

BasQ
Basque Quantum

QISKIT 2024 2024 FALL FEST

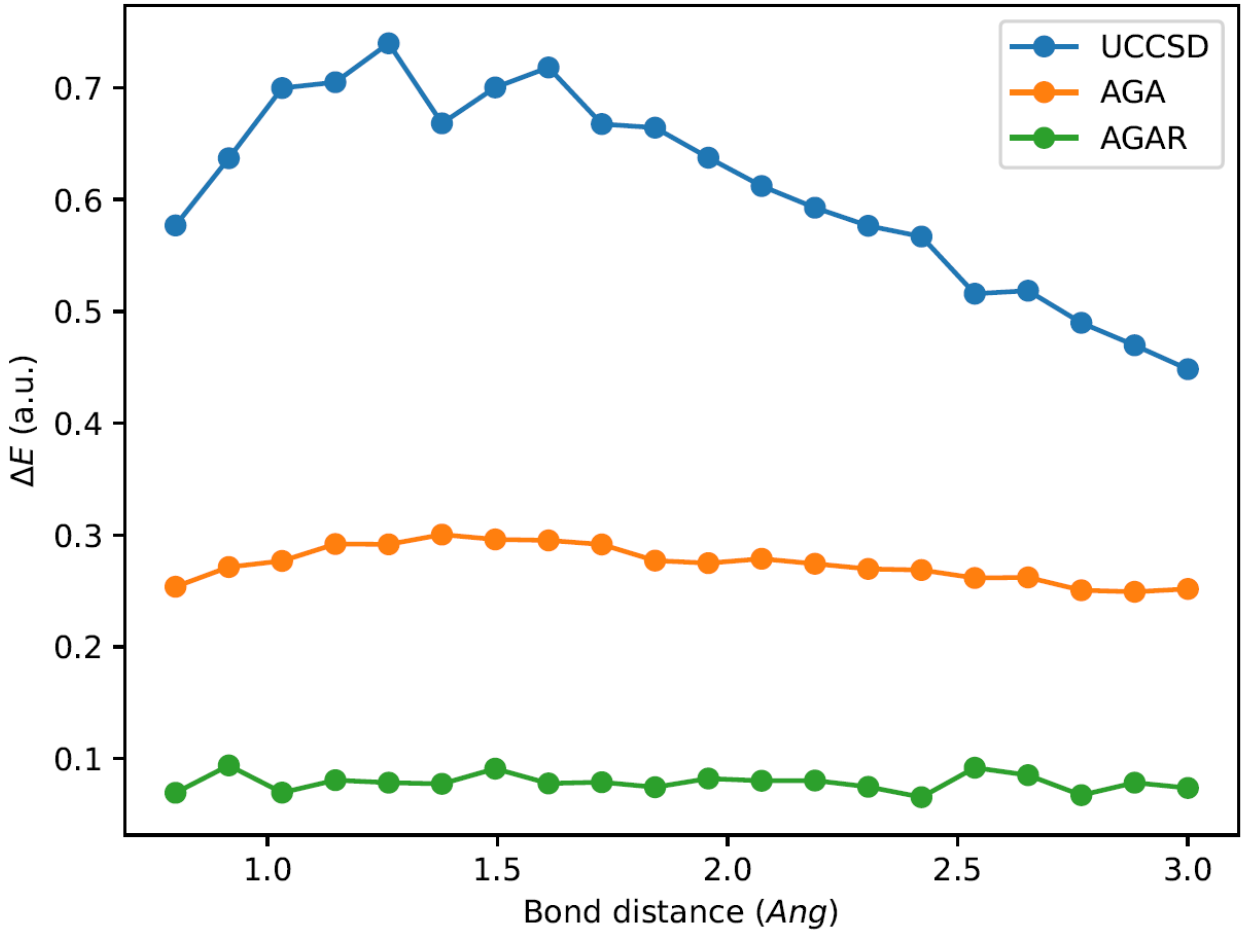
Transpiler

```python
#-----------------------------------
# Transpile the hamiltonian and the ansatz to the device lenguage
#-----------------------------------
pm = TranspilerService(backend_name="ibm_*****",ai="auto",optimization_level=3)

circuit_isa = pm.run(circuit)
isa_depth = pm.run(circuit).depth()

n_tr = 20
for _ in range(0,n_tr):
    circuit_isa_aux = pm.run(circuit)
    if circuit_isa_aux.depth() < isa_depth:
        circuit_isa = circuit_isa_aux
        isa_depth   = circuit_isa.depth()

hamiltonian_isa = lih_ham_qiskit.apply_layout(layout=circuit_isa.layout)
```

Estimator and Noise Mitigation

```
47  #----------------------------------
48  # Minimize the cost function
49  #----------------------------------
50  with Session(backend=backend) as session:
51      estimator = Estimator(mode=session, options={"resilience_level": 2})
52      estimator.options.default_shots = 50000
53
54      res = minimize(
55          cost_func,
56          x0,
57          args=(circuit_isa, hamiltonian_isa, estimator),
58          method="cobyla",
59      )
60
61  LiH_cd_energy.append(min(cost_history_dict['cost_history']))
```

Results


UCCSD
AGA
AGAR
0.7
0.6
0.5
0.4
0.3
0.2
0.1
ΔE (a.u.)
1.0
1.5
2.0
2.5
3.0
Bond distance (Ang)

Thanks for your attention

NQUIRE

arXiv:2407.20957v1

tecnalia
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE