


Universitatea Politehnica Bucureşti
Facultatea de Automatică si Calculatoare
Departamentul de Automatică şi Ingineria Sistemelor

LUCRARE DE LICENŢĂ

Clasificare defecte într-o rețea de apă de mari dimensiuni

Absolvent
Cazan Cristian-Claudiu

Coordonator
Conf. dr. ing. Florin Stoican

Bucureşti, 2018

Cuprins

Listă de figuri

Listă de tabele

Listă de algoritmi

1. Introducere

1.1. Motivația alegerii temei

Transportul și distribuția apei reprezintă una dintre cele mai vechi preocupări inginerești de proporții, existând de mai mult de 4000 de ani. Civilizația minoică, localizată în insula Creta, este considerată a fi prima care a construit apeducte - structuri pentru transportul apei de la sursă către orașe - în 2500 î.Hr.

Deși majoritatea popoarelor din antichitate care s-au ocupat cu construcția apeductelor întrebuințau aceste sisteme pentru irigația pământului - ocupațiile de bază de atunci fiind în strânsă legătură cu agricultura - romanii au văzut în sistemele de provizionare a apei și un potențial imens în dezvoltarea civilizației, astfel ei sunt ei care aduc cele mai mari contribuții inginerești, apeductele construite de aceștia impresionând și astăzi prin grandoarea și iscusința cu care au fost construite.

Inovațiile în acest domeniu au suferit salturi bruște și puternice în momentul descoperirii unei noi relații matematice care transformă un parametru despre care se puteau face doar niște estimări grosiere într-o mărime bine definită și bine controlată. Istoric vorbind incipitul dezvoltării științei hidraulice s-a bazat pe relația descoperită de Arhimede din Siracuza în sec III î.Hr. $F = V_{obiect} * \rho_{lichid} * g$. O altă contribuție care are o deosebită importanță în domeniul tehnologiei de distribuiție a apei și nu numai o reprezintă tubul lui Pitot folosit la măsurarea vitezei fluidului, inventat de Henri Pitot în sec XVII. Din punct de vedere constructiv tubul are o formă de **L**, scufundarea acestuia într-un fluid (apă sau gaz) va determina creșterea nivelului și a presiunii până la o anumită limită ;[4], ecuația care guvernează depedența nivel - viteză este:

$$u = \sqrt{\frac{2(p_t - p_s)}{\rho}} \tag{1.1}$$

unde:

- u reprezintă viteza fluidului
- p_t reprezintă presiunea de stagnare
- p_s reprezintă presiunea statică
- ρ reprezintă densitatea fluidului

Mergând mai departe, alte contribuții importante apar din partea marilor matematicieni precum Daniel Bernoulli și Leonhard Euler, care au mărit spectrul mecanicii lui Newton și Leibniz spre aria hidraulicii și a termodinamicii. Fluidele considerate sunt incompresibile și au densitatea constantă în timp și uniform distribuită în spațiu. Bernoulli afirmă despre lichidele incompresibile că o creștere în viteză a lichidului este însoțită de o scădere a energiei potențiale a lichidului (i.e. presiune):

$$\frac{v^2}{2} + gz + \frac{p}{\rho} = c \tag{1.2}$$

unde:

- v reprezintă viteza fluidului
- g reprezintă accelerația la care e supus fluidul
- z reprezintă elevația ștragulației conductei
- p reprezintă presiunea într-un anumit punct
- ρ reprezintă densitatea fluidului

În contextul în care se dorește analiza unui caz real este important ca toate diferențele între cazurile ideale și cazurile reale trebuie puse în evidență în mod matematic, astfel se particularizează ecuația generală Navier-Stokes pentru cazuri în care se cunosc anumiți parametrii ai sistemului de analizat. Spre exemplu ecuația Poisuille care modelează începutul fluxului de apă într-o conductă este ;[1]:

$$\frac{\partial u}{\partial t} = \frac{G}{\rho} + \nu \left(\frac{\partial^2 u}{\partial t^2} + \frac{1}{r} \frac{\partial u}{\partial r} \right) \tag{1.3}$$

unde:

- u reprezintă viteza lichidului prin conductă
- t reprezintă timpul
- G reprezintă diferența de presiune
- ρ reprezintă densitatea lichidului
- ν reprezintă vâscozitatea cinematică
- r reprezintă poziția

Se poate observa că pe măsură ce modelul matematic se apropie de realitate, complexitatea acestuia crește și pentru fiecare situație specială - spre exemplu analiza presiunii la introducerea apei într-o conductă vs. analiza presiunii când conducta este încărcată cu apă - are nevoie de o ecuație specială sau de o particularizare a ecuației Navier-Stokes, pentru care încă nu se conoaște dacă există soluții pentru cazul cu 3 dimensiuni și dacă soluțiile acestea sunt netede.

Ținând cont de importanța apei în desfășurarea activităților cotidiene atât pentru oameni cât și pentru actorii importanți ai industriei, este o condiție sine-qua-non ca un oraș să aibă un sistem performant și rezistent la defecte pentru distribuția apei. În contextul actual al dezvoltării tehnologiei este natural să folosim tehnici moderne de monitorizare a diferiților parametrii din cadrul unei rețele pentru a putea face o analiză riguroasă și eficientă cu referire nu numai la mentenanță ci și la consumul global și local în ideea îmbunătățirii și reducerii pierderilor.

1.2. Expunerea problemei

În această lucrare se va aborda problematica identificării prezenței unui defect - *Fault detection* și izolarea defectului *Fault isolation* într-o regiune a rețelei.

O rețea de apă poate fi privită ca un graf neorientat $G = (V, E)$ unde V este mulțimea nodurilor rețelei - acestea reprezentând o abstractizare asupra componentelor precum:

- rezervoare

- tancuri de apă
- puncte de distribuție

E este mulțimea muchiilor reprezentând de fapt țevile care fac legătura între noduri.

Mergând mai departe cu abstractizarea se pot considera rețele de apă active și rețele de apă pasive. Diferența între cele două făcându-se în baza pompelor de apă amplasate în zonele unde presiunea sau elevația vin în detrimentul distribuției apei.

Rețelele de apă care vor fi tratate în această lucrare fac parte din categoria pasivă, astfel putem diviza mulțimea nodurilor V în V^t și în V^j reprezentând mulțimea nodurilor de tip tanc și mulțimea nodurilor joncțiune, cu proprietatea că $V = V^t \cup V^j$. Tancurile și rezervoarele dintr-o rețea de apă au proprietatea că nivelul de apă din acestea se va menține la un nivel oarecum staționar, astfel simulările din capitolele viitoare se vor axa pe nodurile simple de tip joncțiune, deci mulțimea de interes în acest caz va fi V^j pentru care cunoaștem cardinalul.

Caracteristicile care se pot recolta dintr-o rețea de apă pot varia în funcție de elementul inspectat și de senzorii dispuși în rețea, astfel pentru fiecare nod $n_i \in V^j$ putem defini la fiecare moment de timp

- presiunea $p_i(t)$ - măsurată în metri coloană de apă $mH2O$, mărime influențată puternic de presiunea interioară a nodului și de eventualele perturbații exterioare i.e. scurgeri de apă prin țevi
- 'cererea' $d_i(t)$ - măsurată L/s, mărime ce caracterizează profilul de utilizare al utilizatorilor de-a lungul unei zile i.e. debitul de apă care ajunge la consumatori. Acest debit poate varia de-a lungul zilei, putem distinge de exemplu intervale de timp în care cererea este foarte mică și rețeaua intră în regim staționar

De asemenea pentru fiecare conductă a rețelei $e_{ij} \in E$ putem măsura viteza lichidului $v_{ij}(t)$.

Pentru a putea rezolva problema de *Fault Detection and Isolation* este importantă găsirea unei modalități eficente de selecție și prelucrare a datelor de la rețea.Mai mult, punând în lumină aspectul ingineresc al problemei, trebuie găsită o submulțime $V_{opt} \subset V^j$ ai cărei elemente pot aduce informații necesare și suficiente pentru a detecta un defect într-o acoperire destul de mare a rețelei.

