Homework Assigment 4: Modelling pandemics

Scientific Software / Technisch Wetenschappelijke Software 2020

Kobe Sauwens

December 19, 2021

Number of hours spent: 50? I don't know, but I wanted this to be much better than the previous assignment, so a lot.

General

(G.1) Which important concepts, `C++`-specific syntax, or tools from the lectures and exercise sessions did you use? Where? Mentioning four concepts suffices. For each concept, limit your discussion to 3 lines. You can forward reference to your answers for the following questions.

- I used templates troughout my project to make my code general in the sense that it can work with different types of ODE and or precision. question (I.2)
- Lambda expressions.
- Passing references wherever I can.
- I used `.assign()` instead of the overloaded `=` to avoid unnessecary allocations.
- Macro's to erase my print statements.
- VSCode and the VSCode debugger and C/C++ plugin
- · · ·

(G.2) **Optional:** Are there changes or improvements you wanted to make but where unable to implement due to lack of time? If yes, briefly describe these.

I might have forgotten passing a reference here and there to avoid pass by value. There is an error in the backwards Euler which I can't find, it should not be difficult to find but my time is up. It works for simulation 1 but not for the rest. I would have loved to have time to run my code with Valgrind to see where I can be more memory efficient.

(G.3) **Optional:** Are there other general comments you want to make?

Part I: Simulating the pandemic

(I.1) **Optional:** Have you made any modification to your IVP solvers based on the feedback of the second homework?

Made a small adjustment to the jacobian, there was a misplaced bracket on the value for (2,1), which led to subquadratic convergence.

(I.2) Your IVP solvers must now accept more general ordinary differential equations. Which C++ features did you need to use to achieve this extra flexibility? What are the advantages of this approach? (max. 3 lines)

I used templates to achieve this, the advantage is the templating will happen at compile time, the code will be created for all the possible combinations of parameter types so there won't be a decrease in speed. Which is much better than using polymorphism.

(I.3) For `simulation2.cpp` it is asked how can you pass the differentialequation to your IVP solver? Discuss all possibilities. (max. 10 lines)

- A regular function
- Lambda expressions
- Functors using a struct or class.

(I.4) How would you write an IVP solver in Fortran that accepts more general differential equations? (max. 1 line)

External procedures

(I.5) Compare your Fortran code and your `C++` code.

(a) What `C++` specific features did you use? (Keep it short! A list of features suffices. max. 2 lines)

Templates, structs, classes, expression templates(implicitly via ublas), typedefs, metaprogramming

(b) Are there features that you used in Fortran but that are not available in `C++`? (it short! A list of features suffices. max. 2 lines)

Native matrix en vector support, intent (in/out), real subroutines instead of functions returning void

(c) Are there design decisions that you have to take into account when working with `C++` that you do not have to make (or can not make) in Fortran? (max. 6 lines)

Templated methods should be in header files, functions should be defined in the code before you use them.

(d) **Optional:** Was it easier to implement the functionality in Fortran or in `C++`?

In my opninion `C++` was easier because you can declare something wherever you want which is practical and you can use typedefs so you don't have to use the kind parameter all the time. I have written `Java` and `C` before so the syntax was more familiar, also VSCode is more tailored towards `C`

(I.6) In `simulation2.cpp`, how did you avoid using an explicit for or while loop when evaluating the right-hand side of the differential equation? (max. 2 lines)

```
template< class ForwardIt, class Generator >
void generate( ForwardIt first, ForwardIt last, Generator g )
```
This funtion applies a certain function `g` to all elements in an iteratable type, in this case a ublas vector.

(I.7) In `simulation2.cpp`, how did you avoid using an explicit for or while loop when filling the vector with the initial values? (max. 2 lines)

```
template< class ForwardIt, class T >
void iota( ForwardIt first, ForwardIt last, T value );
```
This function fills any iteratable type in range $[first, last[$ with the values with sequantially increasing values evaluating `++value` for every next element

(I.8) **Optional**: Are there other design decisions you want to mention?

Verify your code for the SIQRD model

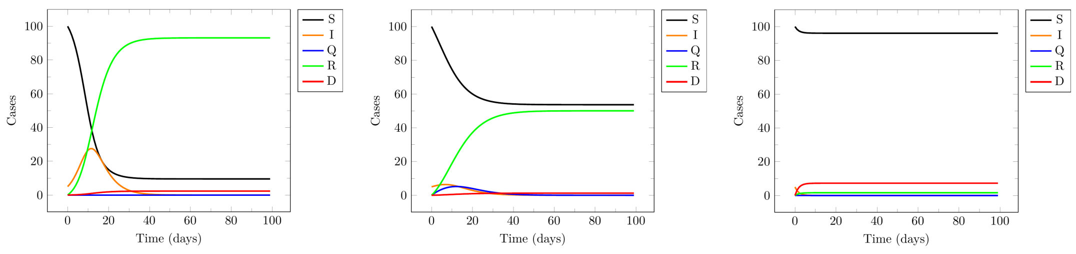


Figure 1: from lef to right: fwe_no_measures, bwe_quarantine, heun_lockdown

Verify your code for the general ODE

I calculated the solution for $f'(x) = -10x^3$ using Wolfram alpha. This tool gave me as solution:

$$f(x) = \frac{-1}{\sqrt{(c_1 + 20x)}}$$

which led me to believe the c_1 is the constant we add for the other equations. My result seem to match this solution.

