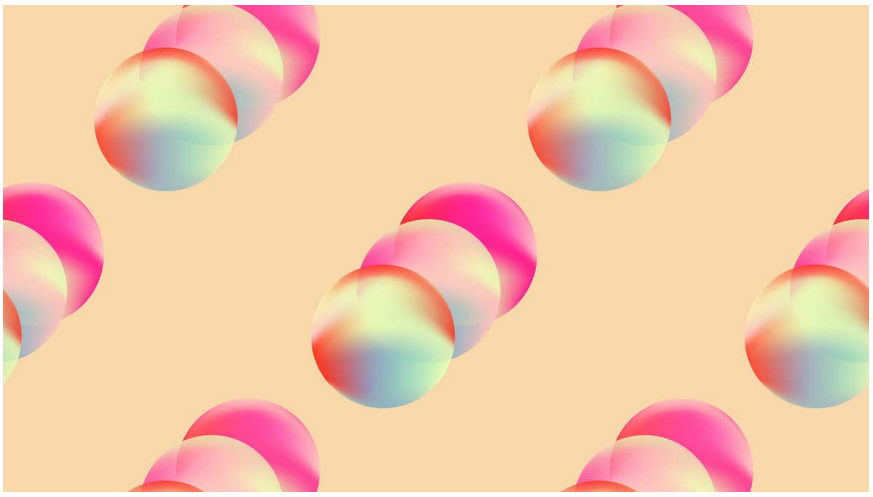
Quantum Computation for Particle Physics

Solomon Malih
Supervisor - Dr. Bipasha Chakraborty

August 2021

Abstract

This research report builds upon the work done in [1] and looks at the use of digital quantum simulation for particle physics. The investigation aims to find if digital quantum simulation can be used to test a gauge theory known as the Schwinger model.



(Image from New Scientist)

Contents

1 Introduction

Quantum mechanics is the branch of physics that deals with the motion and interaction of subatomic particles. In this classical world we live in, energy is a continuous property. A continuous property can take any value within a given range (which may not necessarily be bounded). For example, light is a continuous electromagnetic wave in the classical world, but when we go down to the quantum scale, we see that light is not a continuous wave but rather a stream of discrete packets called photons. Photons are the quanta (singular quantum) of all electromagnetic (EM) waves. Quantum is the term used to refer to the smallest unit of a discrete/quantized property. Note that quantum mechanics alone is not enough for our purposes because it cannot deal with particles that travel at speed comparable to light. To accommodate such high speeds, we would need to introduce Einstein's Theory of Special Relativity [2], which explains how spacetime behaves for objects moving at speed approaching that of light. However, special relativity breaks down on a quantum scale and

produces nonsensical values. What is required is quantum field theory (QFT), which can handle the creation/destruction of high-speed particles and conversion of energy into mass and vice versa on a quantum scale.

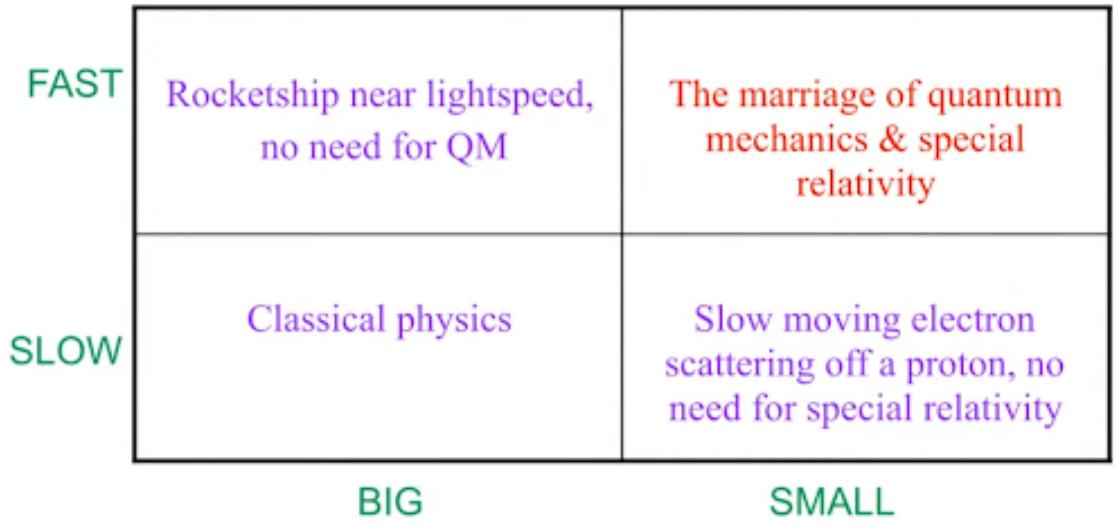


Figure 1: Two-way table showing the necessary areas of physics required for different scenarios (Zee, n.d.)

In quantum field theory, particles and waves are expressed as mathematical objects called fields, which are functions of every spacetime point [3]. Using these fields, one can build equations of motion relating to the particles and waves in question. When two particles interact, they exchange particles called gauge bosons. In quantum chromodynamics (a type of QFT), the gauge boson exchanged is the gluon responsible for the strong force interaction between quarks.

