VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND EGINEERING

PROGRAMMING FUNDAMENTALS

Assignment I

Lecturer: Dr. Le Thanh Sach

July/2017

1 Change log

Date	Version	Detail
10/07/2017	1.0	Creating assignment 1

2 Introduction

Researchers in public health sector has observed and deduced that the height of an 18-year-old depends heavily on his/her height when they were 2 years old. It's important to know if there is such as relationship. If there is, we can focus on improving the height of 2-year-olds and predict the height of an adult. With such motivation, a research group started to experiment to verify the deduction (that there is a relationship between the heights of both ages).

To conduct experiments, N individuals (N is huge enough) are observed from when they are 2 years old to when they reach 18. The height of these individuals are measured twice: once when they are 2 and once when they turn 18. With the collected data, the research team used a technique called "optimization" to find a model that can predict heights of people and verify the above standpoint. The optimization technique will be explained in a later section.

Let x be the height of an individual at 2 years old and t be the height of the same person at 18 years old. Both x and t are measured for each individual. After gathering pairs of (x, t) of N people in the experiment, researchers started to observe the distribution of N data points. Based on that observation, they come up with a more concrete hypothesis: the height of an adult (y) can be computed from the height when he/she is 2 years old using a linear model just as Eq. (1). Therefore, the task of the research group now is to find the coefficients a and b of the linear model in Eq. (1). Data points (x, t) measured from the people and the prediction line y are illustrated in Figure 1.

$$y = a \times x + b \tag{1}$$

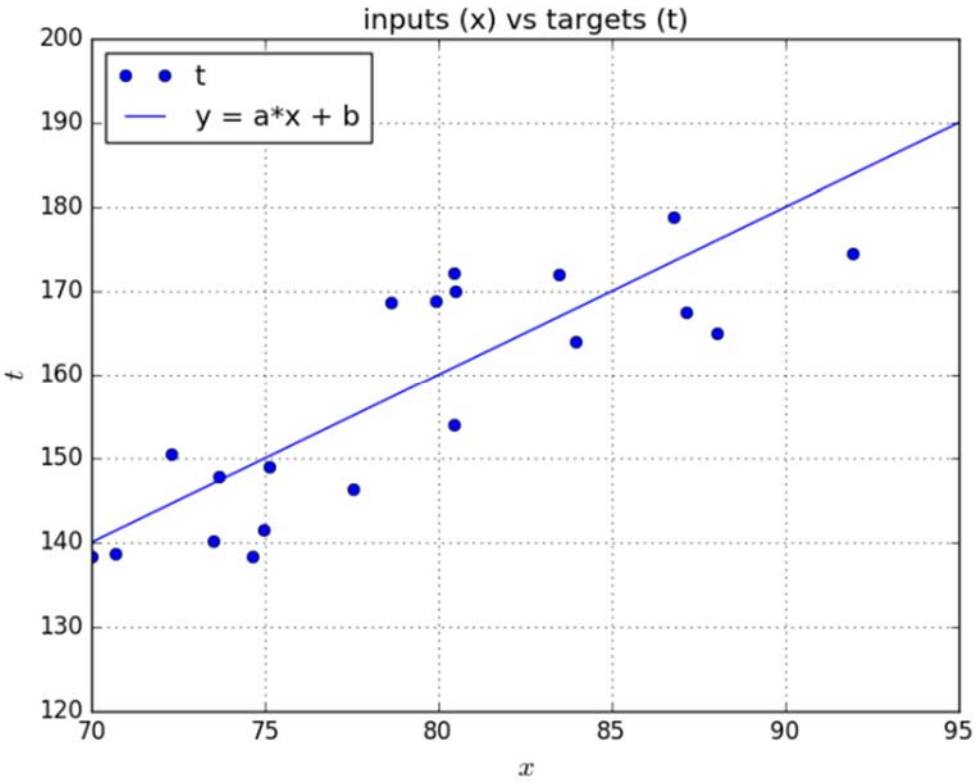

Figure 1: Data points (x, t) in the experiment and the prediction line $y = a \times x + b$.

3 Optimization technique

3.1 Prediction error

With the height x (at 2 years old) as the input, the height when an individual reaches 18 is y, computed as stated in Eq. (1). However, y is just the prediction of the linear model, and it might be a little different from actual measurements t of the people. y can be either smaller or bigger than t. This difference is called *prediction error* $e = t - y$. Here, e can be negative or positive. Usually, error is a non-negative value to indicate if the magnitude of the difference is big or small. One of the functions that has this characteristic is the *sum of squared errors*. This function is defined as follow:

$$L(a,b) = \frac{1}{2} \times \sum_{i=1}^{N} (t_i - y_i)^2$$
$$= \frac{1}{2} \times \sum_{i=1}^{N} (t_i - a \times x_i - b)^2 \quad (2)$$

Because x and t are measured from N actual individuals, only a and b in Eq. (1) are left to be found. That is, the error function in Eq. (2) depends on a and b. The coefficient $\frac{1}{2}$ is only used in Eq. (2) to simplify other equations in the optimization process.

3.2 Derivative, gradient and increasing/decreasing property of functions

The following information are known prior to the optimization:

1. N pairs of (x, t), these are height measurements of N individuals in the experiment.
2. The prediction model is known and it has the form of Eq. (1). In this model, there are two coefficients need to be found: a and b.
3. The error function that is used to represent the total prediction error is also given in Eq. (2): $L(a, b)$.

The optimization method to be used here is "gradient descent" or GD. The goal of GD is to find the pair a and b such that $L(a, b)$ reaches a local minimum. A local minimum might be or might not be a global minimum in the parameter space (of a and b).

This method works based on a property of the derivative of a function. If the derivative $f'(x)$ of a function $f(x)$ is positive at x_0, then $f(x)$ is increasing at x_0. That is, $f(x_0 - \alpha \times f'(x_0)) < f(x_0)$ with a small enough α and $\alpha > 0$. Similarly, if the derivative is negative at a point then it is decreasing at that point: $f(x_0 + \alpha \times f'(x_0)) < f(x_0)$. In short, **on the variable axis (the x axis), if we move along the opposite direction of the derivative of $f(x)$ then this function will decrease in value.**

Applying this property to the error function $L(a, b)$, we will get a similar result. For a function with multiple variables such as $L(a, b)$, there is no single "derivative" for it. However, there is a concept equivalent to derivative on which we will rely to optimize it. The following subsections will explain this concept and its related definitions in detail.

Partial derivative:

From Eq. (2), the partial derivative of $L(a, b)$ w.r.t. (with respect to) a and b are:

$$\frac{\delta L}{\delta a} = \sum_{i=1}^{N} (a \times x_i + b - t_i) \times x_i \tag{3}$$

$$\frac{\delta L}{\delta b} = \sum_{i=1}^{N} (a \times x_i + b - t_i) \tag{4}$$

Gradient vector:

A gradient vector is a vector containing two partial derivatives of $L(a, b)$ w.r.t a and b. The gradient vector in our case here is $g = \left[\frac{\delta L}{\delta a}, \frac{\delta L}{\delta b}\right]^T$.

A point in 2D plane (a, b):

Let P_0 be a point in the plane (a, b). P_0 has the coordinates (a_0, b_0). We can represent P_0 in vector form: $P_0 = [a_0, b_0]^T$.

The relationship between the increasing/decreasing property of $L(a, b)$ and its gradient vector:

Similar to derivative, the direction of the gradient vector is also the direction along which the function will increase when it moves. In our problem, we want to find the minimum value of our function, hence we need to move in the opposite direction to that of the gradient vector. This "moving" is done by subtracting a point by an amount of $a \times g$, as in Eq. (5) and illustrated in Figure 2. A vital note here is we need to know the direction of g so that we can move along its opposite direction with an amount of α. In this movement, the magnitude of g is not important. Therefore, we can normalize g so that it has the length of the unit vector (which is 1). Normalization is done by dividing g to its magnitude $|g|$.