1.3. Exemplul de lucru

În următoarele capitole și în implementarea lucrării consider rețeaua din Hanoi iar pentru simularea scenariilor propuse voi folosi biblioteca și suita de funcții **EPANET** - Environmental Protection Agency NETwork

Rețeaua Hanoi constă într-o mulțime de noduri de tip joncțiune V^j cu $|V^j| = 31$ și mulțimea V^t cu $|V^t| = 1$, ilustrată în figura de mai jos:

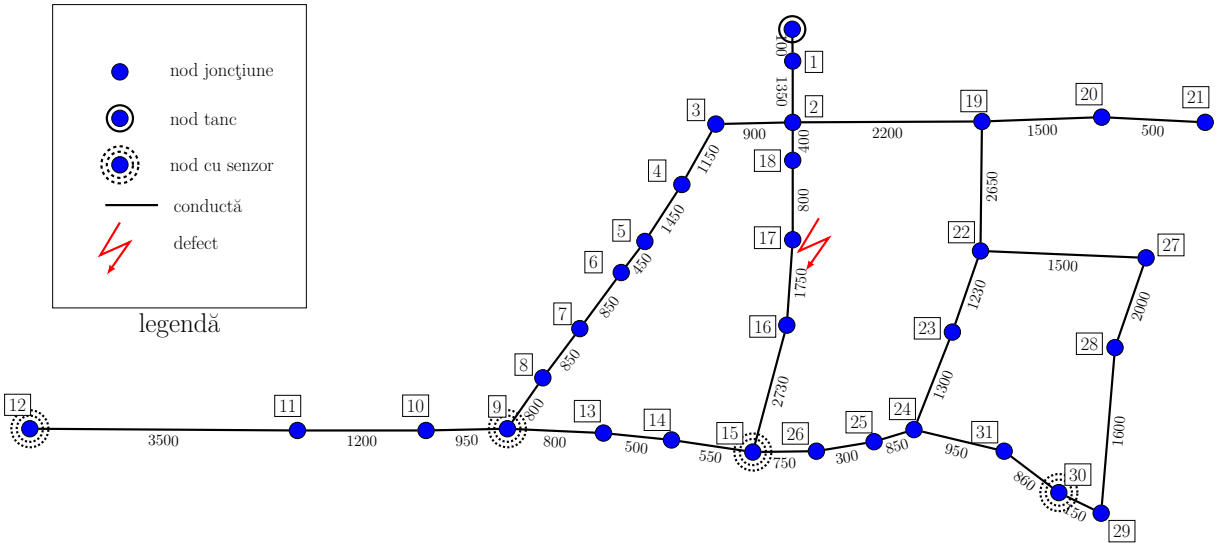


Figura 1.1.: Graful rețelei de apă din Hanoi

După cum se poate observa în figura de mai sus au fost reprezentate două tipuri de node pasive, anume tancurile și joncțiunile. În cazul apariției unui defect în rețea, este important de luat în considerație modalitatea în care acesta va influența dinamica rețelei, spre exemplu este de la sine înțeles că dacă se cosideră un defect în nodul cu indicele 17 - i.e. în acest nod au apărut anumite scurgeri care afectează fluxul de apă către consumatori - nodurile în care se va observa o modificare puternică a caracteristicilor (presiune și debit) vor face parte din mulțimea nodurilor adiacente rețelei $S = V_{16}, V_{18}$, deși pare o concluzie naturală, o modelare matematică riguroasă din care să se tragă această nu este o problemă foarte ușor de rezolvat, anumiți parametrii fiind extrem de greu de estimat chiar și în cazul în care se consideră un regim staționar.

2. Simulări și software folosit

2.1. Dificultatea simulării unei rețele de apă

Găsirea unui set de ecuații al cărei soluție să conducă la o estimare îndeajuns de bună pentru control este o condiție sine qua non pentru detecția unui defect și izolarea acestuia în cadrul nodurilor rețelei. Astfel după cum a fost expus în capitolul 1 ecuațiile care guvernează relațiile intre viteza prin conducte și presiune dintr-un anumit punct sunt particularizări ale ecuațiilor Bernoulli-Euler sau Navier-Stokes. În cadrul unei rețele de apă a unui oraș, complexitatea rezolvării problemei crește semnificativ din varii motive precum:

- ansamblul de coduncte și noduri interconectate dă naștere unui sistem fizic greu de modelat matematic
- parametrii care pot influența calitatea soluțiilor precum: tipul materialului conductei și al nodului, elevația fiecărui nod, rugozitatea fiecărei conducte și depunerile de pe aceasta
- apariția unor factori exogeni care pot fi uneori greu de estimat - tiparul de utilizare al rețelei de către consumatori poate varia puternic
- apariția defectelor precum scurgerile în proximitatea unui nod

Ținând cont de complexitatea problemei în regim dinamic pentru a putea obține o soluție de regim staționar a rețelei este necesar să ignorăm evenimentele imprevizibile precum apariția unei scurgeri sau variațiile bruște ale consumului.

Ecuațiile de regim staționar includ condiții de conservare fluxului de apă:

$$\sum_{j=1}^{n} \mathbf{B}_{ij}\mathbf{q}_j = \mathbf{d}_i \tag{2.1}$$

Unde q_i reprezintă debitul prin fiecare conductă iar $\mathbf{B}$ reprezintă matricea de adiacență a rețelei la echilibru, definită astfel:

$$\mathbf{B}_{ij} = \begin{cases} 1, & \text{conducta j intră în nodul i} \\ 0, & \text{conducta j nu este conectată la nodul i} \\ -1, & \text{conducta j iese din nodul i} \end{cases} \tag{2.2}$$

Partea de estimare a diferenței de presiuni (în engl. "Head-Flow differential") între două noduri interconectate se face utilizând formula Hazen-Williams ;[8]:

$$\mathbf{h}_i - \mathbf{h}_j = \frac{10.67 \cdot L_\ell}{C_\ell^{1.852} \cdot D_\ell^{4.87}} \cdot \mathbf{q}_\ell \cdot |\mathbf{q}_\ell|^{0.852} \tag{2.3}$$

unde:

- **h** reprezintă presiunea - măsurată de obicei în metru coloană de apă
- C_l reprezintă coeficientul de rugozitate al conductei
- D_l reprezintă diametrul conductei
- L_l reprezintă lungimea conductei
- q_l reprezintă debitul

Din ecuația empirică (2.1) termenul $R_{ij} = \frac{10.67 \cdot L_\ell}{C_\ell^{1.852} \cdot D_\ell^{4.87}}$ reprezintă rezistența conductei ij iar dual, putem obține conductivitatea conductei $G_{ij} = \frac{1}{R_{ij}}$.

Având la dispoziție (2.1) și (2.3) putem exprima dependența debit presiune în regim staționar sub o formă matriceală compactă și cu o structură neliniară:

$$\mathbf{BG}\left[\left(-\mathbf{B}^\top \mathbf{h} + \mathbf{B}_f^\top \mathbf{h}_f\right) \times \left|-\mathbf{B}^\top \mathbf{h} + \mathbf{B}_f^\top \mathbf{h}_f\right|^{-0.46}\right] = \mathbf{d} \tag{2.4}$$

unde s-au luat în calcul și nodurile care au variații de presiune foarte mici - spre exemplu nodurile de tip tanc și nodurile de tip rezervor - termenul $\mathbf{B}_f^\top \mathbf{h}_f$ reprezintă contribuția acestor noduri la starea de echilibru a rețelei.

Parametrul **d** reprezintă cosumul pentru noduri (L/s) și reprezintă un factor de incertitudine pentru întregul model. În cadrul unei simulări acesta poate fi estimat sau ales empiric, dar nu poate fi cunoscut cu precizie în fiecare moment. Problema aflării vectorului de consum a nodurilor intră în categoria problemelor legate de serii de timp și a estimării valorilor viitoare. Simulatoarele software transpun în general o variație a parametrilor de rețea (emitter-demand) în modificarea acestui parametru d.

Din cauza dificultății rezolvării unei ecuații matriceale neliniare, software-ul specializat trebuie să folosească diferite metode de optimizare (”Solver”) pentru a putea obține o diferență cât mai mică între cazul estimat și rezultatul real al ecuației. Este important de reținut faptul că rezolvarea problemelor de programare neliniară cu constrângeri poate generea de fapt o problemă NP-completă, sau în unele cazuri chiar NP-dură ;[3].

