Technische Universität Dortmund
Fakultät Maschinenbau
IT in Produktion und Logistik

Master's Thesis

Integrated Hierarchical Forecasting with Mixed Distributed Demand Patterns on Large Data Sets

Lukas Rost
188032

Supervisors Technical University Dortmund:
Coach: Prof. Dr.-Ing. Markus Rabe
Coach: Dipl.-Inf. Anne Antonia Scheidler

Supervisor Erasmus University Rotterdam:
Coach: Prof. Dr. Jan van Dalen

Supervisors Company:
Coach: Robert Simpson
Coach: Anna Eschweiler

Date: March 8, 2018

Eidesstattliche Versicherung
(Affidavit)

Name, Vorname
(Last name, first name)

Matrikelnr.
(Enrollment number)

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem folgenden Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present Bachelor's/Master's* thesis with the following title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution.

Titel der Bachelor-/Masterarbeit*:
(Title of the Bachelor's/ Master's* thesis):

*Nichtzutreffendes bitte streichen
(Please choose the appropriate)

Ort, Datum
(Place, date)

Unterschrift
(Signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to €50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, section 63, subsection 5 of the North Rhine-Westphalia Higher Education Act (*Hochschulgesetz*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:**

Ort, Datum
(Place, date)

Unterschrift
(Signature)

**Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.

Acknowledgements

I would like to express my deep gratitude to Professor van Dalen, Professor Rabe and Dr. Scheidler, for their rich support, their detailed and useful feedback, and their encouragement during the last months.

Furthermore I would like to thank Robert Simpson and Anna Eschweiler for their time and support providing information and data for my research. Their fast and encouraging responses supported me well.

Finally I also would like to thank my family and friends for supporting me.

Lukas Rost
March 8, 2018

Abstract

This thesis aims to develop a new forecasting algorithm, called Intermittent Integrated Hierarchical Forecasting (IIHFC), for intermittent time series with an administrative hierarchy. The IIHFC algorithm is built on the algorithm of Pennings & Van Dalen (2017), from now called Integrated Hierarchical Forecasting (IHFC), which uses a Basical Structural Time Series Model (BSM).

The first and most simple is a naive forecast, where the last non-zero consumption in the estimation data is taken as forecast for the complete forecast period. The next two methods applied are the Exponential Smoothing State Space Model (ETS) and Auto-Regressive Integrated Mean Average (ARIMA) methods, widely used for non-intermittent demand. For intermittent demand Croston's methods (Croston 1972) and the Syntetos-Boylan approximation (SBA) (Syntetos & Boylan 2001)are well established. Finally the IHFC algorithm from Pennings & Van Dalen (2017) and the IIHFC algorithm from this thesis are compared.

The parameters in the last two methods are estimated with the help of the Low memory BFGS algorithm and the Kalman Filter. The hierarchical forecast in the first five methods is created as bottom-up forecast (Kahn 1998), the IHFC and IIHFC algorithms automatically generate reconciled forecasts for all hierarchical levels. The symmetric Mean absolute percentage error, Mean percentage error, Tracking Signal, and Consumption Performance Index are calculated to compare forecasting performance.

Depending on the average inter-demand interval different algorithms perform better. For strong intermittent consumption, Croston's method and the SBA perform similar and very well. Both have a small tendency to underestimate consumption. For less intermittent consumption, ETS and ARIMA perform better, with a small tendency to overestimate consumption. Depending on the data sets the IHFC and IIHFC can compete with the more simple algorithms. They offer more flexibility when creating a demand model, but require much more expertise and computational resources. In particular the estimation of the parameters can pose problematic for short time series.

Contents

List of Figures

List of Tables

1. Introduction

For most humanitarian organisations engaged in field operations, as well as for most industrial companies, demand forecasting is indispensable for a smooth and efficient supply chain. In industrial companies, for instance renown automotive companies, imprecise demand forecasts induces not only higher stock levels but can interrupt the entire production line. Humanitarian organisations, such as Médecins Sans Frontières (MSF), suffer from additional effects. Stock outs of humanitarian aid goods, as a results of unreliable forecasts, lead to an inadequate treatment and lost lives and consequently damage the public perception. To forestall these consequences humanitarian organisations raise their safety stocks which leads, in conjunction with high lead times up to six months, to an inefficient supply chain, more bound money and expired medication.

For a specific medical situation often a wide range of medication and medical items is needed, similar in industrial production the use of many products is often related. Including different hierarchical levels during the forecast process can cancel out noise and include more widespread information. This, together with the inclusion of product dependencies, has a strong impact on forecast accuracy, but induces a computational challenge.

Further complications are induced by different distributed demand. While there exist many applications and efficient algorithms for the convenient normal distribution, it is not suitable for a wide range of products. With its symmetry and negative values it is only applicable to fast moving products. Especially for slow moving products, where demand only occurs intermittent, other probability distributions are much more suitable. For the normal distribution exist a wast range of parameter estimations, e.g. the Kalman filter, many of which generate biased estimations for non-Gaussian distributions. Other parameter estimators, e.g. particle filters, suffer from the curse of dimensionality and hence can be only applied if there are very few products.

How is it then possible to create accurate forecasts, for a large number products, with different demand patterns, including product dependencies?

The top priority of this thesis is to develop a forecasting algorithm with improved forecast performance. To improve the forecast performance product dependencies are included as presented in Pennings & Van Dalen (2017), where the hierarchical structure of the organisation is used. An additional benefit are the integrated forecasts for different hierarchical levels, hence reducing operational complexity.

The second target is to create an algorithm applicable in practical situations, where often the number of products is immense. So significant effort is taken to keep the

computational requirements modest, which necessitates the use of sub-setting methods during the data preparation. Different distributed demand is included by the use of a non-Gaussian probability distribution in the model generation, which requires a modified parameter estimation.

Finally, to assess the performance of the forecasting algorithm real consumption data from Operational Center of Amsterdam of Médecins Sans Frontières (MSF-OCA) is available. The forecast performance of the new algorithm is compared with different algorithms for intermittent and non-intermittent demand.

2. Background

2.1. Statistical Methods

In this section different statistical properties are explained. At first different properties of time series are defined and their meaning is explained. In the next part different schemas for time series classification are presented, which can be sued to choose an appropriate forecasting algorithm. The classification depends on the statistical properties of the first part. Following that, different probability distributions are introduced. These distributions are used to create stochastic models for time series. The last part presents different forecast performance measures. With actual consumption for comparison, the quality of a generated forecast can be calculated. Different measures might produce different results, as errors are different weighted.

2.1.1. Statistical Properties of Time Series

Consumption time series from practical application have significantly different structures. These structures have a strong influence when modelling and forecasting future consumption. Three important aspects are the (relative) fluctuation of the demand size, the average inter-demand interval (adi), and the correlation of a time series with itself in time. In addition the behaviour of these statistical properties, in addition to the average consumption, in time is of interest. For this section a consumption time series of length n is noted as

$$X = (x_1, \ldots, x_n) \tag{1}$$

Coefficient of Variation The cv^2 is the variance of the time series divided by the squared mean. Assuming that $x_i \neq 0, i = 1, \ldots, n$ then the cv^2 is calculated as

$$cv^2 := \frac{\mathrm{Var}(X)}{\mathrm{E}(x)^2}. \tag{2}$$

The R implementation included in the tsintermittent (tsintermittent) package calculates the cv^2 only with the non-zero values (Kourentzes 2014), with no value for cv^2 if all $x_i = 0$. The cv^2 describes how strong the demand of a time series is fluctuating in relation to its mean value. The lower bound for the cv^2 is 0, the upper bound, given that all demand occurrences are non-negative is $n - 1$ (Katsnelson & Kotz 1957).

Average inter-demand interval Often stock review intervals are shorter than the time between consumption. This results in 0 consumption for the stock reviews in between. Not all consumption occurs in regular time intervals. Given a series of inter-demand intervals $p_1, \ldots, p_k, k \leq n$ for a time series X, then the adi p is calculated as

$$p := \frac{1}{k} \sum_{i=1}^{k} p_i. \tag{3}$$

.

An implementation is given by Kourentzes (2014) in the R package tsintermittent. It is important to note that this definition is different from $\frac{k+1}{n}$, which is the number of non-zero demand occurrences divided by the length of X. In formula 3 leading and trailing zero consumptions of X are omitted.

Auto-correlation function The gives the correlation of a time series with itself at a given time lag h and is defined in Brockwell & Davis (2016, p. 14) as

$$\mathrm{acf}(h) := \frac{1}{(n-h)\sigma^2} \sum_{i=0}^{n-h} (X_i - \mu)(X_{i+h} - \mu).$$

The auto-correlation function (acf) can be used to identify trends and seasonality in existing time series. A trend present in a time series, leads to a correlation between x_t and x_{t+1}. The strength of this correlation is calculated with acf(1), the type of trend can be identified by the slope of the acf. Figure B in the appendix shows the plotted acf values for a time series without trend 5, with a linear trend 6, and with trend following a quadratic term 7.

In addition seasonal effects can be identified with the help of the acf. Oscillating

values for the acf represent seasonal influences, where a large peak at lag s can be used to identify the length of a season. Figure 7 shows the acf values of a time series with an alternating seasonal term with seasonal length $s = 2$ 8 and of a sinusoidal seasonality with seasonal length $s = 12$ 9.

With the acf it can also be decided if the time series is uncorrelated in time. In theory, a time series following the standard normal distribution is uncorrelated in time. Taking the 0.975-quantiles of the standard normal distribution ± 1.96 and dividing them by $\sqrt{n}$ a boundary is defined, where a acf value shows correlation if it is outside of this boundaries. If more than 95% of the acf values are between these boundaries, it can be assumed that the time series is uncorrelated in time (Brockwell & Davis 2016, p. 16).

Stationary Time Series According to (Brockwell & Davis 2016, p. 13) a time series is called (weak) stationary if the mean value

$$\mu_X(t) := E(X_t) \tag{4}$$

and if the value of its auto-covariance

$$\mathrm{Cov}(x_t, x_{t+h}) \tag{5}$$

is independent from t for all values of h. The mean, variance, and other statistical properties of a stationary time series are invariant in time. Stationary time series have an important role when forecasting. The invariant mean, variance, and auto-covariance can be used to generate forecasts with similar properties. Given a stationary time series $X = (x_1, \ldots, x_n)$, with no significant auto-covariance values the forecast can be generated as $x_{n+1} = \mathrm{E}(X)$ where the uncertainty is given by $\sigma = \sqrt{Var(X)}$.

One example of a stationary time series is the identical independent distributed (iid) Noise (Brockwell & Davis 2016, p. 6), which is often used to model error terms. Observations in an iid Noise time series are independent draws from the same probability distribution,

$$X_t \sim P(\mu_X, \sigma_X^2) \tag{6}$$

$$\mathrm{Cov}(X_t, X_s) = 0 \qquad t \neq s. \tag{7}$$

An iid Noise time series can be forecast by its mean value, the uncertainty is again given by the variance. Error terms in forecasting are often an iid Noise time series with zero mean, also called White Noise. Terms following a White Noise distribution can be

Table 1: Limits of the SBC classification

Class	adi	cv^2
Smooth	$1 \leq p \leq 1.32$	$0 \leq cv^2 \leq 0.49$
Intermittent	$0.32 < p$	$0 \leq cv^2 \leq 0.49$
Erratic	$1 \leq p \leq 1.32$	$0.49 < cv^2$
Lumpy	$0.32 < p$	$0.49 < cv^2$

omitted when calculating a forecast.

2.1.2. Time Series Classification

Different forecast methods perform better depending on the statistical properties of the time series. In this section the three different classification schemas from the R packages tsintermittent are presented. They can be used to decided if Single Exponential Smoothing (SES), Croston's method, or the Syntetos-Boylan approximation (SBA) are more suitable for forecasting. All classification schemas are build on the adi and the cv^2. The different forecasting methods are presented in detail in section 3.2.4.

Syntetos-Boylan-Crostons classification The classification schema (Syntetos et al. 2005) has four classes, Smooth, Intermittent, Erratic, and Lumpy. The schema is presented in Figure 9 in the appendix. Smooth time series are characterized by a low adi and low variation. Intermittent time series have a higher adi but still a low variation. Erratic (Lumpy) time series are characterized by a high degree of fluctuation and a low (high) adi. Table 1 presents the limits for each class. Syntetos et al. (2005) argue that Crostons method (Crostons method) (Croston 1972) for Smooth time series and the SBASyntetos & Boylan (2001) for the all other classes perform best.

Kostenko-Hyndmans and Petropoulos-Kourentzess exact and approximate classifications Kostenko & Hyndman (2006) introduce an improved classification schema, classed Kostenko-Hyndman (KH) presented in Figure 11 in the appendix. Depending on the smoothing factor α used in the forecasting method, the cut-off value where SBA method produces better estimates than Crostons method method is defined as

$$cv^2 > \frac{4p(2-p) - \alpha(4-\alpha) - p(p-1)(4-\alpha)(2-\alpha)}{p(4-\alpha)(2p-\alpha)}$$

with

$$cv^2 > 2 - \frac{3}{2}p$$

as approximation, Kostenko-Hyndman approximate (KHa), independent from α. This schema was further extend in Petropoulos & Kourentzes (2015), who argues that for non-intermittent time series with $p \leq 1$ SES should be used. The resulting schemas Petropoulos-Kourentzes (PK) and Petropoulos-Kourentzes approximate (PKa) are presented in Figure 12 in the appendix.

2.1.3. Probability Distributions

Time series can be described as values drawn from a probability distribution. If the probability distribution together with its parameters are known, a forecast can be created as newly drawn values from this distribution. If draws from the distribution are independent in time, the mean of this distribution is used as forecast. This section introduces three probability distributions, the Normal distribution, the Bernoulli distribution, and the Possion distribution.

Normal Distribution The normal distribution is a continuous real-valued distribution, with the probability distribution function (pdf) defined as

$$\phi(x|\mu, \sigma^2) := \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{8}$$

where μ is the mean and σ is the standard deviation. If a variable ϵ_t is assumed to be normal distributed with mean μ_ϵ and standard deviation σ_ϵ it is noted as

$$\epsilon_t \sim N(\mu_\epsilon, \sigma_\epsilon^2), \tag{9}$$

in the univariate case and as

$$\epsilon_t \sim N(\mu_\epsilon, \Sigma_\epsilon) \tag{10}$$

in the multivariate case, where μ_ϵ is the vector containing the means and Σ_ϵ is the covariance matrix. Because the Normal distribution is symmetric, it is often used to model non-systematic errors In this case, the error term is assumed to be a White Noise time series with zero-mean Normal distributed values.

Bernoulli Distribution The Bernoulli Distribution is a discrete valued distribution (Johnson et al. 2005, p. 145), taking only the values zero and one. The probability mass function (pmf) is given as

$$\phi(X|\tilde{p}) := \begin{cases} \tilde{p} & X = 1 \\ 1 - \tilde{p} & X = 0 \end{cases}, \tag{11}$$

where $\tilde{p}$ is the probability X to be 1. The variance is then given by $\tilde{p}(1-\tilde{p})$. If a time series X_t is assumed to be Bernoulli distributed with probability $\tilde{p}$ it is noted as

$$X_t \sim Bernoulli(\tilde{p}).$$

The Bernoulli distribution is used to model the occurrence of a event. The probability of the event to happen is then $\tilde{p}$.

2.1.4. Forecast Performance Measures

In order to asses the forecasting performance of the different algorithm and to be able to compare different algorithms, statistical measures are needed. Here are three groups of forecast performance measures presented. Scale dependent measures, measures based on percentage errors, and special measures. In the multivariate case, let $A_{i,t}$ be the realised consumption of product i at time t, similar the forecast for product i at time t is noted as $F_{i,t}$. Let m be the number of products and let n be the length of the forecasting period. In the univariate case $A_t = A_{1,t}$ and $F_t = F_{1,t}$ are defined.

Scale Dependent Measures Mean Error (ME), Mean Absolute Error (MAE), and Root Squared Error (RMSE) are a few of many scale dependent measures. The following definitions are used.

$$ME := \frac{1}{n}\sum_{t=1}^{n} A_t - F_t \tag{12}$$

$$MAE := \frac{1}{n}\sum_{t=1}^{n} |A_t - F_t| \tag{13}$$

$$RMSE := \sqrt{\frac{1}{n}\sum_{t=1}^{n}(A_t - F_t)^2} \tag{14}$$

$$\tag{15}$$

Measures dependent on percentage errors The measures Mean percentage error (Mpe), Mean absolute percentage error (Mape), symmetric Mean percentage error (sMpe), and symmetric Mean absolute percentage error (sMape) are independent from the actual scale of the time series. The definitions of the measures are taken from Hyndman & Koehler (2006).

$$Mpe := \frac{1}{n}\sum_{t=1}^{n}\frac{A_t - F_t}{|A_t|} \tag{16}$$

$$Mape := \frac{1}{n}\sum_{t=1}^{n}\frac{|A_t - F_t|}{|A_t|} \tag{17}$$

$$sMpe := \frac{1}{n}\sum_{t=1}^{n}\frac{2(A_t - F_t)}{|A_t| + |F_t|} \tag{18}$$

$$sMape := \frac{1}{n}\sum_{t=1}^{n}\frac{2|A_t - F_t|}{|A_t| + |F_t|} \tag{19}$$

$$\tag{20}$$

If A_t and F_t are zero, the forecast matches the consumption and the summand in the error formulas is set to zero.

Special measures The Tracking Signal (TS) is used to see if an algorithm has the tendency to over or underestimate. It is calculated in the following way

$$TS := \frac{\sum_{t=1}^{n} A_t - F_t}{\frac{1}{n}\sum_{t=1}^{n}|A_t - F_t|}, \tag{21}$$

as defined in Trigg (2017). Values close to zero represents normal distributed forecast errors without a strong bias. In contrary, values far from zero represent a strong bias, with positive values representing underestimation and negative values overestimation.

The Consumption Performance Index (CPI) is used by MSF and is defined as perfor-

mance measure for multiple products in a project. It is defined as

$$CPI_t := \frac{1}{m}\sum_{i=1}^{m}(|A_{m,t} - F_{i,t}| < 0.5|F_{i,t}|), \tag{22}$$

$$CPI_i := \frac{1}{n}\sum_{t=1}^{t}(|A_{m,t} - F_{i,t}| < 0.5|F_{i,t}|), \tag{23}$$

$$CPI := \frac{1}{mt}\sum_{i=1}^{m}\sum_{t=1}^{n}(|A_{i,t} - F_{i,t}| < 0.5|F_{i,t}|), \tag{24}$$

where $(|A_{m,t} - F_{i,t}| < 0.5|F_{i,t}|)$ evaluates to 1 if the statement is true and to 0 otherwise.

2.1.5. Non-intermittent Demand Modelling and Forecasting

This section starts with a short introduction to different time series decompositions and the state space model. It continues with presenting different forecasting methods for non-intermittent demand. The forecasting methods are naive forecasting, the Box-Jenkins method, exponential smoothing, and Basical Structural Time Series Model (BSM). A time series of demand observation is noted as y_t.

Time series decomposition A time series y_t of demand observation can be decomposed in different components. Possible components include trend, seasonal, cycle, and external components in addition to the residuals. Depending on the us of the model these components can be combined additive, multiplicative, or as mixture of both. The additive model is given by

$$y_t = \mu_t + \gamma_t + c_t + d_t + \epsilon_t, t = 1, \ldots, n \tag{25}$$

$$\tag{26}$$

and the multiplicative model is given by

$$y_t = \mu_t \gamma_t c_t d_t \epsilon_t, t = 1, \ldots, n. \tag{27}$$

$$\tag{28}$$

Model 27 can be restated as model 25 by using the logged values for y_t (Durbin & Koopman 2001, p. 9).

An example of a mixed model, with additive and multiplicative combined components, is given in Hyndman et al. (2002). In this thesis only additive models are considered.

State Space model State space notation is similar for univariate and multivariate cases. For given observation vectors y_t the general discrete state space model is given as

$$y_t = Z_t(\alpha_t) + G_t(\alpha_t)\epsilon_t, \tag{29}$$

$$\alpha_{t+1} = T_t(\alpha_t) + H_t(\alpha_t)\eta_t, \qquad t = 1, \ldots, n. \tag{30}$$

The state vector α_t is considered to be unobserved and therefore unknown. The modelling error is given by $H_t(\alpha_t)\eta_t$, where η_t follows a probability distribution. Similar the measurement error is given by $G_t(\alpha_t)\epsilon_t$. For models with a single source of error, like in Ord et al. (1997) the error is changed so that

$$\eta_t = \epsilon_t, \qquad t = 1, \ldots, n. \tag{31}$$

For an additive time series decomposition an linear state space model. Model (29)-(30) can then be restated as matrix vector product and because of an additive error term $G_t(\alpha_t)$ and $H_t(\alpha_t)$ become independent from α_t. This results in the time variant linear state space model

$$y_t = Z_t\alpha_t + G_t\epsilon_t, \tag{32}$$

$$\alpha_{t+1} = T_t\alpha_t + H_t\eta_t, \qquad t = 1, \ldots, n. \tag{33}$$

A time invariant linear state space model is then given by

$$y_t = Z\alpha_t + G\epsilon_t, \tag{34}$$

$$\alpha_{t+1} = T\alpha_t + H\eta_t, \qquad t = 1, \ldots, n. \tag{35}$$

In literature different notation of state space models are used. The notation in this work follows the implementation in the R package Fast Kalman Filter (FKF) where the model used is similar to

$$y_t = Z_t\alpha_t + G_t\epsilon_t, \qquad \epsilon_t \sim \text{iid(I)}, \tag{36}$$

$$\alpha_{t+1} = T_t\alpha_t + H_t\eta_t, \eta_t \sim \text{iid(I)}, \qquad t = 1, \ldots, n. \tag{37}$$

The matrix I denotes the identity matrix of the dimension of y_t and α_t.

A Gaussian linear state space model in the form of

$$y_t = Z_t\alpha_t + \tilde{G}_t\epsilon_t^*, \epsilon_t^* \sim N(0, \Sigma_\epsilon), \tag{38}$$

$$\alpha_{t+1} = T_t\alpha_t + \tilde{H}_t\eta_t^*, \eta_t^* \sim N(0, \Sigma_\eta), \qquad t = 1, \ldots, n, \tag{39}$$

can be restated as model (34)-(35) if the covariance matrices Σ_ϵ and Σ_η are positive definite. As covariance matrices they are also symmetric, so the Cholesky decomposition Gentle (2009) can be calculated.d Therefore Σ_ϵ and Σ_η can be decomposed in

$$\bar{G}\bar{G}' = \Sigma_\epsilon, \bar{H}\bar{H}' = \Sigma_\eta \tag{40}$$

and model (38)-(39) is transformed into model (34)-(35) by

$$G_t = \tilde{G}_t\bar{G}, \qquad \text{and} \tag{41}$$

$$H_t = \tilde{H}_t\bar{H}, \qquad t = 1, \ldots, n. \tag{42}$$

2.1.6. Naive Forecast

The naive forecasting method is the most simple method. The last observation y_t is taken without modification as forecast for the future demand y_{t+1}. For forecasts h steps ahead this results in

$$y_{t+h} = y_t, h = 1, \ldots. \tag{43}$$

If a new observation is available it replaces the forecast. This forecasting method does not require many computational resources but also fails to model any trend, seasonality or error terms. The assumption is that y_t is a stationary time series with zero variance.

Remark: This forecasting method can be easily adapted for intermittent demand. If y_t is intermittent then the last non-zero consumption $\tilde{y}_t$ is taken instead of y_t. This will result in a strong overestimation of future demand.

2.1.7. Box-Jenkins Method

The Box-Jenkins method (Box et al. 2008) is used to fit an Auto-Regressive Mean Average (ARMA) or Auto-Regressive Integrated Mean Average (ARIMA) model to an univariate time series of observations. The resulting model is then used to generate a forecast. The

Box-Jenkins method does not explicitly model the components 25 of the time series but tries to find a suitable ARMA(p,q) model, where p denotes the order of the AR process and q the order of the MA process. The model is then given as

$$y_t - \alpha_1 y_{t-1} - \ldots - \alpha_p y_{t-p} = \epsilon_t + \theta_1 \epsilon_{t-1} + \ldots + \theta_q \epsilon_{t-q}, \tag{44}$$

with the regression coefficients α_i from the AR(p) process and the smoothing coefficients θ_i for the MA(q) process. The ARMA model requires a the observations to be a stationary time series. The observations y_t can be replaced with the differentiated observations

$$y'_t = y_t - y_{t-1} \tag{45}$$

to remove non-stationary components. It may be necessary to repeat the differentiation 45 several times to archive sufficient stationary.

An ARIMA(p, d, q) process without seasonality is describe by

$$y'_t = \alpha_1 y'_{t-1} + \ldots + \alpha_p y'_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \ldots + \theta_q \epsilon_{t-q}, \qquad \epsilon_t \sim \text{iid}(I) \tag{46}$$

$$y'_t = \sum_{k=0}^{d} \binom{d}{k} (-1)^k y_{t-k}, \tag{47}$$

$$\tag{48}$$

with d being the order of differentiation.

A seasonal component of period s can be included either by seasonal differentiating

$$y^*_t = y_t - y_{t-s} \tag{49}$$

or by adding an AR(s) or MA(s) term to equation 44. The seasonal component itself may follow an ARMA model of higher degree. A seasonal ARIMA model is noted as ARIMA$(p, d, q)(P_s, D_s, Q_s)$ where P_s, D_s, and Q_s denote the order of the seasonal component.

The acf and partial auto-correlation function (pacf) (Brockwell & Davis 2016, p. 16,62)can be used to determine the order of the ARIMA process (Brockwell & Davis 2016, p. 79,83-84). When the order of the model is determined the regression coefficient and smoothing factors need to be estimated. The most used method to estimate these parameters it the maximum likelihood (Brockwell & Davis 2016, p. 140), which is further explained in section 2.2.3.

After the parameters for the ARIMA model are fitted a forecast can be generated by

using equations 46 and 47. From equation 47 follows that

$$y_{t+1} = -y'_{t+1} + \sum_{k=1}^{d} \binom{d}{k} (-1)^k y_{t-k+1} \tag{50}$$

$$\tag{51}$$

and from equation 46 y'_{t+1} can be estimated as

$$y'_{t+1} = \alpha_1 y'_t + \ldots + \alpha_p y'_{t-p+1} + \theta_1 \epsilon_t + \ldots + \theta_q \epsilon_{t-q+1} \tag{52}$$

where ϵ_{t+1} is set to 0. An h step ahead forecast can be generated by iterative application of 50 and 52.

For multivariate time series instead of ARMA and ARIMA the Vector Auto-Regressive Mean Average (VARMA) and Vector Auto-Regressive Integrated Mean Average (VARIMA) models can be used (Box G. E. P. & TIAO 1977).

The assumption that the time series y'_t is stationary can be problematic in real application. Commandeur & Koopman (2007, p. 133) argue that real time series are non-stationary independent form the order of differentiation d. Therefore the question arise how stationary is stationary enough.

2.1.8. Exponential Smoothing

A widely used method to generate forecasts for univariate time series is SES. SES was first presented in Brown (1959, p. 52) and calculates a weighted average of past values. Differently to the Moving Average process in the last section, all past observations are included and the weights are applied multiplicative. The smoothed value $\tilde{y}_t$ for time t for a given time series y_t is

$$\tilde{y}_t = \alpha y_t + (1-\alpha)\tilde{y}_{t-1}, \qquad \tilde{y}_0 = y_0 \tag{53}$$

$$= \alpha y_t + \sum_{k=1}^{t} (1-\alpha)^k y_{t-k}. \tag{54}$$

In this equation α is the smoothing factor. A forecast is then generated by setting $y_{t+1} = \tilde{y}_t$, similar a h step ahead forecast is generated as (Brown 1959, p. 52)

$$y_{t+h} = \tilde{y}_t. \tag{55}$$

In the basic equation 53 seasonality and other components are not explicitly considered.

A refined exponential smoothing model is created by decomposing the time series and then applying exponential smoothing to forecast each component separately. For the additive decomposition in equation 25 this results in

$$y_t = \mu_t + \gamma_t + \epsilon_t, \qquad \epsilon_t \sim N(0, \sigma_\epsilon) \tag{56}$$

$$\mu_t = \mu_{t-1} + \beta_{t-1} + \alpha_1 \epsilon_t, \tag{57}$$

$$\beta_t = \beta_{t-1} + \alpha_1 \alpha_2 \epsilon_t \tag{58}$$

$$\gamma_t = \gamma_{t-s} + \alpha_3 \epsilon_t \tag{59}$$

taken from Hyndman et al. (2002).

Hyndman et al. (2002) created a universal framework to include multiplicative and mixed models. They then rewrite the different equations in state space form, where the state space model is linear when each component is additive. The additive model 59 then has the state vector

$$\alpha_t = (\mu_t, \beta_t, \gamma_t, \gamma_{t-1}, \ldots, \gamma_{t-s+1})'. \tag{60}$$

With the state space notation also multivariate models can be noted.

2.1.9. Basic StructuralModel

BSM where introduced in Harvey (1989) and explicitly model different components of the time series decomposition. If the time series decomposition is additive like in 25 the BSM can be written as linear state space model (34)-(35). In addition BSM offer the possibility to include cycle components and external factors. Differently to the Box-Jenkins method they do not require a stationary time series.

In this section different possibilities to model the trend and seasonal components are presented for univariate time series.

Modelling the Trend component The most simple model for the trend component is the local level model. In this model it is assumed that there is no trend present in the time series. The trend component is then modelled as

$$\mu_{t+1} = \mu_t + \eta_t, \qquad \eta_t \sim N(0, \sigma_\eta^2). \tag{61}$$

The model described a random walk of the trend component μ_t, with an added White Noise error η_t.

A simple, non-stationary linear trend μ_t in the consumption can be modelled by adding a slope β_t which is generated by a random walk. This can extend the local level model into the local linear trend model (Durbin & Koopman 2001, p. 44) defined as

$$\mu_{t+1} = \mu_t + \beta_t + \eta_t, \qquad \eta_t \sim N(0, \sigma_\eta^2) \tag{62}$$

$$\beta_{t+1} = \beta_t + \zeta_t, \qquad \zeta_t \sim N(0, \sigma_\zeta^2). \tag{63}$$

Another possibility is the additive damped trend model Gardner & Mckenzie (1985) defined as

$$\mu_{t+1} = \mu_t + \beta_t + \eta_t, \qquad \eta_t \sim N(0, \sigma_\eta^2) \tag{64}$$

$$\beta_{t+1} = \delta\beta_t + \zeta_t, \qquad \zeta_t \sim N(0, \sigma_\zeta^2). \tag{65}$$

It is also possible that μ_t follows an auto-regressive process, the model can then be written as

$$\mu_{t+1} = \delta\mu_t + \beta_t + \eta_t, \qquad \eta_t \sim N(0, \sigma_\eta^2) \tag{66}$$

$$\beta_{t+1} = \beta_t + \zeta_t, \qquad \zeta_t \sim N(0, \sigma_\zeta^2). \tag{67}$$

If the time series is multivariate the variances σ_η^2 and σ_ζ^2 can be replaced by covariance matrices, to reduce the number of parameters it may be interesting to assume that the covariance matrices are diagonal or even multiplicatives of the identity matrix, compare with the approach in Pennings & Van Dalen (2017). The variables μ_t and β_t are then vectors and δ is replaces with the diagonal matrix $\Delta = \text{diag}(\delta_1, \ldots, \delta_m)$.

Modelling the Seasonal component If the seasonal pattern is constant in time, then it can be noted down for each season as constant γ_j with

$$\sum_{j=1}^{s} \gamma_j = 0 \tag{68}$$

with a seasonal length of s. Equation 68 can be rewritten as

$$\gamma_{t+1} = -\sum_{j=1}^{s-1} \gamma_{t+1-j}. \tag{69}$$

In general seasonality should be allowed to change over time, which can be archived by adding an error term ω_t resulting in

$$\gamma_{t+1} = -\sum_{j=1}^{s-1} \gamma_{t+1-j} + \omega_t, \qquad \omega_t \sim N(0, \sigma_\omega^2). \tag{70}$$

Another possibility to write as seasonal component is in a trigonometric form (Durbin & Koopman 2001, p. 46)

$$\gamma_t = \sum_{j=1}^{\lfloor s/2 \rfloor} \gamma_{j,t}, \qquad \lambda_j = \frac{2\pi j}{s} \tag{71}$$

$$\gamma_{j,t+1} = \gamma_{j,t}\cos(\lambda_j) + \gamma_{j,t}^* \sin(\lambda_j) + \omega_t, \qquad \omega_t \sim N(0, \sigma_\omega^2) \tag{72}$$

$$\gamma_{j,t+1}^* = -\gamma_{j,t}\sin(\lambda_j) + \gamma_{j,t}^* \cos(\lambda_j) + \omega_t^*, \qquad \omega_t \sim N(0, \sigma_{\omega^*}^2. \tag{73}$$

This trigonometric includes damped error terms ω_t and ω_t^*, often parameters are reduced by setting $\sigma_\omega = \sigma_{\omega^*}$. Similar to the trend models the seasonal model in the multivariate case is created by setting $\gamma_t, \gamma_{j,t}, \gamma_{j,t}^*$ as vectors and ω_t, ω_t^* as draws from the multivariate normal distribution $N(0, \Sigma_\omega)$.

Basic Structural Time Series Models A BSM can be written as combination of any trend and seasonal model, additional cycle and external components can be added. Combining a multivariate linear trend model with an autoregressive model for μ_t and a damped seasonal component results in the following model

$$y_t = \mu_t + \Phi\gamma_t + \epsilon_t, \qquad \epsilon_t \sim N(0, \Sigma_\epsilon) \tag{74}$$

$$\mu_{t+1} = \Delta\mu_t + \beta_t + \eta_t, \qquad \eta_t \sim N(0, \Sigma_\eta) \tag{75}$$

$$\beta_{t+1} = \beta_t + \zeta_t, \qquad \zeta_t \sim N(0, \Sigma_\zeta) \tag{76}$$

$$\gamma_t = \sum_{j=1}^{\lfloor s/2 \rfloor} \gamma_{j,t}, \qquad \lambda_j = \frac{2\pi j}{s} \tag{77}$$

$$\gamma_{j,t+1} = \gamma_{j,t}\cos(\lambda_j) + \gamma_{j,t}^* \sin(\lambda_j) + \omega_t, \qquad \omega_t \sim N(0, \Sigma_\omega) \tag{78}$$

$$\gamma_{j,t+1}^* = -\gamma_{j,t}\sin(\lambda_j) + \gamma_{j,t}^* \cos(\lambda_j) + \omega_t^*, \qquad \omega_t \sim N(0, \Sigma_{\omega^*} \tag{79}$$

$$\Phi = \mathrm{diag}(\phi_1, \ldots, \phi_n) \tag{80}$$

$$\Delta = \mathrm{diag}(\delta_1, \ldots, \delta_n). \tag{81}$$

In this model the seasonal effects can be scaled with the diagonal matrix Φ.

This a model can be used to generate a forecast by removing all error terms and projecting the model h steps ahead. Another possibility is to restate model (74)-(81) as state space model in the form of a time invariant (34)-(35) (Durbin & Koopman 2001, p. 46). The h step ahead forecast is then generate by generating the h step ahead state vector α_t with equation (35) and then applying the measurement equation (34). This results in

$$y_{y+h} = Z\alpha_{t+h} \tag{82}$$

$$\alpha_{t+h} = T^h\alpha_t. \tag{83}$$

.

As state above the BSM does not require stationary time series, the disadvantage is that the number of parameters in the BSM model is much higher than the number of parameters used in the Box-Jenkins model. In addition to the model parameters of the BSM also an initial state vector is needed, therefore estimating the parameters for a BSM uses more compsutational resources.

2.2. Intermittent Demand Modelling and Parameter Estimation

The models and forecasting methods in section 2.1.5 perform good on non-intermittent demand. For intermittent, particular with stochastic inter-demand intervals, they perform poorly. The algorithms are build on an update of consumption in the past, which are mainly 0 for intermittent demand. This section first presents two forecasting methods for intermittent demand. These forecasting methods provide a forecast of the average consumption, not a forecast if demand occurs. The second part elaborates on methods for the parameter estimation.

2.2.1. Crostons Method

The first work on modelling and forecasting of intermittent demand is from Croston (1972). He assumes that the occurrence of demand is independent from the demand size. His model is noted as

$$y_t = x_t z_t \tag{84}$$

where $x_t \in \{0, 1\}$ notes if demand occurs at time t and z_t represents the demand size. Further he assumes that

$$x_t \sim Bernoulli(\frac{1}{p_t}), z_t \sim N(\mu, \sigma_\epsilon^2), \tag{85}$$

where p_t is the adi at time t and z_t is drawn independently. Croston uses then separate exponential smoothing equations for demand size and demand occurrence

$$p_t = p_{t-1}, \qquad y_t = 0 p_t = \alpha p_{t-1} + (1 - \alpha) q, \qquad y_t \neq 0 \tag{86}$$

where q is the last inter-demand interval. The exponential smoothing equations are only updated when there is a non-zero demand.

The model is derived from the equation

$$\mathrm{E}(y_t) = \frac{\mu}{p} \tag{87}$$

, and with the update only when there is non-zero demand, the actual demand size is not influenced by the zero consumption.

A forecast is then generated similar to equation (55) for SES as

$$y_{t+h} = \frac{\tilde{z}_t}{p_t}. \tag{88}$$

2.2.2. Syntetos-Boylan-Approximation

Syntetos & Boylan (2001) improved Crostons method, as it containted a mathematical mistake. Croston assumed that

$$\mathrm{E}(y_t) = \mathrm{E}(\frac{z_t}{p_t}) = \mathrm{E}(z_t)\,\mathrm{E}(\frac{1}{p_t}) = \frac{\mathrm{E}(z_t)}{\mathrm{E}(p_t)} \tag{89}$$

where the last equation is not true. Syntetos & Boylan (2001) propose an approximation given by

$$\mathrm{E}(y_t) = \mathrm{E}(z_t)\,\mathrm{E}(\frac{1}{p_t c^{p_t - 1}}) \approx \frac{\mu}{p}. \tag{90}$$

with the second equation equal for $c = \infty$. With a large enough value for c, in general $c > 100$, the approximation removes most of the bias in Crostons method. Instead of the exponential smoothing equation (86) for p_t a similar equation is used for $\frac{1}{p_t c^{p_t - 1}}$, where

p_t is recorded similar. The forecast is then given by

$$y_t = z_t \frac{1}{p_t c^{p_t - 1}}. \tag{91}$$

2.2.3. Maximum Likelihood Estimation

As mentioned before the maximum likelihood method is often used to estimate necessary parameters. For the Box-Jenkins method the parameters are the regression coefficient and the smoothing factors, for the exponential smoothing it is the smoothing factors and for the BSM the parameters are the model parameters in addition to the initial state vector α_0, and the initial covariance matrix P_0. Other possibilities to calculate α_0, P_0 are given in Durbin & Koopman (2001, p. 123-146) If a demand model, a set of parameters θ, and a set of observations $Y_n = (y_1, \ldots, y_n)'$ are given, then the likelihood function $L(Y_n|\theta)$ descries the likelihood of the observations Y_n if the model is given with parameters θ. As the observations in $Y_{t-1} = (y_1, \ldots, y_{t-1})'$ are known when y_t is calculated can the likelihood function be rewritten as

$$L(Y_n|\theta) = p(y_1, \ldots, y_n|\theta) = p(y_1|\theta) \prod_{t=2}^{n} p(y_t|Y_{t-1}, \theta)). \tag{92}$$

Often the log-likelihood is calculated instead to improve the numerical stability. Equation 92 is then restated as

$$LogL(Y_n|\theta) = \sum_{t=1}^{n} \log p(y_t|Y_{t-1}, \theta), p(y_1|Y_0, \theta) := p(y_1|\theta). \tag{93}$$

The maximum likelihood method then aims to maximise $L(Y_n|\theta)$ what is similar to minimizing the negative log-likelihood given by the negative of 93. The different optimisations algorithms are explained in section 2.2.5.

2.2.4. Kalman Filter

In order to efficiently calculate the log-likelihood of parameter set for the BSM given in equation (74)-(81) the Kalman Filter (KF) can be used. The KF was first published in Kalman (1960) and can be used to adjust the state vector of a state space model when observations are available. Durbin & Koopman (2001, p. 43) give a notation of the KF which modified for model (34)-(35) defines the KF as

$$v_t = y_t - Z\alpha_t, \qquad F_t = ZP_tZ' + GG',$$
$$\alpha_{t+1} = T\alpha_t + K_t v_t, \qquad P_{t+1} = TP_t(T - K_tZ)' + HH',$$

where $K_t = TP_tZ'F_t^{-1}$ is called Kalman gain. The distinction between the estimate of the state vector a_t and the state vector α_t was dropped in this notation. If the state vector at time t was given as α_t, then the innovation v_t describes what part of the observations was not explained by the current state vector.

In case an observation is missing the update reduces to

$$\alpha_{t+1} = T\alpha_t, \qquad P_{t+1} = TP_tT' + HH', \tag{94}$$

as shown in (Durbin & Koopman 2001, p. 111).

