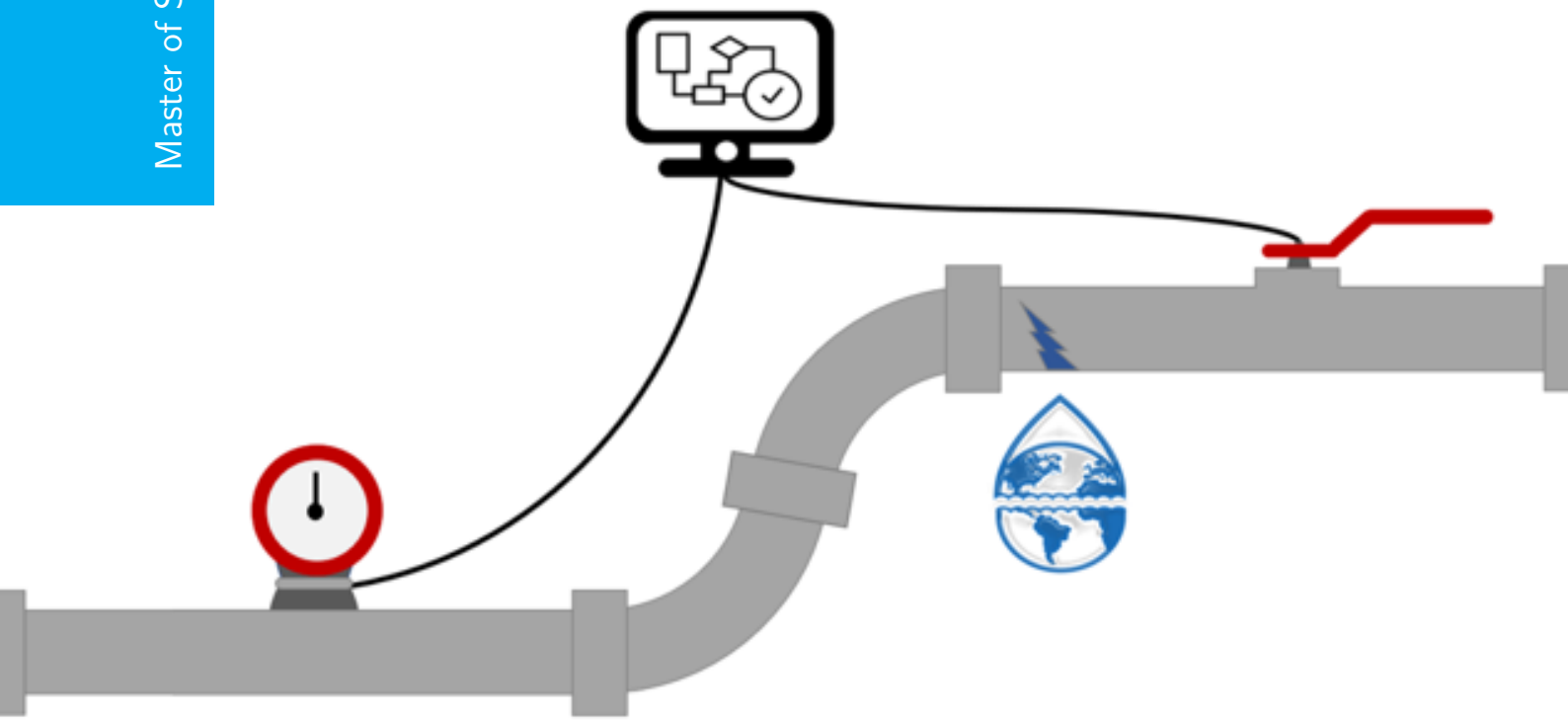
A Bayesian Approach for Active Fault Isolation with an Application to Leakage Localization in Water Distribution Networks

G. van Lagen

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

A Bayesian Approach for Active Fault Isolation with an Application to Leakage Localization in Water Distribution Networks

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

G. van Lagen

February 4, 2020

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

A Bayesian Approach for Active Fault Isolation with an Application to Leakage Localization in Water Distribution Networks

by

G. van Lagen

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: February 4, 2020

Supervisor(s):

dr. P. Mohajerin Esfahani

dr.ir. E. Abraham

Reader(s):

prof.dr.ir. B. de Schutter

Abstract

This thesis deals with an active fault isolation method applied to water distribution networks (WDN) in order to locate leaks. The method relies on the classification of observed residuals to a discrete set of hypothetical faults. The residuals reflect how much output measurements in the presence of a fault deviate from estimated values using a predictive fault-free model. Due to parametric uncertainties, the residuals are random vectors that follow unknown probability distribution functions (PDF). The residual PDFs corresponding with the considered faults are approximated using smooth kernel density estimation (KDE). They are used to calculate the a posteriori probability of each hypothesis, given the observed residuals, by applying Bayes' rule. The difficulty to classify observed residuals to the right fault comes from the overlap between residual PDFs. An active algorithm is introduced that proactively minimises the joint overlap between them by designing optimal control inputs. Due to physical limitations on control inputs and depending on the intensity of parametric uncertainties, full separation, and hence fault isolation, cannot be guaranteed for all observed residuals. Therefore the procedure is iterated over multiple time steps, where the a posteriori probabilities of the previous time step serve as the a priori probabilities for the next time step. The method is applied to locate leaks in a benchmark WDN for different intensities of uncertainty in demands and leak magnitude. Improvements in performance are observed in comparison to the best considered non-invasive method from literature in terms of reliability, speed and robustness.

Glossary

List of Acronyms

DMA	District Metered Area
PDF	Probability Distribution Function
WDN	Water Distribution Network
KDE	Kernel Density Estimation
PRV	Pressure Reducing Valve
FDI	Fault Detection and Isolation
FDA	Fisher Discriminant Analysis
AFD	Active Fault Diagnosis
PFD	Passive Fault Diagnosis
DW	Darcy-Weisbach
HW	Hazen-Williams

Table of Contents

List of Figures

List of Tables

Preface & Acknowledgements

The idea to work on the implementation of Fault Detection and Isolation (FDI) methods on Water Distribution Networks (WDNs) comes from a collaboration between my supervisors dr. ir. P. Mohajerin Esfahani and dr. ir. E. Abraham. The assignment was to improve existing methods for leak detection and localization in WDNs by application of these novel techniques. This subject from the faculty of Civil Engineering illustrates how widely applicable and powerful the system theoretic knowledge gained as a control engineer can be. I accepted this unique challenge of testing and extending my skills a little outside of my comfort zone. Key to this decision was that in these times of increasing water shortage I like to contribute to a reduction of the vast amounts of leakage within WDNs around the globe. Apart from other negative side effects, the fact that the elixir of life seeps away from those who need it, due to badly maintained networks, cannot just be swallowed. Therefore it is important that science does its job to extend and combine existing knowledge essential to solve a problem of this scale. I do not have the illusion to solve a global problem on my own, but hope this work will contribute to a more sustainable world in which resources are more carefully treated.

I would like to thank my supervisors dr. ir. P. Mohajerin Esfahani and dr.ir. E. Abraham for their valuable suggestions and outstanding support throughout the roller coaster of the final thesis project. The former for his expertise in the field of Fault Detection and Isolation and Machine Learning, which I gratefully have been able to use. The latter for his bridging knowledge between Control Engineering and Civil Engineering, which was key to understand the matter properly and being moved double-quick to the cutting edge of the subject.

A huge thank you to my wife, family and friends for their unconditional support. Without all of you I would have had a hard time to cross the finish line.

Delft, University of Technology — G. van Lagen
February 4, 2020

"Beware of little expenses. A small leak will sink a great ship."
— *Benjamin Franklin*

Chapter 1

Introduction

One of the main challenges for water utilities is the diagnosis and control of leakage from an ageing water distribution network (WDN). The early detection and management of leaks, in addition to reducing cost in non-revenue water and conserving energy, is critical to mitigate deterioration of pipes and surrounding infrastructure. Water loss due to leakage varies for well and poorly maintained networks respectively between 5 and 50 per cent of the supplied volume [2]. Reduction beyond the economically optimal level of about 15 per cent [3] is further motivated by stringent regulations and imminent risks. One risk is a poorer water quality due to temporarily negative pressures that allow intrusion of pollutants into the network, potentially jeopardising public health [4]. A further threat is that incipient leaks tend to gradually grow in size, eventually into pipe bursts, which can render (a part of) the network inoperable and result in infrastructural and residential damage as well as economical losses due to flooded areas. Also, leaks can cause destructive and dangerous sink holes due to washed away soil [5]. Finally, a part of the answer to global issues of rapid urbanisation and increasing water scarcity is to reduce the annual global loss of 32 billion m^3 of potable water [4].

Leakage analysis includes the tasks of fault detection, isolation, identification and estimation. These techniques respectively involve the determination of whether or not a leak is present, if so, to localise its location, type and magnitude [6]. This thesis focuses on leak localisation techniques.

Different methods to trace down the location of leaks have been proposed in literature and applied successfully. Conventional techniques include random and regular sounding surveys using listening sticks and acoustic loggers, and step-testing of District Metered Areas (DMAs) through gradual valve closures [7], where DMAs are sub-systems that can be analytically isolated through segregation of the Water Distribution Network (WDN) by means of (dynamic) boundary valves and metering the flows at remaining open connections [8, 9]. More advanced methods like leak noise correlators, pig-mounted acoustic sensing and gas-injection techniques [10] are the most precise at locating leaks. However all these techniques come with expensive equipment cost and are man-hour intensive, and so are not scalable. In ad-

dition, the suppression of leakage sound signatures by reduced pressures in active pressure management has also made these methods of limited application [7, 10].

Recent approaches use model-based analysis of near real-time telemetry data from pressure sensors and flow metres distributed over the network and rely on a calibrated non-linear hydraulic model introduced in [11] to predict steady-state pressures at sensed nodes. Two conventional methods to isolate leaks are based on structured and directional residuals. Structured residuals are constructed to be only sensitive to a subset of leak locations and based on a threshold indicate whether or not a leak has occurred there, whereas directional residuals are constructed by taking the difference between predicted outputs using a leakless model and measured pressure values, after which the angles between the observed residual vector and known signatures for a discrete set of possible leak locations are calculated by projecting it on the sensitivity matrix, which contains the corresponding signature directions [12]. In the remainder of this work, when the term "residual" is used, we refer to the directional type of residual.

As the user demands and leak magnitude incorporated into the network models are uncertain, the outputs and hence the residual vector consists of random variables. This uncertainty can be captured by bounded sets like zonotopes or probability distribution functions (PDF) [13]. Advanced leak localisation methods apply machine learning techniques like k-nearest neighbors, neuro-fuzzy and Bayesian classifiers as well as Fisher Discriminant Analysis (FDA) to classify observed residuals to one of the possible leak locations, which has shown best results when applied over multiple time steps [14–16]. The hardness to classify an observed residual to a leak location arises from the overlap between residual sets due to uncertainties, such that leak isolation cannot be achieved under all possible observed residuals.

A drawback of the advanced methods is that they rely on nominal input-output and so are passive in a sense. Therefore in this work we present a stochastic active fault diagnosis algorithm for designing active pressure control inputs to minimise the probability of misisolation, i.e. the overlap between residual sets aka Bayes error, for faster and more reliable leak isolation [13]. Where pressure control inputs are usually regulated at a minimum level using pressure reducing valves (PRV) [9], we show that PRVs can also be used to enhance leakage diagnosis. Due to physical limits and regulatory constraints, the inputs to the network are bounded. Hence it is plausible that residual sets cannot be fully separated, such that isolation is not guaranteed. However, by minimising the Bayes error together with iteratively applying Bayes' rule over a time horizon, leak diagnosis becomes more reliable and is sped up compared to passive methods. At night time when user demand is low [17],the proposed algorithm is applied to the benchmark Hanoi network for different degrees of uncertainty. Its performance is compared to that of the passive Bayesian classification method proposed in [14]. The residual distributions are estimated by means of extensive Monte Carlo simulations, which realisations are smoothed out using Kernel Density Estimation (KDE) [18].

The thesis is structured as follows. In Chapter 2 the working principle of a WDN and other relevant basic principles are described. In Chapter 3 the active leak isolation problem is stated and an active algorithm is proposed and elaborated for a District Metered Area (DMA). Chapter 4 provides a case study in which the algorithm is tested on different scenarios for the benchmark Hanoi network and compared against a cutting edge method. Finally, in Chapter 5 conclusions are drawn and suggestions for future work are provided.

Chapter 2

Water Distribution Networks

In ancient Rome a system of aqueducts was used to supply fountains and gardens with fresh water from a source outside the city. Using a small inclination along the distribution paths, the Romans managed the water to flow in the intended directions [19, pp. 1.6-1.8]. Nowadays the distribution of tap water to customers in urban areas is executed in a similar fashion.

2-1 Water Supply System

Figure 2-1 provides a schematic representation of a common water supply system. As shown raw or fresh water is extracted from a source such as groundwater or surface water from a lake, reservoir or river. After transport to a treatment plant, the water is made potable and then stored in a reservoir.

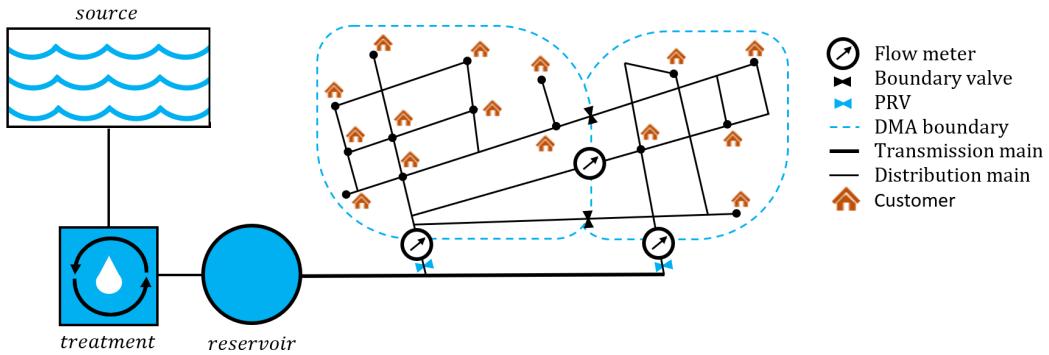


Figure 2-1: Schematic representation of a water supply network. Raw water of a fresh water source is treated and stored in a reservoir. From there on the water is conveyed through a transmission main under high pressure. Then using a Pressure Reducing Valve (PRV) the inlet pressure of the connected distribution mains is regulated through which the water is supplied to customers. Large networks are commonly (temporarily) divided into DMAs that can be analyzed independently. [1, pp. 2-3]

2-1-1 Water Distribution Network (WDN)

The stock of potable water in the reservoir is distributed through a WDN to connected consumers. Components like pipes, pumps and different types of valves are combined to an interconnecting structure between the reservoir and consumers, such that their demands at any time at connected places can be met. Water flows from the reservoir through the larger dimensioned transmission main into smaller distribution mains to which consumers are connected. Some types of consumers are homeowners, industrial and shopping establishments, etc., who each have a typical water demand pattern through the day and season. [1, pp. 2-3]

2-1-2 District Metered Area (DMA)

When the size of a WDN grows, it is common to impose an underlying substructure on it by dividing it into multiple DMAs, which was first introduced in the United Kingdom in 1980 [2] and is succesfully used internationally [20]. This segregation is implemented by (temporarily) closing off connections between DMAs using (dynamic) boundary valves and measuring the flows of remaining open connections and the inlet feed supplied by a transmission main. In this way the boundary of each DMA is determined, and therefore they can be analytically isolated and approached as independent networks. There are two main advantages with respect to the undivided network: (i) Using a Pressure Reducing Valve (PRV) at the inlet of each DMA, with which the inlet pressures can be regulated, controlling the pressure throughout the network becomes easier; (ii) Analysis regarding the estimation of the leakage magnitude and the leakage location can be performed on the smaller zones within the WDN, reducing the complexity of these tasks [20], [9]. Dynamic boundary valves with which a WDN can be temporarily segregated during analysis are preferable to static ones, as the latter induces drawbacks as possible water quality incidents caused by stagnant water at artificially created dead ends and a reduced resilience to failure, like pipe breaks, due to the decrease in supply paths, which in itself also induces suboptimal pressure management [9], [21].

2-1-3 Hydraulic head

The Romans constructed their conduits under a small gradient to make water flow in intended directions. This method relies on the fact that, due to gravity, water flows from point i to j, if a potential of energy is created by differences in elevation that is large enough to overcome the resistance of the trajectory in between these points. Much like the Romans constructed their aqueducts based on decreasing elevation head, a WDN, and thus DMAs, are nowadays designed on the principle that water tends to flow in direction of decreasing piezometric or hydraulic head h in $[m]$. At point i the hydraulic head can be written as:

$$h_i = \psi_i + z_i = \frac{p_i}{\rho g_c} + z_i$$

where $\psi_i = \frac{p_i}{\rho g_c}$ is the pressure head in $[m]$, z_i the elevation head in $[m]$, p_i is the pressure in $[N/m^2]$, ρ is the density of water in $[kg/m^3]$ and g_c is the gravitational acceleration in $[m/s^2]$ [19, p. 4.2]. The hydraulic head can be thought of as the total virtual elevation and is an addition of the elevation head and pressure head, which can be seen as the virtual elevation due to the pressure at point i. A DMA is designed in a way that the hydraulic head is the

highest at the inlet and decreases further down the network due to frictional forces that are exerted by the inner walls of the pipelines on the flow. Pumps are means to increase the hydraulic head by increasing the pressure head.

2-1-4 Network components

Several hydraulic components to regulate flows and pressure heads throughout a network can be installed.

