



B A C H E L O R A R B E I T

Vergleich von Machine Learning und einem modellbasierten Ansatz zur Optimierung der Heizstrategie für die Trinkwarmwasserversorgung

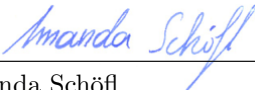
ausgeführt am
Institut für Analysis und Scientific Computing
an der
Technischen Universität Wien

unter der Anleitung von
Prof. Felix Breitenecker,
Dr. Andreas Körner
und
Stephanie Kaschewski

durch
Amanda Schöfl
Matrikelnummer: 0271473
Hlawkagasse 14/24
1100 Wien

Die vorliegende Studienabschlussarbeit beinhaltet vertrauliche Informationen der Robert Bosch GmbH und darf Dritten - außer Mitarbeitern der Universität im Rahmen des hochschulinternen Prüfungsverfahrens - nicht zugänglich gemacht werden. Veröffentlichungen und Vervielfältigungen, auch nur auszugsweise, sind ohne ausdrückliche Genehmigung der Robert Bosch GmbH untersagt. Ab 1.1.2025 unterliegt diese Studienabschlussarbeit nicht mehr der Geheimhaltung.

Wien, am 27. Februar 2019


Amanda Schöfl

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Bachelorarbeit selbstständig und ohne unzulässige Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am 27. Februar 2019


Amanda Schöfl

Danksagung

Ein großes Dankeschön an die Robert Bosch GmbH, die es mir ermöglicht hat, den praktischen Teil dieser Arbeit im Rahmen eines Praktikums im Forschungszentrum in Renningen (Deutschland) durchzuführen. Ich hatte dort die Möglichkeit einen Einblick in die Tätigkeiten der Abteilung CR/AEB2 zu bekommen und auch eigene Ideen bezüglich des in meiner Arbeit behandelten Themas einzubringen.

Außerdem möchte ich mich bei Yoni Sharaby von der Universität Haifa bedanken, der mir sehr ausführlich auf einige Fragen bezüglich seines Papers geantwortet und mir extra Daten in toll aufbereiteter Form zur Verfügung gestellt hat.

Last, but not least, danke ich herzlich Stefanie Kaschewski und Andreas Körner für ihre geduldige und engagierte Betreuung.

Inhaltsverzeichnis

1. Einleitung	1
2. Clustern der Warmwasserverbrauchsdaten	2
2.1. Vorliegende Daten	2
2.2. Methoden für das Clustern von Zeitreihen	3
2.3. Praktische Umsetzung	16
3. Optimierung des Heizverhaltens	18
3.1. Modellierung der Bakterienkonzentration und des Heizverhaltens eines Warmwasserspeichers in Abhängigkeit vom Warmwasserverbrauch	20
3.2. Das implementierte Modell	24
3.3. Abschätzung der durch Annahmen und Vereinfachung entstandenen Fehler	27
4. Ergebnisse	29
4.1. Evaluierung der Partitionen des Warmwasserverbrauchs	29
4.2. Resultate der Optimierung des Heizverhaltens	34
5. Fazit	37
Literaturverzeichnis	39
A. Sourcecode	41
A.1. Clustern	41
A.2. Modellierung	68
B. Silhouettenwerte	74

1. Einleitung

Bei der Speicherung von Trinkwarmwasser gilt es zu beachten, dass die vom Menschen als angenehm empfundenen Temperaturen auch optimale Wachstumsbedingungen für im Trinkwasser vorkommende, gesundheitsgefährdende Bakterien bieten. Dazu gehören unter anderem diverse Stämme von Legionellen, die sich besonders gut in etwa 35 °C warmem Wasser vermehren. Durch das Einatmen von Wasserdampf mit einer hohen Legionellenkonzentration, beispielsweise beim Duschen, können Krankheiten wie Pontiac Fieber oder Lungenentzündung verursacht werden. Dies kann vor Allem für Personen mit schwachem Immunsystem gefährlich werden. Wie stark sich diese Bakterien vermehren, hängt unter anderem vom täglichen Verbrauch und der Temperatur des Wassers ab [2].

Um Warmwasser energieeffizienter speichern zu können, ohne eine zu hohe Legionellenkonzentration zu riskieren, ist es daher erstrebenswert das Heizverhalten des Speichers individuell an den Wasserverbrauch des Nutzers anzupassen.

Der erste Teil dieser Arbeit beschäftigt sich mit der Frage, ob mittels *Unsupervised Machine Learning Methoden* Gruppen von Verbrauchern mit ähnlichem Nutzungsverhalten gefunden werden können. Dafür stehen Daten über den Warmwasserverbrauch von rund 42000 Nutzern in anonymisierter Form zur Verfügung, welche täglich über ungefähr ein Jahr aufgezeichnet wurden. In Kapitel 2 werden zunächst die untersuchten Daten vorgestellt. Anschließend werden die für das Clustern notwendigen Methoden eingeführt und auf deren Umsetzung eingegangen. Zur Evaluierung der Clusterqualität dient der Silhouettenkoeffizient. Er erweist sich, wie in Abschnitt 4.1 erläutert, als nicht zielführend für den vorliegenden Datensatz.

Im zweiten Teil der Arbeit geht es darum, bei gegebenem Wasserverbrauch ein möglichst energieeffizientes Heizverhalten zu finden, sodass die Legionellenkonzentration in einem modellierten Warmwasserspeicher unter einem gewissen Schwellenwert bleibt. In Kapitel 3 wird zunächst ein grundlegendes *Modell* der Legionellenkonzentration abhängig von Temperatur und Wasserverbrauch erstellt. Dies wird vereinfacht und in dem Computersystem GAMS - General Algebraic Modeling System - implementiert. Anschließend wird mittels des Solvers SCIP - Solving Constraint Integer Programms - für einen gegebenen Warmwasserverbrauch ein Heizverhalten gesucht, welches über den Zeitraum von einem Monat zu einem möglichst geringen Energieverbrauch führt, ohne dass die Bakterienkonzentration einen Schwellenwert übersteigt. Außerdem werden die Grenzen des Modells diskutiert. Die Optimierung des Heizverhaltens wird in Abschnitt 4.2 anhand eines Beispiels untersucht.

Am Ende wird eine Einschätzung gegeben, ob sich einer der beiden Ansätze als hilfreich für die Optimierung der Heizstrategie für die Trinkwarmwasserversorgung erweist.

2. Clustern der Warmwasserverbrauchsdaten

2.1. Vorliegende Daten

Für diese Arbeit liegen Warmwassernutzungsdaten von 42000 Warmwassergeräten vor. Dabei wurden 349 Tage lang Parameter gemäß Tabelle 1 für jedes Gerät aufgezeichnet.

<i>Taps</i>	Anzahl der Warmwasserzapfvorgänge pro Tag
<i>Dauer</i>	Gesamtdauer aller Zapfvorgänge des Tages in Sekunden
<i>Durchflussgeschwindigkeit</i>	gemittelt über die Gesamtdauer aller Zapfvorgänge des Tages in Litern pro Minute

Tabelle 1.: Für jedes Gerät täglich aufgezeichnete Parameter

Daraus lässt sich für jeden Tag und Haushalt gemäß

$$\text{Warmwasserverbrauch} = \frac{\text{Dauer} \cdot \text{Durchflussgeschwindigkeit}}{60}$$

der gesamte Warmwasserverbrauch berechnen. Abbildung 1 zeigt diesen für eines der Geräte.

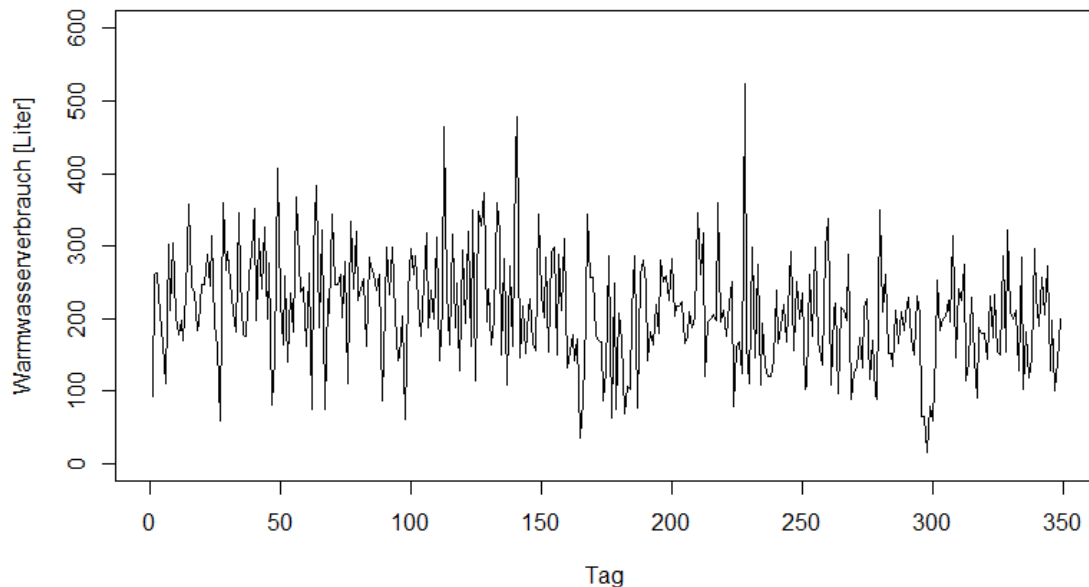


Abbildung 1.: Beispielzeitreihe des täglichen Warmwasserverbrauch über 349 Tage

Da es bei der Aufzeichnung der Daten immer wieder zu Ausfällen und Fehlern gekommen ist, sind nicht bei allen Geräten für jeden Tag Messdaten vorhanden. Aus diesem Grund werden in dieser Arbeit nur jene Messreihen in Betracht gezogen, bei denen mindestens für fünf Prozent der Tage alle Messwerte vorhanden sind. Die fehlenden Datenpunkte werden mittels linearer Interpolation extrapoliert. Werte des Warmwasserverbrauchs, die 600 Liter pro Tag übersteigen werden als Messfehler oder Ausreißer betrachtet und mit 600 ersetzt.

2.2. Methoden für das Clustern von Zeitreihen

Zunächst wird der Begriff der Zeitreihe eingeführt.

Definition 1. Sei \mathbb{T} eine Indexmenge. Ein stochastischer Prozess $(X_t \mid t \in \mathbb{T})$ ist eine Familie von reellwertigen Zufallsvariablen, die auf einem Wahrscheinlichkeitsraum $(\Omega, \mathcal{A}, \mathbb{P})$ definiert sind:

$$\begin{aligned} (\Omega \times \mathbb{T}) &\longrightarrow \mathbb{R}^n \\ (\omega, t) &\longmapsto X_t(\omega) \end{aligned}$$

Für $\mathbb{T} \subseteq \mathbb{Z}$ nennt man die Familie $(x_t = X_t(\omega) \in \mathbb{R}^n \mid t \in \mathbb{T})$ eine Zeitreihe. Sind alle X_t reellwertig, heißt die Zeitreihe univariat.

In dem betrachteten Fall ist $\mathbb{T} = \{1, \dots, 349\}$ und die Werte der Zufallsvariablen X_t geben den Warmwasserverbrauch am Tag t an. Weiters wird im folgenden zwischen *Metrik* und *Abstand* unterschieden.

Definition 2. Sei Ω eine Menge und $d: \Omega \times \Omega \rightarrow [0, \infty)$. Die Abbildung d heißt *Abstand*. Falls die Funktion zusätzlich für alle x, y, z aus Ω erfüllt:

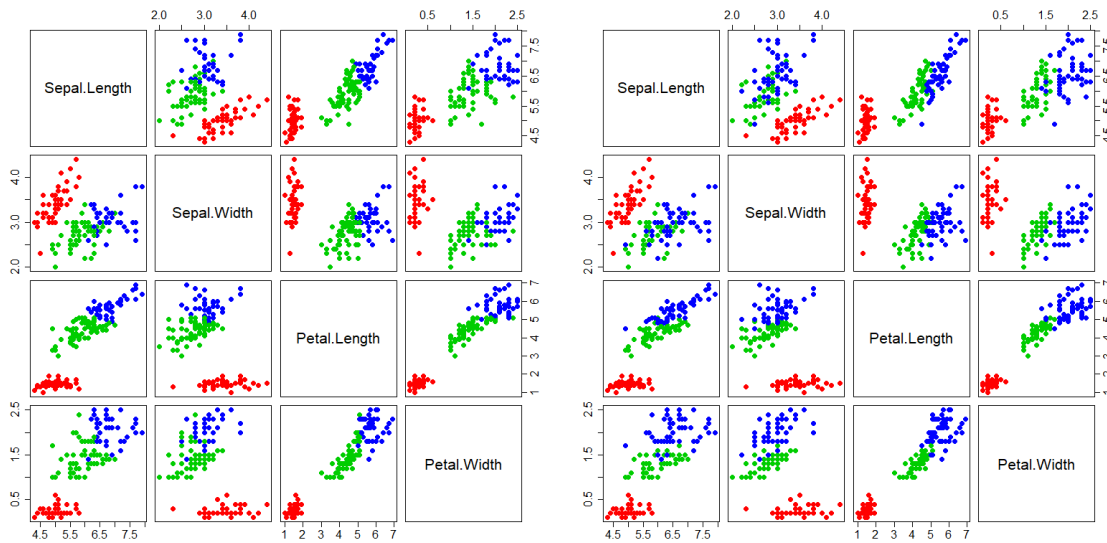
- (i) Identitätsbedingung $d(x, y) = 0 \Leftrightarrow x = y$,
- (ii) Symmetrie $d(x, y) = d(y, x)$,
- (iii) Dreiecksungleichung $d(x, y) \leq d(x, z) + d(z, y)$,

dann wird sie *Metrik* genannt.

Nun kann der Begriff des Clusters eingeführt werden. Die folgende Definition orientiert sich an jener im Artikel [1].

Definition 3. Sei $D = \{z_1, \dots, z_n\}$ eine Menge von Objekten, dann ist das Clustern der Prozess des unüberwachten maschinellen Teilens der Menge D in ein Mengensystem $C = \{C_1, \dots, C_k\}$ sodass gleichartige Objekte basierend auf einem gegebenen Abstands begriff in einer Gruppe C_i zusammengefasst werden. C_i wird ein *Cluster* genannt wobei gilt, dass $D = \bigcup_{i=1}^k C_i$ und $C_i \cap C_j = \emptyset$ für $i \neq j$.

Beispiel 1. Der Vorgang des Clustern kann anhand des in der Statistiksprache R frei verfügbaren Datensatzes *iris* illustriert werden. Er beinhaltet Abmessungen von 150 Schwertlilienblüten anhand derer die Pflanzen in drei verschiedene Arten eingeteilt werden. Weiters können diese Parameter mittels Unsupervised Machine Learning geclustert werden.



(a) Einteilung gemäß des Algorithmus Partitioning Around Medoids.
 blau: Cluster1, grün: Cluster2, rot: Cluster3

(b) Einteilung gemäß Spezieszugehörigkeit.
 blau: virginica, grün: versicolor, rot: setosa

Abbildung 2.: Einteilung von 150 Exemplaren der Schwertlilie nach den Parametern sepal length, sepal width, petal length, petal width, Abmessungen jeweils in cm.

In Abbildung 2 ist die Partition, erzeugt durch den Algorithmus Partitioning Around Medoids, welcher in Abschnitt 2.2 erläutert wird, der Aufteilung gemäß Artzugehörigkeit gegenübergestellt.

Im Allgemeinen enthält die Menge D eine große Anzahl von Elementen. In dieser sollen durch das Clustern nicht offensichtliche Strukturen und Muster erkannt und so Information über die vorliegenden Daten gewonnen werden. In dieser Arbeit sind die in D enthaltenen Objekte Zeitreihen, welche den täglichen Warmwasserverbrauch angeben. Nach [1] setzt sich der Prozess des Clustersns von Zeitreihen aus vier Komponenten zusammen.

- (i) *Repräsentation der Zeitreihen*
- (ii) *Wahl des Abstandsbegriffs*
- (iii) *Wahl des Clusteralgorithmus*
- (iv) *Finden von Prototypen*

Dem sei noch ein fünfter Punkt hinzugefügt, die

- (v) *Bestimmung der Clusterqualität.*

Im Folgenden werden für die oben genannten Punkte die in dieser Arbeit verwendeten Methoden vorgestellt. Die einzige Ausnahme bildet das Finden von Prototypen, da dies im Rahmen der Arbeit nicht explizit durchgeführt wird. Bei der Auswahl der Algorithmen waren vor allem Erfolge in ähnlichen Anwendungsgebieten ausschlaggebend.

Repräsentation der Zeitreihen

Ein einzelner Datenpunkt kann als Zeitreihe mit 349 Werten, oder anders formuliert, als Vektor mit 349 Einträgen angesehen werden. Diese hohe Dimension bringt einige Probleme mit sich. Zum Einen ist ein großer Rechenaufwand nötig um viele Zeitreihen zu verarbeiten, zum Anderen gestaltet sich die Messung des Abstands mit zunehmender Dimension immer schwieriger. Näheres dazu im Abschnitt 2.2. Ein Versuch diese Schwierigkeiten zu umgehen ist die Dimension der Datenpunkte zu reduzieren.

Eine Möglichkeit dies zu bewerkstelligen ist die *Piecewise Aggregation Approximation (PAA)*. Sie wird in [8] als geeignete Methode der Bearbeitung von Zeitreihen vor dem Ermitteln des Dynamic Time Warping Abstands vorgeschlagen, welcher noch in Abschnitt 2.2 besprochen wird. Die Zeitreihe (x_1, \dots, x_n) wird dabei zu $(\bar{x}_1, \dots, \bar{x}_N)$ mit $N < n$. Um die Notation zu vereinfachen sei n ein Vielfaches von N . Dann ergibt sich

$$\bar{x}_i = \frac{N}{n} \sum_{j=\frac{n}{N}(i-1)+1}^{\frac{n}{N}i} x_j.$$

Mit $N = 349$ und $n = 50$ wird die eingangs betrachtete Beispielzeitreihe zu jener, in Abbildung 3.

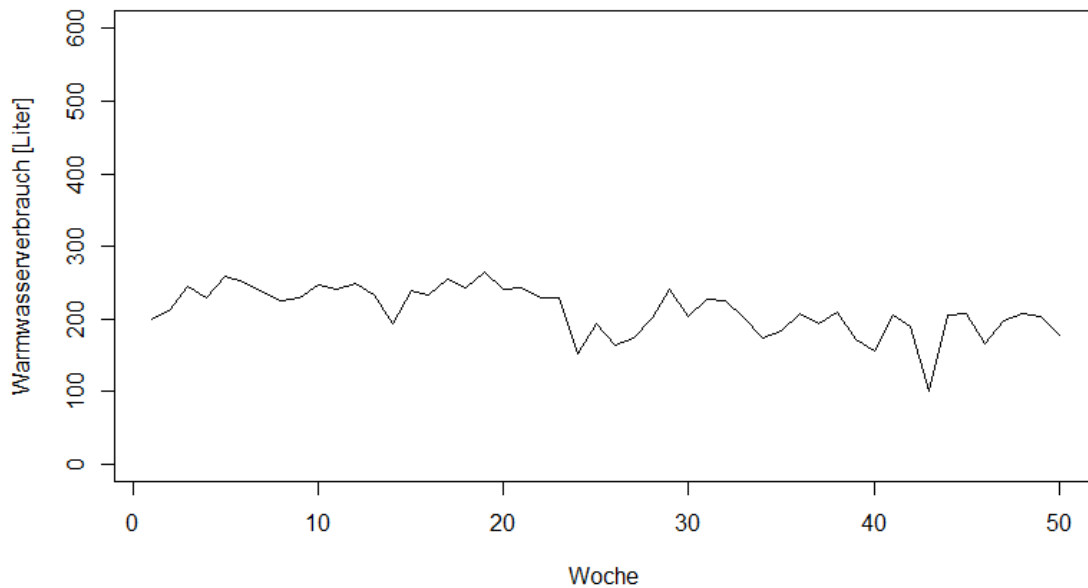


Abbildung 3.: Beispielzeitreihe des täglichen Warmwasserverbrauch über 349 Tage mittels PAA verkürzt auf die Länge 50

Eine weitere Möglichkeit die Dimension zu reduzieren ist nur einen *Ausschnitt der Zeitreihe* als Input für den Clusteralgorithmus zu verwenden. Konkret werden die Tage 128 bis 273 betrachtet. Das entspricht den Sommermonaten Juli, August und September, welche in Europa die Haupt-Urlaubssaison darstellen.

In [13] wird vorgeschlagen aus einer Zeitreihe $X = (x_1, \dots, x_n)$ die ersten vier statistischen Momente zu berechnen. Diese sind

- (i) der *Mittel- oder Erwartungswert* μ gegeben durch $\mu = \sum_{t=1}^n \frac{x_t}{n}$,
- (ii) die *Standardabweichung* σ gegeben durch $\sigma = \sqrt{\sum_{t=1}^n \frac{(x_t - \mu)^2}{n}}$,
- (iii) die *Schiefe* SKEW gegeben durch $\text{SKEW} = \sum_{t=1}^n \frac{(x_t - \mu)^3}{n\sigma^3}$
- (iv) und die *Kurtosis* KURT gegeben durch $\text{KURT} = \sum_{t=1}^n \frac{(x_t - \mu)^4}{n\sigma^4}$.

Angelehnt an selbige Arbeit sollen zusätzlich die Parameter

- (v) *kurze Abwesenheit*: Anzahl wie oft im Jahr an inklusive drei bis exklusive sieben aufeinander folgenden Tagen weniger als fünf Liter Warmwasser pro Tag verbraucht wurden

und

- (vi) *lange Abwesenheit*: Anzahl wie oft im Jahr an sieben oder mehr aufeinander folgenden Tagen weniger als fünf Liter Warmwasser pro Tag verbraucht wurden

ermittelt werden.

Neben der hohen Dimensionalität der Daten ist auch das Rauschen der Zeitreihe ein Problem. Eine sehr einfache und häufig eingesetzte Methode [9] dieses zu reduzieren ist der *symmetric moving average*

$$\hat{m}(t) = \sum_{j=-q}^q w_j X_{t+j}, \quad t = q + 1, \dots, n - q,$$

wobei $q \in \mathbb{N}$, $w_j \in \mathbb{R}$, $\sum_{j=-q}^q w_j = 1$ und $w_j = w_{-j}$ gilt.

Je größer die Zahl q ist, desto geringer wird das Rauschen. Mit steigendem q erhöht sich jedoch auch der Informationsverlust. In Abbildung 4 ist die bereits bekannte Beispielzeitreihe über 7 Tage gemittelt zu sehen.

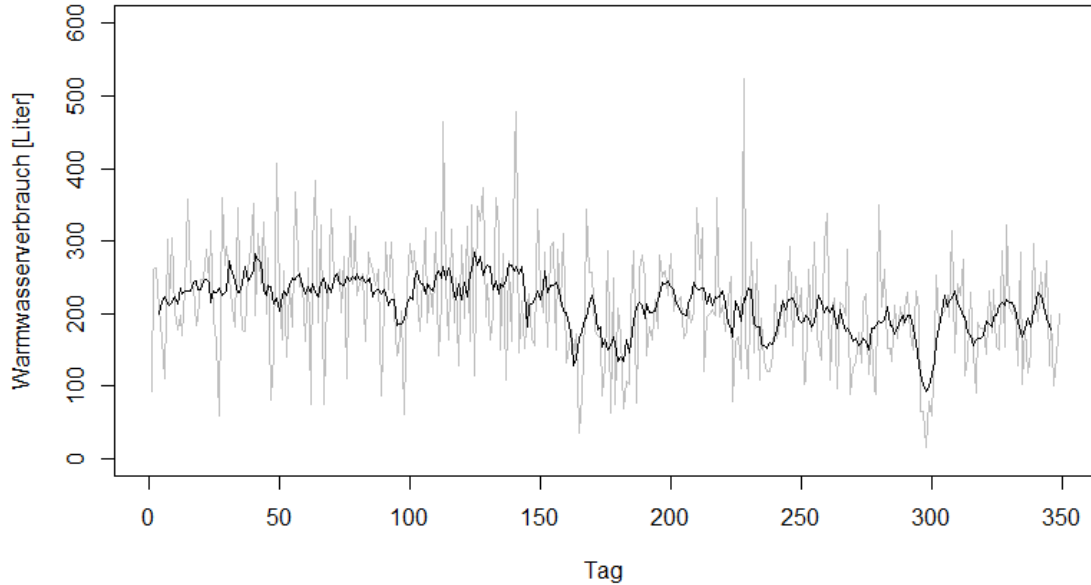


Abbildung 4.: Beispielzeitreihe des täglichen Warmwasserverbrauch über 349 Tage gemittelt über 7 Tage

Zuletzt gilt es die Skalierung des Inputs für die Clusteralgorithmen zu überlegen. Zwei in [9] empfohlene Skalierungsmethoden sind Standardisierung und Logarithmierung der Zeitreihen.

Sei $X = (x_1, \dots, x_n)$ wieder eine Zeitreihe mit Erwartungswert μ und Standardabweichung σ . Die *standardisierte* oder *z-transformierte* Zeitreihe $Z = (z_1, \dots, z_n)$ ergibt sich daraus gemäß

$$z_t = \frac{x_t - \mu}{\sigma}.$$

Der Erwartungswert μ_z von Z ist 0, da gilt

$$\mu_z = \sum_{t=1}^n \frac{z_t}{n} = \sum_{t=1}^n \frac{x_t - \mu}{\sigma n} = \frac{1}{\sigma} \left(\sum_{t=1}^n \frac{x_t}{n} - \mu \right) = \sigma(\mu - \mu) = 0.$$

Analog dazu ergibt sich die Standardabweichung σ_z gemäß

$$\sigma_z = \sqrt{\sum_{t=1}^n \frac{(z_t - \mu_z)^2}{n}} = \sqrt{\sum_{t=1}^n \frac{(x_t - \mu)^2}{n\sigma^2}} = \sqrt{\frac{\sigma^2}{\sigma^2}} = 1.$$

Vor dem *Logarithmieren* der Warmwasserverbrauchsdaten wird im vorliegenden Fall zu allen Werten x_t 1 addiert. Da alle x_t zwar nicht-negativ sind, aber sehr wohl null werden können, sorgt die Verschiebung dafür, dass der Logarithmus überall definiert ist. Die logarithmierte Version $L = (l_1, \dots, l_n)$ von X ergibt sich nun gemäß

$$l_t = \log(x_t + 1).$$

Eine weitere Möglichkeit eine reellwertige beschränkte Folge (f_1, \dots, f_n) zu *skalieren* ist sie durch ihren betragsmäßig maximalen Wert zu dividieren, also

$$f_i \mapsto \frac{f_i}{\max_{i \in \{1, \dots, n\}} |f_i|}. \quad (2.1)$$

Diese Vorgehensweise sorgt dafür, dass alle Werte im Intervall $[-1, 1]$ liegen. Sind negative Werte nicht erwünscht, wird vor dem Skalieren die Konstante

$$c = -\min\{0, \min_{i \in \{1, \dots, n\}} f_i\}$$

addiert. Folglich gilt

$$f_i \mapsto \frac{f_i + c}{\max_{i \in \{1, \dots, n\}} |f_i + c|}, \quad (2.2)$$

wodurch die *normierten* Werte im Bereich $[0, 1]$ liegen.

Sollen die Daten um den Mittelwert

$$\bar{f} = \frac{1}{n} \sum_{i=1}^n f_i$$

zentriert sein, so erfolgt sie Skalierung gemäß

$$f_i \mapsto \frac{f_i - \bar{f}}{\max_{i \in \{1, \dots, n\}} |f_i - \bar{f}|}.$$

Wahl des Abstandsbegriffs

Welcher Abstandsbegriff Ähnlichkeiten zwischen den Datenpunkten am Besten widerspiegelt, hängt maßgeblich von der gewählten Repräsentation der Zeitreihen und dem Anwendungsfall ab, also dem was man als ähnlich definieren möchte. Vorweg sei angemerkt, dass die im folgenden Kapitel eingeführten Abstandsbegriffe im Allgemeinen nicht metrisch sind. Die wohl bekannteste Metrik ist die euklidische Metrik gegeben durch

$$d((x_1, \dots, x_n), (y_1, \dots, y_n)) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}.$$

Sie eignet sich jedoch nur sehr bedingt für das Messen des Abstandes zwischen zwei Zeitreihen. Betrachtet man beispielsweise den Warmwasserkonsum zweier Familien mit sehr ähnlichem Nutzungserhalten in den Sommermonaten, kann es passieren, dass eine Familie im Juli für eine Woche nicht zu Hause ist, während die andere Familie im August für dieselbe Dauer verreist. Trotzdem wäre es wünschenswert, dass die Zeitreihen ihres Warmwasserverbrauchs in einer Gruppe landen. Auf Grund solcher Überlegungen wird im Zusammenhang mit Zeitreihen oft der sogenannte *Dynamic Time Warping (DTW)* Abstand verwendet, siehe [1], [6] und [15].

Grundsätzlich ist der Dynamic Time Warping Abstand auch für unterschiedlich lange und multivariate Zeitreihen definiert. Im Folgenden wird jedoch nur der Spezialfall gleichlanger,

reellwertiger, univariater Zeitreihen betrachtet. Sei dafür $X = (x_1, \dots, x_n)$ eine Testzeitreihe, welche mit einer Referenzzeitreihe $Y = (y_1, \dots, y_n)$ verglichen werden soll und $f : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}_0^+$ eine Funktion, welche die Distanz zwischen zwei Werten (x_i, y_j) angibt. Für diese Arbeit wird $f(x_i, y_j) = |x_i - y_j|$ gesetzt. Sei d eine auf den Indexmengen von X und Y definierte Funktion $d : \{1, \dots, n\} \times \{1, \dots, n\} \rightarrow \mathbb{R}_0^+$ mit $d(i, j) = f(x_i, y_j)$. Weiters sei Φ definiert als

$$\begin{aligned} \Phi : \{1, \dots, l\} &\longrightarrow \{1, \dots, n\} \times \{1, \dots, n\} \\ k &\longmapsto \begin{pmatrix} \Phi_x(k) \\ \Phi_y(k) \end{pmatrix}. \end{aligned}$$

Dabei ist $l \in \mathbb{N}$ mit $n \leq l \leq 2n$. Für Φ soll gelten

- (i) $\Phi(1) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ und $\Phi(l) = \begin{pmatrix} n \\ n \end{pmatrix}$,
- (ii) $\Phi_x(k+1) \geq \Phi_x(k)$ und $\Phi_y(k+1) \geq \Phi_y(k)$.

Anschaulich kann man sich Φ als Pfad durch eine $n \times n$ Abstandsmatrix A mit den Koeffizienten $a_{ij} = d(i, j)$ vorstellen. Die Bedingung (i) sorgt dafür, dass je die ersten und letzten Elemente der Zeitreihen miteinander verglichen werden, während (ii) Schleifen im Pfad verhindert.

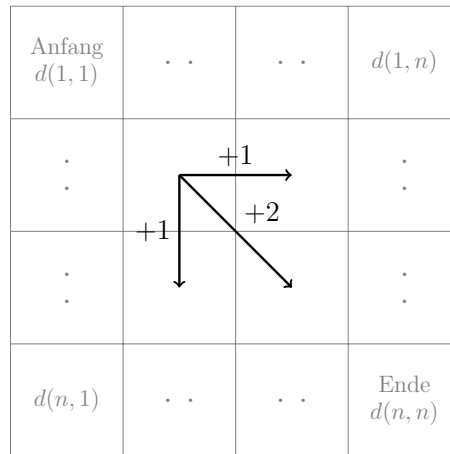


Abbildung 5.: Kosten für das Durchschreiten der Abstandsmatrix gemäß des Schrittmusters *symmetric2*

Nun wird für jeden Pfad Φ eine, von den verglichenen Zeitreihen abhängige, Kostenfunktion

$$d_\Phi(X, Y) = \sum_{k=1}^l d(\Phi_x(k), \Phi_y(k)) \frac{m_\Phi(k)}{M_\Phi}$$

definiert, mit $m_\Phi(k) = \Phi_x(k) - \Phi_x(k-1) + \Phi_y(k) - \Phi_y(k-1)$ für $k \geq 2$ und $m_\Phi(1) = \Phi_x(1) + \Phi_y(1)$ und einem vom Schrittmuster abhängigen *Normierungsfaktor* M_Φ .

Je kleiner der Wert von d_Φ desto besser ist der gefundene Pfad Φ . Jeder Schritt k wird gemäß eines *Schrittmusters* mit $m_\Phi(k)$ gewichtet. Das für diese Arbeit verwendete Schrittmuster, welches in der R-Implementierung des Algorithmus mit *symmetric2* bezeichnet wird, ist in Abbildung 5 illustriert.

Der Normierungsfaktor M_Φ wird vor allem beim paarweisen Vergleich von Tupel unterschiedlicher Länge relevant. Da die vorliegenden Zeitreihen alle gleichlang sind, wird in dieser Arbeit $M_\Phi = 1$ gesetzt. Zu beachten ist weiters, dass nicht jedes Schrittmuster normierbar ist. Für *symmetric2* wäre $M_\Phi = 2n$.

Der Abstand zwischen zwei Zeitreihen X und Y ist nun definiert als

$$d(X, Y) = \min_{\Phi} d_\Phi(X, Y). \quad (2.3)$$

Es sei an dieser Stelle darauf hingewiesen, dass es sich bei (2.3) um keine Metrik handelt. Zwar wird die Symmetrie bei dem verwendeten Schrittmuster offensichtlich erfüllt. Jedoch gilt weder die Dreiecksungleichung noch die Identitätsbedingung.

Beides kann man sich anhand eines Gegenbeispiels klar machen. Seien dafür $X = (0, 1, 1, 1)$, $Y = (0, 1, 0, 0)$ und $Z = (0, 1, 1, 0)$.

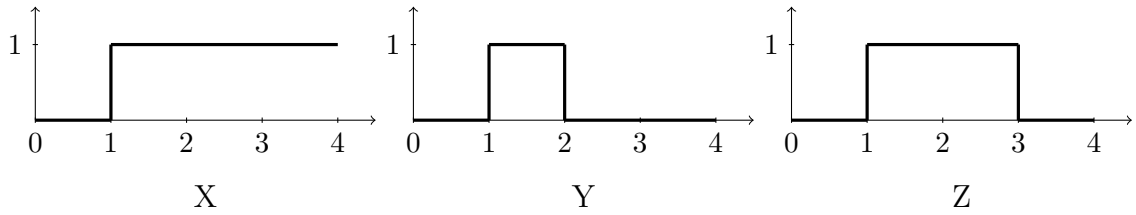


Abbildung 6.: Darstellung der Beispielzeitreihen, X, Y und Z

Mit der obigen Definition sind die zugehörigen Abstandsmatrizen

$$A_{X,Y} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}, \quad A_{X,Z} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \quad A_{Z,Y} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix},$$

wobei die roten Einträge jeweils einen günstigsten Pfad markieren. Dieses Beispiel zeigt, dass es im allgemeinen mehrere Wege geben kann deren Kostenfunktion denselben Wert hat. Es gilt $d(X, Y) = 2$, $d(X, Z) = 1$ und $d(Z, Y) = 0$. Dies steht sowohl im Widerspruch zur Dreiecksungleichung laut der $d(X, Y) \leq d(X, Z) + d(Z, Y)$ gelten sollte, als auch zu $d(Z, Y) = 0 \implies Z = Y$ und damit der Identitätsbedingung.

Das Beispiel verdeutlicht auch welche Zeitreihen als ähnlich bewertet werden.

In [7] wird ein Algorithmus vorgestellt, der die (Un-)Ähnlichkeit zweier Zeitreihen an deren komprimierter Form festmacht. Er baut auf der Theorie der Kolmogorov-Komplexität auf. Die folgenden Ausführungen basieren auf [11] und [7]. Um die Kolmogorov Komplexität definieren zu können, muss formal eingeführt werden was unter einem Programm zu verstehen ist. Dafür wird zunächst der Begriff der Turing Machine benötigt.

Definition 4. Eine Turing Machine T besteht aus dem Finite Control, welcher fähig ist eine Liste von Zellen, auch Tape genannt, zu manipulieren, indem er einen Zugriffszeiger, den Head, verwendet. Die beiden Richtungen des Tapes werden mit links und rechts bezeichnet. Der Finite Control kann sich in einer endlichen Menge Q von Zuständen befinden. Jede Zelle am Tape kann entweder 0, 1 oder ein Leerzeichen, auf englisch blank, B beinhalten. Eine endliche Menge von aneinander angrenzenden Zellen am Tape, welche mit 0 oder 1 gefüllt sind und durch Zellen in denen ein B steht getrennt werden, heißen binäre Strings. Die Zeit ist in diskrete, geordnete Schritte $0, 1, 2, \dots$ unterteilt, wobei 0 jenen Zeitpunkt angibt an dem die Maschine ihre Berechnungen startet. Zum Zeitpunkt 0 zeigt der Head auf eine ausgezeichnete Startzelle am Tape und der Finite Control ist in einem bestimmten Zustand q_0 . Zu Beginn beinhalten alle Zellen ein B , ausgenommen eines Strings, nach rechts von der Startzelle ausgehend. Diese Sequenz wird input genannt.

Die Maschine kann

- (i) eines der Elemente 0, 1 oder B in jene Zelle schreiben auf die der Head gerade zeigt und
- (ii) den Head eine Zelle nach links oder rechts bewegen, wobei der Befehl für ersteres als L und für zweiteres als R notiert wird.

In jedem Zeitschritt wird eine dieser Operationen ausgeführt bis die Maschine stehen bleibt. Am Ende jedes Zeitschrittes nimmt der Finite Control einen Status aus Q an. Die Maschine ist so konstruiert, dass sie in ihren Aktionen einer endlichen Menge von Regeln R gehorcht. Anhand derer wird aus dem momentanen Status des Finite Control, sowie dem Inhalt der Zelle auf die der Head gerade zeigt, die nächste von der Maschine auszuführende Operation bestimmt.

Diese Regeln sind im Format (q, s, a, p) wobei q der momentane Status des Finite Control ist, s ist das Symbol in der Zelle auf die der Head gerade zeigt, a bezeichnet die auszuführende Operation und p gibt an in welchem Zustand sich der Finite Control am Ende des Zeitschrittes befinden soll. Die Kombination der ersten zwei Elemente darf für verschiedene Tupel niemals dieselbe sein. Ist eine Kombination aus q und s nicht in der Menge der Regeln vorhanden, soll die Maschine in dem Fall stehen bleiben. Jener binäre String in dem sich die Zelle befindet auf welche der Head gerade zeigt wenn die Maschine stoppt, heißt Output. Sollte der Head auf ein B zeigen, wird der Output als 0 definiert.

Eine Turing Machine T kann eindeutig durch ihren Anfangsstatus q_0 , sowie ihre r -elementige Menge von Regeln

$$R = \{(q_i, s_i, a_i, p_i) \in Q \times \{0, 1, B\} \times \{0, 1, B, L, R\} \times Q \mid \{0, \dots, r\}\}$$

identifiziert werden. T kann also als Aneinanderreihung von Tupel gemäß

$$T = (q_0, s_0, a_0, p_0) \dots (q_{r-1}, s_{r-1}, a_{r-1}, p_{r-1})$$

dargestellt werden. Diese Schreibweise ist nicht eindeutig, da eine Turing Machine mehrere solche Darstellungen haben kann. Jedoch repräsentiert eine Sequenz höchstens eine Turing Machine, und genau dann eine Turing Machine, wenn keine zwei Tupel dieselbe Kombination an q_i, s_i enthalten. Jede dieser Sequenzen kann eindeutig als binärer String codiert

werden. Sei \mathcal{T} eine Liste aller binären Strings in lexikographischer Ordnung, durch die auf eben beschriebene Weise eine Turing Machine dargestellt wird.

Jede Turing Maschine T erzeugt eine Abbildung

$$\phi : \{(n_1, \dots, n_k) \in \mathbb{N}^k \mid k \in \mathbb{N}\} \rightarrow \mathbb{N},$$

wobei das Tupel (n_1, \dots, n_k) , als binärer String codiert, den Input der Turing Machine darstellt. Sollte ϕ an dieser Stelle definiert sein, wird der Input auf einen, ebenfalls als binären String codierten, Output abgebildet. Man sagt auch ϕ wird von T berechnet und nennt ϕ eine *partiell rekursive Funktion*. Die Abbildung

$$N : \mathcal{T} \rightarrow \mathbb{N}$$

ordnet jedem Element aus \mathcal{T} seinen Index gemäß der lexikographischen Ordnung zu.

Definition 5. Seien ϕ_1, ϕ_2, \dots die von den Turing Maschinen in \mathcal{T} berechneten partiell rekursiven Funktionen. Eine universelle Turing Machine U ist eine Turing Machine, welche das Verhalten jeder beliebigen Turing Machine T imitieren kann. Das heißt sei ϕ die von U berechnete partiell rekursive Funktion, $i = N(T)$ als binärer String codiert, $|i|$ die Länge des Strings, ϕ_i die von T berechnete partielle Funktion, x ein binärer String und

$$\langle i, x \rangle = \underbrace{11 \cdot \dots \cdot 11}_{|i|-\text{mal}} 0ix$$

die Verkettung der binären String i und x . Dann gilt

$$\phi(\langle i, x \rangle) = \phi_i(x).$$

Nun kann der Begriff der Komplexität und des Programms definiert werden.

Definition 6. Seien x, y und π binäre Strings. Jede partiell rekursive Funktion ϕ , gemeinsam mit π und y sodass $\phi(\langle y\pi \rangle) = x$ heißt eine Beschreibung von x . Sei $\{0, 1\}^*$ die Menge aller Binären Strings, dann ist die Komplexität K_ϕ in Abhängigkeit von y definiert als

$$K_\phi(x|y) = \min_{\pi \in \{0,1\}^*} \{|\pi| \mid \phi(\langle y, \pi \rangle) = x\},$$

und $K_\phi(x|y) = \infty$ falls es kein solches π gibt. Dabei wird π das Programm, welches x durch ϕ , gegeben y berechnet, genannt. Für den leeren String ϵ sei x^* definiert als

$$x^* = \operatorname{argmin}_{\pi \in \{0,1\}^*} \{|\pi| \mid \phi(\langle \epsilon, \pi \rangle) = x\}.$$

Anschließend kann überlegt werden, dass es eine partiell rekursive Funktion ϕ_0 gibt mit der für alle $x, y \in \{0, 1\}^*$ gilt

$$K_{\phi_0}(x|y) \leq K_\phi(x|y) + c_\phi,$$

wobei die Konstante c_ϕ nur von ϕ und nicht von x, y abhängt. Eine solche Funktion wird *additiv optimal* genannt.

Um die Aussage zu zeigen sei ϕ_0 jene Funktion, welche von einer universellen Turing Maschine U berechnet wird. Für alle $T_i \in \mathcal{T}$ gilt dann $\phi_0(\langle i, \langle y, p \rangle \rangle) = \phi_i(\langle y, p \rangle)$. Folglich gilt für alle $i \in \mathbb{N}$ mit $c_{\phi_i} = 2|i| + 1$

$$K_{\phi_0}(x|y) \leq K_{\phi_i}(x|y) + c_{\phi_i}$$

Schließlich können die Begriffe Kolmogorov Komplexität, Programm und Informationsdichte definiert werden

Definition 7. Sei ϕ_0 eine additiv optimale partiell rekursive Funktion, berechnet von einer Turing Maschine U . Für $x, y \in \{0, 1\}^*$ wird die bedingte Kolmogorov Komplexität gemäß

$$K(x|y) = K_{\phi_0}(x|y),$$

die unbedingte oder nur Kolmogorov Komplexität gemäß

$$K(x) = K_{\phi_0}(x|\epsilon)$$

definiert, wobei ϵ der leere String ist. Die Informationsdichte ist eine Funktion für welche

$$E(x, y) = \max\{K(x|y), K(y|x)\} + O(\log(\max\{K(x|y), K(y|x)\})),$$

gilt.

Die Informationsdichte erfüllt bis auf einen Fehlerterm $O(1)$ die Bedingungen für eine Metrik. Jedoch stellt diese Abstandsfunktion eine absolute Distanz zwischen zwei Strings dar. Um auch unterschiedlich lange Strings sinnvoll miteinander vergleichen zu können, definiert man durch

$$d_K(x, y) := \frac{K(x|y) + K(y|x)}{K(\langle x, y \rangle)} \quad (2.4)$$

eine Annäherung an eine normalisierte Informationsdichtefunktion.

Diese Funktion ist symmetrisch und genügt der Dreiecksungleichung bis auf einen kleinen Fehlerterm. Jedoch beträgt der Wertebereich $[\frac{1}{2}, 1]$, womit die Identitätseigenschaft, und damit die Bedingungen für eine Metrik, nicht erfüllt sind. Dabei ist $d_s(x, x) = \frac{1}{2}$ und $d_s(x, y) = 1$ wenn x und y überhaupt keine gemeinsamen Merkmale aufweisen.

Definition 8. Eine reellwertige Funktion $f(x, y)$ heißt upper semi-computable, wenn eine rationalwertige rekursive Funktion $g(x, y, t)$ existiert, welche

$$(i) \quad g(x, y, t) \leq g(x, y, t + 1) \text{ und}$$

$$(ii) \quad \lim_{t \rightarrow \infty} g(x, y, t) = f(x, y)$$

erfüllt. Sie heißt lower semi-computable, falls $-f(x, y)$ upper semi-computable ist und computable falls $f(x, y)$ sowohl lower semi-computable als auch upper semi-computable ist.

Sowohl $K(x)$ als auch $K(\langle x, y \rangle)$ und $E(x, y)$ sind upper semi-computable aber nicht lower semi-computable und somit nicht computable.

Da die Kolmogorov-Komplexität nicht berechenbar ist, wird $K(x)$ durch $C(x)$ angenähert. Dabei ist $C(x)$ die Größe des mittels eines Kompressionsalgorithmus komprimierten x . Je besser dieser Algorithmus für die jeweiligen Daten funktioniert, desto genauer ist die Annäherung an das theoretische Optimum $K(x)$. Analog dazu werden auch $C(\langle x, y \rangle)$ und $C(x|y)$ definiert. Der Abstand d_K kann also durch

$$d_C(x, y) := \frac{C(x|y) + C(y|x)}{C(\langle x, y \rangle)}$$

approximiert verwenden.

Bei gängigen, implementierten Kompressionsalgorithmen wie beispielsweise gzip, bzip2, zip oder rar ist die Möglichkeit eines Zusatzinputs zur Berechnung von $C(x|y)$ nicht standardmäßig vorgesehen. Daher wird anstatt von d_C das *Compression-Based Dissimilarity Measure (CDM)* verwendet. Der Abstand zwischen x und y wird dadurch definiert als

$$d(x, y) := \frac{C(\langle x, y \rangle)}{C(x) + C(y)}.$$

Der Wertebereich ist derselbe wie von d_K und d_C , jedoch wird die Dreiecksungleichung gar nicht mehr erfüllt, die Symmetrie hingegen bleibt erhalten. $d(x, y)$ ist nahe bei 1 wenn x und y kaum Gemeinsamkeiten haben, während $d(x, x) = \frac{1}{2}$.

Experimente zeigen, dass $d(x, y)$ einen ähnlich effektiven Abstandsbegriff wie $d_c(x, y)$ darstellt.

Wahl des Clusteralgorithmus

Nach [12] werden fünf verschiedene Arten von Clustering Methoden, nämlich *hierarchische*, *partitionierende*, *modellbasierte*, *dichtebasierte* und *graphenbasierte* Algorithmen unterschieden. Im Zusammenhang mit Zeitreihen werden nach [1] oft Algorithmen aus den ersten beiden Gruppen eingesetzt, weswegen die für diese Arbeit gewählten Algorithmen ebenfalls diesen Kategorien zuzuordnen sind. Außerdem wird in dem Artikel noch eine weitere Gruppe erwähnt, das *Multi-Step-Clustering*. Dabei werden mehrere Methoden miteinander kombiniert.

Zunächst wird auf den verwendeten *hierarchischen* Clusteralgorithmus eingegangen. Es gibt im Wesentlichen zwei Methoden einen hierarchischen Baum zu bilden, von unten oder oben. Erstere Methode wird *bottom-up* genannt, zweite heißt *top-down*. Die in dieser Arbeit verwendete Methode basiert auf einem bottom-up Ansatz. Dabei stellen die Blätter des Baums die einzelnen Datenpunkte dar. Zunächst besteht jeder Cluster aus einem Element. Nun werden iterativ jene Cluster C_i und C_j miteinander fusioniert, für welche die Funktion

$$(C_i, C_j) \mapsto \sum_{z_i \in C_i, z_j \in C_j} d(z_i, z_j) \tag{2.5}$$

minimal wird. Dabei ist d eine beliebige symmetrische Abstandsfunktion auf der Menge der Datenpunkte D und es werden alle Kombinationen von z_i, z_j in der obigen Summe betrachtet. Es können auch andere Funktionen zur Bestimmung der zu fusionierenden Cluster

verwendet werden. Die in dieser Arbeit verwendete Abbildungsvorschrift (2.5) wird *average linking* genannt. Die oben beschriebene Iteration terminiert, wenn nur noch ein Cluster übrig bleibt. Der so entstandene Baum wird *Dendrogramm* genannt. Aus dem Dendrogramm können Cluster gewonnen werden, indem der Baum bei einer gewissen Anzahl an Clustern abgeschnitten wird.

Für die Beschreibung des verwendeten partitionierenden Clusteralgorithmus sei D die Grundmenge der Datenpunkte. Die Idee hinter dem *Partitioning Around Medoids* (PAM) Algorithmus ist, für eine Partitionierung von D in k Cluster, eine geeignete Menge an Prototypen $P = \{p_1, \dots, p_k\}$ mit $p_i \in D$ zu finden. Jedes $z \in D$ wird nun dem nächstgelegenen p_i zugeordnet, woraus sich die Cluster ergeben. Dabei ist d eine beliebige symmetrische Abstandsfunktion. Die folgende Beschreibung orientiert sich an [12].

1. Initialisierungsphase: Suche sequentiell nach einer Initialisierung der Menge an Prototypen P bei der die einzelnen p_i möglichst zentral in der Grundmenge D liegen.
2. Tauschphase: Untersuche alle möglichen Vertauschungen von $p_i \in P$ mit $z_j \notin P$.
 - 2.1. Berechne für jeden möglichen Tausch von $p_i \in P$ mit einem $z_j \notin P$ die Kostenfunktion

$$T(p_i, z_j) = \sum_{x \notin (P \cup \{z_j\})} f(x, p_i, z_j),$$

dabei ist

$$f(x, p_i, z_j) = \begin{cases} 0 & \text{falls } d(x, p_i) > d_{\min}(x, P \setminus \{p_i\}) \\ & \text{und } d(x, z_j) > d_{\min}(x, P \setminus \{p_i\}) \\ d(x, z_j) - d(x, p_i) & \text{falls } d(x, p_i) = d_{\min}(x, P) \\ & \text{und } d(x, z_j) < d_{\min}(x, P \setminus \{p_i\}) \\ d(x, P \setminus \{p_i\}) - d(x, p_i) & \text{falls } d(x, p_i) = d_{\min}(x, P) \\ & \text{und } d(x, z_j) \geq d_{\min}(x, P \setminus \{p_i\}) \\ d(x, z_j) - d_{\min}(x, P) & \text{falls } d(x, p_i) > d_{\min}(x, P \setminus \{p_i\}) \\ & \text{und } d(x, z_j) < d_{\min}(x, P) \end{cases}$$

$$\text{und } d_{\min}(x, P) = \min_{p \in P} d(x, p).$$

- 2.2. Wähle $(\hat{p}, \hat{z}) = \operatorname{argmin}_{p \in P, z \notin P} T(p, z)$.
- 2.3. Stoppe, falls $T(\hat{p}, \hat{z}) \leq 0$, andernfalls tausche \hat{p} mit \hat{z} und beginne wieder bei Schritt 2.1.

Bestimmung der Clusterqualität

Es gibt mehrere Versuche der gefundenen Partitionierung je nach Qualität eine Zahl zuzuweisen. Dabei wird nach [1] zwischen *externen* und *internen* Cluster Indizes unterschieden. Bei ersteren werden die gefundenen Gruppen mit Klassifizierungen der Daten verglichen, welche durch andere Methoden bestimmt wurden. Interne Indizes kommen dann zum Einsatz, wenn keine Klassifizierung vorhanden ist, wie auch bei den vorliegenden Daten. Ein Indikator, der aus der Literatur [14] bekannt ist und zur Einschätzung der Güte einer Partitionierung verwendet wird, ist der Silhouettenkoeffizient. Für ein z aus dem Cluster C_i ist der *Silhouettenindex* gegeben durch

$$S(z) = \begin{cases} 0 & \text{für } \text{dist}(C_i, z) = \text{dist}_{C_i}(C_j, z) = 0 \\ \frac{\text{dist}_{C_i}(C_j, z) - \text{dist}(C_i, z)}{\max\{\text{dist}(C_i, z), \text{dist}_{C_i}(C_j, z)\}} & \text{sonst.} \end{cases} \quad (2.6)$$

Dabei ist $\text{dist}(C_k, z)$ für einen beliebigen Cluster C_k und eine Abstandsfunktion d definiert als

$$\text{dist}(C_k, z) = \frac{1}{n_{C_k}} \sum_{x \in C_k} d(x, z).$$

Der in der Formel (2.6) vorkommende Ausdruck $\text{dist}_{C_i}(C_j, z)$ ist definiert als

$$\text{dist}_{C_i}(C_j, z) = \min_{C_j \neq C_i} \text{dist}(C_j, z).$$

C_j ist der nächstgelegene Cluster von $z \in C_i$. Der Silhouettenkoeffizient einer Partitionierung $\{C_1, \dots, C_k\}$ der Menge D wird durch das Arithmetische Mittel der Silhouettenindizes aller Elemente der Grundmenge D ermittelt.

Es sei an der Stelle angemerkt, dass die Symmetrie aller verwendeten Abstandsbegriffe eine notwendige Voraussetzung ist, damit die Definition des Silhouettenkoeffizientens sinnvoll ist.

2.3. Praktische Umsetzung

Die in Abschnitt 2.2 vorgestellten Methoden wurden in der praktischen Ausführung auf mehrere Arten miteinander kombiniert. In Tabelle 2 sind für jeden Abstandsbegriff alle verwendeten Darstellungsmethoden markiert. Wurden die Zeitreihen mehrfach bearbeitet, ist der zuerst ausgeführte Schritt jener, der zuerst aufgelistet ist.

Für alle betrachteten Kombinationen wurde eine Partitionierung in 2 bis 10 Cluster mit den in Abschnitt 2.2 beschriebenen Clusteralgorithmen Partitioning Around Medoids und hierarchisches Clustern vorgenommen. Weiters wurde für jede Partitionierung und jeden Datenpunkt der jeweilige Silhouettenindex, sowie der durchschnittliche Silhouettenkoeffizient für die Partitionierung berechnet. All dies wurde an drei verschiedenen Samples mit 100, 300 und 600 Datenpunkten durchgeführt. Der R Code zu den Versuchen befindet sich im Anhang.

	DTW	CDM	Euklid	Manhattan
unverarbeitet		x		
moving average	x	x		
normiert-zentriert		x		
PAA	x			
moving average, logarithmiert	x	x		
moving average, normiert-zentriert	x	x		
moving average, z-transformiert	x	x		
normiert-zentriert, PAA	x			
moving average, Ausschnitt	x			
Ausschnitt, PAA	x			
moving average, logarithmiert, Ausschnitt	x			
features, verschoben-normiert			x	x

Tabelle 2.: Kombinationen von Abständen und Darstellungsmethoden, welche zur Auffindung von Cluster benutzt werden

3. Optimierung des Heizverhaltens

Um einen Überblick über die im Folgenden verwendeten Bezeichnungen zu behalten, werden diese in Tabelle 3 zusammengefasst, jedoch teilweise erst später genauer definiert.

Bezeichnung	Wertebereich	Beschreibung
t	\mathbb{N}	Zeiteinheiten in Minuten
Θ	\mathbb{N}	Zeiteinheiten in 2 Stunden
$b(s)$	$(0, b_{max}]$	Bakterienkonzentration im Speicher zum Zeitpunkt s
b_{max}	\mathbb{R}^+	Maximal erlaubte Bakterienkonzentration im Speicher
b_0	$(0, b_{max}]$	Bakterienkonzentration im Frischwasser
$T(s)$	$(T_0, 90)$	Temperatur in °C im Speicher zum Zeitpunkt s
T_0	$(0, 90)$	Minimaltemperatur in °C im Speicher
T_u	$[0, T_0]$	Umgebungstemperatur
$h(s)$	$\{0, 50, 54, 58, 60, 70, 80\}$	Heiztemperatur in °C zum Zeitpunkt s
$d(h(t))$	$\{0, 111, 27, 6, 4, 2, 1\}$	Desinfektionsdauer in Minuten
L	\mathbb{R}^+	Fassungsvermögen des Speichers in Liter
$l(s)$	$[0, L)$	ausströmendes Wasser in Litern im Segment s
$\lambda(s)$	$[1, \infty)$	Bakterienwachstumsrate pro Minute im Segment s
$p(s)$	$\mathbb{N} \cup \{0\}$	Anzahl der Taps finden im Segment s

Tabelle 3.: Bezeichnung und Beschreibung der wichtigsten Variablen und Parameter für das Modell

Die Zeit wird in diesem Kapitel, wenn nichts Gegenteiliges vermerkt ist, grundsätzlich in diskreten Abschnitten angegeben. Wenn also im Folgenden vom Zeitpunkt s die Rede ist, müsste korrekt formuliert werden, der Zeitpunkt am Ende des s -ten Zeitsegments. Die Länge von s beträgt entweder eine Minute, in diesem Fall wird die Zeit mit t statt s bezeichnet, oder zwei Stunden, wofür der Variable Θ verwendet wird. Weiters sei bemerkt, dass in der Physik die Temperatur meist mit Kelvin angegeben wird, in dieser Arbeit aber °C als Einheit dient. Dies hat einerseits den Grund, dass die Bakterienwachstumsraten in der verwendeten Literatur ebenfalls in Abhängigkeit von °C beschrieben sind, andererseits verhindert die Beschäftigung mit Warmwasser das Auftreten von etwaigen negativen Temperaturen. Ein Tap bezeichnet einen Zapfvorgang, bei dem Warmwasser aus dem Speicher entnommen wird und Frischwasser nachströmt. Die Bakterienkonzentration wird in Vielfachheit der Anfangskonzentration angegeben, da letztere noch nicht bekannt ist. Des Weiteren beträgt die maximale im Modell zulässige Desinfektionstemperatur aus Energieeffizienzgründen 70 °C.

Das Wachstum von Legionellen ist stark temperaturabhängig wie ausgeführt in [2] und [16]. Dabei gibt es, wie in Abbildung 7 dargestellt, einen Temperaturbereich in dem die Bakterienkonzentration stagniert, einen Bereich in dem sie steigt und einen in dem sie fällt.

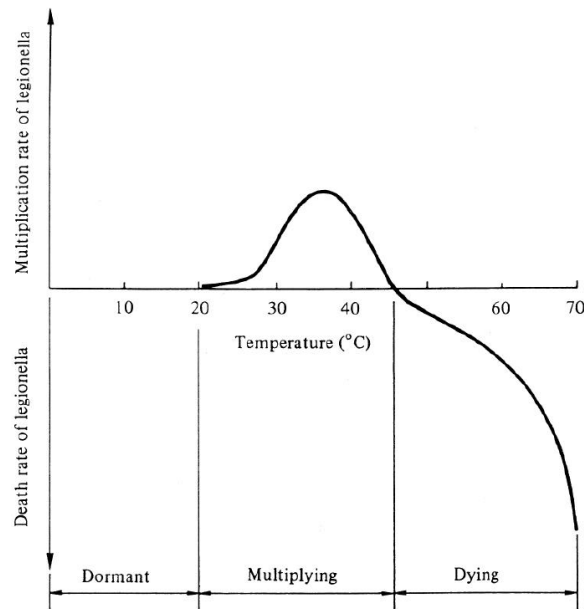


Abbildung 7.: Wachstumsrate von Legionellen in Leitungswasser abhängig von der Temperatur nach [2]

Über den genauen Verlauf des Sterbeprozesses ist in der Literatur jedoch sehr wenig zu finden. Das Ziel besteht darin die tatsächliche Legionellenkonzentration unter oder auf dem Level der modellierten zu halten. Daher wird keine kontinuierliche Funktion verwendet um den Sterbeprozess zu beschreiben, sondern angenommen, dass sich die Anzahl an Legionellen um 90 Prozent verringert, wenn das Wasser für eine bestimmte Zeitspanne auf eine gewisse Temperatur gemäß Tabelle 4 aufgeheizt wird.

Temperatur	Reduktion um 90% nach
50 °C	111 min
54 °C	27 min
58 °C	6 min
60 °C	4 min
70 °C	2 min
80 °C	1 min

Tabelle 4.: Benötigte Minuten und Temperatur nach [2], um die Konzentration von Legionella pneumophila in Leitungswasser um 90 Prozent zu verringern, Minutenanzahl gerundet

Außerhalb dieser diskreten Heizphasen wird die Bakterienkonzentration im Modell als konstant definiert, falls die Temperatur über 46 °C liegt [2]. Auch wenn ein Tap die Desinfektionsphase unterbricht, wird davon ausgegangen, dass keine Bakterien abgetötet werden. Weiters wird die Zeit vernachlässigt, welche das Speichergerät benötigt, um das Wasser auf die gewünschte Temperatur zu bringen.

3.1. Modellierung der Bakterienkonzentration und des Heizverhaltens eines Warmwasserspeichers in Abhängigkeit vom Warmwasserverbrauch

Die Legionellenkonzentration b in einem Warmwasserspeicher wird vor allem von zwei Faktoren beeinflusst: der Temperatur und der Vermischung mit einfließendem Wasser, in welchem die Bakterienkonzentration einen konstanten Grundwert von b_0 hat. Zunächst wird angenommen, dass sowohl die Temperatur als auch die Konzentration der Bakterien im Warmwasserspeicher homogen sind.

Berechnung der Temperatur zum Zeitpunkt t

Im Wesentlichen können drei Temperaturbereiche unterscheiden werden:

- (i) Die *Desinfektionsphase*, wobei die Desinfektionstemperatur in dieser Zeit konstant gehalten wird,
- (ii) die *Abkühlphase*, wobei der Temperaturverlust auf die Vermischung mit kälterem, nachströmendem Wasser und Wärmeabgabe an die Umgebung zurückzuführen ist
- (iii) und jener Zeitraum in dem die Minimaltemperatur T_0 angenommen wird.

In der Abkühlungsphase, ergibt sich die Temperatur zum Zeitpunkt t aus der Temperatur zum Zeitpunkt $t - 1$ gemäß

$$T_a(t) = e^{-k} \left((T(t-1) \frac{L-l(t)}{L} + T_0 \frac{l(t)}{L}) - T_u \right) + T_u - \left(T(t-1) \frac{L-l(t)}{L} + T_0 \frac{l(t)}{L} \right).$$

Dies leitet sich aus dem Newtonschen Abkühlungsgesetz ab, das die Änderung der Temperatur zu einem beliebigen reellen Zeitpunkt x gemäß

$$\frac{d\tilde{T}}{dx}(x) = -k(\tilde{T}(x) - T_u) \quad (3.1)$$

angibt, wobei k eine noch zu bestimmende Konstante und T_u die Umgebungstemperatur darstellt. Durch Lösen der Differentialgleichung ergibt sich für die Temperatur

$$\tilde{T}(x) = e^{-kx+c} + T_u.$$

Mit der Anfangsbedingung $\tilde{T}(0) = T(t-1)$ und folglich der Konstanten $e^c = T(t-1) - T_u$ erhält man

$$\tilde{T}(x) = e^{-kx}(T(t-1) - T_u) + T_u. \quad (3.2)$$

In dem Modell werden Taps höchstens zu ganzzahligen Zeiteinheiten, in dem Fall Minuten, erlaubt. Daher verändert sich die Temperatur im Intervall $(0, 1)$ gemäß (3.2). Der Temperaturverlust in einer diskreten Zeiteinheit beträgt also

$$\tilde{T}(1) - \tilde{T}(0) = e^{-kx}(T(t-1) - T_u) + T_u - T(t-1).$$

Da eventuell stattgefundenen Taps auch noch berücksichtigt werden müssen, wird $T(t-1)$ durch

$$T(t-1) \frac{L-l(t)}{L} + T_0 \frac{l(t)}{L}$$

ersetzt.

Die Konstante k wird auf Basis der Dämmeigenschaften des Warmwasserspeichers bestimmt. Dafür wird ein zylindrischer Warmwasserspeicher mit innerem Radius R_i und Höhe H angenommen. Zur Vereinfachung der Berechnungen seien Grund- und Deckfläche im Modell perfekt isoliert. Die Mantelfläche sei mittels einer 10 cm dicken Schicht exkludiertem Polystyrol-Hartschaum gedämmt, womit sich ein größerer äußerer Radius R_a ergibt. Die spezifische Wärmeleitfähigkeit λ_w des Dämmstoffes beträgt gemäß [4] zwischen 0.023 und 0.046 W/(m·K) und wird daher mit 0.035 W/(m·K) angenommen. Um k zu berechnen wird nun ausgenutzt, dass sich der Wärmestrom \dot{Q} , also die zeitliche Änderungsrate der Wärmeenergie, einerseits durch die Formel

$$\dot{Q} = cm \frac{dT}{dt} \tag{3.3}$$

und andererseits durch

$$\dot{Q} = qA \tag{3.4}$$

beschreiben lässt. Dabei bezeichnet c in (3.3) die spezifische Wärmekapazität, m die Masse des auskühlenden Stoffes und $\frac{dT}{dt}$ die zeitliche Änderung der Temperatur. Für Wasser ist $c = 4.18$ kJ/(kg·K) und m genau das Fassungsvermögen des Speichers. Die Wärmestromdichte q in der Formel (3.4) ist definiert durch

$$q = -\lambda_w \nabla T.$$

Da die Grund- und Deckfläche des Zylinders als perfekt isoliert angenommen werden, vereinfacht sich der Temperaturgradient ∇T zur radialen Änderung der Temperatur $\frac{dT}{dr}$. Damit kann 3.4 als

$$\dot{Q} = -\lambda_w A \frac{dT}{dr}$$

geschrieben werden, wobei die Mantelfläche $A = 2\pi rH$ vom Radius r und der Höhe H abhängen. Durch Umformung und Substitution ergibt sich

$$\int_{R_i}^{R_a} \frac{\dot{Q}}{-\lambda_w 2\pi H r} dr = \int_T^{T_u} dT$$

und wegen $T - T_u \geq 0$ gilt weiters

$$\dot{Q} = \frac{\lambda_w 2\pi H}{\ln(R_a) - \ln(R_i)} (T - T_u).$$

Da Wärme jedoch grundsätzlich vom wärmeren zum kälteren Medium übertragen wird, ist der durch (3.4) gegebene Wärmestrom positiv, denn über die Zeit entweicht Wärme aus dem Warmwasserspeicher. Die Formel (3.3) beschreibt hingegen die Änderung der Wärmeenergie aus Sicht des Speichers, weshalb diese negativ ist. Es gilt also

$$\dot{Q} = cm \frac{dT}{dt} = -qA. \quad (3.5)$$

Nach Umformung ergibt sich die Temperaturänderung gemäß

$$\frac{dT}{dt} = -\frac{\lambda_w 2\pi H}{cm(\ln(R_a) - \ln(R_i))} (T - T_u).$$

Durch Vergleich mit dem Newtonschen Abkühlungsgesetz (3.1) kann k nun direkt abgelesen werden. Es gilt noch zu beachten, dass c und λ_w in unterschiedlichen Einheiten angegeben sind, weshalb sich k gemäß

$$k = \frac{\lambda_w 2\pi H}{c 10^3 m (\ln(R_a) - \ln(R_i))}$$

ergibt.

Außerhalb der Abkühlungsphase bleibt die Temperatur konstant auf $h(t)$ beziehungsweise T_0 . Es ist zu beachten, dass die Temperatur in Desinfektionsphasen für eine gewisse Zeit $d(h(t))$ gehalten wird. Dabei nimmt $d(h(t))$ Werte gemäß Tabelle 4 an, beziehungsweise wird $d(0) = 0$ gesetzt. Die Funktion

$$\text{status}(t) = \begin{cases} 0 & \text{wenn zum Zeitpunkt } t \text{ ein Tap stattfindet} \\ d(h(t)) & \text{wenn } h(t) \neq 0 \\ \max\{0, \text{status}(t-1) - 1\} & \text{sonst.} \end{cases}.$$

wird verwendet, um gemäß

$$\delta_h(t) = \begin{cases} 0 & \text{wenn } \text{status}(t) = 0 \\ 1 & \text{sonst} \end{cases}$$

anzugeben, ob gerade auf Desinfektionstemperatur geheizt wird. In dem Fall ist $\delta_h(t) = 1$ während sonst $\delta_h(t) = 0$ gilt.

Nun wird noch die Funktion

$$T_d(t) = \begin{cases} T_d(t-1) & \text{wenn } h(t) = 0 \\ h(t) & \text{sonst} \end{cases}$$

benötigt, die als Wert jene Desinfektionstemperatur annimmt, auf die das Wasser im Speicher zuletzt erwärmt wurde. Insgesamt ergibt sich die Temperatur gemäß

$$T(t) = \max\{(1 - \delta_h(t))T_a(t) + \delta_h(t)T_d(t), T_0\}. \quad (3.6)$$

Berechnung der Bakterienkonzentration zum Zeitpunkt t

Nur wenn der Abstand zwischen zwei Taps die benötigte Desinfektionsdauer nicht unterschreitet, sinkt die Legionellenkonzentration um 90 Prozent. Andernfalls erfolgt keine Reduktion derselben. Auch wenn ein Heizvorgang einen anderen unterbricht, wird die Desinfektion als nicht effektiv bewertet.

Die Funktion

$$\delta_{\text{heff}}(t) = \begin{cases} 1 & \text{wenn } \sum_{n=t}^{t+d(h(t))} p(n) = 0 \text{ und } \sum_{n=t+1}^{t+d(h(t))} h(n) = 0 \text{ und } h(t) > 0 \\ 0 & \text{sonst} \end{cases}$$

nimmt den Wert 1 an, wenn der Heizmodus $h(t)$ zum Zeitpunkt t zu einer 90 prozentigen Reduktion der Bakterienkonzentration führt und 0 wenn nicht. Wie bereits erwähnt, wird aus Sicherheitsgründen die Wachstumsrate der Bakterien oberhalb von 46 °C grundsätzlich als 1 angenommen. Eine Verminderung der Konzentration passiert also ausschließlich durch die Verdünnung mit Frischwasser und in den Desinfektionsphasen. Die Bakterienkonzentration verändert sich daher abhängig von der Temperatur in einer Minute um den Faktor

$$\lambda(t) = \begin{cases} 0.1 & \text{wenn } \delta_{\text{heff}}(t) = 1 \\ 1 & \text{wenn } T(t) > 46 \\ f(T(t)) & \text{sonst.} \end{cases}$$

$f(T(t))$ ergibt sich hierbei aus den Wachstumsraten in Tabelle 5 und der in [2] angegebenen Information, dass kein Wachstum bei einer Temperatur von über 46 °C stattfindet.

Temperatur	Wachstumsrate [1/h]
25 °C	1,084087452
30 °C	1,108941315
37 °C	1,188971356
42 °C	1,162168495

Tabelle 5.: Temperaturabhängige mittlere Wachstumsraten diverser Legionellenstämme in Wasser nach [16]

Unter der Annahme, dass die Bakterienkonzentration bei einer Temperatur von über 46 °C, außer bei thermischer Desinfektion, konstant bleibt, wird die Wachstumsrate pro Minute mittels einer linearen Funktion

$$f(T) = \frac{k_f T + d_f}{60}$$

angenähert. Dabei soll gelten, dass $f(37) = 1.188971356$ und $f(46) = 1$, woraus sich $k_f = -0.02099682$ und $d_f = 1.965854$ ergibt. Für Werte unter 37 °C ist diese Funktion nicht zur Approximation geeignet. Da es jedoch um die Bakterienkonzentration im Warmwasser geht, sei der Gültigkeitsbereich für das Modell hiermit auf Wassertemperaturen über 37 °C eingeschränkt.

Die Zeit welche zum Ein- und Ausströmen des Wassers, beziehungsweise zum Aufheizen auf die Wunschtemperatur außerhalb der Desinfektionsphasen nötig ist, wird vernachlässigt. Daher kann angenommen werden, dass am Anfang einer Minute die Wunschtemperatur erreicht wird und ein Tap ebenfalls zu Beginn der Minute stattfindet. Die Bakterienkonzentration zum Zeitpunkt $t + 1$, also am Ende der Minute t , wird demnach gemäß

$$b(t) = \left(b(t-1) \frac{L - l(t)}{L} + b_0 l(t) \right) \lambda(t)$$

berechnet.

Berechnung des Energieaufwands zum Zeitpunkt t

Um eine Änderung von ΔT zu bewirken, wird die Energie $Q = mc\Delta T$ benötigt. Wobei m die Masse und c die Wärmekapazität darstellt. Außerhalb der Desinfektionsphasen muss innerhalb eines Zeitsegments nur geheizt werden, falls die Temperatur auf einen Wert unter T_0 abzusinken droht. Analog dazu muss während einer Desinfektionsphase die an die Umgebung abgegebene Energie ausgeglichen werden. Es ergibt sich der durch kontinuierliche Auskühlung verursachte Temperaturverlust ΔT_a also gemäß

$$\Delta T_a = \begin{cases} \max\{T_0 - (1 - T_a(t)), 0\} & \text{für } \delta_h(t) = 0 \\ e^{-k}(T_d(t) - T_u) & \text{für } \delta_h(t) = 1. \end{cases}$$

Zusätzlich ist noch jene Temperaturdifferenz

$$\Delta T_h(t) = \begin{cases} 0 & \text{wenn } h(t) = 0 \\ h(t) - T(t-1) & \text{sonst} \end{cases}$$

zu berücksichtigen, die beim Aufheizen auf Desinfektionstemperatur überwunden werden muss.

Insgesamt beträgt die aufzuwendende Energie zum Zeitpunkt t also

$$Q(t) = mc(\Delta T_a(t) + \Delta T_h(t)).$$

Das Ziel ist es nun, eine Funktion h zu finden, welche zu einer Minimierung von

$$\int_0^t Q(\tau) d\tau$$

führt. Diese muss nicht eindeutig sein. Zusätzlich soll die Bakterienkonzentration $b(t)$ zu keinem Zeitpunkt t einen Grenzwert von b_{\max} mal der Anfangskonzentration b_0 überschreiten.

3.2. Das implementierte Modell

Durch Reduktion der zeitlichen Auflösung wird das Modell noch weiter vereinfacht, um den Rechenaufwand für den Computer möglichst gering zu halten. Es bietet sich eine Unterteilung in Zeitsegmente Θ mit einer Länge von je zwei Stunden an. Einerseits ist diese

Zeitspanne größer als jede, die für jegliche Desinfektionen benötigt wird und andererseits ist ein Tag gut in 12 Teile mit je zwei Stunden teilbar. Trotzdem sind die Veränderungen der Bakterienkonzentration und Temperatur noch relativ gering.

In jedem Abschnitt laufen alle Ereignisse in der gleichen Reihenfolge ab: Heizen auf Desinfektionstemperatur - Taps - konstante Temperatur.

Die ersten beiden Schritte sind optional. Findet mehr als ein Tap in einem Zeitsegment statt, wird die Menge an abgeflossenem Wasser dementsprechend erhöht.

Temperatur am Ende eines Zeitsegments Θ

Um die Temperatur am Ende eines Zeitsegments Θ zu berechnen, wird Formel (3.6) modifiziert. Die relative kontinuierliche Temperaturabnahme pro Minute wird mit einem von der Temperatur unabhängigen Wert \tilde{T}_{abk} approximiert, jener Temperaturdifferenz, die sich für eine Wassertemperatur von 50 °C am Anfang des Zeitsegments Θ ergibt.

Diese Rate ist sehr klein und fällt im Vergleich zu der Temperaturreduktion bedingt durch nachströmendes Wasser kaum ins Gewicht. Relevant wird die an die Umgebung abgegeben Energie eigentlich erst, wenn mehrere Tage kein Wasser aus dem Speicher entnommen wird.

Wird nicht geheizt, beträgt die Ausgangstemperatur $T(\Theta - 1)$, andernfalls $h(\Theta)$. Nach dem eventuell stattfindenden Tap ergibt sich T_p gemäß

$$T_p(\Theta) = \left((1 - \tilde{\delta}_h(\Theta))T(\Theta - 1) + h(\Theta) \right) \frac{L - l(\Theta)}{L} + l(\Theta)T_0.$$

Da jegliche Desinfektionsvorgänge vor Ende eines Zeitsegments abgeschlossen sind, vereinfacht sich $\delta_h(t)$ zu

$$\tilde{\delta}_h(\Theta) = \begin{cases} 0 & \text{wenn } h(\Theta) = 0 \\ 1 & \text{sonst.} \end{cases}$$

Um eine Regulation der Temperatur zu ermöglichen, wird noch ein nicht-negativer Korrekturterm T_{korrr} addiert. Insgesamt ergibt sich die Temperatur gemäß

$$T(\Theta) = T_p + T_{korrr} - \tilde{T}_{abk}. \quad (3.7)$$

Bakterienkonzentration am Ende eines Zeitsegments Θ

Um die Legionellenkonzentration am Ende eines Segments berechnen zu können, braucht es wieder eine Funktion $\tilde{\delta}_{h_{\text{eff}}}$, die angibt, ob ein Heizvorgang zu einer Desinfektion geführt hat. Zwar werden, um das Modell zu vereinfachen, alle Taps eines Segments zu einem Tap zusammengefasst, der stets nach der Heizphase stattfindet. Trotzdem wird die Regel eingeführt, dass ein Desinfektionsvorgang nur dann als nicht unterbrochen gilt, wenn die Anzahl an Taps in einem Segment einen gewissen Schwellenwert nicht übersteigt. $\tilde{\delta}_{h_{\text{eff}}}$ ergibt sich also gemäß

$$\tilde{\delta}_{h_{\text{eff}}}(\Theta) = \begin{cases} 1 & \text{wenn } p(\Theta) < \lfloor 120/d(h(\Theta)) \rfloor \text{ für } d(h(\Theta)) \neq 0 \\ 0 & \text{sonst.} \end{cases}$$

Die gemäß

$$b_h(\Theta) = b(\Theta - 1)(1 - \tilde{\delta}_{h_{\text{eff}}}(\Theta)) + \tilde{\delta}_{h_{\text{eff}}}(\Theta) * 0.1$$

definierte Funktion b_h gibt nun die Bakterienkonzentration vor dem Tap an. Abhängig von einer etwaig stattfindenden Desinfektion und deren Dauer, würde die Konzentration der Bakterien $b(\Theta)$ am Ende von Θ

$$b(\Theta) = \left(b_h \frac{L - l(\Theta)}{L} + l(\Theta)b_0 \right) \tilde{\lambda}^{(120 - d(h(\Theta)))} \quad (3.8)$$

betragen, wobei

$$\tilde{\lambda}(t) = \begin{cases} 1 & \text{wenn } T(\Theta) > 46 \\ f(T(\Theta)) & \text{sonst.} \end{cases}$$

Da ein längeres Wachstumsintervall nur zu einer höheren Konzentration führt und der durch folgende Vereinfachung entstehende Fehler relativ gering ist, wird im Term (3.8) $d = 0$ angenommen. Da die Temperatur am Ende eines Zeitsegments jedenfalls kleiner gleich jener am Anfang ist und die Wachstumsrate mit fallender Temperatur monoton steigt, wird in der der Formel für $\tilde{\lambda}$ die $T(\Theta)$ also am Ende des Zeitsegments verwendet. Insgesamt ergibt sich die Bakterienkonzentration gemäß

$$b(\Theta) = \left(b_h \frac{L - l(\Theta)}{L} + l(\Theta)b_0 \right) \tilde{\lambda}^{120}.$$

Berechnung des gesamten Energieaufwandes

Die insgesamt überwundene Temperaturdifferenz ergibt sich nun aus der Summe der Temperaturdifferenzen jedes Zeitsegments Θ gemäß

$$\Delta T = \sum_{\Theta} (T_{\text{korrr}}(\Theta) + h(\Theta)).$$

Dann ergibt sich die nötige Energie Q um L Liter Wasser um ΔT zu erwärmen gemäß

$$Q = \Delta T L c.$$

Dabei ist c wiederum die spezifische Wärmekapazität von Wasser.

3.3. Abschätzung der durch Annahmen und Vereinfachung entstandenen Fehler

Im folgenden wird auf die Fehler des implementierten Modells eingegangen. Die Annahme, welche das Modell vermutlich am meisten von der Realität unterscheidet ist gleichzeitig jene, deren Auswirkungen am schwierigsten abzuschätzen sind. In Wirklichkeit sind Legionellen nämlich nicht frei im Wasser gelöst, sondern lagern sich an den Wänden des Gefäßes ab, siehe [2]. In welchem Maß sich diese bei einem Zapfvorgang durch im Wasser herrschende Scherkräfte ablösen, müsste noch weiter untersucht werden. Jedenfalls findet die Verringerung der Bakterienkonzentration durch Verdünnung nicht so statt wie im Modell angenommen.

Weiters wird von einer homogenen Verteilung der Temperatur ausgegangen. Da wärmeres Wasser in der Regel jedoch eine geringere Dichte hat als kälteres, entspricht das vermutlich auch nicht der Realität. Die Wachstumsrate der Legionellen könnte also je nach Position der Ablagerung unterschiedlich sein. Außerdem sei angemerkt, dass die Legionellen in den Rohren, welche vom Warmwasserspeicher zum Verbraucher führen, in dem vorliegenden Modell komplett außer Acht gelassen werden.

Die gerade genannten Schwächen des Modells sind besonders deswegen problematisch, da sie tendenziell zu einer Unterschätzung der Legionellenkonzentration führen. Im Gegensatz dazu können die im folgenden diskutierten Ungenauigkeiten eher vernachlässigt werden, da sie entweder sehr gering sind oder bewirken, dass die Bakterienkonzentration im Modell höher ist als sie unter gleichen Voraussetzungen in der Realität wäre.

Im Bereich zwischen 37 °C und 46 °C stehen nur drei empirisch ermittelte Werte für die Wachstumsrate zur Verfügung. Diese werden durch eine lineare Funktion approximiert, welche durch den Messwert bei 37 °C und 46 °C geht. Der Wert bei 42 °C wird dadurch um zirka 0.078 pro Stunde unterschätzt. Bei gleicher Anfangskonzentration b_0 ergibt sich bei 42 °C mit der im Modell verwendeten Wachstumsrate eine Konzentration von etwa 1.2 b_0 , während sie mit der empirisch ermittelten Wachstumsrate rund 1.4 b_0 betragen würde. Im Gegensatz dazu wird bei über 46 °C außerhalb von Desinfektionsphasen eine Stagnation des Bakterienwachstums angenommen, obwohl in Wirklichkeit eine Reduktion der Population stattfindet, siehe [2].

Eine weitere Vereinfachung besteht darin, dass die durch Auskühlung bedingte Temperaturabnahme, unabhängig von der Wassertemperatur, als jener Wärmeverlust angenommen wird, der entstehen würde, betrüge die Anfangstemperatur im Speicher 50 °C. Für einen Warmwasserspeicher mit einem Fassungsvermögen von 100 Litern bei einer Umgebungstemperatur von 18 °C entspricht dies einem Verlust von etwa 0.012 °C. Geht man hingegen von einer Wassertemperatur von 37 bzw. 80 °C aus, beträgt der Wärmeverlust zirka 0.006 bzw. 0.021 °C. Da diese Unterschiede sehr klein sind im Vergleich zur sonstigen Genauigkeit des Modells, können sie vernachlässigt werden.

Die Zusammenfassung aller Zapfvorgänge zu einem großen Zapfvorgang in einem Zeitsegment Θ hat ebenfalls einen Einfluss auf die Bakterienkonzentration zu Beginn des nächsten Segments. Betrachtet man Beispielsweise ein 120 minütiges Zeitintervall wo im ersten Fall ein Zapfvorgang in Minute 1 stattfindet bei dem 60 Liter entnommen werden. Im zweiten Fall fließen jeweils in Minute 1, 40 und 80 je 20 Liter ab. Die Anfangstemperatur betrage in beiden Fällen $45\text{ }^{\circ}\text{C}$, die Temperatur des nach strömenden Wassers $15\text{ }^{\circ}\text{C}$ und die Bakterienkonzentration zu Beginn sei b_0 . Ohne Berücksichtigung des kontinuierlichen Temperaturverlusts, wie bereits gezeigt ist dieser sehr gering, ergibt sich nach 120 Minuten für den ersten Fall eine Konzentration von $1.46 \cdot b_0$ bzw. $1.20 \cdot b_0$ im zweiten Fall. Hier wird die Bakterienkonzentration vom Modell überschätzt. Anders verhält es sich wenn man als abfließende Wassermengen 90 Liter im ersten Fall und jeweils 30 Liter im zweiten Fall wählt. Jetzt ergeben sich Legionellenkonzentrationen von $0.98 \cdot b_0$ bzw. $1.06 \cdot b_0$.

Auf Grund der wesentlich größeren, bereits oben diskutierten Vereinfachungen, ist es mittels des vorliegenden Modells jedoch ohnehin nicht möglich, eine auf die Kommastelle genaue Konzentration der Bakterien anzugeben. Bestenfalls können grobe Zusammenhänge bezüglich des Wasserverbrauchsverhaltens, der Heizstrategie und der Legionellenpopulation wiedergegeben werden. Dabei fallen kleine Abweichungen bei der Aufteilung der Zapfvorgänge, der Wachstumsrate oder der Temperatur kaum ins Gewicht, so lange die Bakterien als frei gelöst angenommen werden.

4. Ergebnisse

4.1. Evaluierung der Partitionen des Warmwasserverbrauchs

Bereits die Arbeiten von [3], [10] und [5] beschäftigen sich mit dem Erstellen von Wasserverbrauchsprofilen mittels Unsupervised Machine Learning Methoden. Im Gegensatz zum vorliegenden Datensatz steht jedoch der Warmwasserverbrauch in stündlicher oder minütlicher Auflösung zur Verfügung. Weiters werden in diesen Arbeiten Profile des wöchentlichen oder täglichen Verbrauchs erstellt, nicht aber des jährlichen. Im Gegenteil, wird in [3] der globale Trend sogar vor dem Clustern extrahiert und fließt somit nicht in die gefundenen Verbrauchsprofile ein. Im Gegensatz dazu wird in dieser Arbeit unter anderem versucht, Gemeinsamkeiten im Trend zu erkennen, welcher mittels des moving average berechnet wird. Aus diesem Grund unterscheidet sich die Herangehensweise im vorliegenden Fall speziell bei der Repräsentation der Zeitreihen als auch den verwendeten Abstandbegriffen und Cluster Algorithmen von den oben angeführten Arbeiten.

In [3] werden die Zeitreihen nach dem Extrahieren des globalen Trends durch eine auf Fourieranalyse basierenden Darstellung repräsentiert. Das Clustern erfolgt anschließend mittels der für jene Repräsentation adaptierten Versionen der Algorithmen K-Means und Expectation Maximization. Ersterer ist ein Vertreter der partitionierenden, zweiterer der modellbasierten Algorithmen. Da die zeitliche Auflösung des Warmwasserverbrauchs dem vorliegenden Datensatz zu niedrig ist, um Periodizitäten erkennen zu können, eignet sich diese Vorgehensweise hier nicht.

Auch in [10] wird mittels Fourieranalyse nach Regelmäßigkeiten im Nutzungsverhalten gesucht. Jedoch werden diese nicht zur Repräsentation der Zeitreihen genutzt, sondern, um herauszufinden welche betrachtete Zeitspanne am besten für die Erstellung von Nutzerprofilen geeignet ist. Es wird festgestellt, dass von allen nicht negativen Frequenzen, jene deren zugehörige Periode 24 Stunden beträgt, den größten Beitrag zur Zeitreihe leistet. Diese Ergebnisse passen zu der Tatsache, dass in den vorliegenden Daten keine nennenswerten Regelmäßigkeiten erkennbar sind, da die zeitliche Auflösung genau 24 Stunden beträgt.

Auf Grund der durchgeführten Fourieranalyse, besteht das Ziel in [10] darin, Nutzerprofile des täglichen Verbrauchs zu erstellen. Hierfür werden einerseits mittels einer Mischung aus einem partitionierenden und hierarchischen Algorithmus, basierend auf der Eulidischen Metrik, Cluster erstellt. Zuvor wird entweder eine Principal Component Analysis zur Dimensionsreduktion durchgeführt oder die Zeitreihen in unverarbeiteter Form dargestellt. Andererseits wird der TADPole Algorithmus für das Finden von Nutzerprofilen verwendet. Der Algorithmus bedient sich einer etwas eingeschränkten Version des Dynamic Time Warping Abstands und anschließend einer Form des partitionierenden Clusters, welche dem Partitioning Around Medoids ähnelt [15]. Allerdings lässt er sich nur auf relativ kleine Da-

4. Ergebnisse

tensätze anwenden, da sehr viel Speicherplatz im Laufe der Berechnungen allokiert wird. Außerdem wird der Dynamic Time Warping Abstand durch die Berechnung von oberen und unteren Schranken angenähert. Dies spart zwar Rechenzeit, birgt aber den Nachteil, dass die Angabe des Abstands ungenauer wird. Die besten Ergebnisse wurden auf Basis der Euklidischen Metrik erzielt. Trotzdem wird in der vorliegenden Arbeit nur im Fall der Repräsentation der Zeitreihen mittels derer statistischen Parametern davon Gebrauch gemacht. Im Gegensatz zu den in [10] betrachteten Messdaten über eine Periode von einem Tag, treten bei den vorliegenden Daten, wie bereits in 2 ausgeführt, wesentlich größere zeitliche Verschiebungen von ähnlichen Mustern auf. Andererseits ist die Länge der Zeitreihen auch wesentlich größer, was ebenfalls gegen die Verwendung der euklidischen Metrik spricht.

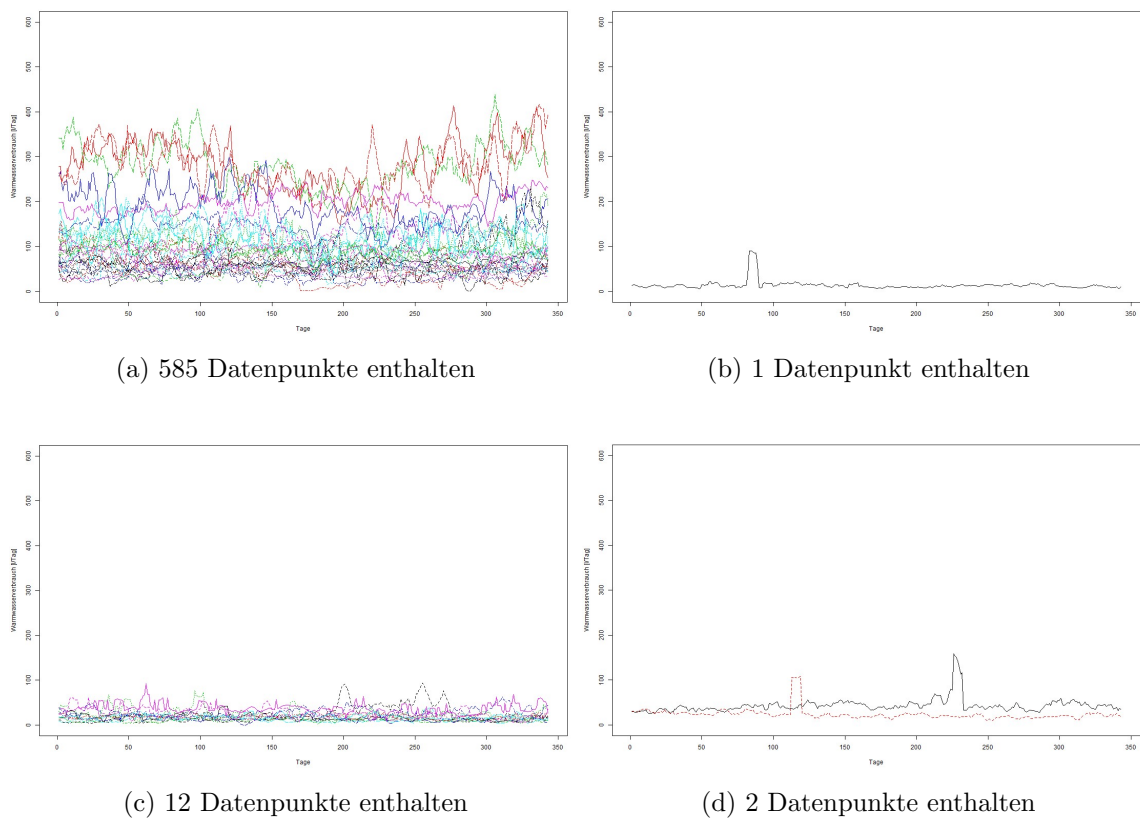


Abbildung 8.: Darstellung der Partitionierung eines Samples von 600 Zeitreihen in 4 Gruppen, wobei höchstens 30 Mitglieder eines Clusters geplottet sind.

Parameter der Partitionierung:

Darstellung: features, verschoben-normiert

Abstand: Euklidisch

Clusteralgorithmus: hierarchisch

Der Umgang mit fehlenden Daten gleicht in [10] und [5] jenem in der vorliegenden Arbeit, während das Thema in [3] kaum behandelt wird.

Zur Evaluierung der Clusterqualität wird in den betrachteten Arbeiten auf unterschiedliche Parameter zurückgegriffen. In [3] wird das Bayesian Information Criterion (BIC) verwendet. Dies ist jedoch für den vorliegenden Fall nicht sinnvoll einsetzbar, da für die Berechnung des BIC eine der Clusterverteilung zugrundeliegende Wahrscheinlichkeitsdichtefunktion angenommen wird, wie es etwa beim Modellbasierten Clustern mittels des Expectation Maximization Algorithmus der Fall ist. In [5] wird die optimale Anzahl der Cluster anhand der within group sum of squares ermittelt. Jedoch ist dafür die Definition eines Clustermittelpunkts nötig. In [5] ist dieser auf kanonische Weise gegeben, da k-Means als Clusteralgorithmus eingesetzt wird. Im vorliegenden Fall ist dies jedoch nicht der Fall, weshalb auch diese Art der Evaluierung nicht verwendet wird. In [10] wird unter anderem der Silhouettenkoeffizient zur Evaluierung der Clusterqualität eingesetzt. Meist wird er dazu verwendet, die Anzahl an Clustern k möglichst günstig zu wählen. Dabei geht man davon aus, dass der Silhouettenkoeffizient bei der Partitionierung mit der klarsten Struktur maximal wird. Bei den betrachteten Daten fällt der Silhouettenkoeffizient, bis auf wenige Ausnahmen, mit zunehmendem k . Die besten Koeffizienten ergeben sich bei der Unterteilung in zwei Gruppen. Dies ist jedoch nicht besonders hilfreich für den vorliegenden Anwendungsfall, da man sich eine höhere Differenzierung zwischen den einzelnen Nutzungsverhalten wünscht.

Es stellt sich weiters heraus, dass der Silhouettenkoeffizient als Indikator zwischen mit verschiedenen Cluster- und Abstandsmethoden gebildete Partitionierungen nicht besonders aussagekräftig ist. Bei allen verwendeten Abstandsbegriffen und jeder Anzahl an Clustern ergibt der hierarchische Ansatz die besten Silhouettenwerte. Allerdings ist das hierarchische Clustern sehr anfällig gegenüber Ausreißern. Dies führt im betrachteten Fall dazu, dass die allermeisten Datenpunkte in einem Cluster landen und die restlichen Gruppen teilweise nur eine Mitgliederanzahl im einstelligen Bereich vorweisen können. Beispielsweise wird der beste Silhouettenkoeffizient für $k = 4$ mit einem Wert von gerundet 0.648 beim Fall 12 in Tabelle 2, 600 betrachteten Zeitreihen, der Euklidischen Metrik als Abstand und dem hierarchischen Clustern als Clustermethode erreicht. Abbildung 8 zeigt höchstens 30 der den jeweiligen Gruppen zugeordneten Zeitreihen. Zur besseren Darstellung ist das Rauschen mittels eines moving average mit einem Fenster von 7 Tagen reduziert. In [17] wird darauf hingewiesen, dass beim hierarchischen Clustern keine Anpassung nach Einordnung eines Datenpunktes in die Baumstruktur mehr passieren kann. Der Algorithmus ist also sehr anfällig dafür, Ausreißer jeweils eigenen Clustern zuzuordnen. Dies hat einerseits zur Folge, dass mitunter die Anzahl an Datenpunkten in einzelnen Clustern sehr klein sein kann. Andererseits sind Cluster, die ausschließlich aus sehr selten auftretenden Verbrauchsprofilen bestehen in dem vorliegenden Fall nicht von Interesse. Abhilfe kann dadurch geschaffen werden, das hierarchische Clustern mit anderen Cluster Methoden zu kombinieren.

