Assignment 01
C++ Basics

Khiem Nguyen Lecturer in Multiscale Materials khiem.nguyen@glasgow.ac.uk

Foreword

Story behind this assignment

You heard stories from your parents like this – click on me!!! I was much luckier than that; I could learn Pascal in highschool, and then C++ at 18. Nevertheless, ...

- I did not have GenAI chatbots like ChatGPT, Gemini, CoPilot. In fact, *stackoverflow* (created by Jeff Atwood and Joel Spolsky in 2008) was not even a thing at that time
- The 100% course grade was entirely decided by one single paper exam. If we were lucky, roughly 30% would be assigned to a team project.

Yes, you did not read it wrong. It was not long time ago. We completed the course by taking one single paper exam in which we must literally write C++ code on paper. Henceforth, the only one to prepare for the exam is to practice as much as possible to get ready for one single shot. There are both advantages and disadvantages:

- **Advantage**: You have to learn basic details with little instructions. You cannot get away with good grades/passing grades with the so-called sufficient knowledge.
- **Disadvantage**: Learning curve is steep and it took sometimes too much time to finish a simple task.

If you still don't belive me, read the file `programming_exams.html`.

POV of a lecturer

First of all, I have taught several courses involving heavy programming course work since 2017. I notice, actually know for sure, from my own teaching and marking experience within the last two years that I have marked ChatGPT's submissions more than students' submissions.

> **That's fine, it's your future, not mine** .

At one point, I know that I cannot stop you to use ChatGPT – So I joined the force since 2025 (just roughly half a year). No matter how many exercises I assigned, one could easily solve all of them in less than 1 hour, without understanding a single line of code. Indeed, I copied/pasted my entire question in ChatGPT or Gemini and got the solution in return without any single error. Here are my own stories:

- As for ENG3091 last year, several students got less than 5/35 points for the coding question and less than 3/15 for questions about C++ programming. Several students got points because they wrote something (I gave 1 point for making attempt), not because they answered the questions correctly.
- If I was not lenient, several students actually got completely 0/50 for the C++ coding part. Some students wrote literall English in coding questions. Ironically, they finished the mid-term assignments with all A-band grades.
- In the Teaching Committee, a few professors stated that for the first time of their teaching experience, they marked 0/100.
- Indeed, last semester a few students just confessed upfront that they used ChatGPT to solve the assignments and asked for the passing grades instead of trying to answer my questions.

Rationale on the length of the assignment

Why is the assignment so long?

Here is my experience. If I give one single assignment with a significantly meaningful topic, you will skip all the practice opportunities and finally end up using ChatGPT to solve the final exam – Again, I saw and marked more than AI-geneted code than you could imagine.

The first part of the assignment gives you the opportunities to practice writing code in C++. The second part of the assignment introduce you to the simplest knowledge in the machine learning world: Linear Regression.

My advices

- I encourage you to try to solve the exercises using just **Google**, **stackoverflow**, before trying to use ChatGPT, Gemini or CoPilot.
- If you decide to use GenAI Chatbots, you should try to refrain yourself from copy/paste the solution in a bruteforce manner. Instead, you should ask them for knowledge you want to learn.
- **Good habits compound, so does bad habits.**

Task 1

In the first task, you need to write C++ programs to solve **two out of three** problems you have solved in Python.

Problem	Points
Fixed-point and Newton-Raphson methods for solving nonlinear equation	15/100
Rouge-like room and moving the player in the room	15/100
The tic-tac-toe game	15/100

Python is not the focus of our course. The problems and their solution in Python code are provided.

- Essentially, you learn to translate Python code into C++ code with the knowledge you will learn in this course.
- You can use object-oriented programming (OOP) if you want. However, it won't give extra points by using OOP.
- You won't have extra points for solving all three problems.
- You need to write **significantly meaningful comments** in your code.
- The deadline for this assignment is pretty far. So don't worry too much.
- Submit the source code. Explain how to compile the source code to obtain the executable files.

Feel free to use GenAI chatbots to learn anything you need.

Task 2

In this task, we will build a linear regression model, which is the first step of building a complex neural network.

Problem description

You can read this wikipedia: Linear Regression for multi-dimensional linear regression. But my description will use different notations.

We are given a task of predicting the house price in Glasgow by using various properties of a house such as number of bedrooms, total living area, distance to schools and city center and so on. That is, we need to build a prediction model that can be represented by a mapping

$$y = f(x_1, x_2, ..., x_n), \quad y = \text{house price}$$

and $x_1, x_2, \ldots x_n$ correspond to the features of the house. For example, x_1 is the number of bedrooms, x_2 is the total iving area, x_3 is the distance to the city center and so on. That is we seek the function f. The prediction model estimates the relationship between a scalar response (dependent variable y).

