Advanced Programming - Exam 17 Feb 2025 - Part 2

Objective

Implement a C++ program that solves a system of linear equations using **Jacobi** and **Gauss-Seidel** iterative methods. The program should use **inheritance, polymorphism, and templates** to make the solver flexible for different numerical types.

Mathematical description

System of equations

Consider a system of linear equations:

$$A\mathbf{x} = \mathbf{b}$$

where:

- A is an $n \times n$ coefficient matrix,
- $\mathbf{x}$ is the unknown solution vector,
- $\mathbf{b}$ is the right-hand side vector.

We assume A is **diagonally dominant** or **symmetric positive definite** to guarantee convergence.

Jacobi method

The **Jacobi method** updates all variables simultaneously using values from the previous iteration:

1. Start with an initial guess $\mathbf{x}^{(0)}$.
2. Iterate until convergence:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right)$$

3. Stop when $||\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}|| <$ tolerance.

Gauss-Seidel method

The **Gauss-Seidel method** updates variables sequentially, using newly computed values immediately:

1. Start with an initial guess $\mathbf{x}^{(0)}$.
2. Iterate until convergence:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right)$$

3. Stop when $||\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}|| <$ tolerance.

Exercise instructions

Overview

Your goal is to implement the Jacobi and Gauss-Seidel iterative methods in C++ while applying object-oriented programming concepts, including:

- **Inheritance**: Create a base class `LinearSolver` and derive `JacobiSolver` and `GaussSeidelSolver` from it.
- **Polymorphism**: Use virtual functions to allow dynamic method dispatching.
- **Templates**: Ensure the solver works with different floating-point types (`float`, `double`, etc.).

We consider the linear system:

$$A\mathbf{x} = \mathbf{b}$$

where

$$A = \begin{bmatrix} 4 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 15 \\ 10 \\ 10 \\ 10 \end{bmatrix}.$$

This system is diagonally dominant, ensuring convergence for both the Jacobi and Gauss-Seidel methods, and has exact solution $\mathbf{x}_{\text{ex}} = [5, 5, 5, 5]^T$.

Tasks

1. (1 point) **Define a base class `IterativeSolver`**
    - Create a pure virtual function `solve()` that will be implemented in derived classes.

- Provide methods to set the matrix and the right-hand side vector and to get the number of iterations performed.

2. (2 points) **Implement derived classes `JacobiSolver` and `GaussSeidelSolver`**
   - Implement the `solve()` function using the **Jacobi** and **Gauss-Seidel** methods, respectively.
   - Ensure they override the base class function correctly.
3. (1 point) **Use polymorphism**
   - In the `main()` function, define a pointer/reference to the base class and allow dynamic method dispatching to use either solver.
4. (2 points) **Test your implementation**
   - Read the system of equations from a user-defined matrix A and vector $\mathbf{b}$.
   - Solve the system using both methods to verify correctness and compare results.
5. (2 points) **Templatize the solvers**
   - In a separate file or using different class names, templatize your code to handle different floating-point types (`float` and `double`).
6. (2 points) **Configuration and compilation**
   - Develop a CMake script for easy compilation of the C++ library.
   - Provide clear instructions on compiling the library.
7. (5 points) **Python bindings using pybind11**
   - Bind the C++ functions, classes and their methods to Python, properly handling exceptions.
   - Ensure the Python interface is user-friendly and adheres to Python conventions.
   - Write a Python script to demonstrate the usage of the classes.
   - Compare your implementation against the solvers provided by SciPy.

Evaluation criteria

- Code organization and correctness.
- Correct use of inheritance and polymorphism to model the problem.
- Effective use of templates for flexibility and type safety.
- Proper memory management and exception handling.
- Intuitive user interface with clear instructions on how to interact with the music database.
- Seamless integration between C++ and Python.