Ein offensichtliches Problem dieser Aufteilung ist nicht nur, dass die Anzahl der Clustermitglieder sehr ungleich ist. Die Zuordnung passiert außerdem nicht im gewünschten Sinne. Beispielsweise bildet die Zeitreihe aus Cluster 8b intuitiv ein ähnliches Nutzungsverhalten wie jene aus Cluster 8d ab.

Dem gegenüber steht die Partitionierung, welche unter den selben Parametern, aber mittels des Cluster Algorithmus Partitioning Around Medoids vorgenommen wurde. Diese ist in Abbildung 9 dargestellt. Obwohl der Silhouettenkoeffizient mit 0.301 deutlich schlechter ist,

4. Ergebnisse

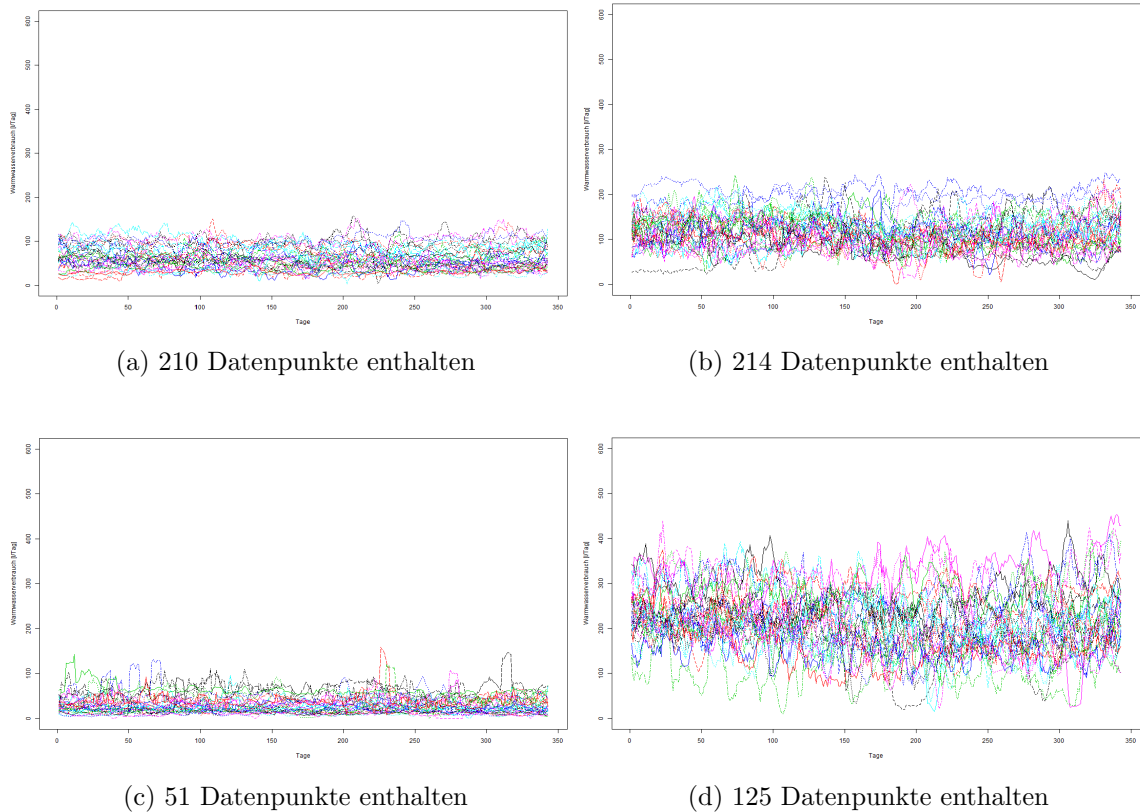


Abbildung 9.: Darstellung der Partitionierung eines Samples von 600 Zeitreihen in 4 Gruppen, wobei höchstens 30 Mitglieder eines Clusters plottet sind.

Parameter der Partitionierung:

{Darstellung:} features, verschoben-normiert

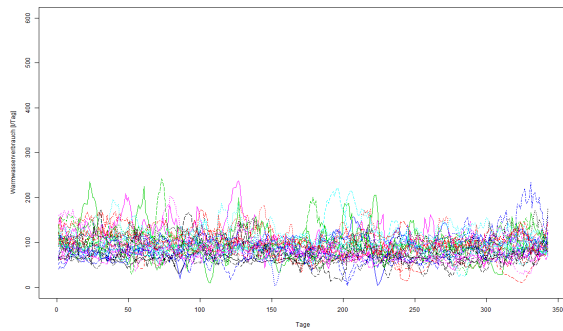
Abstand: Euklidisch

Clusteralgorithmus: Partitioning Around Medoids

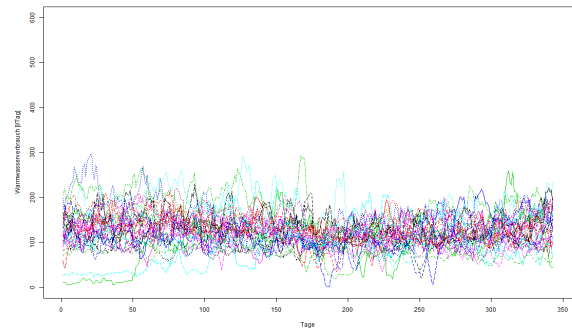
sind einerseits die Datenpunkte besser auf die Cluster verteilt und die Zuordnung scheint plausibel.

Besonders schlecht ist der Silhoettenkoeffizient, wenn als Abstand das Compression Based Dissimilarity Measure gewählt wird. Im Fall 2 in der Tabelle 2 in Kombination mit Partitioning Around Medoids als Cluster Methode, beträgt er für $k = 4$ nur 0.030. Trotzdem weisen die als ähnlich klassifizierten Zeitreihen auch intuitiv Gemeinsamkeiten auf.

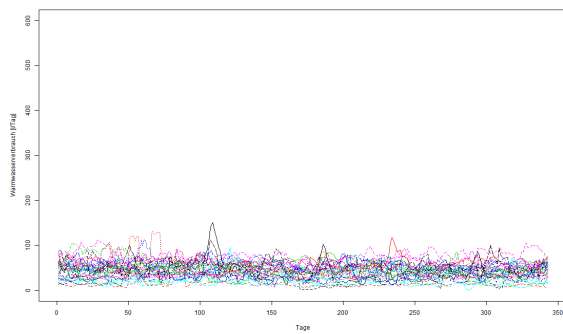
4. Ergebnisse



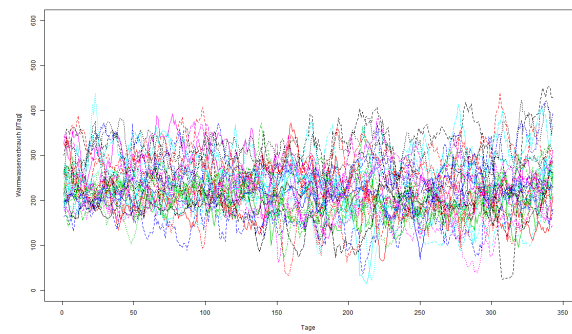
(a) 166 Datenpunkte enthalten



(b) 151 Datenpunkte enthalten



(c) 176 Datenpunkte enthalten



(d) 107 Datenpunkte enthalten

Abbildung 10.: Darstellung der Partitionierung eines Samples von 600 Zeitreihen in 4 Gruppen, wobei höchstens 30 Mitglieder eines Clusters plottet sind.

Parameter der Partitionierung:

Darstellung: features, verschoben-normiert

Abstand: Compression Based Dissimilarity

Clusteralgorithmus: Partitioning Around Medoids

Aus den oben angeführten Beispielen lässt sich schließen, dass der Silhouettenkoeffizient für die Beurteilung der Clusterqualität im vorliegenden Fall eher ungeeignet ist. In Ermangelung anderer, objektiver Qualitätskriterien können auch nicht viele Aussagen darüber gemacht werden, welche Darstellungsmethoden und Clusteralgorithmen bessere oder schlechtere Ergebnisse liefern.

4.2. Resultate der Optimierung des Heizverhaltens

Das in 3 vorgestellte Modell wird über einen Zeitraum von 30 Tagen betrachtet. Dabei wird das optimale Heizverhalten mittels des Optimierungs-Solvers SCIP ermittelt.

5	Maximale Vervielfachung der Bakterienkonzentration
100 / 200 / 300	Fassungsvermögen des Warmwasserspeichers [l]
109	Mittelwert täglicher Warmwasserverbrauch [l]
17	Mittelwert tägliche Anzahl an Taps
7	Standardabweichung tägliche Anzahl an Taps
15	Temperatur des einströmenden Wassers [°C]
40	Minimaltemperatur im Warmwasserspeicher [°C]

Tabelle 6.: Parameter für das Generieren einer exemplarischen Zeitreihe des Warmwasserverbrauchs und der Verhältnisse im Wasserspeicher über 30 Tage

Anhand der in Tabelle 6 aufgelisteten Parameter wird in der Programmiersprache R exemplarisch eine Zeitreihe des Warmwasserverbrauchs über 30 Tagen generiert. Dabei werden die Taps ungleich über den Tag verteilt, sodass morgens und abends am meisten und in der Nacht am wenigsten Wasser verbraucht wird. Die zeitliche Auflösung der so generierten Daten beträgt zwei Stunden. Zur Veranschaulichung ist in Abbildung 11 der angenommene Warmwasserverbrauch dargestellt.

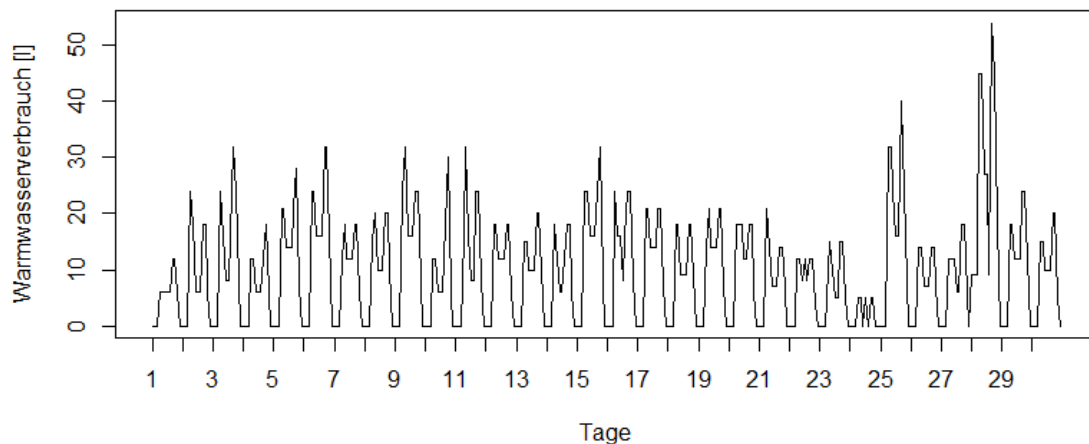


Abbildung 11.: Modellierter Warmwasserverbrauch in zwei Stunden über einen Zeitraum von einem Monat

Der tägliche Warmwasserverbrauch wird bewusst niedrig gewählt, da ein höherer Verbrauch zu weniger Vermehrung der Bakterien führt. Außerdem sei darauf hingewiesen, dass bei einem Konsum von durchschnittlich 109 Litern pro Tag kein Warmwasserspeicher von 200 oder gar 300 Litern Fassungsvermögen verwendet werden sollte.

Jedoch besteht das Bestreben, in dieser Arbeit tendenziell von einem schlechteren Fall als er in der Realität normalerweise besteht auszugehen.

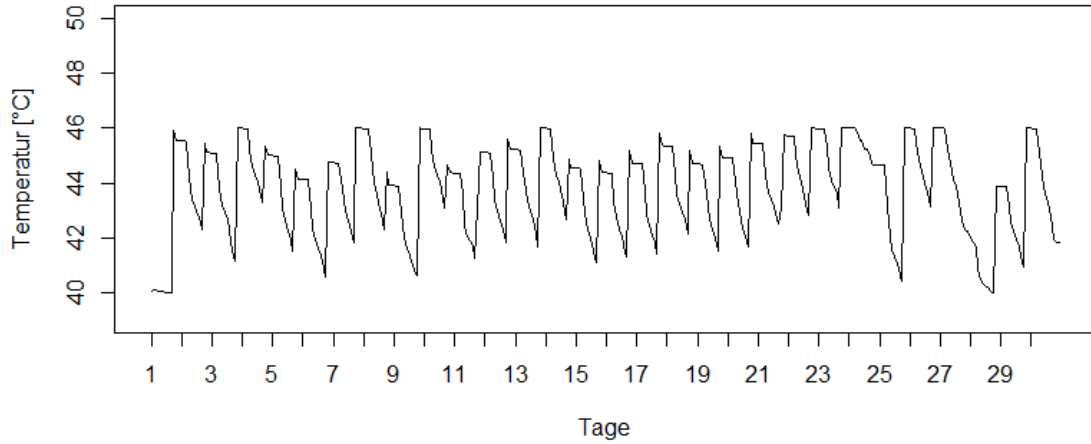


Abbildung 12.: Optimaler Temperaturverlauf im Warmwasserspeicher bei einem Fassungsvermögen von 100 Litern und einer maximalen Legionellenkonzentration von 5 mal der Anfangskonzentration b_0 .

Nach dem zugrunde liegenden Modell ergeben sich, je nach Fassungsvermögen, eine Durchschnittstemperatur, eine mittlere Legionellenkonzentration und ein ungefährender Energieverbrauch gemäß Tabelle 7.

L [l]	\bar{T} [°C]	\bar{b}	Q [kWh]
100	43.8	4.1	12.9
200	44.9	4.2	19.6
300	45.3	4.0	22.8

Tabelle 7.: Mittlere Temperatur, mittlere Bakterienkonzentration und in einem Monat benötigte Energie bei unterschiedlichen Fassungsvermögen.

Bemerkenswert ist dabei, dass es bei keinem der Fälle von Nöten war von der Desinfektionsmöglichkeit des Warmwasserspeichers Gebrauch zu machen. Jedoch steigt die mittlere Temperatur im Speicher und auch der Energieverbrauch mit zunehmendem Fassungsvermögen. Besonders niedrig ist der Energieaufwand bei einem Warmwasserspeicher mit einem Fassungsvermögen von 100 Litern. Die Abbildungen 12 und 13 zeigen den Temperaturbeziehungsweise Bakterienkonzentrationsverlauf für jenen Fall.

Führt man zusätzlich die Nebenbedingung ein, dass mindestens einmal im Monat eine thermische Desinfektion erfolgen soll, ergeben sich Werte gemäß Tabelle 8. Diese soll weder in der ersten noch in der letzten Woche des simulierten Zeitraumes stattfinden, da vor allem das Langzeitverhalten des Systems von Interesse ist.

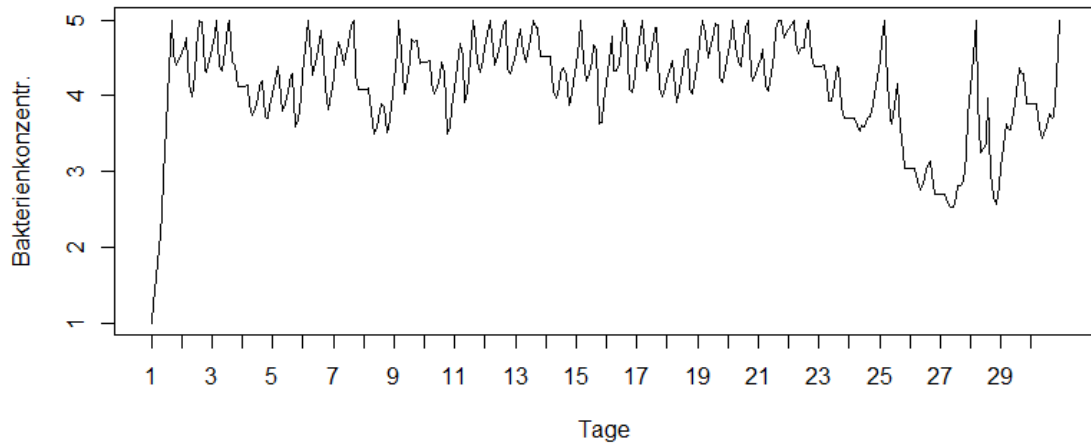


Abbildung 13.: Bakterienkonzentration im Warmwasserspeicher, angegeben als ein Vielfaches der Anfangskonzentration b_0 , bei einem Fassungsvermögen von 100 Litern und einer maximalen Legionellenkonzentration von 5 mal b_0 bei optimiertem Temperaturverlauf

L [l]	\bar{T} [°C]	\bar{b}	Q [kWh]
100	43.9	3.9	13.0
200	45.0	3.7	19.6
300	45.3	3.5	22.7

Tabelle 8.: Mittlere Temperatur, mittlere Bakterienkonzentration und in einem Monat benötigte Energie bei unterschiedlichen Fassungsvermögen, unter der Nebenbedingung, dass mindestens einmal im Monat eine thermische Desinfektion stattfinden muss.

Vergleicht man die Ergebnisse aus Tabelle 7 mit jenen in Tabelle 8, fällt auf, dass der Energieverbrauch durch eine monatliche Desinfektion nicht signifikant steigt, sofern Zeitpunkt und Desinfektionstemperatur optimal sind. In allen drei betrachteten Fällen wird die niedrigste Desinfektionstemperatur, also 50 °C gewählt.

In beiden Fällen liegt die durchschnittliche Temperatur im Wachstumsbereich der Legionellen. Vergleicht man die Abbildung 11 mit der Abbildung 12, bestätigt sich die Erwartung, dass mehr momentaner Verbrauch zu einer niedrigeren optimalen Temperatur führt. Weiters zeigt sich, dass das Wasser in einem zu großen Speicher bei einer höheren mittleren Temperatur gespeichert werden muss, um die Legionellenkonzentration unter dem festgesetzten Schwellenwert zu halten.

5. Fazit

Die in der vorliegenden Arbeit durchgeführte Suche nach hilfreichen Nutzergruppen mittels Unsupervised Machine Learning Methoden erweist sich, wie bereits in den Ergebnissen vermutet werden konnte, als nicht erfolgreich. Zwar können Cluster von Verbrauchern gefunden werden, jedoch fehlen auf den Anwendungsfall abgestimmte Qualitätskriterien, um die Sinnhaftigkeit der Zuteilung beurteilen zu können. Ein gängiges Maß für die Qualität der Partition, der Silhouettenkoeffizient, nimmt für die meisten Einteilungen der Daten niedrige Werte an und fällt mit zunehmender Anzahl der Cluster. Dies deutet auf nicht klar voneinander abgegrenzte Gruppierungen hin. Einzig manche hierarchisch erzeugten Partitionierungen weisen einen höheren Silhouettenkoeffizienten auf. Jedoch besteht bei diesen das Problem, dass die Zeitreihen sehr ungleich auf die gefundenen Cluster verteilt werden, sodass die meisten Gruppen eine Mitgliederanzahl im einstelligen oder niedrigen zweistelligen Bereich aufweisen. Eine solche Partitionierung kann möglicherweise nützlich sein, um Anomalien im Warmwasserverhalten festzustellen, nicht jedoch, um verschiedene Nutzungsprofile aus den Clustern abzulesen.

Diese Ergebnisse können einerseits bedeuten, dass es nicht möglich ist, die vorliegenden Zeitreihen in ähnlich große Gruppen einzuteilen, welche Datenpunkte enthalten, die einander sehr ähnlich sind, sich allerdings zu Datenpunkten aus anderen Clustern stark unterscheiden. Andererseits kann es auch sein, dass eine derartige Partitionierung zwar existiert, aber im Rahmen dieser Arbeit nicht gefunden wurde. Auf Grund der unzähligen existierenden Clustermethoden, werden hier nur sehr wenige jener Methoden umgesetzt, die sich in ähnlichen Anwendungen als erfolgreich erwiesen. Besonders dichte- und modellbasierte Algorithmen werden nicht behandelt. Möglicherweise ergibt sich eine sinnvolle Einteilung der Daten, wenn man eine genauere Vorstellung hat wonach diese erfolgen soll, einen geeigneten Clusteralgorithmus findet und es allen voran klar definierte Kriterien gibt, wann eine Partition gut und wann sie schlecht ist.

Außerdem fällt auf, dass es in Arbeiten, welche sich ebenfalls mit dem Erstellen von Warmwasserverbrauchsprofilen befassen, üblich ist, nicht den täglichen Gesamtverbrauch zu betrachten. Stattdessen werden Messdaten in minütlicher oder stündlicher Auflösung ausgewertet. Zeitreihen dieser Art weisen, im Gegensatz zu den vorliegenden, signifikante Periodizitäten auf und eröffnen somit die Möglichkeit, auf Fourieranalyse basierende Parameter zum Auffinden von Partitionen zu nutzen.

Im Modell zeigt sich sehr deutlich, dass sich ein optimales Heizverhalten an den Wasserverbrauch des Nutzers anpasst. Bei hohem Verbrauch ist es möglich das Wasser bei relativ niedriger Temperatur, im betrachteten Fall zeitweise sogar bei der Minimaltemperatur von 40 °C zu speichern. Ein weiteres Resultat der Arbeit ist, dass kaum zusätzliche Energie nötig ist, um einmal im Monat eine thermische Desinfektion durchzuführen, sofern diese zu einem optimalen Zeitpunkt geschieht.

Würde dieses Modell die Realität hinreichend genau widerspiegeln, wäre es eventuell interessant die optimalen Heizprofile vieler verschiedener Verbraucher hinsichtlich ihrer Ähnlichkeit zu untersuchen. Möglicherweise könnte dadurch aus vergangenen Nutzungsmustern auf ein zukünftig optimales, oder verbessertes, Heizmuster geschlossen werden.

Jedoch besteht ein ähnliches Problem wie bei der Evaluierung der Clusterqualität. Das Modell ist hinsichtlich einiger Aspekte vereinfacht und gibt damit die Realität nicht exakt wieder. Insbesondere ist der Fehler, welcher durch die Annahme entsteht, Legionellen wären homogen im Wasser verteilt, sehr schwer abzuschätzen. Außerdem werden Messdaten in mindestens zweistündiger Auflösung benötigt, wenn man den Tagesverlauf des Wasserverbrauchs nicht künstlich nachbilden möchte. Das Modell vermittelt also bestenfalls eine grobe Idee der Auswirkung auf das System von Veränderungen diverser Parameter. Keinesfalls können genaue Vorhersagen bezüglich der Bakterienkonzentration oder des Energieverbrauch gemacht werden.

Literaturverzeichnis

- [1] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah. Time-series clustering – a decade review. *Information Systems*, 53:16 – 38, 2015.
- [2] G. W. Brundrett. *Legionella and building services*. Butterworth-Heinemann, 1992.
- [3] N. Cheifetz, Z. Noumir, A. Samé, A.-C. Sandraz, C. Féliers, and V. Heim. Modeling and clustering water demand patterns from real-world smart meter data. *Drinking Water Engineering and Science*, 10(2):75–82, 2017.
- [4] DIN 4108-10:2015-12, Wärmeschutz und Energie-Einsparung in Gebäuden – Teil 10: Anwendungsbezogene Anforderungen an Wärmedämmstoffe – Werkmäßig hergestellte Wärmedämmstoffe. Norm, Dec. 2015.
- [5] D. García, D. Vidal, J. Quevedo, V. Puig, and J. Saludes. Water demand estimation and outlier detection from smart meter data using classification and big data methods, 2015.
- [6] T. Giorgino. Computing and visualizing dynamic time warping alignments in r: The dtw package. *Journal of Statistical Software, Articles*, 31(7):1–24, 2009.
- [7] E. J. Keogh, S. Lonardi, C. Ratanamahatana, L. Wei, S.-H. Lee, and J. Handley. Compression-based data mining of sequential data. 14:99–129, 02 2007.
- [8] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 285–289, New York, NY, USA, 2000. ACM.
- [9] J.-P. Kreiß and G. Neuhaus. *Einführung in die Zeitreihenanalyse*. Springer-Verlag Berlin Heidelberg, 2006.
- [10] J. Leitao, N. Simoes, J. A. Marques, P. Gil, B. Ribeiro, and A. Cardoso. Categorisation of urban water consumptions. In G. L. Loggia, G. Freni, V. Puleo, and M. D. Marchis, editors, *HIC 2018. 13th International Conference on Hydroinformatics*, volume 3 of *EPiC Series in Engineering*, pages 1123–1130. EasyChair, 2018.
- [11] M. Li, X. Chen, X. Li, B. Ma, and P. M. Vitanyi. The similarity metric. *IEEE Trans. Inf. Theor.*, 50(12):3250–3264, Dec. 2004.
- [12] A. A. W. Minker. *Semi-Supervised and Unsupervised Machine Learning: Novel Strategies*. ISTE Ltd 2011, 2013.

- [13] A. Nanopoulos, R. Alcock, and Y. Manolopoulos. Information processing and technology. chapter Feature-based Classification of Time-series Data, pages 49–61. Nova Science Publishers, Inc., Commack, NY, USA, 2001.
- [14] P. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, Nov. 1987.
- [15] A. Sarda-Espinosa. Comparing time-series clustering algorithms in r using the dtwclust package. 2017.
- [16] Y. Sharaby, S. Rodríguez-Martínez, and O. Oks. Temperature-dependent growth modeling of environmental and clinical legionella pneumophila multilocus variable-number tandem-repeat analysis (mlva) genotypes. *Applied and Environmental Microbiology*, 2017.
- [17] T. Warren Liao. Clustering time series data — a survey. *Pattern Recognition*, 38:1857–1874, 11 2005.

A. Sourcecode

A.1. Clustern

```
1 #####
2 ## file:      clustering_execute_all.R
3 ## last change: September 21, 2018
4 ## company:   Robert Bosch GmbH, CR\AEB2
5 ## author:    Amanda Schoefl
6 ## notes:     execute clustering scripts
7 #####
8
9 ### clear environment
10 rm(list=ls())
11
12 ### set constants and working dir
13 WORKING_DIRECTORY <- "C:/Users/ams5wi/Bachelorarbeit/06_implementation/"
14 OUTPUT_DIRECTORY <- paste0(WORKING_DIRECTORY, "output/")
15 SAMPLE_DIRECTORY <- paste0(WORKING_DIRECTORY, "samples/")
16 CLUSTER_TYPE <- c("cdm", "dtw", "dtw_full", "fbc")
17 SAMPLE_SIZE <- c(600, 300, 100)
18
19 setwd(WORKING_DIRECTORY)
20
21 ### cluster time series and save output
22 for (type in CLUSTER_TYPE) {
23   source(paste0("clustering_", type, ".R"))
24 }
25
26 ### plot cluster and medoids
27 source("plot_clustering.R")
28
29
30 #####
31 ## file:      clustering_cdm.R
32 ## last change: September 26, 2018
33 ## company:   Robert Bosch GmbH, CR\AEB2
34 ## author:    Amanda Schoefl
35 ## contact:   Stephanie Kaschewski
36 ## required:  correct directories;
37 ##            existing samples of sizes occuring in SAMPLE_SIZE, must be generated
38 ##            by "generate_samples.R"
39 ## notes:     A few things could be solved computationally less expensive
40 ##            (e.g. concatenate for loops or not appending vectors at all), but I decided
41 ##            that the computational loss is not so great in our case and readability is
42 ##            more important
43 #####
44
45 ### set constants if not existing
46 if(!exists("WORKING_DIRECTORY")){
47   WORKING_DIRECTORY <- "C:/Users/ams5wi/Bachelorarbeit/06_implementation/"
48 }
49
50 if(!exists("OUTPUT_DIRECTORY")){
51   OUTPUT_DIRECTORY <- paste0(WORKING_DIRECTORY, "output/")
52 }
53
54 if(!exists("SAMPLE_DIRECTORY")){
55   SAMPLE_DIRECTORY <- paste0(WORKING_DIRECTORY, "samples/")
56 }
57
58 if(!exists("SAMPLE_SIZE")){
59   SAMPLE_SIZE <- c(100, 300, 600)
60 }
61
62
63 ### load required libraries and set working directory
64 library("TSclust")
65 library("cluster")
66 library("xlsx")
67 library("mclust")
68 setwd(WORKING_DIRECTORY)
69
70 for(no_samp in SAMPLE_SIZE) {
71
72   ### empty global environment and load functions
```

A. Sourcecode

```

41  cat("remove variables ", setdiff(ls(), c("no_samp", "SAMPLE_SIZE", "WORKING_DIRECTORY", "OUTPUT
    _DIRECTORY", "SAMPLE_DIRECTORY")), "\n")
42  rm(list = setdiff(ls(), c("no_samp", "SAMPLE_SIZE", "WORKING_DIRECTORY", "OUTPUT_DIRECTORY", "
    SAMPLE_DIRECTORY")))
43  source("clustering_administration_functions.R")
44
45
46  ### load sample and set some parameters
47  tmp_env <- loadSample(when= paste0(SAMPLE_DIRECTORY,"sample", no_samp, ".RData"))
48  no_series <- tmp_env$amount_of_test_series
49  ts_names <- tmp_env$index_used_data
50  k_val <- seq(2,10)
51  data_names <- c("norm", "int", "smoothed_int", "smoothed_zscore", "smoothed_norm", "smoothed_
    log")
52  add_data_names <- c("dendrogram", "medoid")
53
54
55  ### get values
56  hwc <- vector("list", length=length(data_names))
57  names(hwc) <- data_names
58
59  ## get raw values rounded to integer
60  hwc$int <- getIntValues(variable = "hwc", preprocessing = "none")$hwc
61
62  ## get smoothed values rounded to integer
63  hwc$smoothed_int <- getIntValues(variable = "hwc", preprocessing = "smoothed",
    preproc_args = 7)$hwc
64
65
66  ## get log of smoothed values
67  hwc$smoothed_log <- getLogValues(variable = "hwc", preprocessing = "smoothed",
    preproc_args = 7)$hwc
68
69
70  ## get zscore of smoothed values
71  hwc$smoothed_zscore <- getScaledValues(variable = "hwc", preprocessing = "smoothed",
    preproc_args = 7, type = "zscore")$hwc
72
73
74  ## get normed smoothed values
75  hwc$smoothed_norm <- getScaledValues(variable = "hwc", preprocessing = "smoothed",
    preproc_args = 7, type = "norm", centered = T)$hwc
76
77
78  ## get normed raw values
79  hwc$norm <- getScaledValues(variable = "hwc", preprocessing = "none",
    type = "norm", centered = T)$hwc
80
81
82  rm(tmp)
83
84  ## rename time series
85  for(dat in data_names){
86    names(hwc[[dat]]) <- ts_names
87  }
88
89
90  ### initialize dataframes
91  df <- vector("list", length=length(data_names))
92  names(df) <- data_names
93
94  ## generate names of individual data frames
95  output_names <- c("index")
96  for (k in k_val) {
97    output_names <- c(output_names, paste0("hierarchical_cluster_k",k), paste0("pam_cluster_k",k)
    ,
    paste0("hierarchical_sil_k",k), paste0("pam_sil_k",k))
98  }
99
100
101  ## generate empty df of right size
102  for (dat in data_names) {
103    df[[dat]] <- data.frame(matrix(NA, nrow=no_series, ncol=length(output_names)))
104    names(df[[dat]]) <- output_names
105    ## name time series after their absolut indexes
106    df[[dat]][["index"]] <- ts_names
107  }
108
109
110  ### initialize additionally stored data
111  add_data <- vector("list", length = length(data_names))
112  names(add_data) <- data_names
113  for (dat in data_names) {
114    add_data[[dat]] <- vector("list", length = length(add_data_names))
115    names(add_data[[dat]]) <- add_data_names
116    add_data[[dat]][["medoid"]] <- vector("list", length = length(k_val))
117    ## typecast k in k_val to string, otherwise it would be viewed as integer index
118    names(add_data[[dat]][["medoid"]]) <- lapply(k_val, function(k) paste0(k))
119  }
120  add_mclust <- vector("list", length = length(k_val))
121  names(add_mclust) <- lapply(k_val, function(k) paste0(k))
122