2.2. Simulări folosind biblioteca EPANET

Dezvoltat la începutul anilor 90’ de către USEPA (United States Environmental Protection Agency), EPANET a fost inițial privit ca un instrument pentru cercetare, acesta a devenit un standard de industrie la capitolul simulărilor software robuste pentru rețele de apă, foarte multe pachete software proprietare de simulare hidraulică se bazează masiv pe EPANET, diferențele apărând la design-ul interfeței grafice și a manipulării datelor. Aceast program de simulare oferă utilizatorului posibilitatea de a-și defini într-un mod interactiv o rețea de apă configurând tipul de nod, legătura între oricare două noduri și posibilitatea de a adăuga și elemente active în rețea, pompe. Pentru a simula utilizatorul trebuie să își definească pentru fiecare nod un anumit debit cerut de utilizatori, o elevație, și o legătură cu alte noduri. Simularea se va desfășura pe o perioadă de timp definită cu pasul de eșantionare cât mai convenabil ;[7].

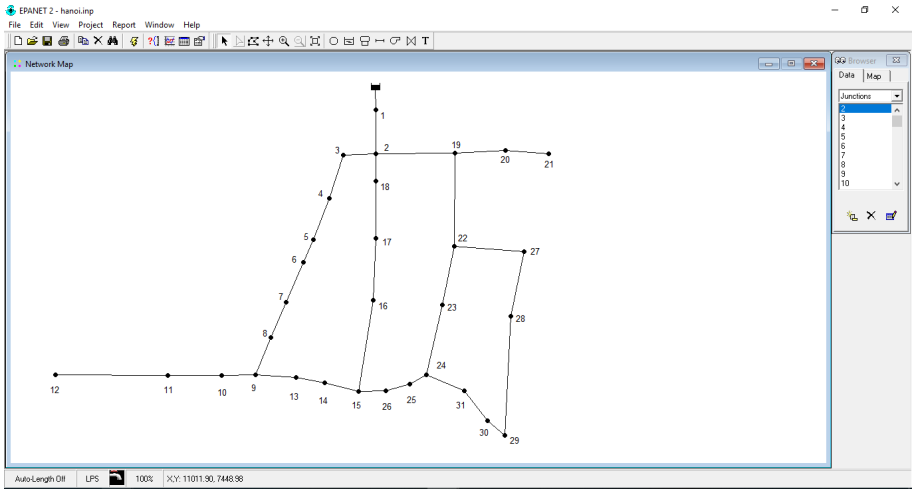


Figura 2.1.: Simulatorul EPANET

În urma execuției simulării EPANET va stoca toate datele în memorie și va putea reliza grafice și alte interogări complexe.

Modalitatea în care simulatorul EPANET reușește să obțină datele de simulare este prin implementarea eficientă a ecuațiilor Hazen-Williams (2.1) Darcy-Weisbach și Chezy-Manning, la fiecare perioadă de eșantionare algoritmul bazat de metoda gradientului rezolvă ecuațiile matriceale neliniare.

Pe lângă posibilitatea de a simula în cadrul unui program de sine stătător toate scenariile dorite, USEPA pune la dispoziție și un API (Application Programming Interface) pentru a putea realiza programatic simulările și modificările aferente fiecărui scenariu. Folosind în acest fel interfața pusă la dispoziție scrisă în limbajul C și oferită sub forma unei biblioteci dinamice (în Windows fișier .dll, în Linux fișier .so) programatorul are posibilitatea de a realiza propriul software specializat pentru simularea rețelelor de apă.

Particularizările programatice includ Demand-ul (debitul de ieșire din nod către utilizatori L/s) fiecărui nod la orice moment de timp, proporționalitatea Demand-ului pentru a putea emula modul în care rețeaua este folosită de utilizatori de-a unei zile de lucru și unul din aspectele importante pe care EPANET le pune la dispoziție programatorilor este posiblitatea de a simula o scurgere în rețea emulată prin intermediul unui Emitter. Emitterul din biblioteca epanet este modelat ca un orificiu (perforație) prin care se poate scurge apa, fie din motive de a elibera presiunea sau din cauza unui defect. Ecuația care guvernează scurgerea prin acest orificu este:

$$q = Cp^{\gamma} \tag{2.5}$$

unde:

- q reprezintă debitul prin emitter
- C reprezintă o constantă de proporționalitate
- p reprezintă presiunea din joncțiune
- γ reprzintă exponentul de presiune

Astfel apariția unui defect într-un anumit nod determină apriția unei relații liniare în funcție de presiunea din nod. Coeficientul γ definde de tipul de joncțiune și de dimensiunea orificiului prin care se va scurge apa, pentru orificii relativ mici acesta ia valori de aproximativ 0.5 ;[7].

2.2.1. Schema bibliotecii ANSI C - EPANET

API-ul pus la dispoziție se bazează pe funcții C scrise într-o manieră modularizată, în ideea de a separa partea de analiză a rețelei și de modificare a parametrilor de partea de simulare hidraulică și calitativă. Astfel în imaginea de mai jos este prezentată modalitatea în care datele ajung și sunt folosite în cadrul simulatorului:

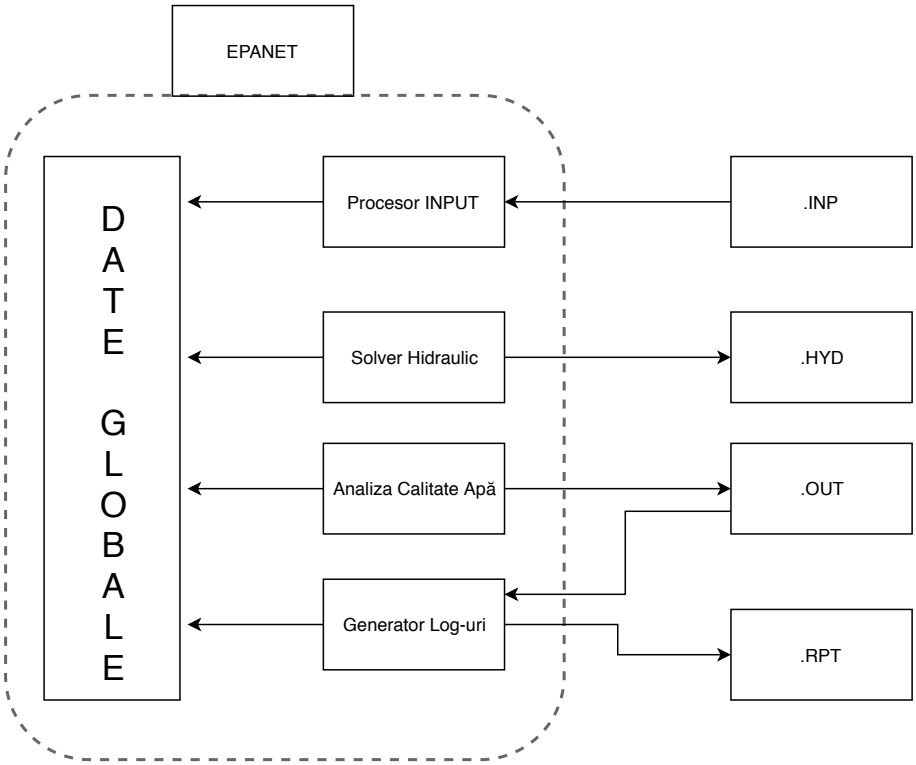


Figura 2.2.: Fluxul de date în biblioteca EPANET

Fișierele prezentate în schema de mai sus în partea dreaptă au semnificația:

- .INP - fișierul în care sunt definite topologia rețelei, tiparele de utilizare, caracteristicile elementelor active, valorile de Emitter și reguli complexe pentru funcționarea conductelor și pompelor
- .HYD - fișerul în care sunt stocate rezultatele simulării hidraulice
- .OUT - fișierul în care sunt stocate datele analizei calitative a apei
- .RPT - jurnalul în care biblioteca trece toate mesajele de eroare și atenționările cu privire la rețea

2.2.2. Structura fișierului de intrare .INP

Ceea ce stă la baza unei simulări este definirea input-ului rețelei de apă, anume fișierul INP. Fișierul INP - este un fișier text care conține anumite directive referitoare la componentele rețelei, tabelul de mai jos descrie toate aceste cuvintele cheie recunoscute de interpretor, împărțite în 4 categorii funcționale:

- Componente - se definesc toate componentele fizice ale sistemului noduri, conducte, valve, emitters

- Simulare - se definesc caracteristicile care trebuie urmate de solver în procesul de simulare
- Calitate - se definesc parametrii relevanți analizei de apă - Cantitatea de substanță dizolvabilă din fiecare nod, constantele de timp pentru reacțiile de ordin I și II
- Jurnalizare - secțiune responsabilă cu fixarea parametrilor de execuție a rețelei - pasul de eșantionare, unitățile de măsură, parametrii impliciți și modul de raportare a simulării

Componente	Simulare	Calitate	Jurnalizare
[TITLE]	[CURVES]	[QUALITY]	[OPTIONS]
[JUNCTIONS]	[PATTERNS]	[REACTIONS]	[TIMES]
[RESERVOIRS]	[ENERGY]	[SOURCES]	[REPORT]
[TANKS]	[STATUS]	[MIXING]	
[PIPES]	[CONTROLS]		
[PUMPS]	[RULES]		
[VALVES]	[DEMANDS]		
[EMITTERS]			

Tabela 2.1.: Structura fișierului INP

Cele mai importante directive au fost trecute în tabelul 2.1 în culoarea albastru. Urmând ca fiecare să fie explicată mai în detaliu.