With the KF the log-likelihood for the BSM can then be calculated according to Durbin & Koopman (2001, p. 171) as

$$LogL(Y_n|\theta) = -\frac{np}{2}\log 2\pi - \frac{1}{2}\sum_{t=1}^{n}(\log|F_t| + +v_t'F_t^{-1}v_t). \tag{95}$$

2.2.5. Optimisation Algorithms

Newton's method is the most known algorithm to minimise a function. In general more than one parameter needs to be estimated, the multidimensional Newton iteration for a function f with parameter θ is then given by

$$\theta_{n+1} = \theta_n - J_f(\theta_n)^{-1}f(\theta_n) \tag{96}$$

where $J_f(\theta_n)^{-1}$ is the inverse of the Jacobi matrix of f at point θ_n. Calculating this inverse is often not possible or to expensive, Quasi-Newton method replace therefore $J_f(\theta_n)$ by an approximation. Parameters which minimize f, at the same time evaluate to 0 in the derivative f'. Quasi-Newton methods try to find parameter sets where the vector values function f' evaluates to zero. The Jacobi matrix of f' is then the Hessian matrix of f, which is symmetric. The most common used Quasi-Newton method is the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) which was original published in BROYDEN (1970), Fletcher (1970), Goldfarb (1970), and Shanno (1970). The BFGS algorithm updates the approximation of the Hessian matrix, so that the updated

approximation stays positive definite.

In order to limit the memory needed to store the approximation of the Hessian matrix, Byrd et al. (1995) introduces a limited memory version of the BFGS. Instead of storing the approximation at each step of the iteration, only a small number of correction pairs for different search directions are stored. This pairs can then be used to define the approximation of the Hessian matrix.

Newton's method, as well as the Quasi-Newton approximations converge in general only toward a local minimum. This problem can partially be avoided by using different, random generated initial parameter sets, also called multi-start optimisation (Tu & Mayne 2002).

2.3. Hierarchical Gorecasting

Most organisations and companies have a hierarchical structure in their administration. Forecasts of future demand may therefore be needed on different hierarchical levels and it is important that the aggregated forecast of the lower levels add up to the forecast of the higher level, the forecasts need to be reconciled.

Three classical methods to create hierarchical forecasts are the top-down, bottom-up (Widiarta et al. 2009), and the middle-out forecasts. This section gives a short introduction on these methods, in addition methods to reconcile forecasts generated on all levels are presented. Finally the section concludes with the forecasting algorithm of Pennings & Van Dalen (2017) which automatically generates a reconciled hierarchical forecast.

2.3.1. Basic Hierarchical Forecast

Top-down The top-down approach only generates a forecast for the highest level in the hierarchy. This forecast is then decomposed in order to create the forecast at lower levels. One common decomposition method uses the average of the historical sales as decomposition weights. This can be either the average of the proportion in each time period, which results in

$$p_j = \frac{1}{n} \sum_{t=1}^{n} \frac{y_{j,t}}{y_t} \tag{97}$$

or the proportion of the complete time period

$$p_j = \frac{\sum_{t=1}^{n} y_{j,t}}{\sum_{t=1}^{n} y_t}. \tag{98}$$

Additional ways to decompose top-level forecasts are presented in Gross & Sohl (1990).

bottom-up The bottom-up approach starts at the lowest level in the hierarchy. For each base product a forecast is generated. These forecasts are then aggregated according to the hierarchical structure to generate the forecasts for the higher levels. Dangerfield & Morris (1992) showed that in most cases this method generates better forecasts then the top-down approach. In addition, the decision how to decompose the forecast is omitted here.

middle-out The middle-out approach combines the top-down and the bottom-up approaches. The forecast is generated on a intermediate level of the hierarchy. From this level the forecasts are then aggregated to create the forecasts for the higher hierarchy and decomposed to generate the forecasts for the lower levels.

2.3.2. Hierarchical Reconciliation

In addition to the above mentioned methods, forecasts can be generated for all hierarchical levels. In order to add up according to the hierarchical structure, these forecasts need to be reconciled. Hyndman et al. (2011) present different methods which can be used to reconcile forecasts. Assuming that S is the design matrix describing the hierarchical structure of the forecast, the reconciliation follows the following approach. From the consumption of the base products x_t the consumption in the other levels of the hierarchy can be calculated as Sx_t. In a first step the consumption of the base products x_t is created from the consumption of all levels. $x_t = Py_t$, x_t is therefore the consumption at bottom level, adjusted with the influence of the top levels. The reconciled forecast y_t for all levels is then generated with $\tilde{y}_t S x_t = SPy_t$. One possibility for P is given as

$$P = (S'S)^{-1}S' \tag{99}$$

2.3.3. Integrated Hierarchical Forecasting

Pennings & Van Dalen (2017) present an additional method to create hierarchical forecasts. Differently from the other methods, the hierarchical structure is not added after the forecasts are generated, but already in the forecasting model. Their model is based on the linear time-invariant Gaussian BSM (38)-(39), with diagonal Σ_ϵ, representing independent measurement errors, and

$$\Sigma_\zeta = \sigma_\zeta^2 \mathrm{I}, \tag{100}$$

$$\Sigma_\omega = \Sigma_{\omega^*} = \sigma_\omega^2 \mathrm{I}. \tag{101}$$

. Dependencies between different products are included in the non-diagonal covariance matrix Σ_η. The observation vector y_t does not only contain observations from the base products but from all hierarchical levels. The hierarchical structure is represented in design matrix S. The measurement equation (74) is then updated to

$$y_t = S(\mu_t + \Phi\gamma_t) + \epsilon_t, \qquad \epsilon_t \sim N(0, \Sigma_\epsilon). \tag{102}$$

While the state vector $\alpha_t = (\mu_t, \beta_t, \gamma_{j,t}, \gamma^*_{j,t})'$ contains only information about the base products, the update step applied by the KF includes also the observations of the higher hierarchical levels.

Pennings & Van Dalen (2017) estimate the parameters by using the the BFGS algorithm with 500 randomized starts and the KF to calculate the log-likelihood.

3. Algorithm Development

In this section a new algorithm for intermittent demand forecasting is developed. The section is divided in three parts where the first part presents the available consumption data and the steps taken to prepare this data. In the second part the theoretical framework of the new algorithm is developed. The last part explains the different steps taken to implement this new algorithm as well as the algorithms used for comparison.

3.1. Data Preparation and Selection

Before developing an advanced forecasting algorithm, data is needed to asses the forecasting performance. Johnston et al. (2003) showed that up to 60% of products can follow an intermittent demand pattern. For medical supplies, particular in humanitarian aid organisations, the similar is true. Therefore, the consumption data of these supplies

can be classified as presented in section 2.1.2. Consumption data from real applications has advantages over randomly generated data, because it includes unexpected cases. This data can be used to estimate parameters of models and algorithms and to asses the forecast performance, with results similar to those in practice. Before developing an advanced algorithm, the available data has to be prepared. In this section, the data sources and the data itself are described, the data is cleaned, and part of the data is selected for testing. For privacy protection all information about projects, departments and products is masked with generated codes.

3.1.1. Data Sources

This thesis is supported by a data sources from an organisations of MSF. The MSF-OCA provides their Consumption Tools from several projects available as an Excel spreadsheet.

Consumption Tool The Consumption Tool is a Excel spreadsheet used by MSF-OCA. For each project the consumption is recorded in a separate Excel file. MSF-OCA provides forty six Consumption Tools files for this work. This file consists of several tabs, including general product information, a mask to enter realised consumption, a mask to enter out-of-stocks, in MSF called ruptures, a interface to show only information for a particular department, and an overview tab which includes a Consumption Performance Indicator.

Separate tabs are used as a database. These databases are read with an R script to extract the product codes, departments, month and amount of consumption. The project names are masked with CT0001 to CT0047.

3.1.2. Data Description

The data tables extracted from the Consumption Tool contain the project code, the department code, the product code, the month, and the amount of consumption. The product code contains additional information about product categorisation. There are three different product categories, which ordered by their hierarchical order are Group, Family, and Root. The Group category describes the general use of the product in projects, such as administrative equipment, logistic supplies, drugs, medical supplements, kits, nourishment, medical tests, and transport equipment. This work focuses on medical supplies, therefore only drugs and medical tests are included. The Family category is in example used to categorize drugs further into vaccines, oral drugs, and injectable drugs among others. The Root category describes the type of medication, in example all vaccines against a specific disease are in the same Root category.

For the further work the product code is separated into the above described parts. This results in a table similar to Table 2 with the columns Project, Department, Group, Family, Root, Product, Month, and Amount.

Table 2: Example flux table with masked data examples.

Project	Department	Group	Family	Root	Product	Month	Amount
CT0001	1	A	ABA	AGGA	AFA	Dez 16	2
CT0001	2	A	ABA	AGGA	AFA	Dez 16	16
CT0001	1	A	ABA	AGGA	AFA	Dez 16	28
CT0001	2	A	ABA	AGGA	AFA	Nov 16	59
CT0001	1	A	ABA	AGGA	AFA	Nov 16	8
CT0001	2	A	ABA	AGGA	AFA	Okt 16	34

3.1.3. Data Cleaning

Before the data can be used to classify demand, estimate parameters, and asses the forecast performance, the data has the be cleaned. At first the duplicates and unreasonable entries are deleted. Next, time series are constructed for each product from the flux table. In the last step, missing values in contrary to zero consumption are identified. Finally, this section closes with a short note why outlier detection was omitted.

Removing Duplicates The flux table 2 contains duplicated entries. These entries contain exactly the same information including the month and amount of consumption. Hence, it is assumed that the duplications results from a software error or human mistake. In the data from MSF-OCA are 350 entries duplicates out of 1 141 204 which leaves 1 140 854 entries.

Identify Unreasonable Entries After removing duplicated entries, entries containing empty fields are removed. With a empty field, it cannot be used to construct the time series because identification of the product or month is not clear. In some cases, the missing values were due to errors in the Excel files. Some lines where shifted in their position, which is not recognised by the parsing algorithm in R.

Construct Time Series Table Next, the time series table is constructed from the flux table. For this, all lines with the same product and project information are combined and the different months are added as new columns. In the end, the monthly columns are sorted by date. This results in table with one line for each distinctive entry in the

Table 3: Example time series table with masked data examples.

Project	Dep.	Group	Family	Root	Product	Jan 2013	Feb 2013	...
CT0001	1	A	AAA	AAAA	AAA	0	0	...
CT0001	1	A	AAA	AAAA	BAA	0	0	...
CT0001	1	B	BAA	BAAA	CAA	83	66	...
CT0001	1	B	CAA	CAAA	DAA	10	4	...
CT0001	1	B	CAA	DAAA	EAA	0	1	...
CT0001	1	B	CAA	EAAA	FAA	0	0	...

flux table, hence 129 312 lines for MSF-OCA. The structure of this table is similar to Table 3.

Identifying Missing Values After the time series tables are constructed, missing values need to be distinguished from zero consumption. Zero consumption is often not entered into the flux table, and therefore also not in the time series table. In order to distinguish between missing values and zero consumption algorithm A.2.1 in the appendix is applied. The algorithm is build on the assumption that there are two reasons why there may be a missing value instead of zero consumption (Lukas Rost n.d.). When applying this algorithm, a entry with zero consumption is treated as consumption in contrary to a missing value.

The first reason may be that a project did not yet start or was already finished. In addition a project may be interrupted for some time for security reasons. In this cases a project is considered closed, which means consumption for all products in this project should be missing from the time series table in the specific months. A project is considered open when it is not closed. This is visualised in Figure 1.

ProjectCode	Jan 13	Feb 13	Mar 13	Apr 13	May 13	Jun 13	Jul 13	Aug 13	Sep 13
CT0001	NA	NA	0	1	122	0	0	0	766
CT0001	NA	NA	NA	76	33	0	NA	0	12
CT0001	NA	NA	0	56	98	NA	0	0	54
CT0001	NA	NA	NA	12	43	0	32	NA	22
CT0001	NA	NA	0	12	123	453	34	0	NA
CT0002	66	45	24	1	32	12	NA	NA	NA
CT0002	NA	64	NA	23	3	43	NA	NA	NA
CT0002	NA	45	567	NA	2	54	NA	NA	NA
CT0002	87	NA	56	43	NA	34	NA	NA	NA
CT0002	0	0	0	0	0	0	NA	NA	NA
CT0002	12	33	7	55	67	83	NA	NA	NA
CT0003	38	NA	0	NA	23	NA	47	NA	0
CT0003	NA	4	87	NA	0	NA	32	213	0
CT0003	53	0	NA	NA	0	NA	3	NA	733
CT0003	62	NA	15	NA	34	NA	23	0	325
CT0003	75	NA	3437	NA	34	NA	63	123	0

Project Open Project Closed

Figure 1: Figure explaining when a project is open or closed.

The second reason is that a single department was opened after the project started, or closed before the project finished. This would be the case when the consumption for all products in department is missing from the beginning up to a certain month, or from a certain month until the end. In this cases we consider this department as closed, in the time between the first consumption and the last consumption it is considered open. This is visualised in Figure 2.

Department	Jan 13	Feb 13	Mar 13	Apr 13	May 13	Jun 13	Jul 13	Aug 13	Sep 13
Dep1	NA	NA	0	1	NA	0	0	NA	NA
Dep1	66	45	24	1	NA	12	NA	NA	NA
Dep1	NA	64	NA	23	NA	43	NA	NA	NA
Dep1	NA	45	567	NA	NA	54	NA	NA	NA
Dep1	87	NA	56	43	NA	34	NA	NA	NA
Dep2	NA	NA	0	0	NA	0	NA	NA	NA
Dep2	NA	NA	NA	76	33	0	NA	NA	12
Dep2	NA	NA	0	56	98	NA	0	NA	54
Dep2	NA	NA	NA	12	43	0	32	NA	22
Dep2	NA	NA	0	12	123	453	34	NA	NA
Dep2	NA	NA	3437	NA	34	NA	63	NA	0

Department Open Department Closed

Figure 2: Figure explaing when a department is open or closed.

If a department, and the associated project, is open but there is a missing value, this value is treated as zero consumption. The full process is visualised in the example in

Figure 3

ProjectCode	Department	Jan 13	Feb 13	Mar 13	Apr 13	May 13	Jun 13	Jul 13	Aug 13	Sep 13
CT0001	Dep1	NA	NA	NA	NA	122	0	0	0	766
CT0001	Dep1	NA	NA	NA	NA	33	0	NA	0	12
CT0001	Dep2	NA	NA	0	56	98	NA	0	0	NA
CT0001	Dep2	NA	NA	NA	12	43	0	32	NA	NA
CT0001	Dep2	NA	NA	0	12	123	453	34	0	NA
CT0002	Dep3	NA	45	24	1	32	12	NA	NA	NA
CT0002	Dep3	NA	64	NA	23	3	43	NA	NA	NA
CT0002	Dep3	NA	45	567	NA	NA	54	NA	NA	NA
CT0002	Dep4	87	NA	56	43	NA	34	NA	NA	NA
CT0002	Dep4	0	0	0	0	NA	0	NA	NA	NA
CT0002	Dep4	12	33	7	55	NA	83	NA	NA	NA
CT0003	Dep5	NA	NA	0	NA	23	NA	47	NA	0
CT0003	Dep5	NA	NA	NA	NA	0	NA	32	213	0
CT0003	Dep2	53	0	NA	NA	0	NA	3	NA	NA
CT0003	Dep2	62	NA	15	NA	34	NA	23	NA	NA
CT0003	Dep2	75	NA	3437	NA	34	NA	63	NA	NA

Project Open Department Closed Project Closed

Figure 3: Example of distinction between missing values and zero consumption.

Outlier Detection Finally, a short note on outlier detection. As many medical supplies are not needed on a regular basis, their demand is intermittent which makes outlier detection difficult. The consumption data for one specific medication can show strong fluctuation in demand, including many very low values. Even so, most of these time series would be marked when applying outlier detection methods, but the time series are reasonable as the related disease occurs irregular. Hence no outlier detection was applied.

3.1.4. Test Case Selection

From the data available, several example cases are used to asses the performance of the different forecasting algorithms. Also their performance in comparison with the judgemental field forecasts can be assessed. In order to have a good basis for the performance assessment, parts of the data are used where errors in the data are more unlikely.

At first, a time period is selected where a large number of time series has no missing values, which were identified in 3.1.3. Then time series are removed, which which have missing values in the selected period or specific statical properties. In the next step, all time series for a single product are grouped together, coming from different projects and department. Each of these sets are then checked for a minimum and maximum number of time series and for a minimal consumption. With this step, a hierarchical structure

is present without going over the computational resources available. For the remaining sets, the hierarchical structure is calculated and examples, which represent the time series classes introduced in 2.1.2 are selected. Finally, each time series is divided in two part, one is used for the parameter estimation, the other to asses the forecasting performance. The used sets are included in the digital appendix.

Time Period Selection In order to prevent erroneous data from influencing the results, only projects and departments which were open for the complete period are used. Not all projects did start or finish at the same time, so taking the maximum time range from the time series table would remove most consumption data. If the period is to small, many data points are lost and the time series may be insufficient for a forecast. Therefore, a suitable period is searched including the most data points.

Assuming a similar weight on the length and number of available time series, the target function is the product of the length of the chosen interval and the number of time series, which have no missing values in the selected interval.

For MSF-OCA this results in the time span from January 2014 until December 2016.

Time Series Exclusion After the best period was selected, several time series are removed. All time series with missing values, only zero consumption, or zero variance are removed. Time series with only zero consumption are products that are so sparsely used that a forecast is not possible because the inter-demand interval is greater then the time period.Variance zero time series have a static continuation of their current value as forecast.Also products in the administrative, logistics and construction groups are removed, as this work focuses on medical supplies.

For MSF-OCA this results in X number of products for testing.How many products where removed in each step is presented in Table 4.

Table 4: Number of products left after cleaning steps.

Cleaning Step	Removed products MSF-OCA	Remaining products MSF-OCA
Original	0	129 312
Missing values	77 200	52 112
< 2 non-zero entries	32 905	19 207
Var $< \epsilon$	39	19 168
Specific Groups	1 734	17 434

Create Product Sets In this step, a set is created for each product, which includes the time series of this product from all projects and departments. Each set can then later be used to apply a hierarchical forecasting algorithm.

After the products are grouped together, the number of each products in a set is calculated. Sets with more than 15 products are excluded to limit the computational effort in the tests. Sets with less than five products are excluded to have a remaining hierarchical structure. Also the overall consumption in the set should be higher than 100. If the overall consumption is too low, either the average inter-demand interval is very large or the consumption is very low.

The number of sets and products which are left is presented in Table 5.

Table 5: Number of products and sets after data cleaning.

Cleaning Step	MSF-OCA Products	
Original	17434	928
< 5 products	16585	500
> 15 products	1585	187
Low Consumption	1564	184

Also in this step, the design matrix S is constructed which is later used. With the help the design matrix, the consumption on higher hierarchical levels is computed and added to the set.

Select Test Cases In section 2.1.2 three different time series classification schemas were given, not including the exact KH and PK models as they need a smoothing factor α for calculation. This work uses the SBC classification. While KH and PK are more accurate when deciding whether to use Croston's method 2.2.1 or the SBA 2.2.2, offers the SBC classification four different classes which can help to asses the performance of the different forecasting algorithms. In particular it is of interest if the performance of the new algorithms are depending on the adi or cv^2 independent from the other. Therefore from the remaining sets, one example for each class sector of the SBC schema is chosen. In order to chose these examples, the average of the coefficient of variations and the average of the average inter-demand intervals are calculated for each set. These examples are later used to test and develop the algorithms and to asses the forecasting performance for different algorithms depending on the demand pattern.

Remark: Figure 13 shows the plot of the average adi against the average cv^2 for the sets of MSF-OCA. As remarked in 1 the strong intermittent demand in humanitarian logistics is clearly visible.

Creating Estimation and Forecast Time Series Finally, the time series are split into two parts. The first part is used to estimate the parameters in the algorithms and the second part is used to asses the forecast performance and compare it with the forecast performance of different algorithms. For MSF-OCA we use the first 24 months for the estimation and last 12 months to compare the forecasting performance.

3.2. Development of the New Forecast Algorithm

3.2.1. Intermittent Integrated Hierarchical Forecast

The advanced model In order to use the prepared data to estimate parameters and assess the forecast performance the advanced demand model is created. The new model is based on the model in Pennings & Van Dalen (2017), which was explained in section 2.3.3, from now on referred to as Integrated Hierarchical Forecasting (IHFC). Following the idea of Croston's method 2.2.1 where demand is modelled as

$$y_t = x_t z_t \tag{103}$$

the measurement equation (102) in the IHFC is modified. The design matrix S is replaced by SX_t where X_t is a diagonal matrix with entries similar defined as in Croston's method.