```

A. Sourcecode

```

123
124 ### compute dissimilarity matrices
125 dissim <- vector("list", length=length(data_names))
126 names(dissim) <- data_names
127 for (dat in data_names) {
128   cat("compute dissimilarity matrix for ", dat, "\n")
129   dissim[[dat]] <- diss(hwc[[dat]], METHOD = "CDM")
130 }
131
132
133 ### compute cluster vectors, silhouette values and additionally stored data
134 for (dat in data_names) {
135   ## compute hierarchical tree store dendrogram
136   hc <- hclust(dissim[[dat]], method = "average")
137   add_data[[dat]][["dendrogram"]] <- hc
138   for (k in k_val) {
139     cat("compute hierarchical cluster and silhouette values for data = ", dat, " und k = ", k,
140       "\n")
141     ## compute hierarchical cluster
142     clus <- cutree(hc, k=k)
143     df[[dat]][["paste0("hierarchical_cluster_k",k)"]] <- matrix(unlist(clus))[,1]
144     ## compute hierarchical silhouette values
145     df[[dat]][["paste0("hierarchical_sil_k",k)"]] <- silhouette(x = clus, dist = dissim[[dat]][,
146       "sil_width"])
147     cat("compute pam cluster and silhouette values for data = ", dat, " und k = ", k, "\n")
148     ## compute pam cluster
149     pam <- pam(x=dissim[[dat]], k)
150     df[[dat]][["paste0("pam_cluster_k",k)"]] <- pam$cluster
151     ## compute pam silhouette values
152     df[[dat]][["paste0("pam_sil_k",k)"]] <- silhouette(x = pam$cluster, dist = dissim[[dat]][, "
153       sil_width"])
154     ## store absolut medoid indexes
155     add_data[[dat]][["medoid"]][["paste0(k)"]] <- pam$medoids
156     ## compute mclust
157     add_mclust[[paste0(k)"]] <- Mclust(dissim[[dat]], G = k, verbose = F)
158   }
159 }
160 rm(k, dat, hc, pam, tmp_env, en)
161
162 ### save everything computed by now
163 cat("save image \n")
164 save_image(file=paste0(OUTPUT_DIRECTORY,"cdm", no_samp, ".RData"))
165
166 ### save df in excel file
167 cat("save in excel file \n")
168 first_iteration <- T
169 for (dat in data_names) {
170   if(first_iteration) {
171     write.xlsx(x = df[[dat]], file = paste0(OUTPUT_DIRECTORY, "cdm", no_samp, ".xlsx"),
172       sheetName = dat,
173       col.names <- T, row.names <- F, append <- F)
174     first_iteration <- F
175   } else {
176     write.xlsx(x = df[[dat]], file = paste0(OUTPUT_DIRECTORY, "cdm", no_samp, ".xlsx"),
177       sheetName = dat,
178       col.names <- T, row.names <- F, append <- T)
179   }
180 }
181
182
183 #####
184 ## file: clustering_dtw.R
185 ## last change: September 26, 2018
186 ## company: Robert Bosch GmbH, CR\AEB2
187 ## author: Amanda Schoefl
188 ## contact: Stephanie Kaschewski
189 ## required: correct directories;
190 ## existing samples of sizes occuring in SAMPLE_SIZE, must be generated by "generate
191   _samples.R"
192
193 ## notes: A few things could be solved computationally less expensive
194 ## (e.g. concatenate for loops or not appending vectors at all), but I decided
195 ## that the computational loss is not so great in our case and readability is
196 ## more important
197 #####
198
199 ### set constants if not existing
200 if(!exists("WORKING_DIRECTORY")){
201   WORKING_DIRECTORY <- "C:/Users/ams5wi/Bachelorarbeit/06_implementation/"
202 }
203 if(!exists("OUTPUT_DIRECTORY")){
204   OUTPUT_DIRECTORY <- paste0(WORKING_DIRECTORY, "output/")
205 }
206 if(!exists("SAMPLE_DIRECTORY")){

```

A. Sourcecode

```

24 SAMPLE_DIRECTORY <- paste0(WORKING_DIRECTORY, "samples/")
25 }
26 if(!exists("SAMPLE_SIZE")){
27   SAMPLE_SIZE <- c(100, 300, 600)
28 }
29
30
31
32 ### load required libraries and set working directory
33 library("dtwclust")
34 library("xlsx")
35 library("mclust")
36 setwd(WORKING_DIRECTORY)
37
38
39 for(no_samp in SAMPLE_SIZE) {
40
41   ### empty global environment and load functions
42   cat("remove variables ", setdiff(ls(), c("no_samp", "WORKING_DIRECTORY", "OUTPUT_DIRECTORY", "
SAMPLE_DIRECTORY")), "\n")
43   rm(list = setdiff(ls(), c("no_samp", "WORKING_DIRECTORY", "OUTPUT_DIRECTORY", "SAMPLE_DIRECTORY"
)))
44   source("clustering_administration_functions.R")
45
46
47   ### load sample and set some parameters
48   subs <- seq(128,273) # subset of used values from each time series
49   tmp_env <- loadSample(when= paste0(SAMPLE_DIRECTORY,"sample", no_samp, ".RData"))
50   no_series <- tmp_env$amount_of_test_series
51   ts_names <- tmp_env$index_used_data
52   k_val <- seq(2,10)
53   data_names <- c("paa", "norm_paa", "smoothed_int", "smoothed_log", "smoothed_zscore", "smoothed
_norm")
54   add_data_names <- c("dendrogram", "medoid")
55
56
57   ### get values
58   hwc <- vector("list", length=length(data_names))
59   names(hwc) <- data_names
60
61   ## get paa of raw values
62   hwc$paa <- computePAA(data = chooseSubset(data = "data_list", subs = subs)$hwc,
desired_length = round(length(subs)/7))
63
64
65   ## get paa of normed values
66   tmp <- getScaledValues(variable = "hwc", preprocessing = "none",
type = "norm", centered = T)
67
68   tmp <- chooseSubset(data = tmp, subs = subs)$hwc
69   hwc$norm_paa <- computePAA(data = tmp, desired_length = round(length(subs)/7))
70
71   ## get smoothed values rounded to integer
72   tmp <- getIntValues(variable = "hwc", preprocessing = "smoothed", preproc_args = 7)
73   hwc$smoothed_int <- chooseSubset(data = tmp, subs = subs)$hwc
74
75   ## get log of smoothed values
76   tmp <- getLogValues(variable = "hwc", preprocessing = "smoothed", preproc_args = 7)
77   hwc$smoothed_log <- chooseSubset(data = tmp, subs = subs)$hwc
78
79   ## get zscore smoothed values
80   tmp <- getScaledValues(variable = "hwc", type = "zscore", preprocessing = "smoothed", preproc_
args = 7)
81   hwc$smoothed_zscore <- chooseSubset(data = tmp, subs = subs)$hwc
82
83   ## get normed smoothed values
84   tmp <- getScaledValues(variable = "hwc", preprocessing = "smoothed",
preproc_args = 7, type = "norm", centered = T)
85   hwc$smoothed_norm <- chooseSubset(data = tmp, subs = subs)$hwc
86
87   rm(tmp)
88
89
90   ## rename time series
91   for(dat in data_names){
92     names(hwc[[dat]]) <- ts_names
93   }
94
95
96   ### initialize dataframes
97   df <- vector("list", length=length(data_names))
98   names(df) <- data_names
99
100   ## generate names of individual data frames
101   output_names <- c("index")
102   for (k in k_val) {
103     output_names <- c(output_names, paste0("hierarchical_cluster_k",k), paste0("pam_cluster_k",k)
,
104     paste0("hierarchical_sil_k",k), paste0("pam_sil_k",k))

```

A. Sourcecode

```

105 }
106
107 ## generate empty df of right size
108 for (dat in data_names) {
109   df[[dat]] <- data.frame(matrix(NA, nrow=no_series, ncol=length(output_names)))
110   names(df[[dat]]) <- output_names
111   ## name time series after their absolut indexes
112   df[[dat]][["index"]] <- ts_names
113 }
114
115
116 ### initialize additionally stored data
117 add_data <- vector("list", length = length(data_names))
118 names(add_data) <- data_names
119 for (dat in data_names) {
120   add_data[[dat]] <- vector("list", length = length(add_data_names))
121   names(add_data[[dat]]) <- add_data_names
122   add_data[[dat]][["medoid"]] <- vector("list", length = length(k_val))
123   ## typecast k in k_val to string, otherwise it would be viewed as integer index
124   names(add_data[[dat]][["medoid"]]) <- lapply(k_val, function(x) paste0(x))
125 }
126 hc <- vector("list", length = length(data_names))
127 names(hc) <- data_names
128 pam <- vector("list", length = length(data_names))
129 names(pam) <- data_names
130 add_mclust <- vector("list", length = length(k_val))
131 names(add_mclust) <- lapply(k_val, function(k) paste0(k))
132
133
134 ### compute cluster vectors, silhouette values and additionally stored data
135 for (dat in data_names) {
136   ## compute hierarchical clusters
137   cat("compute hierarchical cluster and silhouette values for dat =", dat, "und k =", k_val, "\n")
138   hc[[dat]] <- tsclust(hwc[[dat]], k=k_val, distance = "dtw-basic", type = "h",
139                       control = hierarchical_control(method = "average"))
140   ## compute pam cluster
141   cat("compute pam cluster and silhouette values for dat =", dat, "und k =", k_val, "\n")
142   pam[[dat]] <- tsclust(hwc[[dat]], k=k_val, distance = "dtw-basic", type = "p",
143                       centroid = "pam")
144   ## store dendrogram
145   if(typeof(hc[[1]])=="list") {
146     ## if k_val was a vector containing more than 1 entry
147     add_data[[dat]][["dendrogram"]] <- hc[[dat]][[1]]
148   } else {
149     add_data[[dat]][["dendrogram"]] <- hc[[dat]]
150   }
151   j <- 1
152   for (k in k_val) {
153     ## store hierarchical cluster
154     df[[dat]][[paste0("hierarchical_cluster_k",k)]] <- hc[[dat]][[j]]@cluster
155     ## compute hierarchical silhouette values
156     df[[dat]][[paste0("hierarchical_sil_k",k)]] <- silhouette(x = hc[[dat]][[j]]@cluster,
157                     dist = hc[[dat]][[j]]@distmat[,,"sil_width"])
158     ## names(pam[[dat]][[j]]@cluster) <- NULL
159     ## store pam cluster
160     df[[dat]][[paste0("pam_cluster_k",k)]] <- pam[[dat]][[j]]@cluster
161     ## compute pam silhouette values
162     df[[dat]][[paste0("pam_sil_k",k)]] <- silhouette(x = pam[[dat]][[j]]@cluster,
163                     dist = pam[[dat]][[j]]@distmat[,,"sil_width"])
164     ## store absolut medoid indexes
165     add_data[[dat]][["medoid"]][[paste0(k)]] <- ts_names[attributes(pam[[dat]][[j]]@centroids)$
166                       series_id]
167
168     ## compute mclust
169     add_mclust[[paste0(k)]] <- Mclust(pam[[dat]][[j]]@distmat, G = k, verbose = F)
170     j <- j+1
171   }
172 }
173 rm(k, j, dat, en, tmp_env)
174
175 ### save everything computed by now
176 cat("save image \n")
177 save.image(file=paste0(OUTPUT_DIRECTORY,"dtw", no_samp, ".RData"))
178
179 ### save df in excel file
180 cat("save in excel file \n")
181 first_iteration <- T
182 for (dat in data_names) {
183   if(first_iteration) {
184     write.xlsx(x = df[[dat]], file = paste0(OUTPUT_DIRECTORY, "dtw", no_samp, ".xlsx"),
185              sheetName = dat,
186              col.names <- T, row.names <- F, append <- F)
187   } else {
188     first_iteration <- F
189   }
190 }

```

A. Sourcecode

```
188     write.xlsx(x = df[[dat]], file = paste0(OUTPUT_DIRECTORY, "dtw", no_samp, ".xlsx"),
189             sheetName = dat,
190             col.names <- T, row.names <- F, append <- T)
191   }
192 }

1  #####
2  ## file:      clustering_dtw_full.R
3  ## last change: September 26, 2018
4  ## company:   Robert Bosch GmbH, CR\AEB2
5  ## author:    Amanda Schoepl
6  ## contact:   Stephanie Kaschewski
7  ## required:  correct directories;
8  ##           existing samples of sizes occuring in SAMPLE_SIZE, must be generated by "generate
9  ##           _samples.R"
10 ## notes:     A few things could be solved computationally less expensive
11 ##           (e.g. concatenate for loops or not appending vectors at all), but I decided
12 ##           that the computational loss is not so great in our case and readability is
13 ##           more important
14 #####
15
16 ### set constants if not existing
17 if(!exists("WORKING_DIRECTORY")){
18   WORKING_DIRECTORY <- "C:/Users/ams5wi/Bachelorarbeit/06_implementation/"
19 }
20 if(!exists("OUTPUT_DIRECTORY")){
21   OUTPUT_DIRECTORY <- paste0(WORKING_DIRECTORY, "output/")
22 }
23 if(!exists("SAMPLE_DIRECTORY")){
24   SAMPLE_DIRECTORY <- paste0(WORKING_DIRECTORY, "samples/")
25 }
26 if(!exists("SAMPLE_SIZE")){
27   SAMPLE_SIZE <- c(100, 300, 600)
28 }
29
30
31 ### load required libraries and set working directory
32 library("dtwclust")
33 library("xlsx")
34 library("mclust")
35 setwd(WORKING_DIRECTORY)
36
37
38 for(no_samp in SAMPLE_SIZE) {
39
40   ### empty global environment and load functions
41   cat("remove variables ", setdiff(ls(), c("no_samp", "SAMPLE_SIZE", "WORKING_DIRECTORY", "OUTPUT
42     _DIRECTORY", "SAMPLE_DIRECTORY")), "\n")
43   rm(list = setdiff(ls(), c("no_samp", "SAMPLE_SIZE", "WORKING_DIRECTORY", "OUTPUT_DIRECTORY", "
44     SAMPLE_DIRECTORY")))
45   source("clustering_administration_functions.R")
46
47   ### load sample and set some parameters
48   tmp_env <- loadSample(where= paste0(SAMPLE_DIRECTORY, "sample", no_samp, ".RData"))
49   no_series <- tmp_env$amount_of_test_series
50   ts_names <- tmp_env$index_used_data
51   k_val <- seq(2,10)
52   data_names <- c("paa", "norm_paa", "smoothed_int", "smoothed_log", "smoothed_zscore", "smoothed
53     _norm")
54   add_data_names <- c("dendrogram", "medoid")
55
56   ### get values
57   hwc <- vector("list", length=length(data_names))
58   names(hwc) <- data_names
59
60   ## get paa of raw values
61   hwc$paa <- computePAA(data = "data_list", desired_length = round(349/7), variable = "hwc")
62
63   ## get paa of normed values
64   tmp <- getScaledValues(variable = "hwc", preprocessing = "none",
65     type = "norm", centered = T)$hwc
66   hwc$norm_paa <- computePAA(data = tmp, desired_length = round(349/7), variable = "hwc")
67
68   ## get smoothed values rounded to integer
69   hwc$smoothed_int <- getIntValues(variable = "hwc", preprocessing = "smoothed", preproc_args =
70     7)$hwc
71
72   ## get log of smoothed values
73   hwc$smoothed_log <- getLogValues(variable = "hwc", preprocessing = "smoothed", preproc_args =
74     7)$hwc
75
76   ## get scaled smoothed values
```

A. Sourcecode

```

74 hwc$smoothed_zscore <- getScaledValues(variable = "hwc", type = "zscore", preprocessing = "
    smoothed",
75                                     preproc_args = 7)$hwc
76
77 ## get normed smoothed values
78 hwc$smoothed_norm <- getScaledValues(variable = "hwc", preprocessing = "smoothed",
79                                     preproc_args = 7, type = "norm", centered = T)$hwc
80
81 rm(tmp)
82
83 ## rename time series
84 for (dat in data_names){
85   names(hwc[[dat]]) <- ts_names
86 }
87
88
89 ### initialize dataframes
90 df <- vector("list", length=length(data_names))
91 names(df) <- data_names
92
93 ## generate names of individual data frames
94 output_names <- c("index")
95 for (k in k_val) {
96   output_names <- c(output_names, paste0("hierarchical_cluster_k",k), paste0("pam_cluster_k",k)
97                                     ,
98                                     paste0("hierarchical_sil_k",k), paste0("pam_sil_k",k))
99 }
100
101 ## generate empty df of right size
102 for (dat in data_names) {
103   df[[dat]] <- data.frame(matrix(NA, nrow=no_series, ncol=length(output_names)))
104   names(df[[dat]]) <- output_names
105   ## name time series after their absolut indexes
106   df[[dat]][["index"]] <- ts_names
107 }
108
109 ### initialize additionally stored data
110 add_data <- vector("list", length = length(data_names))
111 names(add_data) <- data_names
112 for (dat in data_names) {
113   add_data[[dat]] <- vector("list", length = length(add_data_names))
114   names(add_data[[dat]]) <- add_data_names
115   add_data[[dat]][["medoid"]] <- vector("list", length = length(k_val))
116   ## typecast k in k_val to string, otherwise it would be viewed as integer index
117   names(add_data[[dat]][["medoid"]]) <- lapply(k_val, function(x) paste0(x))
118 }
119 hc <- vector("list", length = length(data_names))
120 names(hc) <- data_names
121 pam <- vector("list", length = length(data_names))
122 names(pam) <- data_names
123 add_mclust <- vector("list", length = length(k_val))
124 names(add_mclust) <- lapply(k_val, function(k) paste0(k))
125
126
127 ### compute cluster vectors, silhouette values and additionally stored data
128 for (dat in data_names) {
129   ## compute hierarchical clusters
130   cat("compute hierarchical cluster and silhouette values for dat =", dat, "und k =", k_val, "\n"
131       )
132   hc[[dat]] <- tsclust(hwc[[dat]], k=k_val, distance = "dtw-basic", type = "h",
133                       control = hierarchical_control(method = "average"))
134   ## compute pam cluster
135   cat("compute pam cluster and silhouette values for dat =", dat, "und k =", k_val, "\n")
136   pam[[dat]] <- tsclust(hwc[[dat]], k=k_val, distance = "dtw-basic", type = "p",
137                       centroid = "pam")
138   ## store dendrogram
139   if (typeof(hc[[1]]) == "list") {
140     ## if k_val was a vector containing more than 1 entry
141     add_data[[dat]][["dendrogram"]] <- hc[[dat]][[1]]
142   } else {
143     add_data[[dat]][["dendrogram"]] <- hc[[dat]]
144   }
145   j <- 1
146   for (k in k_val) {
147     ## store hierarchical cluster
148     df[[dat]][[paste0("hierarchical_cluster_k",k)]] <- hc[[dat]][[j]]@cluster
149     ## compute hierarchical silhouette values
150     df[[dat]][[paste0("hierarchical_sil_k",k)]] <- silhouette(x = hc[[dat]][[j]]@cluster,
151                                                             dist = hc[[dat]][[j]]@distmat)[,"sil_width"]
152     ## store pam cluster
153     df[[dat]][[paste0("pam_cluster_k",k)]] <- pam[[dat]][[j]]@cluster
154     ## compute pam silhouette values
155     df[[dat]][[paste0("pam_sil_k",k)]] <- silhouette(x = pam[[dat]][[j]]@cluster,
156                                                             dist = pam[[dat]][[j]]@distmat)[,"sil_width"]
157     ## store absolut medoid indexes

```

A. Sourcecode

```

157     add_data[[dat]][["medoid"]][[paste0(k)]] <- ts_names[attributes(pam[[dat]][[j]]@centroids)$
158         series_id]
159     ## compute mclust
160     add_mclust[[paste0(k)]] <- Mclust(pam[[dat]][[j]]@distmat, G = k, verbose = F)
161     j <- j+1
162   }
163 }
164 rm(k, j, dat, tmp_env, en)
165
166 ### save everything computed by now
167 cat("save image \n")
168 save.image(file=paste0(OUTPUT_DIRECTORY,"dtw_full", no_samp, ".RData"))
169
170 ### save df in excel file
171 cat("save in excel file \n")
172 first_iteration <- T
173 for (dat in data_names) {
174   if (first_iteration) {
175     write.xlsx(x = df[[dat]], file = paste0(OUTPUT_DIRECTORY, "dtw_full", no_samp, ".xlsx"),
176             sheetName = dat,
177             col.names <- T, row.names <- F, append <- F)
178     first_iteration <- F
179   } else {
180     write.xlsx(x = df[[dat]], file = paste0(OUTPUT_DIRECTORY, "dtw_full", no_samp, ".xlsx"),
181             sheetName = dat,
182             col.names <- T, row.names <- F, append <- T)
183   }
184 }
185 }

1 #####
2 ## file:      clustering_fbc.R
3 ## last change: September 26, 2018
4 ## company:   Robert Bosch GmbH, CR\AEB2
5 ## author:    Amanda Schoefl
6 ## contact:   Stephanie Kaschewski
7 ## required:  correct directories;
8 ##           existing samples of sizes occuring in SAMPLE_SIZE, must be generated by "generate
9             _samples.R"
9 ## notes:     A few things could be solved computationally less expensive
10 ##           (e.g. concatenate for loops or not appending vectors at all), but I decided
11 ##           that the computational loss is not so great in our case and readability is
12 ##           more important
13 #####
14
15 ### set constants if not existing
16 if(!exists("WORKING_DIRECTORY")){
17   WORKING_DIRECTORY <- "C:/Users/ams5wi/Bachelorarbeit/06_implementation/"
18 }
19 if(!exists("OUTPUT_DIRECTORY")){
20   OUTPUT_DIRECTORY <- paste0(WORKING_DIRECTORY, "output/")
21 }
22 if(!exists("SAMPLE_DIRECTORY")){
23   SAMPLE_DIRECTORY <- paste0(WORKING_DIRECTORY, "samples/")
24 }
25 if(!exists("SAMPLE_SIZE")){
26   SAMPLE_SIZE <- c(100, 300, 600)
27 }
28
29
30 ### load required libraries and set working directory
31 library("cluster")
32 library("xlsx")
33 library("mclust")
34 setwd(WORKING_DIRECTORY)
35
36
37 for(no_samp in SAMPLE_SIZE) {
38
39   ### empty global environment and load functions
40   cat("remove variables ", setdiff(ls(), c("no_samp", "SAMPLE_SIZE", "WORKING_DIRECTORY", "OUTPUT
41     _DIRECTORY", "SAMPLE_DIRECTORY", "PLOT_DIRECTORY")), "\n")
42   rm(list = setdiff(ls(), c("no_samp", "SAMPLE_SIZE", "WORKING_DIRECTORY", "OUTPUT_DIRECTORY", "
43     SAMPLE_DIRECTORY", "PLOT_DIRECTORY")))
44   source("clustering_administration_functions.R")
45
46   ### load sample and set some parameters
47   tmp_env <- loadSample(wher= paste0(SAMPLE_DIRECTORY,"sample", no_samp, ".RData"))
48   no_series <- tmp_env$amount_of_test_series
49   ts_names <- tmp_env$index_used_data
50   k_val <- seq(2,10)
51   data_names <- c("manhattan", "euclidean")

```

A. Sourcecode

```

51 add_data_names <- c("dendrogram", "medoid")
52
53
54 ### initialize dataframes
55 df <- vector("list", length=length(data_names))
56 names(df) <- data_names
57
58 ## generate names of individual data frames
59 output_names <- c("index")
60 for (k in k_val) {
61   output_names <- c(output_names,
62                     paste0("hierarchical_cluster_k",k), paste0("pam_cluster_k",k), paste0("
63                               kmeans_cluster_k",k),
64                               paste0("hierarchical_sil_k",k), paste0("pam_sil_k",k), paste0("kmeans_sil_k
65                               ",k))
66 }
67 ## generate empty df of right size
68 for (dat in data_names) {
69   df[[dat]] <- data.frame(matrix(NA, nrow=no_series, ncol=length(output_names)))
70   names(df[[dat]]) <- output_names
71   ## name time series after their absolut indexes
72   df[[dat]][["index"]] <- ts_names
73 }
74 ### initialize additionally stored data
75 add_data <- vector("list", length = length(data_names))
76 names(add_data) <- data_names
77 for (dat in data_names) {
78   add_data[[dat]] <- vector("list", length = length(add_data_names))
79   names(add_data[[dat]]) <- add_data_names
80   add_data[[dat]][["medoid"]] <- vector("list", length = length(k_val))
81   ## typecast k in k_val to string, otherwise it would be viewed as integer index
82   names(add_data[[dat]][["medoid"]]) <- lapply(k_val, function(k) paste0(k))
83 }
84 add_mclust <- vector("list", length = length(k_val))
85 names(add_mclust) <- lapply(k_val, function(k) paste0(k))
86
87
88 ### get and plot features
89 features <- newFeatures(desired_data = "hwc", desired_features = c("mean", "std", "skew", "kurt
90                               ", "little_hwc"),
91                               additional_processing = "scale")
92 invisible(clPairs(features))
93 png(file = paste0(OUTPUT_DIRECTORY, "feature_pairs_",no_samp, ".png"), width = 800, height =
94       800, units = "px")
95 clPairs(features)
96 dev.off()
97
98 ### compute dissimilarity matrices
99 dissim <- vector("list", length=length(data_names))
100 names(dissim) <- data_names
101 for (dat in data_names) {
102   cat("compute dissimilarity matrix with metric", dat, "\n")
103   dissim[[dat]] <- daisy(features, metric = dat)
104 }
105
106 ### compute cluster vectors, silhouette values and additionally stored data
107 for (dat in data_names) {
108   ## compute hierarchical tree store dendrogram
109   hc <- hclust(dissim[[dat]], method = "average")
110   add_data[[dat]][["dendrogram"]] <- hc
111   for (k in k_val) {
112     cat("compute hierarchical cluster and silhouette values for metric = ", dat, " und k = ", k
113         , "\n")
114     ## compute hierarchical cluster
115     clus <- cutree(hc, k=k)
116     df[[dat]][[paste0("hierarchical_cluster_k",k)]] <- matrix(unlist(clus))[,1]
117     ## compute hierarchical silhouette values
118     df[[dat]][[paste0("hierarchical_sil_k",k)]] <- silhouette(x = clus, dist = dissim[[dat]][,
119         "sil_width"])
120     cat("compute pam cluster and silhouette values for metric = ", dat, " und k = ", k, "\n")
121     ## compute pam cluster
122     pam <- pam(x=dissim[[dat]], k)
123     df[[dat]][[paste0("pam_cluster_k",k)]] <- pam$cluster
124     ## compute pam silhouette values
125     df[[dat]][[paste0("pam_sil_k",k)]] <- silhouette(x = pam$cluster, dist = dissim[[dat]][,
126         "sil_width"])
127     ## store absolut medoid indexes
128     add_data[[dat]][["medoid"]][[paste0(k)]] <- pam$medoids
129     ## compute kmeans cluster
130     cat("compute kmeans cluster and silhouette values for metric = ", dat, " und k = ", k, "\n")
131     )

```

A. Sourcecode

```
129 km <- kmeans(dissim[[dat]], centers = k)
130 df[[dat]][[paste0("kmeans_cluster_k",k)]] <- km$cluster
131 ## compute kmeans silhouette values
132 df[[dat]][[paste0("kmeans_sil_k",k)]] <- silhouette(x = km$cluster, dist = dissim[[dat]][,
  "sil_width"])
133 ## compute mclust
134 add_mclust[[paste0(k)]] <- Mclust(features, G = k, verbose = F)
135 }
136 }
137 rm(k, dat, hc, pam, km, en, tmp_env)
138
139 ### save everything computed by now
140 cat("save image \n")
141 save.image(file=paste0(OUTPUT_DIRECTORY,"fbc", no_samp, ".RData"))
142
143 ### save df in excel file
144 cat("save in excel file \n")
145 first_iteration <- T
146 for (dat in data_names) {
147   if (first_iteration) {
148     write.xlsx(x = df[[dat]], file = paste0(OUTPUT_DIRECTORY, "fbc", no_samp, ".xlsx"),
149               sheetName = dat,
150               col.names <- T, row.names <- F, append <- F)
151   } else {
152     write.xlsx(x = df[[dat]], file = paste0(OUTPUT_DIRECTORY, "fbc", no_samp, ".xlsx"),
153               sheetName = dat,
154               col.names <- T, row.names <- F, append <- T)
155   }
156 }
157 }
```

A. Sourcecode

```
1 #####
2 ## file: clustering_administration_functions.R
3 ## last change: October 11, 2018
4 ## company: Robert Bosch GmbH, CR\AEB2
5 ## author: Amanda Schoefl
6 ## contact: Stephanie Kaschewski
7 ## notes: containing only functions;
8 ## if script turns out to be too slow, search this script for "(impr)" and
9 ## you will find parts, that could be improved in that case;
10 ## everything is computed new if there is an option preprocessing (e.g. like in
11 ## getLogValues)
12 ## ToDo: save in Object instead of environment en;
13 ## check input of ALL functions if data type is correct (currently only implemented
14 ## to some extent)
15 ## unitize plot functions
16 ## unitize preprocessing functions (computePAA is very different)
17 #####
18 ## load needed libraries
19 library(zoo)
20 library(TSA)
21 library(stats)
22 library(cluster)
23 library(jmotif)
24
25 #####
26 ## function: loadOriginalData
27 ## input:
28 ## file: filename where to load the data from; must contain variables
29 ## "data_array" and "data_names"; should be generated by script "time_series_by_sysid.R"
30 ## return: void
31 ## details: must be called at some poin if no sample is loaded
32 ##
33 loadOriginalData <- function(file = "useableData.RData") {
34   en <- new.env()
35   cat("load data from", file, "\n")
36   load(file, envir = en)
37   en$colour_vector <- en$colour_vector <- c("red", "darkmagenta", "pink", "green", "darkorange", "
38     blue", "chartreuse2",
39     "cadetblue", "aquamarine", "orange")
40   en$dependent <- vector()
41 }
42 #####
43 #####
44 ## function: storeSample
45 ## input:
46 ## file: filename where to save the sample
47 ## return: void
48 ## details: save variables "data_list", "index_used_data", "percentage_of_na_allowed",
49 ## "max_value", "amount_of_test_series" in file named "filename"
50 ##
51 storeSample<- function(file) {
52   data_list <- en$data_list
53   index_used_data <- en$index_used_data
54   percentage_of_na_allowed <- en$percentage_of_na_allowed
55   max_value <- en$max_value
56   data_names <- en$data_names
57   amount_of_test_series <- en$amount_of_test_series
58   save(data_list, index_used_data, percentage_of_na_allowed,
59     max_value, amount_of_test_series, file=file)
60 }
61 #####
62 #####
63 ## function: loadSample
64 ## input:
65 ## where: filename where to load the sample; must be of type .RData
66 ## return: environment en
67 ## details: retrieve sample former stored by function "storeSample"
68 ##
69 loadSample<- function(when) {
70   en <- new.env()
71   cat("load data from ", when, "\n")
72   load(when, envir = en)
73   en$colour_vector <- c("red", "darkmagenta", "pink", "green", "darkorange", "blue", "chartreuse2", "
74     cadetblue",
75     "aquamarine", "orange")
76   en$dependent <- c("data_list", "index_used_data", "amount_of_test_series", "max_value",
77     "percentage_of_na_allowed", "data_names")
78   return(en)
79 }
80 #####
81 #####
82 #####
```

