protein_folding_model

March 11, 2024

[1]:
```
from qiskit_research.protein_folding.interactions.random_interaction import (
    RandomInteraction,
)
from qiskit_research.protein_folding.interactions.miyazawa_jernigan_interaction␣
 ↪import (
    MiyazawaJerniganInteraction,
)
from qiskit_research.protein_folding.peptide.peptide import Peptide
from qiskit_research.protein_folding.protein_folding_problem import (
    ProteinFoldingProblem,
)

from qiskit_research.protein_folding.penalty_parameters import PenaltyParameters

from qiskit.utils import algorithm_globals, QuantumInstance

algorithm_globals.random_seed = 23
```

[2]:
```
# qiskit_research tutorial chain
# main_chain = "APRLRFY"
# fractal analytics Alzheimer's enzyme related chain
main_chain = "YPYFIP"
main_chain_len = len(main_chain)
print("main_chain_len", main_chain_len)
```

```
main_chain_len 6
```

[3]:
```
side_chains = [""] * main_chain_len
```

[4]:
```
random_interaction = RandomInteraction()
mj_interaction = MiyazawaJerniganInteraction()
```

[5]:
```
penalty_back = 10
penalty_chiral = 10
penalty_1 = 10

penalty_terms = PenaltyParameters(penalty_chiral, penalty_back, penalty_1)
```

[6]:
```
peptide = Peptide(main_chain, side_chains)
```

[7]:
```
protein_folding_problem = ProteinFoldingProblem(peptide, mj_interaction,␣
  ↪penalty_terms)
qubit_op = protein_folding_problem.qubit_op()
```

[8]:
```
print(qubit_op)
```

```
929.4905 * IIIIII
+ 300.0 * IIIZII
- 97.5 * IIIIZZ
+ 97.5 * IIIZZZ
- 100.0 * IZIZII
- 100.0 * IIZIZI
- 100.0 * IZZZZI
+ 202.5 * IIIZZI
- 310.0 * IZIIII
- 207.5 * IZZIII
+ 205.0 * IIIIIZ
+ 102.5 * IIZIIZ
- 102.5 * IZZIIZ
- 924.4905 * ZIIIII
- 302.5 * ZIIZII
- 202.5 * ZIIZZI
+ 310.0 * ZZIIII
+ 207.5 * ZZZIII
+ 102.5 * ZZIZII
+ 102.5 * ZIZIZI
+ 102.5 * ZZZZZI
- 205.0 * ZIIIIZ
+ 100.0 * ZIIIZZ
- 100.0 * ZIIZZZ
- 102.5 * ZIZIIZ
+ 102.5 * ZZZIIZ
- 2.5 * IIIIZI
+ 2.5 * IIZIII
+ 2.5 * ZIIIZI
- 2.5 * ZIZIII
```

[9]:
```
from qiskit.circuit.library import RealAmplitudes
from qiskit.algorithms.optimizers import COBYLA
from qiskit.algorithms import NumPyMinimumEigensolver
from qiskit.algorithms.minimum_eigensolvers import SamplingVQE
from qiskit import execute, Aer
from qiskit.primitives import Sampler

# set classical optimizer
optimizer = COBYLA(maxiter=50)
```

