

Benchmarking gate-based quantum computers via certification of qubit von Neumann measurements

Paulina Lewandowska^{1,*} and Martin Beseda^{1,2}

¹IT4Innovations, VSB - Technical University of Ostrava, 17. listopadu 2172/15, 708 33 Ostrava, Czech Republic

²Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Università dell'Aquila, via Vetoio, I-67010 Coppito-L'Aquila, Italy

*paulina.lewandowska@vsb.cz

Appendix A: PyQBench as CLI

In this subsection, we will demonstrate how `qbench` package can be used as CLI. Here, we describe how to define an experiment file, a backend file and differences between YML output files from asynchronous and synchronous experiments.

Defining the experiment

Now we present how to prepare the configuration files. The first configuration file describes the experiment scenario to be executed.

Listing 1. Defining the experiment

```
1 type: certification-fourier
2 qubits:
3     - target: 0
4       ancilla: 1
5 angles:
6     start: 0
7     stop: 2 * pi
8     num_steps: 8
9 delta: 0.05
10 gateset: ibmq
11 method: direct_sum
12 num_shots: 10000
```

The experiment file contains the following fields:

- `type`: a string describing the type of experiment. Currently, we command two typos: `discrimination-fourier` and `certification-fourier`.
- `qubits`: a list enumerating pairs of qubits on which the experiment should be run. We describe a particular pair of qubits using `target` and `ancilla` keys to emphasize that the role of qubits in the experiment is distinguished. In our case, we run benchmark on one pair of qubits choosing the `target` as qubit 0, and the `ancilla` a qubit 1.
- `angles`: an object describing the range of angles for the parameterized Fourier family of measurements. The range is always uniform, starts at `start`, ends at `stop` and contains `num_steps` points, including both `start` and `stop`. The `start` and `stop` can be arithmetic expressions using `pi` literal. Our example contains eight points: $\frac{k\pi}{4}$, where $k = 0, \dots, 8$.
- `delta`: a float defining the statistical significance chosen from the interval $[0, 1]$.
- `gateset`: a string describing the set of gates for decomposing the circuits describing experiment. We can choose `ibmq` corresponding to decompositions compatible with IBM Q devices. Alternatively, one might wish to turn off the decomposition by using a special value `generic`.

- `method`: a string, either `postselection` or `direct_sum` determining which implementation of the conditional measurement is used.
- `num_shots`: an integer defining number of shots are performed in the particular experiment.

The second configuration file describes the backend. Below we describe an example YAML file describing IBM Q backend named Kyiv. Nevertheless, the syntax for each IBM Q backend will be similar. Note that IBM Q backends typically require an access token to IBM Quantum Experience. The token is configured in `QISKIT_IBM_TOKEN` environmental variable. The following listing presents the description of the IBM Q backend for the asynchronous mode.

Listing 2. Defining IBMQ backend

```
1 name: ibm_kyiv
2 asynchronous: true
3 provider:
4     hub: ibm-q
5     group: open
6     project: main
```

YML output files

Below we present the YML output files of synchronous and asynchronous modes.

The result of running the above command can be twofold:

- If the backend is synchronous, the output will contain measurement outcomes (bitstrings) for each of the circuits run.
- If backend is asynchronous, the output will contain intermediate data containing, `job_ids` correlated with the circuit they correspond to.

