Here we provide some *rough indications* of what could have been good solution directions for the questions of assignment 2. Note: for many of the modeling tasks, multiple solutions could have been viable. However, as was stressed in the assignment description, the emphasis is on the identification of symmetries and structure in the task at hand, and the ability to somehow incorporate those aspects into a solution formulation for a problem setting which was not seen during the practicals. In addition, we have provided example answers for the experiments, but keep in mind that there are many possible valid solutions here.

Q2:
- we have permutation symmetry (equivariance) because the ordering in which the planets are observed does not matter (it is a set of planets);
- we have rotation symmetry (equivariance) because rotating the system does not affect the physical behavior;
- we have periodic translation symmetry (equivariance) because translating the system (through the boundaries) does not affect the physical behavior,;
- we have reflection symmetry (equivariance) because mirroring the system does not affect the physical behavior.

Q3a: The task is a regression problem.
Q3b: Since we have to predict continuous-valued outputs, the task is a regression problem.

Q4: Our design choices are primarily motivated by the symmetries identified in Q2. We choose to use a message passing GNN. A single message passing layer is defined by the following equations: **message passing equations as in the practical**. By using a GNN, we automatically take permutation symmetry into account, and we can also handle the varying amount of atoms per trajectory. To handle the translations, rotation and reflection symmetries (E(2)), we encode the distances on edges between nodes, while we do **not** use absolute positions and velocities as node features, as this will make the model invariant to these symmetries. Instead, we use the magnitude/norm of the velocity, as this is invariant to the aforementioned symmetries. In order to handle periodic boundary conditions, we make use of the minimum image convention when calculating distances.

Q5a: Our model consists of 3 neural message passing layers. We use summation as (nodewise) aggregation function, and implement the node update and edge transfer functions as two-layer MLPs. We use 64 neurons and ReLU activation functions for all hidden layers. The final node update function has linear activation function and 1 output neuron. We then perform global sum pooling to predict the energy. This way the model implicitly learns to calculate the contribution of each atom to the total potential energy of the system. In the input graph, we draw edges between all atoms who are closer than 10Å, since atoms further apart have negligible interactions.
Q5b: We train the model by calculating the MSE between the true and predicted energy. We train for 100 epochs since we observed that training converged after this amount. We use a batch size of 64, and use the Adam optimizer with a learning rate of 0.0001, and default settings for the remaining optimizer parameters.

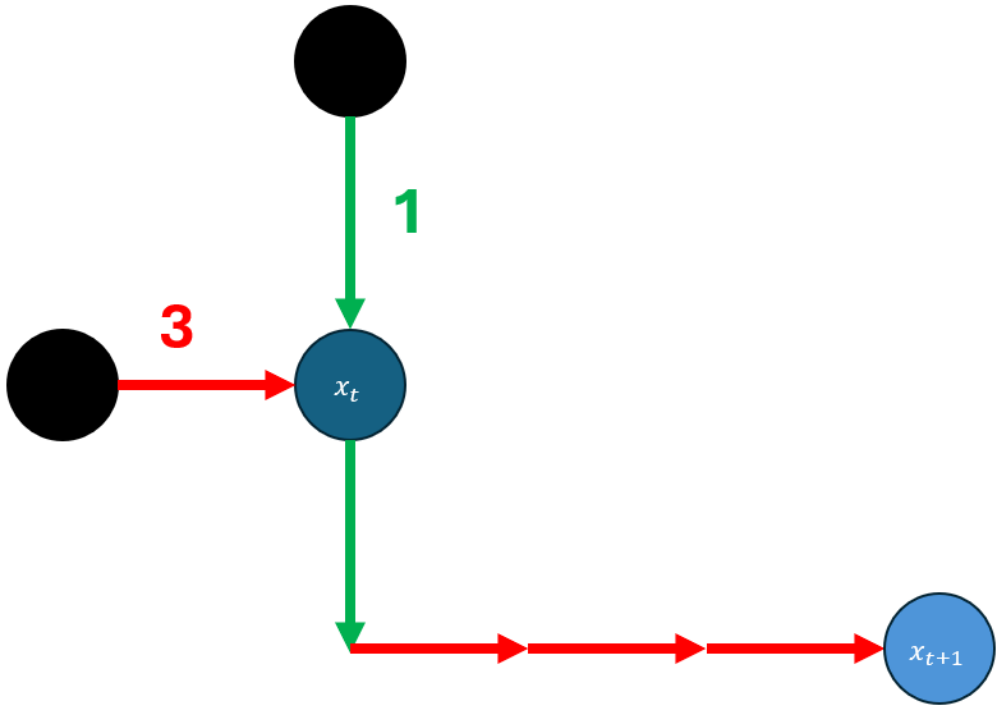
Q6c: We visualize the state of the system for a good prediction and bad prediction. Maybe the model is not good at handling high velocities, which causes a bad prediction.

