Questions:

Q:What is a sparse matrix and what features should it possess?

A sparse matrix is a matrix populated primarily with zeros. For example, in a nxn matrix the number of elements, i, with non-zero values is much less than the total number of elements such that i << n^2. It's unique nature allows the use of specialized algorithms and data structures to solve the linear system more efficiently in both time and space when compared to non-sparse matrices.

Q: What sources of information should you consult to derive features? Do analogies exist? If so with what?

Potential sources of information we can consult for features are existing linear system solution products. These products include Wolfram Alpha, Matlab, and others.

Sparse Matrices are analogous to loosely coupled systems. For example a system that includes a line of balls connected by springs from one to the next is a loosely coupled system, as they have a small number of connections.

Q: Who is likely to be the user of a sparse matrix package? What features are they likely to demand?

Professionals who require a package to assist in solving partial differential equations (ex: Scientists and Engineers). Desired features may include:

- Method to solve for the variables of the input linear system
- Ability to add values to an existing matrix

Q: What is a tridiagonal matrix?
A matrix that has nonzero elements only in the main diagonal, the first diagonal below the main diagonal, and the first diagonal above the main diagonal

Q: What is the relationship between a tridiagonal matrix and a generic sparse matrix?
A tridiagonal matrix is a type of sparse matrix. Since non-zero values exist for only the middle three diagonals, the majority of the matrix is still populated by zeroes.

Q: Are tridiagonal matrices important? And should they impact your design? If so, how?
Tridiagonal matrices are incredibly important as they can be solved with much greater time efficiency when compared to regular sparse matrices. It is important for our design to be able to identify tridiagonal matrices in order to know when to apply a more specific set of algorithms to derive a solution more efficiently.

Q: What is a good data representation for a sparse matrix?

Dictionary of values keyed on location (x and y value in matrix).

Q: Assume that you have a customer for your sparse matrix package. The customer states that their primary requirements as: for a N x N matrix with m non-zero entries

o Storage should be ~O(km), where k << N and m is any arbitrary type defined in your design.

o Adding the m+1 value into the matrix should have an execution time of ~O(p) where the execution time of all method calls in standard Ruby container classes is considered to have a unit value and p << m ideally p = 1

In this scenario, what is a good data representation for a sparse matrix?

A dictionary of values keyed on the matrix location (as mentioned in the previous question) would sufficiently meet these requirements. Storage would be O(m) where m is the number of non-zero values. Insertion into the dictionary would be on the order of O(1).

Q: Explain the design patterns: Delegate and Abstract Factory

Delegate design pattern involves an object creating a new "helper" object to handle a task, delegating the task to the helper object. Abstract factory provides a way to encapsulate a group of individual factories.

Q: Explain how you would approach implementing these two patterns in Ruby

Delegate pattern:

- Delegates will be created as subclasses of a base class for an action
- When the program needs to delegate the task, it creates the delegate class

Abstract Factory:

- All factory classes will include a base abstract class for abstract factories.
- Each factory class corresponds to a subclass in the model
- The model subclasses have inheritance with its base class
- To create an object, simply use the respective factory.

Q: Are these patterns applicable to this problem? Explain your answer! (HINT: The answer is yes)

Yes these patterns are applicable to our problem. We can delegate the solving of the matrices to delegate classes, and we can construct them using the abstract factory class. This way, we can design our solver algorithms independently, using the same sparse matrix data structure.

Q: What implementation approach are you using (reuse class, modify class, inherit from class, compose with class, build new standalone class); justify your selection.

- We will build a standalone class for the data representation of a sparse matrix
- We will compose multiple linear equation solvers (for different solving algorithms) for a sparse matrix that inherit from a base solver class
- We will implement multiple factory classes that inherit from a base factory class to construct all our solver classes.

Q: Is iteration a good technique for sparse matrix manipulation? Is "custom" iteration required for this problem?

Iteration is a critical technique for sparse matrix manipulation. In order to transform a matrix to derive its solution, the matrix must be iterated over. Custom iteration will be required for this problem. Since a sparse matrix is populated mostly by zeroes, a custom iterator is required that iterates over only the non-zero values of the matrix as opposed to every value of the matrix.

Q: What exceptions can occur during the processing of sparse matrices? And how should the system handle them?

Potential exceptions that could occur include:

1. TypeError- The user may have made invalid entries into the matrix (ex: a non-number such as 'a'). The system should prevent this from occurring by checking user input as the sparse matrix is created. If an invalid entry is detected, the user should be notified immediately.
2. IndexError - When solving the sparse matrix the program may attempt to access an element that is outside the bounds of the structure used to store the data.
3. ZeroDivisionError - When solving the sparse matrix the program may incorrectly attempt to divide one element by another element whose value is 0.

Q: What information does the system require to create a sparse matrix object? Remember you are building for a set of unknown customers – what will they want?

The system will require a set of equations that represent the linear system. These equations will be represented by a pre-existing matrix that can be converted to a sparse matrix object. Since the unknown customers will already be using Ruby, it is likely that the customers are already using Ruby's native Matrix class for their system. Therefore, the customer would likely want an easy way to convert a Matrix object into a Sparse Matrix object in order to efficiently derive a solution when necessary. This can be done by having the Sparse Matrix class constructor require a Matrix object as an argument.

Q: What are the important quality characteristics of a sparse matrix package? Reusability? Efficiency? Efficiency of what?

- Reliability -- solutions must be correct
- Reusability -- users can implement their own solving algorithms easily
- Efficiency with respect to time of operations, and memory usage (storage)
- Maintainability -- minimal inter-class dependencies
- Usability -- users should not have to relearn the software to use a new algorithm

Q: How do we generalize 2-D matrices to n-D matrices, where n > 2 – um, sounds like an extensible design?

For solving systems of linear equations with n-dimensional matrices, we would break down the n-d matrix into multiple 2-d matrices that can then be solved. Then the same techniques can be used to solve each 2-d sub-matrix.