This results in the following model

$$y_t = SX_t(\mu_t + A\gamma_t) + \epsilon_t, \qquad \epsilon_t \sim N(0, \Sigma_\epsilon) \tag{104}$$

$$\mu_t = \Delta\mu_{t-1} + \beta_t + \eta_t, \qquad \eta_t \sim N(0, \Sigma_\eta) \tag{105}$$

$$\beta_t = \beta_{t-1} + \zeta_t, \qquad \zeta_t \sim N(0, \sigma_\zeta^2 I) \tag{106}$$

$$\gamma_t = \sum_{j=1}^{\lfloor s/2 \rfloor} \gamma_{j,t}, \qquad \lambda_j = \frac{2\pi j}{s} \tag{107}$$

$$\gamma_{j,t} = \gamma_{j,t-1}\cos(\lambda_j) + \gamma^*_{j,t-1}\sin(\lambda_j) + \omega_{j,t}, \qquad \omega_{j,t} \sim N(0, \sigma_\omega^2 I) \tag{108}$$

$$\gamma^*_{j,t} = -\gamma_{j,t-1}\sin(\lambda_j) + \gamma^*_{j,t-1}\cos(\lambda_j) + \omega^*_{j,t}, \qquad \omega^*_{j,t} \sim N(0, \sigma_\omega^2 I) \tag{109}$$

$$\tag{110}$$

Similar to Pennings & Van Dalen (2017) this model includes variant seasonal effects and variant trend. With a non-diagonal Σ_η also cross-correlation between different products are included. The model can be rewritten in BSM form (34)-(35) which then enables the

use of the KF. The state vector α_t is then defined as

$$\alpha_t = (\mu_t, \beta_t, \tilde{\gamma}_t)' \tag{111}$$

where $\tilde{\gamma}_t$ contains $\gamma_{j,t}$ and $\gamma^*_{j,t}$ for $j = 1, \ldots, \lfloor s/2 \rfloor$ for each base product. The measurement matrix Z_t is then given as

$$Z_t = SX_t\tilde{Z}, \tilde{Z} = (I_n, 0_n, \Phi) \tag{112}$$

and Φ is the block-diagonal matrix where the i-th block contains n times ϕ_i. The matrix G for the measurement error is defined as the Cholesky decomposition of the measurement matrix Σ_ϵ

$$GG' = \Sigma_\epsilon. \tag{113}$$

As Σ_ϵ is a positive diagonal matrix, G is also a diagonal matrix where the entries are the square root of the entries in Σ_ϵ.

The transition matrix for the state vector α_t is then given by

$$T = \begin{pmatrix} \Delta & I_n & \\ & I_n & \\ & & \tilde{U} \end{pmatrix}, \tag{114}$$

where $\tilde{U}$ is the block-diagonal matrix which repeats n times the matrix U defined as

$$U = \begin{pmatrix} \cos(\lambda_1) & & & & \sin(\lambda_1) & & & \\ & \cos(\lambda_2) & & & & \sin(\lambda_2) & & \\ & & \ddots & & & & \ddots & \\ & & & \cos(\lambda_{\lfloor s/2 \rfloor}) & & & & \sin(\lambda_{\lfloor s/2 \rfloor}) \\ -\sin(\lambda_1) & & & & \cos(\lambda_1) & & & \\ & -sin(\lambda_2) & & & & \cos(\lambda_2) & & \\ & & \ddots & & & & \ddots & \\ & & & -\sin(\lambda_{\lfloor s/2 \rfloor}) & & & & \cos(\lambda_{\lfloor s/2 \rfloor}) \end{pmatrix}. \tag{115}$$

The modelling matrix H is then defined as

$$H = \begin{pmatrix} \tilde{\Sigma}_\eta & & \\ & \sigma_\zeta I_n & \\ & & \sigma_\omega I_{sn} \end{pmatrix} \begin{pmatrix} I_n & I_n & \\ & I_n & \\ & & I_{sn} \end{pmatrix} \tag{116}$$

with $\tilde{\Sigma}_\eta \tilde{\Sigma}'_\eta = \Sigma_\eta$ is defined again with the Cholesky decomposition.

Setting Σ_η to be diagonal removes the dependencies between different products from the model. Setting Δ to the identity matrix and removing the seasonal component by setting A zero, the model is reduced to a local linear trend model. If Δ is used for β_t instead of μ_t the model is an additive damped trend model. Removing the measurements for the higher hierarchies and setting S as identity matrix results in a non-integrated model. Finally setting X_t as identity matrix for all t removes the adaptation for intermittent demand.

In the parameter estimation X_t is set to the identity matrix and zero consumption measurements are treated as missing values. When applying the KF with a missing measurement the state vector is not adjusted instead the forecast for this entry is taken as measurement. Let $y_{t,i}$ be the i-th entry of the observation vector, if $y_{t,i}$ is missing then the KF update is done with $\tilde{y}_{t,i}$ defined as

$$\tilde{y}_{t,i} = (Z_t T \alpha_{t-1})_i = (S X_t \tilde{Z} T \alpha_{t-1})_i \tag{117}$$

$$v_{t,i} = \tilde{y}_{t,i} - (S X_t \tilde{Z} T \alpha_{t-1})_i = 0. \tag{118}$$

As the innovation $v_{t,i}$ for this entry is 0 the influence in the second term of the log-likelihood for this entry is removed of equation 95. If $v_{t,i} = 0$ then the i-th row and the i-th column of F_t^{-1} in

$$v_t' F_t^{-1} v_t \tag{119}$$

are multiplied with 0. Therefore the influence of the missing value is removed in the log-likelihood term.

The time variant matrix X_t which is needed in the forecasting process, but not for the estimation of the other model parameters, is generated as a random draw from $\{0, 1\}$. Different possibilities for the random distribution and the estimation of their corresponding parameters are discussed next.

3.2.2. Calculation of the Intermittent Influence

To create the diagonal matrix X_t with entries from $\{0, 1\}$ the following methods are possible.

The first, and most simple, method is to draw the entries of X_t for each product independently as a Bernoulli trial with a time invariant probability $\frac{1}{p}$ with p being the adi. This assumes that the occurrence of demand is independent of the past occurrences and independent between different products and that zero consumption is always a result of X_t and not already present in the state vector α_t.

In order to test the assumption that the occurrence of demand is independent of past occurrences we look at the results of the auto-correlation function. For this the available time series where reduced to {0,1}-time series by setting all non-zero values to one. From this the the auto-correlation values are calculated and presented as in boxplot for each lag value h.

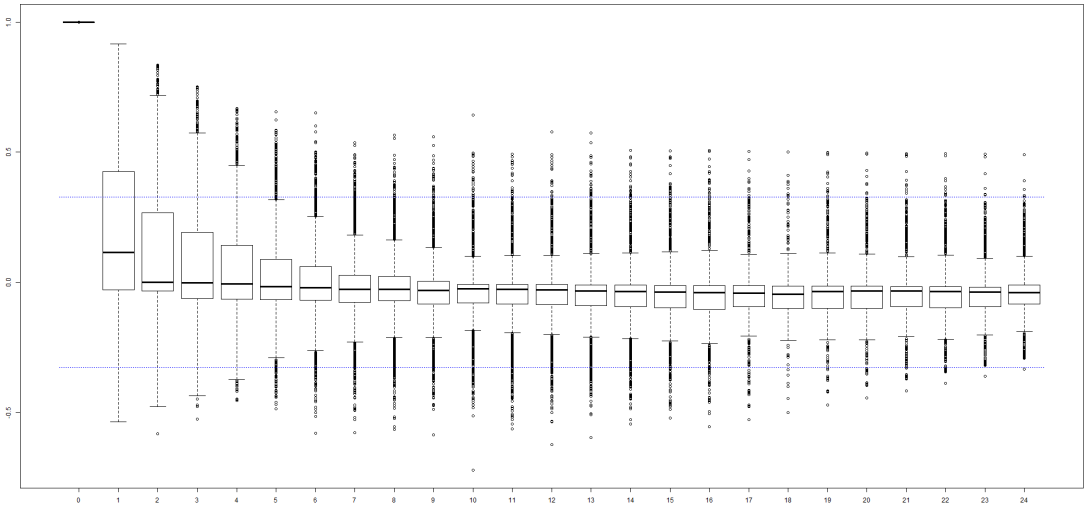

Figure 4: Results of the auto-correlation function for $\sim$ 16000 {0,1}-time series

Figure 4 shows a strong auto-correlation at lag one.

Therefore, the second model to created X_t to include the results from the auto-correlation plot, consist for two dependent probabilities for the Bernoulli trial. If the last demand occurrence was zero the value for X_t is drawn with probability w_0, if the last demand occurrence was non-zero the value is drawn with probability w_1. The prob-

abilities can be calculated with the definition of conditional probabilities.

$$w_0 := P(x_t = 1|x_{t-1} = 0) = \frac{P(x_t = 1 \wedge x_{t-1} = 0)}{P(x_{t-1} = 0)}$$
$$w_1 := P(x_t = 1|x_{t-1} = 1) = \frac{P(x_t = 1 \wedge x_{t-1} = 1)}{P(x_{t-1} = 1)}$$

Where $P(x_{t-1} = 0) := 1 - P(x_{t-1} = 1)$ and $P(x_{t-1} = 1) = 1/\tilde{p}$ with $\tilde{p}$ the average demand interval in the estimation part of the time series.

The third approach follows the idea of Croston's method, where the adi is smoothed each time a non-zero demand occurs. Each diagonal entry in X_t corresponds to a time series of a base product. The smoothed p_t values for these time series can then be created similar to Croston's method during the estimation. The last p_t value is then used for the forecast. This can be further refined by using the SBA to forecast if the time series classification of section 2.1.2 propose this.

The forecasts for all the intermittent methods is then created as average consumption. A h step ahead forecast is then defined as

$$y_{t+h} = S\tilde{X}ZT^h\alpha_t \tag{120}$$

for the first model. Matrix $\tilde{X}$ has as diagonal entries $\frac{1}{p_i}$ where p_i is the adi of the i-th base product. The Matrix $\tilde{X}$ is similar in the third case with p_i either replaced with the last smoothed value of $p_{t,i}$ in Croston's method or with $p_{t,i}c^{p_{t,i}-1}$ for the SBA method. In the second case, where conditional probabilities are used, a large sample is generated from the $(w_{0,i}, w_{1,i})$ pairs. The mean of these samples is then used for the i-th diagonal entry in $\tilde{X}$.

3.2.3. Parameter Estimation

Both models, the one presented in 2.3.3 and the advanced model developed in the last section 3.2.1 require a number of parameters to be estimated.

Parameters The following parameters need to be estimated. For the model itself there are the entries of diagonal matrix A which scales the influence of the seasonal components, this are n parameters. There are the entries of Δ, which are the regression coefficient for the state vector, again n parameters. In addition there is the covariance matrix Σ_η which includes the cross-correlations between the different products in one set. As Σ_η

is symmetric, $n^2/2 + n/2$ parameters are needed. In addition the diagonal covariance matrix of the measurement error is needed, resulting in d parameters. And the variances σ_ζ^2 and σ_ω^2 are needed. This results in $n^2/2 + 7/2n + 2 + d$ parameters.

In addition, to apply the Kalman filter, the initial states are required, as well as the covariance of the initial state vector. For $\alpha_0 = (\mu_0, \beta_0, \gamma_{j,0}, \gamma^*_{j,0})$ this results in $n(s+2)$ parameters. In addition the initial covariance matrix P_0 of the state vector α_0 is needed for the KF Pennings & Van Dalen (2017) used 500 random initialized starts for the BFGS algorithm. As computational resources where limited, different methods are used in this thesis.

The first method uses 200 partly random generated starts of the following structure. Under the assumption that similar products behave similar were 8 random values created for each start. One for the initial state vector α_0, the diagonal of P_0, the diagonal of the covariance matrices Σ_ϵ and Σ_η, for the seasonal scale matrix Φ, the regression coefficient matrix Δ, and for the variances σ_ζ^2 and σ_ω^2. In each vector or matrix the random value is used for all entries, for simplicity are the non-diagonal entries of Σ_η are set to 0. The random values are drawn from a $N(0, 2)$ distribution. Additional for each set of parameters a duplicate was generate where the first part of α_0, namely μ_0 was replaced by the mean of the base product consumption. For each of these 400 parameter sets the log-likelihood was calculated, then the best estimate was taken and further optimised using the Low memory BFGS (L-BFGS) algorithm.

For the second method the parameters were chosen under the initial assumption of a stationary time series without seasonality, without cross-dependencies between different products. The covariance matrices Σ_η and Σ_ϵ where set to the identity matrix, similar the variances σ_ζ^2 and σ_ω^2 were set to 1. The initial state vector α_0 was set to 0 except for the first part μ_0 which was set to the mean of each base product consumption. The regression matrix Δ and scale matrix Φ for seasonal effects was set to the identity matrix. The covariance matrix P_0 was set to 10^7 on the diagonal entries, representing a strong uncertainty in the initial state vector. With this starting parameters again the L-BFGS algorithm was applied.

3.2.4. Implementation Methods

ARIMA, ETS, CRO, SBA implementation The forecasting methods descried in sections 2.1.5-2.2.2 are implemented for univariate time series. Therefore these algorithms neither include the hierarchical structure nor product dependencies.

The hierarchical forecast for these products is generated as bottom-up forecast. The univariate forecasting methods are applied for each base product time series, and the

higher hierarchical levels are then generated with the help of the design matrix S. Additional the reconciliation method presented in Hyndman et al. (2011) was tested but did not result in an improvement.

For the implementation of the ARIMA forecast the methods from the R package forecast were taken. The available auto-arima function provides a automated framework to select the order of the ARIMA model. For this model the required parameters are automatically estimated.

The Exponential Smoothing State Space Model (ETS) implementation, also part of the forecast package and built on Hyndman et al. (2002), provides similar the possibility to automatically select a model and to estimate the required smoothing factors.

In the tsintermittent packages are implementation of Croston's method and the SBA available. These were used to generate the forecasts for these methods. The forecast were generated h steps ahead, without an adjustment to the measurement inbetween.

IHFC and Intermittent Integrated Hierarchical Forecasting (IIHFC) The IHFC and IIHFC were implemented according to the theory in 3.2.1, the utilised code is appended in A.3.1. Starting with the given parameters the function then creates the system matrices. In addition a function which provides the different methods for modelling the adi is added.

3.2.5. Forecast Performance Measure

To measure the performance of different forecast algorithms several measures are available. Some statistical measurements were presented in 2.1.4 and the CPIused by MSF-OCA is described in 2.1.4.

Following Van West (7/5/2016), a work which was build on similar data, the following measurements to asses the forecast performance are used. sMape, sMpe, TS, and CPI. SMape is used as MSF has a good and rich experience with this measure and their data, therefore this measure enables MSF to compare the results from this work with other works based on their data. A similar thing is to say for the CPI which is an intern used measure for MSF. The TS is a important information to show if the the forecast algorithm has the tendency to over or underestimate. SMpe is included to provide a better comparison of the forecasting performance with other publications.

The measures ME, MAE, RMSE and Mean Absolute Squared Error (MASE) are excluded as forecast are compared on time series with different scales, and hence relative measurements are more comparable.

First forecasts were generated with the IHFC and the different version of the IIHFC algorithm. The forecast performance for these algorithm was then evaluated with the different measures chosen in 3.2.5. The IHFC algorithm is used as base line to compare the results of the different version of the IIHFC. Table C.1 presents the average forecast performance including all hierarchical levels when random parameter starts were used in the optimisation step, table C.2 presents the results for the non-random starts. In addition for each set the average cv^2 and adi are included.

It can be seen that the random start performs much worse than the non-random starts. It can be seen that not always one method performs better, but the result is dependent on the data set.

Table C.3 compares the results of the best IIHFC version with the results of the ARIMA forecast, the ETS forecast, Croston's method, and with the results of the SBA forecast.

4. Conclusion and Further Research

4.1. Conclusion

Dependencies between different groups Including dependencies between different products by adding their aggregated and normalised time series did not provide any improvement. The resulting forecast were subject to strong outliers. Therefore this approach is not recommended.

IHFC and IIHFC The IIHFC performed similar to the IHFC, the small differences may stem from the non-optimal parameter estimation and the number of test cases is not significant. The IIHFC offers a working adaptation of Croston's method, and the SBA, to the more flexible BSM used in IHFC. In addition it combines the hierarchical integration with the specific methods for intermittent demand. The same number of parameters like in the IHFC must be estimated to model the demand size, but the number of measurements is decreased as the demand is intermittent and zero consumptions are excluded from the estimation process. In particular for time series with few measurements, or and high adi, this results in a difficult parameter estimation.

IIHFC and simplistic methods In comparison with the naive forecast, the ARIMA, the ETS, Croston's method, and the SBA, the IIHFC performed similar. The IIHFC offers the possibility to include external factors, and know properties of the time series, the disadvantages are the required resources. The IIHFC requires a more advanced understanding of time series model to create the BSM, understanding of the underlying

process, and in particular a large amount of computational power for the parameter estimation.

Practical application Application in practice depends on the available time series data. If only short time series are available, or the adi is large compared to the length of the time series, then the more simplistic forecasting methods have the advantage, as the number of parameters is much lower. In addition the simplistic forecasting methods are preferable if computational resources are limited.

If computational resources are available and the data basis is large enough, then the IIHFC can improve the performance, as product dependencies are included.

4.2. Further Research

Improved parameter selection The main disadvantage, and also challenge in this work, is the increased number of parameters in the IIHFC. Using the more simplistic methods to estimate the initial parameters and type of BSM could save on computational resources. In addition the time series classifications schemas 2.1.2 can be used as automated framework to choose the method in the IIHFC.

Longer time series The current case study does not provide enough evidence for the improvement in the IIHFC. Depending on the number of products in the data set, the length of the time series, and the adi the number of parameters in the IIHFC is higher then the available measurements. Assessing the performance of the IIHFC on a common data set used for intermittent demand could provide further insights.

Product dependencies in the adi The different methods used to model the adi had in common that product dependencies were not considered. Including the dependencies between products not only when estimating the parameters for the demand size model, but also when forecasting the adi could improve the algorithm.

Bibliography

Box, G. E. P., Jenkins, G. M. & Reinsel, G. C. (2008), *Time series analysis: Forecasting and control / George E.P. Box, Gwilym M. Jenkins, Gregory C. Reinsel*, Wiley series in probability and statistics, 4th ed. edn, Wiley, Oxford.
URL: *http://www.loc.gov/catdir/enhancements/fy0805/2007044569-d.html*

Box G. E. P. & TIAO, G. C. (1977), 'A canonical analysis of multiple time series', *Biometrika* **64**(2), 355–365.

Brockwell, P. J. & Davis, R. A. (2016), *Introduction to Time Series and Forecasting*, Springer International Publishing, Cham.

Brown, R. G. (1959), *Statistical forecasting for inventory control Robert G. Brown*, McGraw-Hill, New York [u.a.].

BROYDEN, C. G. (1970), 'The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations', *IMA Journal of Applied Mathematics* **6**(1), 76–90.

Byrd, R. H., Lu, P., Nocedal, J. & Zhu, C. (1995), 'A Limited Memory Algorithm for Bound Constrained Optimization', *SIAM Journal on Scientific Computing* **16**(5), 1190–1208.

Commandeur, J. J. F. & Koopman, S. J. (2007), *An introduction to state space time series analysis*, Practical econometrics series, Oxford University Press, Oxford.

Croston, J. D. (1972), 'Forecasting and Stock Control for Intermittent Demands', *Journal of the Operational Research Society* **23**(3), 289–303.

Dangerfield, B. J. & Morris, J. S. (1992), 'Top-down or bottom-up: Aggregate versus disaggregate extrapolations', *International Journal of Forecasting* **8**(2), 233–241.

Durbin, J. & Koopman, S. J. (2001), *Time series analysis by state space methods*, Vol. 24 of *Oxford statistical science series*, Oxford University Press, Oxford.

Fletcher, R. (1970), 'A new approach to variable metric algorithms', *The Computer Journal* **13**(3), 317–322.

Gardner, E. S. & Mckenzie, E. (1985), 'Forecasting Trends in Time Series', *Management Science* **31**(10), 1237–1246.

Gentle, J. E. (2009), *Computational statistics*, Statistics and computing, Springer, Dordrecht and London.

Goldfarb, D. (1970), 'A family of variable-metric methods derived by variational means', *Mathematics of Computation* **24**(109), 23–26.

Gross, C. W. & Sohl, J. E. (1990), 'Disaggregation methods to expedite product line forecasting', *Journal of Forecasting* **9**(3), 233–254.

