2_Understand_Sampler_Theory

December 22, 2025

1 Task 5.2 Understand Theortical Background behind Sampler

1.1 Objective 1: Get Started with Primitives

Check Task 5.1 objective 1

1.2 Objective 2: Sampler

1.2.1 Attributes

- mode
- options
- version

1.2.2 Methods

- backend()
- run()

1.3 Objective 3: Migrate to Qiskit V2 primitives

1.3.1 Changes from V1 to V2

- Import
  - import is different

from qiskit_ibm_runtime import EstimatorV2 as Estimator

from qiskit_ibm_runtime import SamplerV2 as Sampler

- Input
  - Sampler V2 PUB format: (circuit, parameter values, shots), where parameter values and shots are optional.
  - Estimator V2 PUB format: (circuit, observables, parameter values, precision), where parameter values and precision are optional.

[ ]:
```
# Estimate expectation values for two PUBs, both with 0.05 precision.
estimator.run([(circuit1, obs_array1), (circuit2, obs_array_2)], precision=0.05)
```

[ ]:
```
# Sample two circuits at 128 shots each.
sampler.run([circuit1, circuit2], shots=128)
```

```
# Sample two circuits at different amounts of shots.
# The "None"s are necessary as placeholders
# for the lack of parameter values in this example.
sampler.run([
  (circuit1, None, 123),
  (circuit2, None, 456),
])
```

- Output
  - ouput is PubResult
  - Estimator contains expectation_values and standard_errors
  - Sampler containts per_shot measurments as bitstrings

[ ]:
```
# Estimator V1: Execute 1 circuit with 4 observables
job = estimator_v1.run([circuit] * 4, [obs1, obs2, obs3, obs4])
evs = job.result().values

# Estimator V2: Execute 1 circuit with 4 observables
job = estimator_v2.run([(circuit, [obs1, obs2, obs3, obs4])])
evs = job.result()[0].data.evs
##############################################################################################
# Estimator V1: Execute 1 circuit with 4 observables and 2 parameter sets
job = estimator_v1.run([circuit] * 8, [obs1, obs2, obs3, obs4] * 2, [vals1,␣
 ↪vals2] * 4)
evs = job.result().values

# Estimator V2: Execute 1 circuit with 4 observables and 2 parameter sets

job = estimator_v2.run([(circuit, [[obs1], [obs2], [obs3], [obs4]], [[vals1],␣
 ↪[vals2]])])
evs = job.result()[0].data.evs
##############################################################################################
# Estimator V1: Cannot execute 2 circuits with different observables

# Estimator V2: Execute 2 circuits with 2 different observables.  There are
# two PUBs because each PUB can have only one circuit.
job = estimator_v2.run([(circuit1, obs1), (circuit2, obs2)])
evs1 = job.result()[0].data.evs  # result for pub 1 (circuit 1)
evs2 = job.result()[1].data.evs  # result for pub 2 (circuit 2)
```

[ ]:
```
# Sampler V1: Execute 1 circuit with 3 parameter sets
job = sampler_v1.run([circuit] * 3, [vals1, vals2, vals3])
dists = job.result().quasi_dists

# Sampler V2: Executing 1 circuit with 3 parameter sets
job = sampler_v2.run([(circuit, [vals1, vals2, vals3])])
```

```
counts = job.result()[0].data.meas.get_counts()
##############################################################################################
# Sampler V1: Execute 2 circuits with 1 parameter set
job = sampler_v1.run([circuit1, circuit2], [vals1] * 2)
dists = job.result().quasi_dists

# Sampler V2: Execute 2 circuits with 1 parameter set
job = sampler_v2.run([(circuit1, vals1), (circuit2, vals1)])
counts1 = job.result()[0].data.meas.get_counts()  # result for pub 1 (circuit 1)
counts2 = job.result()[1].data.meas.get_counts()  # result for pub 2 (circuit 2)

##############################################################################################
v2_result = sampler_v2_job.result()
v1_format = []
for pub_result in v2_result:
  counts = pub_result.data.meas.get_counts()
  v1_format.append( {int(key, 2): val/shots for key, val in counts.items()} )
```

