Jobfeed alarm system

Applying change point detection and a particle filter to a random walk

M. Herbrink

Jobfeed alarm system

Applying change point detection and a particle filter to a random walk

by

M. Herbrink

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday April 9, 2018 at 01:30 PM.

Student number:	4237072	
Project duration:	May 22, 2012 – April 1, 2013	
Thesis committee:	Prof. dr. ir. G. Jongbloed,	TU Delft
	Dr. ir. G. N. J. C. Bierkens,	TU Delft, supervisor
	Dr. V. Jijkoun,	Textkernel, supervisor
	Dr. D. Kurowicka,	TU Delft

TU Delft

Abstract

Jobfeed is an online database containing all vacancies posted on the internet. The database obtains this data through a process called spidering. The data is collected by visiting web pages and extracting vacancies from these pages, using machine learning techniques. In the process of spidering, errors can occur. To ensure the quality of the Jobfeed data, an old alarm system is in place to detect possible errors in the spidering process. This alarm system triggers alarms whenever the spidering of a website seems to be malfunctioning. The old alarm system is not performing well, with a precision of 0.028 and recall of 0.18. The goal of this thesis is to develop a new alarm system, better capable of triggering alarms. In order to do so, the data has been modelled as a hidden Makrov model. The core of the alarm system consists of a particle filter, a sequential Monte Carlo method that is able to judge whether the process is showing behaviour that indicates something is wrong. A series of methods to detect a change in distribution of a time series, or a change point, have been developed and tested to detect a change in a random walk with drift. The initial values for the particle filter were partly estimated by applying the best performing change point detection method to find the last stable segment in the historical data. The data in this stable segment can be used to estimate initial values for the system. Sequential decision making theory is used to decide whether an alarm should be triggered or not. The new alarm system has a precision of 0.74 and recall of 0.96, which is a big improvement compared to the old system used.

Preface

When I started my masters in Applied Mathematics I was already interested in the application of statistics and probability theory to tackle issues we are facing in today's data dependent world. I am very happy I got the opportunity to use the theoretical knowledge that I obtained in the course of my studies to contribute to a more reliable Jobfeed. The one thing I am particularly proud of is that the system developed in this thesis is implemented and will be used in practise, which makes me feel the effort put in this thesis was not just for me to graduate, but also benefits others.

I would like to thank my supervisor at the TU Delft, Joris Bierkens, for his support and input. He has been a great help and partner to brainstorm with, without our meetings the project wouldn't have turned out as well as it did. Also a big thank you to my daily supervisor at Textkernel, Valentin Jijkoun, for his help introducing me to the machine learning world and his valuable insights. Special thanks goes to Karlijn Dinnissen and Jan Priebs, who were so kind to help me perform a test by annotating by hand. Last, but not least, I would like to thank the Jobfeed team and all people working at Textkernel for the great atmosphere they created; this made my time at Textkernel truly enjoyable.

M. Herbrink
Amsterdam, April 2018

Contents

List of Figures

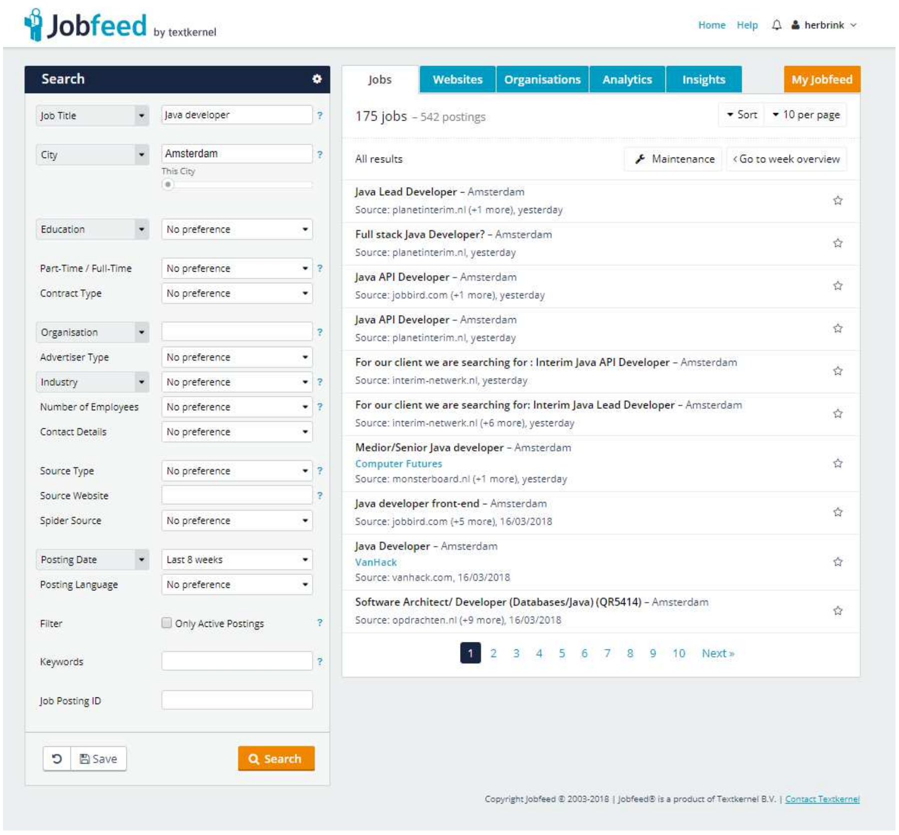
List of Tables

1 Introduction

The impact and importance of data on our society have grown rapidly over the past years. Where only twenty-five years ago it was something special to have access to the internet, there now is a big industry of tech-companies doing business in this field. In many big companies, also traditional ones that do not have their main focus in the tech industry, terms like "big data", "machine learning" and "artificial intelligence" can be heared on a daily basis. A lot of money is involved; in 2017 the estimated revenue in the big data analytics market was $150 billion and is expected to grow to $210 billion in 2020 [1]. One of the challenges when handling data is to obtain as much relevant data as possible, while ensuring the data quality. Since we are dealing with big data, it is not possible to monitor the process of gathering data manually; therefore algorithms have to be used to check for possible errors in the database. Here artificial intelligence and machine learning come in; the process of detecting errors in a database can be improved by learning from past assessments. This thesis focusses on building such a system, that is able to trigger alarms whenever problems seem to occur in the process of obtaining data.

1.1. Jobfeed

This thesis has been executed in partnership with Textkernel, an IT company that specializes in machine intelligence for matching supply and demand in the job market. They have developed several tools to better search and match candidates and vacancies. Besides, Textkernel also developed a tool that can be used to analyse the job market: *Jobfeed*. It is an online tool and database that can be used to analyse and search vacancies posted online and is currently available for over 8 countries, such as The Netherlands, Belgium, Germany, France, the United Kingdom and the United States. The Jobfeed database contains vacancies posted on websites from for example job boards or direct employers. The database can be searched using various search criteria, to find a suitable vacancy, or to look for possible trends in the job market. The search results can be shown based on jobs, websites or organizations. Also a tab named insights is available to see a graphical representation of the search, showing what job boards the vacancies came from, the profession, education level and more. A screen shot of the result of searching on Java developer in Amsterdam is shown in figure 1.1, where the search bar is shown on the left and the results of the search on the right.



Job Title	Java developer
City	Amsterdam (This City)
Education	No preference
Part-Time / Full-Time	No preference
Contract Type	No preference
Organisation	
Advertiser Type	No preference
Industry	No preference
Number of Employees	No preference
Contact Details	No preference
Source Type	No preference
Source Website	
Spider Source	No preference
Posting Date	Last 8 weeks
Posting Language	No preference
Filter	Only Active Postings
Keywords	
Job Posting ID	

175 jobs – 542 postings

All results

Java Lead Developer – Amsterdam
Source: planetinterim.nl (+1 more), yesterday

Full stack Java Developer? – Amsterdam
Source: planetinterim.nl, yesterday

Java API Developer – Amsterdam
Source: jobbird.com (+1 more), yesterday

Java API Developer – Amsterdam
Source: planetinterim.nl, yesterday

For our client we are searching for : Interim Java API Developer – Amsterdam
Source: interim-netwerk.nl, yesterday

For our client we are searching for: Interim Java Lead Developer – Amsterdam
Source: interim-netwerk.nl (+6 more), yesterday

Medior/Senior Java developer – Amsterdam
Computer Futures
Source: monsterboard.nl (+1 more), yesterday

Java developer front-end – Amsterdam
Source: jobbird.com (+5 more), 16/03/2018

Java Developer – Amsterdam
VanHack
Source: vanhack.com, 16/03/2018

Software Architect/ Developer (Databases/Java) (QR5414) – Amsterdam
Source: opdrachten.nl (+9 more), 16/03/2018

Copyright Jobfeed © 2003-2018 | Jobfeed® is a product of Textkernel B.V. | Contact Textkernel

Figure 1.1: Search bar and search result from Jobfeed, for a search "Java developer" in "Amsterdam".

1.2. Goal of the thesis

In this thesis we will be focussing on the way the Jobfeed data is obtained and how we can detect issues in this process. The goal is to build an alarm system that is capable of detecting defects in this data gathering process. Such an alarm system is already in place, but due to the big amount of (false) alarms this system is only used to monitor vacancies coming from the 25 websites with the largest amount of vacancies. These alarms are monitored by the Jobfeed Data Quality team, that consists of two members at the moment. The new alarm system developed during the course of this thesis may be used to monitor websites that have less vacancies, but are still a valuable source for the Jobfeed database; in practice we also want to monitor websites that are on place 26 up to 100 when it comes to the amount of vacancies.

1.3. Structure of the thesis

The research done in this thesis consist of several parts that are related to each other, as shown globally in figure 1.2. The main computation is done by a sequential Monte Carlo method called a particle filter. The particle filter uses the data to calculate the probability that the data gathering process is not functioning properly. This is passed on to a sequential decision making procedure that can decide to trigger an alarm, which is the output of the system. After this, the user has to decide what to do with this alarm; he can either do nothing or decide to reset the particle filter. The particle filter needs initial values to start its computation. These initial values are partly determined using data from the past. For this change point detection has been used to split the past data up in segments in which the data shows similar behaviour; the last segment is used to estimate initial values that are passed on to the particle filter.

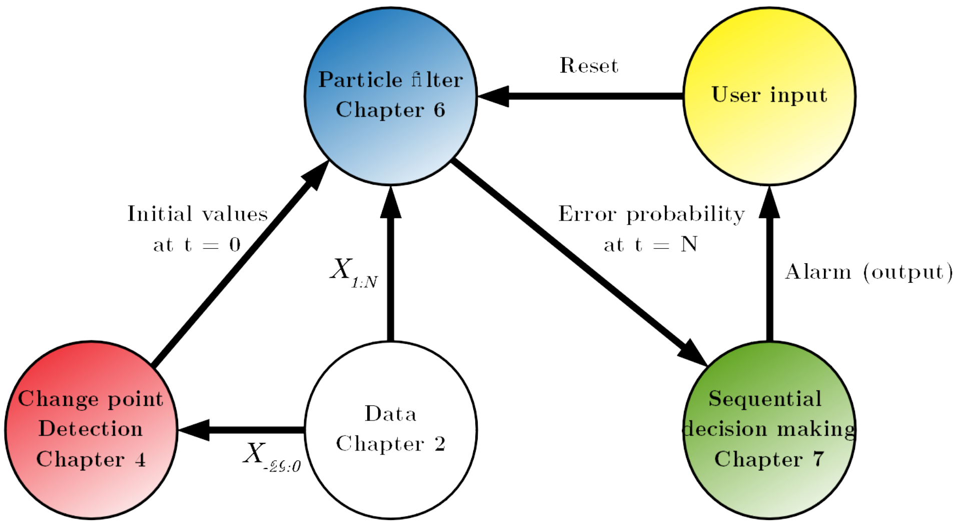


Figure 1.2: Graphical representation of the alarm system.

First, we will first describe the process of gathering the Jobfeed data and describe what possible problems can occur in chapter 2. After this, some preliminaries on statistics and machine learning are given in chapter 3. Next, we will focus on detecting change points and apply this to a random walk with drift in a simulation study in chapter 4. After this we will describe the alarm system developed by first explaining the assumed model in chapter 5 and in chapter 6 a numerical method called a particle filter is described that can be used for inference in a hidden Markov model. To decide at what time to trigger an alarm, a sequential decision making procedure is described in chapter 7. A test has been done in chapter 8 to check the performance of the newly developed alarm system and conclusions are stated is chapter 9.

2 Jobfeed data

In order to get vacancies from the internet into the Jobfeed database, a process called *spidering* is used. This entails a program or a *spider* visiting web pages and looking for vacancies. If the spider comes across what he believes is a valid vacancy, it will save the attributes pertaining to this vacancy, such as job title, location, required educational level, employer, etc. These will be added to the Jobfeed database. The spiders can be divided in two groups:

1. **Scripted spiders**
   These spiders are scripts written for one website in particular. The spider knows where the attributes of the job are located and how to go through all vacancies on the website.

2. **Wild spiders**
   These spiders crawl a big part of the internet, looking for vacancies. They are designed to work for all websites, but the results are less reliable than the vacancies from the scripted spiders and therefore these spiders aren't used on websites a scripted spider is available for.

Not all vacancies detected by the spiders are directly imported into the Jobfeed database. The job openings found are further filtered to remove pages that are not actual vacancies. An example of such an error is that a wild spider often identifies an index page with links to vacancies as a vacancy itself; this should not be imported in the database. At this point we also check for duplication; the vacancy should only be imported once. In case it is posted at multiple locations, it should still be imported as a single vacancy.

The scripted spiders visits the website once every day. Here it will go through all job offers on the website. The output of the spider consists of two parts:

- URL files, indicating the pages the vacancies can be found. For every job opening the spider finds, it will return a URL file. It is necessary to do so every day, because this way we are able to tell when a vacancy is no longer active and should be removed from Jobfeed. In case the URL is not yet in the jobfeed database, an XOS file will be generated.

- XOS files, containing all information available about a vacancy on the website. The XOS files represent the vacancies that are new and have to be added to the Jobfeed database.

The XOS files found by the spider are processed to determine whether they are actual vacancies and to remove duplicates. After this, the new vacancies are imported into the database. The vacancies in the database of which the URLs are no longer found by the spider are expired and removed from the database. In case a vacancy has been expired, but later the URL of the vacancy is found by the spider again, the vacancy will be revived.

In some cases, the spider does not have enough time to finish the spidering before the day is over. When this happens, a different type of spider is activated, a *manual flush* spider. This spider only records the URL files and is therefore a lot faster than the regular spider. This way all URL files are imported in the database, even if not all XOS files are found. Spidering can take quite some time, because most websites can only be visited with a limited frequency. If a website is visited too frequently we will be blocked by the website. Typically a website can be visited once every three seconds.

2.1. Norm URL

When visiting a web page, the URL (Uniform Resource Locator) contain information about the location of the page. A URL consists of several parts; a host name, a path and sometimes a query. In this query you will often find GIT parameters, containing information about the current session. If you would use Google to search for "mathematics", this keyword will be included in the query as GIT parameter ?q=mathematics. Not all of these GIT parameters are relevant for us; often a session ID or the browser used is in it as well. In the example of the Google search, both of these URLs will result in the same page:

```
https://www.google.nl/search?q=mathematics&rlz=1C1AWFC_enNL754NL754&oq=mathematics&aqs=
chrome..69i57j0l5.1381j1j7&sourceid=chrome&ie=UTF-8

https://www.google.nl/search?q=mathematics
```

In the second URL, all unnecessary GIT parameters were left out.

We will call the URL without all these unnecessary parameters the *norm URL*. Note that the norm URL does not necessarily have to be a working URL; it the shortest way to identify an internet page that we can reproduce when coming across this page again. These norm URLs will be used to check the presence of a vacancy in the Jobfeed database. If it is already in the database, the vacancy doesn't have to be imported. In case we would use the regular URL for this, we would identify each vacancy as new, because the GIT parameters are usually different for every session and if they are not removed the new URL doesn't match the URL in the database.

2.2. Heuristics

To improve the stability of the data in Jobfeed, several heuristics are used to correct for possible errors in the data found by the spiders.

1. If more than 25% of the vacancies should be expired according to the spider, none of the vacancies of this website currently in Jobfeed will be expired. If this happens, it will be saved as an "expiration problem". This will be done for a maximum of five day in a row; if the URL files of the vacancies are still not found after five days the vacancies will be expired anyway.

2. In case a spider tends to overlook vacancies, a *grace period* between expiring and the lost of the URL file can be implemented. A grace period of length k means the vacancy will only be expired if the URL file has not been found for k days. Not all websites have such a grace period; usually it is only used for unstable spiders.

3. If more than 250% of the maximum of database imported in the last two weeks is imported on one day, the vacancies will not be imported.

Other rules specifically defined for each website are used as well, mostly to filter out vacancies that have been imported by the spider but shouldn't be imported into the Jobfeed database, because they aren't an actual vacancy or because they contain too little information.

The information contained in de XOS file is just the HTML page the spider found. From this page, all relevant parameters have to be extracted. Textkernel has developed a tool named *Textractor* that is able to abstract these parameters from the page. However, because we often know the location of each parameter for the website that are spidered using a scripted spider, additional *wrapping rules* are in state for most scripted websites, to improve the output quality of Textractor. Often companies use a so called *applicant tracking system* (ATS) in their HR department to process and view the data of applicants. In case the source uses such an ATS to keep track of their vacancies, we can use standard wrapping rules for this ATS.

Apart from wrapping rules, often scripted websites have *site rules* in place. These site rules help to identify and to locate vacancies. The URL of a vacancy often has a specified form, for example:

```
http://company.com/vanancies/vacancy_id
```

A site rule could be that all vacancy pages start with http://company.com/vacancies, or have the word "vacancies" in it. Likewise we can also reject all URLs that take a preset form or contain some word or character indicating we're not actually dealing with a vacancy. If for example on a job board the vacancies from one particular employer are never uploaded correctly, the spider can be forced to skip vacancies that have the name of this employer in their URL.

Another important thing the site rules are controlling is the norm URL. As explained before, this norm URL is used to identify each vacancy, to see whether it is already in the database or not. The site rules state how the norm URL is created from the actual URL.

2.3. Available data

The Jobfeed database contains a lot of parameters about the vacancies found by the spiders and wild-spiders. Users can use these parameters to search through the vacancies and analyze the job market. This data is the result of the spidering; however in this thesis we are interested in the functioning of the spiders themselves. To this end we have several statistics about the spidering available to us:

1. Spider found:	The amount of unique URL files the spider found that day.
2. Spider found cumulative:	The amount of unique URL files the spider found, plus the URLs that did not expire due to a set grace period in the expiration.
3. Spider accepted:	The amount of unique URL files the spider found that day after applying heuristics.
4. Spider imported:	The amount of new vacancies the spider found that day; these vacancies are imported as XOS files.
5. Database imported:	The amount of new vacancies import into the database, after cleaning and removing duplicates.
6. Expired:	The amount of vacancies that expired on that day.
7. Revived:	The amount of vacancies that were expired, but revived because their URLs have been detected again that day.
8. Total active:	The amount of vacancies listed as active for this website.

Furthermore, whenever an expiration error occurs, this will be saved in the database. The methods developed in this thesis have been applied on the spider accepted data, however other statistics about the spidering could be used as well.

2.4. Currently used alarm system

There already is an alarm system in place that is triggering alarms when a spider seems to be malfunctioning. This alarm system is only used for the top 25 websites that have the most vacancies. The system is actually a list of heuristics, describing boundaries for some statistics of the spider, such as its mean and standard deviation. The following procedure is used for each website:

1. For the last ten weeks, the amount of vacancies found within that week is saved as $n_0, n_1, ..., n_9$. Here n_i is the number of vacancies found by the spider in the $i'th$ week before the current day.

2. Take $\mu = \frac{1}{10}\sum_{i=1}^{10} n_i$ the mean amount of vacancies found per week of the last ten weeks and $\sigma = \frac{1}{10}\sum_{i=1}^{10}(\mu - n_i)^2$ the standard deviation assuming normality.

3. Alarms are triggered if:

    - $n_0 > 0$ and $n_j = 0$ for $1 \leq j \leq 9$
    - $\mu > 25$, $n_0 = 0$ and $n_j > 0$ for $1 \leq j \leq 9$
    - $\mu \geq 50$ and $|n_0 - \mu| > \frac{\mu}{2}$
    - $\mu > 25$ and $\frac{|n_0-\mu|}{\sigma} > 2$

This system only takes the weekly amount of vacancies into account. The performance of this algorithm is rather poor, as we will see later in section 8.2.

2.5. Problems and data

For the alarm system developed in this thesis, we will use the spider accepted data as input. This data is collected on a daily basis, so every data point represents the number of vacancies the spider accepted in one day. Some examples are shown in figure 2.1.

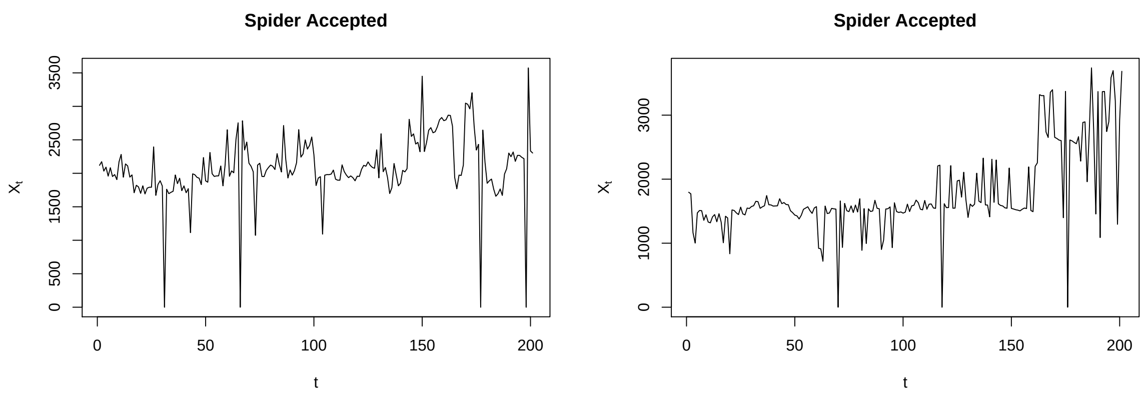


Figure 2.1: Examples of spider accepted data of two different websites.

There are several things that can go wrong in the spidering process. In this section we will indicate some of the issues that can pop up and how they can be recognized in the data. Note that this list is not complete. We will focus on the scripted spiders.

A possible source of a problem is the website that is being spidered. Examples are:

1. **The website is down temporarily**
   The spider will try to find vacancies, but the website is offline due to problems at the host. The spider will not return anything (not even a 0). The next day the website could be online again and everything will be back to normal.

2. **The website has changed**
   The spider will try to find vacancies, but something about the website has changed, which makes the spider gather wrong or incomplete data. In case the lay out of the website has changed, the spider might not be able to find the vacancies any more, which makes the spider accepted go to zero. It could also be that the spider does find the vacancies, but some of the parameters are missing, mixed up or false.

The problem could also be caused by incorrect settings or site rules. This is related to the website itself; if the website has been updated, the site rules will most likely be outdated. Some of these issues are:

1. **Norm URL is broken** The way the norm URL is defined is (no longer) correct for this website. The spider will find new URLs for all vacancies on the website and will try to expire all vacancies already in the database. The heuristics will only allow this to happen after five days. So, for five days the spider will import all vacancies it finds, making the total active go up, while the spider accepted stays approximately constant. Spider and database imported are both high. After these five days the spider will expire all vacancies and keep only those found on the fifth day. This process will repeat itself if the norm isn't fixed and we will see "triangles" in the data.

2. **Wrong ATS selected**
   In order for content to be picked up by the spider, the correct ATS has to be used. If the wrong ATS is selected, the content picked up by the spider will most likely be false.

There are other issues that may occur in the spidering process, that can be seen in the statistics of the spider. Note that if we observe troubling behaviour of the spidering data, this could be caused by a lot of issues and often we cannot see directly from the data whether the issue is in the website, the settings or in the spider itself. Some examples of observations in the data that suggest something is wrong are:

1. **Spider drops to zero**
   The spider still finds URL and XOS files, but they are both empty, so we do not import anything. Spider imported and spider accepted both go to zero. This could be because the spider is picking up wrong or empty pages. It could also be the result of a change in the website.

2. **Spider imported drops to zero**
   The spider does find the URL files, but not the XOS files of new vacancies. Spider imported goes to zero, but spider accepted will still be approximately the same as the day before. Here the spider could be broken, making it unable to find the XOS files. It could also be the start of a *dying spider*; the spider finds less vacancies every day and doesn't find any new vacancies any more. An example of the spider accepted data of a dying spider (that recovers at $t = 145$) is shown in figure 2.2

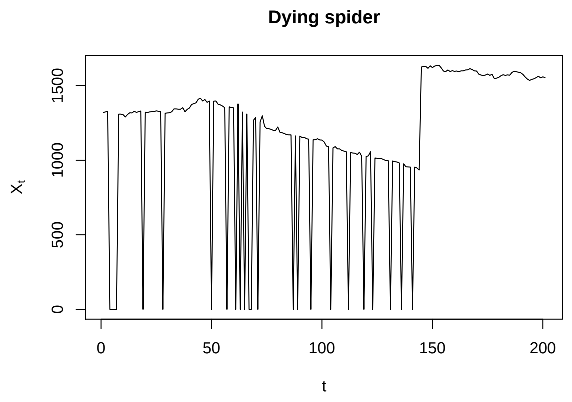


Figure 2.2: Example of spider accepted data of a dying spider, recovering around $t = 145$.