Harvey, A. (1989), *Forecasting, structural time series models and the Kalman Filter*, Cambridge University Press.

Hyndman, R. J., Ahmed, R. A., Athanasopoulos, G. & Shang, H. L. (2011), 'Optimal combination forecasts for hierarchical time series', *Computational Statistics & Data Analysis* **55**(9), 2579–2589.

Hyndman, R. J. & Koehler, A. B. (2006), 'Another look at measures of forecast accuracy', *International Journal of Forecasting* **22**(4), 679–688.

Hyndman, R. J., Koehler, A. B., Snyder, R. D. & Grose, S. (2002), 'A state space framework for automatic forecasting using exponential smoothing methods', *International Journal of Forecasting* **18**(3), 439–454.

Johnson, N. L., Kemp, A. W. & Kotz, S. (2005), *Univariate discrete distributions*, Wiley series in probability and statistics, 3rd ed. edn, Wiley-Interscience, Hoboken, N.J.

Johnston, F. R., Boylan, J. E. & Shale, E. A. (2003), 'An examination of the size of orders from customers, their characterisation and the implications for inventory control of slow moving items', *Journal of the Operational Research Society* **54**(8), 833–837.

Kahn, K. B. (1998), 'Revisiting Top-Down Versus Bottom-Up Forecasting', *Journal of Business Forecasting Methods & Systems* **17**(2), 14–19.

Kalman, R. E. (1960), 'A New Approach to Linear Filtering and Prediction Problems', *Journal of Basic Engineering* **82**(1), 35.

Katsnelson, J. & Kotz, S. (1957), 'On the upper limits of some measures of variability', *Archiv für Meteorologie, Geophysik und Bioklimatologie Serie B* **8**(1), 103–107.

Kostenko, A. V. & Hyndman, R. J. (2006), 'A note on the categorization of demand patterns', *Journal of the Operational Research Society* **57**(10), 1256–1257.

Kourentzes, N. (2014), 'Intermittent demand forecasting package for R'.
URL: *http://kourentzes.com/forecasting/2014/06/23/intermittent-demand-forecasting-package-for-r/*

Lukas Rost (n.d.).

Ord, J. K., Koehler, A. B. & Snyder, R. D. (1997), 'Estimation and Prediction for a Class of Dynamic Nonlinear Statistical Models', *Journal of the American Statistical Association* **92**(440), 1621.

Pennings, C. L. & Van Dalen, J. (2017), 'Integrated hierarchical forecasting', *European Journal of Operational Research* .

Petropoulos, F. & Kourentzes, N. (2015), 'Forecast combinations for intermittent demand', *Journal of the Operational Research Society* **66**(6), 914–924.

Shanno, D. F. (1970), 'Conditioning of quasi-Newton methods for function minimization', *Mathematics of Computation* **24**(111), 647–656.

Syntetos, A. A., Boylan, J. E. & Croston, J. D. (2005), 'On the categorization of demand patterns', *Journal of the Operational Research Society* **56**(5), 495–503.

Syntetos, A. & Boylan, J. (2001), 'On the bias of intermittent demand estimates', *International Journal of Production Economics* **71**(1-3), 457–466.

Trigg, D. W. (2017), 'Monitoring a Forecasting System', *Journal of the Operational Research Society* **15**(3), 271–274.

Tu, W. & Mayne, R. W. (2002), 'Studies of multi-start clustering for global optimization', *International Journal for Numerical Methods in Engineering* **53**(9), 2239–2252.

Van West, K. (7/5/2016), The impact of time series clustering on forecast performance: a case study of Médecins Sans Frontières, PhD thesis, Erasmus University Rotterdam.

Widiarta, H., Viswanathan, S. & Piplani, R. (2009), 'Forecasting aggregate demand: An analytical evaluation of top-down versus bottom-up forecasting in a production planning framework', *International Journal of Production Economics* **118**(1), 87–94.

List of Abbreviations

acf auto-correlation function. 4

ARIMA Auto-Regressive Integrated Mean Average. 26–29, 32–34

CPI Consumption Performance Index. 8, 26, 32, 33

Crostons method Crostons method. 5

ETS Exponential Smoothing State Space Model. 26–29, 32–34

IHFC Integrated Hierarchical Forecasting. 26–29, 32–34

IIHFC Intermittent Integrated Hierarchical Forecasting. 2, 26–29, 32–34

JFC Judgemental Forecast. 26, 32, 33

KH Kostenko-Hyndman. 2, 5, 6

KHa Kostenko-Hyndman approximate. 2, 5, 6

MAE Mean Absolute Error. 6, 7, 26

Mape Mean absolute percentage error. 7

MASE Mean Absolute Squared Error. 26

ME Mean Error. 6, 7, 26

Mpe Mean percentage error. 7

MSF Médecins Sans Frontières. 8, 9, 26, 28

MSF-OCA Operational Center of Amsterdam of Médecins Sans Frontières. 2, 9–12, 14–16, 26, 28, 30

MSF-SO Similar Organisation of Médecins Sans Frontières. 2, 9–12, 15, 16, 30

PK Petropoulos-Kourentzes. 2, 5, 6

PKa Petropoulos-Kourentzes approximate. 2, 5, 6

RMSE Root Squared Error. 6, 7, 26

SBA Syntetos-Boylan approximation. 5

SBC Syntetos-Boylan-Croston. 2, 5

SES Single Exponential Smoothing. 5

SKU Stock Keeping Unit. 10

sMape symmetric Mean absolute percentage error. 7, 26, 32, 33

sMpe symmetric Mean percentage error. 7, 26, 32, 33

SQL Structured Query Language, database language. 9

TS Tracking Signal. 8, 26, 32, 33

Appendices

A. R Code

A.1. Execution Code

```
require(data.table)
##############################################################################
# Describes the process applied
##############################################################################

##############################################################################
# Loading the required functions
source("./Appendix/findStartEndBreak.R")
source("./Appendix/selectTimeSeries.R")
source("./Appendix/generateSets.R")

source("./Appendix/createSSTModel.R")
source("./Appendix/call.fkf.R")
source("./Appendix/runOptim.R")
source("./Appendix/createParameters.R")
source("./Appendix/forecasts.R")
source("./Appendix/measures.R")

source("./Appendix/createTablesAndForecasts.R")
##############################################################################
# Loading the original time series table
load("../Data/Appendix/timeSeriesTable.RData")

```

```
#############################################################################
# Identifying missing values
cleanedTimeSeries <- findStartEndBreak(timeSeriesTable)

# Select a suitable range of time series
selectedTimeSeries <- selectTimeSeries(cleanedTimeSeries)

# Create data sets of time series
setList <- generateSets(selectedTimeSeries)

# Create additional properties for each set
fullSetList <- lapply(setList, generateSetProperties, timeSeriesTable =
    selectedTimeSeries)

# Names of optimised sets, only a few set were optimised
optimSetNames <- c("ABAAJBAAIFA", "ADAARAAAHAA", "ADAAAFAAHAA", "
    AEAAYIAAZDA")

# Select these sets from the list
setSubList <- fullSetList[sapply(fullSetList, function(set) { set$name}) %
    in% optimSetNames]

# Run optimisation with static initial parameters
optimisedStaticInitSetList <- lapply(setSubList, function(set) {
  set <- createStaticParameter(set)
  set <- runOptim(set)
  return(set)
})
# The results are also available from
load("../Data/Appendix/optimisedStaticInitSetList.RData")

# Run optimisation with random initial parameters
optimisedRandomInitSetList <- lapply(setSubList, function(set) {
  set <- createRandomParameter(set, 200)
  set <- runOptim(set)
  return(set)
})
# The results are also available from
load("../Data/Appendix/optimisedRandomInitSetList.RData")

#############################################################################
# First Test
# Compare forecast performance for different initalisiation methods
```

```r
staticInitIHFCTable <- do.call(rbind, lapply(optimisedStaticInitSetList,
    createIHFCTableEntry))
randomInitIHFCTable <- do.call(rbind, lapply(optimisedRandomInitSetList,
    createIHFCTableEntry))
# The conclusion here is that the static init performs much better
##############################################################################

##############################################################################
# Second Test
# Compare forecast performance of the IHFC-Group with simpler methods
staticInitARTable <- rbind(
  createARTableEntry(optimisedStaticInitSetList[[1]], iihfc_method = "NO"),
  createARTableEntry(optimisedStaticInitSetList[[2]], iihfc_method = "
    STATIC"),
  createARTableEntry(optimisedStaticInitSetList[[3]], iihfc_method = "CRO")
    ,
  createARTableEntry(optimisedStaticInitSetList[[4]], iihfc_method = "SBA")
)
##############################################################################

##############################################################################
# Third Test
# Try to include product dependencies
staticInitIHFCTableDEP <- do.call(rbind, lapply(optimisedStaticInitSetList,
     createIHFCTableEntry, dep = TRUE))
##############################################################################
```

Listing 1: Executed Code

A.2. Data Cleaning and Set Generation

A.2.1. Identifying Missing Values

```r
require(zoo)
require(data.table)

# This is necessary to correctly identify the month names
Sys.setlocale("LC_TIME", "C")

##############################################################################
# cleanedTimeSeriesTable <- findStartEndBreak(timeSeriesTable)
#
# The function identifies missing values in a given time series table
##############################################################################
```

```r
findStartEndBreak <- function(timeSeriesTable) {
  # Function to identify a block, which includes includeColl, spanning all
    time series in dataTable where values are either missing or -Inf
  getMaxBlock <- function(dataTable, includeCol) {
    # Get the indices of the columns fullfilling the cirteria
    matchedCols <- which(sapply(dataTable, function(col) {
      return(all(is.na(col) | col == -Inf))
    }))
    # Check if the required column fullfills the criteria
    if(includeCol %in% matchedCols) {
      # Get all columns left (and right) from includeCol, which fullfill
    the criteria
      leftCols <- matchedCols[matchedCols <= includeCol]
      rightCols <- matchedCols[matchedCols >= includeCol]

      # Now selecting the largest contiuous block on the left (right)
      leftNACols <- leftCols[max(which(diff(leftCols) != 1) + 1, 1):length(
    leftCols)]
      rightNACols <- rightCols[1:(min(which(diff(rightCols) != 1), length(
    rightCols)))]
      # These blocks are combined, duplicated column indices are removed
      # and the indices are returned
      return(unique(c(leftNACols, rightNACols)))

    } else {
    # If includeCol does not fulfill the criteria, no column index is
    returned
      return(NULL)
    }
  }

  # Function to identify opening and closing departments in the consumption
     of a single project
  findDepartmentStartEnd <- function(project) {
    # Identify the first column index
    firstCol <- first(which(!is.na(as.yearmon(colnames(project)))))
    # Extract the department codes present in this project
    depCode <- levels(droplevels(project$Department))
    # Get List of the time series by department
    depList <- lapply(depCode, function(dep) { project[project$Department
    == dep, ]})

    # Function to change the columns for time series consumption in each
    department
```

```
48      changeDepBlock <- function(dep) {
          # Selects columns which fullfill the requirement and are at the
        beginnging of the time series
50        startBlock <- getMaxBlock(dep, firstCol)
          # Selects columns at the end
52        endBlock <- getMaxBlock(dep, dim(dep)[[2]])
          # Changes the entry in the columns from NA to -Inf
54        if(length(startBlock) > 0) {
            dep[, startBlock] <- -Inf
56        }
          if(length(endBlock) > 0) {
58          dep[, endBlock] <- -Inf
          }
60
          # Return the changed time series for this department
62        return(dep)
        }
64
        # Changes all departments in this project
66      project <- rbindlist(lapply(depList, changeDepBlock))
        # Return the modified project
68      return(project)
      }
70
      # Function to identify missing values in a complete project
72    setProjectCol <- function(project) {
        # Selecting all columns which fullfill the criteria complete
74      NAColumns <- which(sapply(project, function(col) { all(is.na(col) | col
        == -Inf)}))
        # Set these columns to -Inf
76      project[,NAColumns] <- -Inf
        # Return the modified project
78      return(project)
      }
80
      # Create list of consumption time series by project
82    projectList <- lapply(levels(timeSeriesTable$Project), function(arg) {
        return(timeSeriesTable[timeSeriesTable$Project == arg,])} )

84    # Modify each project to identify missing columns
      projectList <- lapply(projectList, setProjectCol)
86
      # Modify each project to identify the opening and closing of the
        departments
```

```
88    projectList <- lapply(projectList, findDepartmentStartEnd)

90    # Create a modified time series table
      cleanedTimeSeriesTable <- rbindlist(projectList)
92
      # All identified missing values were set to -Inf
94    # Now set all remaing NAs to 0, and the -Inf to NA
      cleanedTimeSeriesTable[is.na(cleanedTimeSeriesTable)] <- 0
96    cleanedTimeSeriesTable[cleanedTimeSeriesTable == -Inf] <- NA

98    # Return the modified time series Table
      return(cleanedTimeSeriesTable)
100 }
```

Listing 2: Code to detect missing Values

A.2.2. Time Period Selection

```
require(data.table)
2
################################################################################
4 # selectedTimeSeriesTable<- findStartEndBreak(timeSeriesTable)
  #
6 # The function identifies the best subset for testing purposes
################################################################################
8 selectTimeSeries <- function(timeSeriesTable){
    # Count the number of values in each column
10  numOfValues <- apply(timeSeriesTable[, -c(1:6), with = FALSE], 2,
      function(arg) { sum(!is.na(arg)) })
    # Create an empty matrix to save the number of series for each selection
      of months
12  numMatrix <- data.frame(start = NA, end = NA, value = NA)
    # For each selection of months, the length of the time period, time the
      number of time series without missing values is calculated
14  for(end in length(numOfValues):36) {
      for(start in 1:(end-35)){
16      numMatrix <- rbind(numMatrix, data.frame(start = start, end = end,
      value = min(numOfValues[start:end]) * (end - start + 1)))
      }
18  }
    # Any entry where no value was calculate is removed
20  numMatrix <- numMatrix[!is.na(numMatrix$value),]
    # The optimal selection of months is extracted
```

```
22    range <- numMatrix[numMatrix$value == max(numMatrix$value, na.rm = TRUE),
          1:2]
      start <- range[[1]]
24    end <- range[[2]]

26    # The non selected months are removed
      selectedTimeSeries <- timeSeriesTable[, c(1:6, (start:end) + 6), with =
        FALSE]
28
      # Time series with missing values in the remaining time period are
        removed
30    selectedTimeSeries <- selectedTimeSeries[apply(selectedTimeSeries[, -c
        (1:6), with = FALSE], 1, function(line) { !any(is.na(line)) }), ]

32    # All time series with less then two values in the first two third of the
         time are removed
      selectedTimeSeries <- selectedTimeSeries[apply(selectedTimeSeries[, -c
        (1:6), with = FALSE], 1, function(line) { sum(line[1:round(2*length(
        line)/3, 0)] != 0) > 2}), ]
34
      # All time series with a variance lower than 1/(2*n) are removed
36    varianceVec <- apply(selectedTimeSeries[, -c(1:6), with = FALSE], 1, var)
      selectedTimeSeries <- selectedTimeSeries[varianceVec > 1/(2*(dim(
        selectedTimeSeries)[[2]]-6)),]
38
      # Remove non medical product groups
40    selectedTimeSeries <- selectedTimeSeries[ Group %in% c("A", "D"),]

42    # Return the selected time series
      return(selectedTimeSeries)
44  }
```

Listing 3: Selecting a specific time frame

A.2.3. Set Generation

```
  require(data.table)
2
  ##############################################################################
4 # setList <- generateSets(timeSeriesTable)
  #
6 # Generates sets for each products and removes sets
  ##############################################################################
8 generateSets <- function(timeSeriesTable) {
```

```
  # Remove levels left from the previous steps
  timeSeriesTable <- droplevels(timeSeriesTable)
  # Create the list of sets by product
  setList <- split(timeSeriesTable, by=c("Group", "Family", "Root", "
    Product"), flatten = TRUE, drop = TRUE)

  # Remove all sets with less then 5 time series
  nolowerFive <- sapply(setList, function(set) { dim(set)[[1]]}) >= 5
  setList <- setList[nolowerFive]

  # Remove all sets with more than 15 time series
  noHigher15 <- sapply(setList, function(set) { dim(set)[[1]]}) <= 15
  setList <- setList[noHigher15]

  # Return a list of time series tables
  return(setList)
}

##############################################################################
# fullSet <- generateSetProperties(set, timeSeriesTable)
#
# Generates a more complex set from the simple time series, including
# different data sets for different optimisation approaches. each data set
# includes estimation data and forecast data for the base products and the
# hierarchical structure, the four data sets are
# - sim:    simple, the original set
# - na:     zeros are set to NA
# - simDep: like sim, but with aggregated consumption for product
#           categories
# - naDep:  like na, but with aggregated consumption for product categories
##############################################################################
generateSetProperties <- function(set, timeSeriesTable) {
  # Generates the product information for the hierarchical consumption
  generateFullProductInfo <- function(baseProductInfo) {
    # Line for complete aggregation
    topMatrix <- baseProductInfo[, lapply(.SD, function(arg){return(arg
    [[1]])}),by = .(), .SDcol = c("Group", "Family", "Root", "Product")]
    topMatrix[, c("Department", "Project")] <- NA
    topMatrix <- topMatrix[, c(6:5,1:4)]
    # Line for aggregation on project level
    projectMatrix <- baseProductInfo[, lapply(.SD, function(arg){return(arg
    [[1]])}),by = .(Project), .SDcol = c("Group", "Family", "Root", "
    Product")]
    projectMatrix[, "Department"] <- NA
```

```
50      # Combined product information
        fullInfoMatrix <- rbind(topMatrix, projectMatrix, baseProductInfo)
52
      # Return the information
54      return(fullInfoMatrix)
    }
56
  # Generates the design matrix S, according to the information about the
    base products
58  generateDesignS <- function(baseProductInfo) {
      # Summation matrix for complete aggregation
60    topMatrix <- matrix(1, ncol = dim(baseProductInfo)[[1]], nrow = 1)

62    # List of projects in this set
      projectList <- unique(baseProductInfo$Project)
64    # Aggregation matrix for project leve
      projectMatrix <- t(sapply(projectList, function(arg) {
66      return(as.numeric(baseProductInfo$Project == arg))
      } ))
68    # Matrix to keep the original time series
      bottomMatrix <- diag(1, dim(baseProductInfo)[[1]])
70
      # Create and return the desing matrix
72    S <- rbind(topMatrix, projectMatrix, bottomMatrix)
      return(S)
74  }

76  # Define variable for the new set
    newSet <- list()
78  # Extract the base product information, add it to the set
    newSet$baseProductInfo <- set[,c("Project", "Department", "Group", "
      Family", "Root", "Product")]
80  # Create the original design matrix
    S <- generateDesignS(newSet$baseProductInfo)
82  # Create the design matrix if aggregation on product categories was added
    dep_S <- rbind(cbind(diag(1, 3), matrix(1, ncol = dim(S)[[2]], nrow = 3))
      ,
84                 cbind(matrix(0, ncol = 3, nrow = (dim(S)[[1]])), S))
    # Create the product information including the hierarchy
86  newSet$fullProductInfo <- generateFullProductInfo(newSet$baseProductInfo)
    # Create a name for the set, choosen by the product
88  newSet$name <- paste0(lapply(set[1, c("Group", "Family", "Root", "Product
      ")], as.character), collapse = "")
```

```
90  # Extract the data part from the set
    fullData <- set[, -c(1:6), with = FALSE]
92  # Get the complete consumption in the set, in order to scale the
      consumption aggregated over the product hierarchy
    setSum <- sum(fullData, na.rm = TRUE)
94  # Get the sum of consumption in each column, in order to remove this from
       the aggregated consumption
    setColSum <- colSums(fullData, na.rm = TRUE)
96  # Define the length of the estimation data, and the length of the
      forecast performance data
    newSet$EstLength <- round(2*dim(fullData)[[2]]/3, 0)
98  newSet$FCLength <- dim(fullData)[[2]] - newSet$EstLength

100 # Create the sim data set
    newSet$sim <- list()
102 newSet$sim$baseProductEstData <- as.matrix(fullData[, 1:newSet$EstLength
      ])
    newSet$sim$baseProductFCData <- as.matrix(fullData[, (newSet$EstLength+1)
      :dim(fullData)[[2]]])
104 newSet$sim$fullEstData <- S %*% newSet$sim$baseProductEstData
    newSet$sim$fullFCData <- S %*% newSet$sim$baseProductFCData
106 newSet$sim$S <- S

108 # Create the na data set
    newSet$na <- list()
110 newSet$na$baseProductEstData <- newSet$sim$baseProductEstData
    newSet$na$baseProductEstData[newSet$na$baseProductEstData == 0] <- NA
112 newSet$na$baseProductFCData <- newSet$sim$baseProductFCData
    newSet$na$baseProductFCData[newSet$na$baseProductFCData == 0] <- NA
114 newSet$na$fullEstData <- newSet$sim$fullEstData
    newSet$na$fullEstData[newSet$na$fullEstData == 0 ] <- NA
116 newSet$na$fullFCData <- newSet$sim$fullFCData
    newSet$na$fullFCData[newSet$na$fullFCData == 0]<- NA
118 newSet$na$S <- S

120 # Extract the Product category names
    rootName <- as.character(newSet$baseProductInfo$Root[[1]])
122 famName <- as.character(newSet$baseProductInfo$Family[[1]])
    grpName <- as.character(newSet$baseProductInfo$Group[[1]])
124 proName <- as.character(newSet$baseProductInfo$Product[[1]])

126 # Create aggregated consumption for the product hierarchy, then scaled to
       the overall consumption in the set
```

```
  # For the Root category
128 rootInfo <- data.frame(Project = NA, Department = NA, Group = grpName,
    Family = famName, Root = rootName, Product = NA)
  fullRootTS <- colSums(timeSeriesTable[Group == grpName & Family ==
    famName & Root == rootName, -c(1:6), with = FALSE])
130 fullRootTS <- fullRootTS / sum(fullRootTS, na.rm = TRUE) * setSum

132 diffRootTS <- fullRootTS - setColSum

134 # For the Family category
  famInfo <- data.frame(Project = NA, Department = NA, Group = grpName,
    Family = famName, Root = NA, Product = NA)
136 fullFamTS <- colSums(timeSeriesTable[Group == grpName & Family == famName
    , -c(1:6), with = FALSE])
  fullFamTS <- fullFamTS / sum(fullFamTS, na.rm = TRUE) * setSum
138 diffFamTS <- fullFamTS - setColSum

140 # For the Group category
  grpInfo <- data.frame(Project = NA, Department = NA, Group = grpName,
    Family = NA, Root = NA, Product = NA)
142 fullGrpTS <- colSums(timeSeriesTable[Group == grpName, -c(1:6), with =
    FALSE])
  fullGrpTS <- fullGrpTS / sum(fullGrpTS, na.rm = TRUE) * setSum
144 diffGrpTS <- fullGrpTS - setColSum

146 # The product info is then updated for these sets
  newSet$dep_baseProductInfo <- rbind(grpInfo, famInfo, rootInfo, set$
    baseProductInfo)
148 newSet$dep_fullProductInfo <- rbind(grpInfo, famInfo, rootInfo, set$
    fullProductInfo)

150 # Convienience
  estLength <- newSet$EstLength
152 FCLength <- newSet$FCLength

154 # Create the simDep case
  newSet$simDep <- list()
156 newSet$simDep$baseProductEstData <- rbind(diffGrpTS[1:estLength],
    diffFamTS[1:estLength], diffRootTS[1:estLength], newSet$sim$
    baseProductEstData)
  newSet$simDep$baseProductFCData <- rbind(diffGrpTS[estLength+1:FCLength],
     diffFamTS[estLength+1:FCLength], diffRootTS[estLength+1:FCLength],
    newSet$sim$baseProductFCData)
158 newSet$simDep$fullEstData <- dep_S %*% newSet$simDep$baseProductEstData
```

```
  newSet$simDep$fullFCData <- dep_S %*% newSet$simDep$baseProductFCData
  newSet$simDep$S <- dep_S

  # Create the naDep case
  newSet$naDep$baseProductEstData <- newSet$simDep$baseProductEstData
  newSet$naDep$baseProductEstData[newSet$naDep$baseProductEstData == 0] <-
    NA
  newSet$naDep$baseProductFCData <- newSet$simDep$baseProductFCData
  newSet$naDep$baseProductFCData[newSet$naDep$baseProductFCData == 0] <- NA
  newSet$naDep$fullEstData <- newSet$simDep$fullEstData
  newSet$naDep$fullEstData[newSet$naDep$fullEstData == 0 ] <- NA
  newSet$naDep$fullFCData <- newSet$simDep$fullFCData
  newSet$naDep$fullFCData[newSet$naDep$fullFCData == 0]<- NA
  newSet$naDep$S <- dep_S

  # Extract the column names for the two dat aparts
  newSet$estNames <- colnames(newSet$baseProductEstData)
  newSet$FCNames <- colnames(newSet$baseProductFCData)

  # Function to return a config depending on the choosen data set
  newSet$Config <- function(data) {
    config <- list()
    config$s <- 12
    config$n <- dim(data$baseProductEstData)[[1]]
    config$d <- dim(data$fullEstData)[[1]]
    config$m <- config$n *( config$s + 2)
    config$S <- data$S
    return(config)
  }

  # Return the new set
  return(newSet)
}
```