A. Sourcecode

```

83 ## function: setMaxValue
84 ## input:
85 ## max_value: must be list with names(max_value) is subset of en$data_names and numeric values
86 ## return: void
87 ## details: the list values indicate a threshold the associated variable should not exceed
88 ## e.g. if max_value = list("hwc" = 600), when choosing a sample, hot water consumption
      values
89 ## at time t exceeding 600 will be set to 600 at time t
90 ##
91 setMaxValue <- function(max_value=NULL) {
92
93   if(missing(max_value)) {max_value <- NULL}
94   else if((typeof(max_value) != "list") | !any(names(max_value) %in% en$data_names)) {
95     cat("Input max_value is ignored. Must be list with names out of the following:", en$data_
      names, "\n")
96     max_value <- NULL
97   }
98   tmp_max_value <- vector("list", length = length(en$data_names))
99   names(tmp_max_value) <- en$data_names
100  for (dat in en$data_names) {
101    if(dat %in% names(max_value)) {
102      cat("Set max_value of", dat, "=", max_value[[dat]], "\n")
103      tmp_max_value[[dat]] <- max_value[[dat]]
104    } else {tmp_max_value[[dat]] <- 0}
105  }
106
107  ## assign final values
108  ## (impr) working with tmp values is more expensive but safer if program is interrupted
109  cat("creating list max_value \n")
110  en$max_value <- tmp_max_value
111 }
112 #####
113 #####
114 #####
115 ## function: newSample
116 ## input:
117 ## amount_of_test_series: amount of test series the sample should contain
118 ## (mandatory if no index is given, irrelevant else)
119 ## percentage_of_na_allowed: percentage of missing data points in time series in the sample
120 ## (only relevant if no index is given)
121 ## max_value: see "setMaxValue" for further information
122 ## (only relevant if no index is given)
123 ## index: optional; if sample should contain time series with certain index
      ; index must
124 ## be a subset of seq(1:dim(en$data_array)[3])
125 ## file: input for "loadOriginalData" if data is not loaded yet
126 ## return: new sample as list of lists
127 ## details: get 'amount_of_series' random time series with maximal 'percentage_of_na_allowed'*100
      percent of
128 ## missing data
129 ## time series list will be saved to en$hwc (list of time series) with approximated
      values instead of NAs
130 ## all variables depending on this sample will be deleted
131 ##
132 newSample <- function(amount_of_test_series, percentage_of_na_allowed=0.05, max_value=NULL,
133                       index, file = "useableData.RData") {
134   ## remove all data dependent variables from en
135   if(exists("en")) {rm(list = en$dependent, envir = en)}
136   else { loadOriginalData(file = file)}
137
138   if(!exists("data_array", where = en)) {
139     answer <- readline(prompt=paste("'data_array' does not exist in environment 'en'. \n
140     If you want to load data from file ", file, ". Please type 'y' + [ENTER]. Otherwise type
141     [ENTER]."))
142     if(answer == 'y'){loadOriginalData(file = file)}
143     else{stop("no data provided \n")}
144   }
145   setMaxValue(max_value)
146
147   if(missing(index)) {
148     ## omit time series without or only little amount of hw_consumption values
149     cat("omit time series with more than ", percentage_of_na_allowed*100, "% of missing values \n
150     ")
151     index <- seq(1:dim(en$data_array)[[en$data_names[1]]][3])
152     for (dat in en$data_names) {
153       #cat("dat: ", dat, "\n")
154       index <- intersect(index, which(
155         (colSums(is.na(en$data_array[[dat]])) < percentage_of_na_allowed*nrow(en$data_array[[dat]
156         ]))))
157     }
158     ## choose maximum of 'amount_of_test_series' random samples
159     cat("choose ", min(amount_of_test_series, length(index)), "random samples \n")
160     ## it is very confusing, that if 'replace' argument is 'FALSE', no duplicate values occur,
      ## but there are duplicate values if 'replace' is 'TRUE'

```

A. Sourcecode

```

161     index <- sample(x=index, replace = F, size=min(amount_of_test_series, length(index)))
162     #if (length(unique(index)) != amount_of_test_series) {stop("duplicate time series in sample \
n")}
163     if (length(unique(index)) != length(index)) {stop("duplicate time series in sample \n")}
164   }
165
166   tmp_data_list <- lapply(en$data_array, function(x) x[, ,index])
167   names(tmp_data_list) <- en$data_names
168
169   for (dat in en$data_names){
170     ## convert into a list of lists
171     tmp_data_list [[dat]] <- as.list(data.frame(tmp_data_list [[dat]]))
172     names(tmp_data_list [[dat]]) <- index
173
174     for (i in index) {
175       ## approximate missing values
176       tmp_data_list [[dat]][[paste0(i)]] <- na.approx(tmp_data_list [[dat]][[paste0(i)]],
na.rm=F)
177
178       ## replace values greater than max_value with max_value if this parameter is > 0
179       if(sum(is.na(tmp_data_list [[dat]][[paste0(i)]])&T) > 0) {
180         tmp_data_list [[dat]][[paste0(i)]] [is.na(tmp_data_list [[dat]][[paste0(i)]])] <-
181           mean(na.omit(tmp_data_list [[dat]][[paste0(i)]]))
182       }
183       if(en$max_value [[dat]]>0) {
184         tmp_data_list [[dat]][[paste0(i)]] [tmp_data_list [[dat]][[paste0(i)]]>en$max_value [[dat]]]
<-
185           en$max_value [[dat]]
186       }
187     }
188   }
189
190   ## assign final values
191   ## (impr) working with tmp values is more expensive but safer if program is interrupted
192   cat("creating list data_list \n")
193   en$data_list <-< tmp_data_list
194   for (dep in c("data_list", "index_used_data", "amount_of_test_series", "max_value",
195               "percentage_of_na_allowed")) {
196     if(!(dep %in% en$dependent)) {en$dependent <-< c(en$dependent, dep)}
197   }
198   en$index_used_data <-< index
199   en$percentage_of_na_allowed <-< percentage_of_na_allowed
200   en$amount_of_test_series <-< amount_of_test_series
201   return(en$data_list)
202 }
203
204 #####
205
206 #####
207 ## function: chooseSubset
208 ## input:
209 ##   data: can be string containing the name of an existing data list (e.g. "data_list", "log-
data", ..)
210 ##   or can be a list of lists of time series (same structure as en$data_list)
211 ##   subs: either bool vector of same length as time series where subset should be chosen from
212 ##   or subset of index vector of time series (subset of seq(1:length(time series)))
213 ## return: subset as data list (same structure as en$data_list)
214 ##
215 chooseSubset <- function(data, subs) {
216   if(typeof(data)=="list") {
217     tmp_sub_ts <- data
218   }else if((typeof(data)=="character") & (data %in% en$dependent) & (typeof(en[[data]])=="list"))
219     {
220     tmp_sub_ts <- en[[data]]
221   }else {
222     stop("invalid data input \n")
223   }
224   sub_ts <- lapply(tmp_sub_ts, function(var) lapply(var, function(ts) ts[subts]))
225   return(sub_ts)
226 }
227
228 #####
229 ## function: plotData
230 ## input:
231 ##   data: optional; if given must be list of lists (with numeric values)
232 ##   variable: variable to plot; must be in en$data_names (e.g. "hwc");
233 ##   only relevant if "data" is not given
234 ##   preprocessing: one of the strings specified in ALLOWED_INPUT_PREPROCESSING
235 ##   (at the moment one of following: "none", "smoothed", "scaled", "log")
236 ##   preproc_args: optional; additional arguments for preprocessing function
237 ##   ylim: argument ylim for plot function
238 ##   subs: subset: bool vector; same length as numeric lists to be plotted
239 ##   colour: either colour vector (same length as data) or cluster vector (same length
as data)
240 ## return: void
241 ##

```

A. Sourcecode

```

242 plotData <- function(data, variable="hwc", preprocessing="none", preproc_args, lim = c(0,600),
243   subs, colour) {
244   ALLOWED_INPUT_PREPROCESSING <- c("none", "smoothed", "scaled", "log")
245   if (missing(data)) {
246     if (preprocessing == "none") {
247       ## choose time series to plot
248       data <- getDataList()[[variable]]
249     } else if (preprocessing=="smoothed") {
250       cat("load smoothed values \n")
251       if (missing(preproc_args)) {
252         data <- getSmoothedValues(variable)[[variable]]
253       } else {
254         data <- getSmoothedValues(variable, preproc_args)[[variable]]
255       }
256     } else if (preprocessing == "scaled") {
257       if (missing(preproc_args)) {
258         data <- getScaledValues(variable = variable)[[variable]]
259       } else {
260         data <- getScaledValues(variable = variable, type = preproc_args)[[variable]]
261       }
262       lim <- c(-3,3)
263     } else if (preprocessing == "log") {
264       data <- getLogValues()[[variable]]
265       lim <- c(0,6)
266     }
267     else {
268       stop(paste0("Invalide input for preprocessing. Plaease choose from ", paste0(ALLOWED_INPUT_
269         PREPROCESSING), "\n"))
270   }
271 }
272 ## set colour
273 if (missing(colour)) {
274   colour <- rep("black", length(data))
275 } else if (length(colour)==length(data) & typeof(colour)=="double"){
276   j <- 1
277   for (col in unique(colour)) {
278     colour[colour==col] <- en$colour_vector[j%%length(en$colour_vector)+1]
279     j <- j+1
280   }
281   if (!missing(subs)) {colour <- colour[subs]}
282 } else if (length(colour)==length(data) & typeof(colour)=="character") {
283   #colour <- sapply(colour, function(x)paste0(x))
284   if (!is.na(subs)) {colour <- colour[subs]}
285 } else {stop("invalid colour input.\n")}
286
287 ## choose subset (must be after setting colour!)
288 if (!missing(subs)) {data <- data[subs]}
289
290 cat("type 'q' + [enter] to quit\n")
291 for (j in 1:length(data)) {
292   data[[j]] <- na.omit(data[[j]])
293   if (length(data) != 0) {
294     plot(data[[j]], type="l", xlab="days", ylab="hot water consumption", main=paste0("time
295       series ", j),
296         col=colour[j], ylim = lim)
297     answer <- readline(prompt=paste("Press [enter] to continue. Plot: ", j))
298   } else {
299     answer <- readline(prompt=paste("Press [enter] to continue. No valid values in: ", j))
300   }
301   if (answer == "q") break
302 }
303 }
304 #####
305
306
307 #####
308
309 ## function: plotCluster
310 ## input:
311 ##   clust:      vector of integers indicating cluster membership (e.g. c(2,2,1,3,1) means
312 ##             that 3 clusters
313 ##             exist and the first element of list of data belongs to the second cluster,
314 ##             ...)
315 ##   data:      optional; if given must be list of time series and length(data) must be
316 ##             equal to length(clust)
317 ##   variable:  variable to plot; must be in en$data_names (e.g. "hwc");
318 ##             only relevant if "data" is not given
319 ##   preprocessing: must be string contained by ALLOWED_INPUT_PREPROCESSING ("none", "smoothed
320 ##             ")
321 ##   preproc_args: optional; additional arguments for preprocessing function
322 ##   max_plot:   maximal amount of series to plot;
323 ##   lim:       argument ylim for plot function
324 ## return:     void

```

A. Sourcecode

```

322 ##
323 plotCluster <- function(clust, data, variable="hwc", preprocessing="none", preproc_args, max_plot
      =50, lim = c(0,600)) {
324   ALLOWED_INPUT_PREPROCESSING <- c("none", "smoothed")
325   if (missing(data)){
326     ## choose time series to plot
327     if(preprocessing=="none"){
328       data <- getDataList()[[variable]]
329       lim <- c(0,600)
330     }else if(preprocessing=="smoothed") {
331       cat("load smoothed values \n")
332       if(missing(preproc_args)) {
333         data <- getSmoothedValues(variable)[[variable]]
334       }else{
335         data <- getSmoothedValues(variable, preproc_args)[[variable]]
336       }
337       lim <- c(0,600)
338     }else {
339       stop(paste0("Invalide input for preprocessing. Plaease choose from ", paste0(ALLOWED_INPUT_
        PREPROCESSING), "\n"))
340     }
341   }
342
343   ## check length of cluster vector
344   if(!(length(clust)==length(data))) {stop("invalid cluster input.\n")}
345
346   ## subset of time series belonging to the right cluster and plot it
347   k_vec <- unique(clust)
348   cat("Plot at maximum ", max_plot, "members of each cluster. Type 'q' + [enter] to quit \n")
349   for(j in k_vec) {
350     cl <- data[clust==j]
351     cl <- matrix(unlist(cl[1:min(sum(clust==j), max_plot)]), nrow=min(sum(clust==j), max_plot),
      byrow=T)
352     matplot(t(cl), type=c("l"), xlab="Tage", ylab=variable,
353             main=paste0("Cluster: ", j, "/", max(k_vec), "\nAnzahl der enthaltenen Datenpunkte: "
      length(clust[clust==j])), ylim=lim)
354     answer <- readline(prompt=paste("Press [enter] to continue. "))
355     if(answer == "q") break
356   }
357 }
358
359 #####
360
361
362
363 #####
364 ## function: savePlotCluster
365 ## input:
366 ##   clust:      vector of integers indicating cluster membership (e.g. c(2,2,1,3,1) means
      that 3 clusters
367 ##
368 ##   data:      optional; if given must be list of time series and length(data) must be
      equal to length(clust)
369 ##
370 ##   output_file_name: file name without file ending (e.g. "plots/plot" instead of "plots/plot.
      png");
371 ##
372 ##   variable:  will be extended by function
      variable to plot; must be in en$data_names (e.g. "hwc");
373 ##
374 ##   preprocessing: must be string contained by ALLOWED_INPUT_PREPROCESSING ("none", "smoothed
      ")
375 ##
376 ##   preproc_args: optional; additional arguments for preprocessing function
377 ##
378 ##   max_plot:  maximal amount of series to plot;
379 ##
380 ##   lim:       argument ylim for plot function
381 ##
382 ##   ylabel:    argument ylab for plot function
383 ##
384 ## return: void
385 ##
386 savePlotCluster <- function(clust, data, output_file_name, variable="hwc", preprocessing="none",
      preproc_args,
387                             max_plot=50, lim = c(0,600), ylabel="Warmwasserverbrauch [l/Tag]") {
388   ALLOWED_INPUT_PREPROCESSING <- c("none", "smoothed")
389   if (missing(data)){
390     ## choose time series to plot
391     if(preprocessing=="none"){
392       data <- getDataList()[[variable]]
393     }else if(preprocessing=="smoothed") {
394       cat("load smoothed values \n")
395       if(missing(preproc_args)) {
396         data <- getSmoothedValues(variable)[[variable]]
397       }else{
398         data <- getSmoothedValues(variable, preproc_args)[[variable]]
399       }
400     }else {
401       stop(paste0("Invalide input for preprocessing. Plaease choose from ", paste0(ALLOWED_INPUT_
        PREPROCESSING), "\n"))
402     }
403   }

```

A. Sourcecode

```

398 }
399
400 ## check length of cluster vector
401 if(!(length(clust)==length(data))) {stop("invalid cluster input.\n")}
402
403 ## subset of time series belonging to the right cluster and plot it
404 k_vec <- unique(clust)
405 cat("Plot at maximum ", max_plot, "members of each cluster \n")
406 for(j in k_vec) {
407   cl <- data[clust==j]
408   cl <- matrix(unlist(cl[1:min(sum(clust==j), max_plot)]), nrow=min(sum(clust==j), max_plot),
409     byrow=T)
410   png(file = paste0(output_file_name, "_",j, ".png"), width = 800, height = 500, units = 'px')
411   matplot(t(cl), type=c("l"), xlab="Tage", ylab=ylabel,
412     main=paste0("Cluster: ", j, "/", max(k_vec), "\nAnzahl der enthaltenen Datenpunkte: "
413       , length(clust[clust==j]), ylim=lim)
414   dev.off()
415 }
416 #####
417
418
419 #####
420 ## function: plotCentroids
421 ## input:
422 ## cent: vector with absolut indexes (= names of time series) of time series which
423 ## are the cluster centroids (e.g. cent=c('2220', '12') means (if no input for "data"
424 ## is given) en$data_list[[variable]][['2220']] is centroid of first cluster and
425 ## en$data_list[[variable]][['12']] is centroid of second cluster)
426 ## data: optional; if given must be list of time series (same structure as en$data_
427 ## list$hwc)
428 ## variable: variable to plot; must be in en$data_names (e.g. "hwc");
429 ## preprocessing: only relevant if "data" is not given
430 ## must be string contained by ALLOWED_INPUT_PREPROCESSING ("none", "smoothed
431 ## ")
432 ## preproc_args: optional; additional arguments for preprocessing function
433 ## lim: argument ylim for plot function
434 ## return: void
435 ##
436 plotCentroids <- function(cent, data, variable="hwc", preprocessing="none", preproc_args, lim = c
437 (0,600)) {
438   ALLOWED_INPUT_PREPROCESSING <- c("none", "smoothed")
439   if (missing(data)){
440     ## choose time series to plot
441     if(preprocessing=="none"){
442       data <- getDataList()[[variable]]
443       lim <- c(0,600)
444     }else if(preprocessing=="smoothed") {
445       cat("load smoothed values \n")
446       if(missing(preproc_args)) {
447         data <- getSmoothedValues(variable)[[variable]]
448       }else{
449         data <- getSmoothedValues(variable, preproc_args)[[variable]]
450       }
451       lim <- c(0,600)
452     }else {
453       stop(paste0("Invalide input for preprocessing. Plaease choose from ", paste0(ALLOWED_INPUT_
454         PREPROCESSING), "\n"))
455     }
456   }
457 }
458
459 ## centroid vector
460 if!(typeof(cent) == "double" | typeof(cent) == "integer" | typeof(cent) == "character" )) {
461   stop("invalid cent input.\n")
462 }
463
464 ## subset of time series belonging to the right cluster and plot it
465 cat("Plot centroid of each cluster. Type 'q' + [enter] to quit \n")
466 for(j in 1:length(cent)) {
467   cl <- data[[paste0(cent[j])] ]
468   plot(cl, type="l", xlab="Tage", ylab=variable,
469     main=paste0("Center: ", j, "/", length(cent)), ylim=lim)
470   answer <- readline(prompt=paste("Press [enter] to continue."))
471   if(answer == "q") break
472 }
473 }
474 #####
475
476 #####
477 ## function: savePlotCentroids
478 ## input:

```

A. Sourcecode

```

476 ## cent: vector with absolut indexes (= names of time series) of time series which
477 ## are the cluster centroids (e.g. cent=c('2220', '12') means (if no input for "data"
478 ## is given) en$data_list[[variable]][['2220']] is centroid of first cluster and
479 ## en$data_list[[variable]][['12']] is centroid of second cluster)
480 ## data: optional; if given must be list of time series (same structure as en$data_
481 ## list$hwc)
482 ## output_file_name: file name without file ending (e.g. "plots/plot" instead of "plots/plot.
483 ## png");
484 ## will be extended by function
485 ## variable: variable to plot; must be in en$data_names (e.g. "hwc");
486 ## only relevant if "data" is not given
487 ## preprocessing: must be string contained by ALLOWED_INPUT_PREPROCESSING ("none", "smoothed
488 ## ")
489 ## preproc_args: optional; additional arguments for preprocessing function
490 ## lim: argument ylim for plot function
491 ## ylabel: argument ylab for plot function
492 ## return: void
493 ##
494 savePlotCentroids <- function(cent, data, output_file_name, variable="hwc", preprocessing="none",
495 ## preproc_args,
496 ## lim = c(0,600), ylabel="Warmwasserverbrauch [l/Tag]") {
497 ALLOWED_INPUT_PREPROCESSING <- c("none", "smoothed")
498 if (missing(data)){
499 ## choose time series to plot
500 if(preprocessing=="none"){
501 data <- getDataList()[[variable]]
502 lim <- c(0,600)
503 }else if(preprocessing=="smoothed") {
504 cat("load smoothed values \n")
505 if(missing(preproc_args)) {
506 data <- getSmoothedValues(variable)[[variable]]
507 }else{
508 data <- getSmoothedValues(variable, preproc_args)[[variable]]
509 }
510 lim <- c(0,600)
511 }else {
512 stop(paste0("Invalide input for preprocessing. Plaease choose from ", paste0(ALLOWED_INPUT_
513 PREPROCESSING), "\n"))
514 }
515 }
516 ## cluster vector
517 if(!(typeof(cent) == "double" | typeof(cent) == "integer" | typeof(cent) == "character")) {stop
518 ("invalid cent input.\n")}
519
520 ## subset of time series belonging to the right cluster and plot it
521 cat("Plot centroid of each cluster.\n")
522 for(j in 1:length(cent)) {
523 cl <- data[[paste0(cent[j])]]
524 png(file = paste0(output_file_name, "_", j, ".png"), width = 800, height = 500, units = 'px')
525 plot(cl, type="l", xlab="Tage", ylab=ylabel,
526 main=paste0("Center: ", j, "/", length(cent)), ylim=lim)
527 dev.off()
528 }
529 }
530 #####
531 #####
532 ## function: getScaledValues
533 ## inputs:
534 ## variable: subset of en$data_names; variables for which to compute scaled values
535 ## preprocessing: must be string contained by ALLOWED_INPUT_PREPROCESSING ("none", "smoothed
536 ## ")
537 ## preproc_args: optional; additional arguments for preprocessing function
538 ## type: type of scaling; must be string contained by ALLOWED_INPUT_TYPE ("norm", "
539 ## zscore")
540 ## centered: T or F; should time series be centered around mean
541 ## return: list of lists of scaled time series (same format as en$data_list)
542 ## details: compute scaled time series
543 ##
544 getScaledValues <- function(variable, preprocessing, preproc_args, type = "zscore", centered = T)
545 {
546 ALLOWED_INPUT_PREPROCESSING <- c("none", "smoothed")
547 ALLOWED_INPUT_TYPE <- c("norm", "zscore")
548 if(!(type %in% ALLOWED_INPUT_TYPE)) {
549 stop(paste0("Invalide input for type. Plaease choose from ", ALLOWED_INPUT_TYPE, "\n"))
550 }
551 if(missing(preprocessing) | preprocessing == "none"){
552 data <- getDataList()
553 }else if(preprocessing=="smoothed") {
554 cat("load smoothed values \n")
555 if(missing(preproc_args)) {

```

A. Sourcecode

```

551   data <- getSmoothedValues(variable)
552 } else {
553   data <- getSmoothedValues(variable, preproc_args)
554 }
555 } else {
556   stop(paste0("Invalid input for preprocessing. Please choose from ", paste0(ALLOWED_INPUT_
      PREPROCESSING), "\n"))
557 }
558
559 ## initialize with NAs (same format as data)
560 tmp_scaled_data <- lapply(data, function(y) lapply(y, function(x) rep(NA, length(x))))
561 ## scale for each desired variable
562 for(var in variable) {
563   if(sum(is.na(tmp_scaled_data[[var]][[1]])) == length(tmp_scaled_data[[var]][[1]])) {
564     for(j in 1:length(tmp_scaled_data[[var]])) {
565       if(type == "zscore") {
566         tmp_scaled_data[[var]][[j]] <- zscore(as.ts(data[[var]][[j]]))
567       } else if(type == "norm") {
568         for(i in 1:length(tmp_scaled_data[[var]][[j]])) {
569           if(!(max(data[[var]][[j]]+max(data[[var]][[j]]*(-1), 0)) == 0)) {
570             tmp_scaled_data[[var]][[j]][i] <- (data[[var]][[j]][i] + max(data[[var]][[j]][i]*(-1)
571               , 0)) /
572               max(data[[var]][[j]] + max(data[[var]][[j]]*(-1), 0))
573           }
574         }
575       } else if(centered) { tmp_scaled_data[[var]][[j]] <- as.ts(as.vector(scale(tmp_scaled_data[[
576         var]][[j]], center=T, scale=F)))
577       } else { tmp_scaled_data[[var]][[j]] <- as.ts(as.vector(tmp_scaled_data[[var]][[j]]))
578     }
579   }
580 }
581
582 ## assign final values
583 ## (impr) working with tmp_scaled_data values is more expensive but safer if program is
584   interrupte
585   cat("creating list scaled_data \n")
586   if(type == "norm") {
587     en$scaled_data_norm <- tmp_scaled_data
588     en$dependent <- c(en$dependent, "scaled_data_norm")
589     return(en$scaled_data_norm)
590   }
591   if(type == "zscore") {
592     en$scaled_data_zscore <- tmp_scaled_data
593     en$dependent <- c(en$dependent, "scaled_data_zscore")
594     return(en$scaled_data_zscore)
595   }
596 }
597 #####
598 #####
599 ## function: getLogValues
600 ## inputs:
601 ##   variable:      variables for which log of time series values should be computed;
602 ##                 must be subset of en$data_names (e.g. c("hwc", "flow_mean"))
603 ##   preprocessing: must be string contained by ALLOWED_INPUT_PREPROCESSING ("none", "smoothed
604 ##   preproc_args:  optional; additional arguments for preprocessing function
605 ## return: log of time series of (variable + 1)
606 ## details:
607 ##
608 getLogValues <- function(variable, preprocessing, preproc_args) {
609   ALLOWED_INPUT_PREPROCESSING <- c("none", "smoothed")
610   if(missing(preprocessing) | preprocessing == "none") {
611     data <- getDataList()
612   } else if(preprocessing == "smoothed") {
613     cat("load smoothed values \n")
614     if(missing(preproc_args)) {
615       data <- getSmoothedValues(variable)
616     } else {
617       data <- getSmoothedValues(variable, preproc_args)
618     }
619   } else {
620     stop(paste0("Invalid input for preprocessing. Please choose from ", paste0(ALLOWED_INPUT_
621       PREPROCESSING), "\n"))
622   }
623
624 ## initialize with NAs (same format as data)
625 tmp_log_data <- lapply(data, function(y) lapply(y, function(x) rep(NA, length(x))))
626 # }
627 for(var in variable) {
628   if(sum(is.na(tmp_log_data[[var]][[1]])) == length(tmp_log_data[[var]][[1]])) {
629     for(j in 1:length(tmp_log_data[[var]])) {
630       tmp_log_data[[var]][[j]] <- as.ts(log(data[[var]][[j]]+1))

```

A. Sourcecode

```

631   }
632   }
633 }
634
635 ## assign final values
636 ## (impr) working with tmp values is more expensive but safer if program is interrupte
637 cat("creating list log_data \n")
638 en$log_data <- tmp_log_data
639 en$dependent <- c(en$dependent, "log_data")
640 return(en$log_data)
641 }
642 #####
643
644 #####
645 ## function: getIntValues
646 ## inputs:
647 ##   variable:      variables for which time series should be rounded to integer;
648 ##                 must be subset of en$data_names (e.g. c("hwc", "flow_mean"))
649 ##   preprocessing: must be string contained by ALLOWED_INPUT_PREPROCESSING ("none", "smoothed
650 ##                 ")
651 ##   preproc_args: optional; additional arguments for preprocessing function
652 ## return: en$int_data (same format as en$data_list)
653 ## details: values of time series rounded to integer
654
655 getIntValues <- function(variable, preprocessing, preproc_args) {
656   ALLOWED_INPUT_PREPROCESSING <- c("none", "smoothed")
657   if(missing(preprocessing) | preprocessing == "none"){
658     data <- getDataList()
659   }else if(preprocessing=="smoothed") {
660     cat("load smoothed values \n")
661     if(missing(preproc_args)) {
662       data <- getSmoothedValues(variable)
663     }else{
664       data <- getSmoothedValues(variable, preproc_args)
665     }
666   }else if(preprocessing=="log") {
667     cat("load log values \n")
668     if(missing(preproc_args)) {
669       data <- getlogValues(variable)
670     }else{
671       data <- getLogValues(variable, preproc_args)
672     }
673   }else {
674     stop(paste0("Invalide input for preprocessing. Plaease choose from ", paste0(ALLOWED_INPUT_
675     PREPROCESSING), "\n"))
676   }
677
678   ## initialize with NAs
679   tmp_int_data <- lapply(data, function(y) lapply(y, function(x) rep(NA, length(x))))
680   # }
681   for(var in variable) {
682     if(sum(is.na(tmp_int_data[[var]][[1]]))!=length(tmp_int_data[[var]][[1]])) {
683       for (j in 1:length(tmp_int_data[[var]])) {
684         tmp_int_data[[var]][[j]] <- as.ts(round(data[[var]][[j]]))
685       }
686     }
687   }
688
689   ## assign final values
690   ## (impr) working with tmp values is more expensive but safer if program is interrupte
691   cat("creating list int_data \n")
692   en$int_data <- tmp_int_data
693   en$dependent <- c(en$dependent, "int_data")
694   return(en$int_data)
695 }
696 #####
697 #####
698 ## function: getSmoothed
699 ## input:
700 ##   variable:      variables for which time series should be smoothed;
701 ##                 must be subset of en$data_names (e.g. c("hwc", "flow_mean"))
702 ##   period:        window for moving average
703 ## return: smoothed_data
704 ## details: if smoothed_data doesn't exist, mooving average with window "period" will be computed
705 ##
706 getSmoothedValues <- function(variable, period=7) {
707   if(!exists("smoothed_data", where=en)) {
708     ## (impr)
709     cat("creating list smoothed_data \n")
710     en$smoothed_data <- getDataList()
711     ## initialize with NAs
712     en$smoothed_data <- lapply(en$smoothed_data, function(y) lapply(y, function(x) rep(NA,
713     length(x))))
714     en$smoothing_period <- 0

```

A. Sourcecode

```

714 }
715 ## check if smoothing period changed, if so, compute new smoothed values
716 for(var in variable) {
717   if((sum(is.na(en$smoothed_data[[var]][[1]])) == length(en$smoothed_data[[var]][[1]])) |
718       (en$smoothing_period != period)) {
719     for (j in 1:length(en$smoothed_data[[var]])) {
720       en$smoothed_data[[var]][[j]] <- as.ts(na.omit(filter(en$data_list[[var]][[j]] ,
721         rep(1/period, period), sides=2)))
722     }
723   }
724 }
725 en$smoothing_period <- period
726 en$dependent <- c(en$dependent, "smoothed_data")
727 return(en$smoothed_data)
728 }
729 #####
730
731 #####
732 ## function: computePAA
733 ## input:
734 ## data: must be either of type character, if so, must be an existing list of lists
735 ## of time series
736 ## (e.g. "data_list" or "smoothed_data", last one must already been computed);
737 ## or must be of type list of time series with same structure as en$data_list$
738 ## hwc
739 ## variable: mandatory if input "data" is of type character; must be string contained by
740 ## en$data_names
741 ## desired_length: desired length of time series
742 ## return: paa_ts
743 computePAA <- function(data, variable, desired_length) {
744   ## get data
745   if (typeof(data) == "list") {
746     tmp <- data
747   } else if ((typeof(data) == "character") & (data %in% en$dependent) & (typeof(en[[data]]) == "list"))
748     {
749     tmp <- en[[data]][[variable]]
750   } else {
751     stop("invalid data input \n")
752   }
753   ## compute paa
754   paa_ts <- lapply(tmp, function(ts) paa(ts = ts, paa_num = desired_length))
755   return(paa_ts)
756 }
757
758 #####
759 ## function: newFeatures
760 ## input:
761 ## desired_data: must be subset of en$data_names
762 ## desired_features: can be subset of c("median", "mean", "std", "skew", "kurt", "no_hwc",
763 ## "little_hwc",
764 ## "no_tap", "little_tap")
765 ## additional_processing: can be "scale" or "none"; if scale, computed features are scaled
766 ## between [0,1]
767 ## delete_all: can be T or F; if T, all previously computed features are deleted
768 ## attribute: attribut for names of features in data frame
769 ## return: en$feature_df
770 ## details: desired feature are stored in en$feature_df
771 ## ToDo: make days an input and compute it generic with input days_no_little_hwc
772 newFeatures <- function(desired_data = "all", desired_features = "all", additional_processing = "none",
773   delete_all = T, attribute = "") {
774   ## (impr) loading the desired list is computationally more expensive, but if the name changes,
775   ## only one function needs to be changed
776   data <- getDataList()
777   if (desired_data == "all") {
778     desired_data <- names(data)
779   } else if (length(setdiff(desired_data, names(data))) > 0) {
780     cat("Invalide desired_data input, choose vector of following:", names(data), "\n")
781   }
782   if (desired_features == "all") {
783     desired_features <- c("median", "mean", "std", "skew", "kurt", "no_hwc", "little_hwc", "no_
784       tap", "little_tap")
785   }
786   ## delete old version of desired features
787   if (delete_all) {
788     rm(feature_df, pos=en)
789   } else {

