SKKU 2024 Challenge #3: Heisenberg Spin Glass

Introduction

Condensed matter physics encompasses many models of real-world physical systems. Studying these simplified models yields insights into the behavior of materials that can be analyzed in a laboratory setting. This challenge defines one such model and leads us on a path to understand some of its behavior using the tools of quantum computing.

The system that we study is a combination of two models that are well-known in condensed matter physics. One is a spin glass (https://en.wikipedia.org/wiki/Spin_glass). This model captures many of the essential features of random systems in which there is competition between order and disorder. The other is the Heisenberg model (https://en.wikipedia.org/wiki/Quantum_Heisenberg_model). This is a simplified model of many quantum systems that occur in nature and is designed to allow theorists to study phase transitions and critical points.

In this challenge, we will first apply symmetry concepts to characterize the Heisenberg model over four spin-$1/2$ quantum degrees of freedom. Given $\{+1, -1\}$ weights for the pairwise interactions between spins, we determine the number of symmetry classes for this model. Next, we apply the adiabatic theorem to determine the ground state for several such models. This represents our first link to quantum computing, because we can then estimate the depth of a circuit which converges to the ground state of one specific model.

In the final portion of the challenge, we use a different approach to generate the ground state of our chosen model, the Heisenberg spin glass, whose Hamiltonian we denote H_{sg}. Recognizing that the ground state exists within a fixed Hamming weight sector of Hilbert space, we examine circuit elements that will take us into that sector (i.e., subspace of the entire Hilbert space) and then rotate states within that sector. This allows us to generate the correct probability distribution over computational basis states for the ground state, but the phases of the basis states are not correctly determined. A final phase rotation allows us to correct the phases and thus yields the correct ground state for the Heisenberg spin glass. Once we have prepared the ground state via a quantum circuit, we measure the ground state energy via measurements in several different computational bases.

Part 1 : A first look at the Heisenberg spin glass

The Pauli $\sigma_x, \sigma_y, \sigma_z$ matrices operate on spin-$1/2$ degrees of freedom. The Heisenberg interaction between a pair of spins i and j is conventionally written as follows:

$$H_{ij} = \sigma_x^{[i]}\sigma_x^{[j]} + \sigma_y^{[i]}\sigma_y^{[j]} + \sigma_z^{[i]}\sigma_z^{[j]}$$

This interaction term may also be expressed using tensor product notation in which the first and second factors in the tensor product operate on spins i and j:

$$H_{ij} = \sigma_x \otimes \sigma_x + \sigma_y \otimes \sigma_y + \sigma_z \otimes \sigma_z$$

The four-spin model that we will consider comprises pairwise interaction terms of the above form between all four spins. Each of the pairwise interaction terms is multiplied by a weight J_{ij} which we take from the set $\{+1, -1\}$. The sign of J_{ij} determines whether the interaction between the corresponding spin pair is ferromagnetic ($J_{ij} = -1$) or antiferromagnetic ($J_{ij} = +1$). The fact that these couplings can be either ferromagnetic or antiferromagnetic means that we are studying a spin glass.

Our starting point is the pure Heisenberg model in which there are only pairwise interactions between spins. The model is thus:

$$H_{Heisenberg} = \sum_{<ij>} J_{ij} H_{ij}$$

There are six J_{ij} coefficients chosen from $\{+1, -1\}$. Any one assignment of $\{+1, -1\}$ values to the J_{ij} coefficients yields a single pure Heisenberg model, and since there are six J_{ij} coefficients, there are $2^6 = 64$ different models that we can study.

Symmetry plays an important role in simplifying the study of many physical systems. This system is no exception. We consider two symmetries, both of which will reduce the number of possible models. The first symmetry is interchanging spins. For example, consider the model in which all the J_{ij} are set to -1 except for J_{01}, which equals $+1$. By applying spin interchange to spins 1 and 2, we could obtain a second model in which all J_{ij} are set to -1 except for J_{02}, which equals $+1$. These two models are equivalent in the sense that their eigenspectra are the same.

These two equivalent models live within a symmetry class of equivalent models. In this class, all the J_{ij} coefficients except for one are equal to -1 and the single remaining coefficient is equal to $+1$. Since there are six coefficients that we could choose to receive the unique value of $+1$, there are a total of six equivalent models in this symmetry class.

