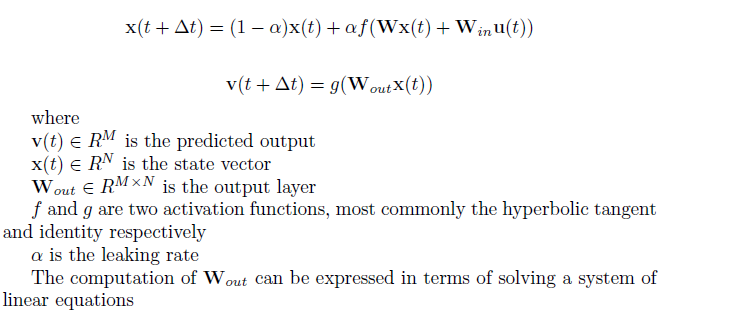
**Brief on classical Reservoir computing for time series predictions:**

**There is a small pdf called classical\_RC.pdf in this folder which one can see**.

We have a multivariate time series u(t) of dimension M, where t=1,2,3,4…….T (discrete time steps). We want to predict the time series at time step T+1.

Steps:

1. We want to map the univariate time-series into a high dimensional space. So we create a weight matrix (Win) of dimension (N X M) where N is the dimension we want to map u(t) to. N>M. We would do a matrix multiplication between u(t) and Win. This Win is called a reservoir coupler.
2. Next we also need a reservoir, let’s call that W. This would be a square matrix of dimension N. You can think of it as having N nodes in a neural network, but the best part is that we are not going to train a neural network here. W is similar to an adjacency matrix where we basically define the connections between the N nodes.
3. For both W and Win, the values are drawn randomly from a uniform distribution, but there are certain conditions that give good results. For W, it is important that it is sparse to some extent and the spectral radius<1: <https://www.kaggle.com/code/utahrandall/reservoir-computing-and-generalized-learning/notebook>
4. Further, here’s how we train and predict the time series:  
   
5. Above, v is the prediction for the next step, but above assumes we have an input u(t) for the previous step. So when we want to make predictions further, we would be using the predicted output from the previous step as the u(t) in the equation.

**The Quantum Analog of Reservoir Computing:**

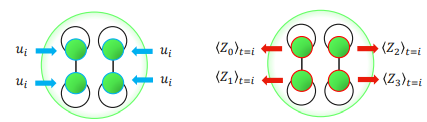
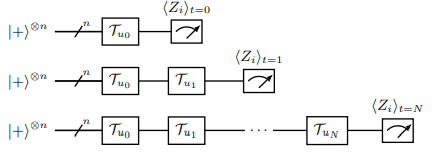
Reference:[**https://arxiv.org/pdf/2303.05488.pdf**](https://arxiv.org/pdf/2303.05488.pdf)

1. In the Quantum Analog, the multivariate regression for training and prediction remains the same as in the classical approach. The only difference now is that Win and W from above are arrived at using Quantum circuits. We will be working with univariate time series here.
2. For using a window of N time steps, we will need N quantum circuits, each with say, n qubits each. Below is how we construct the circuits:

* In each circuit, initialize in uniform superposition (with Hadamards on all qubits).
* For the 1st time step in our training window, we would be using the 1st circuit, where we will apply RX(theta) and RZZ(theta) where theta will be some scaled version of u(i) (Our window is u(i) to u(i+N) and we want to predict for u(i+N+1) onwards). RZZ(theta) would be using the pairwise entanglement scheme. The authors tried other entanglement schemes like linear and found this to be the best.
* For the 2nd time step in our training window, we would do the same and in addition will also repeat RX(theta2) and RZZ(theta2) with theta2 being the scaled version of u(i+1).
* We would repeat the previous 2 steps till u(i+N).
* Further, we will be adding noise channels in a probabilistic manner. The authors settled by reset noise channel after trying other types of noise channels: <https://qiskit.org/documentation/stubs/qiskit_aer.noise.reset_error.html>. *They also mention that with Dynamic circuits, one can easily perform resets by doing a mid-circuit measurement and applying a classically controlled X gate. I think they just mentioned that and did not actually use Dynamic circuits.*
* The probabilities we set for noise channels are like hyperparameters and we need to tune those, which the authors did using Scipy’s dual annealing: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.dual_annealing.html>
* Further, we calculate expectation value for every qubit in all the N circuits. The paper has this:  
  