Synchronous mode

Listing 3. YML output file of synchronous experiment

```
1 metadata:
2   experiments:
3     type: certification-fourier
4     qubits:
5       - {target: 0, ancilla: 1}
6     angles: {start: 0.0, stop: 6.283185307179586, num_steps: 8}
7     delta: 0.05
8     gateset: ibmq
9     method: direct_sum
10    num_shots: 10000
11    backend_description:
12      name: ibm_kyiv
13      asynchronous: false
14      provider: {group: open, hub: ibm-q, project: main}
15  data:
16  - target: 0
17    ancilla: 1
18    phi: 0.0
19    delta: 0.05
20    results_per_circuit:
21    - name: u
22      histogram: {'00': 4787, '01': 4663, '11': 314, '10': 236}
23      mitigation_info:
24        target: {prob_meas0_prep1: 0.005399999999999996,
25                  prob_meas1_prep0: 0.0018}
26        ancilla: {prob_meas0_prep1: 0.004800000000000000265,
27                   prob_meas1_prep0: 0.0018}
```

```

28     mitigated_histogram: {'10': 0.02272105079666005,
29         '11': 0.0308324019917434, '01': 0.4686639719002791,
30         '00': 0.47778257531131735}
31 - target: 0
32   ancilla: 1
33   phi: 0.8975979010256552
34   delta: 0.05
35   results_per_circuit:
36 - name: u
37   histogram: {'01': 3158, '11': 1841, '10': 2121, '00': 2880}
38   mitigation_info:
39     target: {prob_meas0_prep1: 0.005399999999999996,
40         prob_meas1_prep0: 0.0018}
41     ancilla: {prob_meas0_prep1: 0.00480000000000000265,
42         prob_meas1_prep0: 0.0018}
43   mitigated_histogram: {'11': 0.18503494944149518,
44       '10': 0.2119853847642628, '00': 0.2863022864686138,
45       '01': 0.3166773793256281}
46 - target: 0
47   ancilla: 1
48   phi: 1.7951958020513104
49   delta: 0.05
50   results_per_circuit:
51 - name: u
52   histogram: {'11': 3946, '10': 4107, '01': 933, '00': 1014}
53   mitigation_info:
54     target: {prob_meas0_prep1: 0.005399999999999996,
55         prob_meas1_prep0: 0.0018}
56     ancilla: {prob_meas0_prep1: 0.00480000000000000265,
57         prob_meas1_prep0: 0.0018}
58   mitigated_histogram: {'01': 0.09187983560151308,
59       '00': 0.0992818314007015, '11': 0.3977454665741516,
60       '10': 0.4110928664236337}
61 - target: 0
62   ancilla: 1
63   phi: 2.6927937030769655
64   delta: 0.05
65   results_per_circuit:
66 - name: u
67   histogram: {'10': 4905, '11': 4844, '00': 121, '01': 130}
68   mitigation_info:
69     target: {prob_meas0_prep1: 0.005399999999999996,
70         prob_meas1_prep0: 0.0018}
71     ancilla: {prob_meas0_prep1: 0.00480000000000000265,
72         prob_meas1_prep0: 0.0018}
73   mitigated_histogram: {'00': 0.00971145661283892,
74       '01': 0.0107234135301044, '11': 0.4884707846971315,
75       '10': 0.491094345159925}
76 - target: 0
77   ancilla: 1
78   phi: 3.5903916041026207
79   delta: 0.05
80   results_per_circuit:
81 - name: u
82   histogram: {'11': 4727, '10': 5047, '01': 119, '00': 107}

```

```

83     mitigation_info:
84         target: {prob_meas0_prep1: 0.005399999999999996,
85                 prob_meas1_prep0: 0.0018}
86         ancilla: {prob_meas0_prep1: 0.00480000000000000265,
87                 prob_meas1_prep0: 0.0018}
88     mitigated_histogram: {'00': 0.008243326178401936,
89                          '01': 0.0096749343410262, '11': 0.4766264355219874,
90                          '10': 0.5054553039585844}
91 - target: 0
92   ancilla: 1
93   phi: 4.487989505128276
94   delta: 0.05
95   results_per_circuit:
96 - name: u
97   histogram: {'10': 4067, '00': 924, '01': 1045, '11': 3964}
98   mitigation_info:
99       target: {prob_meas0_prep1: 0.005399999999999996,
100              prob_meas1_prep0: 0.0018}
101       ancilla: {prob_meas0_prep1: 0.00480000000000000265,
102               prob_meas1_prep0: 0.0018}
103   mitigated_histogram: {'00': 0.09020755763218545,
104                        '01': 0.103168725838722, '11': 0.39955085514435557,
105                        '10': 0.4070728613847364}
106 - target: 0
107   ancilla: 1
108   phi: 5.385587406153931
109   delta: 0.05
110   results_per_circuit:
111 - name: u
112   histogram: {'10': 1973, '11': 1989, '00': 3069, '01': 2969}
113   mitigation_info:
114       target: {prob_meas0_prep1: 0.005399999999999996,
115              prob_meas1_prep0: 0.0018}
116       ancilla: {prob_meas0_prep1: 0.00480000000000000265,
117               prob_meas1_prep0: 0.0018}
118   mitigated_histogram: {'10': 0.1969715269854367,
119                        '11': 0.2000488072203213, '01': 0.2975337874613869,
120                        '00': 0.305445878332855}
121 - target: 0
122   ancilla: 1
123   phi: 6.283185307179586
124   delta: 0.05
125   results_per_circuit:
126 - name: u
127   histogram: {'00': 4832, '01': 4661, '11': 257, '10': 250}
128   mitigation_info:
129       target: {prob_meas0_prep1: 0.005399999999999996,
130              prob_meas1_prep0: 0.0018}
131       ancilla: {prob_meas0_prep1: 0.00480000000000000265,
132               prob_meas1_prep0: 0.0018}
133   mitigated_histogram: {'10': 0.024153348441373016,
134                        '11': 0.0250715357945843, '01': 0.4684820500233045,
135                        '00': 0.48229306574073816}

