PHAS0100: Research Computing with C++

Assignment 1: Constructing a Small Research Project

Assignment Submission
For this coursework assignment, you must submit by uploading a single Zip format archive to Moodle. You must use only the zip tool, not any other archiver, such as .tgz or .rar. Inside your zip archive, you must have a single top-level folder, whose folder name is your student number, so that on running unzip this folder appears. This top-level folder must contain all the parts of your solution. Inside the top-level folder should be a git repository named PHAS0100Assignment1. All code, images, results and documentation should be inside this git repository. You should git commit your work regularly as the exercise progresses.

Due to the need to avoid plagiarism, do not use a public GitHub repository for your work - instead, use git on your local disk (with git commit but not git push), and **ensure the hidden .git folder is part of your zipped archive** as otherwise the git history is lost. You can push to a private GitHub repository but it is your responsibility to ensure that it is private and that others cannot see or copy your work as pieces of identical (or suspiciously similar) work will both be treated as plagiarised. Whilst it is fine to look up definitions and general documentation for C++/libraries online it is not acceptable to use code taken from the internet or other sources for the problem solutions. The exception to this is the CMakeCatch2 code that can be used as a template CMake project.

You are encouraged to use: CMakeCatch2 as a basis when creating PHAS0100Assignment1, but you can also choose to implement your own project from scratch if you want. Do be aware that the code is expected to compile and run, and marks will be deducted if the markers cannot run the code. Your documentation should give very clear build instructions in `README.md`.

To create your `PHAS0100Assignment1` project from the `CMakeCatch2` you should clone both `CMakeCatch2` and the CMakeTemplateRenamer (see `README.md` for instructions) to your machine and run the following from the directory containing them (all one command):

```
CMakeTemplateRenamer/rename.sh CMakeCatch2 PHAS0100Assignment1 PHAS0100Assignment1 "PHAS0100
Assignment 1 Linear Regression" lrg Y
```

In general you are free to develop in your own environment, but you need to ensure that your code compiles and runs on Ubuntu 18.04.3 and with g++ 7.4.0 (enforcing C++14) as this is the environment the markers will use to test the code.

Preliminaries
The aim is to write a piece of software that can perform linear regression. While this is fairly widespread in many libraries, especially in other languages such as Python, the aim here is to demonstrate that you can produce a C++ project that can be run by other people, on their machines. In class, we have learnt things like:

- Build setup using CMake
- Unit testing
- Error handling using exceptions
- Avoiding raw pointers – either use STL containers, or smart pointers
- Program to Interfaces
- Dependency Injection

and the aim here is to put this all together into a coherent project.

Reporting Errors
You can post questions to the Moodle Discussion/SLACK channel, if you believe that there is a problem with the example CMake code, or the build process in general. To make error reporting useful, you should include in your description of the error: error messages, operating system version, compiler version, and an exact sequence of steps to reproduce the error. Questions regarding the clarity of the instructions are allowed, but obvious "Please tell me the answer" type questions will be ignored.

Important: Read all these instructions before starting coding. In particular, some marks are awarded for a reasonable git history (as described in Part B), that should show what you were thinking as you developed.

Part A: Linear Regression App (55 marks)
The first part of this coursework is to get you to setup the project. These instructions will guide you through.

1. Please read chapter 4 of the "Hands on Machine Learning" that covers linear regression and provides background to the equations used in this assignment. The PDF is on Moodle in the Coursework topic.

   [0 marks]

2. Data can come from many places, e.g. file, network, or randomly generated. So, we first define an interface of what we expect from our data provider. In class we learnt: "program to interfaces", so:
   a. Create a header file, containing a pure virtual interface class, with a method equivalent to:

```
virtual std::vector<std::pair<double, double> > GetData() = 0;
```

   The data returned should be a vector of X, y pairs, where X is the observed feature value, and y is the target/label/predicted value.

   b. Ensure the file is included in CMakeLists.txt, so that your build environment will know it exists.

   c. Create a header and implementation file of a new concrete (i.e. not abstract) class that implements this interface. At first, just write an empty method with the signature above.

   d. Create a unit test file, that will instantiate an instance of your new concrete class.

   e. Check that you can compile and run the test.

   Hints: a suitable folder/file structure within `PHAS0100Assignment1` would be:

   // Pure abstract Interface, with I in the file name
   `PHAS0100Assignment1/Code/Lib/lrgDataCreatorI.h`

   // Concrete implementation
   `PHAS0100Assignment1/Code/Lib/lrgLinearDataCreator.h`
   `PHAS0100Assignment1/Code/Lib/lrgLinearDataCreator.cpp`

   // Unit tests
   `PHAS0100Assignment1/Testing/lrgLeastSquaresSolverTests.cpp`

   [1 mark for each a-e, 5 marks total]

3. Now we implement the class to generate some data. The idea is that if we create some fake data, we know what the answer should be.
   a. In the class that you created as part of 2c, implement a function that generates data that fits the linear model: $y = \theta_1 x + \theta_0 + noise$

b. In class we learnt the RAII pattern and dependency injection pattern, rather than using setters/getters. Ensure that parameters for your generator are passed in via constructor.

c. Write a specific unit test that checks:
   i. The number of returned items is correct
   ii. The distribution of the returned items is correct.

   Hints: Use STL random number generators: http://www.cplusplus.com/reference/random/ For unit testing, it's sufficient for this exercise to test that a random sequence of numbers has the correct mean value.