Listing 4: Creating Sets from the time series table

A.3. Tools to generate Forecasts

A.3.1. IIHFC Model Generation

```
require(tsintermittent)
require(Matrix)
require(expm)
##########################################################################################
```

```r
# SSTModel <- createSSTModel(paraMeters, config, yt)
#
# The function create a state space model with regression coefficient in
# the trend component, and dampened trigonometric modelled seasonality,
# and different methods to include intermittent demand
# The paraMeters include the following information
# - a0:   The inital state vector
# - P0:   The variance of the a0, diagonal matrix
# - T:    The transition matrix for the state vector, including the
#         regression coefficients
# - H:    The modeling error matrix, non-diagonal
# - Z:    The measurement matrix, including the scaling factors for
#         seasonal effects
# - G:    The measurement error matrix, diagonal
# The config includes information created by the Config function of the
# sets, this includes
# - n:    The number of base products
# - d:    The number of products including the hierarchy
# - m:    The length of the state vector a_t, here n *( s+ 2)
# - s:    The length of one season, here 12
# - S:    The design matrix for the hierarchy
# The data set yt includes the consumption of the full products, it is used
# to implement the different methods to model intermittent demad
##############################################################################

createSSTModel <- function(paraMeters, config, yt) {
  # Function to create the system matrices
  create.cov <- function(paraMeters, check = "") {
    # The first part of paraMeters are the standard deviations, the second
    part is the lower triangular part of the correlation matrix, column by
    column
    # Get the dimension
    n <- sqrt(2 * length(paraMeters) + 1/4) - 1/2
    if(n != round(n, 0)) {
      warning("Number of paraMeters for the covariance matrix is wrong (
    create.cov).")
      return(-1)
    }

    cor <- matrix(0, ncol = n, nrow = n)
    cor[lower.tri(cor, diag = FALSE)] <- paraMeters[n + 1:(n*(n-1)/2)]
    cor <- cor + diag(1, n) + t(cor)
    D <- diag(paraMeters[1:n])
    # Check if the result is singular
```

```
    if(check != "") {
47    if(round(det(cor), 10) == 0) {
        warning("Singular Covariance Matrix: ", check)
49    }
    }
51  return(D %*% cor %*% D)
  }
53  create.T <- function(paraMeters) {
    if(length(paraMeters) != config$n) {
55    warning("Wrong number of paraMeters in create.T")
    }
57  if(config$s/2 != round(config$s/2,0)){
    warning("Seasonality is not even, reimplement...")
59  }
    U <- rbind(
61    cbind(diag(cos((1:(config$s / 2)) * 2 * pi / config$s)), diag(sin
    ((1:(config$s / 2)) * 2 * pi / config$s))),
      cbind(diag(-sin((1:(config$s / 2)) * 2 * pi / config$s)), diag(cos
    ((1:(config$s / 2)) * 2 * pi / config$s)))
63  )
    # First part of T
65  T_1 <- rbind(
      cbind(diag(paraMeters), diag(1, config$n)),
67    cbind(matrix(0, ncol = config$n, nrow = config$n), diag(1, config$n))
    )
69  # Create T complete
    T_ <- bdiag(T_1, diag(1, config$n) %x% U)
71  return(array(T_, dim = c(config$m, config$m, 1)))
  }
73  create.HH <- function(paraMeters) {
    # HH is HH' as specified in the FKF package
75  if(length(paraMeters) != config$n * (config$n + 1) / 2 + 2) {
      warning("Wrong number of paraMeters in create.HH")
77  }
    # Create the first and second part of H
79  H_2 <- rbind( cbind(diag(1, config$n), diag(1, config$n), matrix(0,
    nrow = config$n, ncol = config$s * config$n)),
                  cbind(matrix(0, nrow = config$n, ncol = config$n), diag
    (1, config$n), matrix(0, nrow = config$n, ncol = config$s * config$n)),
81                cbind(matrix(0, nrow = config$s * config$n, ncol = 2 *
    config$n), diag(1, config$s * config$n)))
    # Include thjce covariane matrix
83  HH <- H_2 %*% bdiag(create.cov(paraMeters[1:(config$n*(config$n+1)/2)],
     check = "HHT"), paraMeters[[config$n*(config$n+1)/2 + 1]]*diag(1,
```

```
    config$n), paraMeters[[config$n*(config$n+1)/2+2]]*diag(1, config$s*
    config$n) ) %*% t(H_2)
    return(array(HH, dim = c(config$m, config$m, 1)))
  }
  create.Z_2 <- function(paraMeters) {
    if(length(paraMeters) != config$n) {
      warning("Wrong number of paraMeters in create.Z_2")
    }
    Z_2 <- cbind(diag(1, config$n), matrix(0, ncol = config$n, nrow =
    config$n), diag(paraMeters) %x% t(c(rep(1, config$s/2), rep(0, config$s
    /2))) )

    return(Z_2)
  }
  create.GG <- function(paraMeters) {
    # Measurement errors are uncorrelated therefore GG is a diagonal matrix
    d <- length(paraMeters)
    return(array(diag(paraMeters), dim = c(d, d, 1)))
  }

  # For convienience
  d <- config$d
  n <- config$n
  m <- config$m
  # Check if the number of paraMeters matches the required config
  if(length(paraMeters) != (2*m+3*n+n*(n-1)/2+2+d)) {
    warning("Wrong number of paraMeters in create.model")
    return(-1)
  }
  # Initalize the model
  model <- list()
  # Create the model matrices and vectors from the inital parameter vector
  # The inital state vector, and it's initial variance (desribing the
    uncertainty)
  model$a0 <- paraMeters[1:m]
  model$P0 <- diag(paraMeters[m + 1:m])

  # Create the transition matrix for the state vector,
  model$Tt <- create.T(paraMeters[2*m + 1:n])
  # the matrix for the modeling error,
  model$HHt <- create.HH(paraMeters[2*m+n + 1:(n+n*(n-1)/2 + 2)])
  # the second part of the measurement matrix, and
  model$Z_2 <- create.Z_2(paraMeters[(2*m+2*n+n*(n-1)/2+2) + 1:n])
  # the matrix for the measurement error
```

```
123   model$GGt <- create.GG(paraMeters[(2*m+3*n+n*(n-1)/2+2) + 1:d])

125   # The current estimate of the state vector including the uncertainty
      model$at <- model$a0
127   model$Pt <- model$P0

129   # In addition the config variables are added to the model
      model$S <- config$S
131   model$d <- config$d
      model$s <- config$s
133   model$n <- config$n
      model$m <- config$m
135
      # If data was provided different inttermittent models can be created from
         this
137   if(!missing(yt) && !is.null(yt)) {
        # select the consumption for the base products
139     xt <- yt[(model$d-model$n)+1:model$n,]
        # Conditional probabilities for demand occurences
141     w_0 <- apply(xt, 1, function(arg) {
          # Select all which fullfill the condition
143       newArg <- arg[shift(arg,1) == 0]
          # Calculate the probability for a non-zero demand
145       return(sum(newArg != 0, na.rm = TRUE) / (length(newArg) - 1))
        })
147     w_1 <- apply(xt, 1, function(arg) {
          # Select all which fullfill the condition
149       newArg <- arg[shift(arg,1) != 0]
          # Calculate the probability for a non-zero demand
151       return(sum(newArg != 0, na.rm = TRUE) / (length(newArg) - 1))
        })
153
        # If data is given, calculate the average inter-demand interval for the
         base products
155     model$p <- apply(xt, 1, function(arg) { return(idclass(arg, type = "SBC
        ", outplot = FALSE)$p) })

157     # Create vectors creating the diagonal entries of X for all methods
        # For a static inter demand interval
159     model$XVal_Stat <- 1/model$p
        # With conditional probabilities
161     model$XVal_Stat2 <- sapply(1:model$n, function(index) {
          # For each time series select the right probabilities
163       W0 <- w_0[index]
```

```
      W1 <- w_1[index]
      # Select the last value, if non is given,
      lastVal <- xt[index, dim(xt)[[2]]]
      if(is.na(lastVal)) {
        lastVal <- 0
      }
      # Generate 1000 random draws
      xsVal <- rep(NA, 1000)
      for(i in 1:1000) {
        if(lastVal == 0) {
          xsVal[i] <- rbinom(1, 1, W0)
          lastVal <- xsVal[i]
        } else {
          xsVal[i] <- rbinom(1, 1, W1)
          lastVal <- xsVal[i]
        }
      }
      # Return the mean of the random draws
      return(mean(xsVal, na.rm=T))
    })

    # Applying Croston's method and extracting the forecasted inter-demand
    interall
    xt[is.na(xt)] <- 0
    model$XVal_Cro <- apply(xt, 1, function(baseTS) { return(1/crost(baseTS
    , outplot = FALSE)$components$c.out[1,"Interval"])})
    # Applying the SBA
    model$XVal_SBA <- apply(xt, 1, function(baseTS) { return(1/crost(baseTS
    , type = "SBA", outplot = FALSE)$components$c.out[1,"Interval"])})

    # Create the function
    model$Xt <- function(pmodel = "STATIC") {
      if(pmodel == "STATIC") {
        xVal <- model$XVal_Stat
      } else if(pmodel == "STATIC2") {
        xVal <- model$XVal_Stat2
      } else if(pmodel == "CRO") {
        xVal <- model$XVal_Cro
      } else if(pmodel == "SBA") {
        xVal <- model$XVal_SBA
      } else {
        warning("Model is not know, choose non-intermittent model")
        xVal <- rep(1, model$n)
      }
```

```
      Xt <- diag(xVal)

      return(Xt)
    }
  } else {
    # if no data was provided, X is set to the identity matrix for all
    models,
    model$Xt <- function(method) {
      if(missing(method)) {
        # A warning is added if the method was specified, but no data was
    given
        warning("No data for intermittent model was given")
      }
      return(diag(1, model$n))
    }
  }

  # Return the model
  return(model)
}
```

Listing 5: Create State Space model

A.3.2. Applying the Kalman Filter

```
require(FKF)
##############################################################################
# updatedModel <- call.fkf(model, yt)
#
# Function applies the Kalman filter to update the state vector, returned
# is the model with updated state vector at, state vector covariance Pt,
# and updated log-likelihood
##############################################################################
call.fkf <- function(model, yt) {
  # Apply the Kalman filter
  capture.output(result <- fkf(a0 = model$at, P0 = model$Pt, dt = matrix(0,
     nrow = model$m, ncol = 1), ct = matrix(0, nrow = model$d, ncol = 1),
    Tt = model$Tt, Zt = array(model$S %*% model$Z_2, dim = c(model$d, model
    $m, 1)), HHt = model$HHt, GGt = model$GGt, yt = yt))
  # Save the results
  model$logLik <- result$logLik
  o <- dim(result$at)[[2]]
  model$at <- array(result$at[,o], dim = c(length(model$at)))
  model$Pt <- array(result$Pt[,,o], dim = c(dim(model$Pt)))
```

```
  # Return the updated model
  return(model)
}
```

Listing 6: Apply the Kalman filter

A.3.3. Parameter Generation

```
source("./Appendix/runOptim.R")
##############################################################################
# setWithPar <- createRandomParameter(set, n)
#
# Generates a random set of initial parameters for a set
##############################################################################
createRandomParameter <- function(set, n) {
  # Create the random initalised values
  par <- matrix(rnorm(n*8, 0, 2), ncol = 8, nrow = n)
  # Functions to generate a parameter vector from a few random values
  createInitFromRandom <- function(rV, config) {
    par <- c(rep(rV[1], config$m), rep(rV[2]^2, config$m), rep(rV[3],
    config$n), rep(rV[4]^2, config$n), rep(0, config$n*(config$n-1)/2), rV
    [5:6], rep(rV[7], config$n), rep(rV[8]^2, config$d))
    return(par)
  }
  createInitFromRandom2 <- function(rV, config, data) {
    par <- c(apply(data$baseProductEstData, 1, mean, na.rm=TRUE), rep(rV
    [1], config$m - config$n), rep(rV[2]^2, config$m), rep(rV[3], config
    $n), rep(rV[4]^2, config$n), rep(0, config$n*(config$n-1)/2), rV[5:6],
    rep(rV[7], config$n), rep(rV[8]^2, config$d))
    return(par)
  }

  # Evaluate which of the random generate values performs best
  bestPar <- function(data) {
    config <- set$Config(data)
    result1 <- apply(par, 1, function(arg) { logLikP(createInitFromRandom(
    arg, config), config, data)})
    result2 <- apply(par, 1, function(arg) { logLikP(createInitFromRandom2(
    arg, config, data), config, data)})
    if(max(result1) > max(result2)) {
      return(createInitFromRandom(par[which(max(result1) == result1), 1:8],
     config))
    } else {
```

```
      return(createInitFromRandom(par[which(max(result2) == result2), 1:8],
     config))
    }
  }

  # Generate a initial parameter set for each data set
  set$par <- list()
  set$par$simInit <- bestPar(set$sim)
  set$par$naInit <- bestPar(set$na)
  set$par$simDepInit <- bestPar(set$simDep)
  set$par$naDepInit <- bestPar(set$simDep)

  # Return the modifed set
  return(set)
}

##############################################################################################
# setWithPar <- createStaticParameter(set)
#
# Generates a fixed set of initial parameters for a set
##############################################################################################
createStaticParameter <- function(set) {
  # Function to create initial parameters depending on the available data
  bestPar <- function(data) {
    config <- set$Config(data)
    par <- c(apply(data$baseProductEstData, 1, mean, na.rm = TRUE), rep(0,
    config$m - config$n), rep(10^7, config$m), rep(1, config$n), rep(1,
    config$n), rep(0, config$n*(config$n-1)/2), c(1,1), rep(1, config$n),
    rep(1, config$d))
    return(par)
  }

  # For each of the four cases create the inital parameter set
  set$par <- list()
  set$par$simInit <- bestPar(set$sim)
  set$par$naInit <- bestPar(set$na)
  set$par$simDepInit <- bestPar(set$simDep)
  set$par$naDepInit <- bestPar(set$naDep)

  # Return the modified set
  return(set)
}
```

Listing 7: Static Parameter Generation

A.3.4. Parameter Optimisation

```
require(stats)
source("./Appendix/createSSTModel.R")
################################################################################
# optimisedSet <- runOptim(setWithParameters)
#
# Function that runs the optimisation algorithm on a set with attached
# parameters, the optimisation is run for all four cases:
# sim, na, simDep, and naDep
# The resulting optimised parameters are saved in
#   set$par,
# the required time in
#   set$time,
# the log-likelihood value in
#   set$optim,
# and the exit status of the optim algorithm in
#   set$status.
# If the optimisation failed the error message is also saved in
#   set$status.
################################################################################
runOptim <- function(set) {
  # Optimise the sim data set
  set$time$sim <- system.time(
    result <- try(optim(set$par$simInit, logLikP, config = set$Config(set$
    sim), data = set$sim, method = "L-BFGS-B"), silent = TRUE)
  )[3]
  if(class(result) == "try-error") {
    set$status$sim <- FALSE
    set$status$simMSG <- result
  } else {
    set$status$sim <- TRUE
    set$par$simInit_Optim <- result$par
    set$optimVal$sim <- result$value
  }

  # Optimise the na data set
  set$time$na <- system.time(
    result <- try(optim(set$par$naInit, logLikP, config = set$Config(set$na
    ), data = set$na, method = "L-BFGS-B"), silent = TRUE)
  )[3]
  if(class(result) == "try-error") {
    set$status$na <- FALSE
    set$status$naMSG <- result
```

```
  } else {
    set$status$na <- TRUE
    set$par$naInit_Optim <- result$par
    set$optimVal$na <- result$value
  }

  # Optimise the simDep data set
  set$time$simDep <- system.time(
    result <- try(optim(set$par$simDepInit, logLikP, config = set$Config(
    set$simDep), data = set$simDep, method = "L-BFGS-B"), silent = TRUE)
  )[3]
  if(class(result) == "try-error") {
    set$status$simDep <- FALSE
    set$status$simDepMSG <- result
  } else {
    set$status$simDep <- TRUE
    set$par$simDepInit_Optim <- result$par
    set$optimVal$simDep <- result$value
  }

  # Optimise the naDep data set
  set$time$naDep <- system.time(
    result <- try(optim(set$par$naDepInit, logLikP2, config = set$Config(
    set$naDep), data = set$naDep, method = "L-BFGS-B"), silent = TRUE)
  )[3]
  if(class(result) == "try-error") {
    set$status$naDep <- FALSE
    set$status$naDepMSG <- result
  } else {
    set$status$naDep <- TRUE
    set$par$naDepInit_Optim <- result$par
    set$optimVal$naDep <- result$value
  }

  return(set)
}

# Wrapper to call fkf and calculate the log-likelihood
logLikP <- function(par, config, data) {
  return(-call.fkf(createSSTModel(par, config, yt = data$fullEstData), yt =
     data$fullEstData)$logLik)
}
```