2 Scientific Background

Gauge theories are a type of quantum field theory which are Lagrangian invariant; the Lagrangian does not change under local transformations. The Lagrangian gives the mass term and kinetic energy term for electrons and photons in terms of gauge fields. The mathematics behind this lies beyond the scope of this report[1], but we essentially transform the Lagrangian into Hamiltonian, allowing us to express the gauge fields as fermionic fields. This operation is useful as electrons are half-spin fermion particles and can be associated with Pauli-spin matrices using Jordan-Winger transformations. These Pauli-spin matrices can represent quantum circuit gates, which is how we derived the gates in section 3.

[1]For a more detailed explanation delving into the mathematics, see [1].

3 Circuit

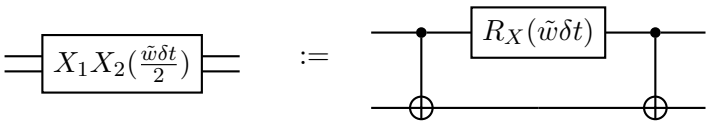
This section covers the various circuit gates describing the implementation of the electron fields; definitions and values for our parameters are given below:

$$
\begin{aligned}
T &= 100, \\
w &= \frac{1}{2a} = 0.5, \\
m_0 &= 1, \\
m &= 0, 0.5, \\
n &= 4, \\
\theta &= \frac{2\pi}{3}, \\
t &= 10, \\
\delta t &= 0.001, \\
g &= 1, \\
a &= 1, \\
J &= \frac{g^2 a}{2}.
\end{aligned}
\tag{1}
$$

where m is the mass of the particle simulated, δt the time evolution interval, n the number of qubits, T the total time, and θ the term for the Schwinger model. We have two values of m, 0 and 0.5, respective of the massless and massive cases.
Moreover, $\tilde{w}$ is defined as

$$\tilde{w} := \frac{t}{T}w - \frac{(-1)^n}{2}((1 - \frac{t}{T})m_0 + m)sin(\theta\frac{t}{T}).$$

3.1 X_1X_2 Gate



The X_1X_2 gate consists of a controlled NOT (CNOT) gate, with the control bit being the first qubit and the target bit the second. A Pauli R_X gate which performs a rotation about the x-axis of $\tilde{w}\delta t$ radians on the first qubit follows the CNOT gate. Another CNOT gate then follows this Pauli gate in the same setup as the previous CNOT gate.

This is an implementation of the X_1X_2 gate in the Python programming language, using the Qiskit package:

```
# Setup circuit incl. quantum and classical registers
```

```
qreg = QuantumRegister(2, 'q')
creg = ClassicalRegister(2, 'c')

circuit = QuantumCircuit(qreg, creg)
circuit.x(qreg[1])
circuit.cx(qreg[0], qreg[1])
circuit.rx(angle1,qreg[0])
circuit.cx(qreg[0], qreg[1])

# measuring qubits from the quantum register unto the classical register
circuit.measure(qreg, creg)
# printing the circuit
circuit.draw(output='mpl')
```

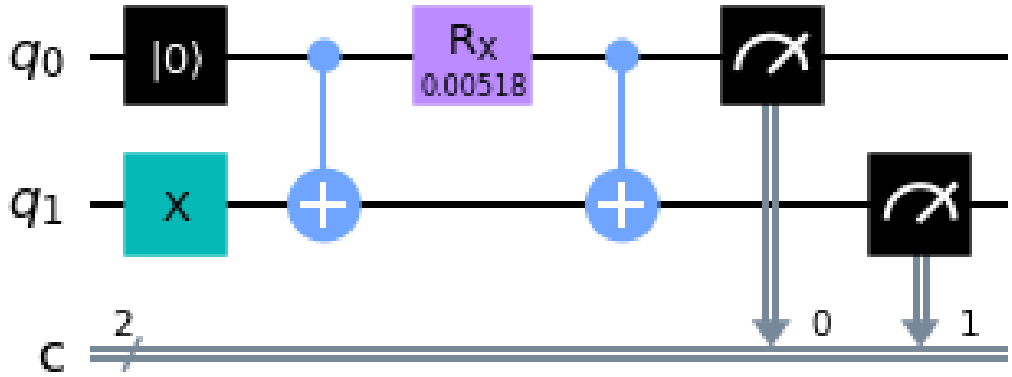
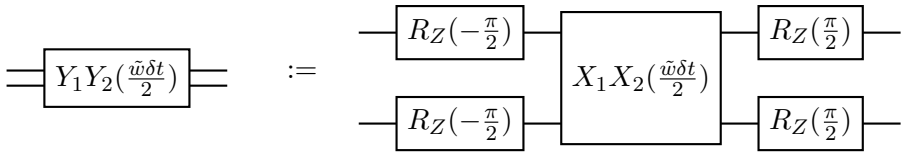


Figure 2: Quantum circuit of the X_1X_2 gate

3.2 Y_1Y_2 Gate



The Y_1Y_2 gate consists of a Pauli R_Z gate applied to each input state, rotating it $-\frac{\pi}{2}$ radians about the z-axis. Next, the X_1X_2 gate described above is applied, completed with

two further R_Z gates, rotating each qubit $+\frac{\pi}{2}$ radians about the z-axis.

