Assignment

Aalto University - PhD Position on
"Physics Driven Machine Learning and Quantum Computing"

I. ORNSTEIN–UHLENBECK PROCESS

Aim : To study the given scalar stochastic differential equation, finding its solutions and plotting its trajectories.

$$\frac{dx(t)}{dt} = -\lambda x(t) + \omega(t),\ x(0) = x_0$$

A. Complete Solution of SDE

The above Linear Non-homogeneous Stochastic differential equation can be solved by multiplying an Integrating Factor (I.F.) on both the sides of the equation[1]. We'll take the I.F. as $e^{\lambda t}$ so that we can modify the equation to our benefit.

$$e^{\lambda t}\frac{dx(t)}{dt} + e^{\lambda t}\lambda x(t) = e^{\lambda t}\omega(t)$$

The LHS can be seen as a differential of product of two functions. So the equation can be simplified as :

$$d(e^{\lambda t}x(t)) = e^{\lambda t}\omega(t)dt$$

$$e^{\lambda t}x(t) - x_0 = \int_0^t e^{\lambda\tau}\omega(\tau)\,d\tau$$

$$\boxed{x(t) = e^{-\lambda t}x_0 + \int_0^t e^{\lambda(\tau-t)}\omega(\tau)\,d\tau}$$

The mean $m(t)$ can be found by taking the expectation of the above equation. White noise $\omega(t)$ is described as a Gaussian process with zero mean [1], therefore the second term in x(t) vanishes when we take expectation.

$$\boxed{E[x(t)] = e^{-\lambda t}x_0 = m(t)}$$

The variance $P(t)$ is described by the following relation [1] :

$$P(t) = E[(x(t) - m(t))(x(t) - m(t))^T]$$

$$= E[(e^{-\lambda t}(x_0 - x_0) + \int_0^t e^{\lambda(\tau-t)}\omega(\tau)\,d\tau)((x_0 - x_0)^T(e^{-\lambda t})^T + \int_0^t \omega^T(\tau)(e^{\lambda(\tau-t)})^T\,d\tau)]$$

$$= E[(\int_0^t e^{\lambda(\tau-t)}\omega(\tau)\omega^T(\tau)(e^{\lambda(\tau-t)})^T\,d\tau]$$

$$= \int_0^t qe^{2\lambda(\tau-t)}\,d\tau$$

We simplified the equation by using the Dirac delta correlation of white noise $E[\omega(t)\omega^T(s)] = \delta(t-s)q$ where q is the spectral density.

On integrating P(t) we get :

$$\boxed{P(t) = \frac{q}{2\lambda}(1 - e^{-2\lambda t})}$$

B. Calculating Limits of Mean and Variance

From our previous calculations, we know that
$m(t) = e^{-\lambda t}x_0$
At t $\to \infty$, $e^{-\lambda t}$ will be very small and can be approximated to zero

$$\boxed{\lim_{t\to\infty} m(t) = 0}$$

By the same logic let us now find out P(t) at $t \to \infty$.
$P(t) = \frac{q}{2\lambda}(1 - e^{-2\lambda t})$

$$\boxed{\lim_{t\to\infty} P(t) = \frac{q}{2\lambda}}$$

Let us now find the mean and variance by solving the stationary state of the differential equations
$\frac{dm}{dt} = 0$ and $\frac{dP}{dt} = 0$

$$m(t) = e^{-\lambda t}x_0$$

$$\frac{dm}{dt} = -\lambda e^{-\lambda t}x_0$$

$$\frac{dm}{dt} = -\lambda m(t) = 0$$

gives m(t) = 0. Similarly,

$$\frac{dP}{dt} = 2\lambda(\frac{q}{2\lambda}e^{-2\lambda t}) = 0$$

Adding and subtracting terms to get the equation in terms of P(t)

$$\frac{dP}{dt} = \frac{q}{2\lambda}e^{-2\lambda t} - \frac{q}{2\lambda} + \frac{q}{2\lambda} = 0$$

