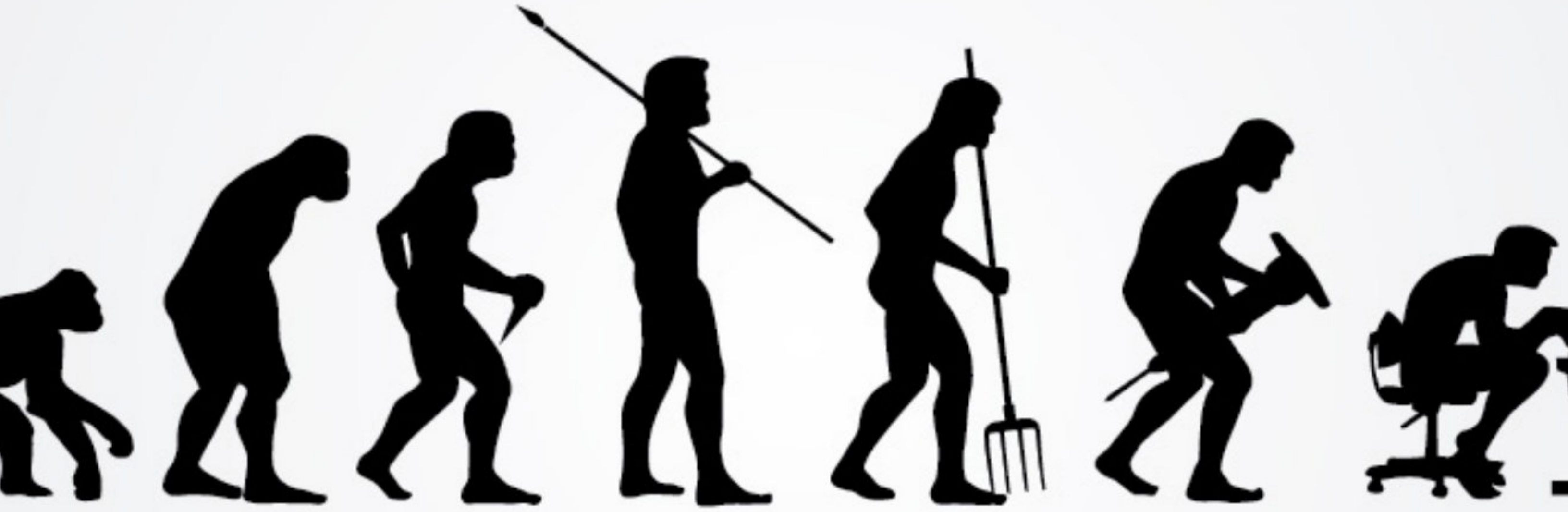
DTU Compute
Department of Applied Mathematics and Computer Science

02122 Software Technology Project

Detecting Structural Breaks in Time Series via Genetic Algorithms

Group 3: Markus B. Jensen, s183816



Contents

1 Introduction

Detecting structural break points is an essential tool in analyzing time series. Structural break points are points on a time series where the pattern of the time series measurements changes in the amplitude. A simple example is shown in figure 1a where the break point is easily detected at $t = 500$. Structural breaks are where the pattern of a time series changes.

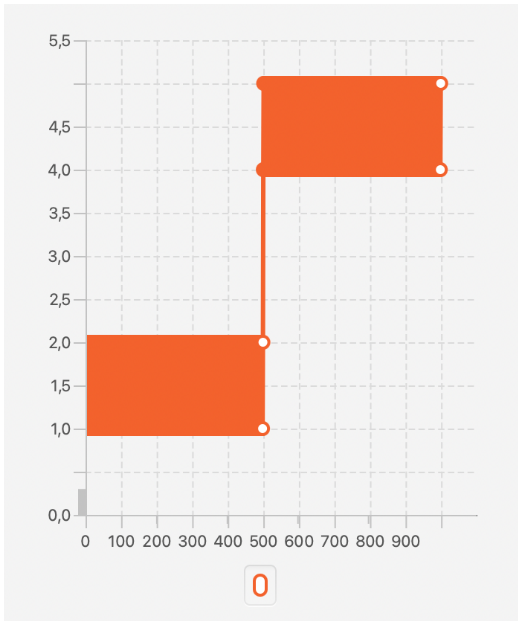
In the example above, the break point is easy to detect. As usual, the real world is far more messy. In figure 1b, which shows the number of patients hospitalized in Denmark due to Covid19, the break points are less obvious. This is where detecting break points is important: Being able to recognize when the pattern changes, so that a pandemic does not run amok or looking for fluctuations in the stock market.