```

A. Sourcecode

```
792     delExistingFeatures(desired_features, attribute)
793   }
794
795   ## check if global_df already exists, if not create it
796   if(!exists("feature_df", where = en)) {
797     tmp_feature_df <- data.frame()
798   } else {
799     tmp_feature_df <- en$feature_df
800   }
801
802   ## compute desired feature and add it to df
803   if("mean" %in% desired_features) {
804     cat("compute mean \n")
805     for (dat in desired_data) {
806       df <- vector(length=en$amount_of_test_series)
807       for (j in 1:len$amount_of_test_series) {
808         df[j] <- mean(data[[dat]][[j]], digits=30)
809       }
810       df <- data.frame(df)
811       names(df) <- paste0(dat, "_mean", attribute)
812       tmp_feature_df <- addDfColumn(dframe=tmp_feature_df, column_to_add=df)
813     }
814   }
815   if("median" %in% desired_features) {
816     cat("compute median \n")
817     for (dat in desired_data) {
818       df <- vector(length=en$amount_of_test_series)
819       for (j in 1:len$amount_of_test_series) {
820         df[j] <- median(data[[dat]][[j]], digits=30)
821       }
822       df <- data.frame(df)
823       names(df) <- paste0(dat, "_median", attribute)
824       tmp_feature_df <- addDfColumn(dframe=tmp_feature_df, column_to_add=df)
825     }
826   }
827   if("std" %in% desired_features) {
828     cat("compute std \n")
829     for (dat in desired_data) {
830       df <- vector(length=en$amount_of_test_series)
831       for (j in 1:len$amount_of_test_series) {
832         df[j] <- sqrt(var(data[[dat]][[j]]))
833       }
834       df <- data.frame(df)
835       names(df) <- paste0(dat, "_std", attribute)
836       tmp_feature_df <- addDfColumn(dframe=tmp_feature_df, column_to_add=df)
837     }
838   }
839   if("skew" %in% desired_features) {
840     cat("compute skewness \n")
841     for (dat in desired_data) {
842       df <- vector(length=en$amount_of_test_series)
843       for (j in 1:len$amount_of_test_series) {
844         df[j] <- skewness(data[[dat]][[j]])
845       }
846       df <- data.frame(df)
847       names(df) <- paste0(dat, "_skew", attribute)
848       tmp_feature_df <- addDfColumn(dframe=tmp_feature_df, column_to_add=df)
849     }
850   }
851   if("kurt" %in% desired_features) {
852     cat("compute kurtosis \n")
853     for (dat in desired_data) {
854       df <- vector(length=en$amount_of_test_series)
855       for (j in 1:len$amount_of_test_series) {
856         df[j] <- kurtosis(data[[dat]][[j]])
857       }
858       df <- data.frame(df)
859       names(df) <- paste0(dat, "_kurt", attribute)
860       tmp_feature_df <- addDfColumn(dframe=tmp_feature_df, column_to_add=df)
861     }
862   }
863   if("no_hwc" %in% desired_features) {
864     param <- "hwc"
865     cat("compute no_hwc \n")
866     df3 <- vector(length = length(data[[param]]))
867     df7 <- vector(length = length(data[[param]]))
868     for (j in 1:length(data[[param]])) {
869       ctmp <- 0
870       counter3 <- 0
871       counter7 <- 0
872       for (i in 1:length(data[[param]][[j]])) {
873         if(data[[param]][[j]][i]==0) {
874           ctmp <- ctmp+1
875         } else {
876           if (ctmp > 7){counter7 <- counter7+1}else
877             if(ctmp >2) {counter3 <- counter3+1}
```

A. Sourcecode

```

878     ctmp <- 0
879   }
880 }
881 if (ctmp > 7){counter7 <- counter7+1}else
882 if (ctmp >2) {counter3 <- counter3+1}
883 ctmp <- 0
884 df3[j] <- counter3
885 df7[j] <- counter7
886 }
887 df3 <- data.frame("no_hwc_3d"=df3)
888 df7 <- data.frame("no_hwc_7d"=df7)
889 tmp_feature_df <- addDfColumn(dframe=tmp_feature_df, column_to_add=df3)
890 tmp_feature_df <- addDfColumn(dframe=tmp_feature_df, column_to_add=df7)
891 }
892 if("little_hwc" %in% desired_features) {
893   param <- "hwc"
894   lim <- 15
895   cat("compute little_hwc \n")
896   df3 <- vector(length = length(data[[param]]))
897   df7 <- vector(length = length(data[[param]]))
898   for (j in 1:length(data[[param]])) {
899     ctmp <- 0
900     counter3 <- 0
901     counter7 <- 0
902     for (i in 1:length(data[[param]][[j]])) {
903       if(data[[param]][[j]][i]<lim) {
904         ctmp <- ctmp+1
905       } else {
906         if (ctmp > 7){counter7 <- counter7+1}else
907         if(ctmp >2) {counter3 <- counter3+1}
908         ctmp <- 0
909       }
910     }
911     if (ctmp > 7){counter7 <- counter7+1}else
912     if(ctmp >2) {counter3 <- counter3+1}
913     ctmp <- 0
914     df3[j] <- counter3
915     df7[j] <- counter7
916   }
917   df3 <- data.frame("little_hwc_3d"=df3)
918   df7 <- data.frame("little_hwc_7d"=df7)
919   tmp_feature_df <- addDfColumn(dframe=tmp_feature_df, column_to_add=df3)
920   tmp_feature_df <- addDfColumn(dframe=tmp_feature_df, column_to_add=df7)
921 }
922 if("no_tap" %in% desired_features) {
923   param <- "taps_no"
924   cat("compute no_tap \n")
925   df3 <- vector(length = length(data[[param]]))
926   df7 <- vector(length = length(data[[param]]))
927   for (j in 1:length(data[[param]])) {
928     ctmp <- 0
929     counter3 <- 0
930     counter7 <- 0
931     for (i in 1:length(data[[param]][[j]])) {
932       if(data[[param]][[j]][i]==0) {
933         ctmp <- ctmp+1
934       } else {
935         if (ctmp > 7){counter7 <- counter7+1}else
936         if(ctmp >2) {counter3 <- counter3+1}
937         ctmp <- 0
938       }
939     }
940     if (ctmp > 7){counter7 <- counter7+1}else
941     if(ctmp >2) {counter3 <- counter3+1}
942     ctmp <- 0
943     df3[j] <- counter3
944     df7[j] <- counter7
945   }
946   df3 <- data.frame("no_tap_3d"=df3)
947   df7 <- data.frame("no_tap_7d"=df7)
948   tmp_feature_df <- addDfColumn(dframe=tmp_feature_df, column_to_add=df3)
949   tmp_feature_df <- addDfColumn(dframe=tmp_feature_df, column_to_add=df7)
950 }
951 if("little_tap" %in% desired_features) {
952   param <- "taps_no"
953   lim <- 3
954   cat("compute little_tap \n")
955   df3 <- vector(length = length(data[[param]]))
956   df7 <- vector(length = length(data[[param]]))
957   for (j in 1:length(data[[param]])) {
958     ctmp <- 0
959     counter3 <- 0
960     counter7 <- 0
961     for (i in 1:length(data[[param]][[j]])) {
962       if(data[[param]][[j]][i]<lim) {
963         ctmp <- ctmp+1

```

A. Sourcecode

```

964     }else {
965         if (ctmp > 7){counter7 <- counter7+1}else
966             if(ctmp >2) {counter3 <- counter3+1}
967         ctmp <- 0
968     }
969 }
970 if (ctmp > 7){counter7 <- counter7+1}else
971     if(ctmp >2) {counter3 <- counter3+1}
972 ctmp <- 0
973 df3[j] <- counter3
974 df7[j] <- counter7
975 }
976 df3 <- data.frame(" little _tap_3d"=df3)
977 df7 <- data.frame(" little _tap_7d"=df7)
978 tmp_feature_df <- addDfColumn(dframe=tmp_feature_df, column_to_add=df3)
979 tmp_feature_df <- addDfColumn(dframe=tmp_feature_df, column_to_add=df7)
980 }
981
982 if(additional_processing=="scale") {
983     for (nam in names(tmp_feature_df)){
984         if (!(max(tmp_feature_df[[nam]]+max(tmp_feature_df[[nam]]*(-1), 0))==0) {
985             tmp_feature_df[[nam]] <- (tmp_feature_df[[nam]]+max(tmp_feature_df[[nam]]*(-1), 0))/
986                 max(tmp_feature_df[[nam]]+max(tmp_feature_df[[nam]]*(-1), 0)
987         }
988     }
989 }
990
991 ## assign final values
992 ## (impr) working with tmp values is more expensive but safer if program is interrupt
993 en$feature_df <- tmp_feature_df
994 rownames(en$feature_df) <- en$index_used_data
995 return(en$feature_df)
996 }
997
998
999
1000
1001 #####
1002 ## function: addDfColumn
1003 ## input:
1004 ##     dframe:          data frame; must be empty or same number of rows as column_to_add
1005 ##     column_to_add:  column(s) to add to data frame (preferable as data frame)
1006 ## return: void
1007 ## details: if dframe is empty, create new data frame with column(s) column_to_add
1008 addDfColumn <- function(dframe, column_to_add) {
1009     dframe <- as.data.frame(dframe)
1010     if(nrow(dframe)==0) {
1011         ret_val <- column_to_add
1012     }else {
1013         ret_val <- cbind(dframe, column_to_add)
1014     }
1015     return(as.data.frame(ret_val))
1016 }
1017 #####
1018
1019 #####
1020 ## function: delExistingFeatures
1021 ## input:
1022 ##     desired_features: character vector; features to delete
1023 ##     attribute:        character; if features were generated with an attribute (see function "
1024 ##     newFeatures")
1025 ## return: void
1026 ## details: deletes columns with given features in en$feature_df if they exists
1027 delExistingFeatures <- function(desired_features, attribute="") {
1028     ## check if a column with desired feature already exists, if so, delete it
1029     for(des_feat in desired_features) {
1030         for (dat in names(en$data_list)) {
1031             #cat(" check if ", des_feat, attribute, " already exists \n")
1032             if(paste0(dat,"_", des_feat, attribute) %in% names(en$feature_df)) {
1033                 en$feature_df[[paste0(dat, des_feat, attribute)]] <- NULL
1034                 cat(" delete column ", des_feat, " \n")
1035             }
1036         }
1037     }
1038     #####
1039
1040     #####
1041     ## function: getFeatures
1042     ## input:
1043     ##     desired_features: should be available in en$feature_df
1044     ## return: data.frame containing desired features
1045     ## details: please only use this function to access en$feature_df
1046     ##
1047     getFeatures <- function(desired_features) {
1048         if(!exists("feature_df", where=en)) {stop("feature_df doesn't exist in environment en \n")}

```

A. Sourcecode

```
1049 df <- data.frame()
1050 labels <- NULL
1051 for(des_feat in desired_features) {
1052   if(!(des_feat %in% names(en$feature_df))) {stop(paste0(des_feat, " is not a feature of en$
1053     feature_df \n"))}
1054   df <- addDfColumn(df, en$feature_df[[paste0(des_feat)]])
1055   labels <- c(labels, paste0(des_feat))
1056 }
1057 colnames(df) <- labels
1058 rownames(df) <- en$index_used_data
1059 return(df)
1060 }
1061 #####
1062
1063 #####
1064 ## function: getDataList
1065 ## input: None
1066 ## return: data_list if it exists in environment 'en', NULL otherwise
1067 ## details: please only use this function to access en$data_list
1068 ##
1069 getDataList <- function() {
1070   if (exists("data_list", where = en)) {
1071     return(en$data_list)
1072   }
1073   else {
1074     stop("data_list does not exist in environment en \n")
1075     return(NULL)
1076   }
1077 }
1078 #####
```

A. Sourcecode

```
1 #####
2 ## file:      plot_clustering.R
3 ## last change: October 24, 2018
4 ## company:   Robert Bosch GmbH, CR\AEB2
5 ## author:    Amanda Schoefl
6 ## contact:   Stephanie Kaschewski
7 ## required:  correct directories;
8 ##           existing samples of sizes occuring in SAMPLE_SIZE, must be generated by "generate
9 ##           -samples.R";
10 ##           for all 'x' in SAMPLE_SIZE "YYY'x'.RData" must be generated by "clustering_YYY.R"
11 #####
12
13 ### set constants if not existing
14 if(!exists("WORKING_DIRECTORY")){
15   WORKING_DIRECTORY <- "C:/Users/ams5wi/Bachelorarbeit/06_implementation/"
16 }
17 if(!exists("OUTPUT_DIRECTORY")){
18   OUTPUT_DIRECTORY <- paste0(WORKING_DIRECTORY, "output/")
19 }
20 if(!exists("SAMPLE_DIRECTORY")){
21   SAMPLE_DIRECTORY <- paste0(WORKING_DIRECTORY, "samples/")
22 }
23 if(!exists("PLOT_DIRECTORY")){
24   PLOT_DIRECTORY <- paste0(WORKING_DIRECTORY, "plots/generic/")
25 }
26 if(!exists("SAMPLE_SIZE")){
27   SAMPLE_SIZE <- c(100, 300, 600)
28 }
29 if(!exists("CLUSTER_TYPE")) {
30   CLUSTER_TYPE <- c("cdm", "dtw", "dtw_full", "fbc")
31 }
32
33
34 ### set working directory
35 setwd(WORKING_DIRECTORY)
36
37 for (type in CLUSTER_TYPE) {
38   for (no_samp in SAMPLE_SIZE) {
39
40     ### empty global environment and load data
41     cat("remove variables ", setdiff(ls(), c("no_samp", "SAMPLE_SIZE", "PLOT_DIRECTORY", "WORKING
42       _DIRECTORY",
43         "OUTPUT_DIRECTORY", "SAMPLE_DIRECTORY", "type", "
44           CLUSTER_TYPE")), "\n")
45     rm(list = setdiff(ls(), c("no_samp", "SAMPLE_SIZE", "PLOT_DIRECTORY", "WORKING_DIRECTORY",
46       "OUTPUT_DIRECTORY", "SAMPLE_DIRECTORY", "type", "CLUSTER_TYPE")))
47     source("clustering_administration_functions.R")
48     load(paste0(OUTPUT_DIRECTORY, type, no_samp, ".RData"))
49     invisible(loadSample(paste0(SAMPLE_DIRECTORY, "sample", no_samp, ".RData")))
50
51     ### get cluster vector
52     used_output <- output_names[grepl(pattern = "cluster", x = output_names)]
53     for (dat in data_names) {
54       for (out in used_output) {
55         ## plot 30 smoothed time series per cluster
56         savePlotCluster(clust = df[[dat]][[out]], output_file = paste0(PLOT_DIRECTORY, type, "_",
57           dat, "_", no_samp, "_", out), variable = "hwc",
58           preprocessing = "smoothed", preproc_args = 7, max_plot = 30)
59       }
60       for (k in k_val) {
61         ## plot medoids for every cluster
62         savePlotCentroids(cent = add_data[[dat]][["medoid"]][[paste0(k)]],
63           output_file = paste0(PLOT_DIRECTORY, type, "_", dat, "_", no_samp, "_",
64             medoids_k", k),
65           variable = "hwc", preprocessing = "smoothed", preproc_args = 7)
66       }
67     }
68   }
69 }
70
71 ### compute silhouette means and save in excel file
72 source("plot_silhouette.R")
73
74 #####
75 ## file:      plot_silhouette.R
76 ## last change: September 21, 2018
77 ## company:   Robert Bosch GmbH, CR\AEB2
78 ## author:    Amanda Schoefl
79 ## contact:   Stephanie Kaschewski
80 ## notes:     save silhouette values in excel file
81 #####
82
83 ### set constants if not existing
84 if(!exists("WORKING_DIRECTORY")){
85   WORKING_DIRECTORY <- "C:/Users/ams5wi/Bachelorarbeit/06_implementation/"
86 }
```

A. Sourcecode

```
13 }
14 if(!exists("OUTPUT_DIRECTORY")){
15   OUTPUT_DIRECTORY <- paste0(WORKING_DIRECTORY, "output/")
16 }
17 if(!exists("SAMPLE_DIRECTORY")){
18   SAMPLE_DIRECTORY <- paste0(WORKING_DIRECTORY, "samples/")
19 }
20 if(!exists("SAMPLE_SIZE")){
21   SAMPLE_SIZE <- c(100, 300, 600)
22 }
23 if(!exists("CLUSTER_TYPE")) {
24   CLUSTER_TYPE <- c("cdm", "dtw", "dtw_full", "fbc")
25 }
26
27
28 ## load required libraries and set working directory
29 setwd(WORKING_DIRECTORY)
30
31
32 ### empty global environment and load data
33 cat("remove variables ", setdiff(ls(), c("no_samp", "SAMPLE_SIZE", "PLOT_DIRECTORY", "WORKING_
34   DIRECTORY",
35                                     "OUTPUT_DIRECTORY", "SAMPLE_DIRECTORY", "type", "CLUSTER_
36   _TYPE")), "\n")
37 rm(list = setdiff(ls(), c("no_samp", "SAMPLE_SIZE", "PLOT_DIRECTORY", "WORKING_DIRECTORY",
38   "OUTPUT_DIRECTORY", "SAMPLE_DIRECTORY", "type", "CLUSTER_TYPE")))
39 source("clustering_administration_functions.R")
40
41 SAMPLE_SIZE <- c(100,300,600)
42 CLUSTER_TYPE <- c("dtw", "dtw_full", "cdm", "fbc")
43 sil <- vector("list", length = length(CLUSTER_TYPE))
44 names(sil) <- CLUSTER_TYPE
45 clus_alg_names <- c("hierarchical", "pam", "kmeans")
46
47 ### compute silhouette means
48 for(type in CLUSTER_TYPE) {
49   first_iteration <- T
50   for (no_samp in SAMPLE_SIZE) {
51     load(paste0(OUTPUT_DIRECTORY, type, no_samp, ".RData"))
52     ## call again in case that old functions are stored in loaded RData file
53     source("clustering_administration_functions.R")
54     ## initialize data frame with first column
55     if(first_iteration) {
56       sil[[type]] <- data.frame("k" = k_val)
57       first_iteration <- F
58     }
59     ## compute mean of silhouette values
60     for (dat in data_names) {
61       for (can in clus_alg_names) {
62         tmp_sil <- NULL
63         for(k in k_val){
64           tmp_sil <- c(tmp_sil, mean(df[[dat]][[paste0(can, "_sil_k", k)]]))
65         }
66         tmp_df <- data.frame(tmp_sil)
67         names(tmp_df) <- paste0(dat, "_", can, "_", no_samp)
68         ## Attention: cbind is computational expensive!
69         sil[[type]] <- addDfColumn(dframe = sil[[type]], column_to_add = tmp_df)
70       }
71     }
72   }
73 }
74
75
76 ### store df containing silhouette means in excel file
77 cat("save in excel file \n")
78 first_iteration <- T
79 for (type in CLUSTER_TYPE) {
80   if(first_iteration) {
81     write.xlsx(x = sil[[type]], file = paste0(OUTPUT_DIRECTORY, "sil_means.xlsx"), sheetName =
82       type, col.names <- T, row.names <- F, append <- F)
83     first_iteration <- F
84   }else{
85     write.xlsx(x = sil[[type]], file = paste0(OUTPUT_DIRECTORY, "sil_means.xlsx"), sheetName =
86       type, col.names <- T, row.names <- F, append <- T)
87   }
88 }
```

A. Sourcecode

```
1 #####
2 ## file: generate_samples.R
3 ## last change: September 27, 2018
4 ## company: Robert Bosch GmbH, CR\AEB2
5 ## author: Amanda Schoepl
6 ## contact: Stephanie Kaschewski
7 ## required: correct directories;
8 ## existing file "usableData.RData" in DATA_DIRECTORY
9 #####
10
11 ### set constants if not existing
12 if(!exists("WORKING_DIRECTORY")){
13   WORKING_DIRECTORY <- "C:/Users/ams5wi/Bachelorarbeit/06_implementation/"
14 }
15 if(!exists("SAMPLE_DIRECTORY")){
16   SAMPLE_DIRECTORY <- paste0(WORKING_DIRECTORY, "samples/")
17 }
18 if(!exists("DATA_DIRECTORY")){
19   DATA_DIRECTORY <- paste0(WORKING_DIRECTORY, "data/")
20 }
21 if(!exists("SAMPLE_SIZE")){
22   SAMPLE_SIZE <- c(100, 300, 600)
23 }
24
25
26 ### set working directory an load functions
27 setwd(WORKING_DIRECTORY)
28 source("clustering_administration_functions.R")
29
30 ### set parameters
31 percentage_of_na_allowed <- 0.05
32 max_value <- list("hwc"=600)
33
34 ### generate and store samples
35 for (amount in SAMPLE_SIZE) {
36   tmp <- newSample(amount, percentage_of_na_allowed, max_value, file = paste0(DATA_DIRECTORY, "
37     useableData.RData"))
38   storeSample(when=paste0(SAMPLE_DIRECTORY, "sample", amount, ".RData" ))
39 }
```

A.2. Modellierung

```

1  $ontext
2  file:      model_v13.gams
3  last change: December 16, 2018
4  company:   Robert Bosch GmbH, CR\AEB2
5  author:    Amanda Schoefl
6  contact:   Stephanie Kaschewski
7  required:  file input_v13.gdx generated by model_v13.R
8  $offtext
9
10 * lade von "model_v13.R" generierte Daten
11 $gdxin C:\Users\Amanda Schoefl\Documents\Studium\Bachelorarbeit\implementation\gams\model_v13\
    input_v13.gdx
12 set t, m;
13 parameter p(t), allowed_modi(t,m), restverh(m), gmax(m), a(t), g0, bmax, k_lin, d_lin, gzust, gab
    , b0, fassungsv;
14 $load t p m allowed_modi gmax restverh a g0 bmax k_lin d_lin gzust gab b0 fassungsv
15 $gdxin
16
17 * definiere und initialisiere zusaetzlich benoetigte Daten
18 sets
19     initsub(t)  Initialisierung fuer h
20     woche1(t)   erste Woche
21     woche4(t)   letzte Woche
22 ;
23 * braucht das Alias, um Rekursionen verwirklichen zu koennen,
24 * keine Ahnung warum aber es funktioniert
25 alias(t, sub);
26 initsub(t) = yes$(not mod(ord(t),12));
27 woche1(t) = yes$(ord(t) < 85);
28 woche4(t) = yes$(ord(t) > 252);
29 parameter
30     init(t)      um beim ersten Durchlauf nicht 0 zu bekommen
31 ;
32 init(t)=0;
33 init('1')=1;
34 p('1')=0;
35
36 $ontext
37 zu optimierende Variablen
38 $offtext
39 binary variable
40     h(t)         wird geheizt oder nicht
41     md(t,m)     in welchem Modus wird geheizt
42 ;
43 equation
44     e_md(t)     verhindere mehrfache Heizmodi gleichzeitig
45 ;
46 e_md(t).. sum(m, md(t,m)) =E= h(t);
47 * heize weder in Woche 1 noch Woche 4
48 h.fx(woche1) = 0;
49 h.fx(woche4) = 0;
50 * initialisiere Heizverhalten
51 h.l(t) = 0;
52 h.l(initsub) = 1;
53 h.l(woche1) = 0;
54 h.l(woche4) = 0;
55 md.l(t,m) = 0;
56 md.l(initsub, '2') = 1;
57 md.l(woche1, '2') = 0;
58
59 * zeige relevante Parameter im Output
60 display initsub, p, a, g0, restverh, bmax, fassungsv;
61
62 * definiere und initialisiere Hauptvariablen und Gleichungen
63 variable
64     K             Kostenfunktion
65     b(t)         Bakterienkonzentration
66     l(t)         Wachstumsrate lambda
67     g(t)         Temperatur in Grad
68 ;
69 b.up(t) = bmax;
70 b.lo(t) = 0;
71 b.fx('1') = b0;
72 g.lo(t) = g0;
73 g.l('1') = g0;
74 l.l(t) = 1;
75 l.lo(t) = 1;
76 equation
77     kost         Kostenfunktion
78     bakt(t)     Bakterienkonzentration
79     lambda(t)   Wachstumsrate
80     grad(t)     Temperatur
81 ;

```

A. Sourcecode

```

82
83
84 $ontext
85 Temperaturberechnung
86 $offtext
87 variable gnheiz(t) Temperatur wenn nicht geheizt wird
88           gpnh(t) Temperatur unter Beruecksichtigung der Taps wenn nicht
89           geheizt wird
90           gheiz(t) Temperatur wenn geheizt wird
91           gph(t) Temperatur unter Beruecksichtigung der Taps wenn geheizt wird
92           gkorrr(t) Korrekturterm damit die Temperatur nicht unter G0 faellt
93
94 gkorrr.lo(t) = 0;
95 equation e_gnheiz(t), e_gpnh(t), e_gph(t), e_gheiz(t);
96 * Nebenrechnungen
97 e_gnheiz(sub).. gnheiz(sub) =E= (1-h(sub))*(gpnh(sub)-GAB);
98 e_gpnh(sub).. gpnh(sub) =E= (1-a(sub))*g(sub-1)+a(sub)*G0;
99 e_gph(sub).. gph(sub) =E= (1-a(sub))*sum(m,md(sub,m)*gmax(m))+a(sub)*G0;
100 e_gheiz(sub).. gheiz(sub) =E= h(sub)*(gph(sub)-GAB);
101 * Hauptrechnung
102 grad(sub).. g(sub) =E= gheiz(sub)+gnheiz(sub)+init(sub)*G0+gkorrr(sub) ;
103
104 $ontext
105 Lambdaberechnung
106 $offtext
107 variable
108           lg(t) temperaturabhaengige Wachstumsrate
109
110 equation e_lg(t);
111 * Nebenrechnungen
112 e_lg(sub).. lg(sub) =G= k_lin*g(sub)+d_lin ;
113 * Hauptrechnung
114 lambda(sub).. l(sub) =E= lg(sub)*lg(sub);
115
116
117 $ontext
118 Bakterienkonzentrationsberechnung
119 $offtext
120 variable
121           bheff(t) ist die Desinfektion effektiv
122
123 equation e_bheff(t), e_h;
124 * Nebenrechnungen
125 e_bheff(sub).. bheff(sub) =E= 1-h(sub)*sum(m,md(sub,m)*allowed_modi(sub,m))*0.9;
126 * folgende Gleichung besagt, dass 1 mal effektiv geheizt werden muss
127 e_h.. sum(t, bheff(t)) =L= sum(t,1)-0.9;
128 * Hauptrechnung
129 bakt(sub).. b(sub) =E= (b(sub-1)*bheff(sub)*(1-a(sub))+b0*a(sub))*l(sub)
130                   +init(sub)*b0;
131
132 $ontext
133 Kostenfunktion
134 $offtext
135 variable gewicht(t) Gewichtung nach Heizenergie;
136 gewicht.lo(t) = 0;
137 equation e_gewicht(t);
138 * Nebenrechnungen
139 e_gewicht(sub).. gewicht(sub) =E= h(sub)*((sum(m,md(sub,m)*gmax(m))-
140                                           g(sub-1))+gkorrr(sub));
141 * Hauptrechnung
142 kost.. K =E= sum(sub, gewicht(sub));
143
144 $ontext
145 definiere Modell und starte Optimierung
146 $offtext
147 model energie /all/;
148 * Zeitberenzung
149 option resLim=3500;
150 * benutzter Solver
151 option minlp=scip;
152 solve energie minimizing K using minlp;

```

A. Sourcecode

```
1 #####
2 ## file:      model_v13.R
3 ## last change: December 16, 2018
4 ## company:   Robert Bosch GmbH, CR\AEB2
5 ## author:    Amanda Schoeﬂ
6 ## contact:   Stephanie Kaschewski
7 ## required:  file "nebenrechnungen_gams.R", correct directories
8 #####
9
10 ## load libraries
11 library("gdxrrw")
12 directory <- "C:/Users/Amanda Schoeﬂ/Documents/Studium/Bachelorarbeit/implementation/gams/model_
    v13/"
13 setwd(directory)
14 source(paste0(directory, "../common/nebenrechnungen_gams.R"))
15
16 ## initialize directories
17 gams_script <- "model_v13.gms"
18 gams_results <- "results_v13.gdx"
19 gams_input <- "input_v13.gdx"
20 igdx("C:/GAMS/win64/24.7")
21 file.remove(gams_input)
22
23 ## TRUE if new test data should be generated
24 ## FALSE if 'data_per_day.RData' containing 'days', 'taps_per_day', 'liter_per_day'
25 ## exists and should be used
26 compute_rnorm <- F
27
28 ## initialize parameters
29 gzust <- 15
30 g0 <- 40
31 bmax <- 5
32 fassungsv <- 300
33 gab <- temp_loss(Ta=18,Ti=50, ra=50, ri=40, lamb=0.035, vol=fassungsv, c=4.18)
34 b0 <- 1
35 taps <- NULL
36 if (compute_rnorm) {
37   days <- 30
38   tmp1 <- round(rnorm(days, mean=17, sd=7))
39   tmp1[tmp1<0] <- 0
40   taps_per_day <- tmp1
41   tmp <- round(rnorm(days, mean=109/17))
42   tmp <- taps_per_day*tmp
43   liter_per_day <- tmp
44   save(days, taps_per_day, liter_per_day, file = "data_per_day.RData")
45 } else {
46   load("data_per_day.RData")
47 }
48 heiztemperaturen <- c(50, 54, 58, 60, 70)
49 heizzeiten <- c(111, 27, 6, 4, 2)
50
51 ## compute needed parameters, see "nebenrechnungen_gams.R" for more information
52 setParam(g0=g0, gzust=gzust, heizzeiten=heizzeiten, heiztemperaturen=heiztemperaturen,
53   taps_per_day=taps_per_day, liter_per_day=liter_per_day, fassungsv=fassungsv,
54   taps=taps)
55 computeAll()
56 taps <- getWeightedTaps()
57 modi <- getAllowedModi()
58 TIMESET <- seq(1,length(taps))
59 MODISET <- seq(1,length(heiztemperaturen))
60
61 ## formatting parameters for gams usage
62
63 fassungsv_parameter <- list(name="fassungsv", type='parameter', ts='Fassungsvermoegen',
64   val=as.matrix(fassungsv))
65
66 g0_parameter <- list(name='g0', type='parameter', ts='Minimaltemperatur', val=as.matrix(g0))
67
68 b0_parameter <- list(name='b0', type='parameter', ts='Bakterienkonzentration im Frischwasser',
69   val=as.matrix(b0))
70
71 bmax_parameter <- list(name='bmax', type='parameter', ts='Maximal erlaubte Bakterienkonzentration
72   ',
73   val=as.matrix(bmax))
74
75 k_lin_parameter <- list(name='k_lin', type='parameter', ts='fuer lineare Fkt zur Approx von
76   Lambda',
77   val=as.matrix(en_gams$k))
78
79 d_lin_parameter <- list(name='d_lin', type='parameter', ts='fuer lineare Fkt zur Approx von
80   Lambda',
81   val=as.matrix(en_gams$d))
82
83 gzust_parameter <- list(name='gzust', type='parameter', ts='Temperatur des zustoemenden Wassers',
84   ',
85   val=as.matrix(gzust))
```

A. Sourcecode

```

82
83 gab_parameter <- list(name='gab', type='parameter', ts='feste Abnahme der Temperatur',
84                       val=as.matrix(gab))
85
86 t_set <- list(name='t', type='set', ts='Indexmenge Zeiteinheiten', uels=list(TIMESET))
87
88 m_set <- list(name='m', type='set', ts='Indexmenge Modi', uels=list(MODISET))
89
90 p_val <- array(dim=c(length(TIMESET),2))
91 p_val[,1] <- TIMESET
92 p_val[,2] <- taps
93 p_parameter <- list(name='p', type='parameter', ts='Taps', uels=list(TIMESET), val=as.array(p_val
94 ))
95
96 ## keine Ahnung was 'full' macht, weil nicht dokumentiert, aber es funktioniert nur mit
97 allowed_modi_parameter <- list(name='allowed_modi', type='parameter', ts='allowed modi', form='
98     full',
99     uels=c(list(TIMESET),list(MODISET)), val=as.matrix(modi))
100
101 gmax_val <- array(dim=c(length(MODISET),2))
102 gmax_val[,1] <- MODISET
103 gmax_val[,2] <- heiztemperaturen
104 gmax_parameter <- list(name='gmax', type='parameter', ts='Toetungstemperatur', uels=list(MODISET)
105 , val=as.array(gmax_val))
106
107 restverh_val <- array(dim=c(length(MODISET),2))
108 restverh_val[,1] <- MODISET
109 restverh_val[,2] <- heizzeiten/120
110 restverh_parameter <- list(name='restverh', type='parameter', ts='Toetungstemperatur', uels=list(
111 MODISET), val=as.array(restverh_val))
112
113 a_val <- array(dim=c(length(TIMESET),2))
114 a_val[,1] <- TIMESET
115 a_val[,2] <- getLiter()
116 a_parameter <- list(name='a', type='parameter', ts='ausstroemend durch Fassungsvermoegen (L)',
117     uels=list(TIMESET), val=as.array(a_val))
118
119 ## save parametrs in file gams_input
120 wgdx(gams_input, allowed_modi_parameter, t_set, m_set, p_parameter, gmax_parameter,
121     restverh_parameter, a_parameter, g0_parameter, bmax_parameter, k_lin_parameter,
122     d_lin_parameter, gzust_parameter, b0_parameter, gab_parameter, fassungsv_parameter)
123
124
125 #####
126 ## file: nebenrechnungen_gams.R
127 ## last change: December 16, 2018
128 ## company: Robert Bosch GmbH, CR\AEB2
129 ## author: Amanda Schoefl
130 ## contact: Stephanie Kaschewski
131 #####
132 en_gams <- new.env()
133
134 ### SET PARAMETERS
135 ## must be called first
136 #####
137 setParam <- function(g0, gzust, heizzeiten, heiztemperaturen, taps_per_day, liter_per_day,
138     fassungsvermoegen, taps=NULL, Ta=18,Ti=50, ra=50, ri=40, lamb=0.035, c=4.18){
139     rm(list=ls(pos=en_gams), pos=en_gams)
140     en_gams$g0 <- g0
141     en_gams$gzust <- gzust
142     en_gams$heizzeiten <- heizzeiten
143     en_gams$heiztemperaturen <- heiztemperaturen
144     en_gams$taps_per_day <- taps_per_day
145     en_gams$liter_per_day <- liter_per_day
146     en_gams$fassungsvermoegen <- fassungsvermoegen
147     en_gams$taps <- taps
148     en_gams$stank <- list(Ta=Ta,Ti=Ti, ra=ra, ri=ri, lamb=lamb, c=c)
149 }
150 #####
151
152 ### COMPUTE PARAMETERS
153 ## must be called second
154 #####
155 computeAll <- function(){
156     ## ATTENTION: order is important
157     computeGabn()
158     if(!exists("taps", where = en_gams)){
159         computeWeightedTaps()
160     }else if(is.null(en_gams$taps)){
161         computeWeightedTaps()
162     }
163     computeLiter()
164     computeAllowedModi()
165     fit_growth_rate()
166 }
167
168