```

The data whereas includes target, ancilla, phi, delta and results_per_circuit information. The first four pieces of information have already been described. The last data results_per_circuit gives us the following

additional information:

- **histogram**: the dictionary with measurements' outcomes. The keys represent possible bitstrings, whereas the values are the number of occurrences.
- **mitigation_info**: the parameters `prob_meas0_prep1` and `prob_meas1_prep0` contains information that are used for error mitigation using the MThree method¹ and can be found using `backends.properties().qubits`. If this information is available, it will be stored in the mitigation info field; otherwise, this field will be absent.
- **mitigation_histogram**: the histogram with probabilities of measurements' outcomes after the error mitigation.

Asynchronous mode

Listing 4. YAML output file of asynchronous mode with intermediate data

```
1 metadata:
2   experiments:
3     type: certification-fourier
4     qubits:
5       - {target: 0, ancilla: 1}
6     angles: {start: 0.0, stop: 6.283185307179586, num_steps: 8}
7     delta: 0.05
8     gateset: ibmq
9     method: direct_sum
10    num_shots: 10000
11  backend_description:
12    name: ibm_kyiv
13    asynchronous: true
14    provider: {group: open, hub: ibm-q, project: main}
15 data:
16 - job_id: cy60ptkcw2k0008jwsag
17   keys:
18   - [0, 1, u, 0.0, 0.05]
19   - [0, 1, u, 0.8975979010256552, 0.05]
20   - [0, 1, u, 1.7951958020513104, 0.05]
21   - [0, 1, u, 2.6927937030769655, 0.05]
22   - [0, 1, u, 3.5903916041026207, 0.05]
23   - [0, 1, u, 4.487989505128276, 0.05]
24   - [0, 1, u, 5.385587406153931, 0.05]
25   - [0, 1, u, 6.283185307179586, 0.05]
```

Listing 5. YAML output file of resolved experiment

```
1 metadata:
2   experiments:
3     type: certification-fourier
4     qubits:
5       - target: 0
6         ancilla: 1
7     angles:
8       start: 0.0
9       stop: 6.283185307179586
10    num_steps: 8
11    delta: 0.05
12    gateset: ibmq
13    method: direct_sum
14    num_shots: 10000
15  backend_description:
```

```

16     name: ibm_kyiv
17     asynchronous: true
18     provider:
19         group: open
20         hub: ibm-q
21         project: main
22 data:
23 - target: 0
24   ancilla: 1
25   phi: 0.0
26   delta: 0.05
27   results_per_circuit:
28 - name: u
29   histogram:
30     '00': 4898
31     '01': 4582
32     '11': 311
33     '10': 209
34   mitigation_info:
35     target:
36       prob_meas0_prep1: 0.005399999999999996
37       prob_meas1_prep0: 0.0018
38     ancilla:
39       prob_meas0_prep1: 0.00480000000000000265
40       prob_meas1_prep0: 0.0018
41   mitigated_histogram:
42     '10': 0.01998449638687051
43     '11': 0.030549024853314708
44     '01': 0.4604864304246869
45     '00': 0.4889800483351278
46 - target: 0
47   ancilla: 1
48   phi: 0.8975979010256552
49   delta: 0.05
50   results_per_circuit:
51 - name: u
52   histogram:
53     '11': 1952
54     '00': 2876
55     '10': 2074
56     '01': 3098
57   mitigation_info:
58     target:
59       prob_meas0_prep1: 0.005399999999999996
60       prob_meas1_prep0: 0.0018
61     ancilla:
62       prob_meas0_prep1: 0.00480000000000000265
63       prob_meas1_prep0: 0.0018
64   mitigated_histogram:
65     '11': 0.19626874194826874
66     '10': 0.2071941128936882
67     '00': 0.28595657203781866
68     '01': 0.3105805731202244
69 - target: 0
70   ancilla: 1