   Spider accepted will slowly go down and spider imported will be zero. The reason for this is often that the website has changed the way they process their vacancies, but the old vacancies remain the same; new vacancies will therefore no longer be detected by the spider.

3. **Sudden drop of spider accepted**
   The spider seems to find only a part of the vacancies on the website; the spider coverage could be incomplete. Because of this spider imported, revived and expired are too high and spider accepted is too low.

4. **Sudden rise of spider accepted**
   The spider suddenly finds a lot more URL and XOS files than before. Spider accepted and spider imported will increase drastically. This can be an indication something broke in the spidering process, but this can also occur when a spider was broken but has been fixed.

The unscripted spiders cause a lot of issues as well, especially since they are not build for one particular website. In this thesis we will focus on the scripted spiders.

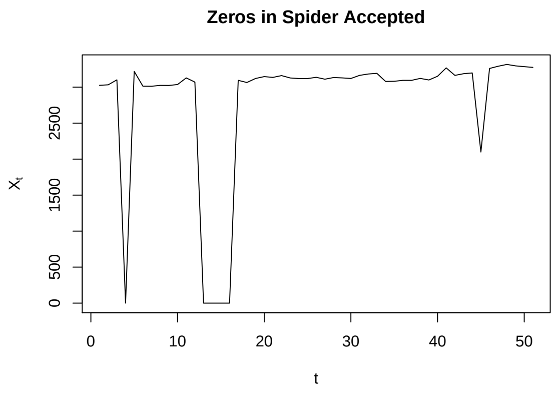
2.5.1. Focus in alarm system

The alarm system developed during the course of this thesis should preferably be able to detect all possible issues with the spider. Since however this is a complicated issue, not all of these problems will be incorporated in the alarm system. After discussing several examples of situations where the spider was malfunctioning with the Jobfeed Data Quality team, we decided the following behaviour of the spider accepted data will be focussed on:

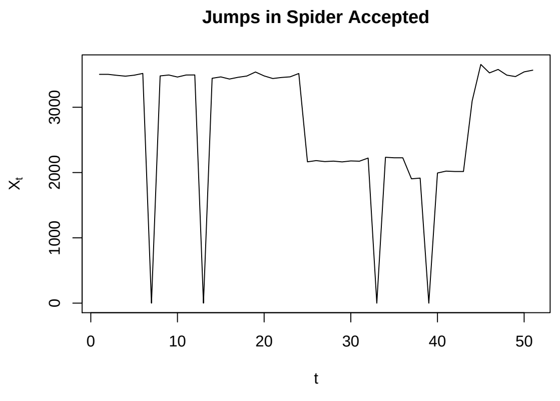
1. **The spider is zero**
   In this situation the spider could have dropped to zero or delivered the wrong content. Besides this, the website could be changed or down. Since a drop to zero occurs quite frequently, one day of a zero value should not be a problem. If the spider accepted is zero for more than one day, this is a problem and an alarm should be triggered.

2. **The spider makes a jump**
   In many cases in which the spider wasn't functioning correctly, the spider accepted data showed some kind of sudden drop or rise. This could be in the case of incomplete spider coverage (drastic decrease), a change of website or a drastic increase. A broken norm URL also shows jumps, but not always right away.

Examples of both situations are shown in figure 2.3.



(a) Zeros.



(b) Jumps.

Figure 2.3: Examples of spider accepted data in case of a longer period of zero and jumps.

The most important problems we are missing by focussing on these two are the dying spider and the case where no vacancies are imported, however the URL files are found. The last one cannot be observed in the spider accepted data and is therefore left out of the alarm system. Dying spiders are rare and can only be seen after observing for a longer period of time, which makes them less urgent than for example zeros or a sudden jump in the spidering data.

3 Preliminaries

In this thesis, several statistical methods will be used. Some basic statistical results will be summarized in this chapter. First the topic of time series [2] will be addressed in section 3.1. After this we will discuss how outliers can be dealed with using robus statistics or filters [3] in section 3.2. Then we will discuss Bayesian statistics [4] in section 3.3, followed by linear regression [5] in section 3.4. Last we will give some measures to monitor the performance of a system [6] in section 3.5.

3.1. Time series

A time series $(X_t)_{t\geq 1}$ is a set of data points indexed over time $t \in \mathbb{N}$, starting at time $t = 1$. We will denote this as X_t. A subset $(X_i)_{s\leq i\leq t}$ will be notated as $X_{s:t}$. The distribution function of X will be denoted as $f_X(x)$. The subscript will often be left out to clean up the notation.

We will define the *backshift operator* ∇ for which:

$$\nabla X_t = X_t - X_{t-1} \tag{3.1}$$

A lot of models have been developed to describe time series showing different types of behaviour. One common model is the AR(p) model. Here the time series X_t follows the recursion:

$$X_t = \delta + \sum_{i=1}^{p} \phi_i X_{t-1} + Z_t \tag{3.2}$$

Here Z_t is a white noise process; from now on we will assume $Z_t \sim N(0,\sigma^2)$ for some standard deviation σ. We also need a starting value X_0. A special type of AR model is the *random walk*; an AR(1) model with $\phi_1 = 1$ and $\delta = 0$:

$$X_t = X_{t-1} + Z_t \tag{3.3}$$

Note that for the expectation of the random walk holds:

$$\begin{aligned}\mathbb{E}(X_t) &= \mathbb{E}(X_{t-1}) + \mathbb{E}(Z_t)\\ &= \mathbb{E}(X_{t-1})\end{aligned} \tag{3.4}$$

Therefore for $n \in \mathbb{N}$ the expected value at $t+n$ is equal to the expected value at t:

$$\mathbb{E}(X_{t+n}) = \mathbb{E}(X_t) \tag{3.5}$$

The variance however is not constant:

$$\begin{aligned}\mathrm{Var}(X_{t+n}) &= \mathrm{Var}\left(X_t + \sum_{i=1}^{n} Z_i\right)\\ &= \mathrm{Var}(X_t) + n\sigma^2\end{aligned} \tag{3.6}$$

Here we used that the Z_i are *independent and identically distributed* (i.i.d.), in particular independent of X_t. This means the variance is increasing over time. In case $\delta \neq 0$, we have a *random walk with drift* δ:

$$X_t = \delta + X_{t-1} + Z_t \tag{3.7}$$

Given a time series $X_{1:N}$, we are often interested in its properties, such as the expected value or the variance of the time series. In case the expected value of the time series is constant in time, a classical way to approximate this is by taking the *sample mean*:

$$\overline{X} = \frac{1}{N}\sum_{i=1}^{N} X_i \tag{3.8}$$

A constant variance can be approximated by the *sample variance*:

$$\sigma_{SV} = \frac{1}{N-1}\sum_{i=1}^{N}(X_i - \overline{X})^2 \tag{3.9}$$

3.2. Outliers

A data set may contain data points that were not generated by the same process as the process we are interested in. These points are called *outliers* and we often wish to exclude them when we are making calculations. One way of dealing with outliers is by using *robust statistics*. These statistics aren't affected as much by outliers or changes in the parametric distribution of the data. This could also be beneficial if we are dealing with data from a wide range of probability distribution.

3.2.1. Robust mean

If we are interested in the level of a time series $X_{1:N}$, the classical approach is to use the mean (3.8). This is not a robust statistic, because by changing only one of the X_i, we can make the mean arbitrarily large. A robust statistic for this is the *median*. The median is defined as the value m such that:

$$\mathbb{P}(X \leq m) \geq \frac{1}{2} \text{ and } \mathbb{P}(X \geq m) \geq \frac{1}{2} \tag{3.10}$$

The median has a *breakdown point* of 50%, indicating what percentage of the data can be outliers before the estimator produces an incorrect result. For the regular mean, this is 0%.

Another, equivalent definition of the median of random variable X is:

$$\text{median}(X) = \arg\max_{m\in\mathbb{R}} \mathbb{E}[|X - m|] \tag{3.11}$$

For the sample median holds:

$$\text{median}(X_{1:N}) = \arg\min_{m\in\mathbb{R}} \sum_{i=1}^{N}[|X_i - m|] \tag{3.12}$$

Note that this is the same as (3.39), but assuming $\beta = 0$.

3.2.2. Robust variance

To estimate the variability of a data sample, the *median absolute deviation* (MAD) can be used. The MAD is defined as:

$$MAD = \underset{i}{\text{median}}(|X_i - \text{median}(X_{1:N})|) \tag{3.13}$$

For a consistent estimator of the standard deviation σ, the MAD has to be multiplied by a factor, depending on the distribution of the data. Because the median is the middle value of the data, we see that if we are dealing with Gaussian data $X_i \sim \text{N}(\mu, \sigma)$ for $1 \leq i \leq n$, holds:

$$\begin{aligned}
\frac{1}{2} &= \mathbb{P}(|X-\mu| \leq MAD) && (3.14)\\
&= \mathbb{P}\left(\frac{|X-\mu|}{\sigma} \leq \frac{MAD}{\sigma}\right) && (3.15)\\
&= \mathbb{P}\left(|Z| \leq \frac{MAD}{\sigma}\right) && (3.16)
\end{aligned}$$

Here $Z \sim \mathrm{N}(0,1)$. Now we see:

$$\begin{aligned}\frac{1}{2} &= \mathbb{P}\left(Z \leq \frac{MAD}{\sigma}\right) - \mathbb{P}\left(Z \leq -\frac{MAD}{\sigma}\right) \\ &= \mathbb{P}\left(Z \leq \frac{MAD}{\sigma}\right) - \left(1 - \mathbb{P}\left(Z \leq \frac{MAD}{\sigma}\right)\right) \\ \frac{3}{4} &= \mathbb{P}\left(Z \leq \frac{MAD}{\sigma}\right) \qquad (3.17)\end{aligned}$$

For Φ the cumulative distribution function of the standard normal distribution, we get:

$$MAD = \Phi^{-1}\left(\frac{3}{4}\right)\sigma \tag{3.18}$$

$$\sigma_{MAD} = \frac{1}{\Phi^{-1}\left(\frac{3}{4}\right)} MAD \tag{3.19}$$

So in the case of Gaussian data, taking 1.4826 MAD will result in an estimator for the standard deviation σ. Because of the use of the median, the MAD also has a breakdown point of 50%.

There are other robust options for estimating the standard deviation. One of them is proposed by Rousseeuw [33]. The estimator is:

$$\sigma_S = c \operatorname*{median}_i(\operatorname*{median}_j |X_i - X_j|) \tag{3.20}$$

This estimator also has a breakdown point of 50% and is shown to be more efficient for Gaussian data. To make this a consistent estimator for σ of a normal distributed sample, a constant $c = 1.1926$ has to be used.

3.2.3. Filters

Another way to deal with outliers is to apply a *filter* to the times series. We will look into the *Hampel filter*. This filter is based on a moving average window of half-width k:

$$W_i^k = \{X_{i-k}, ..., X_i, ..., X_{i+k}\} \tag{3.21}$$

Now, define:

$$m_i = \text{median}(W_i^k) \tag{3.22}$$

the median of the moving average window. The response of the Hampel filter is:

$$h_i = \begin{cases} x_i & \text{for } |x_i - m_i| \leq t\sigma_{MAD,i} \\ m_i & \text{for } |x_i - m_i| > t\sigma_{MAD,i} \end{cases} \tag{3.23}$$

Here t is a threshold parameter and $\sigma_{MAD,i}$ the estimated variance (3.19) using the MAD estimate, calculated with the the data points in window W_i^k. For $t = 0$, this results in the so called *median filter*, which is very efficient in removing outliers from a dataset. The strength of the more sophisticated Hampel filter is the possibility to increase the sensitivity to changes in the time series; increasing t will lead to less replacements of data by the median of the moving average. This way bigger fluctuations stay present in the data.

3.3. Bayesian statistics

Bayesian statistics is field in statistics, in which prior knowledge is updated by observations to calculate probabilities. We will first introduce some notation. Let X, Y be random variables, then we will denote the probability distribution function of X as $f(X = x)$ or $f(x)$. The probability distribution function of X given Y we will denote as $f(X = x|Y = y)$, or $f(x|y)$ shortly.

An important tool in Bayesian statistics this is the *Bayes formula* or *Bayes rule*:

$$f(X = x|Y = y) = \frac{f(Y = y|X = x)f(X = x)}{f(Y = y)} \tag{3.24}$$

In Bayesian statistics, to make inference on a parameter θ in the distribution of a random variable X, θ is assumed to be a random variable itself. We will assume prior distribution $\pi(\theta)$ on θ. Using Bayes formula we can find the *posterior distribution* of θ given the data X:

$$\pi(\theta|x) = \frac{f(x|\theta)\pi(\theta)}{f(x)} \tag{3.25}$$

3.4. Linear regression

In the field of linear regression the aim is to construct a straight line that approximates the time series of interest. To this end, the time series $Y_{1:N}$ is assumed to constructed by the regression line and an noise term Z_i:

$$Y_i = \alpha + \beta x_i + Z_i \tag{3.26}$$

In the best known case the noise Z_i is assumed to be Gaussian, although another type of distribution may be assumed, such as heavy tailed Student's-t noise.

3.4.1. Gaussian noise

First, will assume $Z_i \sim N(0, \sigma^2)$ is a Gaussian error on the data. By doing so we are able to derive estimators for α and β, using maximum likelihood estimation. Provided with data $\{x_i, y_i\}_{1\le i\le N}$ the likelihood function is:

$$\begin{aligned} L(\sigma,\alpha,\beta) &= \prod_{i=1}^{N} f(y_i|x_i,\sigma,\alpha,\beta) \\ &= \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i-\alpha-x_i\beta)^2}{2\sigma^2}} \end{aligned} \tag{3.27}$$

We find for the log likelihood:

$$\begin{aligned} \mathscr{L}(\sigma,\alpha,\beta) &= \sum_{i=1}^{N}\left(\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{(y_i-\alpha-x_i\beta)^2}{2\sigma^2}\right) \\ &= -\frac{N}{2}\log(\pi) - N\log(\sigma) - \sum_{i=1}^{N}\frac{(y_i-\alpha-x_i\beta)^2}{2\sigma^2} \end{aligned} \tag{3.28}$$

An analytic expression for the maximum likelihood estimator of α and β can be found by solving the system:

$$\begin{cases} \frac{\partial \mathscr{L}(\sigma,\alpha,\beta)}{\partial\alpha} = 0 \\ \frac{\partial \mathscr{L}(\sigma,\alpha,\beta)}{\partial\beta} = 0 \\ \frac{\partial \mathscr{L}(\sigma,\alpha,\beta)}{\partial\sigma} = 0 \end{cases} \tag{3.29}$$

Solving this system leads to the *ordinary least square estimate* (OLS) for α and β:

$$\begin{cases} \hat{\beta}_G = \frac{\sum_{i=1}^{N}(x_i-\overline{x})(y_i-\overline{y})}{\sum_{i=1}^{N}(x_i-\overline{x})^2} \\ \hat{\alpha}_G = \overline{y} - \hat{\beta}_G\overline{x} \end{cases} \tag{3.30}$$

Here $\overline{x}$ and $\overline{y}$ denote the sample mean (3.8) of x and y. We also found the maximum likelihood estimator for σ^2:

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{N}(y_i - \hat{\alpha}_G - \hat{\beta}_G x_i)^2 \tag{3.31}$$

Using α and β we can now calculate the Gaussian regression line:

$$l_G(x_i) = \hat{\alpha}_G + \hat{\beta}_G x_i \tag{3.32}$$

These estimates $\hat{\alpha}_G$ and $\hat{\beta}_G$ are called the ordinary least square estimate, because they are also the solution to the optimization problem of minimizing the square of the difference between the data points and the regression line:

$$(\hat{\alpha}_G, \hat{\beta}_G) = \underset{(\alpha,\beta)}{\arg\min}\sum_{i=1}^{N}(y_i - \alpha_G - \beta_G x_i)^2 \tag{3.33}$$

3.4.2. Student's t noise

Using Gaussian errors is common practice in linear regression. Closed expressions are available for the regression estimators and they can be calculated in $\mathcal{O}(N)$. A downside of modelling errors as Gaussians, is that outliers in the data have a big impact on the regression curve, because the normal distribution is not heavy tailed. This means the probability of observing an outlier is very low, compared to a distribution that has a heavy tail. To better incorporate the possibility of outliers, we will therefore perform regression using errors from the heavy-tailed *Student's-t distribution.* It has the following distribution function:

$$f(x|\nu,\rho) = \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})\sqrt{\pi\nu\rho^2}}\left(1+\frac{1}{\nu}\left(\frac{x}{\rho}\right)^2\right)^{-\frac{\nu+1}{2}} \tag{3.34}$$

Here $\nu > 0$ are the *degrees of freedom* and ρ the scaling parameter. The thickness of the tail depends on the degrees of freedom and the scaling parameter can be used to adjust the variance. For X following the Student's-t distribution, the variance is:

$$\begin{cases} \frac{\nu}{\nu-2} & \text{for } \nu > 2 \\ \infty & \text{for } 1 < \nu \le 2 \\ \text{undefined} & \text{else} \end{cases} \tag{3.35}$$

We will again assume (3.26) but this time the Z_i are independent identically distributed according to a Student's-t distribution. For the likelihood function we find:

$$L(\nu,\rho,\alpha,\beta) = \prod_{i=1}^{N} \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})\sqrt{\pi\nu\rho^2}}\left(1+\frac{1}{\nu}\left(\frac{y_i-\alpha-\beta x_i}{\rho}\right)^2\right)^{-\frac{\nu+1}{2}} \tag{3.36}$$

Now for the log likelihood we get:

$$\begin{aligned}\mathscr{L}(\nu,\rho,\alpha,\beta) = & \sum_{i=1}^{N}\log\left(\frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})\sqrt{\pi\nu\rho^2}}\right) - \left(\frac{\nu+1}{2}\right)\log\left(1+\frac{1}{\nu}\left(\frac{y_i-\alpha-\beta x_i}{\rho}\right)^2\right) \\ = & N\log\left(\frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})\sqrt{\pi\nu\rho^2}}\right) - \sum_{i=1}^{N}\left(\frac{\nu+1}{2}\right)\log\left(1+\frac{1}{\nu}\left(\frac{y_i-\alpha-\beta x_i}{\rho}\right)^2\right)\end{aligned} \tag{3.37}$$

Again we could find the maximum likelihood estimators by solving system (3.29). In this case however, there is no exact solution, therefore a numerical approximation has to be used. After solving we can calculate the Student-t regression line:

$$l_{St}(x_i) = \alpha_{St} + \beta_{St}x_i \tag{3.38}$$

3.4.3. Robust regression as LP problem

The OLS estimators are found by looking at the square of the difference between the data and the regression line (3.33). Using this square, outliers have a big effect on the outcome. To reduce the effect outliers have on the regression, we can use the absolute difference instead. We are looking for coefficients α and β such that the absolute distance between the line $\alpha+\beta x$ and the data Y is minimized; we will find the robust linear regression coefficients by:

$$\underset{\alpha,\beta\in\mathbb{R}}{\arg\min}\sum_{i=1}^{N}[|Y_i-\alpha-\beta x_i|] \tag{3.39}$$

This can be written as a linear programming problem (LP-problem). Write $c_i = |Y_i-\alpha-\beta x_i|$, then we see that minimizing (3.12) is equivalent to the following LP problem:

$$\begin{aligned} \min\ & c_i \\ \text{s.t. } & Y_i-\alpha-\beta x_i \le c_i \\ & -Y_i+\alpha+\beta x_i \le c_i \\ & c_i \ge 0 \end{aligned} \tag{3.40}$$

After solving, using for example the Simplex algorithm, we can find the robust regression line:

$$l_R(x_i) = \alpha_R + \beta_R x_i \tag{3.41}$$

3.5. System performance

If we are dealing with a system that has a binary outcome, the outcome of the system can take two values, either a positive, or a negative. An example of such a system is an alarm system; the result is either an alarm (positive) or no alarm (negative). Two important measures for the performance of these systems are the *precision* and *recall*. We define:

- A positive outcome of the system is a *true positive* if the outcome should have been positive as well.
- A positive outcome of the system is a *false positive* if this outcome should have been negative.
- A negative outcome of the system is a *false negative* if the outcome should have been positive.

To test the performance of the system, a test can be performed by running the system multiple times. Let n_{tp} the number of true positives, n_{fp} the number of false positives and n_{fn} the number of false negatives found by the system. We define the precision of the system as the ratio of true positives over the number of positives found by the system:

$$\text{Precision } = \frac{n_{tp}}{n_{tp} + n_{fp}} \tag{3.42}$$

The recall of the system is the ratio of true positives over the number of outcomes that should have been positive:

$$\text{Recall } = \frac{n_{tp}}{n_{fn} + n_{tp}} \tag{3.43}$$

The recall is also refereed to as sensitivity. Note that if the system only has positive outcomes, the precision will be low, but the recall will be 1 because not a single positive has been missed ($n_{fn} = 0$).

4 Change point detection

When analysing a time series, we are often interested in the distribution of our observations. This distribution can change over time. A point in time where the distribution of the time series undergoes an abrupt change is referred to as a *change point.* In *sequental change point detection* the aim is to detect such a change as soon as possible after its occurrence.

Let $X_1, X_2, \ldots$ be the time series of interest. The definition of a change point is quite general; a change point is a point in time τ where $X_{1:\tau}$ has been generated according to some process, that is different from the process generating data $X_{\tau+1}, X_{-\tau+2}, \ldots$. Most literature discuss the case where the data before and after the change point are independent and identically distributed random variables, but other models such as an AR model (3.2) could be used to describe the data.

In this chapter we will first give an overview of theory on change point detection, starting with early frequentist methods in section 4.1 and Bayesian methods in section 4.2. After this we will cover finding a single change point in a time series using a cost function and an estimator in section 4.3. Here several options for the cost function and estimator will be discussed and the use of a linear regression line as estimator is proposed. In section 4.4 several algorithms to detect multiple change points using a cost function are discussed. The theory will be applied to find changes in a random walk process in section 4.5. To check the performance of the change point detection methods, a simulation test has been done in section 4.6. We will use the best performing change point detection method to help estimate initial values for the alarm system, as illustrated in figure 1.2.

4.1. Frequentist approach

Detecting a change in distribution function could for example be a change in mean of the process, a change in variance, both of them or the entire type of distribution could be different. Early attempts focussing on detecting a change were made by Shewhart [7], introducing the concept of *state of statistical control.* The idea is that the probability distribution of the data depends on some parameter θ, which can be monitored using some statistic Y_t. If Y_t exceeds certain control limits, the process is no longer in the state of control and the hypothesis made on θ must be rejected. The test statistic Y_t is calculated on a so called process inspection scheme, taking samples of fixed size N at regular time intervals. For a change in mean of the time series, the mean of the samples in used as test statistic, resulting in the so called *Shewhart $\overline{X}$-chart.* If assumed the data is i.i.d. Gaussian, the control limits used are:

$$\overline{X} \pm \frac{B\sigma}{\sqrt{N}} \tag{4.1}$$

Here B was calculated to be 3.09. We can define the stopping time for this procedure as:

$$T = \inf\left\{ t : X_t > \overline{X} + \frac{B\sigma}{\sqrt{N}}, X_t < \overline{X} - \frac{B\sigma}{\sqrt{N}} \right\} \tag{4.2}$$

Other control limits can be used in the stopping time to detect changes in a different parameter.

Another class of schemes was introduced by Page [8], cumulative sum (CUSUM) schemes. If the X_i are i.i.d. up to $i = \tau$ and i.i.d. as well for $i > \tau$, the stopping time for this procedure takes the from:

$$T_{CS} = \inf\left\{ n : \max_{1 \leq j \leq n} \left[\sum_{i=j}^{N} \log\left(\frac{f_1(X_i)}{f_0(X_i)} \right) \right] \right\} \tag{4.3}$$

Something similar can be done in case the X_i left and right from the change point are not independent. Note that for this approach, the likelihood ratio of f_1 and f_0 must be known, which is often not the case in applications.

4.2. Bayesian approach

The first Bayesian description of the change point problem was formulated by Shiryaev [20]. He assumed that we are looking for a single change point, where the probability distributions before f_0 and after f_1 the change point are known. When approaching the change point detection in a Bayesian way, the change point itself is considered to be a random variable and a prior distribution for the change point has to be defined:

$$\mathbb{P}(\tau = k) = \pi_k \tag{4.4}$$

The delay in detecting a change point should be as short as possible. However, detecting a change point when there is not an actual change in distribution is also undesirable. To this end we define the *average delay to detection* as:

$$ADD^{\pi}_{\tau}(T) = \mathbb{E}(T - \tau | T > \tau) = \frac{\max\left\{0, \mathbb{E}^{\pi}(T - \tau)\right\}}{\mathbb{P}^{\pi}(T > \tau)} \tag{4.5}$$

and the *probability of false alarm*:

$$PFA^{\pi}(T) = \mathbb{P}^{\pi}(T \leq \tau) = \sum_{k=1}^{\infty} \pi_k \mathbb{P}_k(T \leq k) \tag{4.6}$$

Here T is the time of detecting the change point. Now, the aim is to minimize ADD, but not let the PFA exceed a fixed critical probability α. So for our fixed α we want an optimal stopping time T_{opt} with:

$$T_{opt} \text{ such that: } \begin{cases} PFA^{\pi}(T_{opt}) \leq \alpha \\ ADD^{\pi}(T_{opt}) = \inf_{T} ADD^{\pi}(T) \end{cases} \tag{4.7}$$

For his first approach, Shiryaev used a geometrical distribution with the probability that the change point is at $t = 0$ equal to π as prior distribution:

$$\mathbb{P}(\tau = k) = \begin{cases} \pi & \text{for } k = 0 \\ (1-p)^{k-1} p & \text{for } k > 0 \text{ given } \tau > 0 \end{cases} \tag{4.8}$$

Using this prior, an exact solution for the optimal stopping time is available. This results in a procedure with stopping time:

$$T_S = \inf\left\{ n \geq 1 : R_{n,p} \geq A_{\alpha} \right\} \tag{4.9}$$

Here:

$$R_{n,p} = \frac{q}{(1-q)p} \prod_{j=1}^{n} \left(\frac{\mathscr{L}_j}{1-p} \right) + \sum_{k=1}^{n} \prod_{j=k}^{n} \left(\frac{\mathscr{L}_j}{1-p} \right) \tag{4.10}$$

with $\mathscr{L}_j = \frac{f_1(X_j)}{f_0(X_j)}$. Note that A_{α} depends on α in the sense that we pick A_{α} in the following way:

$$T_S(A_{\alpha}) \text{ such that: } \begin{cases} PFA^{\pi}(T_S(A_{\alpha})) = \alpha \\ ADD^{\pi}(T_S(A_{\alpha})) = \inf_{T} ADD^{\pi}(T) \end{cases} \tag{4.11}$$

This, like the frequentist method, depends on the likelihood ratio $\mathscr{L}$. We will now focus on methods where knowledge of $\mathscr{L}$ is not necessarily required.

