Problem 1

For this problem we read in a sequence of gates which specify the rotation axis and the rotation angle. We then attempt to optimise this sequence so that we can apply the total logic circuit but with fewer operations.

The optimisation is done with the help of a minimisation function which optimises the rotation angles. We also know that any rotation sequence can be carried out with only three gates, a rotation around the x-axis, y-axis and then the z-axis. The order of these three rotations does not really matter, but of course the angles of rotation may change if the order is changed. So we have a minimisation procedure with three variables (the three rotation angles).

There is a final check in case any angles are zero, in which case these amount to an identity operator and these rotations can be removed.

Testing:

Test case 1: input_str = "{X}(90), {X}(90)"

Output 1: out_str = “{X}(180.0)”

Thus the code can combine rotations about the same axis correctly.

Test case 2: input_str = “{X}(90), {Y}(180), {X}(90)"

Output 2: out_str = “{Y}(180.0)”

Test case 3: input_str = “{X}(75), {Y}(180), {X}(90), {Z}(35), {Y}(40)"

Output 2: out_str = “{X}(18.208), {Y}(227.274), {Z}(-22.354)”

So the code can handle long sequences of gates and some non-standard strange input angles.

Problem 2

This problem is similar to the previous case, but instead of rotations about the y-axis, we must apply Y(theta) = Z(90)X(theta)Z(-90). The approach to implementing this is the same as before, but in our optimisation function (Rot_min_2.py) we include the above substitution for the Y rotations.

Testing:

Test case 1: input_str = "{X}(90), {X}(90)"

Output 1: out_str = “{X}(180.0)”

This is the same as before. Meaning that no Y rotations gives the same as before.

Test case 2: input_str = “{X}(90), {Y}(180), {X}(90)"

Output 2: out_str = “{Z}(90), {X}(180.0), {Z}(-90.0)”

This is directly the Y rotation substitution

Test case 3: input_str = “{X}(75), {Y}(180), {X}(90), {Z}(35), {Y}(40)"

Output 2: out_str = “{X}(18.208), {Z}(90), {X}(227.274), {Z}(-112.354)”

This is good. The code can handle complicated sequences and will absorb sequential Z gates into one rotation.

Problem 3

This problem is similar to the previous case, but now, there is a time cost for applying the gates. This means that fewer gates are preferable.

The approach is to initialise the minimisation (as in problems 1 and 2) at different points in parameter space (three parameters, three rotation angles) using a pseudo-random number generator. The minimisation will find a different combination of rotations (as these are not unique). We then check which rotation sequence has the smallest time cost and then we use that one.

Testing:

Test case 1: input_str = "{X}(90), {X}(90)" ; lenZ=100; lenX=10

Output 1: out_str = “{X}(180.0)”; time = 10

Test case 2: input_str = “{X}(90), {Y}(180), {X}(90)" ; lenZ=100; lenX=10

Output 2: out_str = “{X}(180.0), {Z}(-180.0)”; time = 110

This is different to problems 1 & 2, but the total operational is equivalent. We have found an equivalent operation but now with fewer gates.

Test case 3: input_str = “{X}(75), {Y}(180), {X}(90), {Z}(35), {Y}(40)" ; lenZ=100;lenX=10

Output 2: out_str = “{X}(-161.792), {Z}(90), {X}(-47.274), {Z}(67.646)”; time = 220

Problem 4

In this problem we now have one and two qubit gates and the goal is to read in a gate sequence and attempt to optimise and reduce the number of logic operations (and the time taken for the circuit).

The approach for problem 4 is to read the sequence of gates, find any rotations around the same axis on the same qubit and attempt to bring these together in the sequence (so they can be combined and applied as a single rotation). This is done by attempting to commute one rotation through all the rest in the sequence.

We know that applications on different qubits can be interchanged (commuted) and CX(0,1) commutes with R_X (on qubit 1) etc.

For an X and Z gate, we attempt to variationally minimise the error for applying rotations around these axes in reverse order but with different angles.

Testing:

Test case 1: input_str = "{X}(1,90), {Z}(1,180), {CX}(0,1), {X}(1,90)”

Output 1: out_str = “{Z}(1, 180.0, 0.0), {CX}(0,1,10.0)”

This is the example given in the question to make sure things are working properly.

Test case 2: input_str = “{X}(1,90), {Z}(1,180), {CX}(1,0), {X}(1,90)”

Output 2: out_str = “{X}(1, 90.0, 0.0), {Z}(1, 180.0, 10.0), {CX}(1,0,20.0), {X}(1, 90.0, 120.0)”

This is to show the limitations of the approach. CX(1,0) and X(1) do not commute as these involve rotations on the same qubit, so the optimisation procedure stops.
In principle, the code could be improved by attempting a variational optimisation procedure for the commutation relation [CX(1,0),X(1)].

Test case 3: input_str = “{X}(1,90), {Z}(0,180), {CX}(0,1), {X}(1,90)"

Output 2: out_str = “{X}(1, 180.0, 0.0), {Z}(0, 180.0, 0.0), {CX}(0,1,10.0)”

The code can recognise that qubit Z is on a different qubit (so can commute X straight through. Secondly, the code can infer that the X rotation and the Z

rotation can be carried out simultaneously at time 0 followed by the CX gate at time 10ns.