IBM's Opens Science Prize 2021 | The Iterative Bayesian Unfolding Generator

Article · October 2022

2 authors:

Rabah Hacene Benaissa
Saad Dahlab University

Imene Ouadah
Saad Dahlab University

IBM'S OPENS SCIENCE PRIZE 2021

THE ITERATIVE BAYESIAN UNFOLDING GENERATOR

Rabah Hacene Benaissa,[a] Imene Ouadah[b]

Physics Department, Faculty of Sciences, University Saad Dahleb blida 1, P.O.Box 270, Soumaa Road, Blida, 09000 Algeria

(Dated: October 22, 2022)

Abstract

Our proposed solution to the IBM's open science prize 2021 is simply to apply an error mitigation process. We will use a technique well known in the field of high energy physics called the iterative Bayesian unfolding. But we will apply it through a new way which will greatly increases its error-mitigation capacities. Our solution is short, simple and fast.

I. INTRODUCTION

The IBM's open science prize 2021 aim to simulate the evolution of the $|110\rangle$ state under the XXX Heisenberg model Hamiltonian on IBM Quantum's 7-qubits Jakarta system with the best fidelity as possible using Trotterization [1]. We will use the iterative Bayesian unfolding (IBU) (also known as Richardson-Lucy deconvolutio)[2][3] to increase the fidelity. The IBU first offered to the QIS community by Benjamin Nachman in 2020 [4], which shows that this technique is powerful than `matrix inversion` and also the `least squares` technique as shown also in [5].

This paper is organized as follows. Sec. II. introduces the Iterative Bayesian Unfolding (IBU). Sec. III. Introduces the IBU Generator (IBUG). Sec. IV. Simulating the $|110\rangle$ evolution under the XXX Heisenberg model Hamiltonian. Important results in Sec. V. The conclusion is in Sec. VI.

II. THE ITERATIVE BAYESIAN UNFOLDING

To correct for detector effects, high energy physics experimentalists has developed many unfolding (correction) algorithms and a widely used one is the IBU technique, which derives from Bayes' theorem. It's formula is written as

$$t_i^{n+1} = \sum_j \left(\frac{\mathcal{R}_{ji}\, t_i^n}{\sum_k \mathcal{R}_{jk}\, t_k^n} \right) m_j, \tag{1}$$

where n is the number of iterations, t is the true distribution, m is the measured distribution and $\mathcal{R}$ is the response matrix. The unfolding procedure starts by choosing a prior *truth spectrum* t^0 from the best knowledge of the process under study. Otherwise, one can take it as a uniform distribution $t_i^0 = N_{\text{shot}}/2^{n_{\text{qubit}}}$, where N_{shot} is the number of shots. The number of iterations needed to converge depends on the desired precision. The implementation of IBU in python language is given as below:

```
def IBU(m,t0,Rin,n):
    tn = t0
    for i in range(n):
        Rjitni = [
        np.array(Rin[:][i])*tn[i] for i
                        in range(len(tn))]
        Pm_given_t = Rjitni /np.matmul(Rin,tn)
        tn = np.dot(Pm_given_t,m)
        pass
    return tn
```

This technique is powerful but it remains incapable of significantly attenuate the errors of the open prize situation; so for that we aim in this work to use IBU through a new way which considerably increases its efficiency.

III. THE IBU GENERATOR TECHNIQUE

The new way we propose is to iterate IBU itself. In this way one have a simple concept, which is to consider the corrected vector (or counts) as a noisy one and so we can correct it again and we can repeat this procedure several times, which reduces errors. mathematically speaking we can define a function $\mathcal{B} : m \mapsto t$, i.e. $t = \mathcal{B}(m)$ or $m^{[1]} = \mathcal{B}(m^{[0]})$, where $m^{[1]} \equiv t$ and $m^{[0]} \equiv m$ which is the original measured vector (counts). Giving this, the recursion procedure can be written as

$$m^{[1]} = \mathcal{B}(m^{[0]}) \to \cdots \to m^{[p]} = \mathcal{B}(m^{^{[p-1]}}), \tag{2}$$