Q7a: a (sequence) regression problem / sequence prediction problem / sequence generation problem.
Q7b: We need to predict a sequence of continuous-valued outcomes.

Q8: Since the state of the system at t+1 is only dependent on the state of the system at t, we want to build a model which predicts x_{t+1} and v_{t+1} for each atom. We use a similar

model as in task 1, however, we need to make a few changes, since the predictions need to be equivariant with respect to E(2).

To this end, we change our GNN to predict two scalar values (1 for position, 1 for velocity) per edge. For each node, we calculate the change in position using the following process. For each incoming edge, we multiply the position scalar with the edge vector (vector between the nodes). We take the sum of the multiplied incoming edges, and use the resulting vector to update the position. We follow a similar process for velocities. Naturally, when calculating the vectors, we take all the periodic boundary conditions into account. Since the edge vectors are equivariant to E(2), our model also becomes equivariant to E(2). An illustration of this process is visible below. Here, the black atoms are charged the same as the dark blue atom, so the dark blue atom should be pushed away. If the forces were attractive between the atoms, this specific model would need to predict negative values at the edges.



However, not all model which use vectors between nodes instead of distances remain equivariant to E(2), as you can only make linear transformations based on these vectors (multiplying them by scalars). Therefore, taking a “normal” message passing layers where the edge attribute is the edge vector will not result in an equivariant model.

Additionally, if you choose to only work with the positions and not with velocities, you do not have access to the full state of the system. In such a case, you could use a solution where each node retains a “memory” (like an LSTM) to implicitly capture information about velocities.

Q9a: the model implementation remains unchanged with respect to task 1, except for the edge wise output layer, and the new position/velocity calculation mentioned in the model formulation.
Q9b: We train the model using all pairs of t and t+1 in the training data (so out of 1 trajectory we create 39 training samples). We calculate the periodic squared distance between predicted and true positions (MSE) and the MSE between predicted and true velocities and use this to train the model. The rest of the training procedure is the same as in task 1.

Q9c: We give the model velocities, positions and charges of atoms at t=0. Then, we predict velocities and positions at t=1. We then keep feeding in the predicted positions and velocities autoregressively to predict the rest of the sequence until t=39.
Q10c: Example: Visualize a bad and good trajectory. The bad trajectory could be realistic, but due to a small mistake in the simulation, some atoms follow a different path, which gives a higher MSE than the baseline. The good trajectory could be easy to predict for the model, because there are few interactions present (atoms are far apart).
Q10e: Example: Rather than predicting the next time step, we autoregressively predict the full sequence and calculate the loss to train the model. Then we compare the performance of the 2 models.

Q11a: a *conditional* (sequence) regression problem / sequence prediction problem / sequence generation problem.
Q11b: We now also need to include the crystal in our model, which affects how the atoms move. Therefore we are conditioning the trajectory generation on the crystal.

Q12: Our model remains the same as in task 2. We include the crystal atoms in the graph, but do not update their positions and velocities.

Q13: The architecture and training remains identical to task 2. During training, we only calculate the loss with respect to the positions and velocities of the moving atoms. During inference, we set the velocities of crystal atoms to 0 and do not allow these to move.

Q14c: Example: We visualize the distribution of true velocities, and velocities predicted by our model. If the distributions are (reasonably) matching, this is an indication that our model generates realistic behaviour. Otherwise, there might be bad trajectories generated.

Q14e: Example: During training and inference, we allow the crystal atoms to move. We then check if this improves predictions of the trajectories, and whether the model learns to keep the crystal atoms in place.