$$\frac{dP}{dt} = \frac{q}{2\lambda}(e^{-2\lambda t} - 1) + \frac{q}{2\lambda} = 0$$

$$\frac{dP}{dt} = \frac{q}{2\lambda}(1 - e^{-2\lambda t}) = \frac{q}{2\lambda}$$

gives $P(t) = \frac{q}{2\lambda}$

C. Trajectories

To simulate trajectories we will use the Euler-Maruyama Method which is a modified version of Euler method applied on our Stochastic ODE. [1][2]

$$\Delta x = -\lambda x(t)\Delta t + \omega(t)\Delta t$$

$$x_{k+1} = x_k - \lambda x_k \Delta t + \omega(t)\Delta t$$

$\omega(t)\Delta t$ can be replaced by a Gaussian random variable with distribution N(0, $q\Delta t$) where q is the spectral density.
Using this equation we simulate 1000 trajectories and plot them corresponding to time.

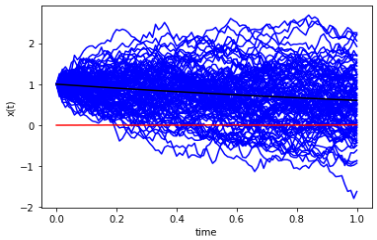

FIG. 1. Plotting 1000 trajectories of the system.
The black line represents the noise-free mean trajectory m(t) and the red line represents the variance P(t).

We know that $m(t) = e^{-\lambda t}x_0$, for $x_0 = 1$ and $\lambda = \frac{1}{2}$, lies between 0.6 to 1 which is approximately correct for the mean trajectory shown in the above plot.
Similarly, P(t) ranges from 0 to 0.4 which is again coherent with the results we have got in the plot. The difference in the accuracy of these trajectories is because we have used an approximate method which is bound to give deflected values.
The mean and variance trajectories look like straight lines in the given time range, but if we consider a longer duration we can easily see that the graphs are exponential.

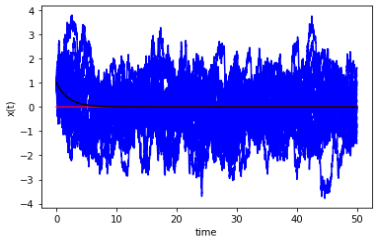

FIG. 2. Plotting 20 trajectories of the system for time range 0 to 50 units.

Hence, we can say that the behaviour of mean and variance trajectory is in accordance with the theoretical values we calculated.

II. HHL ALGORITHM

Aim : To solve the following set of linear equations through Classical (Gaussian elimination and NumPy) and Quantum (HHL: Qiskit) methods and compare the results.

$$x - \tfrac{1}{4}y = 4$$
$$-\tfrac{1}{4}x - y = 0$$

A. Canonical Form

The set of above given linear equations can be translated to a vector equation including a matrix A, vector x and constant vector b where $A = \begin{bmatrix} 1 & -1/4 \\ -1/4 & 1 \end{bmatrix}$, $\vec{x} = \begin{bmatrix} x \\ y \end{bmatrix}$ and $\vec{b} = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$, such that we can write the set of linear equations as A $\vec{x} = \vec{b}$

$$\begin{bmatrix} 1 & -1/4 \\ -1/4 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$$

B. Gaussian elimination

Now we solve the vector equation using the Gaussian Elimination Method. The motive is to obtain an identity matrix from the A matrix by applying row operations like swap, scalar multiplication and addition on the system. This would leave the variables x, y on the LHS and their values in the modified b vector.
Let's apply some row operations on the following system :

$$\left[\begin{array}{cc|c} 1 & -1/4 & 4 \\ -1/4 & 1 & 0 \end{array}\right]$$

$$R_2 \to R_2 + \tfrac{1}{4}R_1$$

$$\left[\begin{array}{cc|c} 1 & -1/4 & 4 \\ 0 & \frac{15}{16} & 1 \end{array}\right]$$

$$R_1 \to R_1 + \tfrac{4}{15}R_2$$

$$\left[\begin{array}{cc|c} 1 & 0 & \frac{64}{15} \\ 0 & \frac{15}{16} & 1 \end{array}\right]$$

$$R_2 \to \tfrac{16}{15}R_2$$

$$\left[\begin{array}{cc|c} 1 & 0 & \frac{64}{15} \\ 0 & 1 & \frac{16}{15} \end{array}\right]$$