Valves Isolation valves are used to close of an existing pipeline connection. These are used as boundary valves to construct DMAs in a WDN, or when a part of the network needs to closed off from supply for maintenance and emergency purposes. Directional valves open when water flows in one direction and closes when the flow turns in the opposite direction. These valves are for instance used to prevent water to flow backwards through a pump, which might cause damage. Air release valves are used to release trapped air in the system. PRVs throttle high pressure inflow to an outflow with lower pressure. Within a DMA usually at least one PRV is used to transpose the upstream pressure head in the transmission main to a lower downstream pressure head, see Figure 2-1. The downstream pressure head ψ_d can be regulated between the upstream pressure head ψ_u and zero, which can be modelled as follows:

$$\psi_d = \alpha_{PRV}\psi_u, \qquad 0 \leqslant \alpha \leqslant 1$$

where the PRV coefficient α_{PRV} can be varied between 0(status PRV closed) and 1(status PRV open). Minor head losses due to the components structure need to be considered only if the PRV is fully opened [1].

Pumps Pumps are used to increase the pressure head in a network to overcome elevation differences and friction losses throughout a WDN. Mechanical energy is transferred to the hydraulic head to be able to supply the whole network. The head added by a pump can be described as

$$h_p = h_0 - c_p q_p^{m_p}$$

where h_0 is the shutoff head in $[m]$, q_p and c_p and m_p are coefficients to describe the shape of a pump curve which maps q_p to h_p.

Reservoirs and tanks Reservoirs and tanks are used to store water. The difference between them is that the elevation head of a reservoir is (kept) constant while the elevation head in a tank fluctuates due to inflow and outflow. Both are used to supply the water needed to meet the demands in a network. Figure 2-2 shows a sketch of a reservoir at an elevation z with respect to a horizontal datum, which is usually taken as the lowest point in a DMA.

The height of the water column in a reservoir adds h_{res} to the hydraulic head. For this reason, water is often stored or preferably taken from sources at relative high altitude, such that less pump work is needed within a WDN to maintain the hydraulic head at a sufficient level.

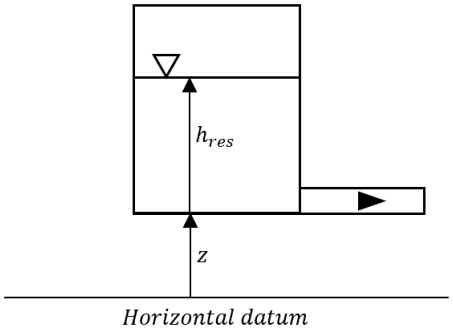


Figure 2-2: The water column in a reservoir adds h_{res} to the hydraulic head, $h = h_{res} + z$.

2-2 Flow models for a single pipeline

Static and dynamic flow models have been developed to describe the flow through a single pipeline in steady state and in the case transient events occur, respectively. In this work, we focus on the steady state models.

2-2-1 Steady state flow model

To observe how the hydraulic head changes with a fully developed flow through a single pipeline, the equation of conservation of energy for a single pipeline is elaborated in Appendix A based on [19] and [22]. As a result for a steady state flow q_j through a single pipeline j with a constant perimeter connecting points i and k, the following relation is obtained:

$$h_i - h_k \quad = \quad h_f - h_p \tag{2-1}$$

where h_i and h_k are the hydraulic heads at points i and k, respectively, h_f is the head loss due to friction of flow q_j through pipeline j and h_p is the increase in pressure head by a pump, being zero in the absence or shutdown of a pump between i and k. Minor head losses due to for instance the geometry of components like joints are assumed negligible [23].

2-2-2 Head loss models

The head loss h_f over a pipeline is generated by frictional shear forces between the flow and the inside wall. Different models have been proposed in literature to describe this relation, from which the Darcy-Weisbach (DW) and Hazen-Williams (HW) models are used the most for water flow modelling. Given a turbulent flow q_j in $[m^3/s]$ through a pipeline j connecting i and k, with diameter D_j in $[m]$ and length L_j in $[m]$, the head loss h_f is obtained by:

- *Darcy-Weisbach*:

$$h_f = f_j \frac{8L_j}{g\pi^2 D_j^5} q_j|q_j|, \qquad f_j = fcn\Big(Re, \frac{\varepsilon}{D_j}\Big)$$

  where f_j is the dimensionless DW friction factor, which is an implicit function of the dimensionless Reynolds number Re and the relative roughness, $\frac{\varepsilon_j}{D_j}$, with ε_j the internal

pipe roughness in $[m]$. By numerically solving the implicit Colebrook-White equation

$$\frac{1}{\sqrt{f_j}} = -0.86 \ln\left(\frac{\varepsilon}{3.7D_j} + \frac{2.51}{Re\sqrt{f_j}}\right) \tag{2-2}$$

or by using an empirical, explicit approximation of equation (2-2), like the Swamee-Jain formula, the value of f_j can be determined.

- *Hazen-Williams*:

$$h_f = \frac{10.67 L_j}{C_j^{1.852} D_j^{4.87}} q_j |q_j|^{0.852}$$

where C_j is the dimensionless HW pipe roughness coefficient, which is actually a smoothness coefficient as it has lower values for rougher pipelines.

The DW friction model is based on first principles as Newton's second law, is applicable to any Newtonian fluid in any flow regime and is mainly used in Europe. The HW friction model is based on experimental data, applicable to water in the turbulent flow regime and mainly used in the United States. [1, pp.30-38] Both models can be written in the form

$$h_f = r_j |q_j|^{n-1} q_j \tag{2-3}$$

where the resistance coefficient $r_j = \alpha L_j / (C_j^n D_j^m)$ and the triplet α, n and m varies with head loss model [9].

2-3 Modelling a DMA

2-3-1 Graphical representation

The structure of a DMA network can be represented by a graph $\mathcal{G}(V, E)$, where V is the set of n_n vertices or nodes at which multiple pipelines are connected and E is the set of n_p edges or pipelines. Water flows through the pipelines to supply water at the nodes. The state of the DMA consists of the heads at the nodes and the flows through the pipelines. An directed incidence matrix, A_{12}, can be constructed as follows:

$$A_{12}(j,i) = \begin{cases} 1, & \text{if pipeline } j \text{ enters node } i \\ 0, & \text{if pipeline } j \text{ and node } i \text{ are not connected} \\ -1, & \text{if pipeline } j \text{ leaves node } i \end{cases} \tag{2-4}$$

which stores the connectivity between nodes and the directions of positive flows through the pipelines. [11] Usually multiple residencies that are close to each other are aggregated and spatially allocated to a single junction, as shown in Figure 2-4.

Example 1. *Consider the small DMA network in Figure 2-3 with* $V = \{V_1, V_2, V_3\}$ *and* $E = \{E_1, \ldots, E_4\}$. *At the inlet a PRV is installed with which the pressure head at the inlet can be regulated. The state of the network is given by* $x = [q_1\ q_2\ q_3\ q_4\ h_1\ h_2\ h_3]^T$. *The black triangular arrows on the pipelines indicate the direction of positive flows. The incidence matrix for this small DMA is:*

$$A_{12} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 1 \end{bmatrix}$$

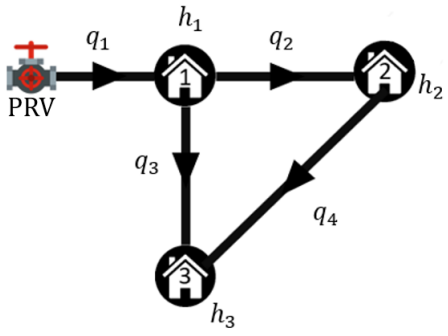


Figure 2-3: A small DMA with three nodes, $n_n = 3$, and four pipelines, $n_p = 4$. The pressure of the incoming flow at the inlet is regulated with a PRV.

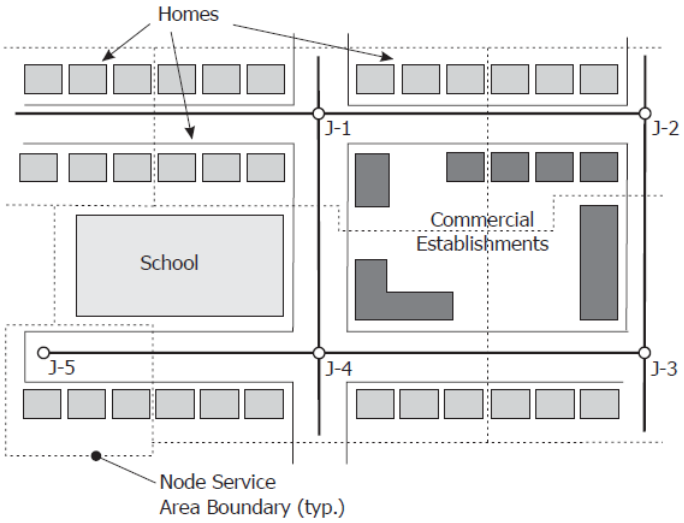


Figure 2-4: Usually multiple closely located residencies are aggregated and spatially allocated to a single junction. The dashed line depicts the boundary within the residencies are assigned to the corresponding junction. (Source: [1])

2-3-2 Nodal demands

The water demand or consumption throughout a network is the driving force behind the hydraulic dynamics in a DMA [1]. Each nodal demand d_i, $i = 1, \ldots, n_n$ equals the sum of demands of the residencies that are spatially allocated to the corresponding junction, see Figure 2-4. As the application of smart metering that gives online access to the consumption rates of customers are still in their infancy and expensive to implement, other ways are used to approximate the demands within a network. To determine the average demand of a single residency, billing records and historical flow data can be used. Based on this data a diurnal demand pattern which has a typical shape for each kind of customer can be assigned to it. Typical diurnal curves for businesses, a single family, a factory and a restaurant are depicted in Figure 2-5. The demand multiplier has an average of 1 and is multiplied with the average demand of a single customer that day. The uncertainty for a single customer is relatively large

compared to an aggregated group of the same customers, which is one of the key reasons to not model each single residency as a junction.

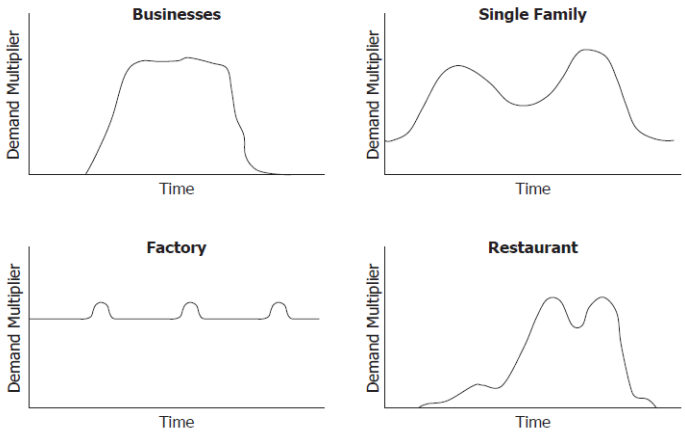


Figure 2-5: Some typical diurnal demand patterns for various user types. Characteristic for businesses is that demands are only nonzero during working times. For a single family two typical peaks at the morning and evening can be observed. A factory can have a repeating structure due to repetition of the same process. The restaurant's diurnal demand peaks are around lunch and diner time. (Source: [1])

2-3-3 Hydraulic equations

Steady state model Consider a network with n_n nodes with unknown hydraulic heads and n_p pipelines with unknown flows. It is assumed that by far the largest part of head losses through a DMA network are due to friction and that minor head losses due to the geometry of system components like joints and valves are negligible. In steady state, the conditions that relate flows with unknown heads are obtained by applying the head loss equation (2-1) to each pipeline, which yields the matrix equation [11]

$$A_{11}(q)q + A_{12}h + A_{10}h_0 = 0 \tag{2-5}$$

where $q = [q_1, \ldots, q_{n_p}]^T$ and $h = [h_1, \ldots, h_{n_n}]^T$ represent the unknown flows and pressure heads, respectively, $A_{12} \in \mathbb{R}^{n_p \times n_n}$ is the incidence matrix defined in equation (2-4), which relates the pipelines with unknown head nodes, $h_0 \in \mathbb{R}^{n_0 \times 1}$ represents the n_0 known heads of for example a reservoir, $A_{10} \in \mathbb{R}^{n_p \times n_0}$ is the incidence matrix relating the n_p pipelines with the n_0 nodes with known heads, similarly constructed as in (2-4) and $A_{11} \in \mathbb{R}^{n_p \times n_p}$ is a diagonal matrix with

$$A_{11}(j,j) = r_j|q_j|^{n-1}, \qquad j - 1, \ldots, n_p \tag{2-6}$$

which represents a part of the head loss model in equation (2-3). Furthermore in steady state, for each unknown head node a mass balance can be written, which relates demands and flows, yielding the following matrix equation

$$A_{12}^T q - d = 0 \tag{2-7}$$

where $d = [d_1, \ldots, d_{n_n}]^T$. Finally the nonlinear matrix equation g_h that describes the steady state conditions is given by merging equations (2-5) and (2-7)

$$g_h(q,h) \coloneqq \begin{bmatrix} A_{11}(q) & A_{12} \\ A_{12}^T & 0 \end{bmatrix} \begin{bmatrix} q \\ h \end{bmatrix} + \begin{bmatrix} A_{10}h_0 \\ -d \end{bmatrix} = 0 \tag{2-8}$$

Example 2. *The small DMA network in Figure 2-3 has the following steady state conditions*

$$\begin{aligned} h_0 - h_1 &= r_1|q_1|^{n-1}q_1 \\ h_1 - h_2 &= r_2|q_2|^{n-1}q_2 \\ h_1 - h_3 &= r_3|q_3|^{n-1}q_3 \\ h_2 - h_3 &= r_4|q_4|^{n-1}q_4 \\ q_1 - q_2 - q_3 &= d_1 \\ q_2 - q_4 &= d_2 \\ q_3 + q_4 &= d_3 \end{aligned}$$

can be rewritten in the form of equation (2-8) as

$$\left[\begin{array}{cccc|ccc} r_1|q_1|^{n-1} & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & r_2|q_2|^{n-1} & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & r_3|q_3|^{n-1} & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & r_4|q_4|^{n-1} & 0 & -1 & 1 \\ \hline 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{array}\right] \left[\begin{array}{c} q_1 \\ q_2 \\ q_3 \\ q_4 \\ \hline h_1 \\ h_2 \\ h_3 \end{array}\right] + \left[\begin{array}{c} -h_0 \\ 0 \\ 0 \\ 0 \\ \hline -d_1 \\ -d_2 \\ -d_3 \end{array}\right] = 0$$

Solving the nonlinear hydraulic equation The hydraulic equation $g_h(x)$ in (2-8) is nonlinear in $x \coloneqq [q\ h]^T$ and cannot be directly solved. Methods have been introduced to numerically solve g_h , i.e. finding its root x^* at which $g_h(x) = 0$, of which the Newton-Raphson algorithm is computationally most efficient [19] and introduced by Todini and Pilati [11], who also proved that a solution is unique and always exists. Using a Taylor series expansion of g_h to obtain a first order approximation of g_h at x^k it follows

$$g_h(x^k) + \frac{\partial g_h}{\partial x}\Big|_{x^k}(x^{k+1} - x^k) = 0$$

which can be rewritten as

$$x^{k+1} = x^k - \frac{g_h(x^k)}{\partial g_h/\partial x|_{x^k}} \tag{2-9}$$

Equation (2-9) can be iteratively applied until the error $\Delta x = x^{k+1} - x^k$ is small enough [19]. Writing out this equation for g_h yields

$$\begin{aligned} q^{k+1} &= (I - N^{-1})q^k - N^{-1}A_{11}^{-1}(A_{12}h^{k+1} + A_{10}h_0) \\ h^{k+1} &= -(A_{12}^T N^{-1} A_{11}^{-1} A_{12})^{-1}(A_1 2^T N^{-1}\Big(q^k + A_{11}^{-1}A_{10}h_0 + (d - A_{12}^T q^k)\Big) \end{aligned}$$

where N is a diagonal matrix with $N(j,j) = n-1$. This algorithm can be used when A_{11} is not (close to) singular, i.e. no close to zero flows are present, see equation (2-6). This algorithm is implemented in hydraulic solvers like EPANET. With the use of Python packages like WNTR [24] extensive hydraulic simulations can be performed.

Chapter 3

Methodology

3-1 Active Fault Diagnosis (AFD)

3-1-1 Problem Statement

Consider a District Metered Area (DMA) fed by n_u inlets that supplies consumers at n_n nodes with fresh water through a network of n_p pipelines. Over the nodes $n_s \ll n_n$ pressure gauges are distributed. At a single time snapshot, consider the steady-state model

$$M : \begin{cases} f(x,u,d) = 0 \\ y = h(x) \end{cases}, \tag{3-1}$$

which, once the inputs $u \in \mathbb{R}^{n_u}$ at the inlets and the demands $d \in \mathbb{R}^{n_n}$ at the nodes are known, uniquely determines the DMA's state $x \triangleq [q\ h]^T \in \mathbb{R}^{n_x}$ at steady state and outputs $y \in \mathbb{R}^{n_s}$, being the gauged pressures within state determined by $h(x) : \mathbb{R}^{n_x} \mapsto \mathbb{R}^{n_s}$, where $q \in \mathbb{R}^{n_p}$ is the vector with the flows through the n_p pipelines expressed in m^3/s and $h \in \mathbb{R}^{n_n}$ is a vector which contains the n_n hydraulic heads at the nodes, $h_i = \psi_i + z_i,\ i \in \mathcal{I}_n \triangleq \{1, \ldots, n_n\}$, that sum up the pressure head ψ_i and elevation head z_i at node i, expressed in mH_2O. The non-linear algebraic function $f(\cdot) : \mathbb{R}^{n_u} \times \mathbb{R}^{n_n} \mapsto \mathbb{R}^{n_x}$ implicitly describes the non-linear hydraulic relations between state variables based on first principles of conservation of mass and energy at steady state and includes network parameters that are assumed to be well calibrated and time-invariant. The inputs $u \in \mathcal{U} \subset \mathbb{R}_+^{n_u}$ corresponding with the pressure heads at the n_u inlets of the DMA are regularised with Pressure Reducing Valves (PRVs).