The second symmetry is sign flip of all the J_{ij} coefficients. Since we require the J_{ij} coefficients to come from the set $\{+1, -1\}$, we can flip the signs of all the coefficients to generate a new model. It is possible that the spectrum of the first model is the same as the spectrum of the second model in which the signs of all the J_{ij} coefficients have been flipped. In this case, the two models are equivalent under sign change.

The first exercise is to determine the number of symmetry classes for the four-spin Heisenberg model, in which all the J_{ij} coefficients must come from the set $\{+1, -1\}$. Two models are members of the same symmetry class if the spectra of the two models are identical. We are not yet building quantum circuits, so the tools we will use to solve this problem are programs for classical computers which extract eigenvalues for quantum Hamiltonians.

We will provide some starting steps, in Python and using NumPy and SciPy, for this exercise. The Hamiltonian for the four-spin system is a 16×16 Hermitian matrix which is composed of a sum of individual terms. Each term can be written as tensor products of Pauli matrices and a 2×2 identity matrix. For example, the pairwise interaction term between spins 0 and 1 can be expressed programmatically as follows:

```
import numpy as np

X = np.array([[0., 1.], [1., 0.]])

Y = np.array([[0., -1j], [1j, 0.]])

Z = np.array([[1., 0.], [0., -1.]])

I = np.array([[1., 0.], [0., 1.]])

H_0_1_XX = np.kron(I, np.kron(I, np.kron(X, X)))

H_0_1_YY = np.kron(I, np.kron(I, np.kron(Y, Y)))

H_0_1_ZZ = np.kron(I, np.kron(I, np.kron(Z, Z)))

H_0_1 = J_0_1 * (H_0_1_XX + H_0_1_YY + H_0_1_ZZ)
```

The cascaded series of NumPy Kronecker products represents the mathematical tensor product. With the ordering of the Kronecker products above, the bit-ordering corresponds to Qiskit's convention. Extending the above code fragment, build the entire pure Heisenberg Hamiltonian for all 64 possible four-spin models. Then, using an appropriate numerical library (e.g., SciPy) extract the spectrum for all 64 models. If the spectra of two models is the same, then the models are in the same symmetry class. If they are different, then the models are in different symmetry classes.

Exercises for Part 1: [easy – ☆]

1.a) How many symmetry classes are there?

1.b) For each symmetry class, provide one set of J_{ij} coefficients for a model in that symmetry class.

1.d) What is the eigenspectrum of each symmetry class?

1.e) Do any of the symmetry classes have an eigenspectrum that is sign invariant? In other words, if you change the sign of each eigenvalue, does the overall set of eigenvalues remain unchanged?

Part 2 : Adiabatic evolution to the ground state

The adiabatic theorem of quantum mechanics (https://en.wikipedia.org/wiki/Adiabatic_theorem) states that if a system is prepared in its ground state and then the Hamiltonian is gradually changed, the system will remain in its ground state. We will use this method to create the ground state for a Heisenberg spin glass. The adiabatic theorem is relevant primarily for quantum annealing, but it also can be used to provide insight into quantum circuits for gate model quantum computing platforms.

In the context of quantum annealing, the starting Hamiltonian is usually the ground state of a mixer operator H_m. This mixer is typically a sum over all qubits of σ_x:

$$H_m = \sum_i \sigma_x^{[i]}$$

For a single qubit system whose Hamiltonian is σ_x, the ground state is a $50/50$ superposition of the $|0\rangle$ and $|1\rangle$ states, with a phase of $+1$ between the two states. For a multi-qubit system, the ground state of the Hamiltonian H_m is a tensor product of this ground state over all qubits.

In our adiabatic evolution, we will start our four-qubit system in this ground state and then change the Hamiltonian into the Heisenberg spin glass. We use the variable s to determine how far we are in this evolution process and so we have a time-dependent Hamiltonian which is a function of s. As s goes from 0 to 1, the Hamiltonian interpolates linearly from the pure mixer operator to the Heisenberg spin glass:

$$H(s) = (1 - s)H_m + sH_{sg}$$

Construct the time-dependent Hamiltonian programmatically (Python programs can use NumPy and SciPy for convenience). As s interpolates from 0 to 1, extract the sixteen eigenvalues. Divide the s interval from 0 to 1 into 1000 equally spaced points and compute the eigenvalues at each point. Graph the sixteen eigenvalues as a function of s - for each symmetry class obtained in Part 1. In each plot, the horizontal axis represents s as it varies from 0 to 1 and the vertical axis represents energy.