```

```

71 phi: 1.7951958020513104
72 delta: 0.05
73 results_per_circuit:
74 - name: u
75   histogram:
76     '01': 905
77     '11': 4022
78     '10': 4085
79     '00': 988
80   mitigation_info:
81     target:
82       prob_meas0_prep1: 0.005399999999999996
83       prob_meas1_prep0: 0.0018
84     ancilla:
85       prob_meas0_prep1: 0.00480000000000000265
86       prob_meas1_prep0: 0.0018
87   mitigated_histogram:
88     '01': 0.08902728596975329
89     '00': 0.09669850424566845
90     '11': 0.4054328268424948
91     '10': 0.40884138294208333
92 - target: 0
93   ancilla: 1
94   phi: 2.6927937030769655
95   delta: 0.05
96   results_per_circuit:
97   - name: u
98     histogram:
99     '10': 4990
100    '11': 4825
101    '00': 81
102    '01': 104
103    mitigation_info:
104      target:
105        prob_meas0_prep1: 0.005399999999999996
106        prob_meas1_prep0: 0.0018
107      ancilla:
108        prob_meas0_prep1: 0.00480000000000000265
109        prob_meas1_prep0: 0.0018
110    mitigated_histogram:
111      '00': 0.005669915398114243
112      '01': 0.008121105338749025
113      '11': 0.4865404579166901
114      '10': 0.4996685213464466
115 - target: 0
116   ancilla: 1
117   phi: 3.5903916041026207
118   delta: 0.05
119   results_per_circuit:
120   - name: u
121     histogram:
122     '11': 4826
123     '10': 4964
124     '00': 103
125     '01': 107

```

```

126     mitigation_info:
127         target:
128             prob_meas0_prep1: 0.005399999999999996
129             prob_meas1_prep0: 0.0018
130         ancilla:
131             prob_meas0_prep1: 0.00480000000000000265
132             prob_meas1_prep0: 0.0018
133     mitigated_histogram:
134         '00': 0.007888851269304064
135         '01': 0.008418779091074404
136         '11': 0.4866456850507466
137         '10': 0.49704668458887485
138 - target: 0
139   ancilla: 1
140   phi: 4.487989505128276
141   delta: 0.05
142   results_per_circuit:
143 - name: u
144   histogram:
145       '11': 3954
146       '10': 4124
147       '00': 889
148       '01': 1033
149   mitigation_info:
150       target:
151           prob_meas0_prep1: 0.005399999999999996
152           prob_meas1_prep0: 0.0018
153       ancilla:
154           prob_meas0_prep1: 0.00480000000000000265
155           prob_meas1_prep0: 0.0018
156   mitigated_histogram:
157       '00': 0.08667377941797003
158       '01': 0.10197127796072936
159       '11': 0.398532348147248
160       '10': 0.41282259447405256
161 - target: 0
162   ancilla: 1
163   phi: 5.385587406153931
164   delta: 0.05
165   results_per_circuit:
166 - name: u
167   histogram:
168       '00': 3129
169       '10': 1964
170       '01': 2977
171       '11': 1930
172   mitigation_info:
173       target:
174           prob_meas0_prep1: 0.005399999999999996
175           prob_meas1_prep0: 0.0018
176       ancilla:
177           prob_meas0_prep1: 0.00480000000000000265
178           prob_meas1_prep0: 0.0018
179   mitigated_histogram:
180       '11': 0.19408825484296274

```

```

181         '10': 0.19608690118683395
182         '01': 0.2983573535373757
183         '00': 0.3114674904328276
184 - target: 0
185   ancilla: 1
186   phi: 6.283185307179586
187   delta: 0.05
188   results_per_circuit:
189 - name: u
190   histogram:
191     '01': 4768
192     '00': 4696
193     '10': 241
194     '11': 295
195   mitigation_info:
196     target:
197       prob_meas0_prep1: 0.005399999999999996
198       prob_meas1_prep0: 0.0018
199     ancilla:
200       prob_meas0_prep1: 0.00480000000000000265
201       prob_meas1_prep0: 0.0018
202   mitigated_histogram:
203     '10': 0.02325138419612959
204     '11': 0.02889276720310536
205     '00': 0.4685898728546358
206     '01': 0.4792659757461291

```

Appendix B: PyQBench as Python library

In this appendix, we will demonstrate how `qbench` package can be used with user-defined measurement. When used as a library, PyQBench allows the manipulation of certification scheme by, e.g. adding a noise model. The user then defines a unitary matrix U that describes the measurement to be certified, the optimal initial state $|\psi_0\rangle$ and unitaries V_0 and V_1 to create the final measurement. The PyQBench library provides then the following functionalities: assembling circuits for certification scheme, running the whole circuits defining benchmark on specified IBM Q backend and, finally, interpreting the obtained outputs by computing the minimized probability of type II error. Below we present the usage example which can be found in².