[JUNCTIONS]
Reprezintă structurile care definesc obiectele de tip nod din rețea, acestea vor avea ca atribute setate:

- ID - indicele nodului
- Elevation - înălțimea nodului (m)
- Base demand - debitul inițial către utilizatori (s-a folosit termenul demand pentru a desemna cerința de apă din rețea)
- Demand pattern ID - pentru fiecare nod se poate fixa un anumit profil de utilizare prin legarea acestuia cu un anumit obiect PATTERN

[PIPES]
Pipes reprezintă structurile care modelează conductele din rețea, pentru care atributele sunt:

- ID - indexul fiecărei conducte
- Start node ID - indexul nodului de plecare
- End node ID - indexul nodului destinație
- Length - lungimea conductei (m)
- Diameter - diametrul conductei (mm)
- Roughness coef - coeficientul de rugozitate al conductei
- Minor loss coef - coeficientul de pierdere al conductei
- Status - conducta poate fi în 3 stări , deschisă - OPEN, închisă - CLOSED sau polarizată - CV (lasă să treacă fluxul de apă într-o singură direcție)

[EMITTERS]
În lucrarea de față folosesc EMITTERS ca modalitate de a modela o scurgere de apă într-un anumit nod. Caracteristicile structurii sunt:

- Junction ID - indexul nodului unde se află scurgerea
- Flow coeff - coeficientul C din ecuația (2.5)

[PATTERNS]
Prin intermediul structurilor PATTERN se definesc profilurile de utilizare ale rețelei. Modalitatea de funcționare a demand-ului este prin multiplicarea valorii PATTERN cu valoarea de Base Demand a fiecărui nod. Astfel motorul de simulare va diviza întreaga perioadă de simulare astfel încât fiecare multiplicitate să primească un interval de timp egal.

[DEMANDS]
Structură care definește gradul de utilizare pentru fiecare nod din rețea. Atributele care definesc utilizarea sunt:

- Junction ID - indexul nodului unde se dorește modificarea debitului de utilizare
- Base Demand - magnitudinea debitului de utilizare

2.3. Integrarea EPANET cu Python

Pentru a ușura mecanismul de procesare și interpretare am decis să folosesc funcțiile de bibliotecă EPANET adaptate din C la limbajul Python de proiectul open-source [6]. Motivele pentru care am ales limbajul Python sunt rapiditatea dezvoltării aplicației, robustețea soluției și ușurința tratării erorilor. Python este un limbaj interpretat - deci tipurile variabilelor sunt decise dinamic în momentul rulării aplicației în schimb acesta nu permite conversia implicită a variabilelor la run-time, eliminând astfel foarte multe erori greu de depistat.

Prin utilizarea unei clase de adaptare, A.1, la API-ul de Python pus la dispoziție de [6] doresc să realizez o interfață mai ușoară pentru definirea, rularea simulărilor dar și pentru salvarea datelor într-un format care permite operabilitatea între mai multe programe, de exemplu Python-Matlab. Structura clasei de adaptare numită în engleză Wrapper (i.e. ambalaj al bibliotecii de funcții scrise în C) se bazează pe instanțierea unui obiect prin care se pot apela diferite funcții din biblioteca dinamică astfel încât să se obțină datele necesare. Metoda clasei prin care se face interogarea rețelei este:

```
1    def query_network(self, sim_dict):
```

Listing 2.1: Funcția de query

Parametrul *sim_dict* primit de funcția 2.1 reprezintă un fișier JSON (Java Script Object Notation) care are o structură ce permite interogarea rețelei în legătură cu diferite mărimi dorite.

```
1        {
             simulation_name : "name",
3            simulation_type: "H" or "Q"
             emitter_values : [ (node_index, emitter_value) ]
5            query : {
                 nodes : [ "EN_PRESSURE"
7                ]
                 links : [ "EN_VELOCITY"
```

```
                ]
            }
        }
```

Listing 2.2: Structură JSON intrare

Unde câmpurile reprezintă:

- simulation_name - numele simulării
- simulation_type - tipul simulării H - hidraulic, Q - calitate a apei
- emitter_values - un vector de cupluri (node_index, emitter_value), node_index reprezintă indexul joncțiunii iar emitter_value reprezintă magnitudinea scurgerii din acel nod
- query - reprezintă valorile care se doresc a fi interogate pentru fiecare componentă:
  - EN_PRESSURE - întoarcerea valorilor presiunilor pentru fiecare nod
  - EN_DEMAND - întoarcerea valorilor demand-ului pentru fiecare nod
  - EN_VELOCITY - întoarcerea valorilor vitezei apei prin fiecare conductă

La apelarea funcției query_network se va returna de asemenea un dicționar, care poate fi convertit la format-ul JSON, pentru a putea fi folosit în alte procese de simulare și validare, spre exemplu într-un mediu de dezvoltare Matlab/Octave. Structura fișierului de ieșire este:

```
{
    SIM_NAME = simulation-name
    NODE_VALUES = [
        {
            "EN_PRESSURE" : [ [values for each node]]
            "EMITTER_VAL" :
            "EMITTER_NODE" :
        }
    ]

}
```

Listing 2.3: Structură JSON ieșire

Câmpul NODE_VALUES reprezintă un vector JSON-uri, fiecare element al acestui vector fiind în fapt rezultatul numeric al unei simulări pentru câmpurile specificate la fișierul de intrare 2.2.

Un pas important în dezvoltarea Wrapper-ului peste EPANET a reprezentat tratarea erorilor returnate de funcțiile de C. În momentul apelării unei funcții de C, EPANET returnează o pereche (a, b) unde a reprezintă valoarea efectivă interogată iar b reprezintă codul erorii. Tratarea excepțiilor s-a făcut cu ajutorul funcției:

```
    @staticmethod
    def _getncheck(ret_val):

        # check the return code
        if isinstance(ret_val, list):
            if ret_val[0] == 0:
                # everything OK
                return ret_val[1]
            else:
                err_msg = ENgeterror(ret_val[0], 100)
                raise EpanetError(err_msg)
        else:
            if ret_val is not 0:
```

```
14                err_msg = ENgeterror(ret_val, 100)
                  raise EpanetError(err_msg)
```

Listing 2.4: Cod pentru tratarea erorilor

și prin definirea unei excepții customizate pentru clasa ENWrapper scrisă

```
class EpanetError(Exception):
2
    def __init__(self, err_msg):
4       super().__init__(err_msg)
```

Listing 2.5: Definirea Excepției

2.4. Simulări pe rețeaua de apă din Hanoi

Având ca susținere fișierul pentru rețeaua de apă din Hanoi 1.1 și biblioteca EPANET cu codul de adaptare Python, pot realiza simulări asupra acestei rețele de apă. Inițial voi explicita profilurile de utilizare a rețelei de apă urmând ca mai apoi să arăt modul în care rețeaua funcționează în mod nominal (fără nici un defect).

Parametrii funcționali rețelei hanoi.inp sunt următorii:

- pasul de eșantionare al simulării 15min
- pasul de eșantionare al pattern-ului 15min
- Perioada de simulare este de 24h

Graficul de utilizare al rețelei are o distribuție centrată în jurul orelor dimineții - acest lucru fiind corelat cu faptul că atunci este momentul în care cei mai mulți utilizatori își încep activitatea și consumă apă. Astfel rețeaua este supusă unui stres mai mare în timpul orel matinale față de exemplu de perioada nopții unde valorile multiplicității ajung la valori aproximativ de două ori mai mici.