This is an implementation of the Y_1Y_2 gate:

```
# Setup circuit incl. quantum and classical registers

qreg = QuantumRegister(2, 'q')
creg = ClassicalRegister(2, 'c')

circuit = QuantumCircuit(qreg, creg)
circuit.x(qreg[1])
circuit.rz(-pi/2, qreg[0])
circuit.rz(-pi/2, qreg[1])
circuit.cx(qreg[0], qreg[1])
circuit.rx(angle1,qreg[0])
circuit.cx(qreg[0], qreg[1])
circuit.rz(pi/2, qreg[0])
circuit.rz(pi/2, qreg[1])

circuit.measure(qreg, creg)
# printing the circuit
circuit.draw(output='mpl')
```

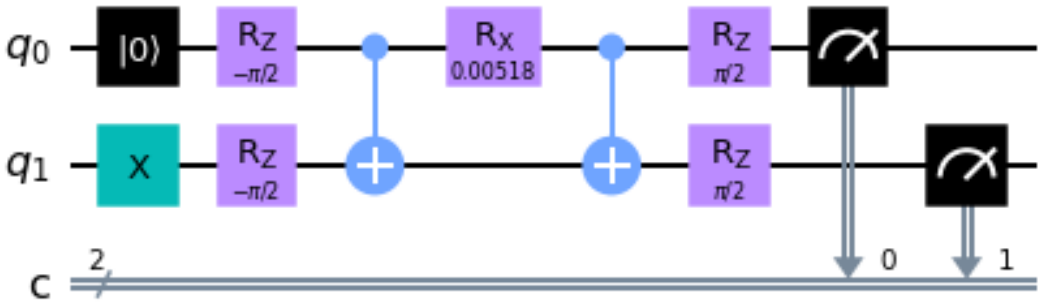
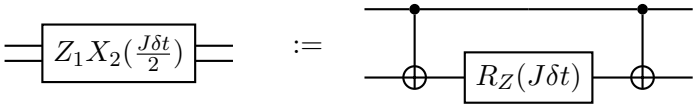


Figure 3: Quantum circuit of the Y_1Y_2 gate

3.3 Z_1Z_2 Gate



The Z_1Z_2 gate is similar to the X_1X_2 gate, except that the gate between the two CNOT gates is an R_Z gate rotating the second qubit $J\delta t$ radians about the z-axis.

This is an implementation of the Z_1Z_2 gate:

```
# Setup circuit incl. quantum and classical registers

qreg = QuantumRegister(2, 'q')
creg = ClassicalRegister(2, 'c')

circuit = QuantumCircuit(qreg, creg)
circuit.x(qreg[1])
circuit.cx(qreg[0], qreg[1])
circuit.rz(j*delta_t, qreg[1])
circuit.cx(qreg[0], qreg[1])

circuit.measure(qreg, creg)
circuit.draw(output='mpl')
```

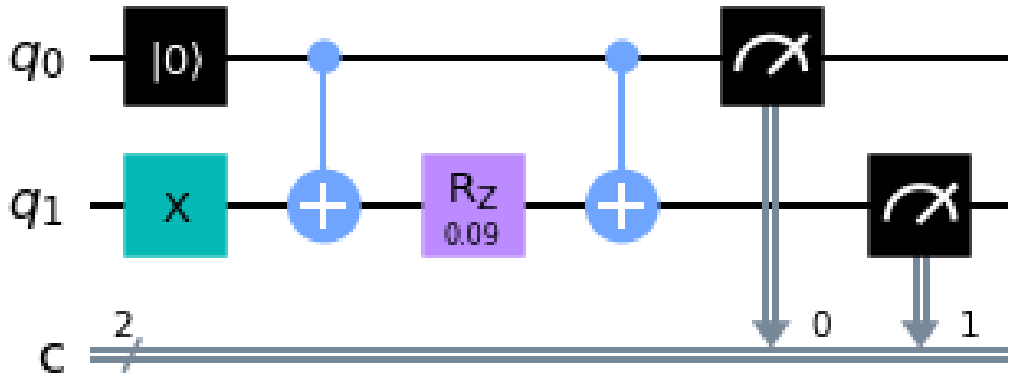
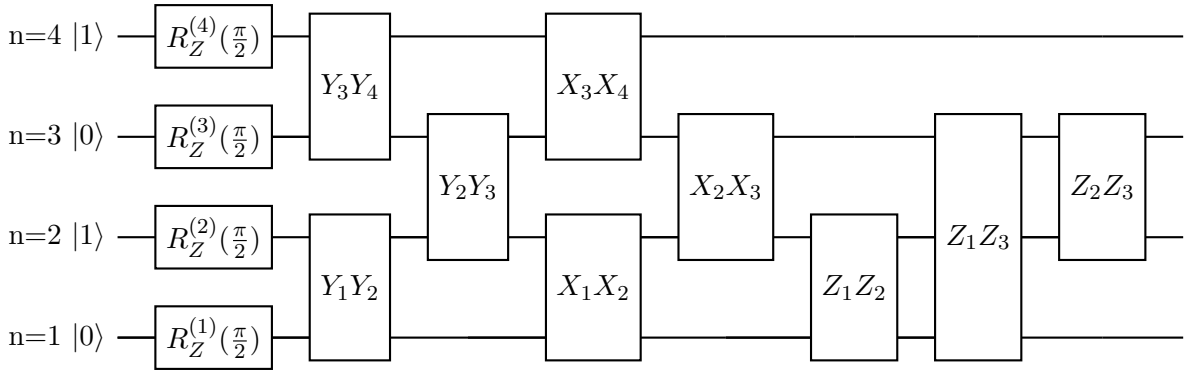


Figure 4: Quantum circuit of the Z_1Z_2 gate

3.4 Full Circuit



The full circuit consists of four qubits, initiated in the $|0\rangle$, $|1\rangle$, $|0\rangle$, $|1\rangle$ states respectively. Each qubit then has the Pauli R_Z gate applied to it, rotating it $\frac{\pi}{2}$ radians about the z-axis. The circuit diagram above shows how the various quantum gates are used in conjunction with each other to represent the half-spin electrons.