or we can also write

$$m^{[p]} = \underbrace{\mathcal{B}(\cdots(\mathcal{B}(\mathcal{B}(m^{[0]}))\cdots)}_{(p-1)\ times}, \tag{3}$$

where $p \in \mathbb{N}$ is the number of IBU recurrences. To implement it in python, we need to define a generator as shown below:

```
def IBU_generator(c,t0,Rin,n):
    c0 = to_list(c)
    c1 = IBU(c0,t0,Rin,n)
    while(True):
        c2 = IBU(c1,t0,Rin,n)
        yield c2
        c0,c1=c1,c2
```

[a] rhbenaissa@gmail.com
[b] imene.ouadahpsi@gmail.com

Here the `c` argument denotes the original noisy counts. As IBU handles with a list form. Therefore, we have to convert our counts. Hence, the `to_list()` function perform this conversion. For clarity, let's take a quick example:

```
# after an experiment on a 2-qubit system, we get:
counts = {'01':50,'11':10,'00':12,'10':48 }

# arrangement and convertion:
my_list = to_list(counts)
print(my_list)
```

```
# output:
[12, 50, 48, 10]
```

After defining the `IBU_generator` function, we define the `IBUG` function which returns the p^{th} result of the error-mitigation process by storing p value of `next(g)` in a list `k`, then extracting only the p^{th} element by `k[-1]`.

```
def IBUG(counts, t0, Rin, n, p):
    g = IBU_generator(counts,t0,Rin,n)
    k = [next(g) for x in range(p)]
    return k[-1]
```

IV. SIMULATING THE XXX MODEL

The Heisenberg spin model Hamiltonian for 3 spin-*(1/2)* aligned in a chain is written as follow:

$$H_{\text{Heis3}} = \sum_{\langle ij \rangle}^{N=3} J\left(\sigma_x^{(i)}\sigma_x^{(j)} + \sigma_y^{(i)}\sigma_y^{(j)} + \sigma_z^{(i)}\sigma_z^{(j)}\right), \quad (4)$$

where $\sigma_x, \sigma_y, \sigma_z$ are Pauli matrices acting on the i^{th} and j^{th} qubit. We take $N = 3, J = 1$. Its corresponding evolution operator can be written as

$$U_{\text{Heis3}}(t) = e^{-iH_{\text{Heis3}}\, t}. \quad (5)$$

For our proposed solution, we follow the following steps:

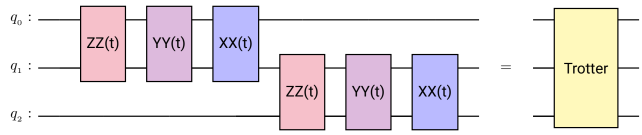
1. Decomposing $U_{\text{Heis3}}(t)$ into quantum gates.
2. Executing the simulation circuit in `ibmq_jakarta`.
3. Error mitigation using IBUG.
4. Compute the state tomography fidelity.

STEP 1. DECOMPOSING THE EVOLUTION OPERATOR INTO GATES

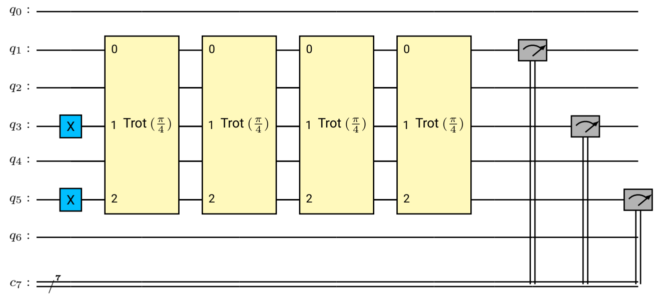
We decompose the evolution operator $U_{\text{Heis3}}(t)$ by the same way done in the provided Jupyter Notebook in order to get:

$$U_{\text{Heis3}}(t) \approx \Big[XX\left(\tfrac{2t}{n}\right)^{(0,1)} YY\left(\tfrac{2t}{n}\right)^{(0,1)} ZZ\left(\tfrac{2t}{n}\right)^{(0,1)} \\ XX\left(\tfrac{2t}{n}\right)^{(1,2)} YY\left(\tfrac{2t}{n}\right)^{(1,2)} ZZ\left(\tfrac{2t}{n}\right)^{(1,2)} \Big]^n \quad (6)$$