Listing 8: Optimisation Algorithm

A.3.5. Forecast Algorithms and Measurements

```
# Required to read months correctly
Sys.setlocale("LC_TIME", "C")
require(forecast)
require(tsintermittent)
require(zoo)
require(expm)


##############################################################################################
# fullForecast <- forecast.naive_set(set, h, rec = "simple")
#
# Generates a naive forecast, rec gives the reconciliation method
##############################################################################################
forecast.naive_set <- function(set, h, rec = "simple") {
  if(rec == "simple") {
    estData <- set$baseProductEstData[, dim(set$baseProductEstData)[[2]]]

    # Generate the forecast
    fcMatrix <- matrix(rep(estData, h), ncol = h, nrow = length(estData))
    # Apply bottom-up forecast
    recFcMatrix <- set$S %*% fcMatrix
  } else {
    estData <- set$fullEstData[, dim(set$fullEstData)[[2]]]

    # Generate the forecast
    fcMatrix <- matrix(rep(estData, h), ncol = h, nrow = length(estData))
    # Reconcile the forecast
    recFcMatrix <- S %*% ginv(t(S) %*% S) %*% t(S) %*% fcMatrix
  }

  return(recFcMatrix)
}

##############################################################################################
# fullForecast <- forecast.arima_set(set, h, rec = "simple")
#
# Generates an ARIMA forecast, rec gives the reconciliation method
##############################################################################################
forecast.arima_set <- function(set, h, rec = "simple") {
  date <- as.yearmon(colnames(set$baseProductEstData)[[1]])
  start <- as.numeric(format(date, "%Y")) + (as.numeric(format(date, "%m"))
    - 1)/12
```

```
  if(rec == "simple") {
    estData <- ts(t(set$baseProductEstData), frequency = 12, start = start)

    # Fit model and generate a forecast
    fitList <- apply(estData, 2, auto.arima)
    fcMatrix <- t(sapply(fitList, function(fit) { forecast(fit, h = h)$mean
    }))
    # Apply bottom-up forecast
    recFcMatrix <- set$S %*% fcMatrix
  } else {
    estData <- ts(t(set$fullEstData), frequency = 12, start = start)

    # Fit model and generate a forecast
    fitList <- apply(estData, 2, auto.arima)
    fcMatrix <- t(sapply(fitList, function(fit) { forecast(fit, h = h)$mean
    }))
    # Reconcile full forecast
    recFcMatrix <- set$S %*% ginv(t(set$S) %*% set$S) %*% t(set$S) %*%
    fcMatrix
  }

  return(recFcMatrix)
}

##############################################################################
# fullForecast <- forecast.ets_set(set, h, model = "ZZZ", rec = "simple")
#
# Generates an ETS forecast, rec gives the reconciliation method and
# model gives the ETS model
##############################################################################
forecast.ets_set <- function(set, h, model = "ZZZ", rec = "simple") {
  date <- as.yearmon(colnames(set$baseProductEstData)[[1]])
  start <- as.numeric(format(date, "%Y")) + (as.numeric(format(date, "%m"))
    - 1)/12
  if(rec == "simple") {
    estData <- ts(t(set$baseProductEstData), frequency = 12, start = start)

    # Fit model and generate a forecast
    fitList <- apply(estData, 2, ets, model = model)
    fcMatrix <- t(sapply(fitList, function(fit) { forecast(fit, h = h)$mean
    }))

    # Apply bottom-up forecast
```

```
81      recFcMatrix <- set$S %*% fcMatrix
    } else {
83      estData <- ts(t(set$fullEstData), frequency = 12, start = start)

85      # Fit model and generate a forecast
        fitList <- apply(estData, 2, ets, model = model)
87      fcMatrix <- t(sapply(fitList, function(fit) { forecast(fit, h = h)$mean
        }))
        # Reconcile full forecast
89      recFcMatrix <- set$S %*% ginv(t(set$S) %*% set$S) %*% t(set$S) %*%
        fcMatrix
    }
91
    return(recFcMatrix)
93  }

95  ##############################################################################
    # fullForecast <- forecast.crost(set, h, rec = "simple")
97  #
    # Generates a forecast using Croston's method, rec gives the reconciliation
99  # method
    ##############################################################################
101 forecast.crost <- function(set, h, rec = "simple") {
    date <- as.yearmon(colnames(set$baseProductEstData)[[1]])
103 start <- as.numeric(format(date, "%Y")) + (as.numeric(format(date, "%m"))
      - 1)/12

105 if(rec == "simple") {
      estData <- ts(t(set$baseProductEstData), frequency = 12, start = start)
107
      # Apply bottom-up forecast
109   fc <- t(apply(estData, 2, function(ts) { crost(ts, h = h, type = "
      croston")$frc.out }))
      # Apply bottom-up forecast
111   recFcMatrix <- set$S %*% fc
    } else {
113   estData <- ts(t(set$fullEstData), frequency = 12, start = start)

115   # Generate full forecast
      fc <- t(apply(estData, 2, function(ts) { crost(ts, h = h, type = "
      croston")$frc.out }))
117   # Reconcile full forecast
      recFcMatrix <- set$S %*% ginv(t(set$S) %*% set$S) %*% t(set$S) %*% fc
119 }
```

```
  return(recFcMatrix)
}

################################################################################
# fullForecast <- forecast.sba(set, h, rec = "simple")
#
# Generates a forecast using the SBA, rec gives the reconciliation
# method
################################################################################
forecast.sba <- function(set, h, rec = "simple") {
  date <- as.yearmon(colnames(set$baseProductEstData)[[1]])
  start <- as.numeric(format(date, "%Y")) + (as.numeric(format(date, "%m"))
    - 1)/12

  if(rec == "simple") {
    estData <- ts(t(set$baseProductEstData), frequency = 12, start = start)

    # Apply bottom-up forecast
    fc <- t(apply(estData, 2, function(ts) { crost(ts, h = h, type = "sba")
    $frc.out }))
    # Apply bottom-up forecast
    recFcMatrix <- set$S %*% fc
  } else {
    estData <- ts(t(set$fullEstData), frequency = 12, start = start)

    # Generate full forecast
    fc <- t(apply(estData, 2, function(ts) { crost(ts, h = h, type = "sba")
    $frc.out }))
    # Reconcile full forecast
    recFcMatrix <- set$S %*% ginv(t(set$S) %*% set$S) %*% t(set$S) %*%
    fcMatrix
  }

  return(recFcMatrix)
}

################################################################################
# fullForecast <- forecast.ihfc(model, h)
#
# Generates an IHFC forecast
################################################################################
forecast.ihfc <- function(model, h) {
  # Function to generate the h step ahead forecast
```

```
  singleFc <- function(h) {
    return(model$S %*% model$Z_2 %*% (model$Tt[,,1] %^%(h-1)) %*% model$at)
  }
  forecast <- sapply(1:h, singleFc)

  return(forecast)
}

##############################################################################
# fullForecast <- forecast.iihfc(model, h, method = "STATIC")
#
# Generates an IIHFC forecast, method describes the intermittent model
##############################################################################
forecast.iihfc <- function(model, h, method = "STATIC") {
  Xt <- model$Xt(method)
  # Function to generate the h step ahead forecast
  singleFc <- function(h) {
    return(model$S %*% Xt %*% model$Z_2 %*% (model$Tt[,,1] %^% (h-1)) %*%
    model$at )
  }
  forecast <- sapply(1:h, singleFc)

  return(forecast)
}
```

Listing 9: Forecast algorithms for Sets

```
##############################################################################
# valueVector <- aMeasureForecast(act_data, fc_data)
#
# Returns a vector of of all the measurements for this data sets
##############################################################################
aMeasureForecast <- function(act_data, fc_data) {
  return(c(aSMPE(act_data, fc_data), aSMAPE(act_data, fc_data), aTS(act_
    data, fc_data), aCPI(act_data, fc_data)))
}

##############################################################################
# value <- <averageMeasureFunction>(act_data, fc_data)
#
# Measures a forecast for a multivariate time series, returnd the average
##############################################################################
aSMPE <- function(act_data, fc_data) {
  return(mean(sapply(1:dim(act_data)[[1]], function(i) {
    return(SMPE(act_data[1,], fc_data[1,]))
```

```
  }))))
}

aSMAPE <- function(act_data, fc_data) {
  return(mean(sapply(1:dim(act_data)[[1]], function(i) {
    return(SMAPE(act_data[1,], fc_data[1,]))
  })))
}

aCPI <- function(act_data, fc_data) {
  return(mean(sapply(1:dim(act_data)[[1]], function(i) {
    return(CPI(act_data[1,], fc_data[1,]))
  })))
}

aTS <- function(act_data, fc_data) {
  return(mean(sapply(1:dim(act_data)[[1]], function(i) {
    return(TrackingSignal(act_data[1,], fc_data[1,]))
  })))
}

##############################################################################
# value <- <measureFunction>(actual, forecast)
#
# Measures a forecast for a univarite time series
##############################################################################
TrackingSignal <- function(actual, forecast) {
  mad <- sum(abs(actual-forecast), na.rm= TRUE) / sum(!is.nan(forecast), na
    .rm = TRUE)
  ts <- sum(actual-forecast, na.rm = TRUE) / mad

  return(ts)
}

SMPE <- function(actual, forecast) {
  return(round(mean(2 * (actual - forecast)/(abs(actual)+ abs(forecast)),
    na.rm = TRUE)))
}

CPI <- function(actual, forecast, digits = 3){
  return(round(mean(as.numeric(abs(actual - forecast) < 0.5 * forecast), na
    .rm = TRUE)))
}
```

Listing 10: Forecast algorithms for Sets

```
require(tsintermittent)
##############################################################################
# measuredTable <- createIHFCTableEntry(set, dep = FALSE)
#
# Calculate the forecast performance of the different (I)IHFC methods
# If dep = TRUE, then the sets with aggregates are used
##############################################################################
createIHFCTableEntry <- function(set, dep = FALSE) {
  # Choose the right parameters
  if(dep) {
    data_sim <- set$simDep
    data_na <- set$naDep
    if(set$status$simDep) {
      simPar <- set$par$simDepInit_Optim
    } else {
      simPar <- set$par$simDepInit
    }
    if(set$status$naDep) {
      naPar <- set$par$naDepInit_Optim
    } else {
      naPar <- set$par$naDepInit
    }
  } else {
    data_sim <- set$sim
    data_na <- set$na
    if(set$status$sim) {
      simPar <- set$par$simInit_Optim
    } else {
      simPar <- set$par$simInit
    }
    if(set$status$na) {
      naPar <- set$par$naInit_Optim
    } else {
      naPar <- set$par$naInit
    }
  }

  # Add the statistical values
  cv2 <- round(mean(apply(cbind(data_sim$fullEstData, data_sim$fullFCData),
     1, function(ts) { return(idclass(ts, type = "SBC", outplot = FALSE)$
    cv2) }), na.rm = TRUE), 3)
```

```
  p <- round(mean(apply(cbind(data_sim$fullEstData, data_sim$fullFCData),
    1, function(ts) { return(idclass(ts, type = "SBC", outplot = FALSE)$p)
    }), na.rm = TRUE), 3)

  # Generate IHFC forecast
  fc_ihfc <- forecast.ihfc(createSSTModel(simPar, config = set$Config(data_
    sim), yt = data_sim$fullEstData), set$FCLength)

  # Generate the different IIHFC forecasts
  model <- createSSTModel(naPar, config =  set$Config(data_sim), yt = data_
    na$fullEstData)
  fc_iihfc_stat <- forecast.iihfc(model, set$FCLength, "STATIC")
  fc_iihfc_stat2 <- forecast.iihfc(model, set$FCLength, "STATIC2")
  fc_iihfc_cro <- forecast.iihfc(model, set$FCLength, "CRO")
  fc_iihfc_sba <- forecast.iihfc(model, set$FCLength, "SBA")
  actual <- data_sim$fullFCData

  # Remove the added lines to calculate the forecast performance
  if(dep) {
    fc_ihfc <- fc_ihfc[-c(1:3),]
    fc_iihfc_stat <- fc_iihfc_stat[-c(1:3),]
    fc_iihfc_stat2 <- fc_iihfc_stat2[-c(1:3),]
    fc_iihfc_cro <- fc_iihfc_cro[-c(1:3),]
    fc_iihfc_sba <- fc_iihfc_sba[-c(1:3),]
    actual <- actual[-c(1:3),]
  }
  # Calculate the results
  result <- rbind(c(set$name, rep(NA, 6)),
                  c("IHFC", p, cv2, aMeasureForecast(actual, fc_ihfc)),
                  c("IHFC-1", p, cv2, aMeasureForecast(actual, fc_iihfc_
    stat)),
                  c("IHFC-2", p, cv2, aMeasureForecast(actual, fc_iihfc_
    stat2)),
                  c("IHFC-CRO", p, cv2, aMeasureForecast(actual, fc_iihfc_
    cro)),
                  c("IHFC-SBA", p, cv2, aMeasureForecast(actual, fc_iihfc_
    sba)))

  # Name and return the results
  colnames(result) <- c("Algorithm", "p", "cv2", "sMpe", "sMape", "TS", "
    CPI")
  return(result)
}
```

```
75 ##############################################################################
   # measuredTable <- createARTableEntry(set, iihfc_method, dep = FALSE)
77 #
   # Calculate the forecast performance of the different methods with the
79 # (I)IHFC method of choice
   # If dep = TRUE, then the sets with aggregates are used
81 ##############################################################################
   createARTableEntry <- function(set, iihfc_method, dep = FALSE) {
83   # Choose the right parameters
     if(dep) {
85     data_sim <- set$simDep
       data_na <- set$naDep
87     if(set$status$simDep) {
         simPar <- set$par$simDepInit_Optim
89     } else {
         simPar <- set$par$simDepInit
91     }
       if(set$status$naDep) {
93       naPar <- set$par$naDepInit_Optim
       } else {
95       naPar <- set$par$naDepInit
       }
97   } else {
       data_sim <- set$sim
99     data_na <- set$na
       if(set$status$sim) {
101      simPar <- set$par$simInit_Optim
       } else {
103      simPar <- set$par$simInit
       }
105    if(set$status$na) {
         naPar <- set$par$naInit_Optim
107    } else {
         naPar <- set$par$naInit
109    }
     }
111
     # Generate the statistical values
113  cv2 <- round(mean(apply(cbind(data_sim$fullEstData, data_sim$fullFCData),
        1, function(ts) { return(idclass(ts, type = "SBC", outplot = FALSE)$
       cv2) }), na.rm = TRUE), 3)
     p <- round(mean(apply(cbind(data_sim$fullEstData, data_sim$fullFCData),
       1, function(ts) { return(idclass(ts, type = "SBC", outplot = FALSE)$p)
       }), na.rm = TRUE), 3)
```

```
115
    # Create the right (I)IHFC forecast
117 if(iihfc_method == "NO") {
      fc_iihfc <- forecast.ihfc(createSSTModel(simPar, config = set$Config(
      data_sim), yt = data_sim$fullEstData), set$FCLength)
119   fc_string <- "IHFC"

121 } else {
      model <- createSSTModel(naPar, config =  set$Config(data_sim), yt =
      data_na$fullEstData)
123
      if(iihfc_method == "STATIC"){
125     fc_iihfc <- forecast.iihfc(model, set$FCLength, "STATIC")
        fc_string <- "IIHFC-1"
127   } else if (iihfc_method == "STATIC2") {
        fc_iihfc <- forecast.iihfc(model, set$FCLength, "STATIC2")
129     fc_string <- "IIHFC-2"
      } else if(iihfc_method == "CRO") {
131     fc_iihfc <- forecast.iihfc(model, set$FCLength, "CRO")
        fc_string <- "IIHFC-CRO"
133   } else if(iihfc_method == "SBA") {
        fc_iihfc <- forecast.iihfc(model, set$FCLength, "SBA")
135     fc_string <- "IIHFC-SBA"
      }
137 }

139 # Create the other forecasts
    actual <- data_sim$fullFCData
141 fc_arima <- forecast.arima_set(data_sim, set$FCLength)
    fc_ets <- forecast.ets_set(data_sim, set$FCLength)
143 fc_cro <- forecast.crost(data_sim, set$FCLength)
    fc_sba <- forecast.sba(data_sim, set$FCLength)
145
    # Remove the added lines to calculate the forecast performance
147 if(dep) {
      fc_iihfc <- fc_iihfc[-c(1:3),]
149   fc_arima <- fc_arima[-c(1:3),]
      fc_ets <- fc_ets[-c(1:3),]
151   fc_cro <- fc_cro[-c(1:3),]
      fc_sba <- fc_sba[-c(1:3),]
153   actual <- actual[-c(1:3),]
    }
155 # Calculate the forecast performance
    result <- rbind(c(set$name, rep(NA, 6)),
```

```
                    c("ARIMA", p, cv2, aMeasureForecast(actual, fc_arima)),
                    c("ETS", p, cv2, aMeasureForecast(actual, fc_ets)),
                    c("CRO", p, cv2, aMeasureForecast(actual, fc_cro)),
                    c("SBA", p, cv2, aMeasureForecast(actual, fc_sba)),
                    c(fc_string, p, cv2, aMeasureForecast(actual, fc_iihfc)))

  # Name and return the results
  colnames(result) <- c("Algorithm", "p", "cv2", "sMpe", "sMape", "TS", "
    CPI")
  return(result)
}
```