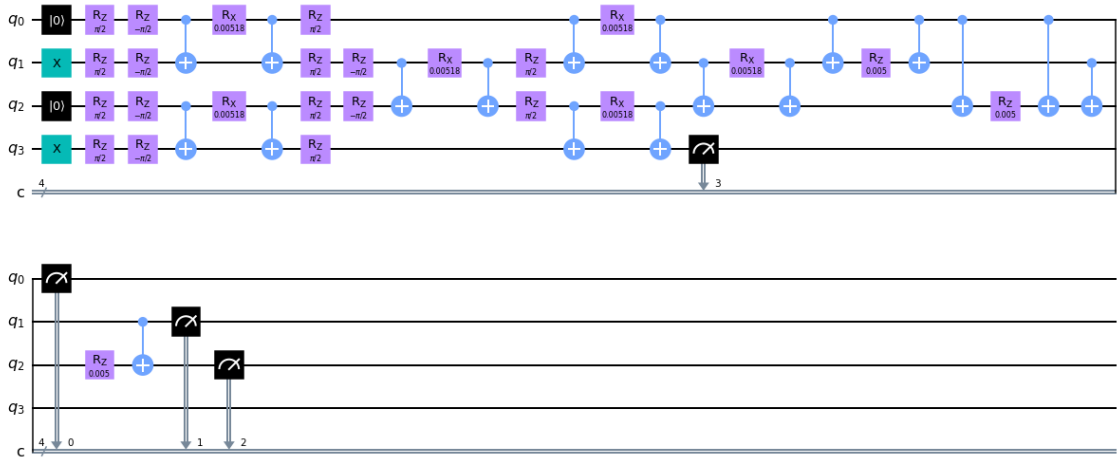


Figure 5: Full quantum circuit

4 Methodology

This section describes how data concerning the output of the complete circuit (Fig 5) was collected.

For my simulations, δt and `shots` were my independent parameters. I ran the quantum simulator for six different values for δt: 0.001; 0.005; 0.01; 0.05; 0.1; 0.5, with `shots` set to

20,000. The variable `shots` describes how many times the simulator runs the simulation. I also ran the quantum simulator for six different values of `shots`: 100; 500; 2,000; 5,000; 10,000; 20,000, with δt set to 0.001 (the value set in section 3). Additionally, the algorithm running the simulator itself was executed ten times, meaning for each value of the independent parameter, the simulation ran a total of `shots` multiplied by ten times. This ensured that the data I was collecting was reliable. See Appendix A for a walkthrough of the algorithm.

5 Data Analysis

This section analyses primary and secondary data relating to the complete quantum circuit.

5.1 Quantum Circuit Data

5.1.1 Independent Parameter - δt

Table 1 shows the state and probability of the state being outputted by the circuit for each value of δt. Any states that were not outputted at all are not included in the table. Probability has been used rather than frequency as quantum mechanics is inherently probabilistic.

	δt					
State	0.001	0.005	0.01	0.05	0.1	0.5
0011	0.0000	0.0000	0.0000	0.0000	0.0000	0.0060
0101	0.0000	0.0000	0.0000	0.0000	0.0000	0.1105
0110	0.0010	0.0010	0.0010	0.0180	0.0870	1.5735
1001	0.0000	0.0050	0.0025	0.0655	0.2810	6.1220
1010	99.9990	99.9975	99.9925	99.8360	99.3740	85.8565
1100	0.0000	0.0010	0.0040	0.0805	0.2580	6.2180
1111	0.0000	0.0000	0.0000	0.0000	0.0000	0.1135

Table 1: Circuit Output - Independent Parameter δt

Table 1 shows that using a very small value of δt will cause the circuit to output $|1010\rangle$ with near 100% probability. However, as we begin to increase the value of δt, the general trend is that the probability of measuring $|1010\rangle$ falls away, and the probability of measuring other states increases. Interestingly, although no formal statistical test of significance has been conducted to confirm this, the data suggests that the states measured are not random. The table shows that the other states are introduced progressively as the value of δt increases. It is not the case that a state appears for a low value of δt (e.g., $\delta t = 0.001$) and then disappears again. Once measured, the state continues to be measured. Moreover,

states first measured for smaller values of δt have a greater probability of being measured at $\delta t = 0.5$ than states first measured later on for larger values of δt.

5.1.2 Independent Parameter - `shots`

Table 2 shows the state and probability of the state being outputted by the circuit for each value of `shots`.