4.3. Single change point

We will investigate the case where a change point defined by a change in the underlying model generating the data, where the data is not necessarily i.i.d. within each segment. In order to do so, we will first assume that our time series contains only one change point that splits the time series into two segment. To find this change point, most modern change point methods use a cost function $\gamma(X,\theta)$ that depends on some statistic θ of the time series [11][12]. This cost function describes the cost of a single data point in the time series. If available often a negative log likelihood can also be used as a cost function [14], as can minus the posterior distribution in a Bayesian setting [17]. Other cost functions can be used if the likelihood ratio is not available.

We will assume the change we are looking for can be identified by statistic θ of the time series. If we have a cost function $\gamma(X,\theta)$, we can find the cost of a segment by summing up the cost of the data points in the segment. We define:

$$\mathscr{C}(X_{s:t}) = \min_{\theta} \sum_{i=s}^{t} \gamma(X_i,\theta) \tag{4.12}$$

Here we assumed θ is constant throughout the segment. We see that the sum of the cost of each data point is minimized over the statistic θ. In practice, minimizing over all possible θ is very time consuming, which is why an estimate $\hat{\theta}$ for θ is used:

$$\mathscr{C}(X_{s:t},\hat{\theta}) = \sum_{i=s}^{t} \gamma(X_i,\hat{\theta}) \tag{4.13}$$

We will also include the possibility for θ_i to vary within a segment, under the condition that θ_i is proportional to the time. In this case, the cost of a segment is defined as:

$$\mathscr{C}(X_{s:t},\hat{\theta}_i) = \sum_{i=s}^{t} \gamma(X_i,\hat{\theta}_i) \tag{4.14}$$

If θ_i is not constant within the segment, a change point represents for example a sudden jump in the statistic $\theta_{\tau+1}$ of the time series that is not according to the proportionality of θ_τ.

Assume we have a time series $X_{1:N}$ and we are looking for a change point $0 < \tau < N$ for which the statistic θ of the time series has a sudden change. This can be done by minimizing the sum of the cost of the two segments that are obtained by splitting up the time series at time τ. Therefore if we use a (possibly time-varying) estimate $\hat{\theta}_i$ for the statistic θ, the change point can be found by:

$$\tau_c = \underset{0\le\tau<N}{\arg\min} \left[\sum_{t=1}^{\tau} \mathscr{C}(X_{1:\tau},\hat{\theta}_i) + \sum_{t=\tau+1}^{N} \mathscr{C}(X_{\tau+1:N},\hat{\theta}_i) \right] \tag{4.15}$$

If $\tau = 0$, no change point was found in the time series.

We will focus on the case where θ_i represents the level of the time series. Several ways to approximate (possibly time varying) θ_i will be discussed and we will formulate cost functions that can be used.

4.3.1. Cost functions for mean

A common cost function in literature, used to find the mean of a time series is the quadratic loss function:

$$\gamma(X,\theta) = (X-\theta)^2 \tag{4.16}$$

This loss function results from looking at normally distributed data and taking minus the log likelihood as a cost function. Another commonly used loss function is the absolute error:

$$\gamma(X,\theta) = |X-\theta| \tag{4.17}$$

The absolute error is less sensitive to outliers than the quadratic loss, because the contribution of outliers to the result is decreased due to the absolute value instead of the square. Another, more robust option is to use the biweight cost function [10]:

$$\gamma(X,\theta) = \begin{cases} (X-\theta)^2 & \text{if } |X-\theta| < K \\ K^2 & \text{else} \end{cases} \tag{4.18}$$

Here the cost function is bounded by K^2. K can be varied to make the method more robust to outliers. Decreasing K will reduce the contribution of outliers to the result, however it might also affect the cost of data points that are not outliers.

4.3.2. Estimating θ

In case θ represents the mean of the time series, the most straight forward way to estimate it would be to look at the sample mean (3.8) of the time series:

$$\hat{\theta}_{s:t} = \frac{1}{t-s+1} \sum_{i=s}^{t} X_i \tag{4.19}$$

Or we could use the more robust median (4.20):

$$\hat{\theta}_{s:t} = \text{median}(X_{s:t}) \tag{4.20}$$

If we use (4.19) or (4.20), we assume the level of the time series is constant from time s to t. If this is not the case, θ is a function of time and so should our estimate be. We will assume our time series follows a linear trend, that is the level is proportional to the time t. Instead of looking at the distance between each data point and the line $y = \theta$, we propose to look at the distance to a fitted line that is not necessarily horizontal. To this end we will use linear regression, as described in section 3.4. These regression lines can be seen as the trend of the time series and can be used as estimation for a changing θ:

$$\hat{\theta}_i = l_G(i) \tag{4.21}$$

$$\hat{\theta}_i = l_{St}(i) \tag{4.22}$$

$$\hat{\theta}_i = l_R(i) \tag{4.23}$$

4.4. Multiple change points

The methods mentioned above are designed to detect a single change point and can be used online, meaning they can be used on real-time data. Given a data set, we also want an algorithm that is able to detect multiple change points. For this we can use the *Optimal Partitioning* algorithm described in section 4.4.1, or the *PELT* algorithm, which is an improved version of the OP algorithm, as described in section 4.4.2. Another option is the *Binary Segmentation* algorithm described in section 4.4.3 or its improved version, the *Wild Binary Segmentation* algorithm described in section 4.4.4.

Assume we have a data set $\{X_t\}_{t \leq N}$ for $N \in \mathbb{N}^+$ finite. We assume there are m change points in our data. Our goal is to split our data into $m+1$ segments, where the change point will be the last point in each segment. By setting $\tau_0 = 0$ and $\tau_{m+1} = N$ we write $\tau = (\tau_0, ... \tau_{m+1})$ for the set of change points. We will write $\mathscr{T}_t = \{\tau_{1:k} : 0 < \tau_1 < ... < \tau_m < t\}$ the set of possible change points for $X_{1:t}$

To find a proper segmentation of our data, we will use an appropriate cost function $\gamma(X, \theta)$ for a single observation X, and a location parameter θ pertaining to the segment X is in. The end goal is to find a segmentation that minimizes the overall cost for this segmentation. Using the definition for the cost of one segment (4.12), we define the overall cost of a segmentation as:

$$Q(X_{1:N}, \tau_{0:m+1}) = \sum_{i=1}^{m+1} \mathscr{C}(X_{\tau_{i-1}+1:\tau_i}) \tag{4.24}$$

To prevent the algorithm of finding too many change points, a penalty $\beta f(m)$ is introduced, depending on the amount of change points. We can define the penalized cost for a segmentation as:

$$Q(X_{1:N}, \tau_{0:m+1}) = \sum_{i=1}^{m} [\mathscr{C}(X_{\tau_{i-1}+1:\tau_i})] + \beta f(m) \tag{4.25}$$

Often the penalty is linear in the amount of change point, adding a constant for every change point the algorithm finds, but there are more possibilities. Examples are using the Aikake information criterion (AIC) or Bayesian information criterion (BIC) [13]. We will be using a linear penalty, which results in:

$$Q(X_{1:N}, \tau_{0:m+1}) = \sum_{i=1}^{m} [\mathscr{C}(X_{\tau_{i-1}+1:\tau_i}) + \beta] \tag{4.26}$$

4.4.1. Optimal Partitioning

An algorithm to find the optimal segmentation, or optimal partitioning, that minimizes (4.26) is the OP-algorithm suggested by Jackson et al [17]. It makes use of dynamic programming to find the optimal segmentation. It is a recursive procedure that first determines the optimal partition of $X_{1:t}$ to calculate the optimal partition of $X_{1:t+1}$. If we define $F(t)$ the value of (4.26) when we found the optimal segmentation of $X_{1:t}$, we can see that for $s < t \leq N$ holds:

$$\begin{aligned} F(t) &= \min_{\tau \in \mathcal{T}_t} \left\{ \sum_{i=1}^{m+1} [\mathscr{C}(X_{\tau_{i-1}+1:\tau_i}) + \beta] \right\} && (4.27) \\ &= \min_{s} \left\{ \min_{\tau \in \mathcal{T}_s} \sum_{i=1}^{m+1} [\mathscr{C}(X_{\tau_{i-1}+1:\tau_i}) + \beta] + \mathscr{C}(X_{(s+1):N}) + \beta \right\} && (4.28) \\ &= \min_{s} \left\{ F(s) + \mathscr{C}(X_{(s+1):N}) + \beta \right\} && (4.29) \end{aligned}$$

Using this recursion we can formulate the OP-algorithm as follows:

```
Result: Vector $cp(N)$ containing all ordered change points
Initialize:
$F(0) = -\beta$;
$cp(0) = NULL$;
for $\tau^* = 1, ..., N$ do
    $F(\tau^*) = \min_{0 \leq \tau < \tau^*} [F(\tau) + \mathscr{C}(X_{(\tau+1):\tau^*}) + \beta]$ ;
    $\tau' = \arg\{\min_{0 \leq \tau < \tau^*} [F(\tau) + \mathscr{C}(X_{(\tau+1):\tau^*}) + \beta]\}$ ;
    $cp(\tau^*) = (cp(\tau'), \tau')$
end
```

Algorithm 1: OP-algorithm.

This algorithm computes the change points in $\mathcal{O}(N^2)$.

4.4.2. PELT-algorithm

The OP-algorithm is able to find the desired change points in $\mathcal{O}(N^2)$; the PELT algorithm developed by Fearnhead et al [14] uses the same principle but has $\mathcal{O}(N)$. It is able to do so by removing the values of τ that will never be minima from the minimization of $F(\tau*)$. In order to do this, the following is assumed:

Assumption 1. *There exists a constant K such that for all $s < t < N$ holds:*

$$\mathscr{C}(X_{(s+1):t}) + \mathscr{C}(X_{(t+1):N}) + K \leq \mathscr{C}(X_{(s+1):N}) \tag{4.30}$$

In other words, by adding a change point, the value of the cost function will reduce. As a result, if:

$$F(s) + \mathscr{C}(X_{(s+1):t}) + K \geq F(t) \tag{4.31}$$

we know that for some $N > t$, t will never be the optimal change point prior to N. Using this principle, the PELT algorithm 2 has been proposed.

```
Result: Vector $cp(N)$ containing all ordered change points
Initialize:
$F(0) = -\beta$;
$cp(0) = NULL$ ;
$R_1 = \{0\}$;
for $\tau^* = 1, ...N$ do
    $F(\tau^*) = \min_{\tau \in R_{\tau^*}} [F(\tau) + \mathscr{C}(X_{(\tau+1):\tau^*}) + \beta]$ ;
    $\tau^l = \arg\{\min_{\tau \in R_{\tau^*}} [F(\tau) + \mathscr{C}(X_{(\tau+1):\tau^*}) + \beta]\}$;
    $cp(\tau^*) = [cp(\tau^l), \tau^l]$;
    $R^*_\tau + 1 = \{\tau^* \cap \{\tau \in R_{\tau^*} : F(\tau) + \mathscr{C}(X_{(\tau+1):\tau^*}) + K < F(\tau^*)\}\}$;
end
```

Algorithm 2: PELT-algorithm.

4.4.3. Binary Segmentation (BS)

Previous methods mentioned for multiple change point detection both optimize using some dynamic programming to determine multiple change points at the same time. The Binary Segmentation approach [18] is different, in the sense that in every step, a new change point is found and added to the set of change points. After this, the interval is split in half and in both new intervals the search for a new change point is continued. This will go on until some stopping condition is met.

In the original algorithm, designed to detect a change in mean, where the data was i.i.d. Gaussian, a CUSUM statistic of the following form is maximized:

$$\widehat{X}_{s,t}^{\tau} = \sqrt{\frac{t-\tau}{N_i(\tau-s+1)}}\sum_{j=s}^{\tau} X_j - \sqrt{\frac{\tau-s+1}{N_i(t-\tau)}}\sum_{j=\tau+1}^{t} X_j \tag{4.32}$$

Here s and t are the left and right end of the time interval, N_i is the length of this interval (equal to $t-s+1$) and τ is the parameter over which we are maximizing. The value of τ pertaining to the maximum value of $\widehat{X}_{s,t}^{\tau}$ will be the change point found. Note that maximizing $|\widehat{X}_{s,t}^{\tau}|$ is equivalent to the least square estimate of a piecewise-constant function on $X_s,...,X_t$ that has exactly one change point [16].

The algorithm will terminate if $|\widehat{X}_{s,t}^{\tau}| \leq \zeta_N$, for some threshold ζ_N, depending on the length N of the time series.

```
Result: Vector $cp$ containing all ordered change points
Initialize:
$s = 0$;
$t = N$;
$cp = [\,]$;
BinSeg($s, t, \zeta_N$):
if $t - s < 1$ then
  | stop
else
  | $\tau' := \arg\max_{\tau\in\{s,...,t-1\}} |\widehat{X}_{s,t}^{\tau}|$ ;
  | if $|\widehat{X}_{s,t}^{\tau}| > \zeta_N$ then
  |   | Add $\tau$ to $cp$;
  |   | BinSeg($s, \tau', \zeta_N$) ;
  |   | BinSeg($\tau' + 1, t, \zeta_N$) ;
  | else
  |   | stop
  | end
end
Order $cp$
```

Algorithm 3: BS-algorithm.

This algorithm typically takes $\mathcal{O}(Nlog(N))$, which is better than the OP-method but computationally more expensive than the PELT-method. Note that while searching for change points, other statistics than CUSUM could be used; minimizing a cost function would also be an option. In fact, any change point detection method designed to detect one change can be used. The difference with the penalized cost approach from previous methods is that here the penalty of detecting too many change points is in the threshold ζ_N and not in a penalty per change point.

4.4.4. Wild Binary Segmentation (WBS)

One of the downsides of the BS-method is that it searches for only one change point each time. Since the maximizing CUSUM statistic (4.32) is actually fitting a piecewise constant function with one change point, the presence of multiple change points can cause the algorithm to find wrong or no change point, especially when the actual change points are close to each other. The same problem occurs when using a different cost funtion to find the single change points. As a result, the change points have to be a minimum distance of order $N^{\frac{3}{4}}$, quite big. To overcome this problem, the *Wild Binary Segmentation method* (WBS) was proposed by Fryzlewicz [19].

Instead of looking for one change point in $(X_1,...,X_N)$, the WBS-algorithm first randomly draws a number of subsamples, $X_s,...,X_t$ where $1 \leq s < t \leq N$ integers. Now we search for a change point candidates by maximizing the CUSUM statistic (4.32) (or minimizing some other cost function) within each subsample. The largest value will be tested against some threshold. If this results in detecting a change point, this procedure will be repeated on the left and right side of the change point found.

If sufficiently many subsamples are created, from different sizes, the probability of having only one change point in one of the subsamples will increase and therefore this algorithm will be more accurate than the BS-algorithm.

Let G_N^M be a set of M random intervals $[s_m, t_m], m = 1, ..., M$. Here the start- and end-points have been drawn from $\{1, ..., N\}$ uniformly. Furthermore let ζ_N be a threshold parameter, then we can formulate the WBS-algorithm:

```
Result: Vector $cp$ containing all ordered change points
Initialize:
$s = 0$;
$t = N$;
$cp = [\,]$;
WildBinSeg($s, t, \zeta_N$):
if $t - s < 1$ then
    stop
else
    $\mathscr{M}_{s,t}$ := set of those indices $m$ for which $[s_m, t_m] \in F_N^M$ such that $[s_m, t_m] \subseteq [s, t]$ ;
    $(m_0, \tau') := \arg\max_{m \in \mathscr{M}_{s,t}, \tau \in \{s_m, ..., t_m - 1\}} |\widehat{X}_{s_m, t_m}^{\tau}|$ ;
    if $|\widehat{X}_{s_0, t_0}^{\tau'}| > \zeta_N$ then
        Add $\tau$ to $cp$ ;
        WildBinSeg($s, \tau', \zeta_N$) ;
        WildBinSeg($\tau' + 1, t, \zeta_N$) ;
    else
        stop
    end
end
Order $cp$
```

Algorithm 4: WBS-algorithm.

4.5. Change points in random walks

We want to apply the change point detection methods discussed in section 4.3 to a random walk with drift. Recall that a random walk with constant drift δ and constant variance σ can be described as:

$$X_t = X_{t-1} + Z_t + \delta \tag{4.33}$$

$$Z_t \sim N(0, \sigma) \tag{4.34}$$

Two types of sudden changes will be considered; a change in level, meaning that we observe a "jump" in the time series and a change in drift. We will first focus on finding a single change point and later apply this to find multiple change points. We can describe a change in drift as a change in δ, from δ_1 to δ_2 after change point τ_c. A sudden change in level with change point τ_c of size Δ can be described as:

$$X_{\tau_c+1} = X_{\tau_c} + Z_{\tau_c+1} + \delta + \Delta \tag{4.35}$$

Here Δ is large compared to the standard deviation σ of Z_{τ_c+1}. After the change in level the random walk continuous its normal behaviour (4.34).

To detect a change point we will make use of the cost functions as described in section 4.3.1. Recall that finding a single change point in time series $X_{1:N}$ can be done by minimizing the total cost of the two segments over the position τ of the change point separating these segments:

$$\tau_c = \underset{0 \le \tau < N}{\arg\min} \left[\sum_{t=1}^{\tau} \mathscr{C}(X_{1:\tau}, \hat{\theta}_i) + \sum_{t=\tau+1}^{N} \mathscr{C}(X_{\tau+1:N}, \hat{\theta}_i) \right]$$

If we want to find multiple change points, we can use this method of finding one change point in the WBS algorithm 4, or apply the PELT algorithm 2 using the described cost function. When using the WBS method, a threshold ζ_N and a way of measuring is needed to judge whether the point detected by minimizing (4.15) is indeed a change point or if it is not significant enough. Depending on the estimator used for θ_i and the type of change the threshold and method may differ.

We will discuss several cases and specify what estimate for θ can be used to find a single change point. Here we consider a change is in the level, in drift and a change in both of them. Also the cases $\delta = 0$ and $\delta \neq 0$ will be considered. For each type of change point we will also discuss the choice and usage of ζ_N. Plots showing examples of these change points can be found in figure 4.2.

- **Change in level where $\delta = 0$**
  Here $\delta = 0$ constant and we only observe a change in level at some change point τ_c. To find the position of this change point, we minimize (4.15) applying one of the cost functions in section 4.3, using an estimator for the mean of the times series. For this the mean (3.8) or median (3.11) can be used, because there is no drift and therefore the level will be constant within each segment.

  To check the significance of the change point in case we want to detect multiple change points, we can look at the absolute difference in mean resp. median of the two segments and only accept the change point in case this difference is bigger than some ζ_N^Δ. So, here ζ_N^Δ describes what the minimum value of Δ should be.

- **Change in drift**
  Note that:

  $$\nabla X_t = Z_t + \delta \tag{4.36}$$

  Therefore a change in δ is the same as a change in level for ∇X_t. Since no trend is involved in (4.36), we can again use the mean or median as estimate for θ. Using X_t is also possible to find a change in drift, if we use a linear regression line as estimator; a change in drift should result in a change in the regression line as well.

  To check the significance in the case of multiple change points, if we use ∇X_t we can again look at the difference of the mean resp. median of ∇X_t of the two segments and set an appropriate value for ζ_N^δ. Here ζ_N^δ describes what the minimal absolute difference of drifts δ_1 and δ_2 should be. If we use the linear regression line as estimator, we can look at the absolute difference of β (3.30) of the regression line before and after the change point and again use ζ_N^δ and accept the change point if the difference in β is larger than ζ_N^δ.

- **Change in level where $\delta \neq 0$**
  Here $\delta \neq 0$ constant. Now simply using the mean or median as estimator will not suffice. If for example $\delta > 0$ and the drop $\Delta < 0$, we could get a situation as shown in figure 4.1.

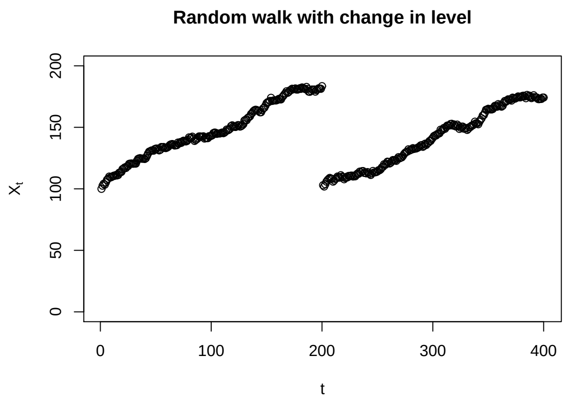


Figure 4.1: Random walk of length $N = 400$ with $\delta = 0.4$ constant and a single change in level at $\tau_c = 199$.

Here the means of the two segments are equal; therefore no change point will be detected in the middle, even though there is a clear change there. This problem can be solved by using an estimate for θ_i that is proportional to the time, to estimate the drift of the time series as a whole. For this we can use linear regression lines (3.32), (3.38) or (3.41). Using Gaussian regression is computationally the least expensive and therefore easier to implement.

If we are looking for multiple change points, we will look at the absolute difference between the last point of the regression line θ_τ and the first point of the regression line in the second segment $\theta_{\tau+1}$. If this is bigger than a threshold ζ_N^Δ the change point should be accepted. Again here ζ_N^Δ represents the minimum value of Δ.

- **Change in level and drift**
  This kind of change points is most interesting for our application, because a change in level of the spider often also includes a change in drift. For such a change, we can still use the linear regression lines as an estimate for θ_i.

  Here two types of changes can be observed, therefore two thresholds are set if we are looking for multiple change points. The first ζ_N^Δ to indicate the minimal value of Δ, as described in the case we have a change in level with $\delta \neq 0$. The second ζ_N^δ accounts for the change in δ. This change can be calculated by taking the difference of β (3.30) of the two regression lines.

4.5.1. Dealing with outliers

We will now look into a random walk with drift, where outliers are present. These outliers are false data points that are not generated according to the random walk. We want to minimize the influence of these outliers on the result of our change point method. One way to do this is by using a cost function that is robust for outliers, meaning it reduces the effect of the presence of outliers. Using the square cost function (4.16) is not a very robust choice. More robust options are the absolute cost (4.17) and the biweight cost function (4.18).

Another way to deal with outliers is to apply a filter before using a change point detection method, such as the median or Hampel filter (3.30).

4.6. Simulation study change point detection

We want to apply the change point detection methods described in section 4.5 to find changes stable segments (without change points) in a random walk where changes occur. To this end we will simulate data of a random walk with a single change point and test the change point detection methods. The drift δ and the standard deviation σ stay constant within the segment. For now we will also assume σ is the same in both segments, but a shift in level or drift may occur.

On top of this we assume there are outliers present, where the outliers follow the outlier distribution f^2 (5.17). At each time t the probability of having an outlier O_t is equal to p_o, meaning that after n data points the number of outliers is binomially distributed. The combined model can be described as a hidden Markov model, which is explained in chapter 5 with hidden state Y_t and observable X_t. Here the hidden sequence $Y_t = \begin{pmatrix} T_t \\ A_t \end{pmatrix}$ consists of T_t, where if $T_t = 0$ we observe the random walk and if $T_t = 1$ we have an outlier. Here A_t is the value of the random walk, meaning $X_t = A_t$ if $T_t = 0$.

