Homework Assigment 4: Modelling pandemics

Scientific Software / Technisch Wetenschappelijke Software

Clémence Duplat r0977801

December 20, 2023

1 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.

- Use of headers files and sources files to organize my code in the best way. I used endif in the beginning to be sure it hasn't been already declared elsewhere.
- Lambda expressions: used for creating inline instaces of my derivatives and Jacobians and such that they can be passed in argument when I use my IVPs (also called functors: functions as arguments)
- Standard template library: std, to avoid loops
- ublas (more optimized) boost for matrices and vectors.

(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.

Yes a lot, I had a lack of time due to a huge amount of homework those last weeks. I tried to finish part 2 during hours but I stayed stuck at this part. I then started to try optimize part 1 by making it Object oriented in my SIQRD model but I didn't had the time to entirely finish it, so it is incomplete. If I had time, I also would had create a template variable instead of using i.e. *vector¡double¿*. In addition, my code could be better arranged by adding namespaces, more classes, ...

(G.3) **Optional:** Do you have any other general comments?

I know my homework is not good, I want to apologize for it.

2 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?

Yes, I changed the way I defined the parameters such that only the SIQRDparameters are defined generally

(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 Function pointers and std:: function to define my derivatives and Jacobian as function pointers. Thanks to that I am able to use a large variety of Jacobian and derivatives based on the EDO I want to use. I also used ublas for matrix and vectors in order that my EDO's can have different sizes.

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

The first straightforward solution is to pass the EDO as a direct function pointer. The IVP solver will then use the pointer to this function to evaluate the EDO at the required points. However, this is not very optimal so I didn't implement it that way. The second solutions is the functors. The idea is to define a class with a operator that will define the EDO. This is a good way to proceed but I didn't implement this one. The third solution is to use lambda expressions for my EDOs and pass it to the IVP solvers. I choose this way to proceed.

(I.4) 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)

Functors, lambda expressions, STL algorithms, auto reference, namespaces, boost libraries

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

We see that in C++ they are way more memory leeks. Therefore, we have to be very careful on how to manage this memory. I also know a very useful technique in C++ is to use generic programming, work with different data types, for flexibility. However, I was too short in time to implement that. Also, C++ begins to iterate at 0 and Fortran at 1. The use of classes make it posible to make my IVP way more general than in Fortran

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

Yes, the implicit loops over the arrays. I used the STL algorithms like transform to avoid those loops. That was not possible in Fortran.

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

I found it easier and more straightforward to implement in Fortran but I know that if we have very difficult implementations C++ will be way more efficient.

(I.5) 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)

`std:: transform`: used to avoid explicit loops when evaluating the right-hand side of my EDO. It applies a given function(lambda function) to each element of a vector by iterating over them and stores the result in another vector.