	`shots`					
State	100	500	2,000	5,000	10,000	20,000
0110	0.0000	0.0200	0.0000	0.0080	0.0010	0.0015
1001	0.0000	0.0000	0.0100	0.0040	0.0020	0.0015
1010	100.0000	99.9800	99.9900	99.9860	99.9950	99.9955
11100	0.0000	0.0000	0.0000	0.0020	0.0020	0.0015

Table 2: Circuit Output - Independent Parameter `shots`

Table 2 shows increasing the number of shots (at $\delta t = 0.001$) slightly increases the probability of states other than $|1010\rangle$ being measured. However, Table 2 also reveals for example that $|0110\rangle$ had a greater probability of being measured at 500 shots than any subseqeunt greater number of shots. Beyond introducing new states, increasing `shots` does not really increase the probability of each state being measured. This point is reinforced by the fact that the jump from 2,000 to 20,000 shots saw a decrease in the probability of the $|1001\rangle$ state being measured. Compared to Table 1 where the probabilities are strictly increasing as δt increases, the probabilities in Table 2 seem more varied.

5.2 Regression Analysis

An explanation of the dataset in Table 3 and what it represents unfortunately lies beyond the scope of this report and was not covered during the placement. However, once again, [1] contains an in-depth explanation of this data and how it has been obtained. Various regression analysis algorithms were performed on the dataset to determine the model that best fits the data. The models tested were linear ($y = mx + c$), quadratic ($y = ax^2 + bx + c$) and exponential ($a + e^{xb-c}$). The R^2 and adjusted R^2 statistical measures used to assess the goodness of fit reveal just how much variance in the independent parameter determines variance in the independent parameter. The adjusted R^2 measure takes into account the number of 'features', or in our case, the $\{a, b, c, m\}$ terms, and is always lower than the R^2 value. In order to perform the regression analysis, the Scikit-learn library was used..

$\frac{w}{N}$	$\langle\bar{\psi}\psi\rangle$
0.125	-0.23911894987506038
0.08333333333333333	-0.2395207628740405
0.0625	-0.23785917474471302
0.05	-0.23637765033965558
0.041666666666666664	-0.23555598888423035
0.03571428571428571	-0.2346960333464121

Table 3: Dataset

The best linear fit of this dataset is $y = -0.05040470830441562x - 0.23384278085718985$. This fit produced an R^2 goodness measure of 0.737377 and an adjusted R^2 goodness measure of 0.671721.

	Regression Model		
Goodness Measure	Linear	Quadratic	Exponential
R^2	0.737377	0.995268	0.962054
Adjusted R^2	0.671721	0.992114	0.936756

Table 4: Goodness measures used to assess the goodness of fit to the dataset

Using Table 4, we can see that the quadratic model best fits our dataset. The quadratic model has the highest R^2 and adjusted R^2 values, suggesting that the independent parameter most dramatically affects the dependent variable in the quadratic model; or in other words, changes in the independent parameter in the quadratic model most strongly correlate to actual changes in the dependent variable. In the quadratic model, there is a slight decrease from R^2 to adjusted R^2, suggesting that we are using an appropriate power of x. On the other hand, the exponential model sees a decrease of 0.3, suggesting that one or more of the $\{a, b, c\}$ terms improve the model by less than expected. We can conclude

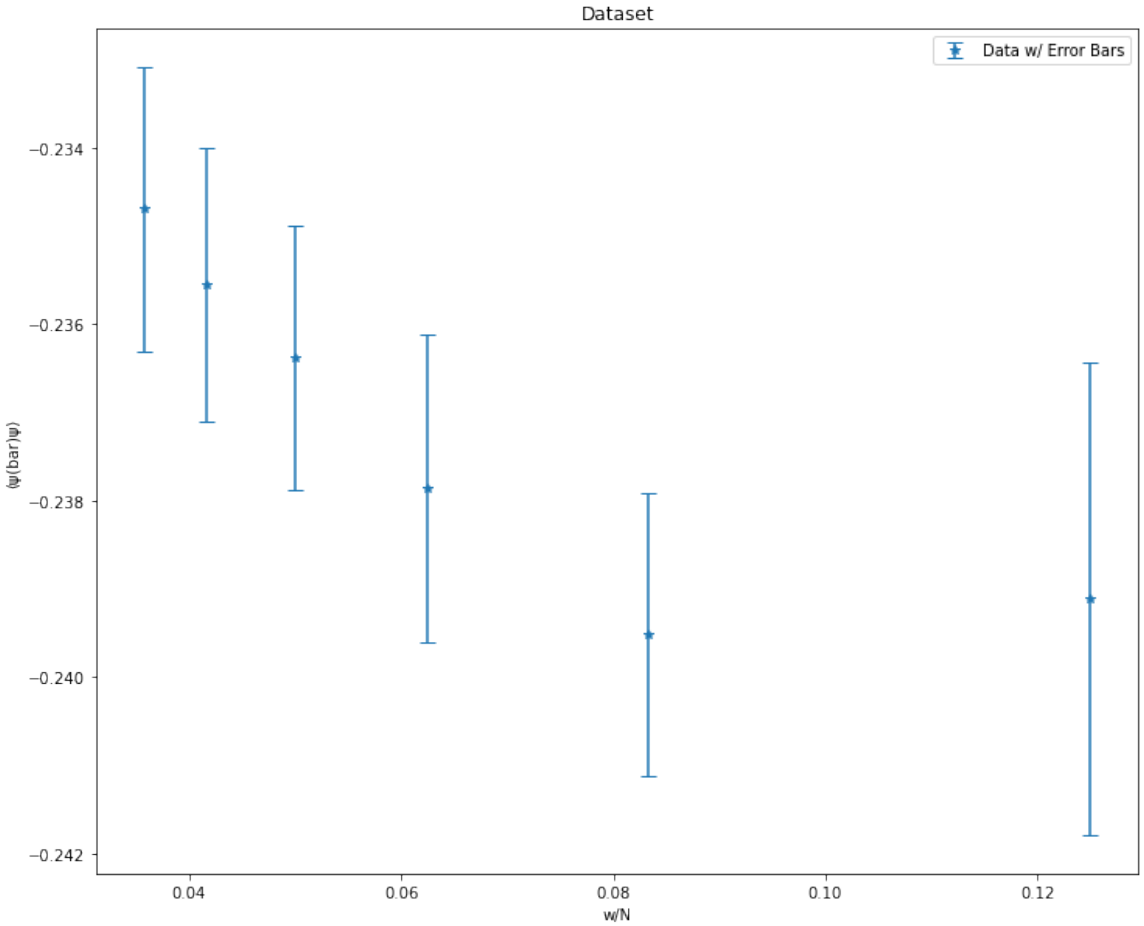


Figure 6: Dataset Plot

that the $\{a, b, c\}$ terms of the quadratic model make a more significant contribution to the model's overall performance than those of the exponential model. However, note that the goodness of fit values produced for the exponential model are very high and that the model is still relatively accurate. The linear model performs the worst by all goodness measures in the table; this could have been deduced by simply inspecting the nature of the dataset in Figure 6.

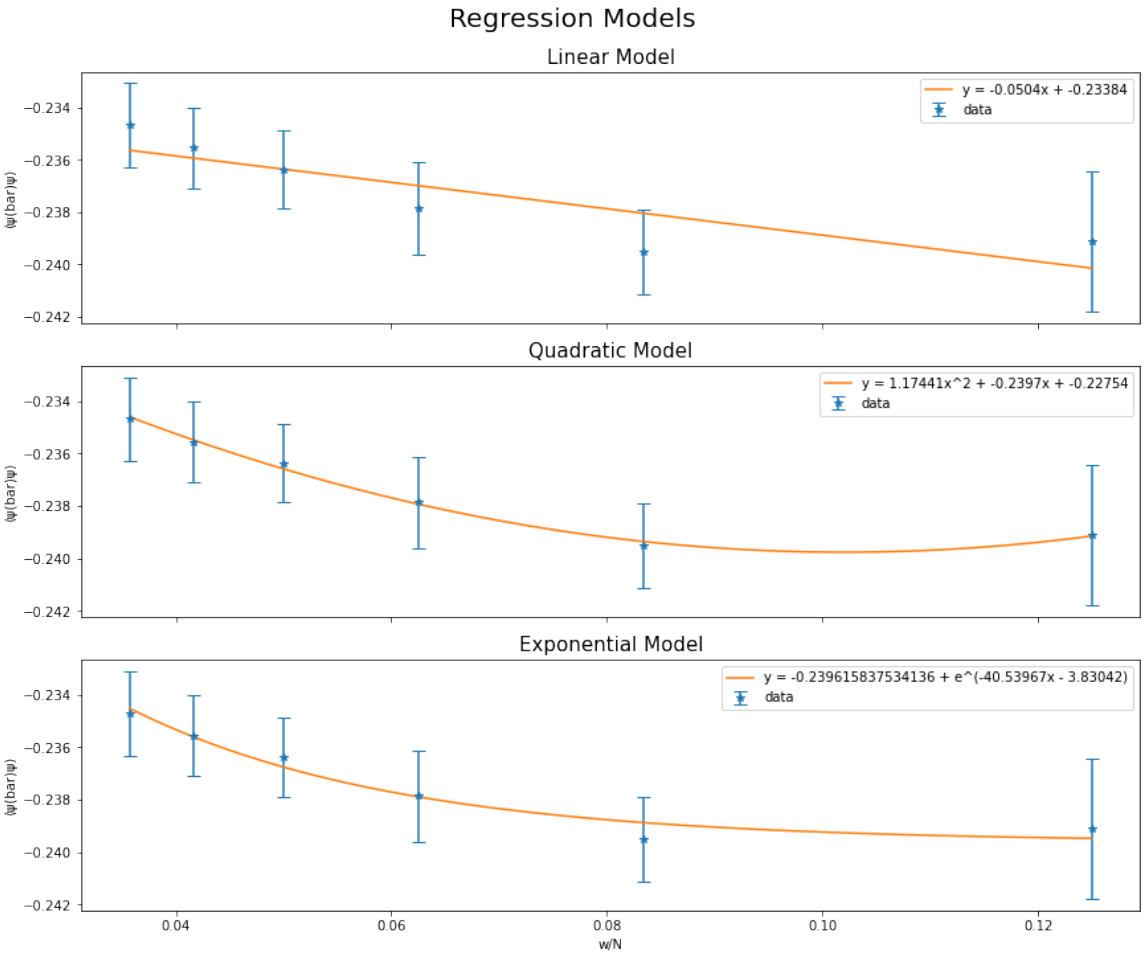


Figure 7: Regression Models

6 Conclusion

6.1 Reflection from a Scientific Standpoint

6.2 Reflection from a Personal Standpoint

Throughout this placement, I have grown and developed skills and qualities that will aid me in higher education study and my future career.