Simple linear regression

Let us simplify the above problem to prediction of the house price based on the total area of the house. That is, the vector $\mathbf{x}$ becomes a scalar x. We go around the city of Glasgow and collect the dataset $\mathcal{D}$ of m data points:

$$\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots (x^{(m)}, y^{(m)})\}.$$

The simplest prediction model we can build is the simple linear regression. In statistics, simple linear regression is a linear regression model with a single independent variable. Consider the model function

$$(M): \qquad y = f_{w,b}(x) = wx + b$$

which describes a line with the slope w and y-intercept value b. We wan to look for w and b so that the line **fits** the dataset $\mathcal{D}$ as much as possible.

Building a linear regression model

Let us assume that we have only 2 data points. Obviously, the best line to fit two data points is the line going through both data points. However, when we have more than 2 data points and they do not lie on one single line, we have to find a way to measure the **fitness** of the model.

Using the model (M), the absolute error between the prediction $\hat{y}^{(1)} = f_{w,b}(x^{(1)})$ and the observed value $y^{(1)}$ is

$$|\hat{y}^{(i)} - y^{(i)}| = |f_{w,b}(x^{(i)}) - y^{(i)}|$$

The average error of all the absolute errors between the prediction of the model at the data input $x^{(i)}$ and the data output $y^{(i)}$ is given by

$$Q(m, w) = \frac{1}{m} \sum_{i=1}^{m} |\hat{y}^{(i)} - y^{(i)}| = \frac{1}{m} \sum_{i=1}^{m} |wx^{(i)} + b - y^{(i)}|.$$

We see that this average error Q is just a function of the parameters w and b, because $x^{(i)}$ and $y^{(i)}$ are given from the dataset. To find the best fit of the model, we only need to minimize the the average error $Q(w, b)$:

$$\min_{w,b} Q(w, b) \quad \longrightarrow w, b$$

However, it turns out that minimizing the function $Q(w, b)$ is not convenient because it involves absolutes of multiple values. To overcome this issue, we can consider the squared error

$$\varepsilon^2 = (\hat{y}^{(i)} - y^{(i)})^2 = (wx^{(i)} + b - y^{(i)})^2$$

and consider the least squared error function $\mathcal{L}$ instead of the function Q as follows

$$\mathcal{L}(w, b) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2 = \frac{1}{2m} \sum_{i=1}^{m} (wx^{(i)} + b - y^{(i)})^2 \tag{1}$$

Function $\mathcal{L}$ is called **loss function** in the optimization theory and machine learning community. By minimizing the loss function $\mathcal{L}$ with respect to two parameters w and b, we obtain the linear regression model that best fit the dataset $\mathcal{D}$:

$$\min_{w,b} \mathcal{L}(w, b) \quad \longrightarrow w, b$$

Computation of the model parameters

No matter how many data points $\mathcal{D}$ has, the loss function $\mathcal{L}(w, b)$ is obviously a quadratic function of w and b. If you don't believe me, try to compute $\mathcal{L}(w, b)$ with 2 data points, and with 3 data poitns and then with 4 data points to convince yourself. For this reason, the minimization of $\mathcal{L}$ can be easily computed and there is a unique optimal solution (w^*, b^*) to minimize $\mathcal{L}(w, b)$.

A – Normal equation

Indeed, from basic mathematics , you should have learned that the optimal solution (w^*, b^*) is obtained by solving the the system of equations

$$\nabla \mathcal{L}(w, b) = 0 \quad \Leftrightarrow \quad \begin{cases} \dfrac{\partial \mathcal{L}}{\partial w} = 0 \\[2mm] \dfrac{\partial \mathcal{L}}{\partial b} = 0 \end{cases}$$

By carrying out some algebraic computation, we end up with the solution[1]:

$$w^* = \frac{\sum\limits_{i=1}^{m} (x^{(i)} - \overline{x})(y^{(i)} - \overline{y})}{\sum\limits_{i=1}^{m} (x^{(i)} - \overline{x})^2},$$

$$b^* = \overline{y} - w^* \cdot \overline{x}, \tag{2}$$

where $\overline{x}$ and $\overline{y}$ are the average of the $x^{(i)}$ and $y^{(i)}$, respectively:

$$\overline{x} = \frac{1}{m} \sum_{i=1} x^{(i)}, \qquad \overline{y} = \frac{1}{m} \sum_{i=1} y^{(i)}$$

The above result can be expanded to the following expression:

$$w^* = \frac{m \sum_{i=1}^{m} x^{(i)} y^{(i)} - \sum_{i=1}^{m} x^{(i)} \sum_{i=1}^{m} y^{(i)}}{m \sum_{i=1}^{m} (x^{(i)})^2 - \left(\sum_{i=1}^{m} x^{(i)}\right)^2}$$

$$b^* = \frac{\sum_{i=1}^{m} y^{(i)} \sum_{i=1}^{m} (x^{(i)})^2 - \sum_{i=1}^{m} x^{(i)} \sum_{i=1}^{m} x^{(i)} y^{(i)}}{m \sum_{i=1}^{m} (x^{(i)})^2 - \left(\sum_{i=1}^{m} x^{(i)}\right)^2} \tag{3}$$

The solution formulas given by Equation 2 and Equation 3 are equivalent. Either of them is called **Normal Equation**.

