BEng Biomedical Engineering
Object-Oriented Programming

Coursework 1

Objective

To gain practical experience of C++ coding to solve a simple problem.

Introduction

Magnetic Resonance Imaging is a powerful imaging technique capable of producing images of internal organs with good soft tissue contrast. It relies on a number of hardware components, as illustrated in Figure 1: a powerful permanent magnet; magnetic field gradients provided by dedicated gradient coils (Figure 1, right); and radio-frequency coils to excite the nuclear spins (typically ^{1}H) and detect the resulting nuclear magnetic resonance signal.

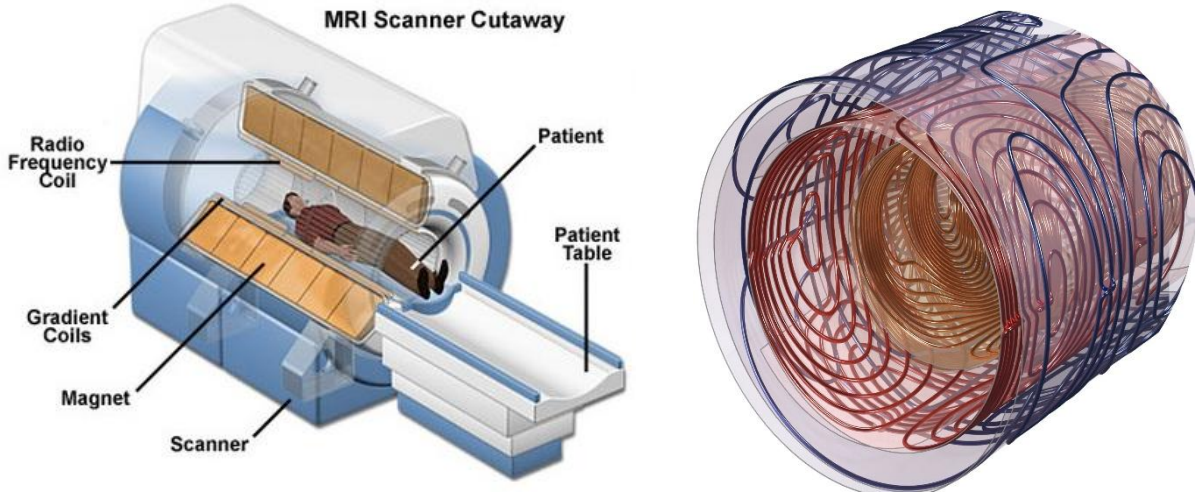


Figure 1: the different hardware components of a typical MRI scanner. Left: a cutaway view showing the arrangement of the main magnet, gradient coils and RF coil. Right: an example of the design of a gradient coil, showing the different concentric windings required to impart magnetic field gradients along the x, y & z axes.

The role of the gradient coils is to impart a linear change in the resonance frequency of the ^{1}H nuclear spins as a function of position, which allows the position of the signal to be inferred. To provide full images, these magnetic field gradients need to be rapidly modulated in time, which is done by driving very large currents through the gradient coils and switching them rapidly on & off at appropriate times along the appropriate axes.

Eddy-currents in MRI

Because of Faraday's law of electromagnetic induction, these rapid and intense changes in the magnetic fields induce voltages, which in turn drive currents in nearby conductive structures. These *eddy currents* create magnetic fields that oppose the change in magnetic field that created them (this is called Lenz's law).

In MRI, the net result is that the temporal evolution of the magnetic field gradient actually produced will differ from the waveform provided to the gradient amplifier. This is illustrated in Figure 2, showing how an ideal trapezoidal input waveform will end up distorted and lagging behind what was expected, and the resulting impact on images acquired using echo-planar imaging (a single-shot imaging technique). Eddy currents can affect the images in many different ways, depending on the type of MRI sequence used and other factors, but they are invariably problematic. For this reason, a variety of techniques are used to minimise eddy-currents, including hardware solutions (e.g. actively shielded gradients) and gradient pre-emphasis.