Eq. (5) can be computed through Eq. (6) and (7) for each component in the vector. Here, a and b are two components of the gradient vector g.

$$P_1 = P_0 - \alpha \times g \tag{5}$$

$$a_1 = a_0 - \alpha \times g_a \tag{6}$$

$$b_1 = b_0 - \alpha \times g_b \tag{7}$$

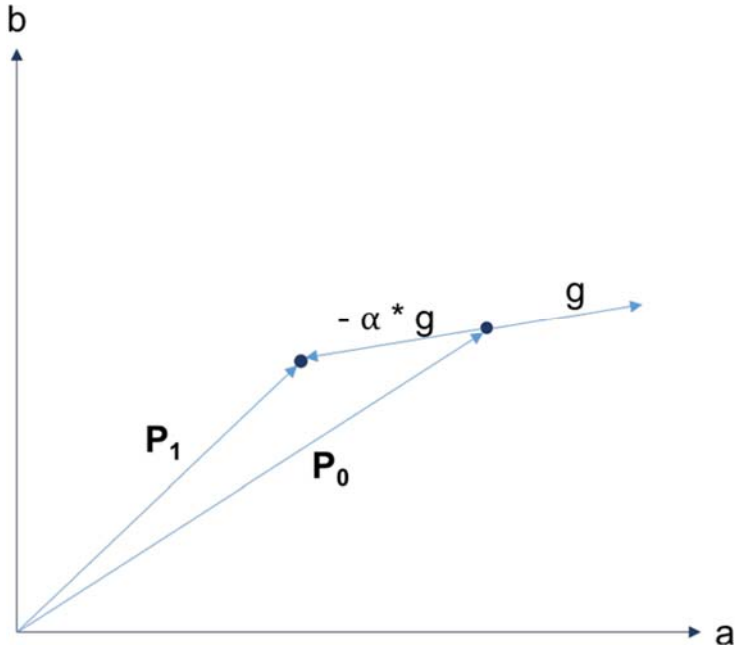

Figure 2: Gradient vector $\boldsymbol{g}$ and the movement from $\boldsymbol{P_0}$ to $\boldsymbol{P_1}$ by subtracting $\boldsymbol{P_0}$ with $-\boldsymbol{\alpha} \times \boldsymbol{g}$.

3.3 The "Gradient descent" optimization method

In the above discussion, given $\boldsymbol{P_0}$, if $\boldsymbol{P_1}$ is found from Eq. (5) or (6) and (7) then we will have $\boldsymbol{L(P_1) < L(P_0)}$. Consequently, if we assign $\boldsymbol{P_0 = P_1}$ and start to compute $\boldsymbol{P_1}$ again using the new $\boldsymbol{P_0}$, we can reduce the value of $L(a, b)$ even further until it reaches a local minimum.

Therefore, to find a minimum of $L(a, b)$, we can use the following iterative method:

1. Let P be a point in the plane (a, b), this point can be initialized with any value at the beginning.
2. Assign a real positive value to α.
3. Iterate these steps until a stopping condition is satisfied:
    a. Compute the prediction value y_i for all N samples.
    b. Compute the gradient vector using Eq. (3) and (4).
    c. Normalize the gradient vector and assign $g = [g_a, g_b]^T$.
    d. Update P using Eq. (6) and (7).

Note:

- The starting value for P: in general, the error function ($L(a, b)$) will have a global minimum and multiple local minima. Because of this, choosing a different starting point for P will make GD run differently. Therefore, running GD multiple times with different starting points is a solid strategy to find the best P and it can help us find lower error values for L. In this assignment, the starting value for P is read from an input file therefore you don't need to be worried about it.

- The hyper-parameter α: it's also called the learning rate of GD. This value has a crucial property to GD. If α is too small, it will take a long time for P to reach a minimum because the change of P in each iteration is small. In contrast, if α is big, the optimization process can be much faster. However, when α is *too* big, P can actually jump over the minima and never be able to converge. In this case, P will move back and forth, around the minimum but it won't get there. In this assignment, this α parameter is also given in the input file.
- Stopping condition: there are many heuristics to decide when should GD be stopped. In this assignment, the GD will stop when the number of iterations reach a limit. This limit is specified in the input file.

4 Assignment 1 task list

In this assignment, you are required to write a program that does the following jobs:

1. Read the input file.
2. Evaluate the prediction model in Eq. (1) using the given input.

The rest of this section will explain your tasks in details.