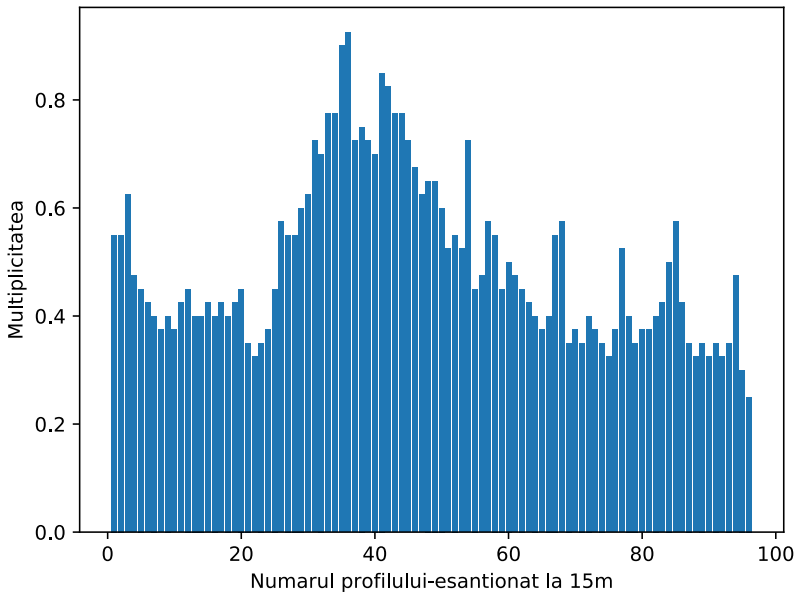


Figura 2.3.: Profilul normal de utilizare

Pentru rularea unei simulări nominale fișierul JSON de intrare poate avea câmpul de emitter_values gol, obținându-se asftel caracteristica tranzitorie a presiunii, a debitului din noduri și a vitezei fluxului de apă din conducte.

2.5. Prezentarea măsurilor de interes preluate din simulări

Problema care se pune în continuare alegera unui subset din mărimile prelevate din simulator astfel încât discernerea între un caz de utilizare normal și unul defectuos să fie cât mai ușoară.

2.5.1. Simulare dinamică pentru încărcare nominală

În regim nominal ansamblul rețelei de apă nu este supus la nici un defect, funcționarea fiind corespunzătoare profilului de utilizare normal 2.3. Datele care sunt extrase din simularea nominală și prezintă interes pentru analiză sunt corespunzătoare presiunii din fiecare nod. De asemenea vor fi prezentate și celelalte măsurători - viteza apei prin conducte și debitul către utilizatori.

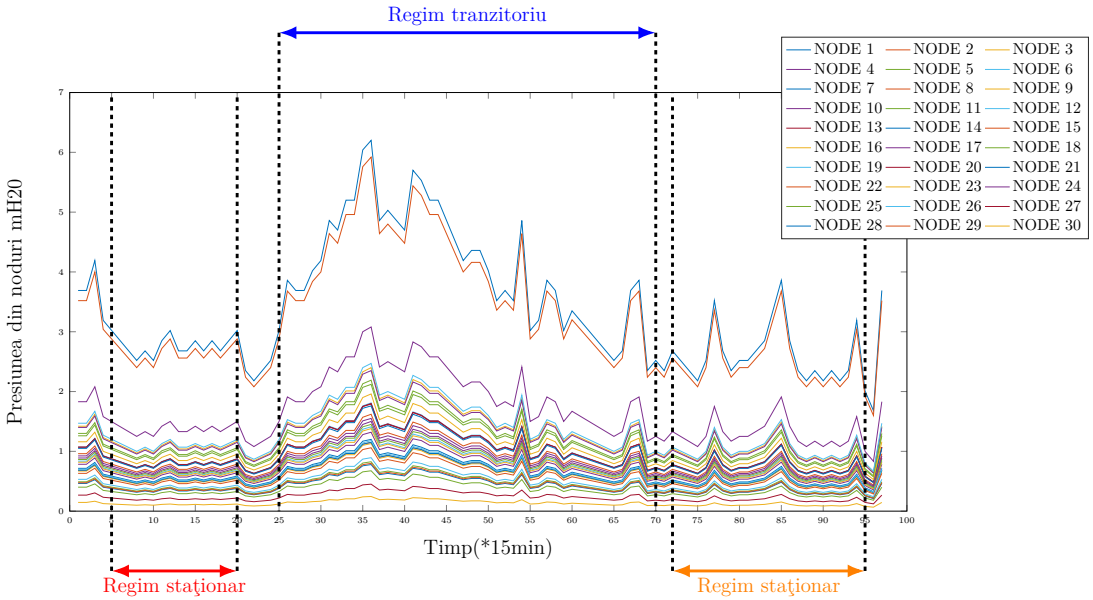
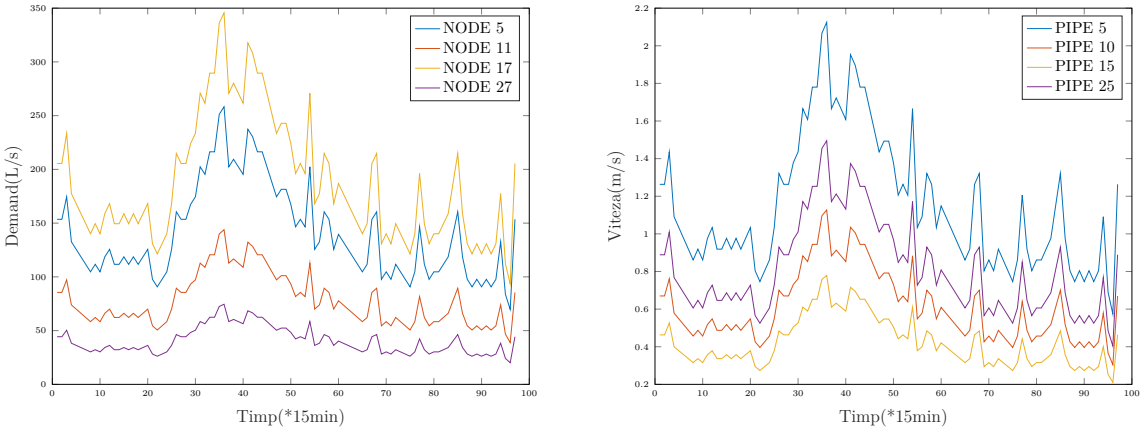


Figura 2.4.: Profilul nominal al presiunii

După cum se poate observa în 2.4 profilul nominal al presiunii intră în tiparul profilului de cerere prezentat în 2.3, unde în jurul orelor dimineții, când activitatea utilizatorilor este asociată unui regim dinamic al rețelei, iar orele nopții sunt caracterizate de marimi lent variabile. Astfel pe acest grafic s-au putut evidenția 3 zone importante, anume, cele două zone de regim staționar în intervalele $[5, 20] * 15min$ și $[72, 95] * 15min$ și zona de regim dinamic în intervalul $[23, 70] * 15min$. Este de menționat că aceste intervale au fost selectate empiric în baza unei analize grafice asupra caracteristicilor presiune-viteză-debit.

2.5.2. Alte mărimi de interes din simulatorul EPANET

Caracteristicile de viteză și debit urmează o caracteristică asemănătoare cu caracteristica de presiune din fiecare nod, o explicație naturală provenind de la faptul că presiunea este considerată ca fiind înălțimea nivelului de apă dintr-un nod.



(a) Profil nominal demand

(b) Profil nominal viteza

Figura 2.5.: Rezultate simulări nominale

După cum se observă în 2.5 zonele de regim staționar și de regim dinamic se păstrează în continuare și pentru caracteristicile de viteză și debit. De asemenea este important de menționat că profilurile nominale ale celoralte noduri decât cele ilustrate în figurile de mai sus urmează aceeași caracteristică dinamică, alegerea nodurilor și conductelor respective a fost făcută pentru a păstra figura curată.