Now:

$$X_t = \begin{cases} A_t & \text{if } T_t = 0 \\ O_t & \text{if } T_t = 1 \end{cases} \tag{4.37}$$

Here:

$$A_t = A_{t-1} + Z_t + \delta$$

$$T_t = \begin{cases} 0 & \text{with probability } p_o \\ 1 & \text{with probability } 1 - p_o \end{cases}$$

$$O_t \sim f^2(A_t)$$

A simulation study as been performed, by generation data according to this model and applying several filters, cost functions and estimators for θ_i to detect a single change point. These are described in section 4.3. In the simulation, a random walk has been simulated using a standard deviation of $\sigma = 1$ and a starting value $X_0 = 100$. The test has been performed by simulating a random walk of length $N = 400$ with one change point at time $\tau_c = 299$. The outlier probability has been set $p_o = 0.3$. The data will be pre-processed in the following ways:

- No filter

- Median filter using $k = 5$
- Hampel filter using $k = 5$ and $t = 3$

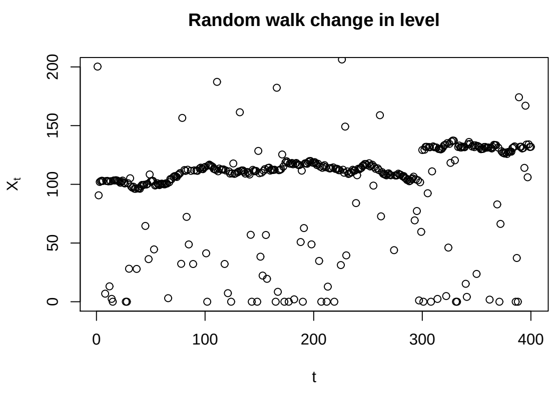
Besides, also the difference ∇X_t after applying these filters has been tested. The following cost functions have been used:

- Squared cost function
- Absolute cost function
- Biweight cost function, taking $K = 3$

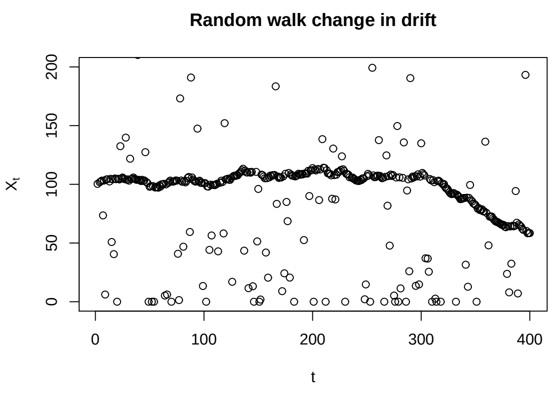
The following estimators for θ_i were applied:

- Mean
- Median
- Linear Gaussian regression

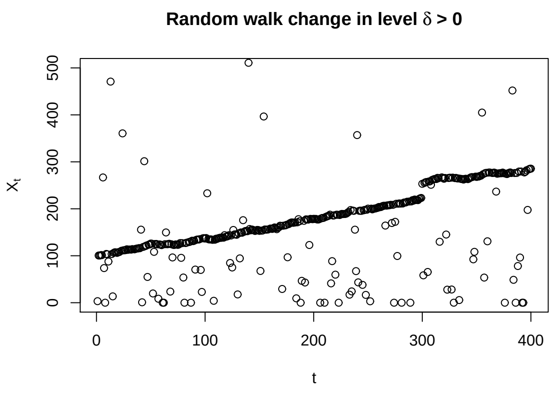
Note that only a Gaussian regression line has been used instead of other, more robust regression lines 3.38 and 3.41. The reason for this is that calculating these two regression lines is computationally too expensive to use for a test on this scale.



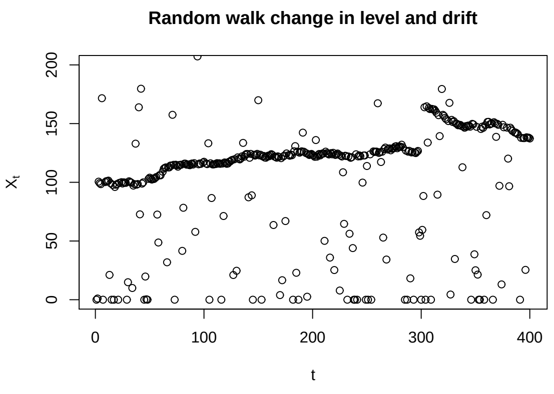
(a) Change in level.



(b) Change in drift.



(c) Change in level, where $\delta > 0$ constant.



(d) Change in both level and drift.

Figure 4.2: Examples of random walks with change point $\tau_c = 299$ used in to test the change point detection method.

The following change points have been considered:

- Change in level where $\delta = 0$ constant, $\Delta = 30$
- Change in drift where $\delta_1 = 0.1$ up to τ_c and $\delta_2 = -0.4$ after
- Change in level where $\delta = 0.5$ constant, $\Delta = 30$
- Change both level and drift, where $\delta_1 = 0.1$ up to τ_c, $\delta_2 = -0.4$ after and $\Delta = 30$

An example of a simulated data set for each of these changes in mean and level has been plotted in figure 4.2.

Each simulation has been performed 500 times. Afterwards the fraction of change points that exactly matched τ_c has been calculated. Also the fraction of change points τ found that were at most two off the actual value, so $|\tau - \tau_c| \leq 2$ has been calculated.

After this, the best resulting method is used to detect multiple change points $\{\tau_1 = 99, \tau_2 = 299, \tau_3 = 399\}$ in a data set of length $N = 500$. The data is simulated in the same way as before, however this time taking three jumps in level where $\{\Delta_1 = 30, \Delta_2 = 40, \Delta_3 = -50\}$ and the drifts of the four segments that make up the time series are $\{\delta_1 = 0.1, \delta_2 = 0.5, \delta_3 = -0.1, \delta_4 = -0.6\}$. As threshold constants, $\zeta_N^{\Delta} = 25$ and $\zeta_N^{\delta} = 0.3$ have been used. An example of the simulated data is plotted in figure 4.3.

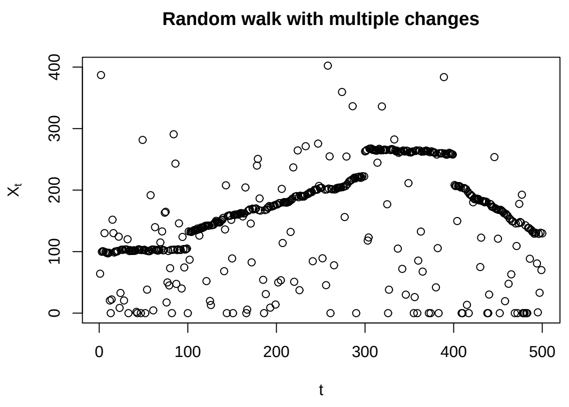


Figure 4.3: Example plot of data simulated for multiple change point test.

4.6.1. Results single change point

Not all combinations of cost function, estimator and filter lead to reasonable results. A description of what methods could lead to finding the correct change point can be found in section 4.5. We will discuss the most interesting results. The full tables with the precision for all combinations of cost function, estimator and filter can be found in Appendix A.1 for using X_t and in Appendix A.2 for using ∇X_t.

The result of applying methods using a mean or median cost function to find a change point in level where $\delta = 0$ constant are shown in table 4.1. We see that best results are obtained by first applying the Hampel filter and using the absolute cost function. Taking the median as estimator gives better results than taking the mean. Also worth noticing is that using the Hampel filter the results are much better for the exact precision than using the median filter, but if we look at the precision of $|\tau - \tau_c| \leq 2$, there is not much of a difference. Note that without filtering, using the median as estimator combined with the absolute or biweight cost function gives results almost as good, or in the case of the biweight cost function even better, as applying them with a filter. The prefered method to use here is the Hampel filter combined with an absolute cost function and the median estimator.

Cost function	Estimator	Type of filter	$\tau = \tau_c$	$\lvert\tau - \tau_c\rvert \leq 2$
Square	Mean	No filter	0.044	0.104
Square	Median	No filter	0.058	0.146
Absolute	Mean	No filter	0.184	0.324
Absolute	Median	No filter	0.502	0.758
Biweight	Mean	No filter	0.072	0.132
Biweight	Median	No filter	0.532	0.75
Square	Mean	Median filter	0.22	0.746
Square	Median	Median filter	0.22	0.736
Absolute	Mean	Median filter	0.216	0.746
Absolute	Median	Median filter	0.22	0.754
Biweight	Mean	Median filter	0.192	0.668
Biweight	Median	Median filter	0.184	0.586
Square	Mean	Hampel filter	0.43	0.704
Square	Median	Hampel filter	0.426	0.702
Absolute	Mean	Hampel filter	0.51	0.788
Absolute	Median	Hampel filter	0.534	0.816
Biweight	Mean	Hampel filter	0.38	0.64
Biweight	Median	Hampel filter	0.462	0.714

Table 4.1: Precision of finding a change in level, where $\delta = 0$ constant, using X_t.

The results of detecting a single change in drift using ∇X_t are not great; the methods that apply a Hampel filter perform the best and are shown in table 4.2. The reason these methods don't lead a very satisfying result could be that the change in drift is very small compared to the rest of the data and the outliers; $\delta_2 - \delta_1 = -0.4$, so the change in mean of ∇X_t that we are trying to detect is small compared to the outliers, which are in of the order of X_0 in size (remember $X_0 = 100$). Therefore detecting such a small change in drift can be difficult.

Cost function	Estimator	Type of filter	$\tau = \tau_c +$	$\lvert\tau - \tau_c\rvert \leq 2$	$\lvert\tau - \tau_c\rvert \leq 10$
Square	Mean	Median filter	0.038	0.13	0.282
Square	Median	Median filter	0.026	0.12	0.382
Absolute	Mean	Median filter	0	0.014	0.054
Absolute	Median	Median filter	0.016	0.078	0.286
Biweight	Mean	Median filter	0.046	0.216	0.562
Biweight	Median	Median filter	0.028	0.136	0.416
Square	Mean	Hampel filter	0.004	0.018	0.070
Square	Median	Hampel filter	0.03	0.088	0.266
Absolute	Mean	Hampel filter	0.016	0.06	0.190
Absolute	Median	Hampel filter	0.024	0.102	0.328
Biweight	Mean	Hampel filter	0.042	0.168	0.444
Biweight	Median	Hampel filter	0.046	0.152	0.430

Table 4.2: Precision of finding a change in drift, using ∇X_t.

To get some more insight, also the percentage of change points found τ for which $\lvert\tau - \tau_c\rvert \leq 10$ is also shown in the table. A difference of 10 is too big to be used in applications, but it shows that the change points found are at least around τ_c to a certain extend. Another option is to use X_t, combined with estimator l_G. Some of these results are shown in table 4.3.

Cost function	Estimator	Type of filter	$\tau = \tau_c$	$\lvert\tau - \tau_c\rvert \leq 2$	$\lvert\tau - \tau_c\rvert \leq 10$
Square	l_G	Median filter	0.006	0.068	0.346
Absolute	l_G	Median filter	0.014	0.082	0.376
Biweight	l_G	Median filter	0.014	0.07	0.362
Square	l_G	Hampel filter	0.014	0.064	0.288
Absolute	l_G	Hampel filter	0.01	0.06	0.358
Biweight	l_G	Hampel filter	0.012	0.07	0.364

Table 4.3: Precision of finding a change in drift, using X_t.

We see that also these results are not very promising; the best precision for $|\tau - \tau_c| \leq 10$ is 0.376 for using the absolute cost function with a median filter. Note that even if we don't use filtering, the results are similar. This could indicate that the reason for the bad results is not in the presence of outliers, but is that the random walk is "too random". As seen in section 3.1, the variance of a random walk increases in time. Therefore the difference of the random walk to the trend line can get very big in time, causing the algorithm the fail in detecting this trend change.

The result of using l_G to detect a change in level where $\delta \neq 0$ constant is shown in table 4.4. Using the absolute cost function gives the highest precision for each filter. Again the best results are obtained by using the Hampel filter, although the results is almost the same as applying the median filter. For this type of change point a Hampel filter combined with an absolute cost function and l_G as estimator is the preferred choice. Also note that the precision is significantly lower than in the case where $\delta = 0$. Using other estimators results in even lower precision.

Cost function	Estimator	Type of filter	$\tau = \tau_c$	$\lvert\tau - \tau_c\rvert \leq 2$
Square	l_G	No filter	0.002	0.014
Absolute	l_G	No filter	0.046	0.076
Biweight	l_G	No filter	0.02	0.044
Square	l_G	Median filter	0.258	0.424
Absolute	l_G	Median filter	0.36	0.604
Biweight	l_G	Median filter	0.236	0.456
Square	l_G	Hampel filter	0.272	0.442
Absolute	l_G	Hampel filter	0.364	0.604
Biweight	l_G	Hampel filter	0.26	0.47

Table 4.4: Precision of finding a change in level, where $\delta \neq 0$ constant, using X_t.

The last type of change point, a change in both level and drift, is most interesting for our application in the alarm system. The result of using estimator l_G is shown in table 4.5. As before, the absolute cost function shows better results than the other cost functions. The result of using other estimates than l_G is bad. The best results are obtained by using a Hampel filter, absolute cost function and estimator l_G.

Cost function	Estimator	Type of filter	$\tau = \tau_c$	$\lvert\tau - \tau_c\rvert \leq 2$
Square	l_G	No filter	0.022	0.046
Absolute	l_G	No filter	0.084	0.146
Biweight	l_G	No filter	0.028	0.066
Square	l_G	Median filter	0.21	0.742
Absolute	l_G	Median filter	0.212	0.764
Biweight	l_G	Median filter	0.2	0.678
Square	l_G	Hampel filter	0.464	0.71
Absolute	l_G	Hampel filter	0.492	0.76
Biweight	l_G	Hampel filter	0.316	0.578

Table 4.5: Precision of finding a change in both level and drift, using X_t.

4.6.2. Results multiple change points

We have applied the best performing method for a single change point in both level and drift, which is using a Hampel filter with absolute cost function and estimator l_G, in the WBS algorithm 4 to find multiple change points in a random walk with drift. There are three change points that the method should have detected. The average number of change points found is 10.6, a lot more than the desired 3. This also includes change points that were very close together and the change points that we wanted to detect. To see how many change points the method found that were truly different, the change points were put in bins of size 5. If multiple change points were in the same bin, we will count them as one. By taking this number of change points, on average 7.5 change points were found, of which 2.7 were true change points. Using these numbers, the values of the precision and recall have been calculated for the total and for each change point separately in table 4.6.

	τ_1	τ_2	τ_3	total
Precision	0.105	0.120	0.131	0.355
Recall	0.784	0.896	0.978	0.886

Table 4.6: Precision and recall of finding multiple changes in a random walk with outliers.

Here the precision per change point is taken by dividing the number of times change point τ has been detected over the total number of change point that was found. By taking the sum of these three, the precision of the whole change point method can be calculated. Note that the performance of this change point method is not very high, especially the precision is quite low. To see if outliers play a big role, the same test has been performed, without outliers and without using a Hampel filter. The precision and recall can be found in table 4.7.

	τ_1	τ_2	τ_3	total
Precision	0.320	0.340	0.338	0.999
Recall	0.938	0.998	0.992	0.976

Table 4.7: Precision and recall of finding multiple changes in a random walk without outliers.

We see that without outliers, all of the change points are usually detected, although the second and the third change point are detected a little more often than the first one. This is probably because Δ_1 was smaller than Δ_2 and Δ_3. The average number of change points found, when using data without outliers is approximately 2.9. This indicates that the poor precision in table 4.6 is due to the outliers. Apparently, even though a Hampel filter and an absolute cost function have been used, the outliers still affected the outcome of the change point detection method.

Another reason for the low precision is that the detection of a single change point also shows limitations; in the best case the single change point detection method was able to detect the right change point in 81.6% of the time.

Something else that plays a role is the that the method of detecting a single change point often fails if multiple changes are in the time series. Therefore if the random interval selected by the WBS algorithm contains two or more change points, it will most likely select a false change point. Since there are in fact changes in the interval, using ζ_N can result in accepting the change point, which means we detect more change points than we should.

4.6.3. Conclusion change point detection tests

The precision of the best performing change point detection methods for each type of change point is shown if table 4.8. If we would have to choose one method to detect any of these four types of change points it would be applying a Hampel filter with an absolute cost function and estimator l_G.

Change point type	Cost function	Estimator	Type of filter	Data	$\tau = \tau_c$	$\lvert\tau - \tau_c\rvert \le 2$
Level, $\delta = 0$	Absolute	Median	Hampel filter	X_t	0.534	0.816
Drift	Biweight	Mean	Median filter	∇X_t	0.046	0.216
Level, $\delta \neq 0$	Absolute	l_G	Hampel filter	X_t	0.364	0.604
Level and drift	Absolute	l_G	Hampel filter	X_t	0.492	0.76

Table 4.8: Overview of best performing single change point detection methods for each type of change point and their precision.

Using this method to detect a single change point in WBS algorithm 4, we get a precision of 0.36 and a recall of 0.886. This means that although the algorithm detects too many change points, usually the actual change points are detected.

5 Model for Spider Accepted data

In machine learning and statistics different kinds of models are used to describe a system and to make inference. A commonly used model is the *hidden Markov model*. We will first introduce this model and name some of its properties. In section 5.1 we will further elaborate on how the hidden Markov model is used to model the spider accepted data. Some analytic results about how we can derive information when using this model are stated in section 5.2. After this, we will describe how this model can be implemented in section 5.3. In section 5.4 we will describe how the initial values are picked and how we can use the change point detection techniques from chapter 4 in this process.

We will first introduce the notion of a *discrete state Markov chain* or *discrete state Markov process*.

Definition 1. ***Discrete state Markov chain*** *or* ***discrete state Markov process***
Let $(\Omega, \mathscr{F}, P)$ *be a probability space with filtration* $(\mathscr{F}_t, t \in \mathbb{N})$ *and state space* $(Z, \mathscr{Z})$ *a measurable space. Then the process* Y_t *that takes values in* Z *is a discrete time Markov Chain if the Markov Property holds:*

$$\mathbb{P}(Y_t | Y_{t-1}, Y_{t-2}, ..., Y_0) = \mathbb{P}(Y_t | Y_{t-1}) \tag{5.1}$$

In case the state space is not finite, we have a *continuous state Markov process*; here the Markov property holds for the probability distributions instead of the probability itself:

$$f(y_t | y_{t-1}, y_{t-2}, ..., y_0) = f(y_t | y_{t-1}) \tag{5.2}$$

Let Y_t be a Markov chain. In case the corresponding state space Z is discrete, we can define the transition probability as the probability of the Markov chain moving to a next state, given the previous state.

Definition 2. ***Transition probability***
If Y_t *a discrete time Markov Chain, with finite state space* Z, *the transition probability from state* $y_i \in Z$ *to* $s_j \in Z$ *is:*

$$\mathbb{P}(Y_t = s_j | Y_{t-1} = y_i) = p_{ij} \tag{5.3}$$

Since we have a finite number of possible states for the Markov Chain Y_t, we can put these transition probabilities in a matrix, the *transition probability* matrix:

$$\mathscr{P} = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ p_{n1} & \cdots & \cdots & p_{nn} \end{pmatrix} \tag{5.4}$$

Note that the rows of $\mathscr{P}$ sum up to 1. If the Markov chain has continuous states, the transition probability is a probability distribution $f(y_t | y_{t-1})$.

Markov processes are useful tools for modelling. For a discrete state space, one of the advantages is that if the transition probabilities are known and we are given data of the current state of the Markov process, we can easily calculate what the expected value in the future will be:

$$\mathbb{E}(Y_{t+1}|Y_t = y_i) = \sum_j p_{ij} y_j \tag{5.5}$$

An extension to this Markov process is the hidden Markov process. Here not the observed valuable itself is a Markov chain, but a *latent* or *hidden* variable is. This hidden state Y_t determines the distribution of the observed series X_t. Here the density of X_t does not depend on $X_s, s < t$ and can be observed, unlike the hidden Markov process Y_t. A graphical interpretation of the model is shown in figure 5.1

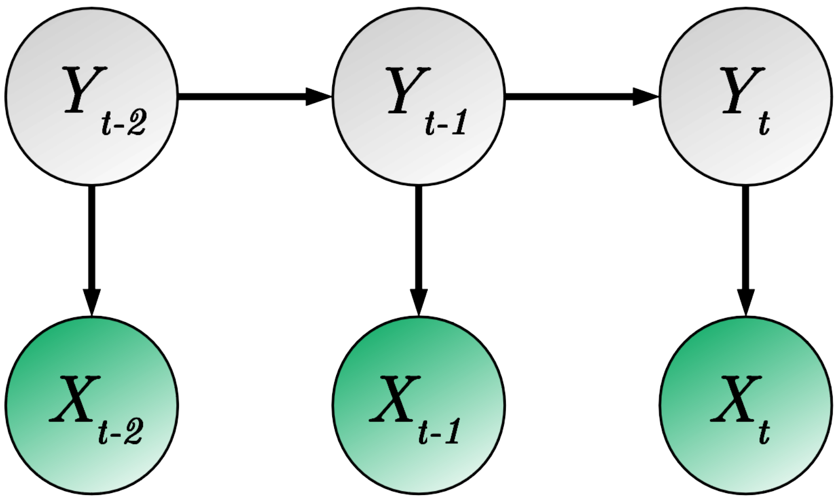


Figure 5.1: Hidden Markov model with hidden states Y_t and our time series X_t.

Here Y_t is a Markov process, and the probability density of X_t given hidden state Y_t is known.

Definition 3. ***Emission probability***
The emission probability density of a hidden Markov model with hidden state Y_t and observed variable X_t is:

$$f(x_t|y_t) \tag{5.6}$$

5.1. Model description

We will assume the spider accepted data is generated according to a hidden Markov model. Here the hidden Markov process contains information about the state the spider is in. The state of the spider describes what the situation of the spider is. We will consider five possible spider states:

1. The spider shows normal behavior
2. The spider shows abnormal behavior for a day
3. The spider is zero for a day
4. The spider is broken
5. The spider is zero for several days

We will denote the spider state at time t as T_t. The probability of the state transitioning is captured in a transition matrix $\mathscr{P}$ of the hidden Markov model. These transition probabilities are constant in time and may be learned from past observations. Note that we do not observe the state, but we observe the time series resulting from these states. Depending on the state the process is in, the time series will show different behaviour. The model for X_t given T_t is:

1. X_t is generated according to a random walk
2. X_t is an outlier
3. $X_t = 0$
4. X_t is an outlier or zero
5. $X_t = 0$

Note that the model of state 3 and state 5 is the same. Here the difference is that the spider will only be in state 3 for a single day; this is not problematic, whereas the spider being zero for multiple day (state 5) is a problem. Because we want to distinguish these two cases, they are modelled as separate spider states. We first elaborate on the hidden state Y_t, then we will discuss the transition probability of Y_t and after this the emission probability distributions $f(x_t|y_t)$ will be described.

5.1.1. Hidden sequence Y_t

In our model, the hidden process Y_t consists of two part; the spider state denoted as $T_t \in \{1,2,3,4,5\}$ indicating in which of the five spider states the spider is in at time t and $A_t \in \mathbb{R}$, the typical normal amount of vacancies of the spider at time t. Here T_t is a discrete Markov chain with transition probability matrix $\mathscr{P}$. The normal amount of vacancies A_t is assumed to be a random walk:

$$A_t = A_{t-1} + Z_{t-1} \tag{5.7}$$

$$Z_{t-1} \sim N(0,\sigma^2) \tag{5.8}$$

Here Z_{t-1} is a normally distributed random variable with mean 0 and fixed standard deviation σ. Note that A_t is continuous and:

$$f(a_t|a_{t-1}, a_{t-2}, ...,) = f(a_t|a_{t-1}) \tag{5.9}$$

This means A_t is a continuous Markov chain. By writing:

$$Y_t = \begin{pmatrix} T_t \\ A_t \end{pmatrix} \tag{5.10}$$

therefore Y_t is indeed a continuous Markov process. In figure 5.2 the hidden sequence with components T_t is shown, with its dependencies.

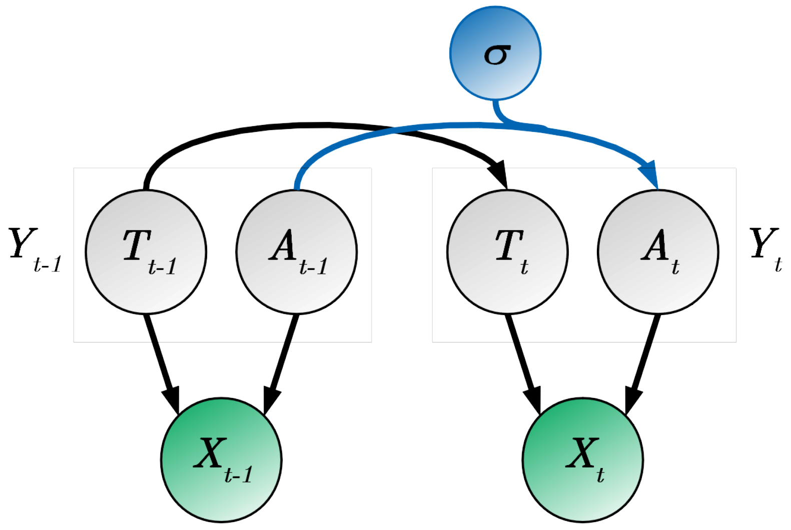


Figure 5.2: Graphical representation of hidden state Y_t.

5.1.2. Transition probability

Because T_t is a discrete Markov process, its transition probabilities $\mathscr{P}$ can be written in matrix form (5.4). We will elaborate on what the exact value of the transition probabilities in this matrix should be in section 5.3.2.

Since A_t is continuous the transition probabilities for A_t cannot be written in matrix form, but as a continuous probability distribution. From (5.7) we see that:

$$A_t|A_{t-1} \sim N(A_{t-1}, \sigma^2) \tag{5.11}$$

So A_t is normally distributed with mean A_{t-1} and standard deviation σ.

5.1.3. Emission probability

The form of the emission probabilities is assumed to depend on the spider state T_t, as described in section 5.1. Here we will elaborate on the precise distributions.

- $T_t = 1$

  We will assume that the spider, if it is in a normal situation, can be modelled as a random walk. To this end we modelled the normal value of the spider A_t as a random walk as well. In case the spider shows normal behaviour, we therefore expect that $X_t \approx A_t$.