Assume the presence of a single leak at node i^*, which magnitude g in m^3/s is determined by [25]

$$g(x) = C_d A \sqrt{\frac{2}{\rho_w}} (\rho_w g_c \psi_{i^*})^{\alpha} = K p_{i^*}^{\alpha} \tag{3-2}$$

where $C_d > 0$ is the unitless discharge coefficient, A is the area of the hole in m^2, ρ_w denotes the density of water in kg/m^3, g_c denotes gravity in m/s^2, ψ_{i^*} is the pressure head in mH_2O,

p_{i^*} is the gauge pressure in N/m^2 at node i^*, the discharge constant $K > 0$ and $0 < \alpha < 1$ are parameters dependent on the leak size and pipeline material.

Now, the active leak isolation problem can be stated as:

Problem 1 (**Active leak isolation**). *Given a DMA with uncertainties in demands d and leak magnitude g but known inputs u and measured outputs $\tilde{y}$, the aim is to optimally design inputs u that improve the input-output (I/O) data, such that, by use of estimated outputs $\hat{y}$ of leakless model M in equation (3-1), injected with estimated demands $\hat{d}$ and known inputs u, classification of the observed residual, $\tilde{r} = \hat{y} - \tilde{y} \in \mathbb{R}^{n_s}$, to leak location $\hat{i}^*$, ultimately corresponding with i^*, is sped up and more reliable compared to passive methods.*

3-2 Stochastic AFD

3-2-1 Residual Space

Since the uncertainties in estimated demands $\hat{d} \sim \mathbb{P}_d$ propagate into the estimated outputs $\hat{y}$, the residual vector $\tilde{r}$ is a random vector with unknown probability distribution function $\mathbb{P}_{r|i^*}$. It is assumed that a leak detection algorithm has detected the presence of a leak at a single time snapshot and that an estimate $\hat{g} \sim \mathbb{P}_g$ of the real unknown leak magnitude g is provided. It is further assumed that hypothetical leak locations are restricted to the nodes, such that the hypothesis set can be denoted by $\mathbb{H} \triangleq \{H^{[i]}\}_{i \in \mathcal{I}_n}$.

The residual space $\mathcal{R}$, which reflects the uncertainty propagation into all possible residuals i.e. $\mathbb{P}_d \times \mathbb{P}_g \mapsto \mathbb{P}_r(i, u)$, is described in Definition 1:

Definition 1 (**Residual space**). *Given $\mathbb{H}$ and the demand and leak magnitude PDFs $\mathbb{P}_d$ and $\mathbb{P}_g$, the residual space $\mathcal{R} \subset \mathbb{R}_+^{n_s}$ is the set of all possible residual realisations of the form*

$$r^{[i]} = \hat{y} - y^{[i]} \sim \mathbb{P}_r(i, u),\ i \in \mathcal{I}_n \tag{3-3}$$

Residual realisations $r^{[i]},\ i \in \mathcal{I}_n$ reflect the effect of a leak with magnitude $\hat{g} \sim \mathbb{P}_g$ at node i on the pressure head at gauged nodes. The outputs $y^{[i]}$ are calculated by solving the n_n algebraic equations of corresponding leaky models

$$M_i : \begin{cases} f^{[i]}(x^{[i]}, u, \theta^{[i]}) = 0 \\ y^{[i]} = h(x^{[i]}) \end{cases} \qquad i \in \mathcal{I}_n \tag{3-4}$$

where $\hat{g}$ is modelled as an additive demand at node i, such that $\theta^{[i]} = [\hat{d}_1, \ldots, \hat{d}_i + \hat{g}, \ldots, \hat{d}_{n_n}]^T$ and $d \sim \mathbb{P}_d$.

The parametric uncertainties propagate through non-linear algebraic equations $f^{[i]}(\cdot)$ into state $x^{[i]}$ and outputs $y^{[i]}$ such that residual vector $r^{[i]} \sim \mathbb{P}_r(i, u)$ is a random vector.

The hardness to classify an observed residual $\tilde{r}$ at a single time snapshot to the right hypothesis $H^{[i^*]}$, arises from the overlap of marginal residual PDFs $\mathbb{P}_{r|i}(u),\ i \in \mathcal{I}_n$, such that leak isolation cannot be achieved under all possible observed residuals in $\mathcal{R}$. A proper way to deal with these overlapping marginals is to introduce Stochastic AFD, which aim is to design inputs u that minimise the overlap between marginal residual PDFs $\mathbb{P}_{r|i}(u),\ i \in \mathcal{I}_n$, and hence the

risk of misdiagnosis [13]. It directly follows that the stochastic AFD is equivalent to solving Problem 1.

Since demand and leak magnitude PDFs are usually not available, approximations $\hat{\mathbb{P}}_d$ and $\hat{\mathbb{P}}_g(u)$ are needed to capture the uncertainty propagation through $f^{[i]}(\cdot)$ and to approximate $\mathbb{P}_r(i,u)$. A marginal Probability Distribution Function (PDF) of the form $\hat{\mathbb{P}}_{r|i}(u)$ can be estimated by construction of a discrete PDF through extrapolation of information contained in N random realisations of $r^{[i]}$.

3-2-2 Proposed algorithm

Algorithm 1 activates the Bayesian classification method for leak isolation proposed in [14] by extending it with the stochastic AFD method described in [26] and aims to improve leak isolation over consecutive time steps. In this section a single time step is elaborated and time index k is dropped.

The algorithm consists of a two-step procedure: 1) update the control inputs u and 2) update the current likelihood vector

$$\mathcal{Q} = \begin{bmatrix} \Pr(H^{[i]}) & \dots & \Pr(H^{[n_n]}) \end{bmatrix} \in \mathbb{R}^{n_n},$$

which reflects the belief of individual hypotheses to match with reality. It further should hold that $\sum_{\mathcal{I}_n} \mathcal{Q}(i) = 1$.

1) The active part in lines 2-7 shows how the inputs u are updated in order to minimise the overlap between marginal residual PDFs. Mathematically, this objective is captured as:

$$\max_u J(u) \quad \text{s.t. } u \in \mathcal{U}$$

which maximises the weighted objective function

$$J(u) = \frac{1}{2} \sum_{i,j} \mathcal{Q}(i) \mathbb{D}(i,j,u) \mathcal{Q}(j), \ i,j \in \mathcal{I}_n \tag{3-5}$$

with respect to $u \in \mathcal{U} \subset \mathbb{R}_+^{n_s}$, where

$$\mathbb{D}(i,j,u) = \mathbb{D}(\mathbb{P}_{r|i}(u), \mathbb{P}_{r|j}(u))$$

is a stochastic metric that quantifies the overlap between marginal residual PDFs of hypothesis $H^{[i]}$ and $H^{[j]}$. However, as we have no access to the real residual PDFs, and hence, have no analytical description of $J(u)$, estimates $\hat{\mathbb{P}}_{r|i}$ need to be constructed $\forall i | \mathcal{Q}(i) \neq 0$ and used to approximate $\mathbb{D}(i,j,u)$ and $J(u)$. Now $J(u)$ is stepwise maximised using the gradient method which takes u a step η in direction of gradient

$$\nabla_u J(u) \triangleq \begin{bmatrix} \frac{\partial J(u)}{\partial u_1} & \dots & \frac{\partial J(u)}{\partial u_{n_u}} \end{bmatrix}^T.$$

The elements of u and step vector $\eta \nabla J(u)$ are constrained by upper bounds u_{max} and $\Delta_{u,max}$, respectively.

In line 8 observed residual $\tilde{r}$ is constructed, see Problem 1.

2): In lines 9-11 the belief $\mathcal{Q}$ is updated using Bayes' rule:

$$\mathcal{Q}(i) \leftarrow \Pr(H^{[i]}|\tilde{r}) = \underbrace{\frac{\Pr(\tilde{r}|H^{[i]})}{\Pr(\tilde{r})}\mathcal{Q}(i)}_{\zeta}, \quad \forall i \in \mathcal{I}_n.$$

where $\Pr(H^{[i]}|\tilde{r})$ denotes the posterior likelihood that, given observation $\tilde{r}$, the leak is located at node i, $\Pr(\tilde{r}|H^{[i]})$ denotes the probability that $\tilde{r}$ corresponds with hypothesis $H^{[i]}$, which can be estimated using $\hat{\mathbb{P}}_{r|i}(u)$, $\Pr(\tilde{r}) = \sum_{\mathcal{I}_n} \Pr(\tilde{r}|H^{[i]})\Pr(H^{[i]})$ is a normalising term to ensure that $\sum_{\mathcal{I}_n} \mathcal{Q}(i) = 1$ and ζ is a dummy variable. From line 5 it follows that approximations of $\mathbb{P}_{r|i}(u)$ need to be constructed when $\mathcal{Q}(i) \neq 0$. Therefore, for computational purposes, $\mathcal{Q}(i)$ is set to zero whenever it drops below threshold $\epsilon \approx 0$. The likelihood vector needs then of course be renormalised.

Algorithm 1 Update u and $\mathcal{Q}$ over subsequent time snapshots

Require: $\mathcal{I}_n,\ \lambda,\ u_0,\ \mathcal{Q}_0,\ \epsilon,\ \mathbb{D},\ \eta,\ \mathbb{H}$
Ensure: $i, j \in \mathcal{I}_n, k = 1$

1: **while** $\max_i \mathcal{Q}_{k-1}(i) < \lambda$ **and** $k \leqslant k_{max}$ **do**
2: **if** $k = 1$ **then**
3: $u_k = u_0$
4: **else**
5: $\nabla_u J(u_{k-1}) = \frac{\sum_{i,j} \mathcal{Q}_{k-1}(i)\mathcal{Q}_{k-1}(j)\nabla_u \mathbb{D}(i,j,u_{k-1})}{2}$
6: $u_k = u_{k-1} + \eta \nabla_u J(u_{k-1})$ s.t. constraints
7: **end if**
8: Construct $\tilde{r}_k$
9: **for** $i \in \mathcal{I}_n$ **do**
10: $\mathcal{Q}_k(i) = \begin{cases} \underbrace{\frac{\Pr(\tilde{r}_k|H^{[i]})}{\Pr(\tilde{r}_k)}\mathcal{Q}_{k-1}(i)}_{\zeta}, & \zeta > \epsilon \\ 0, & \zeta \leqslant \epsilon \end{cases}$
11: **end for**
12: $k \leftarrow k + 1$
13: **end while**

Classification: The best guess at each time step is the hypothesis with the maximum likelihood, i.e.:

$$H^{[\hat{i}^*]} \leftarrow \hat{i}^* = \arg\max_i \mathcal{Q}(i)$$

which at best corresponds with i^*. The classification result can be improved by iteratively applying Algorithm 1 over subsequent single time snapshots. Line 1 shows when the algorithm terminates: when the maximum likelihood crosses threshold λ or when k crosses k_{max}. If no additional information is available, the prior probability of each leak location is equally probable, such that initially $\mathcal{Q}_0(i) = \frac{1}{n_n},\ i \in \mathcal{I}_n$.

Ultimately, over subsequent time snapshots, the estimated leak location $\hat{i}^*$ converges to i^* or equivalently $\mathcal{Q}_k$ to δ_{i^*}. At the same time, the stochastic distances between marginal residual

PDFs are, proportional weighed with their corresponding likelihoods, maximised by updating inputs u between subsequent time snapshots in direction $\nabla_u J(u)$. In this way faster and more reliable leak localisation can be achieved with respect to the passive Bayesian classification method.

3-3 Elaboration and specifications for a DMA

3-3-1 The non-linear hydraulic algebraic equation

For a DMA at steady state with n_u inlets, n_n nodes and n_p pipelines, the non-linear algebraic equation $f^{[i]}(\cdot)$ in (3-4) becomes matrix equation [11]

$$f^{[i]}(\cdot) = \begin{bmatrix} A_{11}(q^{[i]}) & A_{12} \\ A_{12}^T & 0 \end{bmatrix} \begin{bmatrix} q^{[i]} \\ h^{[i]} \end{bmatrix} + \begin{bmatrix} A_{10}u \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -\theta^{[i]} \end{bmatrix} = 0 \tag{3-6}$$

where the diagonal matrix $A_{11} \in \mathbb{R}^{n_p \times n_p}$ consists of elements

$$A_{11}(j,j) = R_j |q_j^{[i]}|^{\gamma-1}, \quad j = 1, \ldots, n_p,$$

with R_j the resistance coefficient of pipeline j. $y^{[i]} = Cx^{[i]}$ with $C \in \mathbb{R}^{n_x \times n_s}$. We use the Hazen-Williams (HW) energy loss model to describe the friction of pipelines to flow, where $\gamma = 1.852$, $R_j = 10.670 L_j / (C_j^{1.852} D_j^{4.871})$ and L_j, C_j, D_j are the length in m, unitless roughness coefficient and diameter in m of pipeline j, respectively. The matrices $A_{12} \in \mathbb{R}^{n_p \times n_n}$ and $A_{10} \in \mathbb{R}^{n_p \times n_u}$ are incidence matrices that denote the connectivity for the n_n unknown head nodes and the n_u nodes with regulated pressure heads $u \in \mathcal{U}$, respectively. It is assumed that the demand estimation procedure yields realisations $\hat{d}_j$, $j \in \mathcal{I}_n$ from a Gaussian PDF $\hat{\mathbb{P}}_d$. Likewise, $\hat{g}$ is assumed to be a realisation of uniform PDF $\hat{\mathbb{P}}_g$ on $[g^-, g^+]$. It is further assumed that the parameters describing network characteristics like D_j and C_j are calibrated using historical data and time-invariant.

3-3-2 Estimation residual PDFs

To estimate the marginal residual PDFs, Monte Carlo simulations are performed to capture the propagation of parametric uncertainty in d and g into r. Estimates $\hat{\mathbb{P}}_{r|i}(u)$, $\forall i | \mathcal{Q}(i) \neq 0$, are constructed by computing $r^{[i]}$, $\forall i | \mathcal{Q}(i) \neq 0$ for N random realisations of $\hat{\mathbb{P}}_d$ and $\hat{\mathbb{P}}_g$ and assigning a probability $1/N$ to each computed residual using equation (3-3), which yields following data sets

$$\mathcal{D}^{[i]} \triangleq \{\hat{r}_j^{[i]}\}_{j=1,\ldots,N}, \quad \forall i | \mathcal{Q}(i) \neq 0.$$

Now, as the amount of residuals to be realised scales with at maximum $N \times n_n$, it becomes computationally very demanding to take a large N. Therefore, we use Kernel Density Estimation (KDE) to generalize less samples to smooth PDFs and randomly resample these with $N_{\mathcal{K}} \gg N$ data points to obtain data sets

$$\mathcal{D}_{\mathcal{K}}^{[i]} \triangleq \{\hat{r}_j^{[i]}\}_{j=1,\ldots,N_{\mathcal{K}}}, \quad \forall i | \mathcal{Q}(i) \neq 0.$$

Then, discrete PDFs $\hat{\mathbb{P}}_{r|i}(u),\ \forall i|\mathcal{Q}(i) \neq 0$ can be constructed by normalising histogram representations of the resampled data, where the residual space $\mathcal{R}$ is divided in m equally dimensioned bins as in [26], such that

$$\hat{\mathbb{P}}_{r|i}(u) \triangleq \{p_{i,n}(u)\}_{n=1,\dots,m}, \quad \forall i|\mathcal{Q}(i) \neq 0. \tag{3-7}$$

The probability assigned to a partition equals the number of realised residuals for a leak at i that fall within it times $1/N_{\mathcal{K}}$.

3-3-3 Similarity measure

The affinity between two PDFs aka the Bhattacharyya coefficient is closely related to the Bayes error and can therefore be used as a measure of overlap between two PDFs. It is defined for $\mathbb{P}_{r|i}$ and $\mathbb{P}_{r|j}$ as

$$\rho^{[i,j]} \triangleq \rho(\mathbb{P}_{r|i}, \mathbb{P}_{r|j}) = \int \sqrt{d\mathbb{P}_{r|i} d\mathbb{P}_{r|j}}$$

and is bounded by $0 \leqslant \rho^{[i,j]} \leqslant 1$, where 0 corresponds to no overlap and 1 to identical PDFs. Now the Hellinger distance

$$H^{[i,j]} = \sqrt{1 - \rho(\mathbb{P}_{r|i}, \mathbb{P}_{r|j})}$$

can be constructed, which is a symmetric, positive-definite function that meets the triangle inequality condition to qualify as a distance metric [27] and is bounded by $0 \leqslant H^{[i,j]} \leqslant 1$, where 1 corresponds to fully separated PDFs. The metric $\mathbb{D}(i,j,u)$ is specified to be the Hellinger distance, because it is applicable to PDFs of any form. The sample estimate of $H^{[i,j]}$ between PDFs of the form in equation (3-7) is given by:

$$\hat{H}^{[i,j]}(u) = \sqrt{1 - \sum_{n=1}^{m} \left(p_{i,n}(u) p_{j,n}(u)\right)^{\frac{1}{2}}}.$$

3-4 Input Design

Substitution of the Hellinger distance for $\mathbb{D}(i,j,u)$ in equation (3-5) yields:

$$J(u) = \frac{1}{2} \sum_{i,j} \mathcal{Q}(i)\mathcal{Q}(j) H^{[i,j]}\Big(\mathbb{P}_{r|i}(u), \mathbb{P}_{r|j}(u)\Big), \quad i,j \in \mathcal{I}_n.$$

Now the goal is to maximise the objective function with respect to the input to obtain $u^* = \arg\max_u J(u)$. However, since we have no access to continuous residual PDFs, u^* cannot be straightforwardly obtained. Therefore the gradient method is applied, which gradually maximises the objective function with respect to u by taking steps in direction of the gradient $\nabla_u J(u)$, which is the direction in which the objective function increases the fastest. The inputs are updated over subsequent time snapshots according to

$$u_{k+1} = u_k + \eta \nabla_u J(u_k) \tag{3-8}$$

where $\eta \in \mathbb{R}$ is a constant design parameter referred to as the step size or learning rate. As we have no access to the real, continuous residual PDFs, the gradient

$$\nabla_u J(u) = \frac{1}{2} \sum_{i,j} \mathcal{Q}(i)\mathcal{Q}(j)\nabla_u H^{[i,j]}(u) \in \mathbb{R}^{n_u}. \tag{3-9}$$

is approximated by applying the following steps:

1. Construct n_u shifted data sets

$$\bar{\mathcal{D}}_u^{[i]} = \left\{ \hat{r}_j^{[i]} + \eta_r \nabla_u \hat{r}_j^{[i]} \right\}_{j=1,\dots,N}, \forall i | \mathcal{Q}(i) \neq 0 \tag{3-10}$$

   by taking every residual sample in data sets $\mathcal{D}^{[i]},\ \forall i|\mathcal{Q}(i) \neq 0$ and shifting it by design parameter $\eta_r \in \mathbb{R}$ in directions within $\nabla_u \hat{r}_j^{[i]} = \left[\frac{\partial \hat{r}_j^{[i]}}{\partial u_1}, \dots, \frac{\partial \hat{r}_j^{[i]}}{\partial u_{n_u}} \right]^T$;

2. Estimate the shifted residual PDFs by consecutively applying KDE, resampling and fitting normalised histograms to the shifted data sets;

3. Approximate the n_u elements of $\nabla_u H^{[i,j]}$ in (3-9) with

$$\frac{\partial H^{[i,j]}(u)}{\partial \nu} \approx \bar{\hat{H}}_{u_\nu}^{[i,j]}(u) - \hat{H}^{[i,j]}(u),\ \nu = 1, \dots, n_u.$$

In this way, the gradient is basically approximated by taking the difference between the objective of the residual PDFs corresponding with data sets $\mathcal{D}^{[i]},\ \forall i|\mathcal{Q}(i) \neq 0$ and the objectives obtained by slightly shifting these residual PDFs in n_u possible directions.