Looking at the eigenspectra as a function of s, there are two things to note. The eigenspectrum of the Heisenberg spin glass is captured by the right-most vertical slice through these graphs. For many of the symmetry classes, several energy eigenvalues

converge at the ground state when $s = 1$. We will not use these models for the last part of our challenge, because the ground state is not unique.

The second thing to note is that for many of these models, the line tracing the ground state crosses another eigenstate at some intermediate s value. These crossings cause the adiabatic algorithm to fail. At these crossings, it is possible for probability which has remained in the ground state to leak into the excited state.

All the symmetry classes for the Heisenberg spin glass suffer from one or both pathologies. For that reason, we will modify our Heisenberg spin glass in two ways before we do any further analysis. First, we will allow the J_{ij} coefficients lie outside the set $\{+1, -1\}$. Second, we will include a new term in our Hamiltonian which captures an interaction between the Z-component of a magnetic field and the spin. The final model then has this form:

$$H_{sg} = H_{Heisenberg} + \sum_i h_i \sigma_z^{[i]}$$

For the remainder of the challenge, here is the specific set of coefficients that we will use for the J_{ij} and h_i terms in the Hamiltonian:

$$h_0 = 1;\ h_1 = -1;\ h_2 = 2;\ h_3 = 3;$$

$$J_{01} = -1;\ J_{02} = 2;\ J_{03} = 3;\ J_{12} = -2;\ J_{13} = -3;\ J_{23} = -4;$$

Using this Hamiltonian, which we call H_{sg}, compute the eigenspectrum during the adiabatic evolution from the $s = 0$ Hamiltonian which is comprised purely of the mixer operator to the $s = 1$ Hamiltonian which is the Heisenberg spin model defined above. This is exercise 2.b below.

We will now use the adiabatic algorithm to demonstrate that it is possible to converge to the ground state of H_{sg}. The first portion of this exercise is programmatic. After demonstrating convergence using numerical integration, you will estimate the quantum circuit depth required to implement the adiabatic solution.

Adiabatic evolution is simply time evolution of the Schrödinger equation. The only additional complexity is that the Hamiltonian is not time independent. We address this by breaking the time evolution into many small-time steps. Within each small-time step, the Hamiltonian changes very little and so we can approximate the time evolution during one time step by assuming that the Hamiltonian is fixed. However, as we progress from time step to time step, the Hamiltonian will evolve. In other words, we will take the effective Hamiltonian from time s to $s + ds$ to be fixed, and then for the next time step, from $s + ds$ to $s + 2ds$, the Hamiltonian will be slightly different.

The time evolution during one small time step is simply given by exponentiating the Hamiltonian after multiplying it by ds and $-i$. The Hamiltonian is a 16×16 Hermitian matrix and so the exponentiation requires a numerical package, such as SciPy, that provides a function for matrix exponentiation. For this exercise, you will fix two parameters. The first sets the total time τ for the adiabatic evolution and the second sets the number of time steps N into which the total time interval will be divided.

A key aspect of the adiabatic theorem says that the more slowly the Hamiltonian evolves from its initial state to its final state, the higher the probability of remaining in the ground state. Thus, we expect to get better convergence to the ground state of H_{sg} if we choose the total time τ to be large. Choosing τ to be large means, however, that we should increase the number of steps N so that the numerical integration remains accurate.

For this exercise, we will not try to match the exact ground eigenstate. Instead, simply attempt to match the probabilities of the sixteen computational basis states. More precisely, you can diagonalize H_{sg} exactly (e.g., using tools from SciPy) and then convert the ground state eigenstate into probabilities by taking the norm of each of the sixteen probability amplitudes. This vector of sixteen probabilities forms the target for our adiabatic evolution. Now, after writing a program to carry out adiabatic evolution, experiment with the total time and number of steps.

Once you have determined τ and N needed for adiabatic evolution to get close to the exact answer, you can estimate the circuit depth required to implement this computation. For each step, we must simulate time evolution of a Hamiltonian that is a superposition of H_{sg} and H_m. Since individual time step will be small, we can approximate the exponentiated Hamiltonian by assuming that all the terms in H_{sg} and H_m commute with one another. Thus, we can identify individual terms in H_{sg} and H_m with circuit elements.

For example, if we had a term like σ_z in our Hamiltonian, then the exponentiated version of that term would give rise to an RZ gate on the corresponding qubit. Likewise, if we had a term like $\sigma_z \otimes \sigma_z$ in the Hamiltonian, the exponentiated version of that term would give rise to an RZZ gate on the corresponding qubit pair. Both these terms are present in H_{sg} and there are additional terms as well.