  We assume:

$$X_t = A_t + Z_t \tag{5.12}$$

  Here $Z_t \sim N(0, \sigma^2)$. So as emission distribution we get:

$$f_t^1(x_t, a_t, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x_t - a_t)^2}{2\sigma^2}} \tag{5.13}$$

- $T_t = 2$

  We are dealing with an outlier. Observing an outlier means the spidering is not close to the normal value A_t; the amount of vacancies found is significantly higher or lower than what we would expect. It often happens that not zero, but a lot less vacancies are found by the spider than what we would consider normal. How much lower exactly is hard to say; when the spider coverage is incomplete it could be anything between zero and the normal value. Therefore we will use a uniform distribution to model the value of an outlier. Outliers higher than A_t also occur, but are not expected to be much higher than the normal value A_t. Because of this, we will set the uniform distribution up to $1.5A_t$. Here we will exclude zero, because it will be treated separately. The uniform probability density is:

$$f_t^U(x_t|a_t) = \frac{1}{1.5a_t} \text{ for } x_t \in (0, 1.5a_t] \tag{5.14}$$

  Although we don't expect outliers to take extremely high values, there is a possibility they will get larger than $1.5A_t$. Therefore we will use the heavy-tailed pareto distribution to model the probability of high outliers. Because we want the total distribution to be continuous, the uniform and pareto distribution will be patched together at $1.5A_t$. The used pareto distribution is:

$$f_t^p(x_t|a_t) = 2\frac{(1.5a_t)^2}{x_t^3} \text{ for } x_t \in [1.5a_t, \infty) \tag{5.15}$$

  To construct a continuous probability distribution on $(0, \infty)$, both parts of the distribution have to be multiplied by a scaling factor. In order for the distribution to be continuous, we need $f_t^U(x_t = 1.5a_t|a_t) = f_t^p(x_t = 1.5a_t|a_t)$. This can be done by multiplying $f_t^p(x_t|a_t)$ by:

$$\frac{f_t^U(x_t = 1.5a_t|a_t)}{f_t^p(x_t = 1.5a_t|a_t)} \tag{5.16}$$

  The normalization factor for the combined probability density can easily be found:

$$\begin{aligned}\int_0^{\infty} f_t^U(x_t|a_t) + \frac{f_t^U(x_t=1.5a_t|a_t)}{f_t^p(x_t=1.5a_t|a_t)} f_t^p(x_t|a_t)dx_t =& \int_0^{1.5a_t} f_t^U(x_t,a_t)dx_t \\ &+ \int_{1.5a_t}^{\infty} \frac{f_t^U(x_t=1.5a_t|a_t)}{f_t^p(x_t=1.5a_t|a_t)} f_t^p(x_t|a_t)dx_t \\ =& 1 + \frac{f_t^U(x_t=1.5a_t|a_t)}{f_t^p(x_t=1.5a_t|a_t)}\end{aligned}$$

As final outlier distribution we get:

$$f_t^2(x_t|a_t) = \begin{cases} \left(1+\frac{f_t^U(x_t=1.5a_t|a_t)}{f_t^p(x_t=1.5a_t|a_t)}\right)^{-1}\frac{1}{1.5a_t} & \text{for } x \in (0, 1.5a_t] \\ \frac{1}{1+\left(\frac{f_t^U(x_t=1.5a_t|a_t)}{f_t^p(x_t=1.5a_t|a_t)}\right)^{-1}}\frac{2(1.5a_t)^2}{x_t^3} & \text{for } x_t \in (1.5a_t, \infty) \end{cases} \tag{5.17}$$

- $T_t = 3$
  Here $X_t = 0$, so the corresponding emission probability is:

$$f_t^3(x_t) = \delta_0(x_t) \tag{5.18}$$

  Here δ_0 is the Dirac-delta function; in other words, $X_t = 0$ with probability 1.

- $T_t = 4$
  The spider is broken. The distribution of X_t is the same as for $T_t = 2$, but here also a value of zero can occur. We will use a mixture distribution, where $X_t = 0$ with probability p_z and X_t has distribution function f_t^2 with probability $1 - p_z$. We get:

$$f_t^4(x_t|a_t) = p_z\delta_0(x_t) + (1-p_z)f_t^2(x_t|a_t) \tag{5.19}$$

- $T_t = 5$
  Here $X_t = 0$, so the corresponding emission probability is:

$$f_t^5(x_t) = \delta_0(x_t) \tag{5.20}$$

5.2. Analytic inference

We will derive some theoretical results, that can be used to calculate probability densities within the model described above. First, we will find the likelihood function of the hidden states $Y_{1:N}$, $f(y_{1:N})$, for some $N > 1$. Note that the elements of Y_t only depend on other elements in Y_t and on Y_{t-1}. Let $f(y_t|y_{t-1})$ be the distribution of Y_t given Y_{t-1}, then:

$$f(y_t|y_1, y_2, ...y_{t-1}) = f(y_t|y_{t-1}) \tag{5.21}$$

This is the Markov property, so indeed Y_t is a Markov process. Because of this Markov property, we get:

$$f(y_{1:N}) = f(y_1)\prod_{i=2}^{N} f(y_i|y_{i-1}) \tag{5.22}$$

Here $f(y_1)$ is the initial distribution of Y_1.

Since Y_t consists of different components, we can write $f(y_t|y_{t-1})$ as a product of transition probabilities of these components:

$$\begin{aligned}f(y_t|y_{t-1}) =& f(t_t|y_{t-1})f(a_t|y_{t-1},t_t) \\ =& f(t_t|t_{t-1})f(a_t|a_{t-1},t_t)\end{aligned} \tag{5.23}$$

We see using (5.22) that holds:

$$f(y_{1:N}) = f(a_{1:N}, t_{1:N}) = f(a_1, t_1)\prod_{i=2}^{N} f(t_t|t_{t-1})f(a_t|a_{t-1}, t_{t-1}) \tag{5.24}$$

The hidden state Y_t is called hidden because it cannot be measured directly. However, we indirectly have access to information about Y_t by the emission X_t that we observe. So, we would like to be able to say something about the hidden state, given the time series X_t. We would like to find the distribution of Y_t given $X_{1:t}$:

$$f(y_t|x_{1:t})$$

Using the Bayes rule (3.25) we get that:

$$f(y_{1:t}|x_{1:t}) = \frac{f(x_{1:t}|y_{1:t})f(y_{1:t})}{f(x_{1:t})} \tag{5.25}$$

Since the distribution of X_t only depends on Y_t, we get $f(x_{1:t}|y_{1:t}) = f(x_{1:t-1}|y_{1:t-1})f(x_t|y_t)$. Using this, we see:

$$\begin{aligned} f(y_{1:t}|x_{1:t}) &= \frac{f(x_{1:t}|y_{1:t})f(y_{1:t})}{f(x_{1:t})} \\ &= \frac{f(x_{1:t-1}|y_{1:t-1})f(y_{1:t-1})}{f(x_{1:t-1})}\frac{f(x_t|y_t)f(y_t|y_{t-1})}{f(x_t|x_{1:t-1})} \\ &= f(y_{1:t-1}|x_{1:t-1})\frac{f(x_t|y_t)f(y_t|y_{t-1})}{f(x_t|x_{1:t-1})} \end{aligned} \tag{5.26}$$

Here:

$$\begin{aligned} f(x_t|x_{1:t-1}) &= \int f(x_t|y_t)f(y_t|x_{1:t-1})dy_t \\ &= \int f(x_t|y_t)\left\{\int f(y_t|y_{t-1})f(y_{t-1}|x_{1:t-1})dy_{t-1}\right\}dy_t \end{aligned} \tag{5.27}$$

To find $f(y_t|x_{1:t})$ we have to integrate out $y_{1:t-1}$ of $f(y_{1:t}|x_{1:t})$. This results in:

$$\int f(y_{1:t}|x_{1:t})dy_{1:t-1} = \int f(y_{1:t-1}|x_{1:t-1})\frac{f(x_t|y_t)f(y_t|y_{t-1})}{f(x_t|x_{1:t-1})}dy_{1:t-1}$$

$$f(y_t|x_{1:t})\int f(y_{1:t-1}|x_{1:t-1})dy_{1:t-1} = f(x_t|y_t)\frac{\int f(y_{1:t-1}|x_{1:t-1})f(y_t|y_{t-1})dy_{1:t-1}}{f(x_t|x_{1:t-1})}$$

$$f(y_t|x_{1:t}) = f(x_t|y_t)\frac{\int f(y_t|y_{t-1})f(y_{t-1}|y_{1:t-1})dy_{t-1}\int f(y_{1:t-1}|x_{1:t-1})dy_{1:t-1}}{\int f(y_{1:t-1}|x_{1:t-1})dy_{1:t-1}f(x_t|x_{1:t-1})}$$

$$f(y_t|x_{1:t}) = f(x_t|y_t)\frac{\int f(x_t|y_{t-1})f(y_{t-1}|x_{1:t-1})dy_{t-1}}{f(x_t|x_{1:t-1})}$$

We get:

$$f(y_t|x_{1:t}) = \frac{f(x_t|y_t)f(y_t|x_{1:t-1})}{f(x_t|x_{1:t-1})} \tag{5.28}$$

Here:

$$f(y_t|x_{1:t-1}) = \int f(y_t|y_{t-1})f(y_{t-1}|x_{1:t-1})dy_{t-1} \tag{5.29}$$

In theory (5.27), (5.28) and (5.29) could be calculated sequentially. In practice however, calculating 5.28 exactly is often not possible because of the embedded integrals. There are some cases where an exact solution is available, such as the well know Kalman Filter [9]. For our model however, we will have to make use of a numerical approximation to calculate $f(y_t|x_{1:t})$.

5.3. Model implementation

The difficulty of using the model proposed earlier is that we don't know what value for σ to use. It can vary for different websites and is important because it is a measure for how "unstable" a spider is. A large value of σ means A_t is likely to vary much over time, so big differences in the amount of jobs accepted by the spider are assumed to be likely. If σ is small, only a small difference from the last day are expected. The value of σ will be estimated using data from the past. Because σ only appears in the distribution in case $T_t = 1$, we will only use the data points where the $T_t = 1$.

For the implementation of the algorithm it is useful to extend the hidden state Y_t, adding 3 components in order to estimate σ for each spider. We will call this extended hidden state S_t as hidden state in the HMM, instead of Y_t.

S_t contains all information to determine the emission density $f(x_t|s_t)$. We will assume that $S_t \in \{1,2,3,4,5\} \times \mathbb{R} \times \mathbb{R} \times \mathbb{N} \times \mathbb{R}$ and write:

$$S_t = \begin{pmatrix} T_t \\ A_t \\ \sigma_t \\ n_t \\ l_t^A \end{pmatrix} = \begin{pmatrix} Y_t \\ \sigma_t \\ n_t \\ l_t^A \end{pmatrix} \tag{5.30}$$

Here:

- T_t is the state of the spider itself
- A_t is the value of the spider if it is showing normal behaviour
- σ_t is an estimation of the variance of the difference of A_t
- n_t is the number of points in time where $T_t = 1$, up to time t
- l_t^A is the value of A_s when the spider state was $T_s = 1$ for the last time, for $s \leq t$

Note that σ_t, n_t and l_t^A are only used to estimate σ; σ itself is still assumed to be constant. In fact σ_t, n_t and l_t^A are calculated alongside the hidden Markov model, but don't have transition probabilities since they will be calculated deterministically. The results from sections 5.1 and 5.2 still apply, where we take $Y_t = S_t$ and $y_t = s_t$.

5.3.1. Estimating σ

The standard deviation σ will be estimated using previous data and the previous states of the hidden sequence Y_t. We denote σ_t the estimate of σ at time t. Recall that in case $T_t = 1$, so the spider shows normal behaviour, we have:

$$A_t - A_{t-1} \sim N(0, \sigma^2) \tag{5.31}$$

We will estimate σ by using the sample variance (3.9), so it is calculated in a deterministic way. An advantage of using the sample variance, opposed to another estimate as described on section 3.2.2, is that we can update the estimate in a sequential way, as we will see. Since $A_t - A_{t-1} \sim N(0, \sigma^2)$, the sample variance is:

$$\sigma_t^2 = \frac{1}{n-1} \sum_{i=1}^{n} (A_i - A_{i-1})^2 \tag{5.32}$$

Rewriting this gives:

$$\sigma_t^2 = \frac{n-2}{n-1} \frac{1}{n-2} \sum_{i=1}^{n-1} (A_i - A_{i-1})^2 + \frac{1}{n-1} (A_i - A_{i-1})^2 \tag{5.33}$$

So, we get:

$$\sigma_t^2 = \frac{n-2}{n-1} \sigma_{t-1}^2 + \frac{1}{n-1} (A_i - A_{i-1})^2 \tag{5.34}$$

So, only σ_{t-1}, A_{t-1} and A_t are needed for the estimation of σ_t. Here we assumed the spider is showing normal behaviour at every time t, in other words $T_t = 1 \ \forall \ t$. This is not always the case; therefore we will only use the values of A_t that have $T_t = 1$. In order to do this we will keep track of the last value of A where $T = 1$, l_t^A. We will also save the amount of data point where T has been equal to 1 up to time t, n_t. Note that by doing this, the outliers are filtered out and therefore there is no need to use a robust estimator for σ.

Computing the values of l_t^A and n_t is straightforward:

$$l_t^A = \begin{cases} A_t & \text{if } T_t = 1 \\ l_{t-1}^A & \text{else} \end{cases} \tag{5.35}$$

$$n_t = \begin{cases} n_{t-1} + 1 & \text{if } T_t = 1 \\ n_{t-1} & \text{else} \end{cases} \tag{5.36}$$

We will only update σ_t if $T_t = 1$. So we can update estimate σ_t^2 as follows:

$$\sigma_t^2 = \begin{cases} \frac{n_t-2}{n_t-1}\sigma_{t-1}^2 + \frac{1}{n_t-1}\left(A_t - l_{t-1}^A\right)^2 & \text{if } T_t = 1 \\ \sigma_{t-1}^2 & \text{else} \end{cases} \tag{5.37}$$

Note that at the beginning, only a limited amount of data points n_t has been available to calculate the estimation of σ, σ_t. We will define a threshold th_σ, indicating what the minimum value of n_t should be before we can use σ_t as estimation of σ. Also we define σ_{start}, a starting value of sigma that can be used in case $n_t < th_\sigma$. We get as estimate $\hat{\sigma}$ for σ:

$$\hat{\sigma} = \begin{cases} \sigma_{start} & \text{if } n_t < th_\sigma \\ \sigma_t & \text{if } n_t \geq th_\sigma \end{cases} \tag{5.38}$$

5.3.2. Estimating $\mathscr{P}$

The spider state $T_t \in \{1,2,3,4,5\}$ has a finite state space and is a Markov chain. Therefore we can specify the transition probabilities in a transition probability matrix $\mathscr{P}$ as in (5.4). Using the past outcome of the particle filter, we would like to estimate the values in this matrix. First, we will assume the matrix has the following form:

$$\mathscr{P}(p_{12}, p_{13}, p_{14}, p_{35}) = \begin{pmatrix} 1-(p_{12}+p_{13}+p_{14}) & p_{12} & p_{13} & p_{14} & 0 \\ 1-(p_{22}+2*p_{13}+p_{14}) & p_{12} & 2*p_{13} & p_{14} & 0 \\ 1-(3*p_{12}+p_{35}) & 3*p_{12} & 0 & 0 & p_{35} \\ 0 & 0 & 0 & 1 & 0 \\ 1-(p_{12}+p_{14}+0.75) & p_{12} & 0 & p_{14} & 0.75 \end{pmatrix} \tag{5.39}$$

This might seem a bit arbitrary, but some thought has been put into it. The idea is that the probability of transitioning to state 2 is a multiple of the probability of transitioning from state 1 to state 2, p_{12}. The same thing is assumed for states 3 and 4, but with p_{13} resp. p_{14}. Note that $p_{32} > p_{12}$. This is because if a spider has dropped to zero, so is in state 3, if often happens that the day after the amount of vacancies accepted by the spider is not back to normal yet, but somewhere halfway or its accepting more vacancies than expected. This is not a problem if the spider turns back to normal two days after it was zero. To incorporate this in the model, we assume that the transition probability from zero to an outlier, p_{32}, is bigger than transitioning to 2 from another state. The reverse is also true; after an outlier there is a bigger probability of the spider dropping to zero, although this effect is not as big. Therefore p_{23} is chosen to be larger than p_{13}. We assume that once the spider is broken, it will remain broken. For this reason state 4 is called an *absorbing state*. We get $p_{44} = 1$ and $p_{4i} = 0$ if $i \neq 4$.

Also we will assume that if the spider is zero for one day, in state 3, it can only transition into state 2, 5 or 1, meaning that if it is zero for another day we are in state 5 and it doesn't get into the broken state 4 right away, but an outlier state 2 is in between. State 5 can only be reached after being state 3 or 5, meaning the spider had to be zero already. This makes sense, since state 5 represents being zero for multiple days. Unlike state 4, we will assume state 5 is not absorbing and we allow the possibility that the spider is restored by itself (for example in case a website has been down for several days but comes back online). The value $p_{55} = 0.75$ is fixed.

By using this form of $\mathscr{P}$, we only have to estimate p_{12}, p_{13}, p_{14} and p_{35}. We will do this by looking at the outcome of the particle filter. It tells us what the most likely spider T_t was in the past times $s \leq t$. If we have enough information, we should therefore be able to estimate the transition probabilities we need by looking at how many times this transition took place in the past. In order to do this we need at least 100 data points where it calculated the most likely state of T. We will denote the number of data points in state i as n_i, and the number of transitions from state i to state j as n_{ij}. Note that $n_i - \mathbb{1}_{(T_t=i)}$ is the number of transitions where the transition started in state i. Now we can calculate ratio of transitioning from state i to state j by dividing n_{ij} by $n_i - \mathbb{1}_{(T_t=i)}$. In the case of p_{14} we will take any transition that transitioned to 4, since this is quite unlikely to happen. To ensure these transition probabilities are all nonzero, we will take the maximum of the calculated transition probability and a minimum value of $p_{ij} = 0.01$. problems also occur in case $p_{ij} = 1$; therefore we set p_{12}, p_{13}, p_{14} and p_35 to be 0.8 at most. We get:

- $p_{12} = \min(\max(0.01, \frac{n_{12}}{n_1 - \mathbb{1}_{(T_t=1)}}), 0.8)$
- $p_{13} = \min(\max(0.01, \frac{n_{13}}{n_1 - \mathbb{1}_{(T_t=1)}}), 0.8)$
- $p_{35} = \min(\max(0.01, \frac{n_{35}}{n_3 - \mathbb{1}_{(T_t=3)}}), 0.8)$
- $p_{14} = \min(\max(0.01, \frac{n_{14}+n_{24}+n_{34}+n_{44}+n_{54}}{n_1+n_2+n_3+n_4+n_5-1}), 0.8)$

The estimate of the transition probability matrix at time t will be denoted as $\mathscr{P}_t$.

5.4. Initial values

The parameters in the model for the spider accepted data may be different depending on the website, as are the transition probabilities of the hidden Markov model. These will be learned from the data as time progresses. To do this however, initial values have to be chosen. Depending on the hidden spider state T_1 we will pick the other components of S_1. We will assume we have the last 30 data points $X_{-29:0}$ available to us.

First we will describe the choice of the transition probability matrix $\mathscr{P}_1$ containing the transition probabilities of T_t and the procedure for setting $A_1, \sigma_1 n_1, l_1^A$ given T_1. After this we will elaborate on how to estimate σ_{start} from the historical data. In section 5.4.1 a simulation study has been performed on a random walk with drift to find out how σ can best be estimated.

We will start by an initial transition probability matrix $\mathscr{P}_1$:

$$\mathscr{P}_1 = \begin{pmatrix} 0.89 & 0.05 & 0.05 & 0.01 & 0 \\ 0.84 & 0.05 & 0.1 & 0.01 & 0 \\ 0.55 & 0.15 & 0 & 0 & 0.3 \\ 0 & 0 & 0 & 1 & 0 \\ 0.19 & 0.05 & 0 & 0.01 & 0.75 \end{pmatrix} \tag{5.40}$$

Note that $\mathscr{P}_1$ is equal to (5.39) using $p_{12} = 0.05$, $p_{13} = 0.05$ and $p_{14} = 0.01$.

Depending on the value of T_1, we can set A_1, σ_t, n_1 and l_1^A:

- If $T_1 = 1$ we set:
  $A_1 = X_1$
  $\sigma_1 = \emptyset$
  $n_1 = 0$
  $l_1^A = A_1$.
  Note that in this case $n_1 = 0$, because we use the difference of A to estimate σ and the number of differences we have is equal to zero.

- If $T_1 \neq 1$ we set:
  $A_1 = \text{median}(X_{-9:0})$
  $\sigma_1 = \emptyset$
  $n_1 = -1$
  $l_1^A = \emptyset$.
  Here $n_1 = -1$, because we don't have any data yet that can be used to estimate σ, therefore we need to have at least two more data points before we have a difference that can be used for the estimation.

Note that in practice, T_1 is not know. The best thing to do is to get a human assessment of T_1; however we can also assume $T_1 = 1$, since this is the most common situation. In case $T_1 \neq 1$ this will show soon enough.

5.4.1. Initial value $\sigma_{\textbf{start}}$

Before enough data is available to estimate σ as σ_t, an initial value σ_{start} is used to estimate σ as stated in (5.38). To find an appropriate value for σ_{start} we will use the past data $X_{-29:0}$. This is assumed to be a random walk with drift and outliers, possibly with changes in drift and/or level. Using change point detection techniques described in section 4.5 we are able to find the segments of the random walk in which it shows stable behaviour, meaning there are no change points within each segment. We will use the last segment to calculate σ_{start}. However again, in order for this estimation to make sense the last segment has to be longer than a threshold s_{th} (not to be confused with th_σ, which can be different). If the last segment has length smaller than s_{th}, a default value of $\sigma_{start} = \frac{1}{50} \max X_{-29:0}$ is used.

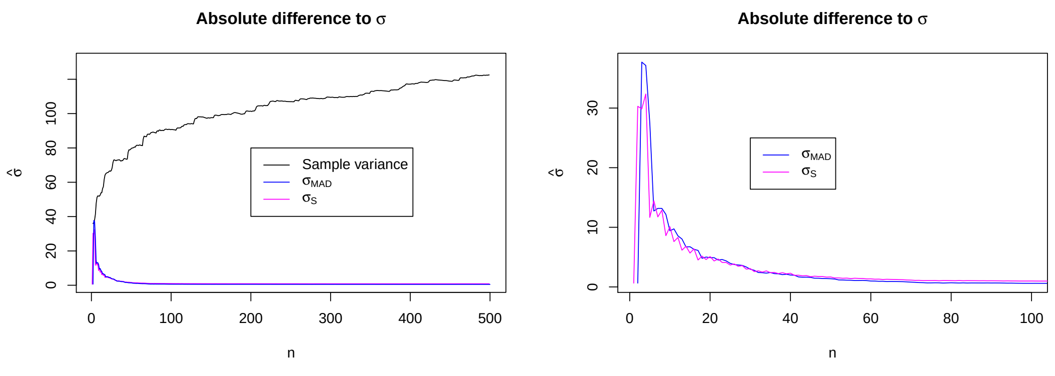
If the last segment has length longer than s_{th}, we can use one of the estimators described in section 3.2.2 to calculate σ_{start}, which are the sample variance (3.9), σ_{MAD} (3.19) or σ_S (3.20). Recall that:

$$\nabla X_t = Z_t + \delta \tag{5.41}$$

Therefore $\nabla X_t \sim N(\delta, \sigma^2)$. This means we can estimate the standard deviation σ by applying one of the estimators for σ to ∇X_t.To find out which estimator to use, a simulation study has been performed. In the end we want to use the estimator that is robust in the presence of outliers and converges to the true value of σ as quickly as possible, so only a limited amount of data points is required for a good estimation. The data used for this test is generated using (4.37). Here we assumed a drift $\delta = 0.1$, $\sigma = 1$ and $X_0 = 100$. The outliers are generated using outlier probability $p_o = 0.3$. We simulated 500 data sets of length $N = 500$ and calculated the estimators $\hat{\sigma}_n$ for an increasing amount of n data points. The absolute differences $|\hat{\sigma}_n - \sigma|$ are taken; of these the mean will be calculated.

5.4.2. Result simulation study $\sigma_{\textbf{start}}$

The mean of the absolute difference of the sample variance, σ_{MAD} and σ_S are plotted in figure 5.3.



(a) All estimators. (b) σ_S,σ_{MAD}.

Figure 5.3: Absolute difference of estimator to real value of σ, versus number of used data points n.

We see that the results of using the sample variance are only getting worse as more data points are used for the estimation. The reason for this is that the sample variance is not a robust method of estimating the standard deviation, therefore the outliers cause the estimate to be much higher than it should. We also see that σ_S and σ_{MAD} are very close in terms of error; the performance of σ_S seems to be slightly better. To see the spread in answers, a box plot for a sample size of 10 is plotted in figure 5.4.