Where $XX(2t) = e^{-it\sigma_x \otimes \sigma_x}$, same thing for $ZZ(2t)$ and $YY(2t)$. So for $n = 1$, $U_{\text{Heis3}}(t)$ can be drawn as follow:



CIRC. 1. Combine subcircuits into a single gate representing one Trotter step (Trotter).



CIRC. 2. Time evolving the $|110\rangle$ state to time $t = \pi$ using four Trotter steps.

STEP 2. EXECUTING THE TIME EVOLUTION CIRCUIT

For execution, we take 11 Trotter steps and add just an `optimization_level=3`, keeping every thing else unchanged.

```
shots = 8192
reps = 8
backend = jakarta

jobs = []
for _ in range(reps):
    # execute
    job = execute(st_qcs, backend, shots=shots,
                  optimization_level=3)
    print('Job ID', job.job_id())
    jobs.append(job)
```

STEP 3. ERROR MITIGATION USING IBUG

Now we have to mitigate the errors, we know that IBUG handles with a dict type, so first we need to extract the `counts` (named `old_counts`) then IBUG correct them (`new_counts`) and finally we push them again to results (`new_result`). Here we are using as parameters: `t0` as a uniform distribution, `n=2` and `p=36`.

```
RES = []
for job in jobs:
    my_result = job.result()
    new_result = deepcopy(my_result)

    for resultidx,_ in enumerate(my_result.results):
        # extract counts from my_result
        old_counts = my_result.get_counts(resultidx)
        # begin unfolding (error mitigtion)
        tp = IBUG(counts=old_counts, t0=np.ones(
                len(R)),Rin=R, n=2, p=36)
        # the new corrected counts
        new_counts = dict(zip(state_labels,tp))
        # push new counts back into the new result
        new_result.results[resultidx].data.counts =
```

```
        new_counts

    RES.append(new_result)
```

STEP 4. COMPUTE THE STATE TOMOGRAPHY FIDELITY

To compute the state tomography fidelity (STF), we use the `state_tomo` function provided in code:

```
#Compute tomography fidelities for each repetition
fids = []
for res in RES:
    fid = state_tomo(res, st_qcs)
    fids.append(fid)

print('state tomography fidelity = {:.4f} \u00B1
      '{:.4f}'.format(np.mean(fids), np.std(fids)))
```

```
# output:
state tomography fidelity = 0.6297 + 0.0299
```

While the fidelity value without error mitigation step is 0.2459 ± 0.0054.

ERROR MITIGATION	STATE TOMOGRAPHY FIDELITY
not mitigation	0.2459 ± 0.0054
pseudo inverse	0.2666 ± 0.0064
least squares	0.2666 ± 0.0064
IBU	0.2861 ± 0.0080
IBUG	0.6297 ± 0.0299

TAB. 1. Maximum STF values using different error mitigation methods. The IBU is used with $n = 10$ iterations and IBUG with $n = 2, p = 36$, where IBUG show about 38% of improvement over the unmitigated value.

V. RESULTS

Figure 1 shows the variation of STF as a function of p where we notice that it increases up to a maximum of 0.6297 ± 0.0299 when $p = 36$. Unfortunately, after the maximum point, the curve begins to decrease.

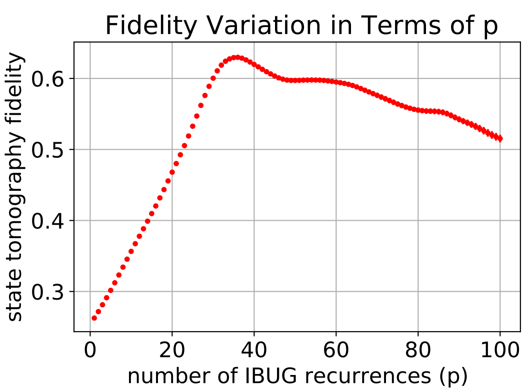


FIG. 1. Variation of state tomography fidelity in terms of IBUG recurrences.