As it is also not straightforward to determine the residual gradient in equation (3-10), Lemma 1 shows how to deal with this. The *Kronecker product* [28] of matrices $P \in \mathbb{R}^{1,n_u}$ and $Q \in \mathbb{R}^{n_s,1}$ is defined as:

$$P \otimes Q \triangleq [p_1 Q \ \dots \ p_{n_u} Q] \in \mathbb{R}^{n_u \times n_s}.$$

Lemma 1 (**Approximate** $\nabla_u \hat{r}_j^{[i]}$)**.** *Given residual realisation $\hat{r}_j^{[i]}$ with corresponding leak magnitude $\hat{g}_j$ at node i, the residual shifts with respect to inputs u in equation (3-10) can be equivalently stated as:*

$$\eta_r \nabla_u \hat{r}_j^{[i]} = \left(\eta_r \frac{\partial \hat{r}_j^{[i]}}{\partial g} \frac{\partial g}{\partial h_i} \right) \otimes \nabla_u h_i \tag{3-11a}$$

$$= \left(\bar{\eta}_r C S_{j,g}^{[i]} \right) \otimes S_{j,u}^{[i]}[n_p + i] \in \mathbb{R}^{n_u \times n_s} \tag{3-11b}$$

$$= \bar{\eta}_r \begin{bmatrix} \frac{\partial \hat{r}_{j,1}^{[i]}}{\partial g} \frac{\partial h_i}{\partial u_1} & \dots & \frac{\partial \hat{r}_{j,n_s}^{[i]}}{\partial g} \frac{\partial h_i}{\partial u_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial \hat{r}_{j,1}^{[i]}}{\partial g} \frac{\partial h_i}{\partial u_{n_u}} & \dots & \frac{\partial \hat{r}_{j,n_s}^{[i]}}{\partial g} \frac{\partial h_i}{\partial u_{n_u}} \end{bmatrix} \tag{3-11c}$$

where

$$\bar{\eta}_r = -\eta_r \frac{\partial g}{\partial h_i} \in \mathbb{R}$$

is a design parameter and $S_{j,g}^{[i]} \in \mathbb{R}^{n_x}$ *and* $S_{j,u}^{[i]} \in \mathbb{R}^{n_x \times n_u}$*, are the sensitivity of the state with respect to leak magnitude* g *and to the inputs* u*, respectively:*

$$S_{j,g}^{[i]} \triangleq \left[\frac{\partial q}{\partial g} \quad \frac{\partial h}{\partial g}\right]_j^{[i],T}, \quad S_{j,u}^{[i]} \triangleq \left[\nabla_u q \quad \nabla_u h\right]_j^{[i],T}. \tag{3-12}$$

Proof. Application of the chain rule in equation (3-11a) gives three derivative terms, which are treated below:

$\frac{\partial \hat{r}_j^{[i]}}{\partial g}$*:* The first term can be elaborated as:

$$\frac{\partial \hat{r}_j^{[i]}}{\partial g} = \frac{\partial \hat{y}_j^{[0]}}{\partial g} - \frac{\partial \hat{y}_j^{[i]}}{\partial g} = -CS_{j,g}^{[i]} \in \mathbb{R}^{n_s}$$

where the elements of the sensitivity of the state with respect to the leak magnitude denote how the state variables at steady state, being the solution of $f_j^{[i]}(\cdot)$ for the jth realisation in $\mathcal{D}^{[i]}$, change with increasing leak magnitude.

$\frac{\partial g}{\partial h_i}$: Given that a single leak is present, the leak magnitude is a function of the hydraulic head at node i, i.e. $\hat{g}_j(h_i)$, which derivative with respect to h_i is an unknown scalar. As we have assumed that the leak magnitude is determined by an underlying leak model of the form in (3-2), with $K, \alpha > 0$, it follows that

$$\frac{\partial g}{\partial h_i} = \rho_w g_c \alpha K \Big(\rho_w g_c (h_i - z_i)\Big)^{\alpha - 1} > 0$$

which implies that by increasing the hydraulic head at the leaky node i, the leak magnitude will increase as well, and vice versa. Since it is a scalar term it can be incorporated in design parameter $\bar{\eta}_r = \eta_r \frac{\partial g}{\partial h_i} \in \mathbb{R}$.

$\nabla_u h_i \in \mathbb{R}^{n_u}$: This derivative term describes how, at the steady-state equilibrium, the hydraulic head at the ith node changes with the different inputs in u and corresponds with the $(n_p + i)$th row in $S_{j,u}^{[i]}$. Note that, due to the Kronecker product, this term does not affect the direction of $\nabla_u \hat{r}_j^{[i]}$ but only its magnitude, which corresponds with the intuition that changing inputs at different locations, while keeping the other inputs constant, have different impacts on h_i. The ratio between these magnitudes varies logically also between possible leak locations. □

3-4-1 Sensitivities

It is often claimed that the sensitivity matrix of the state with respect to leak magnitude g "is extremely difficult to calculate analytically" [29] because the non-linear hydraulic equations in (3-6) are implicit. As a result sensitivities are estimated using extensive simulations. In this thesis the sensitivities are computed analytically by using the implicit function theorem, which reduces computational complexity and makes the proposed approach tractable. In the following theorem and proof we show when and how the sensitivities required in equation (3-11b) can be computed analytically.

Theorem 1 (**Calculate $S_g^{[i]}$, $S_u^{[i]}$ analytically**). *Let x^* be a non-degenerate solution of (3-6), i.e. the Jacobian $\frac{\partial f^{[i]}}{\partial x}(x^*)$ is not singular. Then the sensitivities $S_g^{[i]}$ and $S_u^{[i]}$ as defined in (3-12) can be calculated analytically.*

Proof. Since the function $f^{[i]}(\cdot)$ is continuously differentiable around such a solution x^* [30, Appx. 1], by the implicit function theorem [31, Thm. A.2], we have:

$$\begin{aligned} \frac{\partial f^{[i]}(\cdot)}{\partial x}\frac{\partial x}{\partial g} &= -\frac{\partial f^{[i]}(\cdot)}{\partial g} \\ \underbrace{\begin{bmatrix} N_A A_{11}(\hat{q}^{[i]}) & A_{12} \\ A_{12}^T & 0 \end{bmatrix}}_{A^{[i]}} \underbrace{\begin{bmatrix} \frac{\partial q}{\partial g} \\ \frac{\partial h}{\partial g} \end{bmatrix}^{[i]}}_{S_g^{[i]}} &= \underbrace{\begin{bmatrix} 0 \\ I^{[i]} \end{bmatrix}}_{B_g^{[i]}} \end{aligned}$$

where $N_A = diag(\gamma_i),\ i = 1, \ldots, n_p$ and $I^{[i]} \in \mathbb{R}^{n_n}$ is

$$I^{[i]}(j) = \begin{cases} 1, & j = i \\ 0, & j \neq i. \end{cases}$$

Therefore, the sensitivity of the state with respect to the leak magnitude $S_g^{[i]}$ can be computed by solving the $n_n + n_p$ linear equations, i.e. $S_g^{[i]} = A^{[i]} \backslash B_g^{[i]}$.
Likewise, at the same steady state solution x^* we can write:

$$\begin{aligned} \frac{\partial f^{[i]}(\cdot)}{\partial x}\nabla_u x &= -\nabla_u f^{[i]}(\cdot) \\ A^{[i]} S_u^{[i]} &= B_u^{[i]}, \end{aligned}$$

where $B_u^{[i]} = \begin{bmatrix} A_{10} \\ 0 \end{bmatrix}$ and we obtain $S_u^{[i]} = A^{[i]} \backslash B_u^{[i]}$. □

Remark 1. *The sensitivity matrix $S \in \mathbb{R}^{n_s \times n_n}$ in [29] collects the effect of all n_n possible leaks on the head of the n_s gauged nodes. As we compute only the sensitivity vector $S_g^{[i]}$ that evaluates the effect of a leak at node i on the gauged nodes, S can be calculated by replacing $I^{[i]}$ with identity I_{n_n} and multiplying resulting $S_g = \begin{bmatrix} \nabla_g q & \nabla_g h \end{bmatrix}^T \in \mathbb{R}^{n_x \times n_x}$ with C, i.e. $S = CS_g$, to select the columns in $\nabla_g h$ corresponding with the n_s gauged nodes.*

Remark 2. *The number of linear solves for each k depends on $\mathcal{Q}_k$ and is at maximum $n_n \times n_x \times (N + 1 + n_u)$; the complexity scales quadratically with the number of unrejected hypotheses and linearly with N and n_u.*

Chapter 4

Case Study

The benchmark of the trunk network in Hanoi [32] is used to assess the performance of Algorithm 1 against the passive Bayesian classification method introduced in [14].

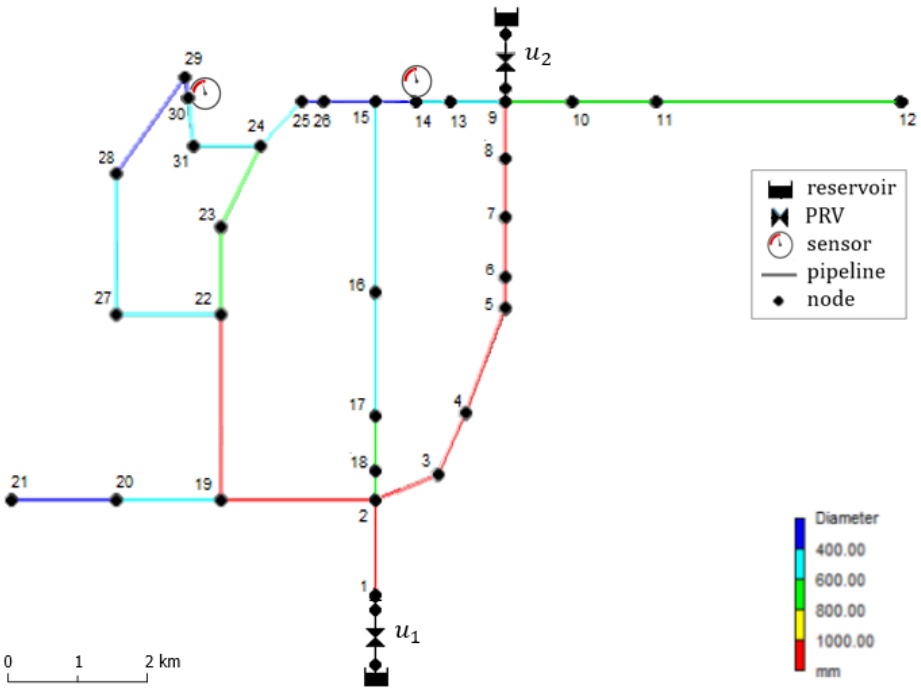


Figure 4-1: The Hanoi network consists of $n_n = 31$ nodes and $n_p = 33$ pipelines. Pressure gauges are installed at nodes 14 and 30, i.e. $n_s = 2$. The downstream pressure head at node 1 and 9 is regulated with Pressure Reducing Valves (PRVs) u_1 and u_2, respectively. The diameter of each pipeline is indicated with colors and shows nicely how the network branches off.

4-1 The Hanoi network

Figure 4-1 shows a graphical representation of the Hanoi network, which consists of 31 nodes connected by 33 pipelines and is fed by two reservoirs. At nodes 14 and 30 a pressure gauge is installed. The pressure heads of node 1 and 9 are regularised with PRVs, referred to as u_1 and u_2, respectively. The District Metered Area (DMA) contains two loose ends: 9-10-11-12 and 19-20-21. Leaks at these nodes of equal magnitude affect the pressure distribution across the looped part of the network in the same way. Therefore leaks at these locations are unisolable and are aggregated at nodes 9 and 19 to prevent the algorithm to getting stuck while attempting diagnose undiagnosable faults.

The elevation is constant over the network, such that the pressure head distribution at steady state is fully determined by the regulated boundaries using the PRVs and the head losses due to flows that satisfy nodal and leakage demands.

Remark 3. *The second PRV and reservoir are added in order to show how the algorithm performs for multiple inputs.*

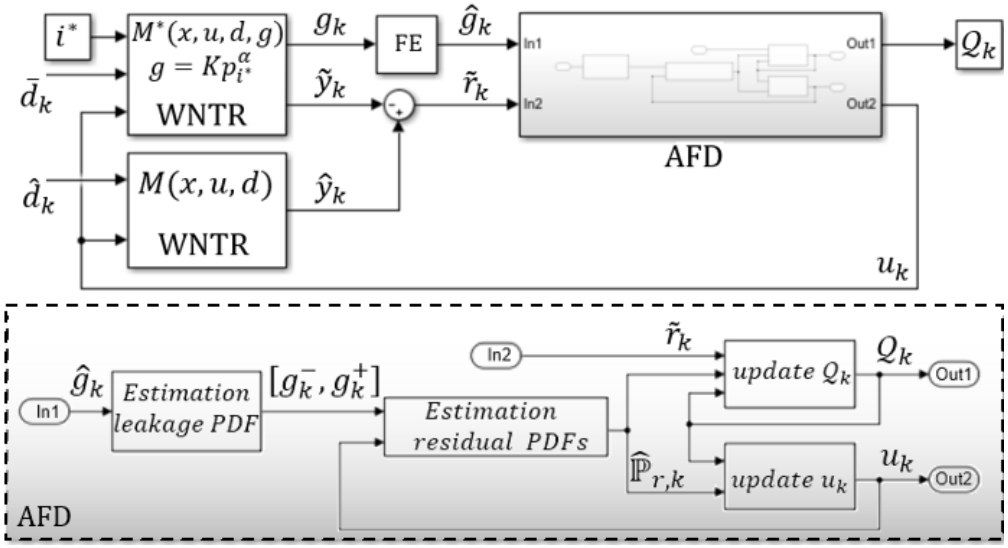


Figure 4-2: Schematic representation of the simulation setup to assess the performance of Algorithm 1 for a leak at node i^*, without using the actual Hanoi network, i.e. the real network is substituted with a model including the pressure dependent leak model in equation 3-2. Demands $\hat{d}_k$ and d of respectively the 'real' (M^*) and leakless model (M) are obtained by equation 4-1. Using th WNTR solver, the observed residual is computed and it is assumed that a fault estimation algorithm provides an estimate of the real leak magnitude g_k. The Active Fault Diagnosis (AFD) scheme shows that first the interval of the uniform leak magnitude Probability Distribution Function (PDF) $\mathbb{P}_g$ based on $\hat{g}_k$ is constructed, which is used to construct the marginal residual PDFs. Then the likelihood vector $\mathcal{Q}_k$ and input vector u_k are updated according to algorithm 1. The input vector is fed back to the models and the likelihood vector can be used at each time step to classify the sequence of observed residuals to the most likely leak location.

4-2 Simulation setup

To mimic the real network with a leak at i^*, it is simulated by merging the leak model in (3-2) with the leakless model in (3-1). Figure 4-2 provides a schematic of how the 'real network' interacts with the algorithm. Table 4-1 lists required constants. The following specifications are involved:

- *Demands*: The demands $\bar{d}_k$ and $\hat{d}_k$ of respectively the 'real' (M^*) and leakless model (M are obtained by [33]:

$$d_{k,j} = \max(Y_d, 0)\mu_k b_j, \ \forall j \in \mathcal{I}_n \tag{4-1}$$

  where b_j is the base demand of the jth node, μ_k is the demand multiplier at time step k and $Y_d \sim \mathcal{N}_{Y_d}(1, \sigma_d)$ is a random draw from normal distribution $\mathcal{N}_{Y_d}$. Figure 4-3 shows the used diurnal pattern for the demand multiplier and the $\mu_k \pm 3\sigma_d$ (99.7%) interval for varying σ_d.

- $\tilde{r}_k$: The residual at each time step is generated by solving the non-linear hydraulic equations of M^* and M and obtaining their respective outputs $\tilde{y}_k$ and $\hat{y}_k$.

- *Fault Estimation (FE)*: The FE block mimics the estimation of the leak magnitude and predicts its value according to $\hat{g}_k = g_k + Y_g$ where $Y_g \sim \mathcal{N}(0, \sigma_g)$.

- *AFD*: The AFD part takes as input the estimated leak magnitude $\hat{g}_k$ and the residual $\tilde{r}_k$ and outputs the updated input u_k and likelihood vector $\mathcal{Q}_k$.

- *Estimation leakage PDF*: It is assumed that the estimated leak magnitude follows a uniform distribution on $[g^-, g^+]$ with interval boundaries $\hat{g}_k \pm 3\sigma_g$.