4.1 Reading input

The input file is named **"assignment1.input.txt"**. This file contains the following parameters to run a GD process, put below the **"Training and Validation Parameters"** header:

- **trn_params.num_iterations**: the number of iterations for GD mentioned in Section 3.3.
- **trn_params.learning_rate**: the learning rate α.
- **trn_params.start_a** and **trn_params.start_b**: the starting point for a and b of point P.
- **trn_params.num_folds**: the method to be used to divide the data. This will be explained later.

Following the above parameters are a list of (x, t) pairs put under the **"Data samples"** header. The first (x, t) pair is on the 12th line in the input file (the first line starts from 1st, not 0). Each line has two numbers x and t, with a few spaces in the middle. The maximum number of samples (number of pairs (x, t)) is NUM_SAMPLES = 20000.

Here is an example of the file **"assignment1.input.txt"**:

```
------------------------------------------------------------
Traning and Validation Parameters
------------------------------------------------------------
trn_params.num_iterations:                              50
trn_params.learning_rate:                              0.1
trn_params.start_a:                                      0
trn_params.start_b:                                      0
trn_params.num_folds:                                    3
------------------------------------------------------------
Data samples
------------------------------------------------------------
76.87       153.78
83.76       167.48
```

4.2 Evaluating the model and output

Let M be the number of points read from the input file. Each point is a pair (x, t). Let K be the number read from **trn_params.num_folds**. To evaluate the prediction model in Eq. (1), you must do the following steps:

- Split M data samples into K smaller collections. Each collection will have $D = M/K$ samples. If M is not divided by K, the last collection can have more samples than the others. After reading M samples, you must store them in an array **in the same order** they are listed in the input file. Therefore, the first collection will have all samples from 0 to $(D - 1)$, the second collection will have all samples from D to $(2D - 1)$, etc. **The last collection will contain remaining samples after you finish putting samples into the previous $K - 1$ collections.**
- The program will run GD and evaluate the resulted model K times. Each running time is marked with an index $k = 1,2, \dots, K$. In iteration numbered k, the program must do the following jobs:
    1. Split the entire dataset (M samples) into two smaller sets, one is called **TRN** and the other is called **TST**. The subset **TST** contains all samples from collection numbered k. The number of samples in **TST** is D. Based on the above explanation, $D = \lfloor M/k \rfloor$ for all collections from 1 to $K - 1$ and $D \geq M/k$ for the last collection. The set **TRN** contains all samples from the other $K - 1$ collections. Let the number of samples in **TRN** be N. We have $N = M - D$. Note: for each k ($k = 1,2, \dots, K$), we will have different TRN and TST.
    2. The program uses samples from **TRN** to find the parameters (a and b) in the prediction model in Eq. (1), using the GD optimization method given in Section 3.3.

3. The program uses the optimized prediction model to calculate the average prediction error for all samples in **TST**. This error is calculated using Eq. (8).

$$E_{rmsd} = \sqrt{\frac{1}{D} \times \sum_{i=1}^{D} (a \times x_i + b - t_i)^2} \quad (8)$$

4. Apart from calculating the average error in Step 3, the program must also calculate the error **histogram** from e_i from TST. Here, $e_i = y_i - t_i$. y_i is the predicted value and t_i is the measured value from the i[th] individual in **TST**, $i = 1,2,\dots,D$. e_i can be positive or negative. Section 4.2.1 will explain how to calculate the histogram in details.
5. The program will output into a file. The output consists of K lines. Each line has the following numbers: a and b; the prediction error of D samples calculated in Step 3; 10 numbers represent the error histogram of error. In total, each line consists of 13 values. Each value is a real number printed in **fixed floating-point format**, width is **10**, decimal precision is **5**, alignment is set to **right**. Because the program performs evaluation K times, there will be K lines in the output file (this hasn't included other text lines, please check the sample output file). An example line in the output file **"assignment1.output.txt"**:

```
1.99995    0.51018    9.14897 ... (There are 10 more numbers)
```

4.2.1 Computing the histogram of errors

Assume that we have already found the parameters a and b for the prediction model in Eq. (1), there will be D samples (x_i, t_i) for us to test on, $i = 1,2,\dots,D$. We must do the following steps to compute the histogram of errors:

1. Compute the errors $e_i = y_i - t_i$, $i = 1,2,\dots,D$. Reminder: y_i is the predicted value and t_i is the measured value from the i[th] individual in **TST.** e_i can be positive or negative.
2. Find the average value $\bar{e}$ and the variance σ of all e_i, using the following equations:

$$\bar{e} = \frac{1}{D} \times \sum_{i=1}^{D} e_i \quad (9)$$

$$\sigma = \sqrt{\frac{1}{D} \times \sum_{i=1}^{D} (e_i - \bar{e})^2} \quad (10)$$

3. Calculate the lower bound and the upper bound of e_i. These bounds will be used to compute the histogram using Eq. (11) and (12). Note: the lower bound V_{min} will be $3 \times \sigma$ smaller than $\bar{e}$, the upper bound V_{max} will be $3 \times \sigma$ greater than $\bar{e}$.

$$V_{min} = \bar{e} - 3 \times \sigma \quad (11)$$

$$V_{max} = \bar{e} + 3 \times \sigma \quad (12)$$

4. Split the newly-found $[V_{min}, V_{max}]$ interval into 10 sub-intervals with equal lengths. Therefore, we need to calculate 9 splitting points. These points will be called $L_1, L_2, \dots, L_9$. With this method, the first sub-interval is $[V_{min}, L_1)$, the second sub-interval is $[L_1, L_2)$, etc. and the last sub-interval is $[L_9, V_{max}]$. Note: $[V_{min}, L_1)$ includes V_{min} but not L_1 in the interval, $[L_9, V_{max}]$ includes both L_9 and V_{max} in the interval.
5. Using an array of 10 elements to represent the histogram. The first element of the array stores the number of e_i values lies inside the interval $[V_{min}, L_1)$. Similarly, the next element in the array stores the number of e_i lies inside the interval $[L_1, L_2)$, etc.
6. Finally, we need to normalize the histogram so that it has the property of a probability distribution. We normalize the histogram by dividing each element in the array by the sum of all of its elements. Note: the sum here might be less than D because some e_i will lie outside of the $[V_{min}, V_{max}]$ interval. Another note: the sum of the normalized should be 1, but the program might not print exactly 1 due to the numerical of the computer and the effect of **setprecision**.

5 Guidelines

To develop the program specified in this assignment, you must learn to use control structures (if...else and loops), struct data type and array. This assignment does not heavily require your ability to organize your source code and project. Therefore, you can follow these guides to implement your program:

1. Create an empty project in Visual Studio or any equivalent IDE (Dev C++, Code::Block, xcode, etc.)
2. Create a file called **"program.cpp"**, all source code of your program **MUST** be put in this file. The reason for this is the auto-grader program will only look for the file called **"program.cpp"**, build a program from it and grade. If you put your codes in multiple files, when the grader builds your program only with **"program.cpp"**, the built program won't run properly. Note that, if you submit a file with a wrong name (for example: "program1.cpp" or "assignment1.cpp"), the grader can't also compile.
3. Algorithm-wise, your program only includes two steps in Section 4: (1) reading input and (2) evaluate the prediction model using the test samples from the input.

4. For reading input:
   - **How to read input from the file? Answer:** revisit previous labs.
   - **How to store data? Answer:** you can define a new struct to describe your data. Since the number of data points can be up to 20000, you can use array to store them. You can also use the **vector** data type (#include <vector>) if you want.
   - The number of data samples in the input file will be guaranteed to be ≤ 20000. You can use **macro** to define this number.
5. The evaluation step:
   - If your program starts to execute this step, it means that the input was read without error.
   - In the input, there is a number for **trn_params.num_folds**. This is K, we will have K iterations to evaluate. In each iteration, we will do the followings: (1) Create two subsets called **TRN** and **TST**, (2) train the prediction model using **TRN**, i.e. finding a and b of Eq. (1), (3) evaluate the trained model on **TST**, and (4) compute the histogram of errors.
6. Run and test your program.
7. When you are allowed to submit your work, you should upload only **"program.cpp"** onto the online grader system. The grade will be shown to you after a short time. You can fix your codes accordingly. Before the final deadline, you can submit your codes multiple times.

