**Assignment 3**

a) Implement a function GaussSolve that solves a linear system of equations Ax=b for a square matrix A using forward elimination, backward substitution, **and** partial pivoting as shown during class.

The function should accept two inputs (A and b) and return one output (x). The function also needs to include error handling to

* ●  check whether the matrix A is square or not
* ●  whether the dimensions of b and A fit
* ●  whether during the forward elimination step any of the leading coefficients after partial pivoting become 0

If any conditions become critical, then the function should abort, telling the user the reason for it.

The assignment is asking you to implement a function called GaussSolve that solves a linear system of equations of the form Ax = b for a square matrix A. The function should accept two inputs, A and b, and return one output, x, which is the solution vector to the system of equations.

The function should use the following algorithm to solve the system of equations:

1. Perform forward elimination with partial pivoting. This involves eliminating the lower-triangular part of the matrix A by subtracting appropriate multiples of the rows above it. Partial pivoting is a technique that ensures that the leading coefficient of each row is non-zero.
2. Perform backward substitution. This involves solving the upper-triangular system of equations that results from the forward elimination step.

The function should also include error handling to check for the following conditions:

* Whether the matrix A is square.
* Whether the dimensions of A and b match.
* Whether any of the leading coefficients after partial pivoting become zero.

If any of these conditions are met, the function should abort and tell the user the reason for it.

Here is a really simple explanation of the GaussSolve code, as if I were talking with a child:

Imagine you have a system of equations that looks like this:

ax + by = c

dx + ey = f

You want to solve this system of equations to find the values of x and y.

The GaussSolve function can help you do this. It works by breaking the system of equations down into smaller pieces, and then solving each piece one at a time.

The first thing the GaussSolve function does is check to make sure that the matrix A is square. This means that the matrix must have the same number of rows as columns. If the matrix is not square, then the GaussSolve function will raise an error.

Next, the GaussSolve function checks to make sure that the dimensions of A and b fit. This means that the number of rows in A must be the same as the number of rows in b. If the dimensions do not fit, then the GaussSolve function will raise an error.

Once the GaussSolve function has checked the inputs, it begins to solve the system of equations. It does this by performing a process called forward elimination.

Forward elimination works by eliminating the lower triangular part of the matrix A. This is done by subtracting appropriate multiples of the rows above a given row from that row.

Once the lower triangular part of the matrix A has been eliminated, the GaussSolve function can perform backward substitution.

Backward substitution works by solving the upper triangular system of equations that results from the forward elimination step.

Once the GaussSolve function has performed backward substitution, it has solved the system of equations and found the values of x and y.

Here is a simple example of how to use the GaussSolve function:

Python

import numpy as np

A = np.array([[2, 1, -1],

[-3, -1, 2],

[-2, 1, 2]])

b = np.array([8, -11, -3])

x = GaussSolve(A, b)

print(x)

Certainly, let's break down the code step by step in detail:

1. Import Required Libraries:

- The code begins by importing the necessary libraries. In this case, it imports NumPy to handle matrix operations.

2. Define the `GaussSolve` Function:

- The `GaussSolve` function is defined to solve a system of linear equations using the Gaussian Elimination method with partial pivoting.

- The function accepts two arguments: `A` (the coefficient matrix) and `b` (the right-hand side vector).

3. Check for Square Matrix:

- The code checks whether the matrix `A` is square (i.e., it has an equal number of rows and columns). If `A` is not square, it raises a `ValueError` with the message "Matrix A is not square."

4. Check for Dimension Compatibility:

- It checks if the dimensions of `A` and `b` are compatible for matrix multiplication. If the number of columns in `A` doesn't match the length of `b`, it raises a `ValueError` with the message "Dimensions of A and b do not match."

5. Initialize Variables:

- It initializes `n` to the number of rows in matrix `A`.

6. Data Type Fixation:

- It ensures that both matrix `A` and vector `b` have a common data type by explicitly casting them to `float`. This is done to avoid data type mismatches during operations.

7. Augmenting the Matrix:

- The code augments matrix `A` with vector `b` to create an augmented matrix `[A|b]` that will be used for Gaussian Elimination.

8. Partial Pivoting and Forward Elimination:

- The code performs partial pivoting to avoid division by zero and reduce round-off errors. It iterates through the rows and columns to find the maximum absolute value in each column, and if the maximum is not zero, it swaps the current row with the pivot row. If the maximum is zero, it raises an error.

- During the forward elimination, it reduces the augmented matrix to upper triangular form. It iterates through the rows and subtracts appropriate multiples of rows to create zeros below the diagonal.

9. Backward Substitution:

- After the forward elimination, the code performs backward substitution to find the solution vector `x`. It starts from the last row of the upper triangular matrix and computes the values of `x` in reverse order.

10. Return the Solution Vector:

- The function returns the solution vector `x`.

11. Example Usage:

- The code includes an example usage of the `GaussSolve` function. It creates a sample coefficient matrix `A` and right-hand side vector `b` for a system of linear equations.

12. Try-Except Block for Error Handling:

- It wraps the call to `GaussSolve` in a try-except block to handle any exceptions (ValueError) that may be raised during the execution.

- If the solution is found successfully, it prints the solution vector `x`. If any of the error conditions are met, it prints an error message.

This code provides a comprehensive implementation of the Gaussian Elimination method for solving linear systems, along with error handling and data type fixations to ensure the integrity of the calculations.