This system translates to the following equation

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{64}{15} \\ \frac{16}{15} \end{bmatrix}$$

which implies that $\mathbf{x} = \frac{64}{15}$ and $\mathbf{y} = \frac{16}{15}$

C. HHL Algorithm : Working

In this section we will use the HHL algorithm to find the solution vector to the given linear equations. For an equation $A\vec{x} = \vec{b}$, with sparse A having a small condition number, this algorithm can find a function of $\vec{x}$, in run-time of ploy(log(N)) as compared to the fastest classical algorithms of run-time $N\sqrt{\kappa}$.[3] It uses the Quantum Phase Estimation method to get information on the eigenvalues of A and using them to apply unitary operators on the state $|b\rangle$.
Note that the matrix A is hermitian. So, we will be using it's spectral decomposition, as given below, in the calculations.[4]

$$A = \sum_{j=0}^{N-1} \lambda_j |u_j\rangle\langle u_j|, \quad \lambda_j \in \mathbb{R}$$

The state $|b\rangle$ and $|x\rangle$ can also be written in a similar form. Note that $|b\rangle$ is expressed in the eigenbasis of A.

$$|b\rangle = \sum_{j=0}^{N-1} b_j |u_j\rangle, \quad b_j \in \mathbb{C}$$

$$|x\rangle = A^{-1}|b\rangle = \sum_{j=0}^{N-1} \lambda_j^{-1} b_j |u_j\rangle$$

We will use three sets of quantum registers : n_l, n_b, n_a. n_l stores the binary representation of the eigenvalues of A. So, the bigger the matrix, more the number of qubits/quantum registers will be required. n_b contains the vector solution and it is the set of qubits where the quantum state $|b\rangle$ is loaded. All the other qubits are auxiliary qubits and are required for other intermediate steps. Given below is a circuit diagram for the HHL algorithm.

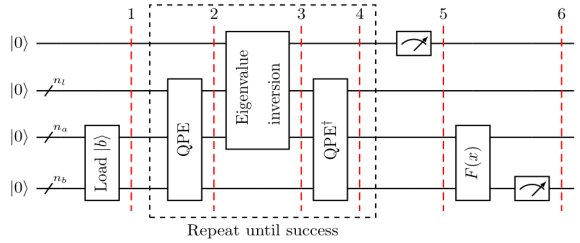

FIG. 3. Workflow of the HHL algorithm.
Source : Qiskit Notebook [4]

Here is the workflow of the algorithm [4]:

1. First we load the state $|b\rangle$ in the circuit.
   $|0\rangle_{n_b} \mapsto |b\rangle_{n_b}$
2. Applying Quantum Phase Estimation on $|b\rangle$.

   $$U = e^{iAt} := \sum_{j=0}^{N-1} e^{i\lambda_j t} |u_j\rangle\langle u_j|$$

   The obtained state can be written in the eigenbasis of A as :

   $$\sum_{j=0}^{N-1} b_j |\lambda_j\rangle_{n_l} |u_j\rangle_{n_b}$$
3. We then add an auxiliary qubit to the circuit and apply a gate which is a function of $|\lambda_j\rangle$ such that we receive the following state.

   $$\sum_{j=0}^{N-1} b_j |0\rangle_{n_l} |u_j\rangle_{n_b} \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}}|0\rangle + \frac{C}{\lambda_j}|1\rangle\right)$$
4. Now, we measure the auxiliary qubit in the computational basis. If the output is 1, then we can extract the quantum state $|x\rangle = A^{-1}|b\rangle = \sum_{j=0}^{N-1} \lambda_j^{-1} b_j |u_j\rangle$ from the system state which now looks like this :

   $$\left(\sqrt{\frac{1}{\sum_{j=0}^{N-1} |b_j|^2/|\lambda_j|^2}}\right) \sum_{j=0}^{N-1} \frac{b_j}{\lambda_j} |0\rangle_{n_l} |u_j\rangle_{n_b}$$
5. We can now apply any observable M and measure the state stored in n_b to to find it's expectation value.