- *Estimation residual PDFs*: Only the residual PDFs for leak locations with nonzero likelihood are constructed. Required are N, $N_{\mathcal{K}}$, and $\bar{\eta}_r$.

- *Update* $\mathcal{Q}_k$: Using the estimated residual PDFs, the likelihood vector is updated, requiring ϵ.

- *Update* u_k: The estimated residual PDFs and $\mathcal{Q}_k$ are used to update u_k. The inputs are upper bounded by u_{max} and lower bounded by regulation of a minimal pressure head of 15 m at the critical point (CP), node 30, being a node in the lowest pressure region [9]. The input step at a single time step is bounded by $\Delta_{u,max}$. It follows that g has bounds $[0.0193, 0.0498]$ m^3/s between 0.96% and 2.47% of the mean total base demand at nighttime.

Remark 4. *All simulations are performed by using the* WNTR *Python package [24] and its built-in hydraulic solver, see also section 2-3-3. See Appendix C for the written software.*

Table 4-1: Simulation constants

C_d	N	$\bar{\eta}_r$	ϵ	$u_{max}[mH_2O]$
0.75	50	-0.1	$5/10^4$	100
α	$N_{\mathcal{K}}$	η	$\sigma_g[m^3/s]$	$\Delta_{u,max}[mH_2O]$
0.5	10^5	50	0.003	5

4-3 Analysed scenarios

The nominal scenario includes the Hanoi network depicted in Figure 4-1 with 2 pressure gauges, 2 inputs and a leak with area $A = 0.0015\ m^2$ at i^*. The following experiments are performed during nighttime between 0AM and 5AM with a time step of 5 minutes:

i The AFD algorithm is five times tested on the nominal scenario for each of the considered leak locations, i.e. all nodes except 10-12, 20, 21, and compared against the Passive Fault Diagnosis (PFD) algorithm.

ii i is executed for different levels of uncertainty in the nodal demands, where σ_d is varied between 0.01 and 0.09, see Figure 4-3;

iii Steps i and ii are repeated for the case that a third sensor at node 5 is added;

Performance is measured using two metrics: accuracy and average distance, where accuracy is the percentage of leaks that are classified to the right leak location and the average distance indicates the mean distance between the classified and right leak location calculated using Dijkstra's algorithm [34].

Remark 5. *The PFD algorithm differs from the AFD algorithm by that it only regulates the pressure at CP to be 15 mH_2O and skips the active part of updating u_k.*

Remark 6. *The algorithms terminate whenever one unrejected hypothesis remains or $k = 60$ (5AM).*

Remark 7. *The experiments are executed during midnight when the ratio between leakage and total inflow is the largest [35] and it is easier to estimate user demands [17]. Through the multiplicative uncertainty in (4-1) the latter effect is mimicked as shown in Figure 4-3.*

4-4 Results

The obtained results of experiments i through iii are summarised in Figure 4-4. The left subplots show the accuracy of the nominal case, with sensors at nodes 14 and 30, and the average distance to the real leak location from the estimated one. The solid and dashed lines correspond with the performance of the AFD and PFD algorithms, respectively, where the different colors distinguish between the results corresponding with the different σ_d values in Figure 4-3. Compared to the PFD algorithm, the accuracy of the AFD algorithm rises with 7, 18, 14 per cent for increasing σ_d, whereas the average distance drops with respectively 0.23,

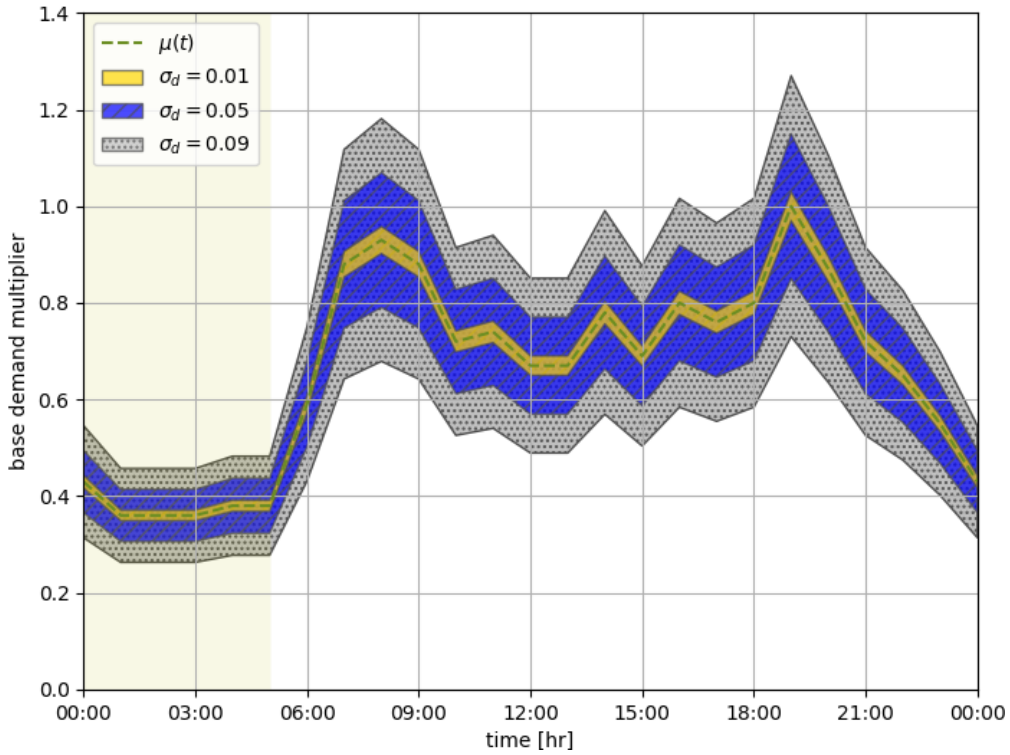


Figure 4-3: Diurnal demand pattern $\mu(t)$, from which μ_k can be looked up, and the $\mu(t) \pm 3\sigma_d$ (99.7%) interval for $\sigma_d \in [0.01, 0.05, 0.090]$. The hours at nighttime during which the analysis is performed are highlighted in yellow.

0.82 and 0.87 kilometer. The notion that $\sigma_d = 0.01$ has the lowest yields, together with the observation that at this uncertainty level the vast majority of incorrectly classified leaks are located in the region around node 5, gives rise to the suspicion that the installed sensors do not produce enough distinctive residuals to isolate the leaks in this region. Therefore, it seems reasonable to place the third sensor for experiment iii at node 5, in order to further improve the results of the AFD algorithm. The corresponding results are graphically displayed in the right-hand subplots in Figure 4-4. Which stands out immediately is the increase in accuracy corresponding for $\sigma_d = 0.01$ to 88 per cent. Compared to the PFD algorithm, the accuracies and average distances of the AFD algorithm respectively rise with 20, 18, 22 per cent and 0.38, 0.64, 0.91 kilometer. Noticeable is that for larger values of σ_d, the additive sensor has no or even a negative effect on the accuracy, as well as on the average distance. This can be explained by the fact that at a certain point, when additive information becomes more uncertain, it distracts the algorithm more than it adds. Furthermore the mean diagnosis time for AFD with respect to PFD increases with 35, 55, 40 minutes of varying σ_d in the nominal case and 50, 65, 50 minutes with an extra sensor.

Figure 4-5 shows the input and likelihood trajectories resulting from a correct actively isolated leak at node 1 for the nominal scenario with $\sigma_d = 0.09$. It shows how the algorithm dynamically updates the inputs based on changing likelihoods. Two periods are distinguished: 1) region selection in the first 45 minutes and further 2) specification of leak location.

1) : The pressures in the network are low and a lot of hypotheses are involved, such that the

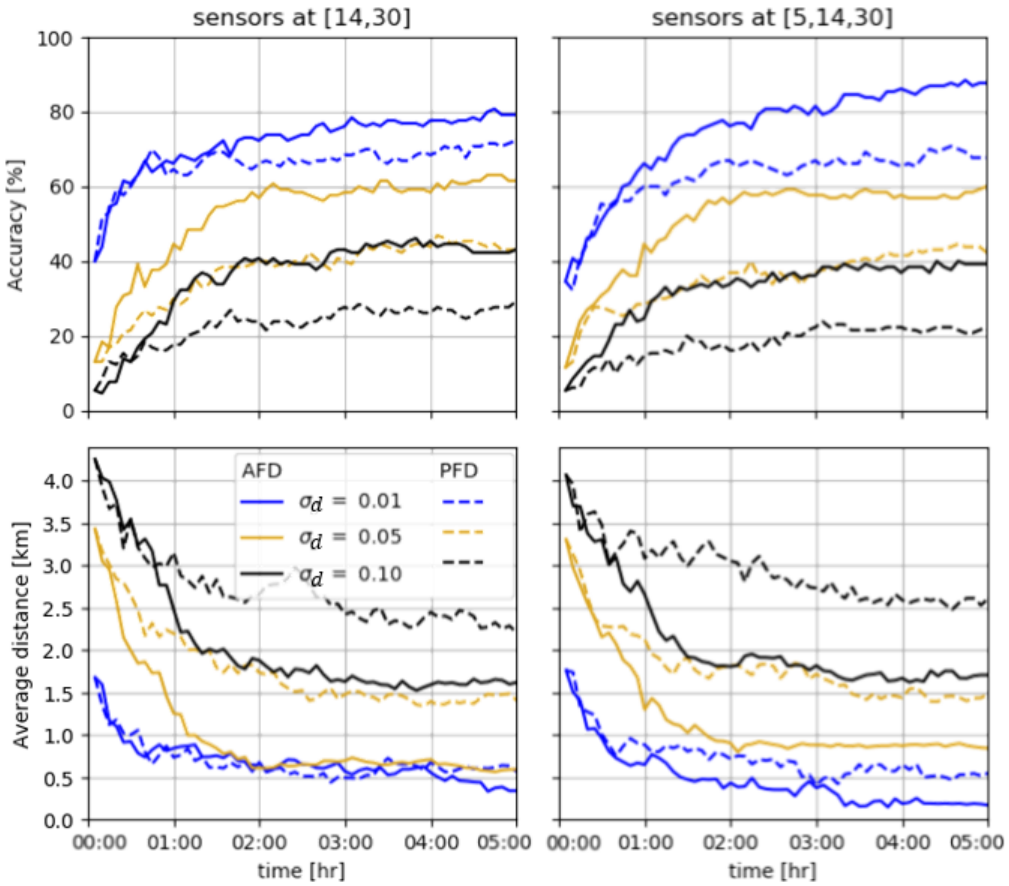


Figure 4-4: The left subplots show the nighttime trajectories of the accuracy and average distance resulting from the nominal scenario for different values of σ_d. The subplots at the right show the same trajectories for the scenario with 3 sensors.

algorithm aims to fan out the corresponding densely packed residual PDFs and to magnify the leak magnitude by increasing the inputs.

2) : A lot of residuals have been rejected and the algorithm mainly focuses on separating the residual PDFs of node 2 and 6, to which increasing the second input has negligible contribution such that it stagnates. The separation between 2 and 6 succeeds as u_1 stagnates as well at 1:30. Then, between 1:30 and 2:00, hypothesis 1 increases in likelihood and the algorithm needs to separate PDFs of nodes 1, 2 and 6, for which u_1 starts increasing once again until it reaches u_{max}. At 2:55 the algorithm terminates, as 1 crosses $\epsilon = 0.95$.

Remark 8. *Extensive simulation results for more values of σ_d are provided in Appendix B. The accuracy and average distance at 5AM are listed for $\sigma_d \in [0.01\ 0.02\ \ldots\ 0.09]$ for both the nominal scenario (Table B-1) and the scenario with 3 sensors (Table B-2).*

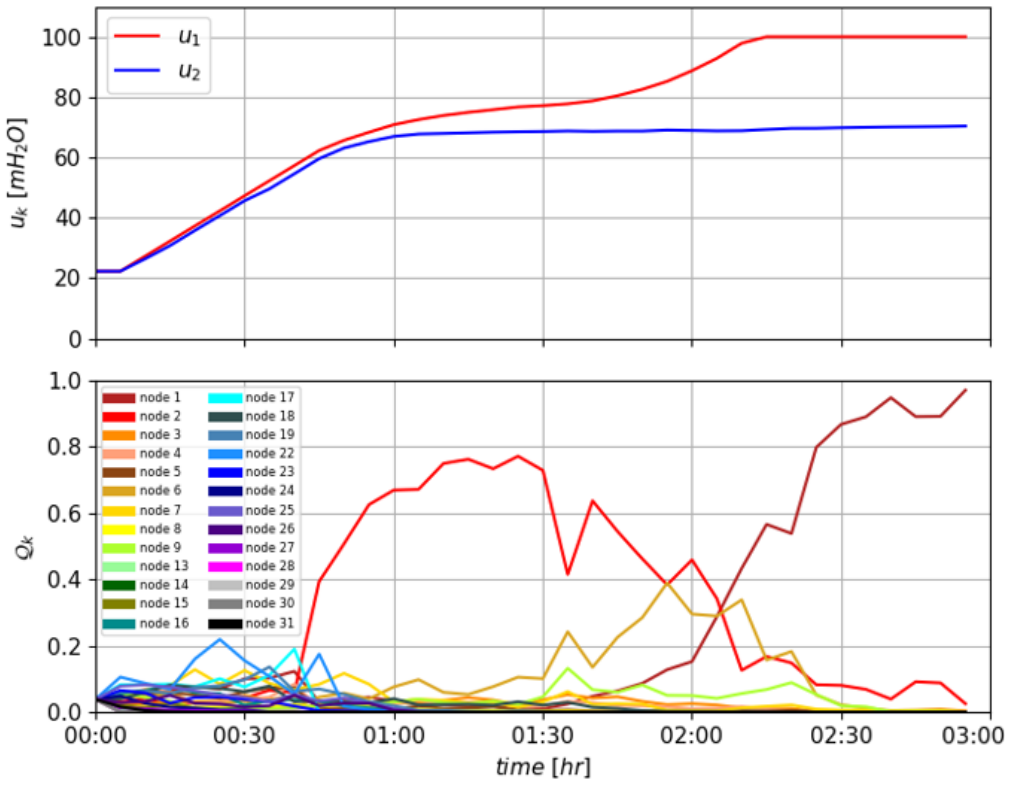


Figure 4-5: Input and likelihood trajectories of resulting from AFD for the nominal scenario with $i^* = 1$. It nicely shows the interaction between the various inputs and the current state of belief. Clearly, when still a lot of hypotheses are involved, the inputs increase at the maximum allowed rate. When hypothesis 2 becomes dominant, close to u_1, u_2 stagnates and u_1 increases slightly to separate the minimal overlap between residual corresponding with hypotheses 2 and 6. This changes when hypothesis 1 comes up and the overlap between 1, 2 and 6 is minimised, which makes the algorithm to increase u_1 at a faster rate up to u_{max}.

Chapter 5

Conclusion

In this thesis a cutting-edge leak localisation method for water distribution networks based on Bayesian classification is improved by actively using inputs to the network that enrich the information in I/O data. It is further shown when and how sensitivity matrices can be computed analytically. The classification is based on a residual, which reflects the deviation of the real network from the predictions made by a stochastic hydraulic leakless model. At each time step the residual PDFs for considered leak locations are estimated, based on which the likelihoods of each leak location is updated according to Bayes' rule and the inputs are updated in an economical way to maximise the overlap between the residual PDFs weighed with their corresponding likelihood, i.e. inputs are not increased if it not improves I/O data. The method is applied to the benchmark Hanoi network for different degrees of uncertainty and has shown to be significantly faster and more accurate than the passive method applied to the same scenario.

For future follow-up studies it is recommended to focus on the optimal placement of sensors and inputs to facilitate AFD, as both have significant impact on its performance.

Appendix A

Static head loss model

Consider the schematic representation in Figure A-1 of a single pipeline with radius R connecting i and k. Water flows in from the cross section at i and flows out at the cross section at k. One-dimensional, steady flow is assumed, which means that the velocity profile v does not vary with longitudinal coordinate x but only with radial coordinate r and does not change over time, i.e. $\partial u/\partial x = 0$ and $\partial u/\partial t = 0$.

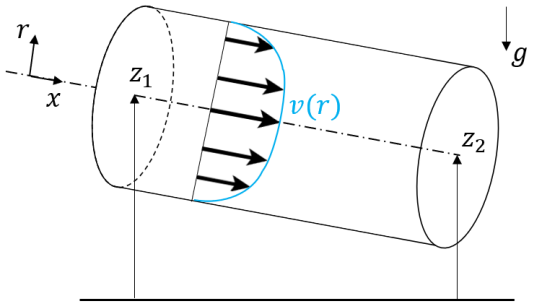


Figure A-1: Schematic representation of a single pipeline between node i and k with radius R and velocity profile $v(r)$.