- Options
  - V2 have their Options class
  - use update method

[ ]:
```
########################################## V2
from dataclasses import asdict
from qiskit_ibm_runtime import QiskitRuntimeService
from qiskit_ibm_runtime import EstimatorV2 as Estimator

service = QiskitRuntimeService()
backend = service.least_busy(operational=True, simulator=False)

# Setting options during primitive initialization
estimator = Estimator(backend, options={"resilience_level": 2})

# Setting options after primitive initialization
# This uses auto complete.
estimator.options.default_shots = 4000
# This does bulk update.
estimator.options.update(default_shots=4000, resilience_level=2)

# Print the dictionary format.
# Server defaults are used for unset options.
print(asdict(estimator.options))

########################################## V1
from qiskit_ibm_runtime import QiskitRuntimeService, Sampler, Options

service = QiskitRuntimeService()
```

```
backend = service.least_busy(operational=True, simulator=False)

# Setting options during primitive initialization
options = Options()
# This uses auto complete.
options.resilience_level = 2
estimator = Estimator(backend=backend, options=options)

# Setting options after primitive initialization.
# This does bulk update.
estimator.set_options(shots=4000)
```

[ ]:
```
########################################## V2
from dataclasses import asdict
from qiskit_ibm_runtime import QiskitRuntimeService
from qiskit_ibm_runtime import SamplerV2 as Sampler

service = QiskitRuntimeService()
backend = service.least_busy(operational=True, simulator=False)

# Setting options during primitive initialization
sampler = Sampler(backend, options={"default_shots": 4096})

# Setting options after primitive initialization
# This uses auto complete.
sampler.options.dynamical_decoupling.enable = True
# Turn on gate twirling. Requires qiskit_ibm_runtime 0.23.0 or later.
sampler.options.twirling.enable_gates = True

# This does bulk update.  The value for default_shots is overridden if you␣
 ↪specify shots with run() or in the PUB.
sampler.options.update(default_shots=1024,␣
 ↪dynamical_decoupling={"sequence_type": "XpXm"})

# Print the dictionary format.
# Server defaults are used for unset options.
print(asdict(sampler.options))
########################################## V1
from qiskit_ibm_runtime import QiskitRuntimeService, Sampler, Options

service = QiskitRuntimeService()
backend = service.least_busy(operational=True, simulator=False)

# Setting options during primitive initialization
options = Options()
# This uses auto complete.
options.resilience_level = 2
```

```
sampler = Sampler(backend=backend, options=options)

# Setting options after primitive initialization.
# This does bulk update.
sampler.set_options(shots=2000)
```

- Error Mitigation
  - Sampler doesn't support resilience level
  - Estimator doesn't support resilience level 3 using PEC, it support levels 0,1,2

[1]:
```
########################################## V2
from dataclasses import asdict
from qiskit_ibm_runtime import QiskitRuntimeService
from qiskit_ibm_runtime import EstimatorV2 as Estimator

service = QiskitRuntimeService()
backend = service.least_busy(operational=True, simulator=False)

# Setting options during primitive initialization
estimator = Estimator(backend)

# Set resilience_level to 0
estimator.options.resilience_level = 0

# Turn on measurement error mitigation
estimator.options.resilience.measure_mitigation = True

########################################## V1
from qiskit_ibm_runtime import Estimator, Options

estimator = Estimator(backend, options=options)

options = Options()

options.resilience_level = 2
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 21
     18 ########################################## V1
     19 from qiskit_ibm_runtime import Estimator, Options
---> 21 estimator = Estimator(backend, options=options)
     23 options = Options()
     25 options.resilience_level = 2

NameError: name 'options' is not defined
```

[ ]:
```python
############################################ V2
from qiskit_ibm_runtime import SamplerV2 as Sampler

sampler = Sampler(backend)
# Turn on dynamical decoupling with sequence XpXm.
sampler.options.dynamical_decoupling.enable = True
sampler.options.dynamical_decoupling.sequence_type = "XpXm"

print(f">> dynamical decoupling sequence to use: {sampler.options.
 ↪dynamical_decoupling.sequence_type}")

############################################ V1
from qiskit_ibm_runtime import Sampler, Options

sampler = Sampler(backend, options=options)

options = Options()

options.resilience_level = 2
```

- Transpilation
  - V2 supporrt circuits that adhere to ISA of a prticular backend
- Job Status
  - new RunttimeJobv2

[ ]:
```python
############################################ V2
job = estimator.run(...)

# check if a job is still running
print(f"Job {job.job_id()} is still running: {job.status() == "RUNNING"}")


############################################ V1
from qiskit.providers.jobstatus import JobStatus

job = estimator.run(...)

#check if a job is still running
print(f"Job {job.job_id()} is still running: {job.status() is JobStatus.
 ↪RUNNING}")
```

1.3.2 Steps to migrate to Esimtator V2

1. change import to EstimatorV2
2. remove import options
3. update options
4. group circuit with observables and parameter values in a PUB
5. reshape your arrays form observables and pparameter values

6. specify precision
7. update run method
8. use index to get results from pub_result

Steps to mmigrate to Sampler V2

1. change import to SamplerV2
2. remove import options
3. update options
4. group circuit with observables and parameter values in a PUB
5. update run method
6. use index to get results from pub_result