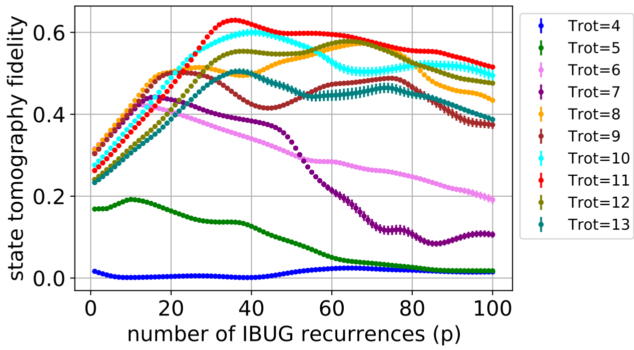


FIG. 2. Variation of state tomography fidelity in terms of number of IBUG recurrences for several trotter steps.

TROTTER STEPS	MAXIMUM S.T.F.	(P) NUMBER
04	0.0244 ± 0.0047	64
05	0.1920 ± 0.0024	10
06	0.4237 ± 0.0040	12
07	0.4423 ± 0.0084	16
08	0.5730 ± 0.0099	68
09	0.5014 ± 0.0113	22
10	0.6000 ± 0.0943	40
11	0.6297 ± 0.0299	36
12	0.5783 ± 0.0436	64
13	0.5042 ± 0.0817	37

TAB. 2. Maximum STF values for several trotter steps with error-mitigation via IBUG ($n = 2$).

We note that the best STF value is for 11 trotter steps. But it is important to note that this is not a fixed value for any experiment, as in many cases the best value is for 8 trotter steps; there is a high probability of obtaining the best STF value for 8 trotter steps.

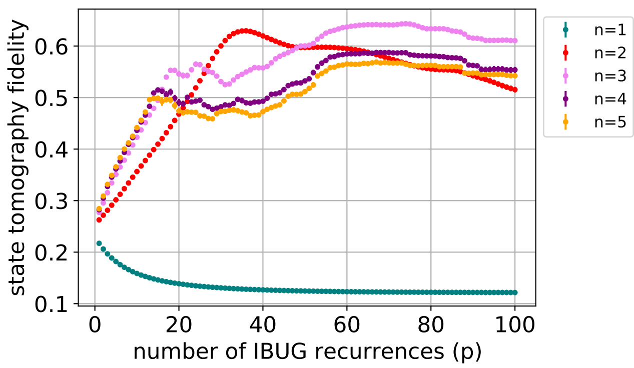


FIG. 3. Variation of STF in terms of number of IBUG recurrences for several number of iterations.

Figure 3 shows the variation of STF as a function of p for several n values, where we can see that for $n = 1$ error mitigation fails. The curves for $n \geq 3$ have a similar evolution and increase faster compared to the curve of $n = 2$ in the interval $p \in [1, 25]$. For $n \geq 5$, all the curves are very close to each other.

NUM OF ITERATIONS	MAXIMUM S.T.F.	(P) NUM
1	0.2171 ± 0.0038	01
2	0.6297 ± 0.0299	36
3	0.6434 ± 0.0214	74
4	0.5876 ± 0.0319	67
5	0.5690 ± 0.0313	67
6	0.5658 ± 0.0376	69
7	0.5659 ± 0.0334	67
8	0.5660 ± 0.0335	67
9	0.5661 ± 0.0337	67

TAB. 3. Maximum STF values for several number of iterations with error-mitigation via IBUG.

From Tab. 3, we observe two maximum STF values, for 2 and 3 iterations with $p = 36$ and $p = 74$ respectively. For a high value of n ($n \geq 5$) the values of STF and p stabilize. We note that the best STF value is obtained for $n = 3$ ($p = 74$). It is importante to note that the execution time of IBUG increases when the number of iterations increase. For this, a few iterations are enough to mitigate errors.