Part II: Parameter estimation from observations

(II.1) **How did you optimize the performance (execution time and memory usage) of the parameter estimator code?**

(a) What part of the code has the most influence on the execution time (and therefore needs to be implemented as efficient as possible)? (max. 1 line)

The ODE solvers, because each LSE calculation has to solve over and over for new parameters. It tried to do hessian approximation in one line but the compiler complained so I split it up in small steps sacrificing some memory, I don't know how to fix this.

(b) How did you avoid unnecessary memory usage or copies? (max. 4 lines)

By using references and `assign()`. The observations are all in the LSE class so only a fixed amount of memory is needed.

(c) What tools did you use to assess the performance of your code? (max. 4 lines)

I used gprof, I also measured the speed using time and the `std::clock()`

(d) Are there other things you have done to improve performance?

I used the `-DNDEBUG flag`, to improve the speed of UBLAS and also my code because I put my print statements inside an `#ifdef DEBUG ... #endif` construction, I also declared variables outside loops as much as possible.

(II.2) **Flexibility.** How can you switch between the solvers for the initial value problem? (max. 3 lines)

Cange the module variable inside estimation1.cpp at line 52 to `EULERF`, `EULERB`, or `HEUN`

(II.3) **Verification.** How did you verify the different parts of your implementation while coding? What tools did you use to debug your code? (max. 6 lines)

I used the VSCode debugger with the json file from the excercise session. I also used print statements as ublas vectors and matrices were annoying to display properly while debugging. First I implemented a barebones BFGS with fixed step size, I debugged this implementation until I had a nice fit on the data I then implemented the line search, which I find difficult to test.

(II.4) What value did you use for ϵ in the finite difference approximation? Why?

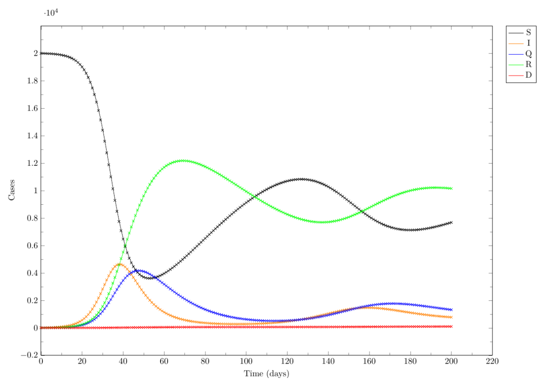
I used $\epsilon = 10^{-8}$ according to my observations from assignment 2 this seemed to produce the fastest and most accurate results. In the course Optimization we saw that $\sqrt{\epsilon_{mp}}$ with ϵ_{mp} the distance between 1 and the next bigger floating point number is a good ϵ value for finite difference derivative approximation. For IEEE double precision floating point numbers this is about $2.22 \cdot 10^{-16}$ so $\sqrt{2.22 \cdot 10^{-16}} = 0.000000014899664$ which is a bit bigger than 10^{-9}, so a fine choiche in my opinion.

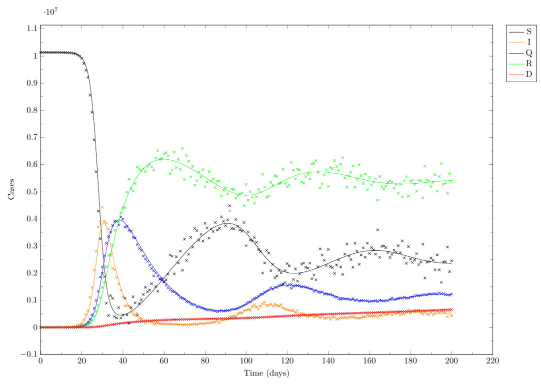
(II.5) Which modifications would you need to make to your code to add another optimization algorithm? (max. 5 lines)

I would have to rewrite the code in the `BFGS.hpp` header but assuming the cost function stays the same that would be the only file that has to be changed.

(II.6) **Optional:** Mention the difficulties you encountered while implementing this part. (max. 10 lines)

Sadly there is still a small error in the parameter estimation using the eulerbackwards method, I was not able to find why. The hessian approximation gets overwritten as one could forget :').





Verify your code

Extra questions

1. What C++-features did you use to achieve this?

2. Did you make other improvements? How big is their effect on the execution time, the number of calls to the IVP solver or the memory usage?