Listing 11: Generating measurement tables

B. Figures

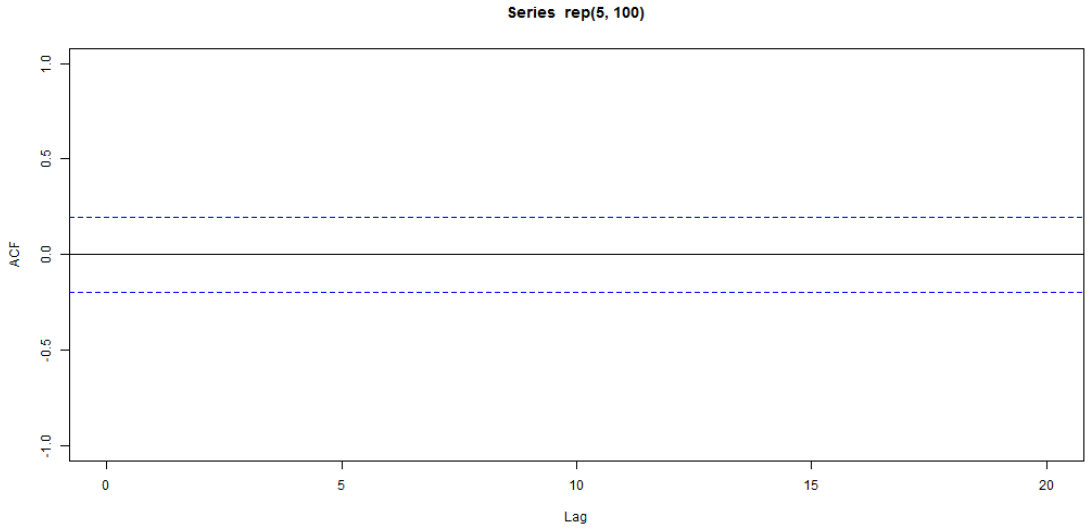

Figure 5: ACF of a constant function

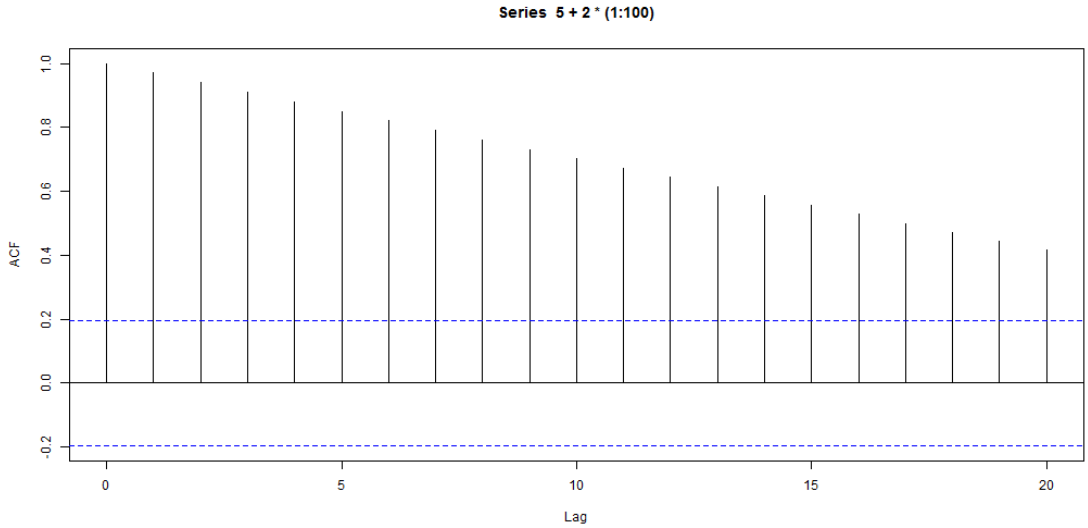

Figure 6: ACF of a linear function

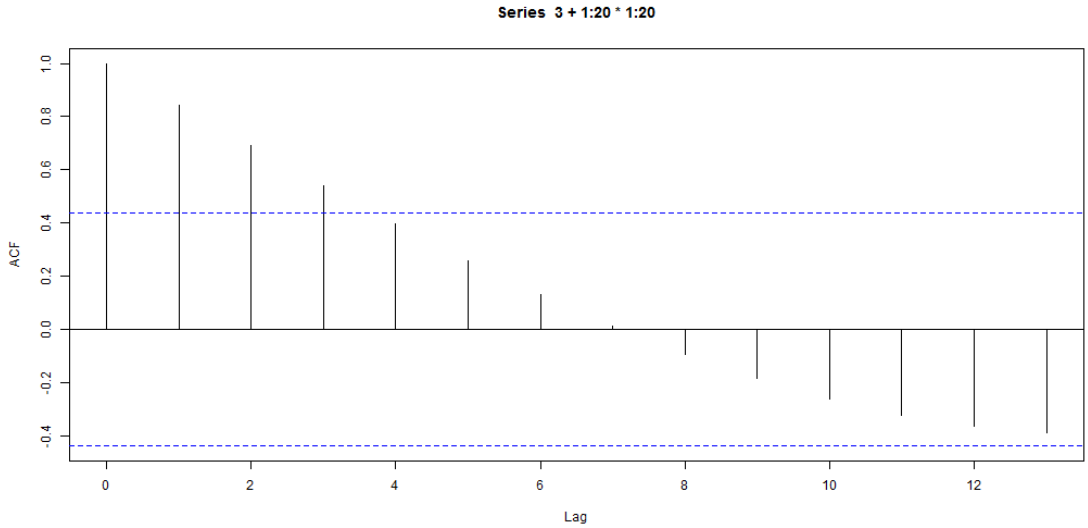

Figure 7: ACF of a quadratic function

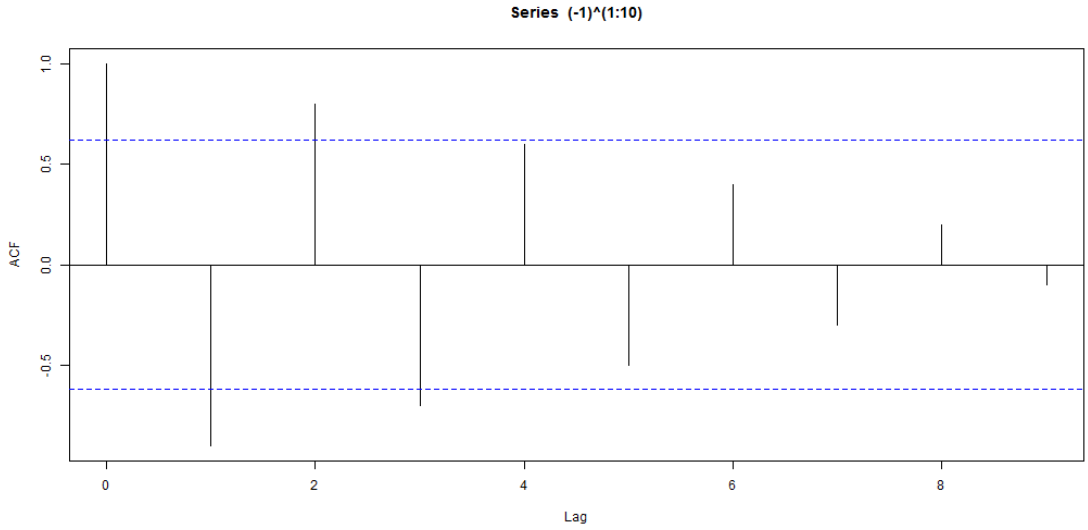

Figure 8: ACF of an alternating function

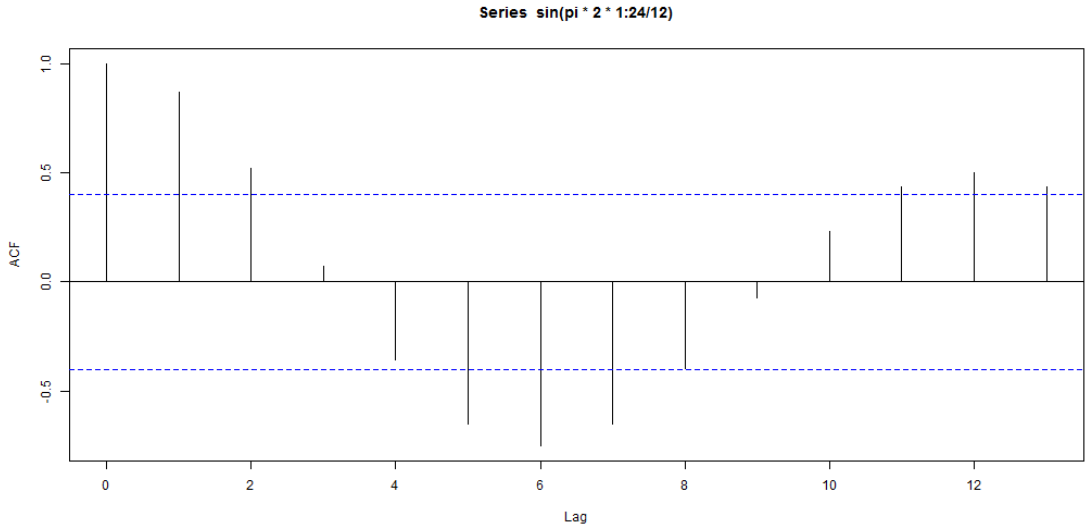

Figure 9: ACF of a sinusoidal function

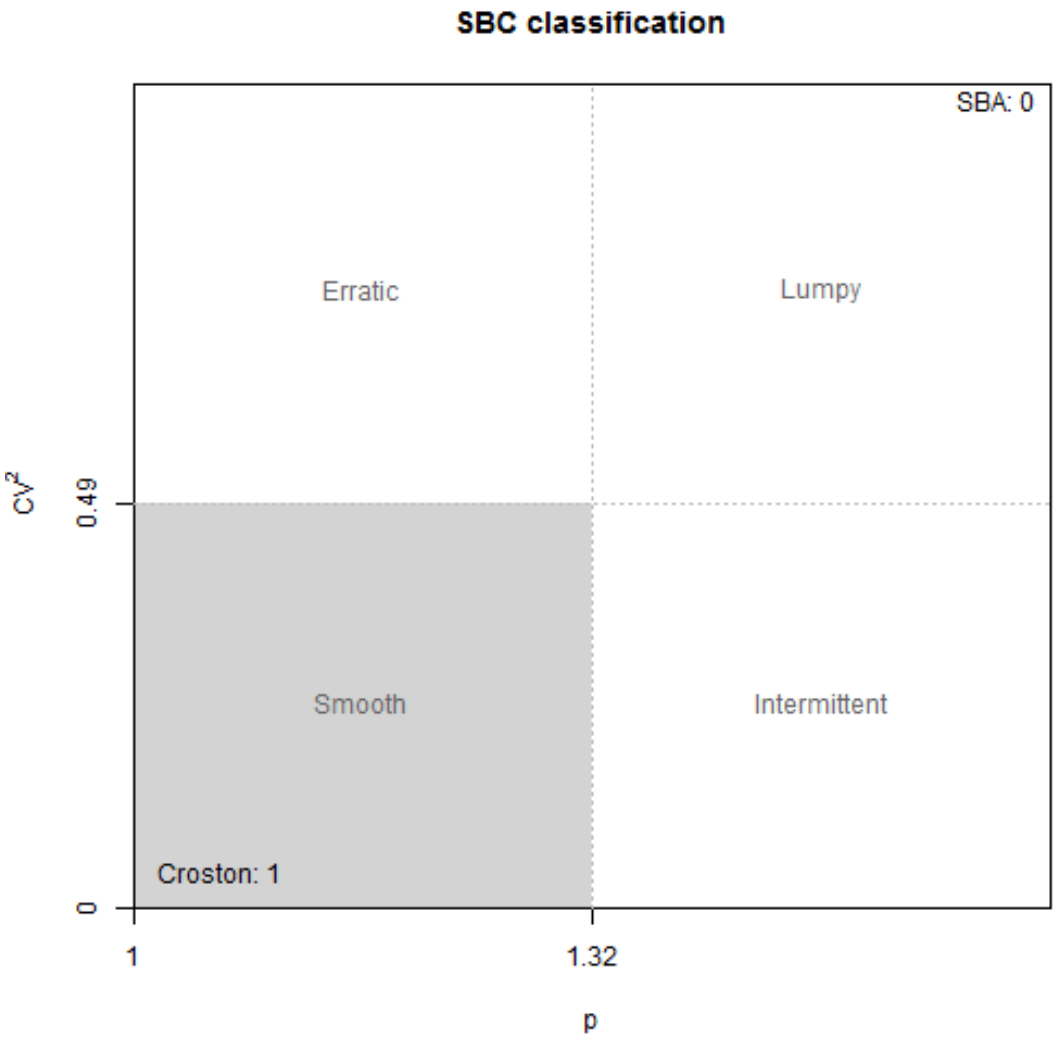

Figure 10: SBC classification schema

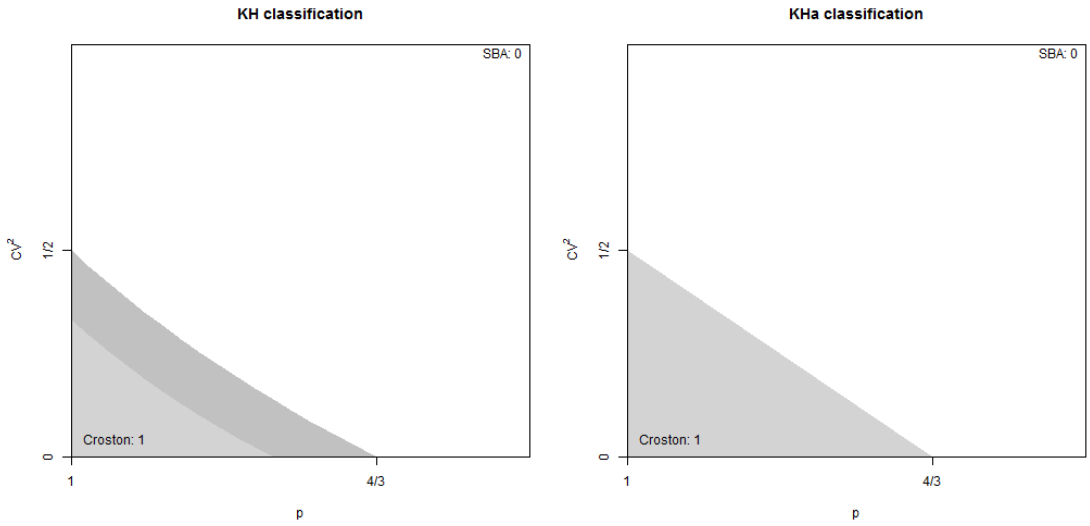

Figure 11: KHs exact and approximate classification schemas

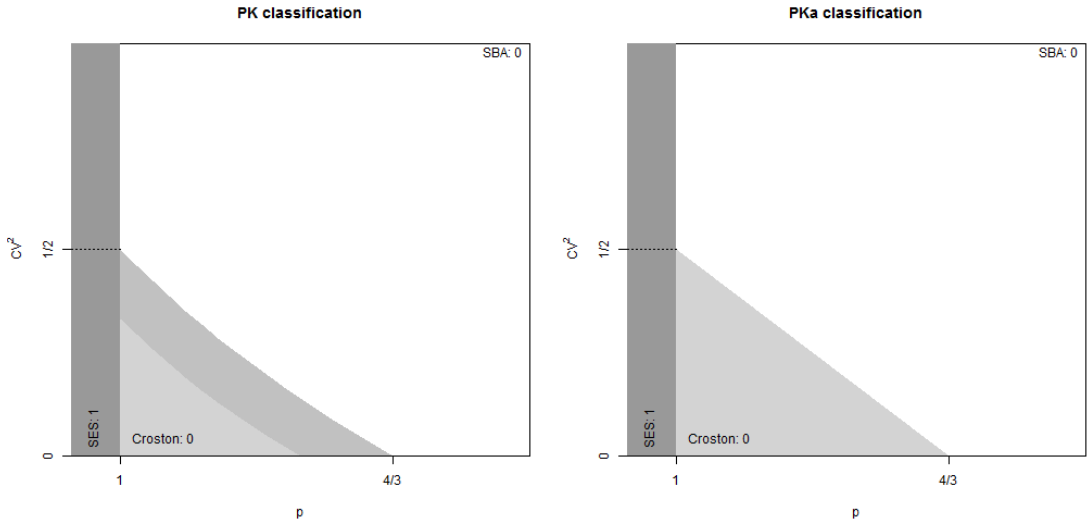

Figure 12: Classification schemas PK and PKa

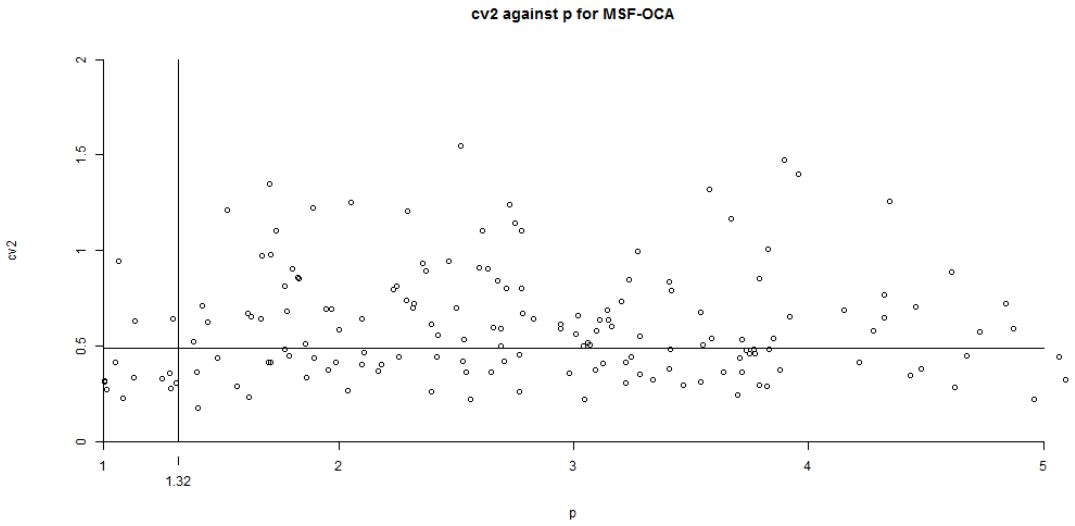

Figure 13: Plot of average inter-demand interval against the coefficient of variation for products sets from OCA.

C. Extensive Tables

C.1. Random Initialised Parameters

C.2. Static Initialised Parameters

C.3. Comparison with simplistic Forecasting Methods

Table 6: Forecast performance with random initialised optimisation

Algorithm	adi	cv^2	sMpe	sMape	TS	CPI
ABAAJBAAIFA						
IHFC	2.639	0.601	2	1.96	12	0
IHFC-1	2.639	0.601	2	1.978	12	0
IHFC-2	2.639	0.601	2	1.991	12	0
IHFC-CRO	2.639	0.601	2	1.983	12	0
IHFC-SBA	2.639	0.601	2	1.984	12	0
ADAARAAAHAA						
IHFC	2.166	0.841	2	1.943	12	0
IHFC-1	2.166	0.841	2	1.777	12	0
IHFC-2	2.166	0.841	2	2	12	0
IHFC-CRO	2.166	0.841	2	1.835	12	0
IHFC-SBA	2.166	0.841	2	1.837	12	0
ADAAAFAAHAA						
IHFC	1.857	0.712	2	2	12	0
IHFC-1	1.857	0.712	1	1.423	11.742	0
IHFC-2	1.857	0.712	2	1.825	12	0
IHFC-CRO	1.857	0.712	1	1.474	11.948	0
IHFC-SBA	1.857	0.712	1	1.477	11.947	0
AEAAYIAAZDA						
IHFC	1.477	0.31	1	1.575	1.374	0
IHFC-1	1.477	0.31	2	2	12	0
IHFC-2	1.477	0.31	2	1.977	12	0
IHFC-CRO	1.477	0.31	2	2	12	0
IHFC-SBA	1.477	0.31	2	2	12	0

Table 7: Forecast performance with static initialised optimisation

Algorithm	adi	cv^2	sMpe	sMape	TS	CPI
ABAAJBAAIFA						
IHFC	2.639	0.601	0	0.402	-5.356	1
IHFC-1	2.639	0.601	-1	0.73	-11.77	0
IHFC-2	2.639	0.601	0	0.403	7.587	1
IHFC-CRO	2.639	0.601	0	0.52	-9.841	1
IHFC-SBA	2.639	0.601	0	0.458	-8.709	1
ADAARAAAHAA						
IHFC	2.166	0.841	1	0.555	11.874	0
IHFC-1	2.166	0.841	0	0.418	0.345	1
IHFC-2	2.166	0.841	2	1.994	12	0
IHFC-CRO	2.166	0.841	0	0.462	9.497	1
IHFC-SBA	2.166	0.841	0	0.47	10.135	0
ADAAAFAAHAA						
IHFC	1.857	0.712	0	0.616	-7.02	1
IHFC-1	1.857	0.712	-1	0.807	-10.866	0
IHFC-2	1.857	0.712	1	1.34	11.192	0
IHFC-CRO	1.857	0.712	-1	0.63	-7.814	1
IHFC-SBA	1.857	0.712	-1	0.632	-7.908	1
AEAAYIAAZDA						
IHFC	1.477	0.31	0	0.253	6.962	1
IHFC-1	1.477	0.31	0	0.255	-9.398	1
IHFC-2	1.477	0.31	1	1.118	12	0
IHFC-CRO	1.477	0.31	0	0.249	6.683	1
IHFC-SBA	1.477	0.31	0	0.221	4.575	1

Table 8: Forecast performance with static initialised optimisation, comparing with non-integrated algorithms

Algorithm	adi	cv^2	sMpe	sMape	TS	CPI
ABAAJBAAIFA						
ARIMA	2.639	0.601	-1	0.664	-11.277	1
ETS	2.639	0.601	0	0.391	-4.129	1
CRO	2.639	0.601	0	0.365	0.348	1
SBA	2.639	0.601	0	0.365	0.962	1
IHFC	2.639	0.601	0	0.402	-5.356	1
ADAARAAAHAA						
ARIMA	2.166	0.841	1	0.516	11.833	0
ETS	2.166	0.841	0	0.476	10.73	0
CRO	2.166	0.841	0	0.524	11.643	0
SBA	2.166	0.841	1	0.529	11.682	0
IIHFC-1	2.166	0.841	0	0.418	0.345	1
ADAAAFAAHAA						
ARIMA	1.857	0.712	0	0.602	-5.652	1
ETS	1.857	0.712	0	0.594	-5.416	1
CRO	1.857	0.712	0	0.599	-5.955	1
SBA	1.857	0.712	0	0.601	-6.087	1
IIHFC-CRO	1.857	0.712	-1	0.63	-7.814	1
AEAAYIAAZDA						
ARIMA	1.477	0.31	0	0.257	-9.138	1
ETS	1.477	0.31	0	0.265	-9.362	1
CRO	1.477	0.31	0	0.248	6.635	1
SBA	1.477	0.31	0	0.242	6.228	1
IIHFC-SBA	1.477	0.31	0	0.221	4.575	1