6 Grading

There is a sample program, created by TAs of the course called **CTM**. The output of your program (written in **assignment1.output.txt**) will be compared to the output of **CTM**. If both output files are **identical** then your program works correctly on that specific test case. Otherwise, your program is incorrect. Two numbers in both output files are considered the same if their difference is smaller than 10^{-4}. This is the reason why the decimal printing precision was set to 5. (5 digits after the dot).

The grader will grade your program using 5 undisclosed datasets. Your final score will be the average scores calculated from all of these 5 datasets. The score for each dataset consists of 3 main categories, corresponds to 3 main output categories you need to print:

1. The score for training the correct prediction model (50%): you printed the correct pair of a and b in Eq. (1).
2. The score for computing the correct prediction error (20%): refer to Eq. (8).
3. The score for computing the correct histogram of errors (30%): refer to Eq. (9) to (12).

Since there are many iterations for each dataset (many lines of output for each dataset), the score for that dataset will be the average of all lines.

There will be a public dataset, not used for grading but only for testing purpose. For this dataset, the output is available for you so that you can fix your code based on it.

7 Penalty

You will receive 0 if you do one of the following:

- Plagiarism. If your codes were found to be very similar to the codes of another student, both will get 0.
- There is a compiling error. **The grader system will use g++ to compile your codes. Therefore, make sure that your codes work with g++.**
- Wrong printing format.
- The output values are different from the outputs of the **CTM**.
- Your program stops at some point and can't finish. **Please remove system("pause") or any cin in your file before submitting.**

You might receive 0 for a dataset if:

- The output file does not have K lines and 13 columns.
- The output file has correct format but its values are incorrect.

8 Submitting deadline

1. The grading system will be **opened in 19:00, Monday, 17, July, 2017**.
2. The system will **close in 13:00, Thursday, 28, July, 2017**.
3. Students can submit codes several times. The last submission will count as your final score.

9 Submission guideline

9.1 Verifying the program before submitting

The auto-grader will use the **"g++"** compiler from Linux to compile your codes. To test if your codes can work with "g++":

1. Visit https://www.tutorialspoint.com/compile_cpp11_online.php, this is an online IDE. This site allows you to upload your codes and compile it with g++.
2. Delete the default "main.cpp" file on the site.
3. Go to **"File" > "Upload file"** to upload your **"program.cpp"** and **"assignment1.input.txt"**. Remember that you must choose "root" each time you upload.
4. Go to **"Project"** > **"Compile Options"** to specify compiler commands. A dialog will pop up, it has two fields. Enter the following commands in the corresponding fields:

Compilation Command: `g++ -std=c++11 -o program -c program.cpp`

Execution Command: `program < "assignment1.input.txt" > "output.txt"`

5. Hit **"Compile"** to compile your program. If there is an error, it will be shown in the green terminal. You should fix your codes until there is no error left.
6. After compiling successfully, the terminal will let you know. You can then hit **"Execute"** to run the program. After running successfully, open **"output.txt"** to view your result. Remember to hit the **"Refresh"** button on the left panel to see **"output.txt"**.

9.2 Submitting

You must do the following to submit your codes:

1. Visit http://cse.hcmut.edu.vn/onlinejudge/
2. Hit Reset password. The site will open a dialog for you to enter your Student ID.
3. Enter your ID, for example: "1610129". The site will send a password to the corresponding email account based on your ID (1610129@hcmut.edu.vn). At this point, you will have the account name (your ID) and the password to login.
4. Login the site.
5. Choose **"Nộp bài"** and then:
   - In **"Task"**, choose **"PF-assignment-1".**
   - Check the **"Deadline"**.
   - Hit **"Choose File"** to choose your **"program.cpp"** file. **Remember to check if your file has the correct name. The grader will only look for a file with this exact name.**
   - Hit **"send"**. The site will show **"Upload done"** to announce that you have uploaded successfully.
6. After submitting, browse tab **"Kết quả"** to check the grading results. This tab consists multiple lines of information came from all of your submissions. In this tab, there will be a **"Score"** column, it is the corresponding score for that submission. You can click on your ID or the score to view the detailed grading results.

Note: Usually, the system will return your grades after a few seconds. However, it can take longer if there are multiple students using the site as the same time.