Exercises for Part 2: [medium]

2.a) Plot the eigenspectra as a function of s for each symmetry class.

2.b) Plot the eigenspectrum for H_{sg} as a function of s.

2.c)[☆] Write a program to carry out adiabatic evolution, starting from the pure mixer Hamiltonian and ending with H_{sg}. Two parameters govern this program: the time τ over which the adiabatic evolution takes place and the number of time steps N.

2.d) In order to match the target vector of sixteen probabilities by performing adiabatic evolution, what value do you need to use for τ and N? The exact target probabilities and the probabilities resulting from adiabatic evolution should match to three significant decimal digits.

2.e) List out the terms in H_{sg} and H_m and, for each term, provide the gate that arises when that term is exponentiated. You can use any gate in Qiskit's standard gate set. How many single-qubit gates are needed for each time step? How many two-qubit gates are needed for each time step?

2.f)[☆] Assess the circuit needed to carry out the adiabatic evolution. Roughly how many single-qubit gates and two-qubit gates would be required to achieve three-digit accuracy for the sixteen probabilities characterizing the ground state? Do you think this is practical?

Part 3 : A simpler path to the ground state

In this final section, we will use a more direct technique to build a circuit to generate the ground state of H_{sg}. To do this, we will first notice an interesting property of the ground state. This property suggests that the mixer used in the prior section is not well-matched to the problem. We will introduce a different mixing operator which is a better match to the problem and then use it to construct an efficient quantum circuit to produce the ground state.

Using a programmatic technique (e.g., SciPy's eig()) routine, extract the ground state eigenvector of H_{sg}. Of the sixteen probability amplitudes, note that twelve vanish and four are non-vanishing. Do you see a similarity amongst the four basis states corresponding to non-vanishing probability amplitudes?

These four basis states share an invariant called Hamming weight. The Hamming weight of a basis state is simply the number of qubits in the basis state that are "turned on". For example, the basis state $|1110\rangle$ has Hamming weight three, because three of the four qubits are "turned on". The four basis states that appear in the ground state of H_{sg} all have Hamming weight three. They are $|1110\rangle, |1101\rangle, |1011\rangle, |0111\rangle$.

Note that the four-qubit Hilbert space can be divided up into sectors according to Hamming weight. The possible Hamming weights are 0 through 4, inclusive. The

dimension of each of these five sectors is given by a row in Pascal's triangle. The approach that we follow in this section is to use the Hamming weight invariant to simplify the task of building a circuit to generate the ground state of H_{sg}.

We now introduce a useful tool to help us with this approach: the XY-mixer. This mixer operator has been used in a variety of quantum circuits to solve constrained optimization problems. It is useful because it will allow us to move a statevector within a Hamming weight subspace without pushing the statevector out of the subspace. For more information about the XY-mixer, see:

> *Hadfield, S. et al. From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. Algorithms 12, 34 (2019)*

and:

> *He, Z., Shaydulin, R., Chakrabarti, S. et al. Alignment between initial state and mixer improves QAOA performance for constrained optimization. npj Quantum Inf 9, 121 (2023)*

The form of the XY-mixer operator is:

$$e^{-i\theta(\sigma_x \otimes \sigma_x + \sigma_y \otimes \sigma_y)}$$

It is a two-qubit operator and takes a single parameter θ, which may be thought of as the angle through which the rotation occurs. Note that when $\theta = \pi/4$ the XY-mixer becomes a SWAP operator. A SWAP operator also preserves the Hamming weight.

We can now outline the strategy for this approach. A quantum circuit starts with all qubits in the "turned off" state: $|0\rangle$. We need to move the statevector into the Hamming weight three subspace, so the first step is to apply gates that will "turn on" three of the qubits. Since an X gate interchanges $|0\rangle$ and $|1\rangle$ states, we can apply three X gates to move the statevector into the desired Hamming weight subspace.

The next step is to adjust the position of the statevector within this Hamming weight subspace using XY-mixer operators. Each XY-mixer is defined over two qubits and thus there are a total of six XY-mixer operators we could apply to rotate the statevector within this subspace. Each XY-mixer includes an angle parameter θ. The goal is to find a sequence of XY-mixers that transforms the statevector so that it has the correct probability for each of the four non-vanishing basis states with Hamming weight three.