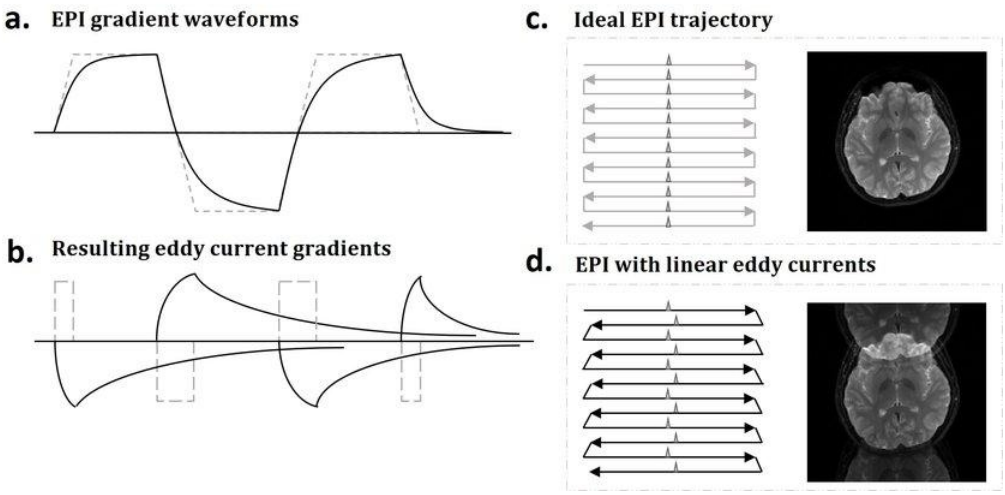


Figure 2: illustration of the effect of eddy-currents. (a) the expected waveform (dotted line) is distorted due to eddy-currents, resulting in the delayed time evolution of the actual gradients (solid line). (b) The eddy-currents (solid line) originate from the rapid changes in the magnetic field that occur when there is a change in the gradient (dotted line). (c) image that would ideally be acquired using echo-planar imaging (EPI). (d) the image acquired when eddy currents introduce significant delays in the signal shows clear 'ghosting' artefacts.

Gradient pre-emphasis

Gradient pre-emphasis involves modifying the signal time course provided to the gradient amplifier to compensate for the expected effect of eddy-currents, as illustrated in Figure 3. If the properties of the eddy-currents are known, it is possible to predict the time course of the actual gradient for a given input waveform. With pre-emphasis, the input waveform is modified so that the time course of the actual gradient matches the desired waveform more accurately.

For the purposes of pre-emphasis, eddy-currents are typically modelled as multiple components, each with a given amplitude and decay rate constant. The parameters of the different eddy-current components are measured during routine maintenance as part of the calibration of the MRI scanner, and stored on the system for use in gradient pre-emphasis.

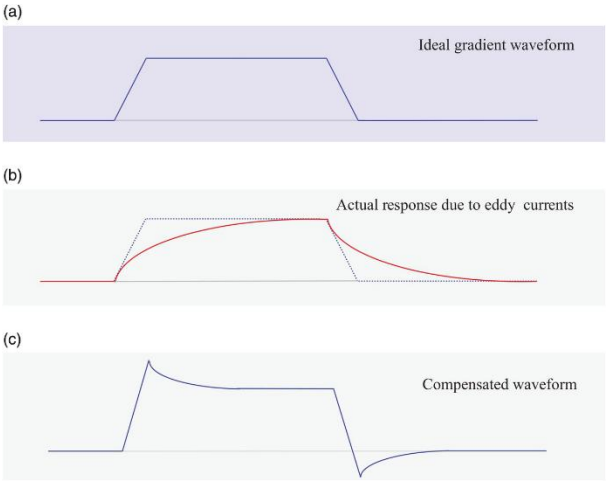


Figure 3: illustration of gradient pre-emphasis. (a) the desired gradient waveform. (b) the actual gradient time course obtained differs due to the effects of eddy-currents. (c) with pre-emphasis, a modified waveform is provided as input to the amplifier, designed in such a way that the actual gradient time course matches the desired waveform.