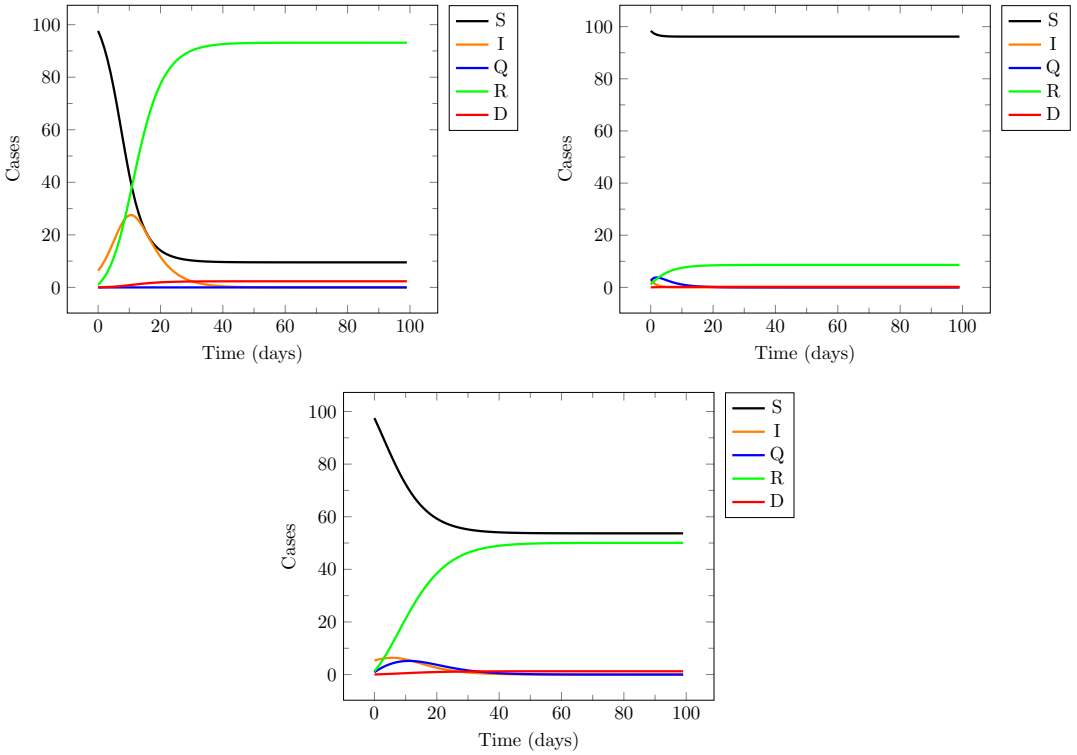
(I.6) In `simulation2.cpp`, how did you avoid using an explicit for or while loop when filling the vector with the initial values? (max. 2 lines)

> `Fstd::  iota`: will fill a range with sequentially increasing values, starting from a specific value.

(I.7) **Optional**: Are there other design decisions you want to mention?

2.1 Verify your code for the SIQRD model

Here are the `plot.tex` based on `fwe_no_measures.out`, `bwe_quarantine.out` and `heun_lockdown.out`. We see that it is correct, as it represent the plots of assignement 1. Note that I had to manually change the value of delta.



2.2 Verify your code for the general ODE

We can verify it by comparing to the real analytical solution. We see that it is correct.

3 Part II: Parameter estimation from observations

My code doesn't work for this part so I wasn't able to answer the questions. Indeed I have 1 iteration in my BFGS algorithm, leading to the same optimal paramaters as the initial ones. However, I answered with theoretically information I should have done.

(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 efficiently as possible)? (max. 1 line)

The most influential part on the execetuion time are in general the iterative loops. Here in particular, the iterative loop for the BFGS algorithm.

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

I have declarer the variables as much as possible outside the loops. I also used *assign* to transfer the values from one vector to another more efficiently. Thanks to that I will avoid redundant memory allocations and it will be usefull when I will have very very large problems. I also used ublas routines to compute the norms.

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

I hadn't the time, because my code doesn't work, but I would use Valgrind to analyze the execution time and see if I can make improvements.

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

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

I could use a functor pointing to the specific IVP. However, I didn't focus on this part, I firts tried to solve the problem with my code.

(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)

The most common manner to verify the final result is to plot it and compare it to an analytical solution. While coding, I tried to print different values to see where my problem could have occured.

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

I used a value of $1e-5$ for my espilon. I think it is good choice because it offers me a balance between accuracy and numerical stability. I don't think this is the cause of my problem.

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

I would define a function with the same signature as the BFGS and use the same parameters to compare the results as clear as possible.

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

3.1 Verify your code: `estimation1`

Include the plots generated by `plot_predictions.tex` based on `observations1.in` and `observations2.in`: Due to the fact that the parameters doesn't change, I wasn't able to have good plots for this part.

3.2 Verify your code: `estimation2`

For `observations1.in`, compare the different methods for solving the ODE, while using the BFGS method to compute the minimizers. Do the obtained parameters differ? Why? What about execution time?

Didn't had the time to make that but I the theoritically answer should be. With Forward Euler give the same parameters with almost the same number of iterations. For Backward Euler, it could be that the stopping criteria are not met, leading to the maximum number of iterations. However, the answer should remain almost the same.