```

A. Sourcecode

```

45
46 ### compute termic temperature loss per hour
47 computeGabh <- function(){
48   vol <- en_gams$fassungsvermoegen
49   h <- vol/(pi*en_gams$stank$ri*en_gams$stank$ri)
50   k <- en_gams$stank$lamb*2*pi*h/(4.18*vol*1000*(log(en_gams$stank$ra)-log(en_gams$stank$ri)))
51   en_gams$gabh <<- en_gams$stank$Ti-(exp(-k*120*60)*(en_gams$stank$Ti-en_gams$stank$Ta)+
52     en_gams$stank$Ta)
53 }
54
55 ### compute occuring taps per timestep
56 computeWeightedTaps <- function() {
57   if(!(exists("taps_per_day", where = en_gams))){
58     stop("please call setParam")
59   }
60   no_taps_day <- en_gams$staps_per_day
61   rm(taps, allowed_modi, pos=en_gams)
62   ## distribution of taps over one day
63   prob <- c(0.05/3, 0.05/3, 0.05/3, 0.28/2, 0.28/2, 0.25/3,
64     0.25/3, 0.25/3, 0.34/2, 0.34/2, 0.07, 0.01)
65   en_gams$staps <<- vector(length=length(no_taps_day)*12)
66   index <- seq(1,12)
67   for (no in no_taps_day) {
68     ## compute and correct number of taps: must be integer and sum up to no
69     tmp <- round(no*prob)
70     while(sum(tmp) != no) {
71       counter <- sample(seq(4,10), size=1)
72       if (sum(tmp) < no) {
73         tmp[counter] <- tmp[counter]+1
74         cat("Abschnitt", counter, " nach oben korrigiert")
75       }
76       if (sum(tmp) > no) {
77         tmp[counter] <- max(tmp[counter]-1,0)
78         cat("Abschnitt", counter, " nach unten korrigiert")
79       }
80     }
81     en_gams$staps[index] <<- tmp
82     index <- index+12
83   }
84 }
85
86 ### compute perencatage of tank capacity of water flowing out in each time step
87 computeLiter <- function() {
88   if(!(exists(c("liter_per_day", "fassungsvermoegen"), where=en_gams))) {
89     stop("please call 'setParam'")
90   }
91   if(!(exists("taps", where = en_gams))){
92     stop("please call 'getWeightedTaps' before")
93   }
94   liter_day <- en_gams$liter_per_day
95   fassungsvermoegen <- en_gams$fassungsvermoegen
96   liter <- vector(length = length(en_gams$staps))
97   index <- seq(1,12)
98   for (no in liter_day) {
99     # weniger rechenintensive Alternative: no_taps_day auch mit abspeichern und noch
100    # eine indexvariable einfuehren oder mit Dataframes arbeiten
101    liter[index] <- en_gams$staps[index]*no/max(1,(sum(en_gams$staps[index])))/fassungsvermoegen
102    index <- index+12
103  }
104  liter[liter>1] <- 1
105  en_gams$liter <<- liter
106 }
107
108
109 ### compute effective thermal disinfection possibilities for every time step
110 computeAllowedModi <- function() {
111   if(!(exists("taps", where = en_gams))){
112     stop("please call 'computeWeightedTaps' before")
113   }
114   if(!(exists("heizzeiten", where = en_gams))){
115     stop("please call 'setParam'")
116   }
117   dur <- en_gams$heizzeiten
118   dur[dur==0] <- 120
119   taps <- en_gams$staps
120   ## maximum of allowed taps +1 for each modi
121   ## ATTENTION: amount of taps must be SMALLER than max_tap
122   max_tap <- round(120/dur)
123   en_gams$allowed_modi <<- array(0, dim=c(length(taps),length(en_gams$heizzeiten)))
124   count_t <- 1
125   for (t in taps) {
126     count_m <- 1
127     for (m in max_tap) {
128       if (t < m){
129         en_gams$allowed_modi[count_t, seq(count_m, length(dur))] <<- 1
130         break

```

A. Sourcecode

```
131     }
132     count_m <- count_m+1
133   }
134   count_t <- count_t+1
135 }
136 en_gams$allowed_modi[,dur==120] <<- 0
137 }
138
139 ### approximate growth rate of bacteria
140 fit_growth_rate <- function(){
141   if(g0 < 37) {stop("Temperature g0 is too low. Must be at least 37 degree celsius.\n")}
142   temp <- c(21, 25,30,37,42, 46)
143   ## growth rate per hour
144   en_gams$k <<- 0.188971356/(37-46)
145   en_gams$d <<- 1.188971356-en_gams*k*37
146 }
147 #####
148
149 ### GET FUNCTIONS
150 #####
151 getWeightedTaps <- function(){
152   if(!exists("taps", where=en_gams)){
153     stop("please call computeWeightedTaps")
154   }
155   return(en_gams$taps)
156 }
157
158 getLiter <- function(){
159   if(!exists("liter", where=en_gams)){
160     stop("please call computeLiter")
161   }
162   return(en_gams$liter)
163 }
164
165 getAllowedModi <- function(){
166   if(!exists("allowed_modi", where=en_gams)){
167     stop("please call computeAllowedModi")
168   }
169   return(en_gams$allowed_modi)
170 }
171
172 getGabn <- function() {
173   if(!exists("gabn", where=en_gams)){
174     stop("please call computeGabn")
175   }
176   return(en_gams$gabn)
177 }
178
179
180 getTemp <- function(){
181   if(!exists("temp", where=en_gams)){
182     stop("please call setParam")
183   }
184   return(en_gams$temp)
185 }
186 #####
```

B. Silhouettenwerte

In allen folgenden Tabellen steht *ma* für moving average und *int* für die Rundung auf Ganzzahlen. Alle anderen Abkürzungen werden in der Arbeit erwähnt.

Dieser Abschnitt zeigt Silhouettenwerte verschiedener Partitionen, für deren Erstellung der DTW Abstand verwendet und nur ein Ausschnitt der Zeitreihen betrachtet wurde.

k	ma_log_hierarchical_600	ma_log_pam_600	ma_zscore_hierarchical_600
2	0.5435903584841999	0.58382167206350877	0.23564126850059883
3	0.49381401614389647	0.4876992511761627	0.15683889642678706
4	0.44736118035419231	0.4210823517665101	0.13186031689088187
5	0.39574858209320923	0.41130673893966052	8.0551669472359338E-2
6	0.38366929933539184	0.345617527108667	7.3726745738040042E-2
7	0.37106920168953905	0.25771835335147364	6.8781046923724748E-2
8	0.41072932421592367	0.21226834064250036	5.6883148139619769E-2
9	0.39707473798613208	0.20083727991951719	4.9831388291728326E-2
10	0.39098832708185238	0.23072332078988741	4.6515869141022903E-2
k	norm_paa_pam_300	ma_int_hierarchical_300	ma_int_pam_300
2	4.4535041683767734E-2	0.67477677969591832	0.60483844618408067
3	0.10207827662718533	0.56024897483049907	0.54790038688359222
4	4.6609807863889054E-2	0.60250969884214001	0.45165061667805573
5	4.5959696540089755E-3	0.5040596465148568	0.34899744882771661
6	1.1843056716020449E-2	0.50065544859985489	0.27853635158179613
7	5.0379898125359965E-2	0.49553873278766253	0.27765493631369459
8	4.5648464906757813E-3	0.47754276510986282	0.2261464581138532
9	2.1083742775631468E-2	0.47399697503298194	0.20379857721729058
10	2.3119866689352789E-3	0.42600435408413084	0.18590410620989856
k	ma_log_hierarchical_300	ma_log_pam_300	ma_zscore_hierarchical_300
2	0.49310058665485673	0.57873149368889099	0.27750067733264383
3	0.45569311425033393	0.50726034176248957	0.18099773754048035
4	0.42835499552701672	0.4658832152248592	0.13364701287137321
5	0.41640468883156601	0.38137786717946631	8.9173961344894137E-2
6	0.37570915621127976	0.34434677334378655	6.557399230239723E-2
7	0.38169709798457419	0.25059542779852462	4.3674809387074437E-2
8	0.34336246816443239	0.32300631448708589	3.2100849913972981E-2
9	0.31857727033667688	0.20637567027243589	2.4666971726858496E-2
10	0.30274099291036205	0.15922418454281112	1.4921638082404896E-2

B. Silhouettenwerte

k	ma_zscore_pam_300	ma_norm_hierarchical_300	ma_norm_pam_300
2	7.2211616711910398E-2	0.62565641228882796	0.16597721573231969
3	6.7378923420941345E-2	0.35139135488545786	7.1301894838100746E-2
4	2.8231614264309104E-2	0.2652926800044183	2.7641311852392648E-2
5	2.8205587229725756E-2	0.23722606616871847	1.4213482206474103E-2
6	4.2644704016564124E-2	0.20996708596270564	3.6941395704992323E-2
7	2.7477342109756126E-2	0.19849950405943287	2.583120268916253E-2
8	1.955223927176079E-2	0.16629285789550219	3.6417714111875998E-2
9	3.5014446322438322E-3	0.13103087493803123	2.7178736179910352E-2
10	2.5517405950812436E-2	9.3148607342287124E-2	7.5299044070184038E-3

k	paa_hierarchical_600	paa_pam_600	norm_paa_hierarchical_600
2	0.75595716510162136	0.50873955729047671	0.70485005689621882
3	0.63358744528181266	0.42764534270702859	0.52225474343462663
4	0.62696652932992036	0.44504023097910045	0.45337419311414828
5	0.57540164711938202	0.26968511995734246	0.42874931728171006
6	0.55764285974975247	0.23972945021288433	0.40148616709407509
7	0.54753607137373594	0.26032019262829981	0.36999872559104341
8	0.47653813569057468	0.15063018818915741	0.34600267250477507
9	0.47554046838622194	0.1646581699908049	0.33739352906793951
10	0.46494276428640602	0.18480536176852042	0.32783770435532844

k	paa_hierarchical_100	paa_pam_100	norm_paa_hierarchical_100
2	0.80212996254638824	0.58280149712324358	0.61270690744084044
3	0.6744482425022672	0.47139927685905231	0.49768102632756755
4	0.57277235693143613	0.29241361567635765	0.32124477999188333
5	0.49359486731908603	0.20834428780312303	0.27100419483262167
6	0.47616161439275745	0.19605015740394185	0.23913029054101334
7	0.439743223319492	0.16292286140829765	0.21581238531646196
8	0.37209436819528791	0.10340296418023087	0.21668479688435671
9	0.35787140458969052	6.1572274085850018E-2	0.19286829702302594
10	0.33613612171978846	0.11915689240438213	0.19095780325771433

k	ma_zscore_pam_100	ma_norm_hierarchical_100	ma_norm_pam_100
2	0.085490222	0.411492191	0.245044647
3	0.078514588	0.318137124	0.092753448
4	0.045890171	0.24735377	0.078061792
5	0.053584637	0.203941056	0.036485832
6	0.041197373	0.150178935	0.051117848
7	0.051955417	0.139077786	0.032796739
8	0.047847594	0.122359501	0.030460135
9	0.038479348	0.100074107	0.020964952
10	0.035976744	0.09164119	-0.012311231

B. Silhouettenwerte

k	paa_hierarchical_300	paa_pam_300	norm_paa_hierarchical_300
2	0.656876099	0.580722069	0.805835337
3	0.525871844	0.51290315	0.583343492
4	0.570382814	0.376473938	0.3641907
5	0.473918079	0.319577752	0.307625338
6	0.471294218	0.292231368	0.299363715
7	0.465959273	0.225874086	0.263465526
8	0.459676269	0.20347721	0.231318979
9	0.452530935	0.168727866	0.220359761
10	0.450396145	0.170391164	0.211004311

k	norm_paa_pam_300	ma_int_hierarchical_300	ma_int_pam_300
2	0.044535042	0.67477678	0.604838446
3	0.102078277	0.560248975	0.547900387
4	0.046609808	0.602509699	0.451650617
5	0.00459597	0.504059647	0.348997449
6	0.011843057	0.500655449	0.278536352
7	0.050379898	0.495538733	0.277654936
8	0.004564846	0.477542765	0.226146458
9	0.021083743	0.473996975	0.203798577
10	0.002311987	0.426004354	0.185904106

k	norm_paa_pam_600	ma_int_hierarchical_600	ma_int_pam_600
2	0.232129179	0.723838237	0.643439379
3	0.09725294	0.627111934	0.497388629
4	0.065187899	0.616458331	0.410218738
5	0.016793677	0.607974425	0.426223276
6	0.027420706	0.526204988	0.358780993
7	-0.003507437	0.480536378	0.308239756
8	-0.005424028	0.479248085	0.231781853
9	-0.002399641	0.478237243	0.177494858
10	0.032999122	0.472702069	0.177389265

k	ma_zscore_pam_600	ma_norm_hierarchical_600	ma_norm_pam_600
2	0.065457882	0.593653933	0.054295916
3	0.040949309	0.402722104	0.071504804
4	0.041524672	0.353221849	0.055068555
5	0.035317888	0.348092449	0.071608074
6	0.020833841	0.289464356	0.054978562
7	0.028907456	0.258873797	0.029225966
8	0.026709897	0.245904042	0.047902237
9	0.020355516	0.223618787	0.001511492
10	0.016494962	0.210008667	0.027110712

B. Silhouettenwerte

Dieser Abschnitt zeigt Silhouettenwerte verschiedener Partitionen, für deren Erstellung der DTW Abstand verwendet und die ganzen Zeitreihen betrachtet wurde.

k	paa_hierarchical_100	paa_pam_100	norm_paa_hierarchical_100
2	0.78690375	0.617846931	0.629088456
3	0.660835586	0.409102787	0.547054545
4	0.553990611	0.332162068	0.359015889
5	0.397971755	0.264035293	0.316619322
6	0.37872372	0.267845986	0.31224202
7	0.341979031	0.200802133	0.280226056
8	0.334690693	0.096183394	0.250283925
9	0.321678672	0.202192675	0.234334561
10	0.317738451	0.105193196	0.190728111

k	norm_paa_pam_100	ma_int_hierarchical_100	ma_int_pam_100
2	0.231563254	0.801353856	0.61491077
3	0.34042517	0.607051964	0.440184995
4	-0.015030969	0.588230321	0.32107895
5	-0.002773819	0.389014113	0.179700263
6	0.036878949	0.365352611	0.348654942
7	0.129803005	0.446506687	0.17322187
8	0.005306933	0.4033064	0.068376038
9	0.054003354	0.391146055	0.190335946
10	-0.026114782	0.335358468	0.16731839

k	ma_log_hierarchical_100	ma_log_pam_100	ma_zscore_hierarchical_100
2	0.569156741	0.52752527	0.134628852
3	0.475824937	0.497189495	0.111654762
4	0.419617276	0.452601002	0.091647406
5	0.442989811	0.265302928	0.084640714
6	0.425977151	0.203306775	0.073564616
7	0.428373443	0.280282445	0.071602439
8	0.42141911	0.251328478	0.06398229
9	0.404000123	0.111992032	0.060587963
10	0.366955433	0.198452613	0.054002542

k	ma_zscore_pam_100	ma_norm_hierarchical_100	ma_norm_pam_100
2	0.124859774	0.565710187	0.131220565
3	0.047209145	0.437609517	0.110281876
4	0.031307516	0.177299442	0.041200563
5	0.036672134	0.140674977	0.055826417
6	0.046177418	0.132621045	-0.004603862
7	0.018351383	0.101951131	0.012699784
8	0.014216553	0.095972426	0.039823941
9	0.001751011	0.068509529	0.014602805
10	0.003076183	0.057886101	-0.008100491

B. Silhouettenwerte

k	paa_hierarchical_300	paa_pam_300	norm_paa_hierarchical_300
2	0.669687191	0.554084644	0.705025501
3	0.619176332	0.524713699	0.48571297
4	0.601975507	0.234556264	0.440755359
5	0.535764046	0.344689383	0.393702405
6	0.510938223	0.235747314	0.321866263
7	0.509952912	0.20843981	0.271388109
8	0.506334484	0.192707717	0.241095721
9	0.504691194	0.16797189	0.205958216
10	0.445691084	0.10864824	0.187659238

k	norm_paa_pam_300	ma_int_hierarchical_300	ma_int_pam_300
2	0.197164645	0.679892631	0.675404228
3	0.06278448	0.632327225	0.548071582
4	0.038559968	0.614359368	0.421988478
5	-0.007970782	0.554762525	0.413126482
6	0.037955248	0.527046632	0.257555777
7	0.087155096	0.519253448	0.245831515
8	-0.014688505	0.518270543	0.241872471
9	0.024033398	0.516617352	0.174593818
10	0.023033413	0.392354196	0.225517608

k	ma_log_hierarchical_300	ma_log_pam_300	ma_zscore_hierarchical_300
2	0.562787204	0.588367086	0.149308181
3	0.396679413	0.499589767	0.093613539
4	0.444172827	0.411278615	0.067005923
5	0.40360914	0.370764568	0.049830548
6	0.383387217	0.281079756	0.032424851
7	0.374143655	0.261672276	0.008717679
8	0.384691701	0.215487124	0.002970691
9	0.37548328	0.188933202	0.028564569
10	0.347161929	0.194511145	0.024615862

k	ma_zscore_pam_300	ma_norm_hierarchical_300	ma_norm_pam_300
2	0.046934841	0.454345275	0.036783132
3	0.040742406	0.271914592	0.052523918
4	0.025495612	0.220666833	0.042577665
5	0.026349733	0.177167939	0.022529439
6	0.020773557	0.155731781	0.037554194
7	0.013964487	0.144160662	-0.006144064
8	0.02508423	0.114290619	0.028406647
9	0.005461093	0.110770205	0.022986581
10	0.013205803	0.095685817	0.033212812

B. Silhouettenwerte

k	paa_hierarchical_600	paa_pam_600	norm_paa_hierarchical_600
2	0.698333717	0.553290801	0.558138595
3	0.60871067	0.489286987	0.512492281
4	0.589545361	0.370056564	0.471176527
5	0.495135462	0.341792051	0.411940661
6	0.491357872	0.290604079	0.392724645
7	0.491602266	0.315727678	0.358510955
8	0.479517162	0.200016784	0.356196164
9	0.478117633	0.171502956	0.345496345
10	0.441727447	0.162387637	0.330192189

k	norm_paa_pam_600	ma_int_hierarchical_600	ma_int_pam_600
2	0.140274488	0.651739839	0.520203777
3	0.016553231	0.589253709	0.519480772
4	0.108241983	0.519277032	0.424672931
5	0.049915572	0.5150912	0.298388685
6	0.021211325	0.499487447	0.317932885
7	-0.005954406	0.496983231	0.289344821
8	0.062480192	0.474823073	0.246884636
9	1.87482E-05	0.472571102	0.20384797
10	0.022144815	0.471851576	0.20835066

k	ma_log_hierarchical_600	ma_log_pam_600	ma_zscore_hierarchical_600
2	0.554891807	0.589145822	0.176532846
3	0.469812232	0.487016561	0.138571576
4	0.484573083	0.403630662	0.120414283
5	0.435750091	0.378916819	0.104562093
6	0.420058949	0.275400661	0.090735483
7	0.403489842	0.28347685	0.062165051
8	0.402776955	0.247776901	0.061231185
9	0.32668229	0.264025812	0.055442319
10	0.314239951	0.17312689	0.05349961

k	ma_zscore_pam_600	ma_norm_hierarchical_600	ma_norm_pam_600
2	0.095912959	0.372438839	0.082554453
3	0.052390039	0.240072298	0.071858225
4	0.055497393	0.211148614	0.048716523
5	0.030141991	0.159047198	0.035679641
6	0.038044512	0.155143818	0.020106003
7	0.03131071	0.150646228	0.043301113
8	0.026093189	0.144474752	0.025573679
9	0.03417414	0.142478845	0.004568548
10	0.025805613	0.140967741	0.024121535

B. Silhouettenwerte

Dieser Abschnitt zeigt Silhouettenwerte verschiedener Partitionen, für deren Erstellung der CDM Abstand verwendet und die ganzen Zeitreihen betrachtet wurde.

k	norm_hierarchical_100	norm_pam_100	int_hierarchical_100
2	0.022013284	0.022013284	0.03129681
3	0.019707595	0.019157768	0.025750163
4	0.019166189	0.002129866	0.018616747
5	0.01886633	0.000725889	0.017004165
6	0.018463464	0.000703511	0.015769483
7	0.017956476	0.000624507	0.012912277
8	0.017800898	0.00066579	0.012557384
9	0.017333204	0.000751436	0.011799838
10	0.017051693	0.000655494	0.01082031

k	int_pam_100	ma_int_hierarchical_100	ma_int_pam_100
2	0.028503982	0.068983789	0.066356654
3	0.018812155	0.064229166	0.045178537
4	0.015234525	0.058016521	0.025170117
5	0.010451895	0.039855832	0.024734744
6	0.010493191	0.033651767	0.017789214
7	0.009538082	0.029814823	0.012371821
8	0.005826901	0.028979938	0.008170967
9	0.005572791	0.022544733	0.008916633
10	0.004764675	0.021747468	0.008200075

k	ma_zscore_hierarchical_100	ma_zscore_pam_100	ma_norm_hierarchical_100
2	0.005608779	0.000122264	0.007737174
3	0.003725106	4.00119E-05	0.002404497
4	0.003164802	-7.61213E-05	0.001613906
5	0.002380542	-0.000132577	0.001305644
6	0.001985748	-0.000286695	0.000854259
7	0.00173225	-0.000214065	0.00045389
8	0.001436146	-0.000190431	0.000269973
9	0.001072468	-0.000224092	0.000226191
10	0.000747037	-0.000191976	0.000182231

k	ma_norm_pam_100	ma_log_hierarchical_100	ma_log_pam_100
2	0.000915311	0.00685863	0.00743697
3	-0.000593387	0.005992254	0.006990037
4	-0.001240389	0.004893328	0.003768676
5	-0.001291705	0.005141384	0.002920843
6	-0.001231077	0.004536127	0.001984425
7	-0.001304828	0.003494614	0.001339201
8	-0.001422347	0.003173228	0.001297169
9	-0.00141641	0.002788444	0.001262757
10	-0.001389401	0.002699868	0.001146752

B. Silhouettenwerte

k	norm_hierarchical_300	norm_pam_300	int_hierarchical_300
2	0.022211535	0.022211535	0.058337122
3	0.020688073	0.002134019	0.032050901
4	0.019370228	0.000107577	0.017436222
5	0.019045862	-0.000447414	0.016274022
6	0.018315099	-0.000548132	0.014579367
7	0.018153884	-0.000409241	0.013169503
8	0.018029798	-0.000348865	0.0105135
9	0.01801146	-4.54481E-05	0.010086588
10	0.017992821	0.000164456	0.010573421

k	int_pam_300	ma_int_hierarchical_300	ma_int_pam_300
2	0.024785894	0.058398713	0.06273213
3	0.018968489	0.05641129	0.04988597
4	0.013841453	0.04436238	0.029052833
5	0.010903967	0.041019388	0.025397066
6	0.007806689	0.025454119	0.021777512
7	0.005663483	0.023706074	0.017499307
8	0.003835804	0.023174104	0.012960577
9	0.003404608	0.021440304	0.011972631
10	0.002978871	0.021170278	0.010663303

k	ma_zscore_hierarchical_300	ma_zscore_pam_300	ma_norm_hierarchical_300
2	0.007265899	-5.18972E-05	0.00430968
3	0.00375655	-0.000116962	0.002942268
4	0.002625965	-0.000140058	0.002008681
5	0.002164822	-0.000219013	0.001332837
6	0.001493792	-0.000288163	0.000942656
7	0.001091762	-0.000287666	0.000668215
8	0.0009332	-0.000393929	0.000352947
9	0.000763456	-0.00042057	0.000271345
10	0.000512755	-0.00041848	9.66839E-05

k	ma_norm_pam_300	ma_log_hierarchical_300	ma_log_pam_300
2	-4.14528E-05	0.00746513	0.007194913
3	-9.24759E-05	0.006863304	0.006034132
4	-0.00017895	0.006194769	0.004220248
5	-0.000194228	0.005933466	0.003793855
6	-0.000195877	0.004987481	0.003088422
7	-0.000203149	0.003914199	0.002144252
8	-0.000254918	0.003659136	0.001546414
9	-0.000470593	0.003517994	0.001500307
10	-0.000498889	0.003367358	0.001234899

B. Silhouettenwerte

k	norm_hierarchical_600	norm_pam_600	int_hierarchical_600
2	0.021691916	0.021524651	0.03650858
3	0.020475615	0.002085957	0.031392281
4	0.019912436	0.000162384	0.018946943
5	0.019598649	4.95661E-05	0.014874912
6	0.019113638	9.53354E-05	0.013734427
7	0.018796398	5.38915E-05	0.012169952
8	0.01869328	6.87121E-05	0.01187172
9	0.018701429	0.000130199	0.010535856
10	0.018238956	5.32225E-05	0.010004052

k	norm_paa_pam_600	ma_int_hierarchical_600	ma_int_pam_600
2	0.140274488	0.651739839	0.520203777
3	0.016553231	0.589253709	0.519480772
4	0.108241983	0.519277032	0.424672931
5	0.049915572	0.5150912	0.298388685
6	0.021211325	0.499487447	0.317932885
7	-0.005954406	0.496983231	0.289344821
8	0.062480192	0.474823073	0.246884636
9	1.87482E-05	0.472571102	0.20384797
10	0.022144815	0.471851576	0.20835066

k	int_pam_600	ma_int_hierarchical_600	ma_int_pam_600
2	0.035485967	0.060836239	0.062237191
3	0.02166362	0.052963509	0.042764149
4	0.014644926	0.046339445	0.030185696
5	0.011524929	0.042987327	0.021855684
6	0.007122616	0.029898554	0.018313361
7	0.005829309	0.025368343	0.015843148
8	0.005306033	0.024484147	0.013442788
9	0.004275544	0.020524465	0.011808123
10	0.003726885	0.019784565	0.009963713

k	ma_zscore_hierarchical_600	ma_zscore_pam_600	ma_norm_hierarchical_600
2	0.01019191	2.89536E-05	0.00722228
3	0.007650855	-3.79376E-05	0.004495595
4	0.005223772	-8.27223E-05	0.003688319
5	0.003551311	-0.00011999	0.002762906
6	0.003004039	-0.000179507	0.002113195
7	0.002720001	-0.000279191	0.001596663
8	0.002402569	-0.000296314	0.001325195
9	0.00200324	-0.000326108	0.001120844
10	0.001831872	-0.000349305	0.000855465

B. Silhouettenwerte

k	ma_norm_pam_600	ma_log_hierarchical_600	ma_log_pam_600
2	3.62423E-05	0.007624586	0.007257355
3	-8.80869E-05	0.006623842	0.006405561
4	-0.000113373	0.005560775	0.004221104
5	-0.000219056	0.005037343	0.002996673
6	-0.000263222	0.004759171	0.002731768
7	-0.000280642	0.004109861	0.002036207
8	-0.00035368	0.003819163	0.001669467
9	-0.000301589	0.003759293	0.001387248
10	-0.000288091	0.003190954	0.001217093

B. Silhouettenwerte

Dieser Abschnitt zeigt Silhouettenwerte verschiedener Partitionen, für deren Erstellung die Euklidische und Manhattan Metrik verwendet und features der Zeitreihen berechnet wurden.

k	norm_hierarchical_100	norm_pam_100	int_hierarchical_100
2	0.022013284	0.022013284	0.03129681
3	0.019707595	0.019157768	0.025750163
4	0.019166189	0.002129866	0.018616747
5	0.01886633	0.000725889	0.017004165
6	0.018463464	0.000703511	0.015769483
7	0.017956476	0.000624507	0.012912277
8	0.017800898	0.00066579	0.012557384
9	0.017333204	0.000751436	0.011799838
10	0.017051693	0.000655494	0.01082031

k	euclidean_hierarchical_600	euclidean_pam_600	euclidean_kmeans_600
2	0.71940456	0.401648102	0.577335179
3	0.691021806	0.243287114	0.410554849
4	0.648172825	0.301185766	0.298282591
5	0.635580835	0.328845058	0.263308174
6	0.539270523	0.267731015	0.300673699
7	0.434277618	0.262402124	0.266469684
8	0.400582181	0.279450595	0.220941721
9	0.353344554	0.243964637	0.222636074
10	0.343952848	0.22457353	0.231676118

k	manhattan_hierarchical_300	manhattan_pam_300	manhattan_kmeans_300
2	0.702509941	0.385565211	0.530456076
3	0.679855363	0.296575222	0.456913328
4	0.639079131	0.333508473	0.307331616
5	0.524131786	0.284510777	0.308322867
6	0.490677905	0.247625984	0.289001798
7	0.429853674	0.222314988	0.199693913
8	0.423422299	0.21327798	0.230790434
9	0.426994139	0.239365924	0.143888446
10	0.361053572	0.246482317	0.188970955

B. Silhouettenwerte

k	euclidean_hierarchical_300	euclidean_pam_300	euclidean_kmeans_300
2	0.68111037	0.352823514	0.501040549
3	0.668769888	0.276912181	0.447867205
4	0.618385818	0.33214097	0.329178142
5	0.614302079	0.263301836	0.329048976
6	0.483858019	0.29236044	0.34009385
7	0.462892404	0.254049802	0.189311022
8	0.476916713	0.270985563	0.167255453
9	0.462829928	0.222694069	0.217979511
10	0.460087574	0.240561203	0.209463462

k	manhattan_hierarchical_100	manhattan_pam_100	manhattan_kmeans_100
2	0.74208731	0.318340101	0.777323549
3	0.58877923	0.382909388	0.462288513
4	0.533925029	0.30562615	0.404917174
5	0.488128655	0.316160537	0.300650168
6	0.398839242	0.254610841	0.235542086
7	0.392955679	0.261836396	0.199345517
8	0.377533656	0.275971483	0.165920766
9	0.363681215	0.282086257	0.188381881
10	0.361801417	0.272756545	0.154606393

k	euclidean_hierarchical_100	euclidean_pam_100	euclidean_kmeans_100
2	0.701438231	0.348059974	0.623478759
3	0.633241336	0.404040382	0.554141902
4	0.58296116	0.22810988	0.306350648
5	0.504431008	0.249996201	0.253252149
6	0.499680657	0.235275347	0.271682122
7	0.377761463	0.236140379	0.215907408
8	0.365078781	0.258129225	0.219135462
9	0.320789479	0.274146229	0.20139385
10	0.323923167	0.250404493	0.183585853