3. Detecția și izolarea defectelor

3.1. Definirea defectelor

Defectele sunt simulate modificând parametrul C din ecuația emitter-ului (2.5). Modalitatea prin care se execută în cod simularea unui defect este prin apelarea metodei:

```
    def set_emitter(self, node_index, emitter_val):
2       if self.network.nodes[node_index].node_type is EN_JUNCTION:
            ENSim._getncheck(self.ENsetnodevalue(node_index, EN_EMITTER, emitter_val
    ))
```

Listing 3.1: Funcție pentru simularea defectelor

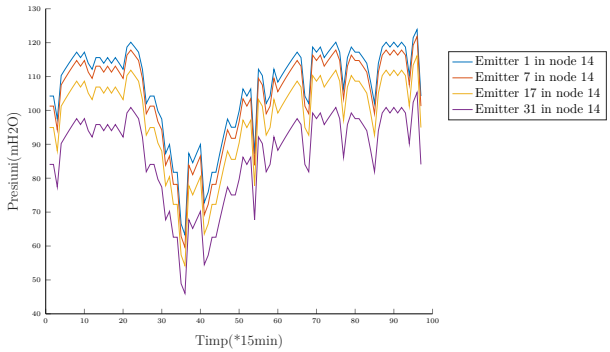
parametrii funcției *set_emitter* sunt:

- node_inde - indexul nodului în care se simulează defectul
- emitter_val - magnitudinea defectului

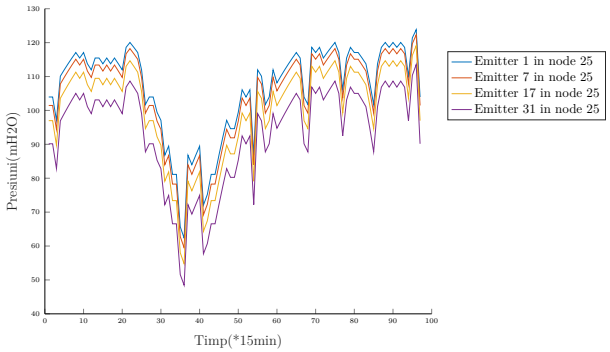
Metoda mai întâi vefică dacă nodul cu indexul $node_index$ reprezintă doar o jonncțiune apoi setează magnitudinea defectului în nodul primit cu ajutorul funcției de bibliotecă **ENsetnodevalue**

3.2. Simulare dinamică pentru defecte în diferite noduri

În continuare vom considera un scenariu de defect pentru rețea care constă în modificarea succesivă a parametrului de proporționalitate din relația de calcul a debitului de emitter (2.5). În imaginile următoare voi considera mai multe magnitudini de defect într-un anumit nod și voi reprezenta grafic răspunsul în timp al rețelei în același nod.



(a) Profile cu defect în nodul 14



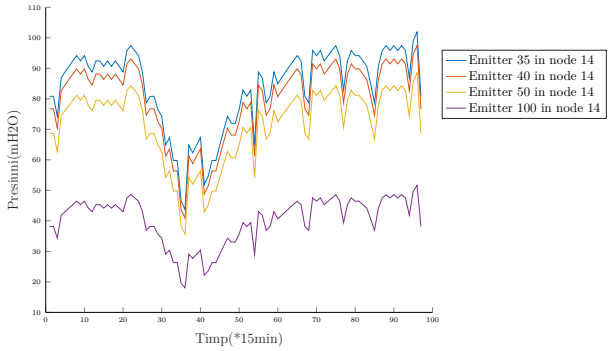
(b) Profil cu defect în nodul 25

Figura 3.1.: Rezultate simulări defecte ușoare

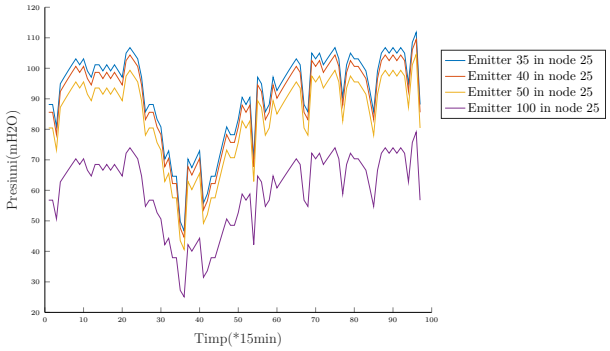
După cum se poate observa în imaginile 3.1 variația emitter-ului într-un nod produce în mod evident o modificarea a modului comun al caracteristicii $timp - presiune$. Din punctul de vedere al magnitudinolor de simulare pentru defecte, am considerat 2 clase de defecte, anume:

- defecte ușoare (soft faults) - cu valorile coeficientului de emitter mai mici de 35
- defecte puternice (hard faults) - cu valorile emitter mai mari de 35

Cele din urmă produc și modificări ale caracteristicii dinamice, introducând distorsiuni sau aplatizări ale mărimilor măsurate. Reprezentarea defectelor hard este reprezentată în figurile de mai jos:



(a) Profile cu defect puternic în nodul 14



(b) Profil cu defect puternic în nodul 25

Figura 3.2.: Rezultate simulări defecte puternice

Se observă de exemplu că pentru o valoare a emitter-ului de 100 caracteristica dinamică este deja modificată din cauza scurgerilor puternice din nod.

Este relevantă împărțirea defectelor în mai multe clase de magnitudini pentru a putea valida un model de clasificare. Spre exemplu este normal să se întrebe dacă un model antrenat pe baza unui set date corespunzător unor magnitudini normale $C \in (0, 35)$ poate da rezultate semnificative pentru un set de date cu magnitudini ale emitter-ului puternice $C \geqslant 35$.

3.3. Preprocesarea datelor

În urma extragerii datelor din rețea este extrem de importantă etapa de prelucrare și preprocesare a datelor. Domeniul de preprocesare a datelor este unul extrem de vast și important în domeniul de învățare automată (engl. Machine Learning) și procesare de semnal. Preprocesarea datelor este etapa în care datele de intrare pentru un algoritm sunt aduse la o formă optimă pentru desfășurarea procesului impus, de exemplu în domeniul clasificării este important ca algoritmii să primească date care să fie scalate într-un anumit domeniu, pentru a asigura convergența[5], [2]. Alegerea metodei de preprocesare este strâns legată de tipul de date disponibile și de starea acestora. În cazul rețelelor de apă, unde am ales caracteristica presiunii ca mărime de intrare pentru algoritm și ținând cont de răspunsul în timp al rețelei am considerat ca fiind necesare următoarele operații:

- eliminarea frontului comun și extragerea diferenței dintre semnalul nominal și cel măsurat în rețea
- filtrarea semnalului obținut anterior

3.4. Nomenclatura mărimilor alese

Pentru a menține rigurozitatea și eleganța metodelor folosite este nevoie de o definire matematică pentru toate mărimile și metodele de filtrare folosite.

3.4.1. Presiunea în regim dinamic

Reprezintă o funcție de timp:

$$p_i : \mathbb{R} \longrightarrow \mathbb{R}^n, i \in V \tag{3.1}$$

unde n reprezintă numărul de noduri al rețelei, iar i reprezintă indexul nodului. Deoarece cazurile tratate în această lucrare reprezintă momente discrete de timp este important să definim presiunea măsurată în intervalele discrete în care este simulat procesul:

$$\mathbf{p}_i \in \mathbb{R}^{n \times p_{sim}} \tag{3.2}$$

unde p_{sim} reprezintă numărul de eșantioane pentru fiecare măsurătoare. Mergând mai departe în analiza simulării este de asemenea important să definim mărimea afectată de un defect în nodul j, de magnitudine m și măsurată în nodul i:

$$\mathbf{p}_i^{j,m} \in \mathbb{R}^{n \times p_{sim}} \tag{3.3}$$

Pentru cazul în care magnitudinea m ia valori nule, atunci vom considera notația mărimii nominale:

$$\mathbf{p}_i^{j,0} = \mathbf{p}_i^{nom}, \forall j \in V \tag{3.4}$$

Pentru valorile presiunii recoltate din rețea în nodul i despre care nu se cunoaște nici o informație, vom considera notația

$$\widehat{\mathbf{p}}_i \tag{3.5}$$

3.4.2. Presiunea în regim static

Considerând o plajă de momente de timp situate între indicii $rs_1 : rs_2$ unde se afla valorile de regim staționar ale procesului, putem defini o medie a regimului static în felul următor:

$$\overline{\mathbf{p}}_i^{j,m} = \frac{1}{rs_1 - rs_2 + 1} \sum_{k=rs_1}^{rs_2} \mathbf{p}_i^{j,m}[k] \tag{3.6}$$

În aceeași manieră definim și media presiunii nominale în regim static:

$$\overline{\mathbf{p}}_i^{j,0} = \overline{\mathbf{p}_i}^{nom}, \forall j \in V \tag{3.7}$$