VI. CONCLUSION

The proposed technique (IBUG) represents the simplest solution to increase simulation fidelity. Compared to other primary methods (pseudo inversion, least squares and also IBU itself) IBUG have a higher mitigation capacity. For fast and effective error mitigation, few iterations should be performed (starting from $n = 2$). The challenge posed by IBUG is how to predict the best value of the number of recurrences (p), which is the subject of future works. The IBUG as an error mitigation method is interesting for both fields, quantum computing and high energy physics.

VII. DATA AVAILABILITY

The code for the work presented here is available at: https://github.com/QuanTeamOB/IBM_Open_Science_Prize_2021

APPENDIX A. THE RESPONSE MATRIX

The response matrix[1] that we used for error-mitigation is presented in Figure 4.

```
qr = QuantumRegister(7)
qubit_list = [1,3,5]  # q_1, q_3, q_5
meas_calibs, state_labels =
complete_meas_cal(qubit_list,qr, circlabel='mcal')
jobr = execute(meas_calibs, backend=jakarta,
                         shots=10**5)
cal_res = jobr.result()
meas_fitter = CompleteMeasFitter(cal_res,
           state_labels, circlabel = 'mcal')
R = meas_fitter.cal_matrix
```

[1] The response matrix represents a principle ingredient for IBU and IBUG as well. Its elements change every time we measure it, but this change don't have a high influence of the error-mitigation process.

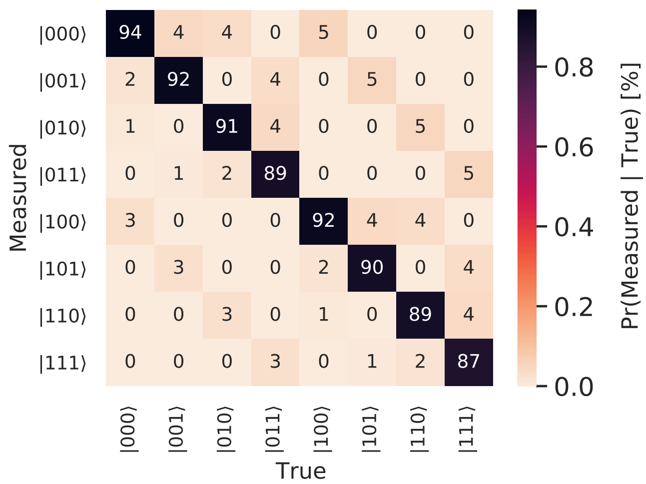


FIG. 4. The `ibmq_jakarta` specefic response matrix for the three qubits q_1, q_3, q_5 using 10^5 shots.

APPENDIX B. THE DICT TO LIST CONVERTER

As stated before, the IBU deals with a list form, so we use the `to_list()` function to sort and then convert the counts (whose type is a dict) to a list type.

```
def to_list(mycounts):
    # sorting the counts dictionary
    d = dict(sorted(mycounts.items(),
                    key = lambda x:x[0]))
    return list(d.values())
```

APPENDIX C. THE LEAST SQUARES GENERATOR

Through the same algorithm used to implement the IBU generator, we can construct the least squares generator (LSQRG). Fig. 5 shows its mitigation effects compared to IBUG, where we observe a small improvement of STF values with a maximum of 0.4516 ± 0.1063 (for $p = 14$). For $p > 20$, STF values converge to a constant value (0.1931 ± 0.0153). Therefore, LSQRG is only effective for a few recurrences. Moreover the execution time of LSQRG is too long; the LSQRG curve present in Fig. 5 takes around 3 hours to complete, but the IBUG takes around 4 minutes.