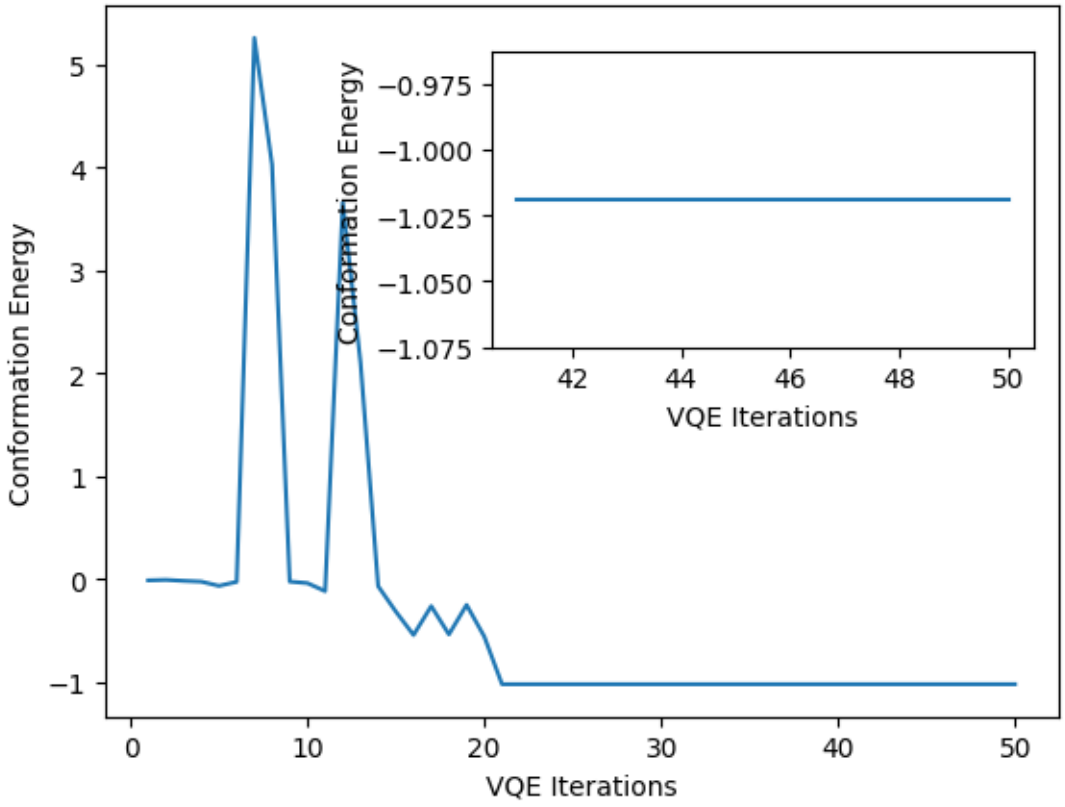
```
# set variational ansatz
ansatz = RealAmplitudes(reps=1)

counts = []
values = []


def store_intermediate_result(eval_count, parameters, mean, std):
    counts.append(eval_count)
    values.append(mean)


# initialize VQE using CVaR with alpha = 0.1
vqe = SamplingVQE(
    Sampler(),
    ansatz=ansatz,
    optimizer=optimizer,
    aggregation=0.1,
    callback=store_intermediate_result,
)

raw_result = vqe.compute_minimum_eigenvalue(qubit_op)
print(raw_result)
```

```
SamplingMinimumEigensolverResult:
        Eigenvalue: -1.0190000000001191
        Best measurement
: {'state': 37, 'bitstring': '100101', 'value': (-1.0190000000001191+0j),
'probability': 0.223520808846322}
```

[10]:
```
import matplotlib.pyplot as plt

fig = plt.figure()

plt.plot(counts, values)
plt.ylabel("Conformation Energy")
plt.xlabel("VQE Iterations")

fig.add_axes([0.44, 0.51, 0.44, 0.32])

plt.plot(counts[40:], values[40:])
plt.ylabel("Conformation Energy")
plt.xlabel("VQE Iterations")
plt.show()
```

```
C:\Users\nat\miniconda3\envs\qa2_py39\lib\site-
```

```
packages\matplotlib\cbook.py:1699: ComplexWarning: Casting complex values to
real discards the imaginary part
  return math.isfinite(val)
C:\Users\nat\miniconda3\envs\qa2_py39\lib\site-
packages\matplotlib\cbook.py:1345: ComplexWarning: Casting complex values to
real discards the imaginary part
  return np.asarray(x, float)
```



[11]:
```
result = protein_folding_problem.interpret(raw_result=raw_result)
print(
    "The bitstring representing the shape of the protein during optimization is:
  ↪ ",
    result.turn_sequence,
)
print("The expanded expression is:", result.get_result_binary_vector())
```

```
The bitstring representing the shape of the protein during optimization is:
111011
The expanded expression is: 1_____________________________________________________
_____________________________________1101_1____
```