Media presiunii măsurată în nodul i și despre care nu se cunosc informații în legătură cu valoarea și poziția defectului:

$$\overline{\widehat{\mathbf{p}}}_i = \frac{1}{rs_1 - rs_2 + 1} \sum_{k=rs_1}^{rs_2} \widehat{\mathbf{p}}_i[k] \tag{3.8}$$

3.4.3. Reziduuri

Așa cum a fost discutat în secțiunea 3.3, preprocesarea datelor are un rol important iar în cazul analizei și clasificării defectelor în rețelele cu apă, este nevoie să definim caracteristica prelucrată care va fi folosită mai apoi în procesul de izolare a defectelor. Reziduul absolut

reprezintă diferența dintre valoarea măsurată în rețea și valoarea nominală, aici putem discerne două cazuri: Reziduu temporal:

$$\mathbf{r}_i^{j,m} = \mathbf{p}_i^{j,m} - \mathbf{p}_i^{nom} \tag{3.9}$$

Reziduu atemporal, calculat ca diferența dintre cele două valori mediate pe intervalul staționar al caracteristicii:

$$r_i^{j,m} = \overline{\mathbf{p}}_i^{j,m} - \overline{\mathbf{p}}_i^{nom} \tag{3.10}$$

iar pentru valorile reziduului despre care nu se cunosc încă lucruri folosim notația din stilul anterior:

$$\widehat{r}_i = \overline{\widehat{\mathbf{p}}}_i - \overline{\mathbf{p}}_i^{nom} \tag{3.11}$$

Alte tipuri de reziduuri preprocesate sunt relative:

$$rrelativ_i^{j,m} = \frac{r_i^{j,m}}{\overline{\mathbf{p}}_i^{nom}} \tag{3.12}$$

Reziduurile normate:

$$rnorm_i^{j,m} = \frac{r_i^{j,m}}{\left\| r_{1:n}^{j,m} \right\|} \tag{3.13}$$

Reziduurile scalate:

$$rscal_i^{j,m} = \frac{r_i^{j,m} - \min r_{1:n}^{j,m}}{\max r_{1:n}^{j,m} - \min r_{1:n}^{j,m}} \tag{3.14}$$

Ca semnificație notațiile prezentate în 3.4 care conțin simbolul $\widehat{\ }$ fac referire la datele folosite pentru validarea modelului iar valorile unde se specifică nodul defectului și magnitudinea acestuia sunt considerate ca fiind date de antrenare și testare. Astfel în contextul definirii setului de dat pe care vom aplica algoritmii de calsificare va trebui să definim matricea:

$$\mathbf{R} < \mathbf{tip} >^{j,m} \in \mathbb{R}^{n_d \times n} \tag{3.15}$$

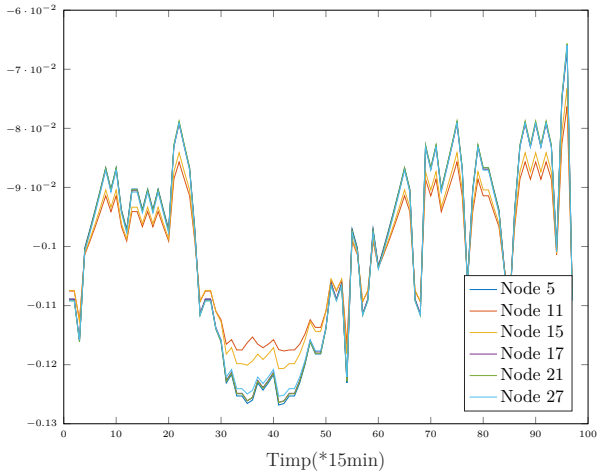
Unde croșetele din formulă reprezintă un înlocuitor pentru metoda de reziduu folosită iar n_d reprezintă numărul de defecte tratate în setul de date. De asemenea pentru fiecare linie a matricei (3.15) putem defini perechea

$$(\mathbf{R} < tip > (d, :), y_d) \tag{3.16}$$

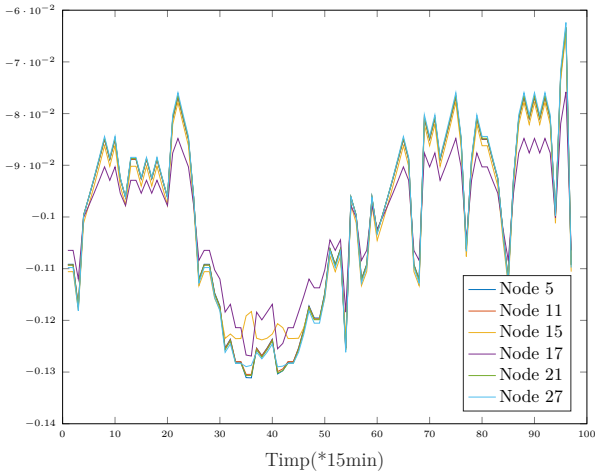
Unde $\mathbf{R} < tip > (d, :)$ reprezintă răspunsul rețelei prin reziduuri la defectul d. Iar y_d reprezintă eticheta pentru acest set de date, anume, nodul în care a avut loc defectul.

3.5. Calcul și prezentare reziduuri

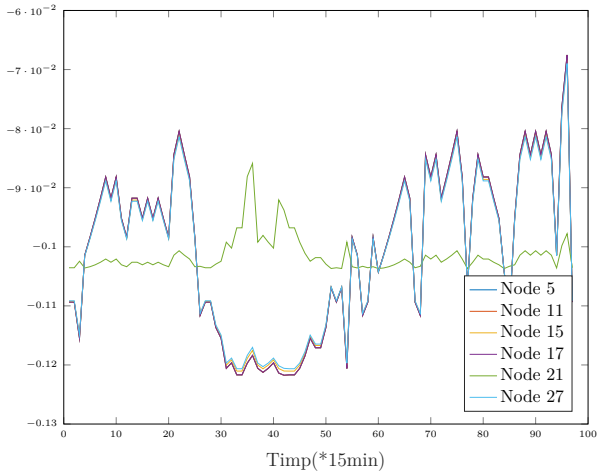
În continuare vom prezenta grafic reziduurile temporale normalizate care apar în rețea pentru diferite scenarii ale defectelor definite anterior. Astfel vom considera nodurile de măsurătoare ca o submulțime a lui $V' \subset V$ și $V = \{5, 11, 15, 17, 21, 27\}$, nodurile în care s-au simulat defecte sunt $V_d = \{11, 17, 27, 29\}$.



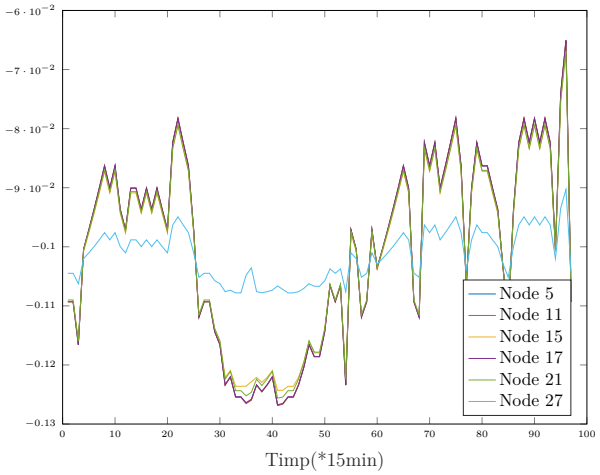
(a) Reziduuri pentru defect în nodul 11, magnitudine 29