Furthermore incompressible flow, $\partial\rho/\partial t = 0$, can be assumed as long as the average velocity v_{avg} in $[m/s]$

$$v_{avg} = \frac{1}{A}\int_A v\, dA = \frac{1}{\pi R^2}\int_0^{2\pi}\int_0^R v(r)\, dr\, d\theta \tag{A-1}$$

does not exceed $Ma = v_{avg}/a \leqslant 0.3$, where ρ is the density of water in $[kg/m^3]$, Ma is the dimensionless Mach number of the flow, a is the speed of sound in water in $[m/s]$ and θ is the angular coordinate in the cross sectional plane with surface A in $[m^2]$. An average velocity of the flow of approximately 450 m/s is needed to cross this limit, which occurrence can practically be excluded in a Water Distribution Network (WDN) [36, p.132]. Combining the conservation of mass principle with $\partial\rho/\partial t = 0$ directly results that the inflow q_i equals the outflow q_k, i.e. $q_i = q_k$ in $[m^3/s]$, from which, given a constant perimeter $A(x) = A$, it

follows that $v_{avg,i} = v_{avg,k}$. As the flow and density are constant it follows that the mass flow is constant as well, i.e. $\dot{m}_i = \dot{m}_k = \dot{m}$ in $[kg/s]$. Substitution of those relationships into the following energy balance with units $[J/s]$

$$\dot{Q} - \dot{W} = -\dot{m}_i(\hat{u}_i + \frac{p_i}{\rho} + \frac{1}{2}v_i^2 + gz_i) + \dot{m}_k(\hat{u}_k + \frac{p_k}{\rho} + \frac{1}{2}v_k^2 + gz_k) \tag{A-2}$$

and division by $\dot{m}$ plus assuming the work done by the water is negligible yields

$$\hat{u}_i + \frac{p_1}{\rho} + gz_i = \hat{u}_k + \frac{p_k}{\rho} + gz_k - Q \tag{A-3}$$

with units $[J/kg]$, where $\dot{Q}$ and Q denote heat added to the system, $\dot{W}$ is the work done by the system, $\hat{u}$ is the internal energy, p is the pressure in $[N/m^2]$ and z is the elevation with respect to a horizontal datum in $[m]$. On top of that, civil engineers prefer to divide by g to obtain the head form with units in $[m]$

$$\left(\frac{p_i}{\rho g} + z_i\right) = \left(\frac{p_k}{\rho g} + z_k\right) + \frac{\hat{u}_k - \hat{u}_i - q_j}{g} \tag{A-4}$$

where $\psi = \frac{p}{\rho g}$ and $h = \frac{p}{\rho g} + z$ are referred to as pressure head ψ and hydraulic head h, respectively. The term $\frac{\hat{u}_k - \hat{u}_i - q_j}{g}$ reflects the head lost along the trajectory between i and k due to friction, h_f, and the head added by a possible installed pump, h_p. Equation (A-4) can now be rewritten as:

$$h_i - h_k = h_f - h_p \tag{A-5}$$

describing the total head change over the pipeline [19], [22].

Appendix B

Extensive simulation results

Table B-1: The accuracy and average distance for the nominal scenario case for σ_d varying between 0.01 and 0.1. "-A" and "-P" correspond with AFD and PFD results, respectively.

σ_d	AVGD-A [km]	AVGD-P [km]	ACC-A [%]	ACC-P [%]
0.01	0.172	0.549	88.46	68.46
0.02	0.536	0.807	73.85	69.23
0.03	0.403	1.154	74.62	55.38
0.04	0.527	1.294	62.31	47.69
0.05	0.844	1.483	60.77	43.08
0.06	0.748	1.323	57.69	41.54
0.07	0.890	1.778	56.15	34.62
0.08	1.114	2.148	50.77	26.92
0.09	1.354	2.268	46.92	25.38
0.10	1.708	2.612	40.00	23.08

Table B-2: The accuracy and average distance for the scenario with 3 sensors. σ_d is varied between 0.01 and 0.1. "-A" and "-P" correspond with AFD and PFD results, respectively.

σ_d	AVGD-A [km]	AVGD-P [km]	ACC-A [%]	ACC-P [%]
0.01	0.354	0.631	80.00	73.08
0.02	0.269	0.954	77.69	59.23
0.03	0.477	1.023	71.54	61.54
0.04	0.546	1.492	70.77	42.31
0.05	0.685	1.454	62.31	43.85
0.06	0.877	1.562	58.46	44.62
0.07	1.023	1.992	53.08	33.85
0.08	1.500	2.015	40.00	30.00
0.09	1.438	2.192	46.15	32.31
0.10	1.508	2.108	43.85	30.00

Appendix C

Software

The software to execute the simulations for the various scenarios is divided in a couple of interacting files, which content is displayed in the coming sections:

A-1 **main.py**

A-2 **scenario_class.py**

A-3 **scenario_class_passive.py**

A-4 **functions.py**

A-5 **Hanoi_constants_mtp_u.py**

A-6 **Hanoi_mtp_u.inp**

A-1 provides the main Python file which uses the *Scenario* object oriented classes provided in sections A-2 and A-3. Some necessary class independent functions, but used in the *Scenario* classes, are included in A-4. The file containing the used constants is provided in section A-5. Finally in A-6 the input file describing the Hanoi network in Figure 4-1 is displayed. It contains node, pipeline, reservoirs and valve characteristics like elevation, base demands, diameter, roughness, etc.

C-1 main.py

```
""" used packages and classes """
import numpy as np
from scenario_class import Scenario
from scenario_class_passive import Scenario_passive

""" leak characteristics """
radius = np.sqrt(0.00150/np.pi)
```

```
8  location = 1
9
10 """ initialize lists to save Q,u over time"""
11 Q_time_list_active = []
12 u_time_list_active = []
13 Q_time_list_passive = []
14 u_time_list_passive = []
15
16 """ initialise leak scenarios specified in Hanoi_constants.py and
       Hanoi_mtp_u.inp """
17 scenario = Scenario()
18 scenario.R = radius
19 scenario_passive = Scenario_passive()
20 scenario_passive.R = radius
21
22 """ run algorithms over 60 time steps between 0AM and 5AM with a time
       step op 5 minutes """
23 for k in np.arange(1,61,step=1):
24
25     """ stopping criterion """
26     if max(scenario.Q_k)<0.95 and max(scenario_passive.Q_k)<0.95:
27         scenario.k = k
28         scenario_passive.k = k
29
30         if k == 1:
31             """ set leak location """
32             scenario.ll = location
33             scenario_passive.ll = location
34
35             """ create ghat, gmax, gmin, rtilde """
36             scenario.Set_leak_ghat_rtilde()
37
38             """ initialise u_k """
39             scenario_passive.u_k = scenario.u_k
40
41             """ constuct residual PDFs """
42             scenario.Construct_P_hat_r()
43
44             """ provide passive method with same initialisation """
45             scenario_passive.res = scenario.res
46             scenario_passive.g = scenario.g
47             scenario_passive.g_hat = scenario.g_hat
48             scenario_passive.g_min = scenario.g_min
49             scenario_passive.g_max = scenario.g_max
50             scenario_passive.P_hat_r_resampled = scenario.
                   P_hat_r_resampled
51             scenario_passive.Edges_resampled = scenario.Edges_resampled
52
53         """ for k>1 """
54         if k != 1:
55             scenario.Set_leak_ghat_rtilde()
56             scenario_passive.Set_leak_ghat_rtilde()
57             scenario.Construct_P_hat_r()
```

```
58              scenario_passive.Construct_P_hat_r()
59
60              """ update u_k """
61              scenario.Approx_dJdg()
62              scenario.Update_u()
63
64          """ update Q_k """
65          scenario.Update_Q()
66          scenario_passive.Update_Q()
67
68          """ print summary """
69          scenario.Summary()
70          scenario_passive.Summary()
71
72          """ collect trajectory """
73          u_time_list_active.append(scenario.u_k)
74          Q_time_list_active.append(scenario.Q_k)
75          u_time_list_passive.append(scenario_passive.u_k)
76          Q_time_list_passive.append(scenario_passive.Q_k)
77
78      """ when passive method is terminated do: """
79      if max(scenario.Q_k)<0.95 and max(scenario_passive.Q_k)>=0.95:
80          scenario_passive.k = k
81          scenario.Set_leak_ghat_rtilde()
82          scenario.Construct_P_hat_r()
83          scenario.Approx_dJdg()
84          scenario.Update_u()
85          scenario.Update_Q()
86          scenario.Summary()
87
88          """ save data """
89          u_time_list_active.append(scenario.u_k)
90          Q_time_list_active.append(scenario.Q_k)
91
92      """ when active method is terminated do: """
93      if max(scenario.Q_k)>=0.95 and max(scenario_passive.Q_k)<0.95:
94          scenario_passive.k = k
95          scenario_passive.Set_leak_ghat_rtilde()
96          scenario_passive.Construct_P_hat_r()
97          scenario_passive.Update_Q()
98          scenario_passive.Summary()
99
100         """ save data """
101         u_time_list_passive.append(scenario_passive.u_k)
102         Q_time_list_passive.append(scenario_passive.Q_k)
```