Do this programmatically using an optimization routine, such as `minimize()` from the SciPy function library. Build a function - not yet a circuit – by doing the following

1) begin with an initial state vector that corresponds to a qubit register of $|0000\rangle$

2) apply X operators to transform $|0000\rangle$ into a state with Hamming weight three

3) apply three XY-mixers to the result of the prior step - each XY-mixer operator has the form shown on the last page, which can be converted into a 16×16 matrix via Kronecker products from NumPy or some other suitable package

4) determine the distance between the final state produced in the prior step and the target state, which is the probability vector for the sixteen basis states in the ground state of H_{sg}

5) use a programmatic function like `minimize()` to tune the angle parameters applied in the prior step

These steps above are exercise 3.a. To complete this step, you may need to experiment with choosing different qubit pairs for the three XY-mixers. Only certain XY-mixers will yield convergence to the ground state probabilities of H_{sg}.

Now that you have determined the three required applications of the XY-mixer, turn these mathematical operations into a quantum circuit. Step 2 is easy - X operators exist within Qiskit's gate set - or the gate set of any quantum programming language. The XY-mixer operators require a little more work. Note that an XY-mixer is an exponential of the sum $\sigma_x \otimes \sigma_x + \sigma_y \otimes \sigma_y$. These two terms individually can be exponentiated and their exponential form can be found in many gate sets. Qiskit, for example, has an RXX operator and an RYY operator. These are the exponentiated forms of $\sigma_x \otimes \sigma_x$ and $\sigma_y \otimes \sigma_y$.

At this point it is reasonable to be concerned that the above paragraph contains an unwarranted simplification. The suggestion above is to implement the XY-mixer via an RXX and an RYY gate element. This assumes tacitly that the exponential of a sum of operators can be replaced with the product of the exponentials of the individual operators. A brief consideration suggests that this may not be correct. After all, it is not the case that the exponential of σ_x plus σ_y is equal to the product of the individual exponentials. It is an interesting coincidence that this identity does, in fact, hold for the $\sigma_x \otimes \sigma_x$ and $\sigma_y \otimes \sigma_y$. The exponential of their sum does in fact equal the product of the individual exponentials. The reason for this is that $\sigma_x \otimes \sigma_x$ does commute with $\sigma_y \otimes \sigma_y$, even though σ_x does not commute with σ_y.

After completing the above steps, you should have a quantum circuit that replicates the ground state probabilities of H_{sg}. We cannot declare victory, yet, since quantum states are characterized not just by the probabilities of individual basis states, but by their relative phases. And so, we must determine whether the circuit above produces the four non-zero computational basis states with the correct relative phases.

Since the H_{sg} Hamiltonian is Hermitian, we know that all the eigenvectors are purely real. At this point you should check the final state produced by your quantum circuit to see if it satisfies this constraint. If you are using Qiskit, this is easy to do with the `Statevector` object that can be constructed from the quantum circuit. The data attribute of the Statevector object contains the complex statevector as a NumPy array. You can easily inspect it by eye to see whether all its non-zero components are real.

At this point, depending on decisions that you made when choosing your XY-mixer operators and rotation angles, your results may differ from the ones I obtained when solving this problem. My statevector did not have purely real components, and so it was necessary to add one further circuit element to address this phase problem.

In case you need to do this, too, here is a hint. In my case, of the four non-zero components of the ground state, three had a real phase and one had a purely imaginary phase with a negative sign. Thus, I needed a gate or circuit element which would rotate the three components with positive phase in one direction and the single component with negative imaginary phase in the opposite direction. An RZ gate operates in exactly this way - it rotates half of the computational basis states with a positive phase and the other half with the negative of that phase. The question to answer here is whether the pattern of positive and negative rotations provided by an RZ gate matches the required rotations.

Remember that there are four possible RZ gates, depending on the qubit to which the gate is applied. Each of the possible RZ gates rotates the states in which the chosen qubit has value $|1\rangle$ in one direction and the states in which the chosen qubit has value $|1\rangle$ in the opposite direction. So, we must look at the four non-zero computational basis states of H_{sg} and determine whether we can find a qubit which has the same value in the three states that must be rotate in one direction and the opposite value in the fourth state that must be rotated in the opposite direction.