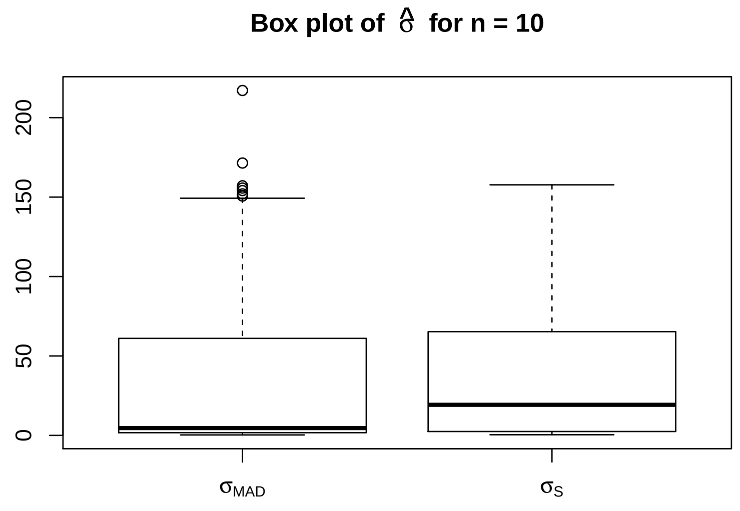


Figure 5.4: Box plots of σ_{MAD} and σ_S for sample size $n = 10$.

We can see that the maximum values in the case of using σ_{MAD} are bigger than those if we use σ_S; there are in fact quite some outliers. However the median of σ_{MAD} is much closer to the real value $\sigma = 1$ than the median of σ_S. Also the third quantile is closer for σ_{MAD}. Therefore there is a slight preference for using σ_{MAD} over σ_S, although the difference is not big. Also note that using σ_{MAD} is easier to implement, since there is a standard function in R for this statistic.

6 Particle filter

We are interested in the value of the hidden state process Y_t of a hidden Markov model, as described in chapter 5. Since we cannot observe the hidden process Y_t directly, we can only use the observed time series X_t. Therefore we would like to find the likelihood $f(y_t|x_1,...,x_t)$. An analytical solution to finding this density in a sequential way is available (5.28), however in practice it is often not possible to calculate all the intergrals needed for this recursion. Even if it is, this can computationally be very expensive. A commonly used computational method to approximate this likelihood is the *particle filter* [28] [29] [30] [31].

In this chapter we will start by explaining *importance sampling* in section 6.1. After this several particle filters are described, the basic particle filter *sequential importance sampling* in section 6.2, the *sequential importance resampling* in section 6.3 and the *Bootstrap filter* in section 6.4. We will see how this Bootstrap filter can be applied to make inference in our model for the spider accepted data in section 6.5.

6.1. Importance Sampling

Importance sampling is a classical technique that can be used to estimate properties from a complicated distribution $f(x)$, only using samples from a different distribution that we can more easily sample from. We define the *importance sampling distribution* $\pi(x)$ that we are able to sample from and has support larger than $f(x)$. Assume we are interested in computing expectations of the form:

$$\overline{h} = \int h(x)f(x)dx \tag{6.1}$$

If we are able to generate random samples $\{x^i\}_{1\le i\le N}$ from f and f is normalized, we can approximate $\overline{h}$ by:

$$\overline{h} \approx \frac{1}{N}\sum_{i=1}^{N} h(x^{(i)}) \tag{6.2}$$

The problem is that we cannot sample directly from f; therefore we use the importance sampling distribution π. We will sample $\{x^i\}_{1\le i\le N}$ from π and by writing:

$$\widehat{\omega}^{(i)} = \frac{f(x^{(i)})}{\pi(x^{(i)})} \tag{6.3}$$

We will also assume f is not necessarily normalized. This way we get:

$$\begin{aligned}\overline{h} &= \frac{\int h(x)\frac{f(x)}{\pi(x)}\pi(x)dx}{\int f(x)dx}\\ &\approx \frac{\frac{1}{N}\sum_{i=1}^{N}\widehat{\omega}^{(i)}h(x^{(i)}}{\frac{1}{N}\sum_{j=1}^{N}\widehat{\omega}^{(j)}}\\ &= \sum_{i=1}^{N}\frac{\widehat{\omega}^{(i)}}{\sum_{j=1}^{N}\widehat{\omega}^{(j)}}h(x^{(i)})\end{aligned}$$

We will call $\widehat{\omega}^{(i)}$ the *unnormalized importance weight* and define the *normalized importance weight* as:

$$\omega^{(i)} = \frac{\widehat{\omega}^{(i)}}{\sum_{j=1}^{N} \widehat{\omega}^{(j)}} \tag{6.4}$$

Now we see:

$$\overline{h} \approx \sum_{i=1}^{N} \omega^{(i)} h(x^{(i)}) \tag{6.5}$$

This means the pairs $\{x^{(i)}, \omega^{(i)}\}_{1\leq i\leq N}$ is a numerical approximation of the distribution of $f(x)$, where $x^{(i)}$ is called the *particle*, hence the name particle filter. The distribution f now approximately is:

$$f(x) \approx \begin{cases} x^{(1)} & \text{with prob. } \omega^{(1)} \\ x^{(2)} & \text{with prob. } \omega^{(2)} \\ \dots & \dots \\ x^{(N)} & \text{with prob. } \omega^{(N)} \end{cases} \tag{6.6}$$

6.2. Sequential Importance Sampling (SIS)

Importance sampling can be used sequentially as well, in the context of a hidden Markov moddel. Assume a hidden Markov model as described in chapter 5 with hidden state Y_t and observables X_t. We will start with a set of particles $y_0^{(i)}$, sampled from an initial distribution $f_0(y_0)$. The procedure consists of two steps:

- **Correction**
  Calculate the normalized importance weights $\left\{\omega_t^{(i)}\right\}_{1\leq i\leq N}$
- **Prediction**
  The particles $\left\{y_t^{(i)}\right\}_{1\leq i\leq N}$ are propagated to $\left\{y_{t+1}^{(i)}\right\}_{1\leq i\leq N}$ by sampling using transition probability distribution $f(y_{t+1}|y_t^{(i)})$

The goal is to approximate the likelihood $f(y_t|x_1,...,x_t)$ at time t by $\left\{y_i^{(t)}, \omega_i^{(t)}\right\}_{1\leq i\leq N}$. By increasing the value of N, we will have more particles and the approximation will be more accurate as well.

We will derive a recursion to calculate the importance weights. Recall that we have (5.26):

$$f(y_{1:t}|x_{1:t}) = f(y_{1:t-1}|x_{1:t-1}) \frac{f(x_t|y_t)f(y_t|y_{t-1})}{f(x_t|x_{1:t-1})} \tag{6.7}$$

Since $f(x_t|x_{1:t-1})$ is a normalization constant that doesn't depend on Y, we have:

$$f(y_{1:t}|x_{1:t}) \propto f(y_{1:t-1}|x_{1:t-1}) f(x_t|y_t) f(y_t|y_{t-1}) \tag{6.8}$$

Also recall that by integrating over $y_{1:t-1}$ we got (5.29):

$$f(y_t|x_{1:t}) = \int \frac{f(x_t|y_t)f(y_t|y_{t-1})f(y_{t-1}|x_{1:t-1})}{f(x_t|x_{1:t-1})} dy_{t-1} \tag{6.9}$$

So if we want to approximate this integral using importance sampling, we set $h = \frac{1}{f(x_t|x_{1:t-1})}$ and $g(x_{1:t-1}, y_t, y_{t-1}) = f(x_t|y_t)f(y_t|y_{t-1})f(y_{t-1}|x_{1:t-1})$. Then:

$$f(y_t|x_{1:t}) = \int h(x_t) g(x_{1:t-1}, y_t, y_{t-1}) dy_{t-1} \tag{6.10}$$

We will use an importance sampling function π that can by propagated in time, without modifying the trajectories that have already been simulated; so for π holds:

$$\pi(y_{1:t}|x_{1:t}) = \pi(y_{1:t-1}|x_{1:t-1}) \pi(y_t|y_{1:t-1}, x_{1:t}) \tag{6.11}$$

The unnormalized importance weights can therefore be calculated using (6.3) as:

$$\begin{aligned}\widehat{\omega}_t^{(i)} &= \frac{g(x_t, y_t, y_{t-1})}{\pi(y_{1:t}|x_{1:t})} &&(6.12)\\ &= \frac{f(y_{1:t-1}|x_{1:t-1})f(y_t|y_{t-1})f(x_t|y_t)}{\pi(y_{1:t}|x_{1:t})}\\ &= \frac{f(y_{1:t-1}|x_{1:t-1})}{\pi(y_{1:t-1}|x_{1:t-1})}\frac{f(y_t|y_{t-1})f(x_t|y_t)}{\pi(y_t|y_{1:t-1}, x_{1:t})}\\ &\propto \widehat{\omega}_{t-1}^{(i)}\frac{f(y_t|y_{t-1})f(x_t|y_t)}{\pi(y_t|y_{1:t-1}, x_{1:t})} &&(6.13)\end{aligned}$$

Here we used (6.8), such that
$f(y_{1:t-1}|x_{1:t-1}) \propto f(y_{1:t-2}|x_{1:t-2})f(x_{t-1}|y_{t-1})f(y_{t-1}|y_{t-2}) = g(x_{1:t-2}, y_{t-1}, y_{t-2})$. After normalizing using (6.4) we have found the importance weights $\omega_t^{(i)}$.

6.3. Sequential Importance Resampling (SIR)

The SIS method works well for short periods of time, but as time increases it has some serious drawbacks and is not effective in the long run. As time progresses, the weights can get highly *degenerate*. This means the weight mass is concentrated in a few particles, while the other particles only have very low weights; as a result there effectively are only a few particles that make up the numerical probability density that is the outcome of the filter. To overcome this problem, *resampling* can be used. The idea is that when the weights become too degenerate, we will take a new sample, using the particles and their corresponding weights as probability distribution. To determine when to resample, we will use the *effective number of particles*, defined as:

$$N_{eff} = \frac{1}{\sum_{i=1}^{N}\left(\omega_t^{(i)}\right)^2} \tag{6.14}$$

If N_{eff} gets below a predetermined threshold value, the will resample according to the importance weights. After resampling, the weights are reset to $\omega_t^{(i)} = \frac{1}{N}$. It is also possible to resample at every iteration, obsoleting the use of N_{eff}.

6.4. Bootstrap filter

We will now focus on the choice of the importance sampling distribution π. A common choice is to use the transition probability $f(y_t|y_{t-1})$ as importance distribution. Doing so will result in the so called *Bootstrap filter*. We see that indeed equation (6.11) holds:

$$f(y_{1:t}|y_{1:t-1}) = f(y_{1:t-1}|y_{1:t-2})f(y_t|y_{t-1})$$

By using this importance function, the unnormalized importance weights can be calculated using equation (6.13) as:

$$\begin{aligned}\widehat{\omega}_t^{(i)} &= \widehat{\omega}_{t-1}^{(i)}\frac{f(y_t|y_{t-1})f(x_t|y_t)}{\pi(y_t|y_{1:t-1}, x_{1:t})}\\ &= \widehat{\omega}_{t-1}^{(i)}\frac{f(y_t|y_{t-1})f(x_t|y_t)}{f(y_t|y_{t-1})}\\ &= \widehat{\omega}_{t-1}^{(i)}f(x_t|y_t) &&(6.15)\end{aligned}$$

We see that this choice of the importance function results in an easy expression for the importance weights. The Bootstrap filter resamples in case $N_{eff} < N_{thr}$ for some threshold N_{thr}.

6.5. Implementation particle filter

Recall that in our implementation, we will estimate σ from the data and the outcome of the particle filter and save it along with the particles. To this end we use the extended hidden state S_t (5.30) instead of Y_t, so we will write $Y_t = S_t$ and $y_t = s_t$. The implementation of this hidden Markov model is described in section 5.3. We will use a Bootstrap filter to calculate the distribution of the hidden state, $f(s_t|x_{1:t})$. This will be done on a daily basis, using the spider accepted data and the outcome of the particle filter for before.

Assume we will use N particles. As a first step, we generate an initial set of particles $\left\{\hat{s}_0^{(i)}\right\}_{1\leq i\leq N}$ as described in section 5.4. We also set a threshold value th_σ, indicating how many data points we need to estimate σ properly and a value σ_{start} that the algorithm can use for σ in case not enough data is available yet. Another threshold N_{eff} has to be set, indicating the minimum amount of effective particles before deciding to resample. Also the transition probability matrix $\mathscr{P}$ is passed to the algorithm. How to construct this $\mathscr{P}$ is described in section 5.3.2.

Now the algorithm will start iterating, on a daily basis. Using the particles of the previous day, next day's set of particles will be generated using $f(s_t|s_{t-1})$:

$$\hat{s}_t^{(i)} \sim f(s_t|s_{t-1} = \hat{s}_{t-1}^{(i)}) \tag{6.16}$$

This is the prediction step. Here the estimator σ_t for σ will only be used if $n_t \geq th_\sigma$. Otherwise σ_{start} is used to calculate the next generation of particles. The corresponding importance weights are calculated using the recursion for the weights (6.15):

$$\hat{\omega}_t^{(i)} = \hat{\omega}_{t-1}^{(i)} f(x_t|s_t = \hat{s}_t^{(i)}) \tag{6.17}$$

This is the correction step. The number of effective particles is calculated and if $N_{eff} < N_{th}$ the particles will be resampled and the weights are all set to $\frac{1}{N}$. Using the importance weights, the density of hidden spider state T_t given the data is estimated using (6.6):

$$\mathbb{P}(T_t = j) = \sum_{i=1}^{N} \omega_t^{(i)} \mathbb{1}_{(t_t^{(i)}=j)} \tag{6.18}$$

The outcome of the algorithm are these probabilities for T_t and the new particles and corresponding importance weights. The procedure is summarized in algorithm (5). The complexity of the algorithm is of $\mathscr{O}(N)$.

```
Input:
Number of particles $N$;
Set of last particles and corresponding weights $(\hat{s}^{(i)}_{t+1}, w^{(i)}{}_{t+1})_{1\le i\le N}$;
Transition probability matrix $\mathscr{P}$ describing the transition probabilities of $T_t$
Threshold $th_\sigma$ indicating the minimum amount of data points needed to estimate $\sigma_t$
Default deviation $\sigma_{start}$ to use if not enough data points are available to estimate $\sigma_t$
Output:
A set of particles and corresponding weights $(\hat{s}^{(i)}_{t+1}, w^{(i)}{}_{t+1})_{1\le i\le N}$ representing $f(s_t|x_{1:t})$;
A vector $\boldsymbol{p}_{t+1}$ of length 5 with $\mathbb{P}(T_{t+1}|x_{1:t+1})$
ParticleFilter($N, (\hat{s}^{(i)}_t)_{1\le i\le N}, (w^{(i)}_t)_{1\le i\le N}, \mathscr{P}, th_\sigma, \sigma_{start}$):
for $i \in \{1,...,N\}$ do
    Generate next set of particles $\hat{s}^{(i)}_{t+1}$
    $\boldsymbol{p} = \mathscr{P}_{t^{(i)}_t, 1\le j\le 5}$
    Sample $t^{(i)}_{t+1}$ from $\{1,2,3,4,5\}$ with probabilities $\boldsymbol{p}$
    if $n^{(i)}_t \ge th_\sigma$ then
        Draw $\epsilon \sim N(0, \sigma^{(i)}_t)$
    else
        Draw $\epsilon \sim N(0, \sigma^{(i)}_{start})$
    end
    $a^{(i)}_{t+1} = a_t + \epsilon$
    if $t^{(i)}_{t+1} = 1$ then
        $l^{A,(i)}_{t+1} = a^{(i)}_{t+1}$
        $n^{(i)}_{t+1} = n^{(i)}_t + 1$
        $\sigma^{(i)}_{t+1} = \sqrt{\frac{n^{(i)}_{t+1}-2}{n^{(i)}_{t+1}-1}\left(\sigma^{(i)}_t\right)^2 + \frac{1}{n^{(i)}_{t+1}-1}\left(a^{(i)}_{t+1} - a^{(i)}_t\right)^2}$
    else
        $l^{A,(i)}_{t+1} = l^{(i),A}_t$
        $n^{(i)}_{t+1} = n^{(i)}_t$
        $\sigma^{(i)}_{t+1} = \sigma^{(i)}_t$
    end
    Calculate unnormalized importance weight $w^{(i)}_{t+1}$
    $\hat{w}^{(i)}_{t+1} = w^{(i)}_t f(x_t|s_t = \hat{s}^{(i)}_{t+1})$
end
Normalize importance weights
for $i \in \{1,...,N\}$ do
    $w^{(i)}_{t+1} = \frac{\hat{w}^{(i)}_{t+1}}{\sum_{i=1}^N \hat{w}^{(i)}_{t+1}}$
end
Resampling
$N_{eff} = \frac{1}{\sum_{i=1}^N \left(\omega^{(i)}_t\right)^2}$
if $N_{eff} < N_{th}$ then
    Sample $(\overline{s}^{(i)}_{t+1})_{1\le i\le N}$ from $(\hat{s}^{(i)}_{t+1})_{1\le i\le N}$ with probabilities $(\omega^{(i)}_{t+1})_{1\le i\le N}$
    Set $(\hat{s}^{(i)}_{t+1})_{1\le i\le N} \longleftarrow (\overline{s}^{(i)}_{t+1})_{1\le i\le N}$
    Set $w^{(i)}_{t+1} = \frac{1}{N}\forall i$
end
Calculate conditional probability $T_t$
for $j \in \{1,2,3,4,5\}$ do
    $p^{t+1}_j = \sum_{i=1}^N w^{(i)}_{t+1} \mathbb{1}_{(t^{(i)}_{t+1}=j)}$
end
$\boldsymbol{p}^{t+1} = \begin{pmatrix} p^{t+1}_1 & p^{t+1}_2 & p^{t+1}_3 & p^{t+1}_4 & p^{t+1}_5 \end{pmatrix}$
return $(\hat{s}^{(i)}_{t+1}, w^{(i)}{}_{t+1})_{1\le i\le N}, \boldsymbol{p}^{t+1}$
```

Algorithm 5: Particle filter algorithm.

7 Decision Making

The purpose of the alarm system described in this thesis is to trigger alarms when a spider seems to be malfunctioning. For this, a decision has to be made every day either to trigger an alarm, or not to trigger an alarm. *Decision making theory* deals with the problem of making such decisions. We will look at the hidden Markov model with hidden state Y_t and spider accepted data X_t, as described in chapter 5. Recall that $Y_t = \begin{pmatrix} T_t \\ A_t \end{pmatrix}$, where T_t is the state of the spider. The spider is broken in case $T_t = 4$ or $T_t = 5$.

We define the state space $\Theta = \{0, 1\}$ of possible states that our decision making process has to choose from. Here $\theta = 0$ means the spider is fine (so $T_t \in \{1,2,3\}$) and $\theta = 1$ means the spider is broken (so $T_t \in \{4,5\}$). Also we define action space $\mathscr{A} = \{0, 1\}$, which contains the possible actions that can be taken. Here $a = 0$ means doing nothing and $a = 1$ is triggering an alarm. This comes down to testing $H_0 : \theta = 0$ versus $H_1 : \theta = 1$, a simple hypothesis test. Note that the spider state T_t may vary over time, which means the same holds for θ. For now however we will assume that θ is constant, even though T_t may change.

Because the algorithm will be run in a sequential fashion, we will be using *sequential decision making theory* for this decision, described in section 7.1. We will describe the *sequential probability ratio test* in section 7.2, a procedure for testing these hypotheses in a sequential way. We will discuss some principles used in Bayesian decision making and discuss a Bayesian method that can easily be used in combination with the particle filter to make sequential decisions about triggering alarms in section 7.3. In section 7.4 we will discuss how, after an alarm has been triggered, the user can reset the particle filter.

7.1. Sequential decision making

Sequential decision making deals with the problem of making a decision in a process, where the data X_t is gathered in a sequential way. This data will be used to pick an action $a \in \mathscr{A}$, where $\mathscr{A}$ is the set of all possible actions.

The task at hand is to come up with a decision procedure, indicating if and when which action should be taken. In a sequential decision problem, this is described in a sequential decision procedure $\boldsymbol{d}(\boldsymbol{\tau}, \boldsymbol{\delta})$.

Definition 4. ***Sequential decision procedure*** $\boldsymbol{d}(\boldsymbol{\tau}, \boldsymbol{\delta})$
A sequential decision procedure

$$\boldsymbol{d}(\boldsymbol{\tau}, \boldsymbol{\delta})$$

is a procedure consisting of stopping rule $\boldsymbol{\tau} = (\tau_0, \tau_1(X_1), \tau_2(X_{1:2}), \ldots,)$*, where* τ_t *indicates the probability of deciding after observing* $X_{1:t}$ *and decision rule* $\boldsymbol{\delta} = (\delta_0, \delta_1(X_1), \delta_2(X_{1:2}), \ldots,)$*, where* δ_t *indicates the action taken after observing* $X_{1:t}$*.*

We will only be looking at non-randomized stopping procedures, meaning that the τ_t can either be 0 or 1 (either we make a decision, or we don't). To decide which procedure is to be preferred, we will define a loss function that describes the loss of taking action $a \in \mathscr{A}$ given that we are in state $\theta \in \Theta$. For the loss of making a decision at time t we will assume the loss function $L(\theta, a_t)$ to take the following form:

$$L(\theta, a_t) = \begin{cases} 0 & \text{if } \theta = 0, a = 0 \\ 0 & \text{if } \theta = 1, a = 1 \\ f & \text{if } \theta = 0, a = 1 \\ g & \text{if } \theta = 1, a = 0 \end{cases} \tag{7.1}$$

Also we will assume the overall cost after n observations has the form:

$$L(\theta, a_t, n) = L(\theta, a_t) + \sum_{i=1}^{n} c_i \tag{7.2}$$

The reason for using this decomposition is that in a sequential decision procedure, there is an extra decision to be made compared to decision making with a fixed sample size; namely whether to stop sampling and make the decision or to wait and take another data point. This decomposition can be described as having a loss $L(\theta, a_t)$ associated with making the decision and a cost $\sum_{i=1}^{n} c_i$ of observing or sampling. In most applications taking more samples will result in costs, therefore usually $c_i > 0$. Taking bigger values for c_i will result in a quicker decision and therefore the size of the total sample used to make this decision will be smaller. We will assume that c_i is constant, such that:

$$L(\theta, a_t, n) = L(\theta, a_t) + cn \tag{7.3}$$

The decision procedure should be such that we minimize this loss.

7.2. Sequential probability ratio test

The sequential probability ratio test (SPRT), described by Wald [24] is a sequential procedure to make a decision in a simple sequential hypothesis test, as described in the introduction of this chapter. The test was designed assuming that the time series X_t is i.i.d. with density either $f_0(x)$ (for $\theta = 0$) or $f_1(x)$ (for $\theta = 1$). Then, the likelihood ratio, after making observations up till time t is:

$$\Lambda_t(X_{1:t}) = \prod_{i=1}^{t} \frac{f_1(x_i)}{f_0(x_i)} \tag{7.4}$$

Now let Λ_0 and Λ_1 be two constant thresholds, then the test will be as follows:

$$\begin{aligned} \text{Accept } H_1 &\text{ if } \Lambda_t \geq \Lambda_1 \\ \text{Accept } H_0 &\text{ if } \Lambda_t \leq \Lambda_0 \\ \text{Continue sampling } &\text{ if } \Lambda_1 \leq \Lambda_t \leq \Lambda_0 \end{aligned}$$

It can be shown, using proper threshold constant that depend on the loss function, that this procedure is optimal in the sense that it minimizes the expected sample size, in the class of all sequential tests. Also it turns out that the same procedure can be used when X_t is not i.i.d. but shows nice statistical properties, such as following an AR model or being a Markov process. Because hidden state Y_t is a Markov chain, this procedure can be used as an optimal decision procedure for triggering alarms. The difficulty is to find appropriate values for the thresholds. By taking a Bayesian perspective we can find a solution to this problem.

7.3. Bayesian sequential decision making

The basic idea behind Bayesian sequential analysis is quite simple. If we want to make inference on a parameter $\theta \in \Theta$ and start with a prior distribution $\pi(\theta)$ for this parameter, we can use Bayes theorem (3.25) to calculate the posterior density $\pi^1(\theta|x_1)$ after observing X_1. Doing so in a sequential way we can keep updating and at time n we have $\pi^n(\theta|x_{1:n})$. We will denote this as $\pi^n(\theta)$. At each time we can either make a decision for a certain θ, or gather more data to further support our decision.

To decide whether we should continue observing or make a decision, we will make use of the *Bayesian expected loss* $\rho(\pi, a)$.

Definition 5. ***Bayesian expected loss*** $\rho(\pi, a)$
For posterior π^*, loss function $L(\theta, a)$ and action a the Bayesian expected loss is defined as:

$$\begin{aligned} \rho(\pi^*, a) &= \mathbb{E}^{\pi^*}[L(\theta, a)] \\ &= \int_{\Theta} L(\theta, a)\pi^*(\theta|x)d\theta \end{aligned}$$

We want to take action a such that $\rho(\pi^*, a)$ is minimized.

Definition 6. ***Bayes action, Bayes risk***
The action a that minimizes $\rho(\pi^*, a)$ is called the Bayes action. We will denote the Bayesian expected loss of the Bayes action as $\rho(\pi^*)$ and call this the Bayes risk.

In the end we want to take the Bayes action, since the Bayes risk is the minimum expected loss. This will be the $\boldsymbol{\delta}$ in our sequential decision procedure. At time t we use as decision rule:

$$\delta_t(X_{1:t}) = \underset{a}{\arg\min}\, \rho(\pi^t(X_{1:t}, a)) \tag{7.5}$$

So, in the end we want a procedure using a stopping time where the actions taken result in the "true" Bayes risk $\rho(\pi)$. The question now is, how many data points should we take into account when making a decision? For this we will use an approximation of the true Bayes risk, where we only take a maximum of m data points into account. We will denote the Bayes risk of a sequential procedure using at most m data points, starting at time t as:

$$\rho_m(\pi^t, t)$$

To decide when to take this action or to keep sampling, we will use a stopping rule. If we have a maximum amount of data point that we want to use to make our decision, we can use the *Bayes m-truncated stopping rule.*

Principle 1. ***Bayes m-truncated stopping rule*** τ^m
Stop sampling and make a decision for the first n, $(n = 0, 1, ..., m)$ as soon as

$$\rho_0(\pi^n, n) = \rho_{m-n}(\pi^n, n)$$

In other words, we stop sampling and make a decision as soon as the Bayes risk of making an immediate decision is smaller than the Bayes risk of taking more observations, no later than at time m.