D. Simulating HHL

In this section we write a program to calculate the solution vector using the numpy.linalg.solve() method and the HHL method. We then compare these two results.
We see that using NumPy we get
numpy vec = [[4.26666667], [1.06666667]]
with a norm of 4.3979.

The HHL algorithm as used from the qiskit.algorithms library implemented the following quantum circuit.

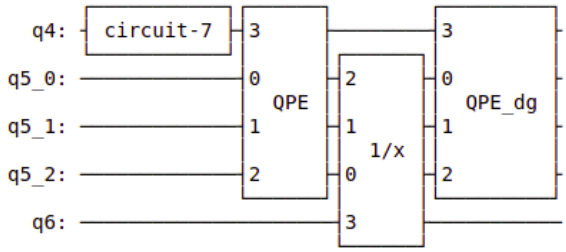

FIG. 4. HHL Quantum Circuit.
Note that the circuit doesn't measure the auxiliary qubit for us and that the operators are applied on different qubits compared to the circuit we used in the explanation part. (FIG. 2.). So, we will have to figure out the auxiliary qubit to flag and the set of qubits to read in order to get the solution vector.

If it were a real quantum computer, we could have measured the auxiliary qubit and then proceeded in

either of the two ways. First, by measuring the remaining quantum state a large number of times so as to get the solution vector. This method is not feasible as the quantum state collapses on measurement and it would consume a lot of resources to generate the state and measure it multiple times. The second way can be used to find an estimate to the expectation value of an operator associated with $|x\rangle$, which is the way the original HHL algorithm works.

As we are working on a quantum simulator, it can give us insights of the obtained state without having to measure it. We will be using Qiskit's Statevector simulator to get the required solution vector .

```
In [6]: from qiskit.quantum_info import Statevector
        hhl_sv = Statevector(hhl_solution.state).data

        hhl_vec = np.array([hhl_sv[16], hhl_sv[17]])
        print(hhl_vec)

        [0.75435554+3.30020041e-16j 0.24564446+2.55487056e-16j]
```

FIG. 5. Extracting x vector from HHL solution
Here we show how the Statevector function can be used to obtain the quantum state of interest. Note that the resultant state is not normalised. The state also has a complex part, but as it is very small, we will be ignoring it.

We now normalize the vectors obtained from the classical and quantum methods and compare them.
Normalised NumPy vector: [0.9701425 0.24253563]
and
Normalised HHL vector: [0.95085646 0.30963202]

We can see that the HHL algorithm has managed to make a good estimate of the $\vec{x}$. The accuracy of results depends on the number of qubits we use to store the eigenvalue of the A matrix.[5] Hence, it can be improved by adding more qubits to the circuit.

III. REFERENCES

[1] S. S arkk a and A. Solin, Applied Stochastic Differential Equations. Cam-bridge University Press, 2019.

[2] Øksendal, B. K. (Bernt Karsten), 1945-. Stochastic Differential Equations : an Introduction with Applications. Berlin ; New York :Springer, 2003.

[3] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," Physical Review Letters, vol. 103, no. 15, p. 150502, 2009.

[4] M. S. ANIS, Abby-Mitchell, H. Abraham, et al., "Qiskit: An open-source framework for quantum computing," 2021.

[5] An Iterative Improvement Method for HHL algorithm for Solving Linear System of Equations - https://arxiv.org/abs/2108.07744