With the rotation angles and ordering of qubit pairs for the XY-mixers that I found above, there is a qubit whose RZ gate provides the necessary rotation pattern. When added at the end of my quantum circuit and adjusted with the appropriate rotation angle, the three states with real phase were rotated in one direction and the one state with negative imaginary phase was rotated in the opposite direction. By choosing the rotation angle for the RZ carefully, I adjusted the phases of the circuit output to be equal to the numerically obtained ground state of H_{sg} - after applying a global phase.

At this point you will have a circuit that generates the correct ground state of H_{sg} - both probabilities and phases. The last step of the challenge is to evaluate the energy of this generated state and validate that the energy matches the ground state eigenvalue computed directly.

To complete this step, recognize that the Hamiltonian is a sum of terms and in any given computational basis, only a subset of those terms can be evaluated. In the Z computational basis, this means that all Hamiltonian terms composed entirely of σ_z operators can be evaluated. But if a term in the Hamiltonian contains σ_x or σ_y, it cannot be handled in this basis. This means that we must analyze the Hamiltonian and partition its terms into several classes. All the terms in one class should be measurable in one computational basis. To determine the eigenvalue of H_{sg} we will then iterate over these classes and for each class, we will compute the expectation value of the terms that are diagonalized in that class.

While our Hamiltonian does combine Pauli operators from different bases, it does so in a particularly simple form. Each term of H_{sg} can be handled in either a Z-basis, an X-basis or a Y-basis. This is because each term in H_{sg} is composed entirely of Pauli operators of the same type (X, Y or Z). There are no combined terms like XY. The fact that H_{sg} has this simple form means that we do not need to worry about "mixed" bases.

Measuring in the Z-computational basis requires no extra work. By convention we always work in the Z computational basis, and so we can use our circuit as developed above without any further additions to measure all the Z terms in our Hamiltonian. We simply multiply the diagonal values of the Z-terms by the probability of measuring the corresponding basis state and sum over basis states. To measure the X-type terms in H_{sg}, we must convert all our qubits into the X-computational basis. Finally, to measure the Y-type terms, we must convert all qubits into the Y-computational basis. Thus, we must measure three contributions to the energy in H_{sg} corresponding to these three different computational bases.

Having seen that the Z-terms can be handled without further extensions to the original circuit, let us turn our attention to the X-type terms. We need identities that re-express $\sigma_x \otimes \sigma_x$ terms in the Hamiltonian in terms of σ_z. Remembering that the Hadamard gate simply interchanges the X and Z axes in the Bloch sphere, we note that:

$$\sigma_x \otimes \sigma_x = (H \otimes H)(\sigma_z \otimes \sigma_z)(H \otimes H)$$

Where H represents the Hadamard gate. This identity tells us that we can apply Hadamard gates to all qubits at the end of the circuit and thus convert the XX terms in the Hamiltonian to ZZ terms. Thus, to measure in the computational X basis, we append to the original circuit a layer of Hadamard gates at the end and repeat the process above for the X-type terms in the Hamiltonian. This addresses the second of the three computational bases.

The last set of terms in H_{sg} are Y-type terms. To convert YY terms to ZZ terms which are accessible to measurement, note that we can rotate the Bloch sphere around the Z

axis through an angle of $-\pi/2$ via an RZ gate to convert the Y axis into the X axis. Then we can apply the transformation used for the X-type terms. These two combined operations allow us to measure the Y-type terms.

With all these building blocks in place, you should be able to verify the energy computation for the ground state of H_{sg} that was carried out on its 16×16 matrix in part 2.

Exercises for Part 3: [hard - ☆]

3.a) Build a numerical program as outlined above, which contains arbitrary angle parameters and qubit pairs for XY-mixers. Experiment with the XY-mixers to obtain angle parameters and qubit pairs that rotate the final state so that its probabilities match the ground state probabilities of H_{sg}. This problem can be solved using three XY-mixers.

3.b) Convert the numerical program into a quantum circuit. Verify that the output of this quantum circuit generates the correct probabilities for the ground state of H_{sg}.

3.c) Check the phases of the non-zero components of the quantum circuit created in 3.b. Are they all real? If so, you can skip this step. Otherwise, find circuit elements that you can add to the end of your circuit so that all non-zero computational basis states have the same phase.

3.d) Use your quantum circuit to measure the expectation value of the ground state of H_{sg} . You will have to repeat your circuit three times - once to measure in each of the X, Y and Z computational bases. Each term of H_{sg} can be measured in one of the three computational bases. Sum the contributions from the three bases and confirm that the summed energy is equal to the ground state eigenvalue.