Tiny Project Report

Doan Le Gia Bao
Student ID: 10423008
Course: Programming 2

1 Introduction

This project is divided into two main parts:

- **Part A** focuses on building classes in C++, including `Vector`, `Matrix`, `LinearSystem`, and `GeneralLinearSystem` classes, with support for solving:
  - Square systems
  - Underdetermined systems
  - Overdetermined systems
- **Part B** applies the developed tools to a **real-world machine learning task**: predicting **CPU performance** using **linear regression** on the **UCI Computer Hardware dataset**.

2 Part A – Building Classes

2.1 Vector Class

Implements **1D dynamic arrays** with custom memory management.

- **Constructors and Destructor:**
  - Default, parameterized, and copy constructors
  - Destructor
- **Overloaded Operators:**
  - Assignment (`=`), negation (`-`), addition (`+`), subtraction (`-`)
  - Scalar multiplication (`*`) and dot product ($\cdot$)
  - Indexing with both `[]` and `()` (1-based)
- **Additional Methods:**
  - Get the size of vector (`size()`)
  - Print all elements of vector (`print()`)
- Designed for use as vectors of unknowns and right-hand sides in linear systems.

2.2 Matrix Class

2D dynamic matrix with safe memory management.

- **Constructors and Destructor:**
  - Default, parameterized, and copy constructors
  - Destructor
- **Supports:**
  - Matrix addition (`+`), subtraction (`-`), multiplication (`*`) of suitably sized matrices, vectors, and scalars
  - Transpose (`transpose()`), determinant (`determinant()`), inverse (`inverse()`), Moore-Penrose pseudoinverse (`pseudoInverse()`)
  - Assert-based dimension checks for safety
- **Additional Methods:**
  - Get the number of rows (`getNumRows()`)
  - Get the number of columns (`getNumCols()`)
  - Print all elements of matrix (`print()`)

2.3 LinearSystem and PosSymLinSystem

- **LinearSystem**:
  - Solves square systems $\mathbf{Ax} = \mathbf{b}$ using Gaussian elimination with partial pivoting.
- **PosSymLinSystem**:
  - Inherits from `LinearSystem`, uses Conjugate Gradient method for solving symmetric positive-definite systems.
  - Automatically checks symmetry.

2.4 GeneralLinearSystem

Solves non-square systems:

- **Overdetermined** $\rightarrow$ Least-squares solution using Moore-Penrose or Tikhonov regularization.
- **Underdetermined** $\rightarrow$ Least-squares solution using Moore-Penrose or Tikhonov regularization.

Forms the computational core for Part B's regression task.

3 Part B – Linear Regression on CPU Performance

3.1 Objective

Build a **linear regression model** for predicting **PRP** (Published Relative Performance):

$$\text{PRP} = x_1 \cdot \text{MYCT} + x_2 \cdot \text{MMIN} + x_3 \cdot \text{MMAX} + x_4 \cdot \text{CACH} + x_5 \cdot \text{CHMIN} + x_6 \cdot \text{CHMAX}$$

3.2 Dataset

- **Source:** UCI Computer Hardware Dataset
- 209 instances with **10 attributes** (6 features used for modeling).

3.3 Methodology

1. **Preprocessed dataset** to extract relevant numeric fields.
2. **Randomized 80/20 train-test split** using C++ `shuffle`.
3. Used `GeneralLinearSystem` to solve for weights via **Moore-Penrose pseudoinverse**.
4. Evaluated predictions using **Root Mean Square Error (RMSE)**.

3.4 Sample Output

Learned model coefficients:

Coefficient	Value
x_1	-0.0374181
x_2	0.0135913
x_3	0.00476773
x_4	0.565409
x_5	0.505904
x_6	1.27356

Root mean square error: 41.8398

4 Conclusion

This project provided **hands-on experience** with:

- Core **linear algebra techniques**
- **Machine learning** applications
- **Numerical stability** and optimization

4.1 Final Thoughts

This project bridges **theory & practice**, demonstrating how **linear algebra powers machine learning**. Future work could extend this to **deep learning optimizations**.

Appendix

- **Code Repository:** `tinyProject`