Now $\tau^m = \inf_t \big(t \in \{0, 1, ..., m\} : \rho_0(\pi^t) = \rho_{m-t}(\pi^t)\big)$ is a stopping time, since it only depends on information up to time t. It turns out we can calculate the Bayesian expected loss $\rho_j(\pi^n)$ at time j after making n observations in a recursive way:

$$\rho_j(\pi^n) = \min\big\{\rho_0(\pi^n), \mathbb{E}^*[\rho_{j-1}(\pi^n(\theta|x_{n+1}))] + c\big\} \tag{7.6}$$

Here the expectation is taken with respect to the marginal distribution of X_{n+1} at time n, $f(X_{n+1}|X_{1:n})$

Theory shows that under some assumptions, this truncated stopping rule converges to the true Bayes risk $\rho(\pi)$. This is the scribed in the following theorem.

Theorem 1. ***Limit of*** $\boldsymbol{\tau^m}$
Assuming:

1. *X_t is a sequentially taken sample*

2. *The loss function has form* (7.2)

3. *There exists a sequential procedure **d** with finite Bayes risk*

Then there exists an optimal Bayes procedure $\mathbf{d}^\pi$, where the procedure is to stop sampling at the first time n for which $\rho_0(\pi^n)$ is finite and

$$\rho_0(\pi^n) = \rho^\infty(\pi^n) \tag{7.7}$$

Here $\rho^\infty(\pi^n)$ can be calculated sequentially using:

$$\rho^\infty(\pi^n) = \min\big\{\rho_0(\pi^n), \mathbb{E}^*[\rho^\infty(\pi^n(\theta|x_{n+1}))]\big\} \tag{7.8}$$

It can be shown that, using this stopping procedure, performing a simple hypothesis test in the Bayesian setting results again in a Sequential Probability Ratio Test. For the proof I refer to [25].

7.3.1. Implementation using particle filter

We will again look at the simple hypothesis test $H_0 : \theta = 0$ versus $H_1 : \theta = 1$. In our model, the information important for making the decision to trigger an alarm is in the hidden state S_t, in particular the hidden spider state T_t. If $T_t = 4$ or $T_t = 5$, there should be an alarm triggered. The particle filter calculates the probability of T_t given the data $X_{1:t}$. So, in other words the posterior distribution of θ given the data $X_{1:t}$. This means that at posterior density at time t we have:

$$\pi^t(\theta = 1|x_{1:t}) = \mathbb{P}(T = 4|X_{1:t}) + \mathbb{P}(T = 5|X_{1:t}) \tag{7.9}$$

Assume we are starting the decision making procedure at time $t+1$. As prior we take the outcome of the particle filter $\pi^t(\theta|x_{1:t})$:

$$\pi(\theta) = \pi^t(\theta|x_{1:t}) \tag{7.10}$$

For the sake of simplicity, we will denote the posterior distribution $\pi^t(\theta)$ as:

$$\pi^t(\theta|x_{1:t}) = \begin{cases} \pi_1^t & \text{if } \theta = 1 \\ 1-\pi_1^t & \text{if } \theta = 0 \end{cases} \tag{7.11}$$

Now we can calculate the Bayes risk $\rho_0(\pi^t, a)$ of making a decision at time t:

$$\rho_0(\pi^t, a) = \begin{cases} g\pi_1^t & \text{if } a = 0 \\ f(1-\pi_1^t) & \text{if } a = 1 \end{cases} \tag{7.12}$$

Clearly, the Bayes action, minimizing the expected Bayes loss, is to accept H_0 if the expected loss is smaller when accepting $\theta = 0$ than the expected loss when accepting $H_1 : \theta = 1$. The corresponding Bayes risk is:

$$\rho_0(\pi^t) = \min\{g\pi_1^t, f(1-\pi_1^t)\} \tag{7.13}$$

Note that holds:

$$\begin{aligned} g\pi_1^t \le f(1-\pi_1^t) &\Leftrightarrow g\pi_1^t + f\pi_1^t \le f \\ &\Leftrightarrow \pi_1^t \le \frac{f}{f+g} \end{aligned} \tag{7.14}$$

Therefore the value of $\rho_0(\pi^t)$ depends on the value of the posterior density π^t in the following way:

$$\rho_0(\pi^t) = \begin{cases} g\pi_1^t & \text{if } \pi_1^t \le \frac{f}{f+g} \\ f(1-\pi_1^t) & \text{if } \pi_1^t > \frac{f}{f+g} \end{cases} \tag{7.15}$$

Ideally, we would like to use theorem 1 as decision procedure. In practice however, it is not possible to calculate $\rho^\infty(\pi^n)$, because we would have to take into account all sequential stopping procedures that are using at least one and up to ∞ data points. Therefore we wil assume the following:

$$\rho^\infty(\pi^n) \approx \rho_1(\pi^n, t+1) \tag{7.16}$$

In other words, we only take the expectation of the next data point into account for our decision. The reason for this is that the spider state T_t can change on a daily basis. So, θ is in fact not constant, even though the sequential decision procedures discussed here assumed it is. If a change has occurred in θ, it doesn't make sense to use data from before the change to make a decision about θ. In practice this results into taking an immediate decision most of the times; however it is possible that the procedure waits a few (usually not more than one) more days before making the decision. Ideally we would actually want the procedure to take a decision every day. If we want to enforce this, we could use the truncated stopping rule provided by theorem (1) and set $m = 1$. In some cases however it could be beneficial to wait a little before making a decision.

Now we want to calculate the Bayes risk of taking another observation. To calculate the expected Bayes risk at time $t+1$, we first calculate the expected distribution of the spider state T_{t+1}, using the current distribution at time t and the transition probabilities of the hidden Markov model. The expected distribution of spider state T_{t+1} is:

$$\begin{pmatrix} \hat{p}_1^{t+1} & \hat{p}_2^{t+1} & \hat{p}_3^{t+1} & \hat{p}_4^{t+1} & \hat{p}_5^{t+1} \end{pmatrix} = \begin{pmatrix} p_1^t & p_2^t & p_3^t & p_4^t & p_5^t \end{pmatrix} \mathscr{P} \tag{7.17}$$

Here $p_i^t = \mathbb{P}(T_t = i|X_{1:t})$, $\hat{p}_i^{t+1}$ the expected probability of p_i^{t+1} and $\mathscr{P}$ is the transition probability matrix of the hidden Markov model:

$$\mathscr{P} = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{pmatrix} \tag{7.18}$$

The expected posterior distribution $\hat{\pi}_1^{t+1}$ at time $t+1$ is equal to the expected probability of the spider being in state 4 or 5:

$$\hat{\pi}_1^{t+1} = \hat{p}_4^{t+1} + \hat{p}_5^{t+1} \tag{7.19}$$

Using this we can calculate the expected Bayes risk at time $t+1$:

$$\mathbb{E}^*[\rho(\pi^{t+1})] = \min\left\{g\hat{\pi}_1^{t+1}, f(1-\hat{\pi}_1^{t+1})\right\} \tag{7.20}$$

Or:

$$\mathbb{E}^*[\rho(\pi^{t+1})] = \begin{cases} g\hat{\pi}_1^{t+1} & \text{if } \hat{\pi}_1^{t+1} \leq \frac{f}{f+g} \\ f(1-\hat{\pi}_1^{t+1}) & \text{if } \hat{\pi}_1^{t+1} > \frac{f}{f+g} \end{cases} \tag{7.21}$$

Now we can use (7.6) to calculate $\rho_1(\pi)$ and we can decide using the stopping rule described in theorem 1 if we should continue sampling or make a decision. We should stop sampling if:

$$\rho_1(\pi) = \rho_0(\pi)$$

If $\rho_1(\pi) < \rho_0(\pi)$, we should keep sampling. The decision making procedure is summarized in algorithm 6 and has a complexity of $\mathcal{O}(1)$.

```
**Result:** Either a decision to trigger an alarm ($\theta = 1$) by accepting $H_1$ or no to trigger an alarm $\theta = 0$) by
        accepting $H_0$
**Initialize:**
$p_1^{t-1} = \mathbb{P}(T_{t-1} = 1|X_{1:t-1})$
$p_2^{t-1} = \mathbb{P}(T_{t-1} = 2|X_{1:t-1})$
$p_3^{t-1} = \mathbb{P}(T_{t-1} = 3|X_{1:t-1})$
$p_4^{t-1} = \mathbb{P}(T_{t-1} = 4|X_{1:t-1})$
$p_5^{t-1} = \mathbb{P}(T_{t-1} = 5|X_{1:t-1})$
$\boldsymbol{p}^{t-1} = \begin{pmatrix} p_1^{t-1} & p_2^{t-1} & p_3^{t-1} & p_4^{t-1} & p_5^{t-1} \end{pmatrix}$
SequentialDecision($t, \boldsymbol{p}^{t-1}, \mathscr{P}, f, g, c$):
$\pi_1 = p_4^{t-1} + p_5^{t-1}$
$\begin{pmatrix} \hat{p}_1^t & \hat{p}_2^t & \hat{p}_3^t & \hat{p}_4^t & \hat{p}_5^t \end{pmatrix} = \boldsymbol{p}^{t-1}\mathscr{P}$
$\hat{\pi}_1^t = \hat{p}_4^t + \hat{p}_5^t$
$\rho_0(\theta) = \min\{g\pi_1, f(1-\pi_1)\}$;
$\rho_1(\theta) = \min\{\rho_0(\theta), \min\{g\hat{\pi}_1, f(1-\hat{\pi}_1)\} + c\}$
**if** $\rho_0 = \rho_1$ **then**
    **Make decision**
    **if** $\pi^{t-1} \le \frac{f}{f+g}$ **then**
        Accept $H_0$
    **else**
        Accept $H_1$ and trigger alarm
    **end**
**else**
    **Take next sample**
    $p_1^t = \mathbb{P}(T_t = 1|X_{1:t})$
    $p_2^t = \mathbb{P}(T_t = 2|X_{1:t})$
    $p_3^t = \mathbb{P}(T_t = 3|X_{1:t})$
    $p_4^t = \mathbb{P}(T_t = 4|X_{1:t})$
    $p_5^t = \mathbb{P}(T_t = 5|X_{1:t})$
    $\boldsymbol{p}^t = \begin{pmatrix} p_1^t & p_2^t & p_3^t & p_4^t & p_5^t \end{pmatrix}$
    $\pi_1 = p_4^t + p_5^t$
    SequentialDecision($t+1, \boldsymbol{p}^t, \mathscr{P}$)
**end**
```

Algorithm 6: Sequential decision making algorithm.

7.4. Resetting the system

After an alarm has been triggered, the user has to decide what to do with this alarm. If he thinks the spidering is broken, he can try to solve the issue to get the amount of vacancies found back to the normal level. Because the normal level of amount of vacancies found by the spider, A_t, remains the same, the alarm will likely be triggered the next day as well. On the other hand, in case he thinks the spidering is actually working as it should be, he can decide to reset the alarm.

By resetting the system, a new set of initial particles is generated and the sequential algorithm will be started over. The new set of particles will be generated the same way the initial set of particles was generated, by setting $A_t = X_t$, $T_t = 1$ and also resetting the numbers l_t^A and n_t to their initial values. The value for σ_{start} will be set to a default value of $\frac{1}{50}A_t$, since we cannot assume the previous value of σ_t is appropriate in the new situation. The probability of hidden state $T_t = 1$ will be reset to 1. The procedure is described in algorithm 7 and has a complexity of $\mathcal{O}(N)$.

```
Result: Reset the the particles after an alarm
ResetParticles($X_t$,$(\hat{s}_t^{(i)})_{1\leq i\leq N}$,$\sigma_{start}$)
for $1 \leq i \leq N$ do
    $t_t^{(i)} = 1$
    $a_t^{(i)} = X_t$
    $\sigma_t^{(i)} = \sigma_{start}$
    $n_t^{(i)} = -1$
    $l_t^{A,(i)} = X_t$
    $\tilde{s}_t^{(i)} = \begin{pmatrix} t_t^{(i)} \\ a_t^{(i)} \\ \sigma_t^{(i)} \\ n_t^{(i)} \\ l_t^{A,(i)} \end{pmatrix}$
end
return $(\tilde{s}_t^{(i)})_{1\leq i\leq N}$
```

Algorithm 7: Reset particles algorithm.

In practice he will do this after some spider issue has been resolved or some change has occurred on the website (e.g. it has merged with another website).

8 System test

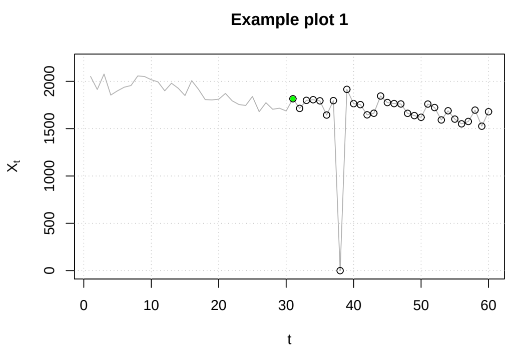
The system described in this thesis should be able to trigger an alarm when a spider is not showing normal behaviour. In order to see how well the system performs, a test has been done. We will use the outcome of the test to set the parameters f, g, c in the decision making algorithm 6, such that the precision (3.42) and recall (3.43) of the method are nearly optimal and the system triggers the right amount of alarms. First we will elaborate on the way the test has been done and the data that was used in section 8.1. After this the results are used to pick values for f, g and c and calculate the precision and recall of the system in section 8.2.

8.1. Test description

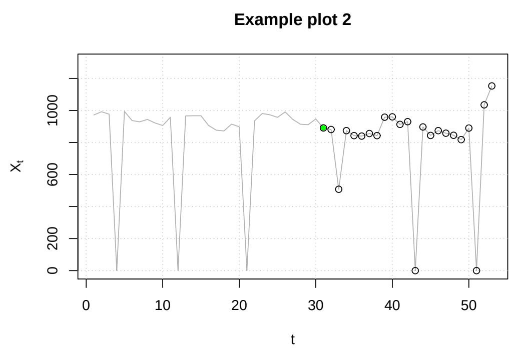
The test has been performed as follows. First the top 100 websites that have the largest number of vacancies is taken and of these the last 75 websites will be used in final the test; these are the websites the system will be used for in practise as well. The data available from these websites is dated from 21-05-2015 to 22-05-2017. In this time range, four non-overlapping intervals of 60 days have been chosen and used for each website. Not all of this data was available for every website. Besides, only the intervals where the median of the spider accepted data was at least 800 and where the spider accepted more than zero vacancies at the starting time of the algorithm have been used. This resulted in 200 data sets of length 60 that can be used in the final test.

For each data set, the algorithm 5 has been run, using $N = 1000$ particles, $N_{eff} = N$, thresholds $th_\sigma = s_\sigma$ and starting at time $t = 31$, taking the first 30 data points to calculate the initial values of the particle filter, as described in section 5.4. Now we will keep running the algorithm until it triggers its first alarm and the time of this alarm is saved. After an alarm has been triggered, the algorithm will stop running for this data set and go on to the next one. If no alarm has been triggered when reaching the end of the interval, this will also be saved. In this case the algorithm will stop at the end of the interval, which is at $t = 60$.

Plots showing the data of the intervals up to the point of the alarm (or the whole interval in case no alarm has been triggered) have been shown to two annotators, the members of the Jobfeed Data Quality team. Two examples of such plots can be found in figure 8.1.



(a) Plot without alarm.



(b) Plot with alarm.

Figure 8.1: Examples of test plots.

Here the green dot represents the starting point of the algorithm. The spider is assumed to be at a normal level here. Note that this might not be the case; the spider could be broken at the starting point. The annotators were asked to judge the outcome of the alarm system given that the green dot was indeed the expected amount of vacancies to be found by the spider.

The algorithm has run for the black dotted data. The first 30 data points are only shown as reference for the annotators, to observe what kind of behaviour is normal for this spider. For each plot, the following questions had to be answered, about the points where the algorithm has run:

1. Should an alarm be triggered at the last data point plotted?
2. Should an alarm be triggered before the last data point? If yes, at what time?
3. At which other times should alarms be triggered?
4. Is the spider stable enough to monitor?

It turned out that the answer to the third question was no at all times, meaning the system never missed more than one alarm. If the spider was too unstable to monitor, the plot has not been used to calculate results.

So, for now only the first two questions are of interest to calculate the precision and recall of the algorithm. Since at most two alarms in each interval where the algorithm has run were required (one at the last point and possibly one before that the algorithm did not trigger), we can make a list of where alarms where required. Using this and the outcome of the algorithm we can calculate the amount of alarms that was triggered correctly (true positives), the amount of alarms that was triggered even though no alarm was required (false positives) and how many times an alarm should have been triggered, but the algorithm failed to do so (false negatives). Using these we can calculate the precision (3.42) and recall (3.43).

The test of which the plots have been shown to the annotators has been performed using the following parameters for the decision making algorithm:

$$\begin{aligned} f &= 1 \\ g &= 1 \\ c &= 0.007 \end{aligned} \tag{8.1}$$

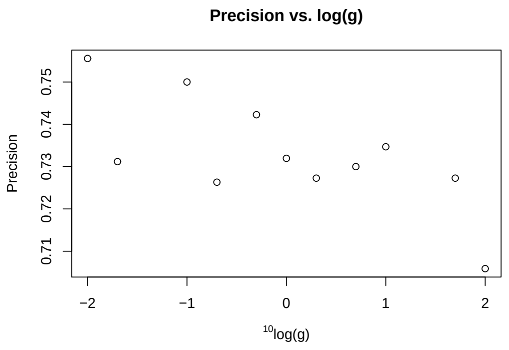
We run the algorithm for the same intervals but with different values of these parameters to see which results in the best values for the precision and recall. As we can see in algorithm 6, the actual value of f does not matter, only the quantity $\frac{f}{f+g}$ is important. Therefore we will keep $f = 1$ fixed and alter g. We will keep $c = 0.007$ fixed. By looking at the values of the precision and recall we will pick an optimal value for g. This value of g will be used when we run the system again, this time altering the value of c, to find what value would be optimal for c.

The old alarm system that is currently in place to trigger alarms has been run on the same data. A description of this algorithm can be found in section 2.4. Using the annotations, we can calculate the precision and recall of the old system and use this as a base line.

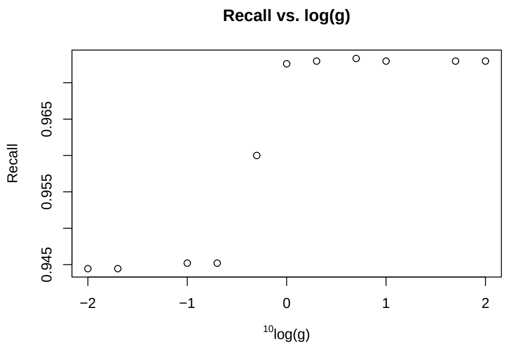
8.2. Results parameter choice

The final test consisted of 200 plot similar to those in figure 8.1. Two annotator looked at the these plots and answered the questions mentioned in the previous section separately. In 25% of the answers, their judgement was different. Therefore the plots where they disagreed were revisited, this time together to come to a final conclusion. Often the disagreement was due to a mistake by one of the two annotators (approximately 40%), or because the spider was too unstable to be used (approximately 12%). In the other cases it was hard to judge where and if an alarm should be triggered, but after debating final answers have been agreed upon. The plots that were too unstable are left out of the results (two of them were judged as too unstable).

The result of the test altering the value of g is shown in figure 8.2. Both the precision and the recall have been plotted versus $^{10}\log(g)$ on the x-axis that was used in the algorithm.



(a) Precision.



(b) Recall.

Figure 8.2: Precision and recall for different values of g.

Also, the average number of alarms per day that we would get if we would implement the algorithm and use it for these 75 websites on a daily basis has been calculated and plotted in figure 8.3.

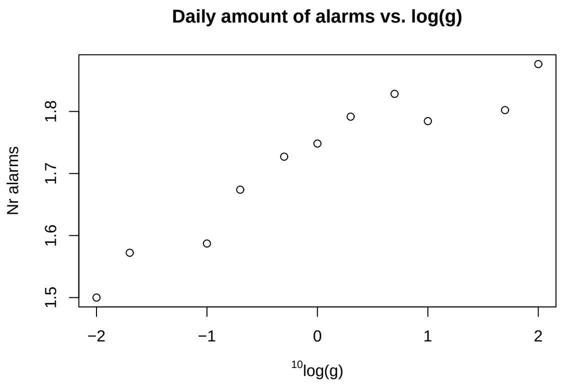
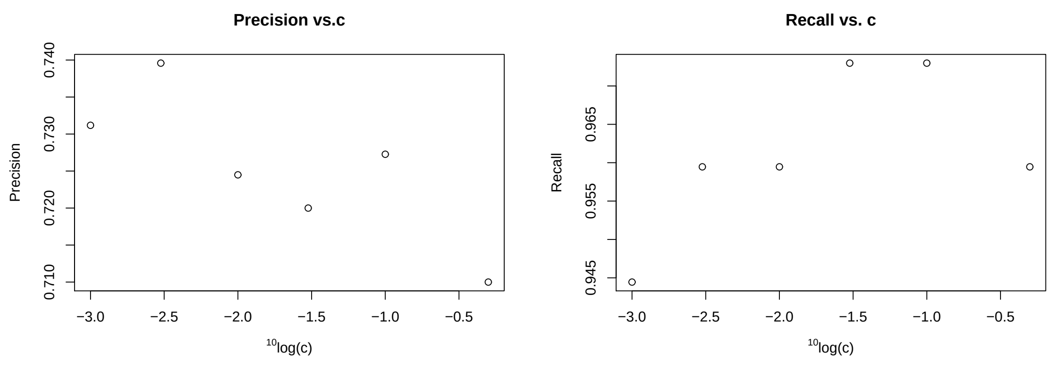


Figure 8.3: Average number of alarms per day for different values of g.

We can see that for $^{10}\log(g) \geq 0$ the recall is approximately equal to 0.97 and stays approximately the same after this. The precision is fluctuating around 0.73 for most values of g. Since we don't want the algorithm to fail triggering an alarm when it should, we will use the optimal value of the recall, so some $g \geq 1$. Because the precision is approximately the same for all these values of g, we will use the g that minimizes the amount of alarms we get per day, which is approximately 1.75 for $g = 1$.

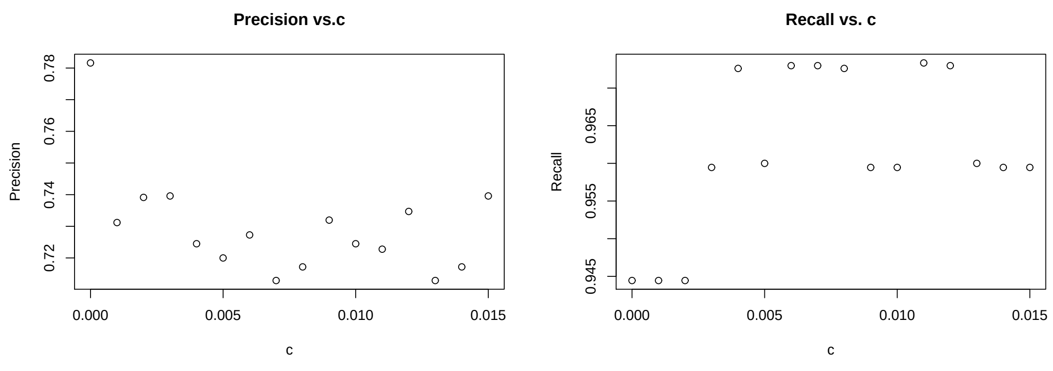
Taking $g = 1$ fixed and running the algorithm for different value of c gives precision and recall values as plotted in figure 8.4.



(a) Precision. (b) Recall.

Figure 8.4: Precision and recall for different values of c.

Note that the precision is fluctuating around 0.725 and doesn't differ a lot, but is maximum for ${}^{10}\log(c) = -2.5$. The recall is smallest for ${}^{10}\log(c) = -3.0$ and for ${}^{10}\log(c) \geq -2.5$ the recall is either approximately 0.960 or around 0.969. Remarkable is that for $c = 0.5$, the recall is slightly lower than for $c = 0.1$. This suggests that the difference in recall for ${}^{10}\log(c) \geq -3.0$ is not caused by the difference in c, but by other limiting factors such as the limited number of particles N or a different outcome of the change point detection that is determining the initial values of the particle filter. Since the results are so similar for $c \geq 0.01$, it seems that the decision procedure here is to take an immediate decision at all times, without waiting for another observation. Therefore it is interesting to have a closer look at the behaviour for smaller c. We will have focus on values of c between 0 and 0.0015. The precision and recall are plotted in figure 8.5.



(a) Precision. (b) Recall.

Figure 8.5: Precision and recall for different values of c.

We see that for $c \geq 0.003$ we observe similar behaviour for both the precision and recall. The average daily amount of alarms for these 75 websites can be found in figure 8.6.