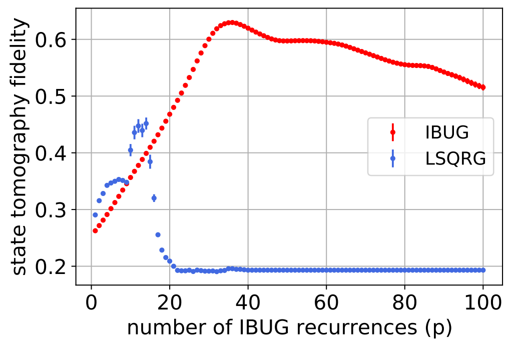


FIG. 5. Variation of STF in terms of p for IBUG and LSQRG.

APPENDIX D. COMPUTING THE STATE TOMOGRAPHY FIDELITY

To compute the STF, we use the same function provided by the Jupyter Notebook of the open prize.

```
# Compute the state tomography based on the
# st_qcs quantum circuits and the results from
# those circuits

def state_tomo(result, st_qcs):
    # The expected final state; necessary to
    # determine state tomography fidelity
    target_state = (One^One^Zero).to_matrix()
    # Fit state tomography results
    tomo_fitter = StateTomographyFitter(result,
                                        st_qcs)
    rho_fit = tomo_fitter.fit(method='lstsq')
    # Compute fidelity
    fid = state_fidelity(rho_fit, target_state)
    return fid
```

APPENDIX E. UNFOLDING A GAUSSIAN DISTRIBUTION

Figure 6 shows a comparison of the four error mitigation methods used (pseudo inverse, least squares, IBU, IBUG). Note that the values closest to the truth are those obtained via IBUG.

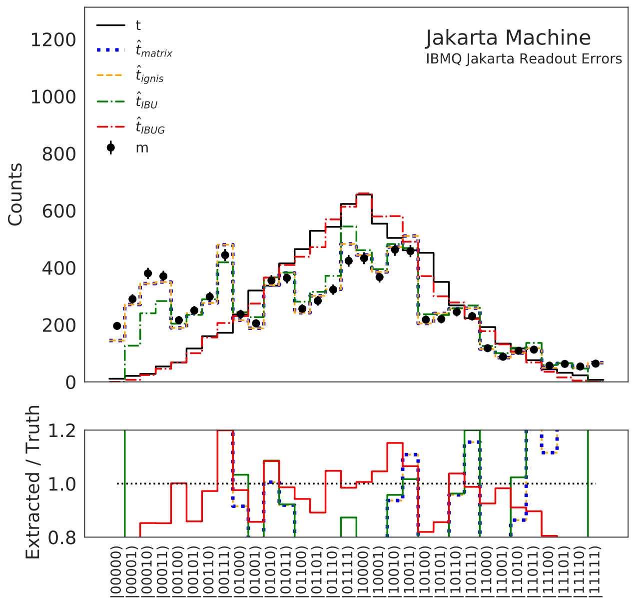


FIG. 6. Unfolding results of a Gaussian distribution based on 5-qubits of the `ibmq_jakarta` machine using four methods: `pseudo inverse` ($\hat{t}_{\text{matrix}}$), `least squares` ($\hat{t}_{\text{ignis}}$), `IBU` ($\hat{t}_{\text{IBU}}$), and `IBUG` ($\hat{t}_{\text{IBUG}}$). The `IBU` and `IBUG` both used with one iteration, a Gaussian prior truth spectrum and $p = 15$ for `IBUG`.

REFERENCES

[1] Y. Salathe, et al., *"Digital Quantum Simulation of Spin Models with Circuit Quantum Electrodynamics, Phys"*. URL: Rev. X 5, 021027 (2015).

[2] Giulio D'Agostini, Nucl. Instrum. Methods Phys. Res. A 362, 487 (1995).

[3] L. B. Lucy, Astron. J. 79, 745 (1974).

[4] Benjamin Nachman et al. *"Unfolding quantum computer readout noise"*. In: npj Quantum Information 6.1 (2020), pp. 1–7. DOI: 10.1038/s41534-020-00309-7.

[5] I.Ouadah and R.H.Benaissa. *"Dealing with quantum computer readout noise through high energy physics unfolding methods"*. MA thesis. 2021. URL: https://arxiv.org/abs/2204.05757.