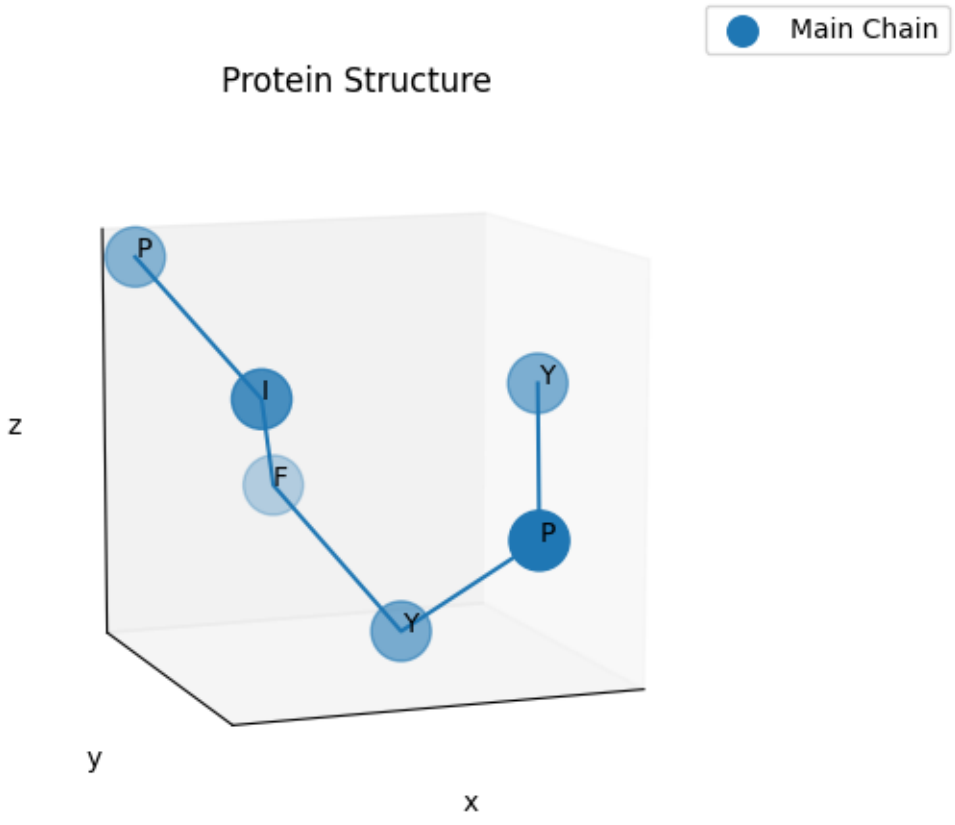
[12]:
```
print(
    f"The folded protein's main sequence of turns is: {result.
 ↪protein_shape_decoder.main_turns}"
)
print(f"and the side turn sequences are: {result.protein_shape_decoder.
 ↪side_turns}")
```

```
The folded protein's main sequence of turns is: [1, 0, 3, 2, 3]
and the side turn sequences are: [None, None, None, None, None, None]
```

[13]:
```
print(result.protein_shape_file_gen.get_xyz_data())
```

```
[['Y' '0.0' '0.0' '0.0']
 ['P' '0.5773502691896258' '0.5773502691896258' '-0.5773502691896258']
 ['Y' '1.1547005383792517' '0.0' '-1.1547005383792517']
 ['F' '1.7320508075688776' '-0.5773502691896258' '-0.5773502691896258']
 ['I' '2.3094010767585034' '0.0' '0.0']
 ['P' '2.886751345948129' '-0.5773502691896258' '0.5773502691896258']]
```

[14]:
```
fig = result.get_figure(title="Protein Structure", ticks=False, grid=True)
fig.get_axes()[0].view_init(10, 70)
```



[ ]:

[ ]: