

# Adiabatic Simulation with Gate Model

August 4, 2023

## 1 Ground state search and time evolution in Gate Model Quantum computer

### References

- Nielsen, M., & Chuang, I. (2010). Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511976667
- Sakurai, J., & Napolitano, J. (2020). Modern Quantum Mechanics (3rd ed.). Cambridge: Cambridge University Press. doi:10.1017/9781108587280
- Cerezo, M., Arrasmith, A., Babbush, R. et al. Variational quantum algorithms. Nat Rev Phys 3, 625–644 (2021). <https://doi.org/10.1038/s42254-021-00348-9>
- Andrew M. Childs, Yuan Su, Minh C. Tran, Nathan Wiebe, and Shuchen Zhu, Theory of Trotter Error with Commutator Scaling, Phys. Rev. X 11, 011020 – Published 1 February 2021
- McArdle, S., Jones, T., Endo, S. et al. Variational ansatz-based quantum simulation of imaginary time evolution. npj Quantum Inf 5, 75 (2019). <https://doi.org/10.1038/s41534-019-0187-2>

### 1.1 Why the ground state of the system is important?

The ground state of a quantum system is essentially a stationary state of the system where energy resides at its lowest level. This fundamental characteristic holds profound importance in both physics and optimization.

In Physics:

Understanding the ground state serves as a cornerstone in analyzing the properties of a quantum system. Quantum critical phenomena reveal themselves at certain levels of observation, but the effects intrinsic to the quantum world are most prominently displayed near the ground state. Additionally, the excited states of the system, characterized by higher energy levels, can be described by examining the ground state and transformations from it. These stationary transformations provide a robust framework to understand the system's behavior at various energy levels.

Practically, in many iterative solve algorithm or simulation, the ground state generally be a good starting point of them. Moreover, for optimizers, the ground state is a solution what they ultimately want to get from the system.

Shortly, the importances of the ground state are

for physicsts,

- It represent the systems's behavior in low energy state in quantum world.
- Dynamics of the system is based on the stable state of the system.
- Using Adiabatic theorem and process, we can make transition to another hamiltonian's ground state.

for optimizers,

- It is a solution itself.
- Knowing existence and specific value of minum state gurantees lots of conviencen in calculation.

Only few examples in quantum mechanics can be solved exactly. The remained cases we are forced to use approximation techniques.

Diagonalization of the Hamiltonian means analyzing the structure of the Hamiltonian in the language of energy determined by the given state.

## 1.2 Quantum simulation problem

Time evolution of the system is

$$|\psi; t_0\rangle \rightarrow |\psi; t_1\rangle$$

Assuming we are living in Heisenberg picture.

$$\psi(t) = U(t)\psi$$

and applying Schrodinger equation of time-dependent form

$$i\hbar \frac{d}{dt}|\psi\rangle = H|\psi\rangle$$

Thus, we get

$$i\hbar \frac{d}{dt}U(t)|\psi\rangle = H|\psi\rangle$$

$$i\hbar \frac{d}{dt}U(t) = H$$

General from of time-evolution unitary operator is

$$U(t, 0) = \exp(-i \int_0^t H(t)dt)$$

There are 3 cases about the system Hamiltonian,

1. Hamiltonian of the system is time independent.

$$U(t, t_0) = \exp(-iH/\hbar (t - t_0))$$

2. It is time dependent but  $H$  at different times commute.

$$U(t, t_0) = \exp \left[ -\frac{i}{\hbar} \int_{t_0}^t H(t') dt' \right]$$

3. Time dependent but not commute at different times.

$$U(t, t_0) = 1 + \sum_{n=1}^{\infty} \int_{t_0}^t dt_1 \int_{t_0}^{t_1} dt_2 \cdots \int_{t_0}^{t_{n-1}} dt_n \Pi_{i=1}^n H(t_i)$$

The last case expression is known as **Dyson Series**.

With given hamiltonian  $H$  and evolution time  $t$ , find a unitary transformation  $U$  such that

$$\|U - e^{-itH}\| < \epsilon$$

The time evolution simulation can be used with different purpose. However, in this document we restrict the purpose to obtain a ground state of the system.

### 1.3 Single spin system with time evolution

Fundamental principal of the physics theory is even we have several different systems if their hamiltonians have same form each other, the system will show same behaviors. The quantum computers is basically,  $N$  number of 1/2 spins system. There are many examples of spin system. For examples,

- Spin of a single electron localized on an impurity in a semiconductor
- Combined spin of several  $d$  electrons in a transition-metal-ion.
- Nuclear spin of an atom in a crystal
- Combined spin and orbital moment of a rare-earth ion.

With  $N$  number of qubits and operators we can simulate those behaviors on our quantum computer. Shortly, Let us simulate simple single spin state with quantum circuit.

This example is referred from “Spin Precession” section of “Modern Quantum Mechanics”, Sakurai and Napolitano.

Suppose we have spin 1/2 system with magnetic moment  $e\hbar/(2m_e c)$  subjected to an external magnetic field  $\mathbf{B}$ . Then, the Hamiltonian of the system is

$$H = -\left(\frac{e\hbar}{2m_e c}\right) \mathbf{S} \cdot \mathbf{B}$$

where,  $\mathbf{S}$  is a total spin of the system. Let the system is restricted to  $z$  directional uniform magnetic field.

- Hamiltonian :  $-\frac{e\hbar B}{2m_e c} S_z$
- Eigenvalues:  $E_{\pm} = \mp \frac{e\hbar B}{2m_e c}$  for each eigen states,  $\pm S_z$ .

Let  $\omega = \frac{|e|B}{m_e c}$ ,  $H = \omega S_z$ , consequently  $U(t) = \exp(i(\omega S_z/\hbar)t)$ . If the initial state of the system is  $S_x+$ , we can calculate the probability of each states,  $S_x+$ ,  $S_x-$  as  $\cos^2(\omega t/2)$ , and  $\sin^2(\omega t/2)$  respectively.

$$\langle S_x \rangle = (\frac{\hbar}{2}) \cos(\omega t)$$

$U(t, 0) = \exp(-i\frac{\omega S_z}{\hbar}t) = \exp(-i\frac{\theta_t}{2}\sigma_z)$ . Let Plank constant  $\hbar = 1$  for convenience in calculation. We have

$$\theta(t) = \omega t$$

```
[ ]: # Qubit simulation
import pennylane as qml
from typing import Literal

from math import pi, cos, sin

# System configureration
B = 3E2
me = 9.1093837015E-31
e = 1.602176634E-19
c0 = 299792458
hbar = 6.62607015E-34/(2*pi)

wb = (e*B)/(me*c0)
theta = lambda t: wb*t

[ ]: shot_n = int(1E4)
dev = qml.device("default.qubit", wires=1, shots=shot_n)

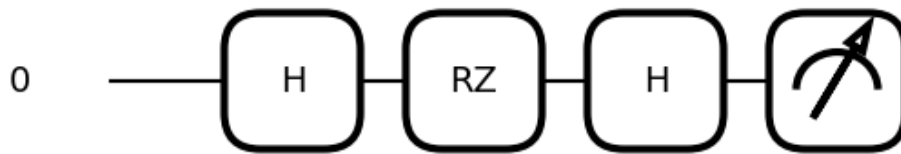
@qml.qnode(dev)
def circuit(t, measure_state:Literal["z", "x", "y"]="z", mode:Literal["sample", "exp"]="exp")="sample"):
    qml.Hadamard(wires=0)
    qml.RZ(theta(t), wires=0)

    if measure_state != "z":
        qml.Hadamard(wires=0)
        if measure_state == "y":
            qml.adjoint(qml.S(wires=0))
    if mode == "sample":
        return qml.counts()
    else:
        return qml.expval(qml.PauliZ(0))

[ ]: fig, ax = qml.draw_mpl(circuit)(0.3, "X")
fig.show()
```

C:\Users\qwqwh\AppData\Local\Temp\ipykernel\_15020\1421198088.py:2: UserWarning: Matplotlib is currently using module://matplotlib\_inline.backend\_inline, which is a non-GUI backend, so cannot show the figure.

```
fig.show()
```



```
[ ]: t = 0.05
      samples = circuit(t, "x")

      print(f"\nState probability approximation \n with {shot_n} sampling")
      print("\tSx+:", samples["0"]/shot_n, "Sx-:", samples["1"]/shot_n)
      print("Precise probability")
      print("\tSx+:", cos(wb*t/2)**2, "Sx-:", sin(wb*t/2)**2)
```

State probability approximation

with 10000 sampling

Sx+: 0.0814 Sx-: 0.9186

Precise probability

Sx+: 0.08371358969134887 Sx-: 0.9162864103086511

```
[ ]: circuit(t, "x", mode="exp")
```

```
[ ]: array(-0.839)
```

```
[ ]: cos(wb*t)
```

```
[ ]: -0.8325728206173023
```

## 1.4 Many body system

In the above example, we haven't introduced anything new; we simply described one spin system with another spin system. However, what if the system consists of many bodies with a complicated structure? We then, are forced to use approximation methods.

In this document, we will focus on finding ground state of the system, especially, with the time evolution process.

Procedure of quantum simulation.

1. Formulating the system hamiltonian with spin operators.
  1. Usually, they are expressed in system's annihilation and creation operators.
  2. Find a mapping to spin operator. Popular transformations are *Jordan Wigner transformation* and *Bravyi-Kitaev transformation*.
2. Decomposing the hamiltonian into several unitary operations with time steps. You can use *Trotter-Szuki decomposition* of  $p$  level.
3. Prepare the measurement sometimes you need to apply QFT to obtain momentum space value from position system.
4. Measurement.

## 1.5 Adiabatic Theorem and Anealing

Adiabatic theorem states that if the system hamiltonian is varying *slowly*, the final state of the system is remained in eigenstate of final hamiltonian which is *corresponding* eigenstate of the initial hamiltonian eigenstate.

For example, if we bring a simple pendulum at ground and slowly climb mountain, without any external manipulation, the pendulum automatically match the period with respect to the reduced gravity field. The term *slowly* is very important, it depends on the initial and final hamiltonian of the system. If the evolution time is  $T$  and the eigenvalues of initial and final hamiltonian are  $E_i$  and  $E_f$ , the evolution time must be satisfies next, see Sakurai et al,

$$\frac{2\pi}{|E_f - E_i|} \ll T$$

$$H_{qc}(t(s), T) = T(1 - f(s))H_{initial} + Tf(s)H_{solve}$$

where,  $H_{qc}$  is a hamiltonian being applied to the circuit,  $H_{initial}$  is a initial state hamiltonian, and  $H_{solve}$  is a problem hamiltonian we want to solve.  $f(s)$  is a monotonic increasing and bounded function  $0 \leq f(s) \leq 1$

What if the final solution is restricted to the combination of  $z$  axis?

In this case, Ising model is used to describe the hamiltonian.

$$H_{solve} = \sum_i q_i \sigma_i^z + \sum_{i>j} h_{ij} \sigma_i^z \sigma_j^z$$

The proper initial state would have uniform probabilities for all state to the measurement axis. We can set the initial hamiltonian whose ground state is a combination of Hadamard basis.

$$H_{initial} = -\sum_i \sigma_i^x$$

The common annealing solutions, such as D-Wave and QuEra, also apply the adiabatic theorem to their initialization process and Hamiltonian form. Quantum annealing is fundamentally based on the phenomenon of quantum fluctuation; however, the starting and final stages, as well as the evolving process, follow the adiabatic theorem. That is why they offer some annealing time and initial state parameters to the user API. Some problems require much more time to achieve the appropriate adiabatic process

See [D-Wave's annealing document](#)

## 1.6 Simulating Adiabatic process with Gate model

Now we know that the gate model quantum computer is one type of **universal** quantum computing model. However, the universality does not mean a convenience model for all routine and solution. In annealing processor, we can find a good approximation of the ground state of the system. However, if we want to implement the process in gate model system, because of restriction of the current devices, noise, there are many restriction and block walls.

Basically, simulating time evolution with gate model is an iterative solve process of differential equation, like Euler method, in a speech of figure.

### 1.6.1 Product Formulation

See details in Andrew et al

It is commonly known as “Trotter-Suzuki Decomposition” of the given Hamiltonian

$$\exp(i \sum H_i t) \approx (\Pi_i \exp(i H_i t / N) * \Pi_i \exp(i H_{n-i} t / N))^N$$

The benefit of the product formulation is ancilla free time evolution method. It means we do not need any ancilla registers to conduct time evolution operation to the state.

### Error of the Trotter decomposition Baker Campbell Hausdorff formula

For given evolution time  $t$  and two hamiltonian  $H = H_1 + H_2$ ,

$$\exp(itH) = \exp(it(H_1 + H_2) + BCH(H, t))$$

where,  $BCH(H)$  is a Baker-Campbell-Hausdorff formula

$$-\frac{t^2}{2}[H_2, H_1] + i\frac{t^3}{12}([H_2, [H_2, H_1]] - [H_1, [H_2, H_1]]) + \dots$$

### 1.6.2 Time dependence of Adiabatic process

Adiabatic process is induced by time-varying hamiltonian of the system. It means that we need to imitate the time varying hamiltonian evolution on circuit. We can approximate this process with generating each step hamiltonian values from hamiltonian generator,  $H_t = H_{qc}(t_i)$ .

Thus, approximated process of adiabatic process in gate model is divided into several time-evolution operations of different hamiltonian,  $H_t$ .

For the given hamiltonians and time-evolution value,  $H_i, H_{solve}, T$ , the adiabatic hamitonian change is defined as next form

$$H_{qc}(t) = (1 - t/T)\mu H_i + (t/T)H_{solve}$$

where,  $\mu$  is a weight coefficient of initial state.

The

[Adiabatic Processs Approximation]

```
H_i <- initial hamiltonian
mu <- weight of the H_i
H_f <- problem hamiltonian
-----
func_evolve(t, H, tn):
    Applying time evolution operator on state
        evolution time: t,
        hamiltonian: H
        trroter decompose level: tn.

func_Hqc(s, t, mu): (1-s/t) mu * H_i + (s/t) H_f
    Return instaneous hamiltonian of the given time step: s.
-----
[Defined Adabatic schedule parameters]

T <- # total evolution time
steps <- # time difference step number
dt <- T/steps

trotter_num <- # trotter decompose level

[Prepare the circuit]

initiate the system with Hadamard basis |+>

while i < step
    t = dt*i
    H_t = func_Hqc(t, T, mu)
    -----
    func_evolve(dt, H_t, trotter_num)
```

Sampling, Measure Expectation Value ... et cetra

See below code, this example is about finding a non-trivial solution of point distribution whose radiation to be flat radiation on another surface.



```
[ ]: import numpy as np
import pennylane as qml
from typing import Literal
from tem import *

kernel = lambda i,j : 1/(1+(i -j)**2)
n0 = 7
K = np.fromfunction(kernel, (n0, n0))
K
```

```
[ ]: array([[1.          , 0.5          , 0.2          , 0.1          , 0.05882353,
0.03846154, 0.02702703],
[0.5          , 1.          , 0.5          , 0.2          , 0.1          ,
0.05882353, 0.03846154],
[0.2          , 0.5          , 1.          , 0.5          , 0.2          ,
0.1          , 0.05882353],
[0.1          , 0.2          , 0.5          , 1.          , 0.5          ,
0.2          , 0.1          ],
[0.05882353, 0.1          , 0.2          , 0.5          , 1.          ,
0.5          , 0.2          ],
[0.03846154, 0.05882353, 0.1          , 0.2          , 0.5          ,
1.          , 0.5          ],
[0.02702703, 0.03846154, 0.05882353, 0.1          , 0.2          ,
0.5          , 1.          ]])
```

- Radiation on given surface =  $r$
- Propagation matrix =  $K$
- Source distribution: Binary vector  $x$

$$Kx = r$$

We want to obtain a flat radiation to some area far from the source plane, so let us apply difference matrix to both term. If flatness is achieved, then the right term would be zero.

$$DKx = Dr = 0$$

This problem can be converted to Quadratic form and forms a minimization problem.

$$\min x^t((K^t D^t)DK)x$$

```
[ ]: D = diff_matrix_three_point(n0, t="") # difference matrix
```

First, we try D-wave anealer.

```
[ ]: from dwave.system import DWaveSampler, EmbeddingComposite
sampler = EmbeddingComposite(DWaveSampler())
def qubo_index(i, N, zero=True):
```

```

max = N*(N-1)/2
if i > max or i<1:
    return None

for j in range(1, N+1):
    Nj = N-j
    if i > Nj:
        i -= Nj
    else:
        if zero:
            return (j-1, j+i-1)
        return (j, j+i)

```

```

[ ]: L = D@K
M = L.T@L
n0 = M.shape[0]
print(M[0,0])
save_M00 = M[0,0]
i, j = np.indices(M.shape)

# Adjust the principal axis effect
# to reduce possiblity of trivial solution, all zero and ones.
M[i==j] = 1.2

```

2.2649231622912462

```

[ ]: q_r_diff = M.diagonal()
J_r_diff = M[np.triu_indices(n0, k = 1)]

linear_r = { (i, i): q_r_diff[i] for i in range(0,n0)}
in_list_r= [qubo_index(i, n0) for i in range(1, int(n0*(n0-1)/2)+1)]
quadratic_r = { in_list_r[i]: M[*in_list_r[i]] for i in range(0, int(n0*(n0-1)/
↪2))}

Quad = {**linear_r, **quadratic_r}

```

```

[ ]: samples_sym_r = sampler.sample_qubo(
    Quad,
    num_reads=1000,
    annealing_time= 2)

```

```

[ ]: print(samples_sym_r.record[0:5])

[([1, 1, 0, 1, 0, 1, 1], -1.06076492, 702, 0.)
 ([1, 1, 0, 0, 0, 1, 1], -0.09977678, 53, 0.)
 ([0, 0, 0, 0, 0, 0, 0], 0., 106, 0.)
 ([1, 1, 0, 1, 1, 1, 1], 0.12485558, 13, 0.)
 ([1, 1, 1, 1, 0, 1, 1], 0.12485558, 12, 0.)]

```

There is a trivial solution, all zeros, but we found some non-trivial solution of the given system.

Convert the hamiltonian to qubit gate hamiltonian

### 1.6.3 Conversion to Adabatic time evolution Hamiltonian on PennyLane

```
[ ]: import matplotlib.pyplot as plt
from pennylane.templates import ApproxTimeEvolution

[ ]: # Circuit setting
n_wires = n0
wires = range(n_wires)

basis_hamiltonian_initial = [qml.PauliX(i) for i in range(0, n0)]

basis_hamiltonian_dig = [qml.PauliZ(i) for i in range(0, n0)]
basis_hamiltonian_off = [qml.PauliZ(i)@qml.PauliZ(j) for (i, j) in in_list_r]

# Adabatic parameter setting
#mu = 10 # Initial hamitlonian weight
#coeff_X = n0*[-mu] # pauli x coeff

# Below solution hamiltonian

#M[i==j] = save_M00 # restore the original quadratic form
M[i==j] = 0
dig_coeff = list(np.sqrt(M.diagonal()))
off_coeff = [M[i, j] for (i, j) in in_list_r]

coeff = np.array(coeff_X + dig_coeff + off_coeff)
basis_hamiltonian = basis_hamiltonian_initial + basis_hamiltonian_dig +
    ↪basis_hamiltonian_off

[ ]: H_init = qml.Hamiltonian(-np.ones(n0), basis_hamiltonian_initial)
H_solve = qml.Hamiltonian(dig_coeff+off_coeff ,basis_hamiltonian_dig +
    ↪basis_hamiltonian_off)

[ ]: # Device
dev = qml.device('default.qubit', wires=n_wires, shots=100)

[ ]: # Util circuit
@qml.qnode(dev)
def flip_circuit(flip=n0*[0], operation="x", mode="e", H= H_solve):
    if len(flip) != n0:
        raise ValueError(f"The flip list must be length {n0} list.")

    for i, f in enumerate(flip):
        if f ==1:
```

```

        if operation == "x":
            qml.PauliX(wires=i)
        elif operation == "h":
            qml.Hadamard(wires=i)

    if mode == "c":
        return qml.counts()
    elif mode == "e":
        return qml.expval(H)
    elif mode == "s":
        return qml.state()

```

```

[ ]: # Time evolution circuit
@qml.qnode(dev)
def circuit_time_evolve(time, initial_weight, steps=5, trotter_num=2,
    ↪, mode="exp", Hamiltonian=H_solve):
    n_list = [i for i in range(0, n0)]
    for i in n_list:
        qml.Hadamard(wires=i)

    #qml.Hadamard(wires=n_list)

    dt = time/steps
    n1 = int(n0*(n0+1)/2)

    if initial_weight < 0:
        initial_weight = -initial_weight
    coeff_X = n0*[-initial_weight]
    coeff = np.array(coeff_X + dig_coeff + off_coeff)

    for i in range(0, steps+1):
        t = i*dt/time
        t_evol = np.array(n0*[1-t] + n1*[t])

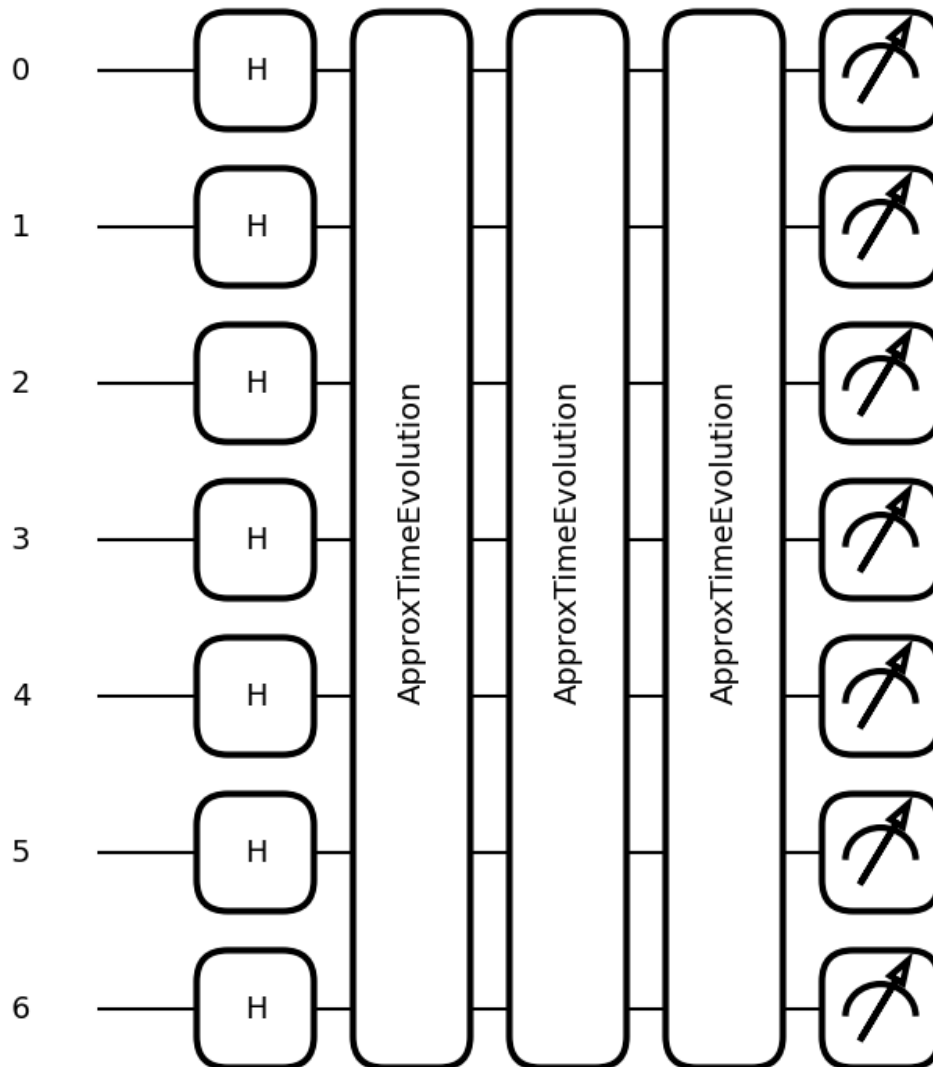
        t_coeff = t_evol * coeff
        #print("step: ", t, 20*"=")
        #print("evolve", t_evol)
        #print("coeff:\n", t_coeff)
        #print("origi:", coeff)
        H_t = qml.Hamiltonian(t_coeff, basis_hamiltonian)

        ApproxTimeEvolution(H_t, dt, trotter_num)
    if mode=="exp":
        return qml.expval(Hamiltonian)
    elif mode=="count":
        return qml.counts()

```

```
[ ]: # Example circuit
fig, ax = qml.draw_mpl(circuit_time_evolve)(1, initial_weight = 1, steps=2)
fig.show()
```

/var/folders/wv/cwvgxcp14lxcs5jn0r6p87sr0000gn/T/ipykernel\_70300/722527748.py:3:  
 UserWarning: Matplotlib is currently using  
 module://matplotlib\_inline.backend\_inline, which is a non-GUI backend, so cannot  
 show the figure.  
 fig.show()



```
[ ]: samples_sym_r.record[0:2]
```

```
[ ]: rec.array([[1, 1, 0, 1, 0, 1, 1], -1.06076492, 702, 0.),  
              ([1, 1, 0, 0, 0, 1, 1], -0.09977678, 53, 0.)],  
              dtype=[('sample', 'i1', (7,)), ('energy', '<f8'), ('num_occurrences',  
              '<i8'), ('chain_break_fraction', '<f8')])
```

```
[ ]: exp_d_wave0 = float(flip_circuit(samples_sym_r.record[0][0]))  
exp_d_wave1 = float(flip_circuit(samples_sym_r.record[1][0]))  
print("D-wave solution expectation value")  
print(samples_sym_r.record[0][0])  
print(exp_d_wave0)  
print(samples_sym_r.record[1][0])  
print(exp_d_wave1)
```

```
D-wave solution expectation value  
[1 1 0 1 0 1 1]  
-7.427300036466925  
[1 1 0 0 0 1 1]  
-0.31944140860829595
```

```
[ ]: flip_circuit(n0*[1], operation="h", mode="e", H=H_init)
```

```
[ ]: array(-7.)
```

```
[ ]: flip_circuit(samples_sym_r.record[0][0], mode="e", H=H_init)
```

```
[ ]: array(-0.08)
```

```
[ ]: # Define adaibatic schedule, Time difference, Totter decompositon level  
  
T = 6 # total time evolution  
iw= 5 #initial hamiltonian weight  
steps = 40 # time steps  
trroter_num =4 # trroter decomposition of each steps
```

```
[ ]: # Measure the problem Hamiltonian expectation value  
circuit_time_evolve(T,  
    initial_weight=iw,  
    steps=steps,  
    trroter_num = trroter_num ,  
    mode="exp",  
    Hamiltonian = H_solve)
```

```
[ ]: array(-6.77453147)
```

Expectation value is well optimized but it takes too much times.

Let's adjust some paprameters.

```
[ ]: T = 8
      iw= 5
      steps = 40
      trroter_num =2
      circuit_time_evolve(T,
        initial_weight=iw,
        steps=steps,
        trroter_num = trroter_num ,
        mode="exp",
        Hamiltonian = H_solve)
```

```
[ ]: array(-6.87789256)
```

We reduced time less than half of the previous evolution process. The trroter depth provide us good appoximation as increasing however, time complexity increases rapidly, too.

```
[ ]: T = 7
      iw= 1
      steps = 30
      trroter_num =2
```

```
[ ]: circuit_time_evolve(T,
      initial_weight=iw,
      steps=steps,
      trroter_num = trroter_num ,
      mode="exp",
      Hamiltonian = H_solve)
```

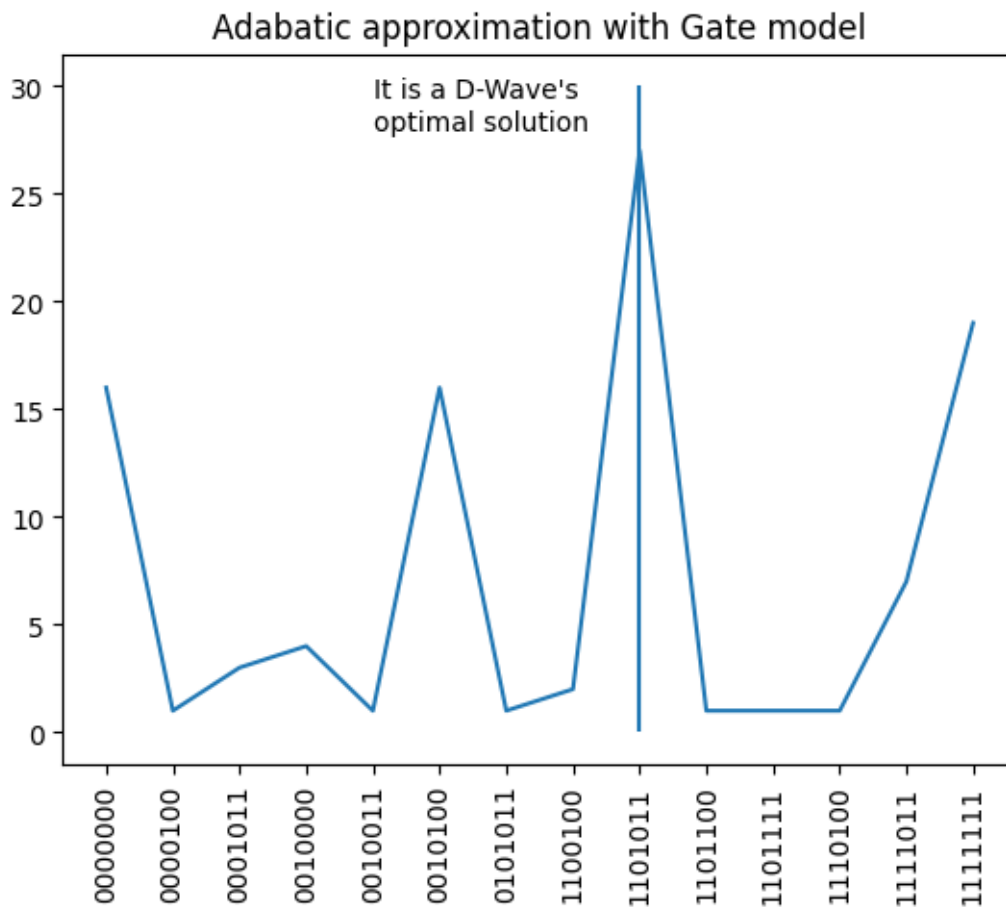
```
[ ]: array(-7.24496947)
```

```
[ ]: sample = circuit_time_evolve(T,
      initial_weight=iw,
      steps=steps,
      trroter_num = trroter_num ,
      mode="count")
      sample
```

```
[ ]: {'0000000': 16,
      '0000100': 1,
      '0001011': 3,
      '0010000': 4,
      '0010011': 1,
      '0010100': 16,
      '0101011': 1,
      '1100100': 2,
      '1101011': 27,
      '1101100': 1,
      '1101111': 1,
```

```
'1110100': 1,
'1111011': 7,
'1111111': 19}
```

```
[ ]: plt.plot(sample.keys(), sample.values())
plt.vlines(8, 0, 30)
plt.xticks(rotation = 90)
plt.title("Adabatic approximation with Gate model")
plt.text(4, 28, "It is a D-Wave's \noptimal solution")
plt.show()
```



See below D-wave solution. Their adabatic scheduling is well defined for solve. The above approximation wasn't good enough comparing to actual anealing system. However, we obtained a good approximation.

```
[ ]: samples_sym_r.record[0:10]
```



```
[ ]: rec.array([( [1, 1, 0, 1, 0, 1, 1], -1.06076492, 798, 0.),
                ([1, 1, 0, 0, 0, 1, 1], -0.09977678, 59, 0.),
                ([0, 0, 0, 0, 0, 0, 0], 0., 42, 0.),
                ([1, 1, 0, 1, 1, 1, 1], 0.12485558, 14, 0.),
                ([1, 1, 1, 1, 0, 1, 1], 0.12485558, 16, 0.),
                ([0, 0, 0, 0, 0, 1, 1], 0.38739265, 9, 0.),
                ([1, 1, 0, 0, 0, 0, 0], 0.38739265, 10, 0.),
                ([1, 1, 1, 0, 0, 1, 1], 0.38937313, 9, 0.),
                ([1, 1, 0, 0, 1, 1, 1], 0.38937313, 5, 0.),
                ([0, 0, 0, 1, 0, 1, 1], 0.50689858, 3, 0.)],
          dtype=[('sample', 'i1', (7,)), ('energy', '<f8'), ('num_occurrences',
'<i8'), ('chain_break_fraction', '<f8')])
```

If the setting is well defined then, the increased time evolution must show same optimal points. Change the time evolution value `T` and difference step number `steps` and check the samples.

```
[ ]: # Remain these values as constant
iw= 1
trroter_num =2
```

```
[ ]: T =100
steps = 400

sample = circuit_time_evolve(T,
    initial_weight=iw,
    steps=steps,
    trroter_num = trroter_num ,
    mode="count")
sample
```

```
[ ]: {'0000000': 37, '0010100': 12, '1101011': 12, '1111111': 39}
```

## 1.7 VQA approach

Considering the first definition of the *quantum simulation*. We want to find an unitary transformation corresponding the given Hamiltonian  $H$  such that,  $\|U - \exp(-iHt)\| < \epsilon$  with a given error  $\epsilon$ .

The time evolution simulation is a good example of a physics law-based method, and in several cases, it can generate good results. The product model frequently used in time-evolution implementation has some advantages, as it preserves the locality of the system, is practical in commutativity systems, and does not require ancilla.

However, the required depth of the circuit for accurate results becomes huge as the size of the system increases. Due to the current device's operational errors, minimizing the circuit's depth is a significant consideration, in general situation.

One good point is measurement *expectation* value of the Hamiltonian represented with spin operators requiring much less gates than time-evolution circuit. Whatever process the system has gone through, it is sufficient if only the expected value is minimized.

- Ansatz for general state of  $N$  qubit system.
- Spin operator hamiltonian being of concern.
- Classic optimization algorithm.

See McArdle et al as an example and comprehensively Cerezo et al. The time-evolution hamiltonian circuit structure is work as a type of ansatz especially in quantum chemistry, simulation and optimization problem.

[ ]: *# Comparing time-evolution and VQE approach*

## 1.8 Further References

- Nishi, H., Kosugi, T. & Matsushita, Yi. Implementation of quantum imaginary-time evolution method on NISQ devices by introducing nonlocal approximation. npj Quantum Inf 7, 85 (2021). <https://doi.org/10.1038/s41534-021-00409-y>

## 1.9 Quantum Walk

Quantum search algorithm is an example of quantum walk technique in circuit model. See chapter 6.2 of Nilsen and Chuang