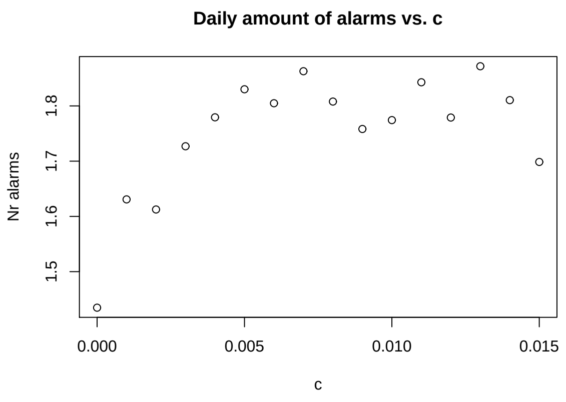


Figure 8.6: Average number of alarms per day for different values of c.

Note that the number of alarms is increasing for $c \leq 0.003$ and after this is stays approximately the same. It seems the value of c doesn't make a big difference in the outcome of the system. We will use $c = 0.003$, because the precision and recall values for this c are similar to the others, but the expected number of daily alarms is slightly lower.

The old alarm system as described in section 2.4 has been used on the annotated data and the precision, recall and the expected number of daily alarms for 75 websites have been calculated. The results of the old and the new system using $g = 1$ and $c = 0.003$ can be found in table 8.1.

	Precision	Recall	Expected number of alarms per day
Old alarm system	0.028	0.182	8.83
New alarm system	0.740	0.959	1.727

Table 8.1: Results of testing the old and new alarm system.

8.2.1. Discussion system test

As noted before, it seems that the sequential decision algorithm 6 almost always decides to make a decision right away, instead of waiting for another observation. This suggests that we could leave out the (somewhat complicated) sequential decision theory and just decide to make a decision right away, every day; this would obsolete the parameter c. In the long run however, using the sequential decision theory algorithm could lead to better results. The test that has been performed here only uses the initial transition probability matrix (5.40) containing the transition probabilities of the hidden spider state T_t, because the particle filter didn't run long enough to update these transition probabilities. How this transition probability matrix is update is described in section 5.3.2. By updating these transition probabilities, prior information about the behaviour of the spider is incorporated in the decision making process.

An example where using the sequential decision algorithm could be useful is the case where the particle filter claims it is most likely for the spider to be in state 5, meaning it is zero for multiple days. If in the past the spider has been zero for two days in a row quite often but has recovered by itself the third day, this will be incorporated the transition probability matrix $\mathscr{P}$ by a bigger value of p_{51}. Therefore we expect that spider is more likely to have $T_{t+1} = 1$. This can cause the decision to be postponed and if the spider is indeed back to normal, no alarm will be triggered.

9 Conclusion

In this thesis an alarm system has been developed that triggers alarm in case it seems there is a problem in the spidering of the Jobfeed data. The precision of this new system is 0.74, which is much higher than the precision of the old system, 0.03. Also the recall of this new system, 0.96, is a lot higher than the recall of the old system, 0.18. If we would implement this new alarm system and use it to monitor 75 websites, on average there would be 1.7 alarms triggered per day, opposed to the 8.8 alarms triggered by the old system.

Also change point methods to detect changes in a random walk process with drift including outliers have been investigated. The method that was most successful in detecting a single change in level and drift was to first filter the data using a Hampel filter and minimizing an absolute cost function, where the level of the spider was approximated by using a Gaussian linear regression line. The precision of this method for finding a single change point, if we allow for an error of maximum two, is 0.76. However when we use this method in the Wild Binary Segmentation method to detect multiple change points, it is less effective; a precision of only 0.36 is obtained. The recall of this multiple change point detection method is 0.89.

9.1. Future research

Although the alarm system developed in the course of this thesis is performing well compared to the old alarm system that was in place, there are several things that could be improved upon.

9.1.1. Improvements change point detection

Detecting change points in a random walk was harder than anticipated and could use some more attention, especially detecting multiple change points. The method using the WBS algorithm 4 to find multiple change points in a random walk has a low precision of only 0.36.

One of the reasons for this is that too many change points are found, because the random intervals taken by the WBS interval too often contain more than one change point. This could be solved in two ways; either by adjusting these random intervals, or by changing the threshold parameter ζ_N that is used to judge whether the change point found is significant enough to keep. If the random intervals are forced to be smaller than a pre-set size, the chance to find only a single change point in the segment is increased; for this knowledge of the typical distance between the change points is required. The other option involves ζ_N. Right now this threshold only depends on the size of the whole time series, which is N. However, we could also adjust ζ such that it depends on the size N_i of the interval chosen by the WBS algorithm. By increasing ζ_{N_i} for increasing interval length N_i, the algorithm is less likely to accept a change point if the interval is large.

Another reason for the low precision in the multiple change point detection method is that outliers still cause the algorithm to detect too many change points. As estimate of the level, a Gaussian regression line (3.32) has been used, instead of more robust options (3.38) or (3.41). This has been done because calculating the more robust linear regression lines was computationally too expensive to implement. If however a way could be found to calculate these regression lines quicker, this could improve the result of the change point detection method.

9.1.2. Improvements particle filter

Also the particle filter used could use some improvements. The filter is using the outcome from the past and the user input to improve the results in two ways: by updating the transition probability matrix $\mathscr{P}(p_{12}, p_{13}, p_{14}, p_{35})$ and by estimating σ. Both of these can be improved upon.

In our implementation we assumed that each transition probability that we are estimating has a minimum value of 0.01. The reason for this is that if the past suggests that one of these probabilities is equal to zero, the algorithm doesn't work any more because some states would not be able to be reached. This value is arbitrarily chosen. To improve this, a Bayesian approach could be taken by starting off with a prior over the values of p_{12}, p_{13}, p_{14} and p_{35} and updating this prior as more observations come in. Here we can take into account that the outcome of the particle filter is the probability of the spider being in a certain state; as it is implemented now, we take the state with the highest probability to be true.

To find a starting value for σ, σ_{start}, first change point detection is used to find the last stable segment of the spider and here the robust σ_{MAD} (3.19) is taken as estimation. If the last segment is too short, a custom value is used, that is proportional to the maximum of the last 30 days. This is quite arbitrary and more sophisticated methods could be used. The choice of σ_{start} could for example be improved by looking at another stable segment that is long enough to estimate properly.

9.1.3. Improvements sequential decision making

As we have seen in section 8.2.1, the perks of using the sequential decision making algorithm over making a decision every day, can be seen as the transition probability matrix describing the transition probabilities of hidden state T_t is updated using the outcome of the particle filter. This means that setting a value for c without running the algorithm for a longer period of time isn't very meaningful. If this alarm system is implemented and running live, the outcome could be used to better tune not only c, but other parameters as well.

9.1.4. Improvements using other data

For the alarm system, only the spider accepted data has been used. The system could be improved by including other statistics of the spidering in the system. The best usable are the total number of active vacancies and the the number of imported vacancies.

The number of active vacancies should be close to the number of vacancies accepted by the spider. If these two differ a lot, this could be a reason to trigger an alarm. This is similar to the current approach, only the difference is that now we model the number of vacancies that the spider should accept as a random walk A_t, instead of comparing it to the number of active vacancies in the database.

If we use the spider imported, it could give a different insight. The number of vacancies the spider imports should be significantly lower than the number of vacancies the spider finds, unless the spider is new. We do however expect the number of new vacancies on a website to be somewhat stable. Therefore we could use the same model as we used for the spider accepted data and apply the same alarm system as well. Here most parameters will be different, because the number of vacancies imported by the spider is much lower and zero's also occur more often. Besides this, the way websites upload their vacancies is more important here than in the case of the spider accepted data.

Bibliography

[1] IDC Worldwide, "Worldwide Semiannual Big Data and Analytics Spending Guide", `https://www.idc.com/getdoc.jsp?containerId=IDC_P33195`, 2017

[2] Shumway, Robert H., and David S. Stoffer. "Time series analysis and its applications." Studies In Informatics And Control 9.4 (2000): 375-376.

[3] Pearson, Ronald K., et al. "Generalized hampel filters." EURASIP Journal on Advances in Signal Processing 2016.1 (2016): 87.

[4] Young, G. Alastair, and Richard L. Smith. Essentials of statistical inference. Vol. 16. Cambridge University Press, 2005.

[5] Montgomery, Douglas C., Elizabeth A. Peck, and G. Geoffrey Vining. Introduction to linear regression analysis. Vol. 821. John Wiley & Sons, 2012.

[6] Powers, David Martin. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation." (2011).

[7] Shewhart, Walter Andrew, and William Edwards Deming. Statistical method from the viewpoint of quality control. Courier Corporation, 1939.

[8] Page, Ewan S. "Continuous inspection schemes." Biometrika 41.1/2 (1954): 100-115.

[9] Kalman, Rudolph Emil, "A new approach to linear filtering and prediction problems.", Journal of basic Engineering 82.1 (1960): 35-45.

[10] Fearnhead, Paul, and Guillem Rigaill. "Changepoint Detection in the Presence of Outliers." Journal of the American Statistical Association just-accepted (2017).

[11] Fearnhead, Paul, and Zhen Liu. "On-line inference for multiple changepoint problems." Journal of the Royal Statistical Society: Series B (Statistical Methodology) 69.4 (2007): 589-605.

[12] Haynes, Kaylea, Paul Fearnhead, and Idris A. Eckley. "A computationally efficient nonparametric approach for changepoint detection." Statistics and Computing 27.5 (2017): 1293-1305.

[13] Badagián, Ana Laura, Regina Kaiser, and Daniel Peña. "Time series segmentation procedures to detect, locate and estimate change-points." Empirical Economic and Financial Research. Springer, Cham, 2015. 45-59.

[14] Killick, Rebecca, Paul Fearnhead, and Idris A. Eckley. "Optimal detection of changepoints with a linear computational cost." Journal of the American Statistical Association 107.500 (2012): 1590-1598.

[15] Wu, Zhengxiao. "Quasi-hidden Markov model and its applications in change-point problems." Journal of Statistical Computation and Simulation 86.12 (2016): 2273-2290.

[16] Yao, Yi-Ching, and Siu-Tong Au. "Least-squares estimation of a step function." Sankhyā: The Indian Journal of Statistics, Series A (1989): 370-381.

[17] Jackson, Brad, et al. "An algorithm for optimal partitioning of data on an interval." IEEE Signal Processing Letters 12.2 (2005): 105-108.

[18] Scott, Andrew Jhon, and M. Knott. "A cluster analysis method for grouping means in the analysis of variance." Biometrics (1974): 507-512.

[19] Fryzlewicz, Piotr. "Wild binary segmentation for multiple change-point detection." The Annals of Statistics 42.6 (2014): 2243-2281.

[20] Shiryaev, Albert N. "On optimum methods in quickest detection problems." Theory of Probability & Its Applications 8.1 (1963): 22-46.

[21] Worsley, K. J. "On the likelihood ratio test for a shift in location of normal populations." Journal of the American Statistical Association 74.366a (1979): 365-367.

[22] Merhav, Neri, et al. "On sequential strategies for loss functions with memory." IEEE Transactions on Information Theory 48.7 (2002): 1947-1958.

[23] Tartakovsky, Alexander, Igor Nikiforov, and Michele Basseville. Sequential analysis: Hypothesis testing and changepoint detection. CRC Press, 2014.

[24] Wald, Abraham, "Sequential Analysis", Sequential analysis, Courier Corporation, 1973.

[25] Berger, James O., "Statistical decision theory and Bayesian analysis", Springer Science & Business Media, 2013.

[26] Cappé, Olivier, Eric Moulines, and Tobias Rydén. "Inference in hidden markov models." Proceedings of EUSFLAT Conference. 2009.

[27] Ghahramani, Zoubin. "An introduction to hidden Markov models and Bayesian networks." International journal of pattern recognition and artificial intelligence 15.01 (2001): 9-42.

[28] Cappé, Olivier, Simon J. Godsill, and Eric Moulines. "An overview of existing methods and recent advances in sequential Monte Carlo." Proceedings of the IEEE 95.5 (2007): 899-924.

[29] Doucet, Arnaud, Nando De Freitas, and Neil Gordon. "An introduction to sequential Monte Carlo methods." Sequential Monte Carlo methods in practice. Springer, New York, NY, 2001. 3-14.

[30] Doucet, Arnaud, and Adam M. Johansen. "A tutorial on particle filtering and smoothing: Fifteen years later." Handbook of nonlinear filtering 12.656-704 (2009): 3.

[31] Arulampalam, M. Sanjeev, et al. "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking." IEEE Transactions on signal processing 50.2 (2002): 174-188.

[32] Arellano-Valle, Reinaldo B., et al. "Student-t censored regression model: properties and inference." Statistical Methods & Applications 21.4 (2012): 453-473.

[33] Rousseeuw, Peter J., and Christophe Croux. "Alternatives to the median absolute deviation." Journal of the American Statistical association 88.424 (1993): 1273-1283.

A

Results change point detection for random walks

A.1. Using X_t

Cost function	Estimator	Type of filter	Exact efficiency	Efficiency $\|\tau - \tau_c\| \leq 2$
Square	Mean	No filter	0.044	0.104
Square	Median	No filter	0.058	0.146
Square	Linear regression	No filter	0.012	0.038
Absolute	Mean	No filter	0.184	0.324
Absolute	Median	No filter	0.502	0.758
Absolute	Linear regression	No filter	0.09	0.18
Biweight	Mean	No filter	0.072	0.132
Biweight	Median	No filter	0.532	0.75
Biweight	Linear regression	No filter	0.028	0.072
Square	Mean	Median filter	0.22	0.746
Square	Median	Median filter	0.22	0.736
Square	Linear regression	Median filter	0.216	0.738
Absolute	Mean	Median filter	0.216	0.746
Absolute	Median	Median filter	0.22	0.754
Absolute	Linear regression	Median filter	0.218	0.744
Biweight	Mean	Median filter	0.192	0.668
Biweight	Median	Median filter	0.184	0.586
Biweight	Linear regression	Median filter	0.2	0.704
Square	Mean	Hampel filter	0.43	0.704
Square	Median	Hampel filter	0.426	0.702
Square	Linear regression	Hampel filter	0.41	0.664
Absolute	Mean	Hampel filter	0.51	0.788
Absolute	Median	Hampel filter	0.534	0.816
Absolute	Linear regression	Hampel filter	0.44	0.736
Biweight	Mean	Hampel filter	0.38	0.64
Biweight	Median	Hampel filter	0.462	0.714
Biweight	Linear regression	Hampel filter	0.33	0.588

Table A.1: Results of testing change point detection methods on a random walk with a single change point in level, without drift using X_t.

Cost function	Estimator	Type of filter	Exact efficiency	Efficiency $\lvert\tau - \tau_c\rvert \leq 2$
Square	Mean	No filter	0	0.016
Square	Median	No filter	0.002	0.014
Square	Linear regression	No filter	0.004	0.012
Absolute	Mean	No filter	0.002	0.004
Absolute	Median	No filter	0	0
Absolute	Linear regression	No filter	0.006	0.026
Biweight	Mean	No filter	0	0.002
Biweight	Median	No filter	0	0.002
Biweight	Linear regression	No filter	0.006	0.036
Square	Mean	Median filter	0	0.002
Square	Median	Median filter	0	0.002
Square	Linear regression	Median filter	0.006	0.068
Absolute	Mean	Median filter	0	0
Absolute	Median	Median filter	0	0
Absolute	Linear regression	Median filter	0.014	0.082
Biweight	Mean	Median filter	0	0.002
Biweight	Median	Median filter	0	0
Biweight	Linear regression	Median filter	0.014	0.07
Square	Mean	Hampel filter	0	0.004
Square	Median	Hampel filter	0.002	0.004
Square	Linear regression	Hampel filter	0.014	0.064
Absolute	Mean	Hampel filter	0	0
Absolute	Median	Hampel filter	0	0
Absolute	Linear regression	Hampel filter	0.01	0.06
Biweight	Mean	Hampel filter	0	0
Biweight	Median	Hampel filter	0	0.002
Biweight	Linear regression	Hampel filter	0.012	0.07

Table A.2: Results of testing change point detection methods on a random walk with a single change point in drift, using X_t.

Cost function	Estimator	Type of filter	Exact efficiency	Efficiency $\lvert\tau - \tau_c\rvert \leq 2$
Square	Mean	No filter	0.02	0.042
Square	Median	No filter	0.016	0.064
Square	Linear regression	No filter	0.002	0.014
Absolute	Mean	No filter	0.032	0.078
Absolute	Median	No filter	0.032	0.126
Absolute	Linear regression	No filter	0.046	0.076
Biweight	Mean	No filter	0.01	0.026
Biweight	Median	No filter	0.024	0.076
Biweight	Linear regression	No filter	0.02	0.044
Square	Mean	Median filter	0.046	0.122
Square	Median	Median filter	0.046	0.124
Square	Linear regression	Median filter	0.258	0.424
Absolute	Mean	Median filter	0.134	0.262
Absolute	Median	Median filter	0.086	0.2
Absolute	Linear regression	Median filter	0.36	0.604
Biweight	Mean	Median filter	0.036	0.09
Biweight	Median	Median filter	0.008	0.032
Biweight	Linear regression	Median filter	0.236	0.456
Square	Mean	Hampel filter	0.046	0.112
Square	Median	Hampel filter	0.048	0.128
Square	Linear regression	Hampel filter	0.272	0.442
Absolute	Mean	Hampel filter	0.148	0.298
Absolute	Median	Hampel filter	0.1	0.218
Absolute	Linear regression	Hampel filter	0.364	0.604
Biweight	Mean	Hampel filter	0.03	0.066
Biweight	Median	Hampel filter	0.014	0.026
Biweight	Linear regression	Hampel filter	0.26	0.47

Table A.3: Results of testing change point detection methods on a random walk with a single change point in level, with a constant drift using X_t.

Cost function	Estimator	Type of filter	Exact efficiency	Efficiency $\lvert\tau - \tau_c\rvert \leq 2$
Square	Mean	No filter	0.018	0.042
Square	Median	No filter	0.04	0.096
Square	Linear regression	No filter	0.022	0.046
Absolute	Mean	No filter	0.036	0.078
Absolute	Median	No filter	0.05	0.106
Absolute	Linear regression	No filter	0.084	0.146
Biweight	Mean	No filter	0.018	0.03
Biweight	Median	No filter	0.026	0.042
Biweight	Linear regression	No filter	0.028	0.066
Square	Mean	Median filter	0.028	0.112
Square	Median	Median filter	0.036	0.118
Square	Linear regression	Median filter	0.21	0.742
Absolute	Mean	Median filter	0.028	0.106
Absolute	Median	Median filter	0.026	0.1
Absolute	Linear regression	Median filter	0.212	0.764
Biweight	Mean	Median filter	0.016	0.072
Biweight	Median	Median filter	0.008	0.036
Biweight	Linear regression	Median filter	0.2	0.678
Square	Mean	Hampel filter	0.064	0.144
Square	Median	Hampel filter	0.062	0.138
Square	Linear regression	Hampel filter	0.464	0.71
Absolute	Mean	Hampel filter	0.064	0.108
Absolute	Median	Hampel filter	0.07	0.11
Absolute	Linear regression	Hampel filter	0.492	0.76
Biweight	Mean	Hampel filter	0.02	0.038
Biweight	Median	Hampel filter	0.024	0.056
Biweight	Linear regression	Hampel filter	0.316	0.578

Table A.4: Results of testing change point detection methods on a random walk with a single change point in both level and drift, using X_t.

A.2. Using ∇X_t

Cost function	Estimator	Type of filter	Exact efficiency	Efficiency $\lvert\tau - \tau_c\rvert \leq 2$
Square	Mean	No filter	0.002	0.014
Square	Median	No filter	0	0
Square	l_G	No filter	0.002	0.014
Absolute	Mean	No filter	0	0.002
Absolute	Median	No filter	0	0
Absolute	l_G	No filter	0	0
Biweight	Mean	No filter	0	0
Biweight	Median	No filter	0	0
Biweight	l_G	No filter	0	0
Square	Mean	Median filter	0.01	0.036
Square	Median	Median filter	0	0
Square	l_G	Median filter	0.036	0.174
Absolute	Mean	Median filter	0.016	0.042
Absolute	Median	Median filter	0	0
Absolute	l_G	Median filter	0	0
Biweight	Mean	Median filter	0.004	0.016
Biweight	Median	Median filter	0	0
Biweight	l_G	Median filter	0	0.008
Square	Mean	Hampel filter	0.026	0.088
Square	Median	Hampel filter	0.016	0.056
Square	l_G	Hampel filter	0.028	0.1
Absolute	Mean	Hampel filter	0.006	0.014
Absolute	Median	Hampel filter	0	0.002
Absolute	l_G	Hampel filter	0.002	0.002
Biweight	Mean	Hampel filter	0	0.01
Biweight	Median	Hampel filter	0	0.008
Biweight	l_G	Hampel filter	0	0.002

Table A.5: Results of testing change point detection methods on a random walk with a single change point in level, without drift using ∇X_t.

Cost function	Estimator	Type of filter	Exact efficiency	Efficiency $\lvert\tau-\tau_c\rvert \leq 2$
Square	Mean	No filter	0	0.004
Square	Median	No filter	0	0
Square	l_G	No filter	0	0.006
Absolute	Mean	No filter	0.006	0.022
Absolute	Median	No filter	0.002	0.024
Absolute	l_G	No filter	0	0.002
Biweight	Mean	No filter	0.006	0.022
Biweight	Median	No filter	0.004	0.024
Biweight	l_G	No filter	0	0.002
Square	Mean	Median filter	0.038	0.13
Square	Median	Median filter	0.026	0.12
Square	l_G	Median filter	0.014	0.04
Absolute	Mean	Median filter	0	0.014
Absolute	Median	Median filter	0.016	0.078
Absolute	l_G	Median filter	0.004	0.008
Biweight	Mean	Median filter	0.046	0.216
Biweight	Median	Median filter	0.028	0.136
Biweight	l_G	Median filter	0.024	0.096
Square	Mean	Hampel filter	0.004	0.018
Square	Median	Hampel filter	0.03	0.088
Square	l_G	Hampel filter	0.002	0.012
Absolute	Mean	Hampel filter	0.016	0.06
Absolute	Median	Hampel filter	0.024	0.102
Absolute	l_G	Hampel filter	0.01	0.04
Biweight	Mean	Hampel filter	0.042	0.168
Biweight	Median	Hampel filter	0.046	0.152
Biweight	l_G	Hampel filter	0.026	0.08

Table A.6: Results of testing change point detection methods on a random walk with a single change point in drift, using ∇X_t.

Cost function	Estimator	Type of filter	Exact efficiency	Efficiency $\lvert\tau - \tau_c\rvert \leq 2$
Square	Mean	No filter	0.002	0.014
Square	Median	No filter	0	0
Square	l_G	No filter	0.002	0.014
Absolute	Mean	No filter	0.002	0.006
Absolute	Median	No filter	0.002	0.002
Absolute	l_G	No filter	0	0
Biweight	Mean	No filter	0	0.004
Biweight	Median	No filter	0	0
Biweight	l_G	No filter	0	0
Square	Mean	Median filter	0.002	0.034
Square	Median	Median filter	0.014	0.038
Square	l_G	Median filter	0.006	0.036
Absolute	Mean	Median filter	0.004	0.024
Absolute	Median	Median filter	0.004	0.008
Absolute	l_G	Median filter	0.002	0.004
Biweight	Mean	Median filter	0.002	0.004
Biweight	Median	Median filter	0.004	0.012
Biweight	l_G	Median filter	0	0.002
Square	Mean	Hampel filter	0.006	0.044
Square	Median	Hampel filter	0.006	0.036
Square	l_G	Hampel filter	0.01	0.044
Absolute	Mean	Hampel filter	0.006	0.02
Absolute	Median	Hampel filter	0.002	0.012
Absolute	l_G	Hampel filter	0.002	0.006
Biweight	Mean	Hampel filter	0.006	0.008
Biweight	Median	Hampel filter	0.006	0.008
Biweight	l_G	Hampel filter	0.002	0.002

Table A.7: Results of testing change point detection methods on a random walk with a single change point in level, with a constant drift using ∇X_t.

Cost function	Estimator	Type of filter	Exact efficiency	Efficiency $\|\tau - \tau_c\| \leq 2$
Square	Mean	No filter	0.006	0.01
Square	Median	No filter	0.002	0.004
Square	l_G	No filter	0.006	0.01
Absolute	Mean	No filter	0.02	0.046
Absolute	Median	No filter	0.012	0.03
Absolute	l_G	No filter	0	0
Biweight	Mean	No filter	0.006	0.018
Biweight	Median	No filter	0.004	0.016
Biweight	l_G	No filter	0	0
Square	Mean	Median filter	0.012	0.032
Square	Median	Median filter	0.002	0.012
Square	l_G	Median filter	0.028	0.086
Absolute	Mean	Median filter	0.026	0.072
Absolute	Median	Median filter	0	0.002
Absolute	l_G	Median filter	0	0
Biweight	Mean	Median filter	0.014	0.068
Biweight	Median	Median filter	0.002	0.006
Biweight	l_G	Median filter	0.002	0.014
Square	Mean	Hampel filter	0.012	0.056
Square	Median	Hampel filter	0.078	0.192
Square	l_G	Hampel filter	0.01	0.062
Absolute	Mean	Hampel filter	0.018	0.136
Absolute	Median	Hampel filter	0.05	0.136
Absolute	l_G	Hampel filter	0.002	0.006
Biweight	Mean	Hampel filter	0.108	0.256
Biweight	Median	Hampel filter	0.056	0.17
Biweight	l_G	Hampel filter	0.012	0.024

Table A.8: Results of testing change point detection methods on a random walk with a single change point in both level and drift, using ∇X_t.