Please find another example than the stock market

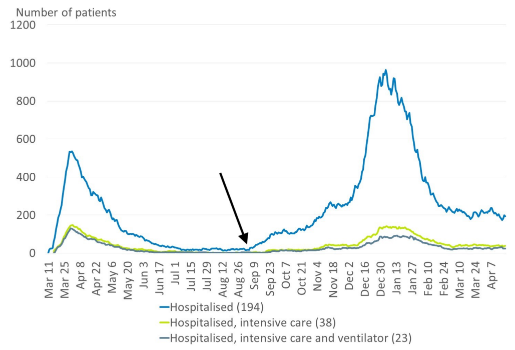
In this project, the break points are found by using a so-called genetic algorithm. This algorithm mimics natural evolution: A number of individuals (solutions that indicate the positions of break points) mutate and mate during generations where survival of the fittest is the rule. In the end, the best solution will (hopefully) have found all the break points.

This project will focus on implementing the algorithm in the programming language Java and make a simple application. The application will give the user the ability to tweak some of the algorithm's parameters and see the break points directly on the time series graph. Further improvements to the speed of the algorithm will also be implemented "under the hood".

Make the left figure more pretty!



(a) Graph with break point at $t = 500$



(b) Number of people hospitalized due to Covid19 in Denmark. The arrow indicates a break point. Source: the Danish regions

Figure 1: Two figures illustrating break points in time series

2 Goals

A prioritized list of the functional goals for the project is listed below. While adding a lot of features seems very appealing, I will in this project focus on usability, stability, and speed. This leads to a somewhat conservative list of potential features, but the quality of the final product will reflect this decision.

1. Implement the algorithm using the rectangle-method from [Doerr et al., 2017] in Java.
2. Visualize two-dimensional time series graphs together with the rectangles produced by the algorithm in a simple graphical user interface (GUI). This GUI will allow user to load a time series data file and see the output of the algorithm on the time series. The user will also be able to tweak certain parameters of the algorithm.
3. Make the algorithm more flexible by allowing the user to alter the values of algorithm-parameters in the GUI.

A previous goal was also to implement further fitness functions (see explanation for *fitness functions* in section 3). Since this project ended up being a one-man-project, this goal was removed. In stead, a priority is to design the project so that it is easy to implement other fitness methods. This will be discussed further in section 4.

2.1 Non-project goals

As for non-project goals, there are a few:

- Learn the Maven project structure for Java. While previous courses have dealt with Maven a little bit, it has never been fleshed out. It seems to be a structure that is widely used and thus a good system to learn.
- Working on big project. This is the first big project I am working on. A big focus here is on the project management, report writing workflow and keeping track of sources in a bibliography. Especially the report writing workflow can become crucial, as I have a tendency to postpone it to the very last minute. Here, I will make it a part of my weekly work.

My ambition level is quite high; I am a perfectionist to the core. While I will attempt to keep the perfectionism to a minimum, I like working on bigger projects and making it work well. This will probably result in me working a lot on this project, purely because it will be fun, and I like improving my less-than-optimal solutions. I am aware that this course is only 5 ECTS points and will thus keep track of my hours spent on the project as to not overdo it.

3 Terminology

The terminology used in this report is specified below. The terminology differs a bit from [Doerr et al., 2017]. It takes inspiration from other sources ([Thede, 2004, Point,]) and follows a more biological narrative.

Individual An individual consists of a solution string. It is one possible solution to the problem.

Genome An individual consists of a genome. A genome is an array of genes. The genome *is* the solution string.

Allele A gene's value is called an allele. In this project, the allele is thus responsible for holding the information of whether or not a certain gene is a break point. An allele is the value at a certain gene in the genome/solution string.

Population A group of individuals.

Fitness function A function that measures how well the solution from an individual fits with the data. This can be any function relevant for a particular problem.

Parent and offspring individual Since genetic algorithms mimic evolution, procedures must include parent and offspring individuals. A parent individual is a parent to the offspring individual, meaning that any procedure must take one (or two) parent individual(s) and the procedure creates an offspring.

Crossover (Procedure) Crossover-procedures takes two parent individuals and creates an offspring from the genomes of the two parents. This project will incorporate *one-point crossover* and *uniform crossover.*

One-point crossover extracts the genes in the interval 0 to a random gene $i-1$ from the first parent. Then, it appends the genome with the genes i to the last gene $n-1$ from the second parent. Thus, the offspring's genome consists of the first i genes from parent one and $n-i$ last genes from parent two.

Uniform crossover is more simple. For each gene in the offspring's genome, there is a fifty-fifty chance whether it will be the gene from parent one or the gene from parent two.

Mutation (Procedure) Just like the corona virus, the genome of an individual can mutate. It means, that for any gene, there is a random chance that the allele will change. In this case, the change will be either creating break points or removing them.

4 Problem analysis and design

The main challenges are already hinted at in section 2.

5 Implementation

6 Test

7 Project Management

8 Conclusion

References

[Doerr et al., 2017] Doerr, B., Fischer, P., Hilbert, A., and Witt, C. (2017). Detecting structural breaks in time series via genetic algorithms. *Soft Computing*, 21(16):4707–4720.

[Point,] Point, T. Genetic algorithms.

[Thede, 2004] Thede, S. (2004). An introduction to genetic algorithms. *Journal of Computing Sciences in Colleges*, 20.