B – Gradient descent method

To prepare for the future assignment 02, we shall learn gradient descent method to minimize the loss function given by Equation 1 – Wikipedia: Gradient Descent

You will learn later that the classification problem using the logistic regression method cannot be solved easily by deriving an analytical set of equations. There is no single "Normal Equation" for logistic regression.

In the world of machine learning, **Gradient Descent** is the primary algorithm used to "train" models by minimizing error. This is an iterative approach to find the best possible parameters.

---

[1]You can try to prove this by yourself but it is not the core of this assignment.

The intuition: Hiking in the Fog

Imagine you are standing on a mountain in a thick fog. You want to reach the very bottom of the valley, but you cannot see where it is. To get there, you might:

1. Feel the slope of the ground under your feet.
2. Take a small step in the direction where the ground slopes *downward* most steeply.
3. Repeat this process until the ground feels flat under your feet.

In this analogy:

- The Mountain is the Cost Function (the measure of how much error your model is making).
- The Bottom of the Valley is the point where the error is at its minimum.
- The **Steepness** is the **Gradient** (the derivative of the cost function).

Gradient descent formula

Assume that we have a function of multiple parameters (variables) $J = J(\theta_1, \ldots, \theta_n)$. Note that we use the term "parameters" to reflect our linear regression model. What we explain here is valid for a function of multiple variables. Our aim is to minimize J with respect to the parameters Θ using an interative process. To update a set of parameters $\Theta = (\theta_1, \ldots, \theta_n)$, we use the following update rule:

$$\Theta_{\text{new}} = \Theta_{\text{old}} - \eta \cdot \nabla J(\Theta_{\text{old}}). \tag{4}$$

This update will be repeated until

- the value of $J(\Theta)$ does not change much, or
- all the components of the gradient $\nabla J(\Theta)$ are almost equal to 0.

Here is what each part represents:

- Θ – The parameters we are trying to optimize.
- η – The **learning rate**.
  This value determines how big of a step we take in each iteration.
- $\nabla J(\Theta)$ – The gradient.
  This is the derivative of the cost function J with respect to Θ. It tells us the slope at our current position.
- The **minus sign** $(-)$ is crucial.
  It ensures we move *opposite* the gradient (downhill) rather than with it (uphill).

As for our case of linear regression model, the vector of parameters is $\Theta = (w, b)$ and the function to be optimized is the loss function $\mathcal{L}(w, b)$. The gradient descent rule Equation 4 for our case becomes

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{\partial \mathcal{L}}{\partial w}(w_{\text{old}}, b_{\text{old}})$$

$$b_{\text{new}} = b_{\text{old}} - \eta \cdot \frac{\partial \mathcal{L}}{\partial b}(w_{\text{old}}, b_{\text{old}})$$

We can prove that the derivatives of $\mathcal{L}$ following Equation 1 with respect to w and b are given by:

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) \cdot x^{(i)},$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) \cdot 1,$$

Task

First, use Gemini Guided Learning Mode to learn the following techniques/concepts to help you solve the problem:

1. How to create a vector using `std::vector`.
2. How to create random numbers in monder C++ (C++ 11 and later) using `std::random_device`, `std::mt19937`, `std::uniform_int_distribution`, and maybe more.
3. How to write data into a file using `fstream`.
4. Maybe more ...

   ChatGPT/Gemini can easily give you the solution to the entire problem. But you should learn how to use them .

   If I give you a problem that ChatGPT/Gemini cannot solve it, then we are all F_up.

Now, You need to write a C++ program to build a linear regression model by implementing both methods: Particularly, you should do the following

- Generate a dataset $\mathcal{D}$ following a simple linear function $y = kx + m$.
- Add noise to both the $x^{(i)}$ and $y^{(i)}$ values in the generated dataset.
  Note: Be careful with the amount of noise you add. You don't want it looks wild.
- Derive the linear regression model for your generated dataset by computing the model parameters w and b. You must implement both methods:

   1. Analytical method using normal equation

2. Gradient descent method

- Compare the results obtained by the two above-mentioned methods.
- Visualize your result by using Python. Unfortunately, plotting in C++ is a technically involved task.

Python is not the focus of our course. The following code may give you an idea about how the `.csv` file look like.

```python
import pandas as pd
import matplotlib.pyplot as plt

# Read the CSV generated by C++
df = pd.read_csv('data.csv')

# Plot to verify
plt.scatter(df['x'], df['y'], alpha=0.6, label='Data with Noise')
plt.title('Dataset Generated from C++')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

Marking scheme

- Task 1: 30/100
- Task 2: 40/100
- Report: 30/100

Requirements:

- You need to write significantly meaningful comments in your source code.
- You need to write significantly meaningful report.
  - You explain what you have learned.
  - Why you write the code the way you did.