Instructions

Your task in this coursework is to write a C++ program to implement a simple gradient pre-emphasis strategy. This program should perform the following main steps (these are described in more detail below):

1. Load the parameters describing the eddy-currents for the system from the file provided.
2. Load the desired time course for the gradient from the file provided.
3. Make a copy the desired time course, which will be the initial *input* gradient time course.
4. Compute the *predicted* time course for the *input* time course by taking the effects of eddy-currents into account.
5. Compute the maximum absolute deviation between the *desired* and *predicted* time courses, and display this value on the terminal
6. Display the *input* and *predicted* time courses on the terminal.
7. Compute the difference between the *desired* and *predicted* time course.
8. Add that difference back to the current *input* time course.
9. Go back to step 4 and repeat for the desired number of iterations
10. If requested, write the final estimate of the compensated *input* time course to the file specified on the command-line

Command-line interface

Your program should accept at least 2 arguments: the configuration file containing the parameters for the eddy-currents, and the file containing the desired time course. If a third argument is provided, this should be interpreted as a request to store the estimated compensated time course to the file specified in that argument.

Your program should also accept a "`-n num`" command-line option to override the default number of iterations. If left unspecified, the default number of iterations should be set to 10.

Loading the input data

You are provided with a text file called `parameters.txt` file containing the eddy-current parameters. Each line in this file contains two values: the amplitude and the rate constant for each eddy-current component.

You are also provided with a text file called `gradient.txt` file containing the intensity values at regular sampling intervals for the desired gradient time course.

Computing the predicted gradient time course

The impact of the eddy-currents can be predicted using a simple model. Each eddy-current component can be modelled independently, and will have an associated current I_n (with initial value zero). The instantaneous chance in the gradient, dG, causes the current to increase in proportion. At the same time, the resistance in the system causes the current to reduce in proportion to the magnitude of the current. The predicted gradient is then given as the input gradient minus the eddy-current contributions, which is the sum of their currents weighted by their respective amplitude parameter.

This can be written in simplified form as:

$$dG_t = G_t - G_{t-1}$$

$$I_{n,t} = I_{n,t-1} + dG_t - R_n \times I_{n,t-1}$$

$$G'_t = G_t - \sum_{n}^{N} A_n \times I_{n,t}$$

where:

- G_t and G'_t are the *input* and *predicted* waveforms at time point t respectively
- $I_{n,t}$ is the value of the current at time point t for the n^{th} eddy-current component
- A_n and R_n are the amplitude and rate constant parameters for the n^{th} eddy-current component.
- N is the number of eddy-current components listed in the parameter file.

Computing the compensated time course

The compensation to be applied to the waveform can be computed using the following iterative approach:

Given the desired waveform:

- Initialise the current estimate of the (compensated) *input* waveform as the *desired* waveform
- Repeat for the desired number of iterations:
    - Compute the *predicted* waveform for the current *input* waveform
    - Compute the difference between the *desired* and *predicted* waveforms
    - Add the computed difference to the current *input* waveform, to produce the next estimate of the *input* waveform

Displaying the waveforms

The various waveforms can be displayed using the terminal_graphics library. Refer to the relevant course slides and the project README for detailed instructions.

Briefly: download the terminal_graphics.h header and place it in your project folder, alongside your own code. Make sure to `#include` that header into your code. Then use the TG::plot() functionality as illustrated below:

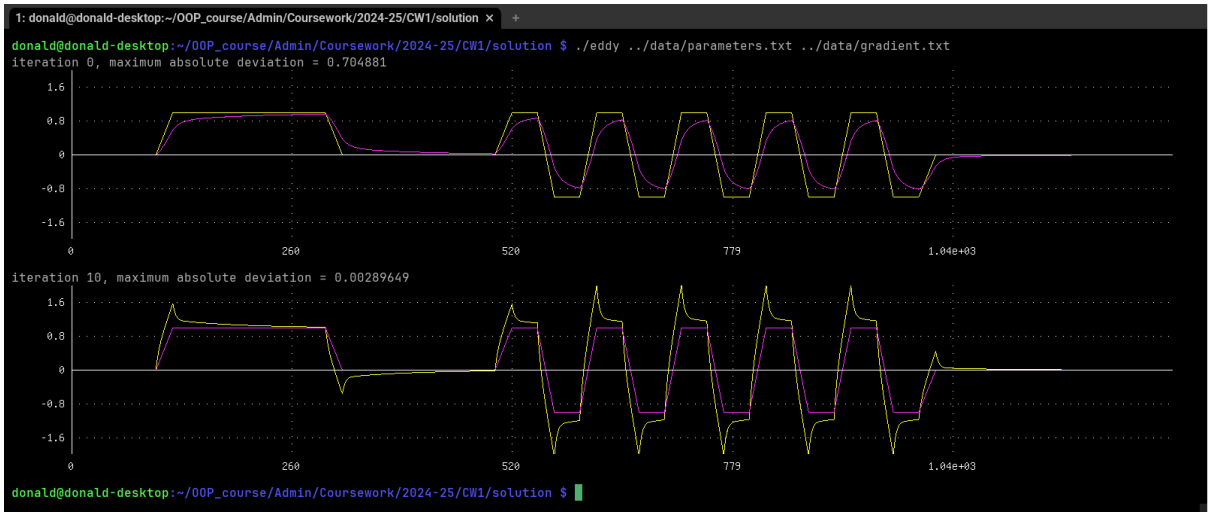
```
#include "terminal_graphics.h"

...

std::vector<float> desired_gradient, predicted_gradient;

...

TG::plot (width, height)
     .add_line (desired_gradient)
     .add_line (predicted_gradient, 3);
```

Writing the final compensated time course

Your program should only store the output if provided with a third argument on the command-line. If this is the case, you should write the final compensated time course to the file specified as the last argument, in the same format as the original desired time course.

Example output

The example below only shows the initial and last iteration of the algorithm:



Reporting Requirements

You should submit a C++ project that meets as many of the requirements as possible. You do not need to submit any written report but do try to use variable/function naming, comments and indentation to make your program as easy to understand as possible. Also try to make your program as resilient to runtime errors as possible.

Submission will be via the KEATS system. The submission point will only allow you to upload a single file so you should combine all files into a single ZIP file. Please only include your `.cpp` and `.h` files (please run "`oop_build clean`" first to remove any executables or object files).

The hand-in date is **25 Feb 2025, 4 pm**. Late submissions (within 24 hours of this deadline) will be accepted but will be capped at the module pass mark (i.e. 40%).

If your program does not meet all requirements, then please submit what you have written by the deadline.

Assessment

Your coursework will be marked on a number of factors:

- Does the program work? Does it meet all requirements? Has it been tested extensively? (60%)
- Program design and appropriate use of C++ language features, e.g. control structures, functions, data types, etc. (30%)
- Use of comments, indentation and variable/function names to make code easy to understand (10%)

The overall mark for this coursework will make up 10% of your total mark for this module.

This is an individual assignment. You are not permitted to work together with any other student. Note that general discussions about design decisions and/or coding strategies are permitted, and such discussions can be a useful learning experience for you. But you should not, under any circumstances, share details of designs or code.

Code generated using AI assistants, etc. is also not permitted. While these tools may (potentially) be useful in your learning to help clarify certain concepts, this is only true if they are used very cautiously and sparingly. Over-reliance on these tools to give you ready answers will inhibit your learning by preventing you from working things out for yourself. **This will result in failure in the final exam** where you will have no access to these tools (note that the final exam accounts for 60% of the overall marks on this module). We therefore strongly discourage the use of any form of AI assistants – in our opinion (and that of many others), these are detrimental to your performance on the course.

If you have any questions about this coursework please contact: J-Donald Tournier (jacques-donald.tournier@kcl.ac.uk).