We will present the certification of the measurement performed in the Hadamard basis to show the usage of PyQBench as a Python library. The explicit formula for initial state in this case reads

$$|\psi_0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \quad (1)$$

with the final measurements of the forms

$$V_0 = \begin{pmatrix} \sqrt{1-\delta} & \sqrt{\delta} \\ -\sqrt{\delta} & \sqrt{1-\delta} \end{pmatrix}, \quad (2)$$

and

$$V_1 = \begin{pmatrix} -\sqrt{\delta} & \sqrt{1-\delta} \\ \sqrt{1-\delta} & \sqrt{\delta} \end{pmatrix}, \quad (3)$$

for some choice of the statistical significance $\delta \in [0, 1]$. Thus, the minimized probability of type II error is equal to

$$p_{\text{II}} = \frac{1}{2}(\sqrt{1-\delta} - \sqrt{\delta})^2. \quad (4)$$

In our example, let assume $\delta = 0.05$. To use the above benchmarking scheme in PyQBench, we first need to construct circuits that can be executed by actual hardware. The circuit taking $|00\rangle$ to the Bell state $|\psi_0\rangle$ comprises the Hadamard gate followed by

CX gate on both qubits. For $V_0^\dagger$ and $V_1^\dagger$ observe that $V_0^\dagger = \text{RY}\left(2 \arcsin\left(\sqrt{0.05}\right)\right)$ and $V_1^\dagger = X \cdot \text{RY}\left(2 \arcsin\left(\sqrt{0.05}\right)\right)$ where $\text{RY}(\theta)$ is rotation gate around the Y axis defined by

$$\text{RY}(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}. \quad (5)$$

We will now demonstrate how to implement this theoretical scheme in PyQBench with some modifications. For this example we will use the Qiskit Aer simulator³. First, we import the necessary functions and classes from PyQBench and Qiskit.

Listing 6. Importing the necessary functions and classes

```
1 from qiskit_aer import StatevectorSimulator
2 from qiskit import QuantumCircuit
3 import numpy as np
4 from qbench.schemes.postselection import
5     benchmark_certification_using_postselection
6 from qbench.schemes.direct_sum import
7     benchmark_certification_using_direct_sum
```

To implement the certification scheme in PyQBench, we need to define all the necessary components as Qiskit instructions. To do so, we first define `QuantumCircuit(2)` acting on qubit 0 and 1 and then use the `instruction()` method.

Listing 7. Defining components for Hadamard experiment

```
1 def state_prep():
2     circuit = QuantumCircuit(2)
3     circuit.h(0)
4     circuit.cx(0,1)
5     return circuit.to_instruction()
6
7 def u_dag():
8     circuit = QuantumCircuit(1)
9     circuit.h(0)
10    return circuit.to_instruction()
11
12 def v0_dag():
13    circuit = QuantumCircuit(1)
14    circuit.ry(2 * np.arcsin(np.sqrt(0.05)), 0)
15    return circuit.to_instruction()
16
17 def v1_dag():
18    circuit = QuantumCircuit(1)
19    circuit.ry(2 * np.arcsin(np.sqrt(0.05)), 0)
20    circuit.x(0)
21    return circuit.to_instruction()
22
23 def v0_v1_direct_sum_dag():
24    circuit = QuantumCircuit(2)
25    circuit.p(-np.pi, 0)
26    circuit.ry(-2 * np.arcsin(np.sqrt(0.05)), 0)
27    circuit.cx(0, 1)
28    return circuit.to_instruction()
```

We also need to construct a backend object, which is an instance Aer simulator.

Listing 8. Defining a backend

```
1 simulator = StatevectorSimulator()
```

Now, when one wishes to run the experiment without any modifications on a given backend, it is enough to run `benchmark_certification_using_postselection` or `benchmark_certification_using_direct_sum` function, depending on the user preference.

Listing 9. Simulation benchmark by using postselection

```
1 postselection_result =
2     benchmark_certification_using_postselection(
3         backend=simulator,
4         target=0,
5         ancilla=1,
6         state_preparation=state_prep(),
7         u_dag=u_dag(),
8         v0_dag=v0_dag(),
9         v1_dag=v1_dag(),
10        num_shots_per_measurement=10000)
```

Listing 10. Simulation benchmark by using direct sum

```
1 direct_sum_result = benchmark_certification_using_direct_sum(
2     backend=simulator,
3     target=0,
4     ancilla=1,
5     state_preparation=state_prep(),
6     u_dag=u_dag(),
7     v0_v1_direct_sum_dag=v0_v1_direct_sum_dag(),
8     num_shots_per_measurement=10000)
```

The `postselection_result` and `direct_sum_result` variables contain now the empirical probabilities of type II error. We can compare them with the theoretical value and compute the absolute error.