C-2 scenario_class.py

```
1  import functions as fnc
2  import numpy as np
3  import pickle
4  import matplotlib.pyplot as plt
```

```
5  import Hanoi_constants_mtp_u as c
6  import scipy.stats as st
7  import pandas as pd
8  import wntr
9
10 class Scenario:
11     # function to initialize scenario
12     def __init__(self):
13         self.name     = c.name      # name network
14         self.k        = c.k_0       # time k
15         self.Q_k      = c.Q_0       # vector with likelihoods P(r_tilde|i
               )
16         self.file     = c.file      # .inp file describing DMA
17         self.n_n      = c.n_n       # number of nodes
18         self.n_p      = c.n_p       # number of pipelines
19         self.n_s      = c.n_s       # number of pressure sensors
20         self.res_loc  = c.res_loc   # list with sensored nodes
21         self.std_g    = c.std_g     # std of leak magnitude estimation in
                Set_leak()
22         self.I_n      = c.I_n       # vector with node labels
23         self.u_k      = c.u_0       # minimum input to be regularised
               with critical point
24         self.u_min    = c.u_min     # minimum input
25         self.u_max    = c.u_max     # maximum input
26         self.dudk_max = c.dudk_max  # maximum input step between two k's
27         self.N        = c.N         # number of Monte Carlo simulations
28         self.eta      = c.eta       # step size
29         self.eta_bar_r= c.eta_bar_r # step size residuals
30         self.epsilon  = c.epsilon   # value below which Q_k(i) is set to
               0
31         self.alpha    = c.alpha     # leak model coefficient
32         self.A_10     = c.A_10      # incidence matrix known head nodes
33         self.A_12     = c.A_12      # incidence matrix unknown head nodes
34         self.grid     = c.grid      # to make raster in residual space
35         self.resam_N  = c.resam_N   # number of datapoints to resample
               from kernelized PDF
36         self.prv_loc  = c.prv_loc   # node numbers at which prv is
               installed
37         self.std_dmds = c.std_dmds  # standard deviation of demands
38         self.n_u      = c.n_u       # number of inputs
39         self.HW_coeff = c.HW_coeff  # Hazen Williams coefficient
40
41     # function to set leak at self.ll and compute r_tilde(k), g_min,
          g_max
42     def Set_leak_ghat_rtilde(self):
43
44         # start analysis, u_0 should not be lower than 15 m anywhere in
               leaky case
45         DMA_SLL = self.create_model()
46         self.change_demands(DMA_SLL)
47         self.change_prv_setting(DMA_SLL)
48         junction_SLL = DMA_SLL.get_node(str(self.ll))
```

```
49          junction_SLL.add_leak(DMA_SLL, area=np.pi*(self.R)**2, start_time
                = 0)
50          results = fnc.simulate_model_wntr(DMA_SLL)
51          while float(results.node['pressure']['31']) <15:
52              if np.any(self.u_k <= self.u_max-0.1*np.ones(self.n_u)):
53                  self.u_k = self.u_k + 0.1*np.ones(self.n_u)
54                  self.change_prv_setting(DMA_SLL)
55                  results = fnc.simulate_model_wntr(DMA_SLL)
56
57          # create model with leak at self.ll
58          DMA_0 = self.create_model()
59          DMA_SL = self.create_model()
60
61          # change demands to time
62          self.change_demands(DMA_0)
63          self.change_demands(DMA_SL)
64
65          # set leak at node k of leak model of size g
66          junction = DMA_SL.get_node(str(self.ll))
67          junction.add_leak(DMA_SL, area=np.pi*(self.R)**2, start_time = 0)
68
69           # set input PRV to u_k
70          self.change_prv_setting(DMA_0)
71          self.change_prv_setting(DMA_SL)
72
73          # simulate nominal model and obtain results
74          results_0 = fnc.simulate_model_wntr(DMA_0)
75          pressure_0 = results_0.node['pressure']
76
77          # simulate leaky model and obtain results
78          results_SL = fnc.simulate_model_wntr(DMA_SL)
79          pressure_SL = results_SL.node['pressure']
80
81          # compute residual vector at self.k
82          res_dum = []
83          for i in range(0,len(self.res_loc)):
84              res_dum.append(float(pressure_0[str(self.res_loc[i])]-
                    pressure_SL[str(self.res_loc[i])]))
85          self.res = res_dum
86
87          # compute g_hat, g_min, g_max at self.k
88          self.g = np.abs(np.sum(np.asarray(results_SL.node['demand'])))
89          self.g_hat = np.random.normal(0,self.std_g)+self.g
90          self.g_min = self.g_hat-self.std_g*3
91          self.g_max = self.g_hat+self.std_g*3
92          self.g_over_total_demands = self.g/sum(np.asarray(results_SL.node
                ['demand'])[0,0:31])
93
94      # This function constructs P_hat_r for all i for which self.Q_k != 0.
            Output list of size self.n_n
95      def Construct_P_hat_r(self):
96          # vary g uniformly over [g_min,g_max]
97          self.g_array = np.linspace(self.g_min,self.g_max,self.N)
```

```
98          # create Data, Data_bar containing Di for which holds that Q[i
                ]!=0
99          self.Data = []
100         self.Data_bar = []
101         for i in range(0,len(self.I_n)):
102             if self.Q_k[i] != 0:
103                 # create N realisations of leaks in [g_min, g_max] at
                        node i
104                 self.i = self.I_n[i]
105                 D,D_bar = self.D_i()
106                 # save to Data
107                 self.Data.append(D)
108                 self.Data_bar.append(D_bar)
109
110         # resample data using kernelized PDF
111         self.Resampled_data = []
112         self.Resampled_data.append(self.Construct_resampled_data(self.
                Data,row=0))
113
114         self.Resampled_data_bar = []
115         for i in range(0,self.n_u):
116             self.Resampled_data_bar.append(self.Construct_resampled_data(
                    self.Data_bar,row=i))
117
118         # compute mines, maxes, mines_resampled, maxes_resampled, ranges
                and nbins for [Data,Resampled_data] and [Data_bar,
                Resampled_data_bar]
119         self.mines_resampled, self.maxes_resampled, self.ranges_data,
                self.nbins = self.Min_max_range_nbins(self.Resampled_data)
120         self.mines_bar_resampled, self.maxes_bar_resampled, self.
                ranges_data_bar, self.nbins_bar = self.Min_max_range_nbins(
                self.Resampled_data_bar)
121
122         # constuct normalized histograms (PDF) P_hat_r, P_hat_r_bar (
                P_hat_r_resampled P_hat_r_bar_resampled)<-- These we need
123         Histograms_resampled, edges_resampled = self.
                Construct_normalized_histograms(self.Resampled_data[0],self.
                nbins[0],self.ranges_data[0])
124         Histograms_resampled = [Histograms_resampled]
125         edges_resampled = [edges_resampled]
126
127         Histograms_resampled_bar = []
128         edges_bar_resampled = []
129         for nu in range(0,self.n_u):
130             Histograms_resampled_bar_dummy, edges_bar_resampled_dummy =
                    self.Construct_normalized_histograms(self.
                    Resampled_data_bar[nu],self.nbins_bar[nu],self.
                    ranges_data_bar[nu])
131             Histograms_resampled_bar.append(
                    Histograms_resampled_bar_dummy)
132             edges_bar_resampled.append(edges_bar_resampled_dummy)
133
134         self.P_hat_r_resampled = []
```

```
135         self.Edges_resampled = []
136         self.P_hat_r_bar_resampled = []
137         self.Edges_bar_resampled = []
138         for nu in range(0,self.n_u):
139             counter = -1
140             self.P_hat_r_bar_resampled.append([])
141             self.Edges_bar_resampled.append([])
142             for i in range(0,len(self.I_n)):
143                 if self.Q_k[i] != 0:
144                     counter = counter + 1
145                     if nu == 0:
146                         self.P_hat_r_resampled.append(
                                Histograms_resampled[nu][counter])
147                         self.Edges_resampled.append(edges_resampled[nu][
                                counter])
148                     self.P_hat_r_bar_resampled[nu].append(
                            Histograms_resampled_bar[nu][counter])
149                     self.Edges_bar_resampled[nu].append(
                            edges_bar_resampled[nu][counter])
150                 else:
151                     if nu == 0:
152                         self.P_hat_r_resampled.append([0])
153                         self.Edges_resampled.append([0])
154                     self.P_hat_r_bar_resampled[nu].append([0])
155                     self.Edges_bar_resampled[nu].append([0])
156 
157     # This function constructs a discrete distribution for node i
158     def Construct_resampled_data(self,Data,row):
159         Resampled_data = []
160         Data_shape = np.asarray(Data).shape #(n_ll,N,(1 or n_u),n_s)
161         for k in range(0,Data_shape[0]):
162             # create list of form [[x],[y],[z]]
163             list_dum = []
164             for i in range(0,Data_shape[3]):
165                 list_dum.append([])
166                 for j in range(0,Data_shape[1]):
167                     list_dum[i].append(Data[k][j][row][i])
168             values = np.vstack(list_dum)
169             P_Density_Function = st.gaussian_kde(values)
170             Resampled_data.append(P_Density_Function.resample(int(self.
                    resam_N)))
171 
172         return Resampled_data
173 
174     # function to determine borders of Data and ranges,nbins
175     def Min_max_range_nbins(self,Resampled_data):
176 
177         """ compute maxes and mines of self.Resampled_data and self.
                Resampled_data_bar """
178         mines_resampled = []
179         maxes_resampled = []
180         Resampled_data_shape = np.asarray(Resampled_data).shape #((1/n_u)
                ,n_ll,n_s,resam_N)
```

```
181         for nu in range(0,Resampled_data_shape[0]):
182             for i in range(0,Resampled_data_shape[2]):
183                 maxes_dum = []
184                 mines_dum = []
185                 for j in range(0,Resampled_data_shape[1]):
186                     mines_dum.append(min(Resampled_data[nu][j][i]))
187                     maxes_dum.append(max(Resampled_data[nu][j][i]))
188                 mines_resampled.append(float(min(mines_dum)))
189                 maxes_resampled.append(float(max(maxes_dum)))
190         mines_resampled = np.asarray(mines_resampled).reshape([
                Resampled_data_shape[0],Resampled_data_shape[2]])
191         maxes_resampled = np.asarray(maxes_resampled).reshape([
                Resampled_data_shape[0],Resampled_data_shape[2]])
192
193         """ residual space borders and number of bins """
194         for i in range(0,Resampled_data_shape[0]):
195             mines_resampled[i] = np.floor(mines_resampled[i]*10)/10
196             maxes_resampled[i] = np.ceil(maxes_resampled[i]*10)/10
197
198         """ construct ranges """
199         ranges_data = []
200         for nu in range(0,Resampled_data_shape[0]):
201             ranges_data.append([])
202             for i in range(0,Resampled_data_shape[2]):
203                 ranges_data[nu].append([mines_resampled[nu][i],
                        maxes_resampled[nu][i]])
204
205         """ number of bins """
206         nbins = []
207         for nu in range(0,Resampled_data_shape[0]):
208             for j in range(0,Resampled_data_shape[2]):
209                 nbins.append(int(round((maxes_resampled[nu][j]-
                        mines_resampled[nu][j])/self.grid)))
210         nbins = np.asarray(nbins).reshape([Resampled_data_shape[0],
                Resampled_data_shape[2]])
211
212         return mines_resampled, maxes_resampled, ranges_data, nbins
213
214     # function to construct Histograms, Edges for (Data,nbins,ranges_data
            )
215     def Construct_normalized_histograms(self,Data,nbins,ranges_data):
216         Histograms = []
217         Edges = []
218
219         for i in range(0,len(Data)):
220             data_dum = []
221             for j in range(0,len(self.res_loc)):
222                 if type(Data[0])== list:
223                     data_dum.append(list(zip(*Data[i]))[j])
224                 if type(Data[0])== np.ndarray:
225                     data_dum.append(Data[i][j])
226
```

```
227             H, edges = np.histogramdd(data_dum, bins=nbins, range=
                    ranges_data)
228
229             Hsum = sum(H)
230             for i in range(0,len(self.res_loc)-1):
231                 Hsum = sum(Hsum)
232             Histograms.append(H/Hsum)
233             Edges.append(edges)
234         return Histograms, Edges
235
236     # function which takes scenario and computes Datasets D and Dbar for
            node i
237     def D_i(self):
238         D = []
239         D_bar = []
240
241         # for N realisations of leaks
242         for m in range(0,self.N):
243
244             "construct D[m]"
245             # create nominal DMA model and model with leak at node k
246             DMA_0 = self.create_model()
247             DMA_L = self.create_model()
248
249             # set input PRV to u
250             self.change_prv_setting(DMA_0)
251             self.change_prv_setting(DMA_L)
252
253             # change demands to time
254             self.change_demands(DMA_0)
255             self.change_demands(DMA_L)
256
257             # set leak at node k of leak model of size g
258             junction = DMA_L.get_node(str(self.i))
259             dummy = junction.demand_timeseries_list[0].base_value
260             dummy = dummy + self.g_array[m]
261             junction.demand_timeseries_list[0].base_value = dummy
262
263             # simulate nominal and leaky model and obtain results
264             results_0   = fnc.simulate_model_wntr(DMA_0)
265             pressure_0  = results_0.node['pressure']
266             results_L   = fnc.simulate_model_wntr(DMA_L)
267             pressure_L  = results_L.node['pressure']
268             flow_L      = results_L.link['flowrate']
269
270             p_0 = []
271             p_L = []
272             for j in self.res_loc:
273                 p_0.append(pressure_0[str(j)])
274                 p_L.append(pressure_L[str(j)])
275
276             # compute residuals
277             r_s = []
```

```
278             for j in range(0,len(self.res_loc)):
279                 r_s.append(float(np.asarray(p_0[j])-np.asarray(p_L[j])))
280             D.append([r_s])
281 
282             """construct D_bar[m]"""
283             "dr/dg"
284 
285             # compute A_11 #
286             A_11 = np.zeros([self.n_p,self.n_p])
287             for k in range(1,self.n_p+1): # pipelines 1-33
288                 pipe = DMA_L.get_link(str(k))
289                 r_j = self.HW_coeff * pipe.length * pipe.roughness
                        **(-1.852) * pipe.diameter**(-4.871)
290 
291                 A_11[k-1,k-1] = r_j * np.abs(float(flow_L[str(k)]))
                        **(0.852)
292 
293             # Create A_0
294             A_0 = np.zeros([self.n_n,self.n_n])
295 
296             # create N_A
297             N_A = np.zeros((self.n_p,self.n_p))
298             np.fill_diagonal(N_A,1.852)
299 
300             # create A=[N_A*A_11 A_12; A21 A_0]
301             dF_1 = np.concatenate((N_A.dot(A_11),self.A_12),axis=1)
302             dF_2 = np.concatenate((np.transpose(self.A_12),A_0),axis=1)
303             A = np.concatenate((dF_1,dF_2))
304 
305             # create B # only valid for H=I
306             G_2 = np.zeros([self.n_n])
307             G_2[self.i-2] = 1
308             G = np.concatenate((np.zeros([self.n_p]),G_2))
309 
310             # compute S
311             S_g = np.linalg.inv(A).dot(G)
312 
313             # [dr_14/dg dr_30/dg]
314             self.dr_dg =[]
315             for j in self.res_loc:
316                 self.dr_dg.append(-S_g[self.n_p+j-1])
317 
318             "dh_i/du"
319             self.dhi_du = []
320             for nu in range(0,self.n_u):
321                 B_unu = np.concatenate((self.A_10[:,nu],np.zeros([self.
                        n_n])))
322                 S_unu = np.linalg.inv(A).dot(B_unu)
323                 self.dhi_du.append(S_unu[self.n_p+self.i-1])
324 
325             # D_bar = [[dr_1/du_1 ... dr_1/du_n], [dr_2/du_1 ... dr_2/
                du_n],...]
326             r_bar = []
```

```
327             for nu in range(0,self.n_u):
328                 r_bar.append([])
329                 for j in range(0,len(self.res_loc)):
330                     r_bar[nu].append(float(r_s[j]+self.eta_bar_r * self.
                            dhi_du[nu] * self.dr_dg[j]))
331             D_bar.append(r_bar)
332 
333         return D, D_bar
334 
335     # Compute approximation of dJ/du
336     def Approx_dJdg(self):
337         ## compute H_hat_i_j
338         self.QHQ_hat = 0
339         self.QHQ_hat_bar = np.zeros([self.n_u])
340 
341         for i in range(0,len(self.I_n)-1):
342             for j in range(i+1,len(self.I_n)):
343                 if self.Q_k[i] != 0 and self.Q_k[j] != 0:
344                     self.QHQ_hat     = self.QHQ_hat     + self.Q_k[i]*fnc
                            .Compute_Hellinger_distance(self.P_hat_r_resampled
                            [i],    self.P_hat_r_resampled[j])     *self.Q_k[j
                            ]
345                     for nu in range(0,self.n_u):
346                         self.QHQ_hat_bar[nu] = self.QHQ_hat_bar[nu] +
                                self.Q_k[i]*fnc.Compute_Hellinger_distance(
                                self.P_hat_r_bar_resampled[nu][i],self.
                                P_hat_r_bar_resampled[nu][j]) *self.Q_k[j]
347         self.dJdg_hat = []
348         for nu in range(0,self.n_u):
349             self.dJdg_hat.append(self.QHQ_hat_bar[nu]-self.QHQ_hat)
350 
351     # Update u
352     def Update_u(self):
353         deltas_u=np.zeros([self.n_u])
354         for nu in range(0,self.n_u):
355             deltas_u[nu] = self.eta*self.dJdg_hat[nu]
356 
357         ## Restrictions on stepsize u
358         # if step is falls within range [0, dudk_max]
359         for nu in range(0,self.n_u):
360             if np.abs(deltas_u[nu]) > self.dudk_max[nu]:
361                 deltas_u[nu] = self.dudk_max[nu]
362 
363         self.u_k = self.u_k + deltas_u
364 
365         ## Restrictions on magnitude u
366         # if updated u_k is larger than u_max or smaller than u_min -->
               set self.u_k=self.u_max
367         for nu in range(0,self.n_u):
368             dummy = np.zeros([self.n_u])
369             if self.u_k[nu] > self.u_max[nu]:
370                 for nuk in range(0,self.n_u):
371                     if nu == nuk:
```

```
372                         dummy[nuk] = self.u_max[nuk]
373                     else:
374                         dummy[nuk] = self.u_k[nuk]
375                 print("Updated u_k is larger than and therefore set to
                        u_max")
376             elif self.u_k[nu] < self.u_min[nu]:
377                 for nuk in range(0,self.n_u):
378                     if nu == nuk:
379                         dummy[nuk] = self.u_min[nuk]
380                     else:
381                         dummy[nuk] = self.u_k[nuk]
382                 print("Updated u is smaller than and therefore set to
                        u_min")
383             else:
384                 dummy=self.u_k
385
386             self.u_k = dummy
387
388     # This function returns the posterior probability vector given
            residual vector r_k=(r_14,r_30)
389     def Update_Q(self):
390         Pr_likelihood =[]
391         Pr_r=0
392
393         # for node 2-31
394         for j in range(0,len(self.I_n)):
395             if self.Q_k[j] != 0:
396                 # compute the likelihood P(r|1^{[j]})
397                 P_hat_r_j = self.P_hat_r_resampled[j]  #PP_hat_r|j
398                 edges_j = self.Edges_resampled[j]
399                 Pr_likelihood_j = fnc.get_hist_value(P_hat_r_j,edges_j,
                        self.res)
400                 Pr_likelihood.append(Pr_likelihood_j)
401                 Pr_r = Pr_r + Pr_likelihood_j * self.Q_k[j]
402             else:
403                 Pr_likelihood.append(0)
404
405
406         self.Q_k = Pr_likelihood*self.Q_k/Pr_r
407         for i in range(0,len(self.Q_k)):
408             if self.Q_k[i] < self.epsilon:
409                 self.Q_k[i] =0
410
411         # normalize Q_k
412         sum_Q_k = sum(self.Q_k)
413         if sum_Q_k != 1:
414             print("Q_k is renormalised")
415             for i in range(0,len(self.I_n)):
416                 self.Q_k[i] = self.Q_k[i]/sum_Q_k
417
418         #Quit and make belief vector of 1 at first element
419         if np.any(np.isnan(self.Q_k)) == True:
420             self.Q_k = np.zeros(len(self.I_n))
```

```
421             self.Q_k[0] = 1
422
423         # calculate maximum(Q_k[i]) --> i*
424         self.i_star_fnc()
425
426     ## function which determines leak location(s) with Q_max
427     def i_star_fnc(self):
428         zippedList = list(zip(self.Q_k))
429
430         column_names = []
431         for i in range(0,len(self.I_n)):
432             column_names.append(str(self.I_n[i]))
433
434         self.Q_k_df = pd.DataFrame(zippedList, index=column_names)
435
436         self.i_star = []
437         Q_max = max(self.Q_k)
438         for i in range(0,len(self.Q_k)):
439             if self.Q_k[i] == Q_max:
440                 self.i_star.append(self.Q_k_df.index[i])
441
442     def Summary(self):
443        # print('Summary AFD '+self.name)
444         print('leak magnitude g('+str(self.k)+')='+str(self.g)+'g_hat(k='
               +str(self.k)+')='+str(self.g_hat)+'['+str(self.g_min)+str(self
               .g_max)+']')
445         print('u('+str(self.k)+')='+str(self.u_k))
446         print('Q('+str(self.k)+')='+str(self.Q_k))
447         print('i_star('+str(self.k)+')='+str(self.i_star))
448
449     """ help functions """
450     # function which creates a wntr model of .inp file x
451     def create_model(self):
452         dma = wntr.network.WaterNetworkModel(self.file)
453         return dma
454
455     # function to change PRV settings to inlet pressures
456     def change_prv_setting(self,dma):
457         counter = -1
458         for i in self.prv_loc:
459             counter = counter +1
460             prv = dma.get_link(str(i))
461             prv.setting = self.u_k[counter]
462
463     # function to change demands at all nodes corresponding with time
464     def change_demands(self,dma):
465         d_mtp_vector = pickle.load(open('d_mtp','rb'))                #
               demand pattern
466         d_mtp = d_mtp_vector[self.k-1]
                                                                  # get demand
               multiplier of time
467         for i in range(1,self.n_n+2):
468             mtp = max(0,d_mtp*np.random.normal(1,self.std_dmds))
```

```
469             fnc.change_node_property(dma,i,'demand',mtp)
```

C-3 scenario_class_passive_snippet.py

This snippet of *scenario_class_passive_snippet.py* shows the three class functions that differ from the *scenario_class.py* in A-2.

```
1     # function to set leak at self.ll and compute r_tilde(k), g_min,
          g_max
2     def Set_leak_ghat_rtilde(self):
3
4         # create model with leak at self.ll
5         DMA_0 = self.create_model()
6         DMA_SL = self.create_model()
7
8         # change demands to time
9         self.change_demands(DMA_0)
10        self.change_demands(DMA_SL)
11
12        # set leak at node k of leak model of size g
13        junction = DMA_SL.get_node(str(self.ll))
14        junction.add_leak(DMA_SL, area=np.pi*(self.R)**2, start_time = 0)
15
16        self.u_k = self.u_min
17        self.change_prv_setting(DMA_SL)
18        results_passive = fnc.simulate_model_wntr(DMA_SL)
19        while float(results_passive.node['pressure']['31'])<15:
20            if np.any(self.u_k <= self.u_max):
21                self.u_k = self.u_k + 0.1*np.ones(self.n_u)
22                self.change_prv_setting(DMA_SL)
23                results_passive = fnc.simulate_model_wntr(DMA_SL)
24            else:
25                print("u_k is larger than u_max")
26
27        # set input PRV to u_k
28        self.change_prv_setting(DMA_0)
29
30
31        # simulate nominal model and obtain results
32        results_0 = fnc.simulate_model_wntr(DMA_0)
33        pressure_0 = results_0.node['pressure']
34
35        # simulate leaky model and obtain results
36        results_SL = fnc.simulate_model_wntr(DMA_SL)
37        pressure_SL = results_SL.node['pressure']
38
39        # compute residual vector at self.k
40        res_dum = []
41        for i in range(0,len(self.res_loc)):
42            res_dum.append(float(pressure_0[str(self.res_loc[i])])-float(
                  pressure_SL[str(self.res_loc[i])]))
43        self.res = res_dum
```

```
44
45          # compute g_hat, g_min, g_max at self.k
46          self.g = np.abs(np.sum(np.asarray(results_SL.node['demand'])))
47          self.g_hat = np.random.normal(0,self.std_g)+self.g
48          self.g_min = self.g_hat-self.std_g*3
49          self.g_max = self.g_hat+self.std_g*3
50          self.g_over_total_demands = self.g/sum(np.asarray(results_SL.node
                ['demand'])[0,0:31])
51
52      # This function constructs P_hat_r for all i for which self.Q_k != 0.
            Output list of size self.n_n
53      def Construct_P_hat_r(self):
54          # vary g uniformly over [g_min,g_max]
55          self.g_array = np.linspace(self.g_min,self.g_max,self.N)
56          # create Data, Data_bar containing Di for which holds that Q[i
                ]!=0
57          self.Data = []
58          for i in range(0,len(self.I_n)):
59              if self.Q_k[i] != 0:
60                  # create N realisations of leaks in [g_min, g_max] at
                        node i
61                  self.i = self.I_n[i]
62                  D = self.D_i()
63                  # save to Data
64                  self.Data.append(D)
65
66          # resample data using kernelized PDF
67          self.Resampled_data = []
68          self.Resampled_data.append(self.Construct_resampled_data(self.
                Data,row=0))
69
70          # compute mines, maxes, mines_resampled, maxes_resampled, ranges
                and nbins for [Data,Resampled_data] and [Data_bar,
                Resampled_data_bar]
71          self.mines_resampled, self.maxes_resampled, self.ranges_data,
                self.nbins = self.Min_max_range_nbins(self.Resampled_data)
72
73
74          # constuct normalized histograms (PDF) P_hat_r, P_hat_r_bar (
                P_hat_r_resampled P_hat_r_bar_resampled)<-- These we need
75          Histograms_resampled, edges_resampled = self.
                Construct_normalized_histograms(self.Resampled_data[0],self.
                nbins[0],self.ranges_data[0])
76          Histograms_resampled = [Histograms_resampled]
77          edges_resampled = [edges_resampled]
78
79
80          self.P_hat_r_resampled = []
81          self.Edges_resampled = []
82          for nu in range(0,self.n_u):
83              counter = -1
84              for i in range(0,len(self.I_n)):
85                  if self.Q_k[i] != 0:
```

```
86                      counter = counter + 1
87                      if nu == 0:
88                          self.P_hat_r_resampled.append(
                                Histograms_resampled[nu][counter])
89                          self.Edges_resampled.append(edges_resampled[nu][
                                counter])
90                  else:
91                      if nu == 0:
92                          self.P_hat_r_resampled.append([0])
93                          self.Edges_resampled.append([0])
94
95      # function which takes scenario and computes Datasets D and Dbar for
            node i
96      def D_i(self):
97          D = []
98
99          # for N realisations of leaks
100         for m in range(0,self.N):
101
102             "construct D[m]"
103             # create nominal DMA model and model with leak at node k
104             DMA_0 = self.create_model()
105             DMA_L = self.create_model()
106
107             # set input PRV to u
108             self.change_prv_setting(DMA_0)
109             self.change_prv_setting(DMA_L)
110
111             # change demands to time
112             self.change_demands(DMA_0)
113             self.change_demands(DMA_L)
114
115             # set leak at node k of leak model of size g
116             junction = DMA_L.get_node(str(self.i))
117             dummy = junction.demand_timeseries_list[0].base_value
118             dummy = dummy + self.g_array[m]
119             junction.demand_timeseries_list[0].base_value = dummy
120
121             # simulate nominal and leaky model and obtain results
122             results_0   = fnc.simulate_model_wntr(DMA_0)
123             pressure_0  = results_0.node['pressure']
124             results_L   = fnc.simulate_model_wntr(DMA_L)
125             pressure_L  = results_L.node['pressure']
126
127             p_0 = []
128             p_L = []
129             for j in self.res_loc:
130                 p_0.append(pressure_0[str(j)])
131                 p_L.append(pressure_L[str(j)])
132
133             # compute residuals
134             r_s = []
135             for j in range(0,len(self.res_loc)):
```

```
136                 r_s.append(float(np.asarray(p_0[j])-np.asarray(p_L[j])))
137             D.append([r_s])
138 
139         return D
140 
141 
142         deltas_u=np.zeros([self.n_u])
143         for nu in range(0,self.n_u):
144             deltas_u[nu] = self.eta*self.dJdg_hat[nu]
145 
146         ## Restrictions on stepsize u
147         # if step is falls within range [0, dudk_max]
148         for nu in range(0,self.n_u):
149             if np.abs(deltas_u[nu]) > self.dudk_max[nu]:
150                 deltas_u[nu] = self.dudk_max[nu]
151 
152         self.u_k = self.u_k + deltas_u
153 
154         ## Restrictions on magnitude u
155         # if updated u_k is larger than u_max or smaller than u_min -->
              set self.u_k=self.u_max
156         for nu in range(0,self.n_u):
157             dummy = np.zeros([self.n_u])
158             if self.u_k[nu] > self.u_max[nu]:
159                 for nuk in range(0,self.n_u):
160                     if nu == nuk:
161                         dummy[nuk] = self.u_max[nuk]
162                     else:
163                         dummy[nuk] = self.u_k[nuk]
164                 print("Updated u_k is larger than and therefore set to
                      u_max")
165             elif self.u_k[nu] < self.u_min[nu]:
166                 for nuk in range(0,self.n_u):
167                     if nu == nuk:
168                         dummy[nuk] = self.u_min[nuk]
169                     else:
170                         dummy[nuk] = self.u_k[nuk]
171                 print("Updated u is smaller than and therefore set to
                      u_min")
172             else:
173                 dummy=self.u_k
174 
175             self.u_k = dummy
```

