The tiny Project

Recommendations:

- Students can work in groups (up to 6 members per group) as well as work individually.
- Students can use Gitlab as management tool.

Part A: You are required to build the ***Vector*** class and ***Matrix*** class. These classes are then combined into a ***LinearSystem*** class (or alternative classes derived from it) which has methods for solving systems of the form **Ax** = **b** for **x**. You are required to define files *.h or .hpp* and *.cpp* for Vector, Matrix and LinearSystem classes.

1) You are required to develop a class of matrices called ***Vector***. It will include constructors and destructors that handle memory management. It will overload the assignment, unary and binary operators to allow addition, subtraction, and multiplication of vectors and scalars. The square bracket operator will be overloaded for the vector class to provide a check that the index of the array lies within the correct range, and the round bracket operator will be overloaded to allow the entries of the vector to be accessed, indexing from 1 rather than from zero. The class should have private members *mSize* (the size of the array) and *mData* that is a pointer to a data element of array.
2) You are required develop a class of matrices called ***Matrix***. It should include the features listed below. Your class should have private members *mNumRows* and *mNumCols* that are integers and store the number of rows and columns, and *mData* that is a pointer to a pointer to a data element of matrix, which stores the address of the pointer to the first entry of the first row:
   - An overridden copy constructor that copies the variables *mNumRows* and *mNumCols*, allocates memory for a new matrix, and copies the entries of the original matrix into the new matrix.
   - A constructor that accepts two positive integers *numRows* and *numCols* as input, assigns these values to the class members *mNumRows* and *mNumCols*, allocates memory for a matrix of size *mNumRows* by *mNumCols*, and initialises all entries to zero.
   - An overridden destructor that frees the memory that has been allocated to the matrix.
   - Public methods for accessing the number of rows, and the number of columns.
   - An overloaded round bracket operator with one-based indexing for accessing the entries of the matrix so that, provided i and j are valid indices for the matrix, *A(i,j)* may be used to access *mData[i-1][j-1]*.
   - Overloaded assignment, unary and binary operators to allow addition, subtraction and multiplication of suitably sized matrices, vectors and scalars. You should use assert statements to ensure the matrices and vectors are of the correct size.
   - A public method that computes the determinant of a given square matrix.
   - A public method that computes the inverse of a given square matrix.
   - A public method that computes the pseudo-inverse (Moore-Penrose inverse) of a given matrix.
3) You are required to develop a class called ***LinearSystem*** that may be used to solve linear systems. Assuming the system is nonsingular, a linear system is defined by the size of the linear system, a square matrix, and vector (representing the right-hand side), with the matrix and vector being of compatible sizes. The data associated with this class may be specified

through an integer variable *mSize*, a pointer to a matrix *mpA*, and a pointer to the vector on the right-hand side of the linear system *mpb*. We suggest only allowing the user to set up a linear system using a constructor that requires specification of the matrix and vector: the member *mSize* may then be determined from these two members. If you do not wish to provide a copy constructor, then the automatically generated copy constructor should be overridden and made private to prevent its use. As with the class of vectors, we recommend that use of the automatically generated default constructor is prevented by providing a specialised constructor but no default constructor. A public method Solve should be written to solve this linear system by Gaussian elimination with pivoting. This method should return a vector that contains the solution of the linear system.

Derive a class called *PosSymLinSystem* (or similar) from the class *LinearSystem* that may be used for the solution of positive definite symmetric linear systems. Make the method *Solve* a virtual method of the class *LinearSystem*, and override this method in the class *PosSymLinSystem* so that it uses the conjugate gradient method for solving linear systems. If you declared *LinearSystem* member data as private, then this should now be declared protected. Your class *PosSymLinSystem* should perform a check that the matrix used is symmetric: testing that the matrix is positive definite would be rather difficult and so we don't suggest performing a check for this property. Test your class using suitable examples.

4) You will develop the solutions for under-determined or over-determined linear system, where the matrix **A** is not square. The solutions can use the Moore-Penrose inverse and/or Tikhonov regularization to deal with the Ill-posed problem.

Part B: You will develop a linear regression prediction of relative cpu performance. Dataset could be downloaded from UCI: https://archive.ics.uci.edu/ml/datasets/Computer%2BHardware. The number of instances is 209. Every instance has 10 features (6 predictive attributes, 2 non-predictive, 1 goal field, and the linear regression's guess):

1. vendor name
2. Model Name: many unique symbols
3. MYCT: machine cycle time in nanoseconds (integer)
4. MMIN: minimum main memory in kilobytes (integer)
5. MMAX: maximum main memory in kilobytes (integer)
6. CACH: cache memory in kilobytes (integer)
7. CHMIN: minimum channels in units (integer)
8. CHMAX: maximum channels in units (integer)
9. PRP: published relative performance (integer)
10. ERP: estimated relative performance from the original article (integer)

We assume that there is a linear regression model for determining the relative performance as

$$PRP=x_1*MYCT+ x_2*MMIN+ x_3*MMAX+ x_4*CACH+ x_5*CHMIN+ x_6*CHMAX, \quad (1)$$

where x_i's are parameters of model. You utilize the above methods for solving systems to find the parameters of (1). The dataset is divided into two sets, training set (80%) is used to determine model parameters and testing set (20%) is used for evaluation. The evaluation can use the criterion of root mean square errors (RMSEs).