On the technical side, I have learnt how to effectively use `matplotlib` to display data in a variety of formats, this is one of the key goals I hoped to achieve by completing this placement. I have also become more familiar with `numpy` and `pandas`, which are both popular Python packages used for data handling. Familiarising myself with these packages involved reading various documentations and searching sites like Stack Overflow, which is very much used by the developer community. This problem-solving process improved my research and programming ability, which will undoubtedly help me in other areas of my life such as leading my school's computing society and making a strong application to university.

The choice to use IBM's Circuit Composer was given to me. However, I wanted to develop my abilities by coding my own circuit composer. I achieved this feat using Jupyter Notebook, a computational notebook very popular within the science community for its ability to run, execute and display code and handle markdown in the same file.

I also wanted to challenge myself by using LaTeX in the production of this report, and it has been no easy feat. LaTeX is commonly used in academia to produce scientific documents. It allows one to easily incorporate amongst other features mathematical expressions. When I started this project, I had a very vague idea of what LaTeX was. Now that I have completed this placement, I am much more confident in using LaTeX to format documents, include a bibliography, automate the contents section and add tables and images with references to them. I am glad that I have become familiar with LaTeX as I will very much need to use throughout my higher education studies. Additionally, by extension I have also become familiar with .bib bibliography documents which will also be required. However, this once again was not without its challenges. I spent a lot of time researching how to achieve specific things in LaTeX and some limitations of the typesetting system soon became apparent.

On the softer side, I have worked on qualities such as communication, time management, organisation and planning, and research. The Nuffield BeFutureReady modules have been useful in preparing me for the workplace. The key thing I learned from these modules is how to handle difficult situations, such as being late to a meeting or noticing an error shortly before a report is due. These modules have taught me how to think smart, and the mindset that I should adopt when approaching these difficult situations. I found that I have grown a lot by working with a subject specialist, I felt constantly encouraged to be more accurate in my terminology when making statements, this has helped me to become more effective

at communication.

In order to complete the work required for this placement, stay on top of my schoolwork and begin working on my application for university, I needed to manage my time well. To help me with this, I used Notion which is where I also stored my research notes and diary entries. I have a calendar listing key dates, this came in handy when planning dates and times for meetings with my supervisor.

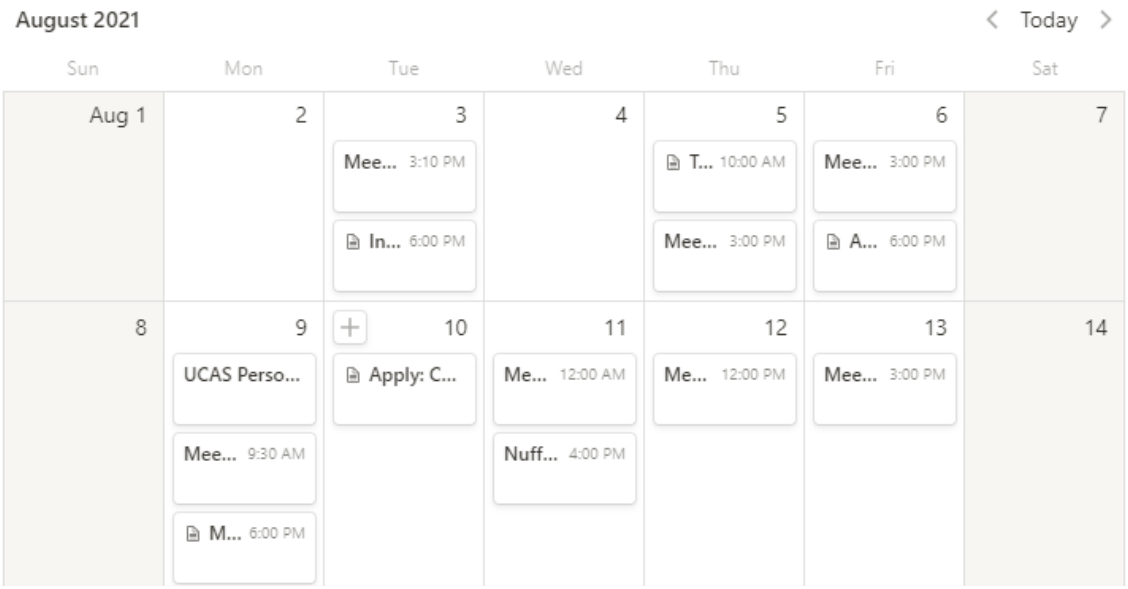


Figure 8: Notion Calendar

Throughout this placement, I have found myself constantly challenged. Aside from the technical challenges regarding the work of this project, staying motivated was difficult at times. The content of this placement has been very theoretical, covering undergraduate and even postgraduate physics topics. The fact I do not study Physics at A Level made things all the more difficult. I had to first gain fundamental understanding of particle physics, before applying newly learned information to complex topics. Although I have certainly enjoyed learning about quantum mechanics and quantum field theory, I wonder if perhaps a more practical placement in the field of computer science would have suited me better. Nevertheless, my supervisor was aware of my lack of knowledge and was happy to explain things a couple of times to reinforce key information. In retrospect, I wish I was more confident in asking questions and asked a lot more. However, I felt a little discouraged as I noticed that we were coming to the end of the placement, and I did not want to slow things down.