Listing 11. Examining the benchmark results

```
1 p_succ = (1/np.sqrt(2) * np.sqrt(0.95) +
2         - 1/np.sqrt(2) * np.sqrt(0.05))**2
3 print(f"Analytical_p_succ={p_succ}")
4 print(f"Postselection: _p_succ={postselection_result},
5 abs_error={np.abs(p_succ-_postselection_result)}")
6 print(f"Direct_sum: _p_succ={direct_sum_result},
7 abs_error={np.abs(p_succ-_direct_sum_result)}")
```

```
1 Analytical p_succ = 0.2820550528229661
2 Postselection: p_succ = 0.28322830780328,
3     abs_error = 0.001173254980313898
4 Direct_sum: p_succ = 0.28333, abs_error = 0.0012749471770339138
```

But what if we want to modify this process? PyQBench provides functions performing: assembly of circuits needed for experiment under providing the components and interpretation of the obtained measurements.

For the rest of this example, we focus only on the postselection case, as the direct sum case is analogous. We continue by importing two more functions from PyQBench.

Listing 12. Assembling circuits

```
1 from qbench.schemes.postselection import (
2     assemble_circuits_certification_postselection,
3     compute_probabilities_certification_postselection)
4
5 circuits = assemble_circuits_certification_postselection(
6     target=0,
```

```

7     ancilla=1,
8     state_preparation=state_prep(),
9     u_dag=u_dag(),
10    v0_dag=v0_dag(),
11    v1_dag=v1_dag())

```

The function `assemble_circuits_certification_postselection` creates two circuits and places them in a dictionary with keys "u_v0", "u_v1". Now we will run our circuits using noisy and noiseless simulation. We start by creating a noise model using Qiskit.

Listing 13. Adding noise model

```

1 from qiskit_aer.noise import NoiseModel, ReadoutError
2
3 error = ReadoutError([[0.75, 0.25], [0.8, 0.2]])
4
5 noise_model = NoiseModel()
6
7 noise_model.add_readout_error(error, [0])
8 noise_model.add_readout_error(error, [1])

```

Now we can execute the circuits with and without noise. To do this, we will use Qiskit's `run` function. It should be mentioned that we have to keep track of which measurements correspond to which circuit. We do so by fixing an ordering on the keys in the circuits dictionary.

Listing 14. Running circuits

```

1 keys_ordering = ["u_v0", "u_v1"]
2
3 all_circuits = [circuits[key] for key in keys_ordering]
4
5 counts_noisy = simulator.run(
6     all_circuits,
7     backend=simulator,
8     noise_model=noise_model,
9     shots=100000).result().get_counts()
10
11 counts_noiseless = simulator.run(
12     all_circuits,
13     backend=simulator,
14     shots=100000).result().get_counts()

```

Finally, we use `compute_probabilities_certification_postselection` function to compute discrimination probabilities.

Listing 15. Computing probabilities

```

1 prob_succ_noiseless =
2 compute_probabilities_certification_postselection(
3     u_v0_counts=counts_noiseless[0],
4     u_v1_counts=counts_noiseless[1],)
5
6 prob_succ_noisy =
7 compute_probabilities_certification_postselection(
8     u_v0_counts=counts_noisy[0],
9     u_v1_counts=counts_noisy[1],)

```

We can now examine the results. From the experiment, we obtain `prob_succ_noiseless = 0.28072503092454` and `prob_succ_noisy = 0.77564942534484`. As expected, for noisy simulations, the result lies further away from the target value of 0.2820550528229661. This concludes our example.

References

1. Mthree package documentation. <https://qiskit.org/documentation/partners/mthree/stubs/mthree.M3Mitigation.html>. Accessed on 2025-01-01.
2. Hadamard certification experiment. https://github.com/iitis/PyQBench/blob/master/Hadamard_certification_experiment.ipynb. Accessed on 2025-07-01.
3. Qiskit Aer GitHub Repository. https://qiskit.github.io/qiskit-aer/stubs/qiskit_aer.StatevectorSimulator.html. Accessed on 2025-01-01.