[3 + 2 + 5, 10 marks total]

4. Similar to part 2, create a pure abstract interface for the solver, and a concrete implementation. Notice how we have separated the thing that generates or provides data from the thing that provides a solution.

   a. Create a header file, containing a pure virtual method to fit data to a model. For this simple exercise, we know that the model only requires 2 parameters, θ_0 and θ_1, so the returned value can be a pair of doubles representing θ_0 and θ_1.

   i.e. equivalent to:

```
std::pair<double, double> FitData(std::vector<std::pair<double, double> >)
```

   b. Ensure the header file is included in your CMakeLists.txt

   c. Create a header and implementation file of a new concrete (i.e. not abstract) class that implements this interface. At first, just write an empty method.

   d. For simplicity reuse the unit test file created in section2d. (0 marks)

   e. Check you can compile and run the test.

   f. At this point, consider using typedef's to simplify your code, as lots of template brackets get ugly (see https://en.cppreference.com/w/cpp/language/typedef).

Hints: a suitable folder/file structure within PHAS0100Assignment1 would be:

```
// Pure abstract interface with I at the end of the file name.
PHAS0100Assignment1/Code/Lib/lrgLinearModelSolverStrategyI.h

// Concrete implemntation
PHAS0100Assignment1/Code/Lib/lrgNormalEquationSolverStrategy.h
PHAS0100Assignment1/Code/Lib/lrgNormalEquationSolverStrategy.cpp

// Same unit test file.
PHAS0100Assignment1/Testing/lrgLeastSquaresSolverTests.cpp
```

[1 mark for each of a,b,c,e,f, 5 marks total]

5. Implement a first solver, using Normal Equations. See equation 4-4 in "Hands on Machine Learning" and the Notations section in Chapter 2. This would be placed in the concrete class created in part 4c.

Equation 4-4. Normal Equation

$$\hat{\theta} = \left(\mathbf{X}^T \cdot \mathbf{X}\right)^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y}$$

- $\hat{\theta}$ is the value of θ that minimizes the cost function.
- $\mathbf{y}$ is the vector of target values containing $y^{(1)}$ to $y^{(m)}$.

Hints: This project includes Eigen. Copy data from STL arrays to Eigen, and solve. Don't worry about performance.

[Implementation 5 marks, Unit tests 5 marks. Both at markers discretion, 10 marks total]

6. Implement a second solver. As mentioned in class the point here is to demonstrate how 2 methods can co-exist in a project, and the project should be able to run both of them.
    a. Create a new solver, e.g. Gradient Descent. Gradient Descent is a bit fiddly to parameterise, so, optionally, you could use try SVD.

    b. If you use Gradient Descent, ensure the parameters are all adjustable, by whoever is using the class (see Dependency Injection covered in lecture 4).

    Hint: a suitable folder/file structure within PHAS0100Assignment1 would be:

    // Another concrete implementation of lrgLinearModelSolverStrategyI.h
    `PHAS0100Assignment1/Code/Lib/lrgGradientDescentSolverStrategy.h`
    `PHAS0100Assignment1/Code/Lib/lrgGradientDescentSolverStrategy.cpp`

    Again, re-using the same, single unit test file.

    [Implementation 5 marks, Unit tests 5 marks. Both at markers discretion, 10 marks total]

7. Now, in reality, you would be processing data produced by some scientific experiment. The idea here is to now create another concrete implementation of the interface defined in section 2a, where instead of randomly generated data, data is read in from a text file.
    a. Create a new header and cpp file, of a concrete class that implements your interface containing the GetData() method.
    b. Implement the class, using STL funtions to read data from a plain text file. Assume 2 values per line, representing X and y, each space separated.
    c. Write unit tests to ensure you can read TestData1.txt and TestData2.txt (provided)

    Hint: a suitable folder/file structure within PHAS0100Assignment1 would be:

    // Another concrete implementation of lrgDataCreatorI.h
    `PHAS0100Assignment1/Code/Lib/lrgFileLoaderDataCreator.h`
    `PHAS0100Assignment1/Code/Lib/lrgFileLoaderDataCreator.cpp`

    [1 + 3 + 1 = 5 marks in total]

8. Now you would plug it all together, and if the unit tests are working, you should be able to use the package to process experimental data (TestData1.txt and TestData2.txt)
    a. Create a command line app, with an appropriate name.

    b. The command line app should print a useful help message to tell the user what arguments to use. Normally the -h or --help switch is used here.

    c. With zero command line arguments, the program should not crash, and should respond with the help message

    d. The command line app should check for the right number of arguments and respond correctly.

    e. The command line app should check if a file was read correctly, and throw exceptions if not.

    f. The aim is to provide to the user BOTH methods of solving the least squares regression problem. So, you should provide a command line argument that enables you to switch methods. And both methods should give you approximately the same answer.

g. The top-level program should catch all exceptions and exit gracefully.

Hints:

You can use basic C++ command line parsing:
http://www.cplusplus.com/articles/DEN36Up4/
Or you could try integrating a command line parser such as:
https://github.com/CLIUtils/CLI11(which is header only)

[1 + 1 + 1 + 2 + 2 + 2 + 1 = 10 marks total]

Part B: Additional Considerations (15 marks)

9. Given $y = \theta_1 x + \theta_0$then TestData1.txt should give θ_0 = 3 and θ_1 = 2, and TestData2.txt should give θ_0 = 2 and θ_1 = 3. Provide some evidence that you have this. E.g. screenshot of a running command line program.

[5 marks]

10. Nice git commit log. Effective commenting.

[5 marks]

11. Update the front page README.md to give clear build instructions, and instructions for use.

[5 marks]