C-4 functions.py

```
1 import wntr
2 import numpy as np
3 import pickle
4 
```

```
5  # function which simulates dma using WNTR Simulater and returns results
6  def simulate_model_wntr(dma):
7      sim = wntr.sim.WNTRSimulator(dma)
8      return sim.run_sim()
9
10 # function to change link properties length, diameter, roughness
11 def change_link_property(dma,link,link_property,multiplier):
12     pipe = dma.get_link(link)
13     if link_property == 'length':
14         pipe.length = pipe.length*multiplier
15     elif link_property == 'diameter':
16         pipe.diameter = pipe.diameter*multiplier
17     elif link_property == 'roughness':
18         pipe.roughness = pipe.roughness*multiplier
19     else:
20         print("You enteret an invalid link_property")
21
22 # function to change node properties elevation and demand multiplier
23 def change_node_property(dma,node,node_property,multiplier):
24     node = dma.get_node(node)
25     if node_property == 'elevation':
26         node.elevation = node.elevation*multiplier
27     elif node_property == 'demand':
28         node.demand_timeseries_list[0].base_value=node.
               demand_timeseries_list[0].base_value*multiplier
29     else:
30         print("You entered an invalid link_property")
31
32
33 # function to compute Hellinger distance between two discrete PDFs
34 def Compute_Hellinger_distance(P_1,P_2):
35
36     #Bhattacharyya coefficient is sqrt(vec(P_1)*vec(P_2))
37     Bh_coef_dummy1 = np.asarray(P_1).flatten()
38     Bh_coef_dummy2 = np.asarray(P_2).flatten()
39     Bh_coef = 0
40     if len(Bh_coef_dummy1) == len(Bh_coef_dummy2):
41         for i in range(0,len(Bh_coef_dummy1)):
42             Bh_coef = Bh_coef + np.sqrt(Bh_coef_dummy1[i]*Bh_coef_dummy2[
                   i])
43         H_dist = np.sqrt(1-Bh_coef)
44     else:
45         print("P_1 and P_2 do not have identical shapes in
               Compute_Hellinger_distance")
46     return H_dist
47
48 # function to determine the histogram value for a new data point --> use
      to find PDF(res)
49 def get_hist_value(Hist,Edges,res):
50
51     # the stepsizes can vary with axis and is the distance between two
          bin edges
52     stepsizes = []
```

```
53      tuple_dum = ()
54      for i in range(0, len(Edges)):
55          stepsizes.append(abs(Edges[i][1]-Edges[i][2]))
56          # calculation in which bin the ith parameter of res falls
57          tuple_dum = tuple_dum + (int(np.trunc((res[i]-Edges[i][0])/
                stepsizes[i])),)
58      return Hist[tuple_dum]
```

C-5 Hanoi_constants_mtp_u.py

```
1  import pickle
2  import numpy as np
3
4  """ constants """
5  file      = 'Hanoi_mtp_u.inp'
6  name      = 'Hanoi network multiple inputs'
7  n_n       = 31
8  n_p       = 35
9  n_s       = 3
10 n_u       = 2
11 res_loc   = np.array([14,22])
12 I_n       = np.array
       ([1,2,3,4,5,6,7,8,9,13,14,15,16,17,18,19,22,23,24,25,26,27,28,29,30,31])

13 A_10      = pickle.load(open('A_10_mtp_u','rb'))
14 A_12      = pickle.load(open('A_12_mtp_u','rb'))
15 k_0       = 1
16 u_0       = np.array([15,15])
17 Q_0       = np.ones([len(I_n)])*(1/len(I_n))
18 prv_loc   = np.array([36, 37])
19 std_g     = 0.003
20 alpha     = 0.5
21 std_dmds  = 0.1
22 HW_coeff  = 10.666829500036352
23
24 """ input constraints """
25 u_min     = np.array([15, 15])
26 u_max     = np.array([100, 100])
27 dudk_max  = np.array([5, 5])
28
29 """ tuning parameters """
30 N           = 80
31 resam_N     = 1e5
32 eta         = 50
33 eta_bar_r   = -0.1
34 epsilon     = 5e-4
35
36 """ bin width histograms residual PDFs """
37 grid      = 0.1
```

C-6 Hanoi_mtp_u.inp

```
[JUNCTIONS]
;ID      Elev  Demand
 1       30      247.22 ;
 2       30      236.11 ;
 3       30      36.11  ;
 4       30      201.39 ;
 5       30      279.17 ;
 6       30      375    ;
 7       30      152.78 ;
 8       30      145.83 ;
 9       30      145.83 ;
 10      30      138.89 ;
 11      30      155.56 ;
 12      30      261.11 ;
 13      30      170.83 ;
 14      30      77.78  ;
 15      30      86.11  ;
 16      30      240.28 ;
 17      30      373.61 ;
 18      30      16.67  ;
 19      30      354.17 ;
 20      30      258.33 ;
 21      30      134.72 ;
 22      30      290.28 ;
 23      30      227.78 ;
 24      30      47.22  ;
 25      30      250    ;
 26      30      102.78 ;
 27      30      80.56  ;
 28      30      100    ;
 29      30      100    ;
 30      30      29.17  ;
 31      30      223.61 ;
 32      30      0      ;
 33      0       0      ;
 36      0       0      ;
 37      30      0      ;

[RESERVOIRS]
;ID Head
 34 1000 ;
 35 1000 ;



[PIPES]
;ID Node1 Node2 Length Diameter Roughness MinorLoss Status
 1  32  1  100   1016  130 0 Open ;
 2   1  2  1350  1016  130 0 Open ;
 3   2  3  900   1016  130 0 Open ;
 4   3  4  1150  1016  130 0 Open ;
 5   4  5  1450  1016  130 0 Open ;
 6   5  6  450   1016  130 0 Open ;
```

```
 7    6   7   850    1016   130  0  Open  ;
 8    7   8   850    1016   130  0  Open  ;
 9    8   9   800    1016   130  0  Open  ;
 10   9   10  950    762    130  0  Open  ;
 11   10  11  1200   762    130  0  Open  ;
 12   11  12  3500   609.6  130  0  Open  ;
 13   37  9   100    1016   130  0  Open  ;
 14   9   13  800    406.4  130  0  Open  ;
 15   13  14  500    406.4  130  0  Open  ;
 16   14  15  550    304.8  130  0  Open  ;
 17   16  15  2730   406.4  130  0  Open  ;
 18   16  17  1750   508    130  0  Open  ;
 19   17  18  800    609.6  130  0  Open  ;
 20   18  2   400    609.6  130  0  Open  ;
 21   2   19  2200   1016   130  0  Open  ;
 22   19  20  1500   508    130  0  Open  ;
 23   20  21  500    304.8  130  0  Open  ;
 24   19  22  2650   1016   130  0  Open  ;
 25   22  23  1230   762    130  0  Open  ;
 26   23  24  1300   762    130  0  Open  ;
 27   25  24  850    508    130  0  Open  ;
 28   26  25  300    304.8  130  0  Open  ;
 29   15  26  750    304.8  130  0  Open  ;
 30   22  27  1500   406.4  130  0  Open  ;
 31   27  28  2000   406.4  130  0  Open  ;
 32   28  29  1600   304.8  130  0  Open  ;
 33   29  30  150    304.8  130  0  Open  ;
 34   31  30  860    406.4  130  0  Open  ;
 35   24  31  950    508    130  0  Open  ;
 40   35  36  100    2000   100  0  Open  ;
 200  34  33  100    2000   130  0  Open  ;


[VALVES]
;ID Node1 Node2 Diameter  Type  Setting     MinorLoss
 36   33     32    1016      PRV  70          0 ;
 37   36     37    1016      PRV  80          0 ;

[OPTIONS]
 Units               LPS
 Headloss            H-W
 Specific Gravity    1
 Viscosity           1
 Trials              40
 Accuracy            0.000001
 CHECKFREQ           2
 MAXCHECK            10
 DAMPLIMIT           0
 Unbalanced          Continue 10
 Pattern             1
 Demand Multiplier   1.0
 Emitter Exponent    0.5
 Quality             None mg/L
```

```
 Diffusivity          1
 Tolerance            0.01

[COORDINATES]
;Node X-Coord Y-Coord
 1   5250.84 5585.28
 2   5227.80 5969.63
 3   5671.58 5975.42
 4   5928.37 6327.59
 5   6097.12 6613.73
 6   6243.86 6863.19
 7   6434.62 7244.70
 8   6610.70 7677.58
 9   6793.22 7990.65
 10  7315.05 7963.72
 11  7872.65 7941.71
 12  8283.52 7941.71
 13  6302.57 7990.65
 14  5683.41 7990.65
 15  5370.77 8000.40
 16  5290.06 7552.85
 17  5246.04 7119.98
 18  5227.80 6542.06
 19  4550.23 5969.63
 20  4006.11 5968.09
 21  3492.53 5968.09
 22  4563.71 6400.96
 23  4556.38 7031.93
 24  4629.74 7971.06
 25  4783.82 8000.40
 26  5025.94 8000.40
 27  3778.67 6422.97
 28  3888.72 7083.29
 29  3932.74 8469.96
 30  4116.16 8264.53
 31  4321.60 8059.10
 32  5250.84 5385.00
 33  5250.84 5185.00
 36  6793.22 8400.00
 37  6793.22 8200.00
 34  5250.84 4985.00
 35  6793.22 8600.00

[END]
```

Bibliography

[1] T. M. Walski, D. V. Chase, D. A. Savic, W. Grayman, S. Beckwith, and E. Koelle, *Advanced Water Distribution Modeling and Management*. Haestad press, 2003.

[2] A. Gupta and K. D. Kulat, "A selective literature review on leak management techniques for water distribution system," *Water Resources Management*, vol. 32, pp. 3247–3269, Aug 2018.

[3] OECD, *Water Governance in Cities*. 2016.

[4] Q. Xu, R. Liu, Q. Chen, and R. Li, "Review on water leakage control in distribution networks and the associated environmental benefits," *Journal of Environmental Sciences*, vol. 26, no. 5, pp. 955 – 961, 2014.

[5] T. T. T. Zan, H. B. Lim, K. Wong, A. J. Whittle, and B. Lee, "Event detection and localization in urban water distribution network," *IEEE Sensors Journal*, vol. 14, pp. 4134–4142, Dec 2014.

[6] M. Blanke, M. Kinnaert, J. Lunze, M. Staroswiecki, and J. Schröder, *Diagnosis and fault-tolerant control*, vol. 2. Springer, 2006.

[7] Z. Wu, "Innovative optimization model for water distribution leakage detection," *Bentley Methods Solution Center*, pp. 1–8, 09 2008.

[8] W. Li, W. Ling, S. Liu, J. Zhao, R. Liu, Q. Chen, Z. Qiang, and J. Qu, "Development of systems for detection, early warning, and control of pipeline leakage in drinking water distribution: A case study," *Journal of Environmental Sciences*, vol. 23, no. 11, pp. 1816 – 1822, 2011.

[9] R. Wright, E. Abraham, P. Parpas, and I. Stoianov, "Control of water distribution networks with dynamic DMA topology using strictly feasible sequential convex programming," *Water Resources Research*, vol. 51, no. 12, pp. 9925–9941, 2015.

[10] R. Puust, Z. Kapelan, D. A. Savic, and T. Koppel, "A review of methods for leakage management in pipe networks," *Urban Water Journal*, vol. 7, no. 1, pp. 25–45, 2010.

[11] E. Todini and S. Pilati, "A gradient algorithm for the analysis of pipe networks," in *Computer applications in water supply: vol. 1—systems analysis and simulation*, pp. 1–20, Research Studies Press Ltd., 1988.

[12] A. Rosich, V. Puig, and M. V. Casillas, "Leak localization in drinking water distribution networks using structured residuals," *International Journal of Adaptive Control and Signal Processing*, vol. 29, pp. 991–1007, 8 2015.

[13] T. A. N. Heirung and A. Mesbah, "Input design for active fault diagnosis," *Annual Reviews in Control*, vol. 47, 03 2019.

[14] A. Soldevila, R. M. Fernandez-Canti, J. Blesa, S. Tornil-Sin, and V. Puig, "Leak localization in water distribution networks using Bayesian classifiers," *Journal of Process Control*, vol. 55, pp. 1–9, 2017.

[15] D. Wachla, P. Przystalka, and W. Moczulski, "A method of leakage location in water distribution networks using artificial neuro-fuzzy system," *IFAC-PapersOnLine*, vol. 48, no. 21, pp. 1216 – 1223, 2015.

[16] G. Romero-Tapia, M. Fuente, and V. Puig, "Leak localization in water distribution networks using fisher discriminant analysis," *IFAC-PapersOnLine*, vol. 51, no. 24, pp. 929 – 934, 2018.

[17] R. Gomes, A. S. Marques, and J. Sousa, "Estimation of the benefits yielded by pressure management in water distribution systems," *Urban Water Journal*, vol. 8, no. 2, pp. 65–77, 2011.

[18] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*. London: Chapman & Hall, 1986.

[19] L. W. Mays *et al.*, *Water distribution systems handbook*. McGraw-Hill,, 2000.

[20] B. B. Azevedo and T. A. Saurin, "Losses in water distribution systems: A complexity theory perspective," *Water Resources Management*, vol. 32, pp. 2919–2936, Jul 2018.

[21] H. Armand, I. Stoianov, and N. Graham, "Impact of network sectorisation on water quality management," *Journal of Hydroinformatics*, vol. 20, pp. 424–439, 10 2017.

[22] F. M. White, *Fluid Mechanics, in SI Units*. McGraw-Hill, New York, 2011.

[23] E. Yeung, D. R. Judi, and W. B. Daniel, "Contingency analysis of water distribution networks using quadratic sensitivity functions," pp. 228–233, May 2017.

[24] K. A. Klise, R. Murray, and T. Haxton, "An overview of the water network tool for resilience (WNTR).," tech. rep., Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2018.

[25] J. Schwaller and J. Van Zyl, "Modeling the pressure-leakage response of water distribution systems based on individual leak behavior," *Journal of Hydraulic Engineering*, vol. 141, no. 5, 2014.

[26] A. Mesbah, S. Streif, R. Findeisen, and R. D. Braatz, "Active Fault Diagnosis for Nonlinear Systems with Probabilistic Uncertainties," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 7079 – 7084, 2014.

[27] G. Buskes and A. van Rooij, *Topological Spaces*. Springer-Verlag New York, 1997.

[28] J. W. Brewer, "Kronecker products and matrix calculus in system theory," *IEEE Trans. Circuits Syst.*, vol. 25, no. 9, pp. 772–781, 1978.

[29] R. Perez, G. Sanz, V. Puig, J. Quevedo, M. A. C. Escofet, F. Nejjari, J. Meseguer, G. Cembrano, J. M. M. Tur, and R. Sarrate, "Leak localization in water networks: A model-based methodology using pressure sensors applied to a real network in barcelona [applications of control]," *IEEE Control Systems Magazine*, vol. 34, pp. 24–36, Aug 2014.

[30] E. Abraham and I. Stoianov, "Sparse null space algorithms for hydraulic analysis of large-scale water supply networks," *Journal of Hydraulic Engineering, ASCE*, vol. 99, p. 99, 10 2015.

[31] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[32] O. Fujiwara and D. B. Khang, "A two-phase decomposition method for optimal design of looped water distribution networks," *Water Resources Research*, vol. 26, no. 4, pp. 539–549, 1990.

[33] D. Hart, J. S. Rodriguez, J. Burkhardt, B. Borchers, C. Laird, R. Murray, K. Klise, and T. Haxton, "Quantifying hydraulic and water quality uncertainty to inform sampling of drinking water distribution systems," *Journal of Water Resources Planning and Management*, vol. 145, no. 1, p. 04018084, 2018.

[34] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[35] J. Thornton, R. Sturm, and G. Kunkel, *Water loss control manual*. McGraw-Hill New York, 2002.

[36] N. Trifunovic, *Introduction to Urban Water Distribution: Unesco-IHE Lecture Note Series*. UNESCO-IHE Lecture Note Series, Taylor & Francis, 2006.