As hinted earlier, I made daily diary entries during my placement. However, as my

motivation began to dip, so did the regularity of my entries. On reflection, I wish was more disciplined in making regular diary entries. But one thing I have learned is that low morale and motivation can leak into other aspects of your work and have a real negative impact on the quality of your work. I found approaching the work with the mindset of wanting to learn as a useful way to combat this effect.

7 Acknowledgement

Firstly, I want to thank my supervisor, Dr Bipasha Chakraborty, for hosting me and guiding me during this placement. I want to thank her for being patient with me; I greatly appreciate her willingness to go into detail so that I can gain a good understanding, and her willingness to repeat information when she noticed that I did not fully understand.

I want to thank the staff at the Nuffield Foundation managing the South East and London placements for making this possible; I want to thank Sally Moore and Julianne Law in particular for their patience and for being diligent in answering all my and those of the other participants' queries.

Finally, I want to thank the authors of *Quantum Computing for the Quantum Curious* for producing such an amazing resource that not only tells people about quantum computation and its applications but also gets them excited about it.

A Quantum Circuit Data Collection Implementation

I will walk through the algorithm implemented to efficiently perform data collection.

To begin with, an array of the independent parameters is created so that the algorithm can easily loop through them when running the simulator. This makes the code more maintainable, as only the elements of the array need to be changed as opposed to the addition/removal of hard-coded for loops.

```
n = 10
shots_list = [100, 500, 2000, 5000, 10000, 20000]
delta_t_list = [0.001, 0.005, 0.01, 0.05, 0.1, 0.5]

states = set()
data = {}
```

`n` is the number of times the simulator (not the simulation) is run. A set called `states` has also been initiated, it will store all the different output states of the circuit. A set has been used as opposed to an array because the set data structure does not allow duplicate values. This property of sets makes subsequent implementation of the algorithm easier.

`data` is the dictionary which will store the recorded outputs of the quantum circuits; the keys are tuples of the form: ({independent variable}, state), and the value is the number of occurrences of the state in the tuple pair.

The below code block runs the algorithm for the different values of δt:

```
1  # Run the simulation n times for every value of delta_t,
2  # and end up with the final result as a pandas dataframe of the mean values
3  for val in delta_t_list:
4      temp = []
5      for i in range(n):
6          angle = w_tilde1*val
7          circuit = makeFullCircuit(angle)
8          simulator = QasmSimulator()
9          compiled_circuit = transpile(circuit, simulator)
10         # shots is the no. of times the circuit is run
11         job = simulator.run(compiled_circuit, shots=shots_list[-1])
12         result = job.result()
13         temp.append((val, result.get_counts(circuit)))
14     for j in range(len(temp)):
15         for k, v in temp[j][1].items():
16             data[(val,k)] = data[(val, k)]+v if ((val, k)) in data.keys() else v
17             states.add(k)
18 for j in range(len(temp)):
19         for key in temp[j][1].keys():
20             states.add(key)
```

There is a for loop at line 3 which loops through the values of δt. There is a nested for loop at line 5 which runs the simulator ten times. The list `temp` temporarily stores the results of the simulations for a given value of δt. The second for loop on line 14 then adds the results to the `data` dictionary. Finally, line 18 loops through all the states observed and includes them in the set `states`.

The below code block transforms the data into a suitable form for a `pandas DataFrame` which will allow the data to be displayed in a table:

```
1  states = sorted(list(states))
2  transformed_data_delta_t = {val: [0]*len(states) for val in delta_t_list}
3  for i in range(len(states)):
4      for k, v in data.items():
5          val = k[0]
6          state = k[1]
7          # divide by n*2000 to find the ratio of mean no. occurences of
8          # statex to total no. of shots and then multiply by 100 to find probability
```

```
        # of observing state x for a given delta t as a percentage
        num_occurs =(v*100)/(n*shots)
        transformed_data_delta_t[val][states.index(state)] = num_occurs

transformed_data_delta_t = pd.DataFrame(transformed_data_delta_t, index=(states))
transformed_data_delta_t.index.name = 'State'
```

The same algorithm is then run, this time with the outer for loop looping through the values of `shots_list` rather than `delta_t_list`. For the full code, see my GitHub repository [2].

References

[1] B. Chakraborty, M. Honda, T. Izubuchi, Y. Kikuchi, and A. Tomiya, *Digital quantum simulation of the schwinger model with topological term via adiabatic state preparation*, 2020.

[2] A. Einstein, "On the Electrodynamics of Moving Bodies", Annalen Phys. **17**, 891–921 (1905).

[3] M. Kuhlmann, "Quantum Field Theory", in *The Stanford encyclopedia of philosophy*, edited by E. N. Zalta, Fall 2020 (Metaphysics Research Lab, Stanford University, 2020).

[2] https://github.com/smalih/Quantum-Computation-for-Particle-Physics