*Which is not very clear to me. My understanding is that we need to get the reduced density matrices and then, find the trace:* [*https://qiskit.org/documentation/stubs/qiskit.quantum\_info.partial\_trace.html*](https://qiskit.org/documentation/stubs/qiskit.quantum_info.partial_trace.html)*, and this nice videos by Diego, this is the one on partial trace:* [*https://www.youtube.com/watch?v=L70TVZHYOsM*](https://www.youtube.com/watch?v=L70TVZHYOsM)

* The authors use Density simulator: 'aer\_simulator\_density\_matrix'.
* Once we get expectations for all qubits in each circuit, we would get something like a N\*n matrix of expectation values. For eg: if n=3, N=5, the 1st circuit’s expectation values would be [z11, z12, z13] (expectation values for qubits 1,2 and 3 in the 1st circuit or 1st time step*)*. Similarly, we would have [z21,z22,z23] for the 2nd time step and so on.
* The expectation values matrix from the above step and the actual time series values will be used for multivariate linear regression.
* For further predictions, we would be using the previous prediction value and creating a new circuit with the addition of RX and RZZ for angle corresponding to this predicted value, get the expectation values for all qubits and just predict with the weights we got as a result of linear regression. The predicted values would have to be scaled back.



**QAOA in QAOA**

Reference**:** [**https://arxiv.org/abs/2205.11762**](https://arxiv.org/abs/2205.11762)

**The paper specifically talks about max-cut, we would have to change our approach for weight calculation a little in the merging stage, I think we can use correlations:**

First step is to just divide the graph (in our case, we would split our portfolio into different sub-portfolios) and perform QAOA on each sub-graph.

**The merge process:**

1. Each subgraph is considered as 1 node. So in the example in the code, the original graph has 100 nodes. This graph is partitioned into 10 sub-graphs, each with 10 nodes. In the merge part of the algorithm, each sub-graph is taken as 1 node, so we have 10 nodes in the merged graph.

2. To compensate for not taking every individual node separately, like in the original graph, the graph from point 1 above is a weighted graph. Here is how the weight is calculated between any 2 sub-graphs subg\_1 and subg\_2:

- for every vertex i in subg\_1 and for every vertex j in subg\_2, 2 types of weights are incremented:

- w\_pos is incremented if i and j are not in the same class (+1 or -1 which is represented as 0 or 1 in bits) and have an edge between them in the original graph.

- w\_neg is incremented if i and j are in the same class and have an edge between them in the original graph.

- the cumulative edge weight between subg\_1 and subg\_2 is taken as the difference between w\_pos and w\_neg. This is the edge weight taken in the adjacency matrix of this merged graph.

- these edge weights are used in the coefficients while framing the Problem Hamiltonian for QAOA.

3. After QAOA on the merged graph, we get a solution with 10 bits because we are dealing with 10 nodes where each node is a sub-graph. This is what they refer to as Global solution in the paper. The solutions of sub-graphs are called Local solutions in the paper.

4. But we would like to interpret the solution in terms of the 100 nodes of the original graph. This is done by combining solutions from the subgraph in the following way:

- if the bit value corresponding to a sub-graph in the solution of merged graph is 1, then the solution of the sub-graph is taken as it is in the overall global solution,

else if it is 0, it is taken as the complement i.e if in the QAOA solution of sub-graph, a node is 0, then it would be taken as 1 in the global solution, else it would be taken as 0.

This point can be seen in the figure from the paper, look closely at the orange and green nodes before and after arriving at the solution of the merged graph.

5. It is to be noted that because the default policy option for partitioning the graph is random, one would get slightly different solutions each time one runs the code.

**DATA:**

**This is something I am most worried about, but I think we should just stick to something from which we can easily fetch data using python** [**https://towardsdatascience.com/time-series-and-correlations-with-stock-market-data-using-python-e66774e3a16f**](https://towardsdatascience.com/time-series-and-correlations-with-stock-market-data-using-python-e66774e3a16f)**. I think we can stick to Yahoo Finance, but we need to decide which stocks we really want to work with.**