(b) Reziduuri pentru defect în nodul 17, magnitudine 29



(c) Reziduuri pentru defect în nodul 21, magnitudine 29

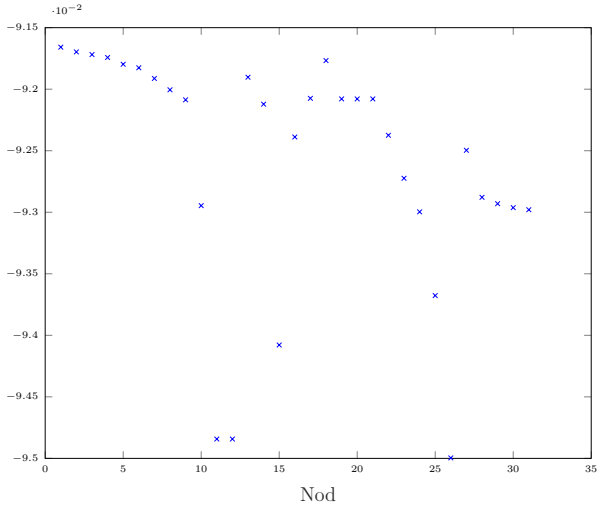


(d) Reziduuri pentru defect în nodul 27, magnitudine 29

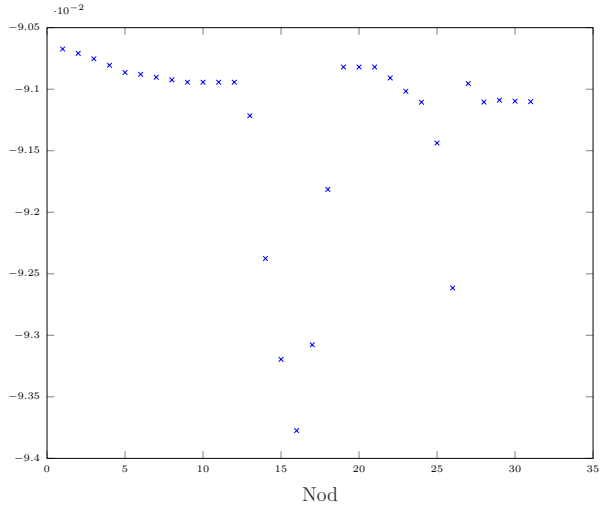
Figura 3.3.: Reziduuri rețea

Se poate observa că în figurile 3.4 reziduul cel mai pronunțat ca funcție de timp se găsește în nodul în care se simulează și defectul - lucru natural și de așteptat. O caracteristică importantă a acestei rețele de apă este faptul că există o dependență între diferitele răspunsuri în timp ale caracteristicii de presiune, fapt care ne permite să exploatăm redundanțele din rețea și să prezicem cu o acuratețe relativ ridicată defectele.

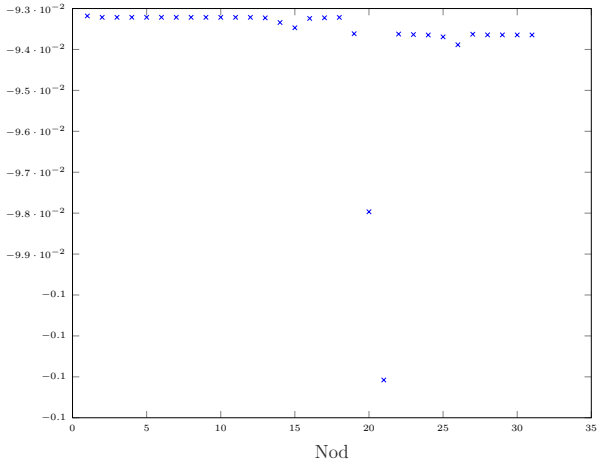
Este necesar acum să prezentăm profilurile reziduurilor atemporale, care în final vor reprezenta caracteristicile de intrare pentru algoritmul de clasificare și selecție de senzori.



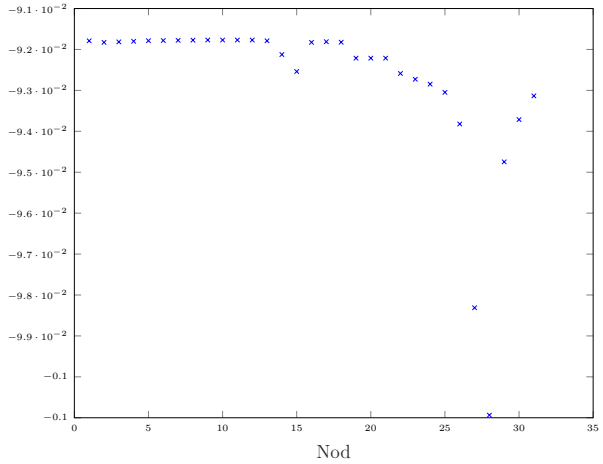
(a) Reziduuri pentru defect în nodul 11, magnitudine 25



(b) Reziduuri pentru defect în nodul 17, magnitudine 25



(c) Reziduuri pentru defect în nodul 21, magnitudine 25



(d) Reziduuri pentru defect în nodul 27, magnitudine 25

Figura 3.4.: Reziduuri atemporale rețea

Asemenea reziduurilor de la 3.4 se poate observa că simularea unui emitter într-un nod va determina un răspuns puternic în nodul respectiv și în vecinătatea nodului afectat

Anexe

A. Fișiere sursă

```
# epanet toolkit
import json

import numpy as np
import pandas as pd
from epanettools.epanet2 import *
from epanettools.epanettools import *
from plotly import tools
from plotly.graph_objs import *
# plotting imports
from plotly.offline import plot

# extending the EPANetSimulation class to ease acces to
# simulation routines
class ENSim(EPANetSimulation):
    EN_INIT = 10

    def __init__(self, network_file, pdd=False):
        # careful when using pdd = true for residues simulations
        self.json_sim = {}
        super().__init__(network_file, pdd)

    def set_emitter(self, node_index, emitter_val):
        if self.network.nodes[node_index].node_type is EN_JUNCTION:
            ENSim._getncheck(self.ENsetnodevalue(node_index, EN_EMITTER, emitter_val
    ))

    def set_basedemand(self, node_index, demand_val):
        if self.network.nodes[node_index].node_type is EN_JUNCTION:
            ENSim._getncheck(self.ENsetnodevalue(node_index, EN_BASEDEMAND,
    demand_val))

    def set_emitters(self, emitter_info=None):

        if emitter_info is None:
            # if arg is none reset emitter values
            for node_index in self.network.nodes:
                self.set_emitter(node_index, 0)
        else:

            for node_index, emitter_val in emitter_info:
                self.set_emitter(node_index, emitter_val)

    def get_nodes_data(self, data_query, emitter=(1, 0)):

        no_nodes = ENSim._getncheck(self.ENgetcount(EN_NODECOUNT)) - ENSim.
    _getncheck(self.ENgetcount(EN_TANKCOUNT))
        t_step = 1
        node_values = {}

        for queries in data_query:
            node_values[queries] = [[] for _ in range(no_nodes)]

        node_values["EMITTER_NODE"] = emitter[0]
        node_values["EMITTER_VAL"] = emitter[1]

        # initialize network for hydraulic process
```

```
56          ENSim._getncheck(self.ENinitH(ENSim.EN_INIT))

58          while t_step > 0:

60              self.ENrunH()

62              for node_index in range(1, no_nodes + 1):
                    for query_type in data_query:
64                      ret_val = ENSim._getncheck(self.ENgetnodevalue(node_index, eval(
    query_type)))
                        node_values[query_type][node_index - 1].append(ret_val)
66
                t_step = ENSim._getncheck(self.ENnextH())
68
            for key in node_values:
70              node_values[key] = np.transpose(node_values[key]).tolist()

72          return node_values

74      def get_links_data(self, data_query, emitter=(1, 0)):

76          no_links = self.ENgetcount(EN_LINKCOUNT)[1]
            t_step = 1
78          link_values = {}

80          for queries in data_query:
                link_values[queries] = [[] for _ in range(no_links)]
82
            link_values["EMITTER_NODE"] = emitter[0]
84          link_values["EMITTER_VAL"] = emitter[1]

86          # initialize network for hydraulic process

88          ENSim._getncheck(self.ENinitH(ENSim.EN_INIT))

90          while t_step > 0:
                ENSim._getncheck(self.ENrunH())
92
                for link_index in range(1, no_links + 1):
94                  for query_type in data_query:
                        ret_val = ENSim._getncheck(self.ENgetlinkvalue(link_index, eval(
    query_type)))
96                      link_values[query_type][link_index - 1].append(ret_val)

98              t_step = ENSim._getncheck(self.ENnextH())

100         for key in link_values:
                link_values[key] = np.transpose(link_values[key]).tolist()
102
            return link_values
104
        def query_network(self, sim_dict):
106         """
            :param sim_dict: a dict containing info about the network
108         has the form
            {
110             simulation_name : "name",
                simulation_type: "H" or "Q"
112             emitter_values : [ (node_index, emitter_value) ]
                query : {
114                 nodes : [ "EN_PRESSURE"
                    ]
116                 links : [ "EN_VELOCITY"
                    ]
118             }
```

```
        }
120     :return: JSON with required data
        output_json format:
122     {
            SIM_NAME = simulation-name
124         NODE_VALUES = [
                {
126                 "EN_PRESSURE" : [ [values for each node]]
                    "EMITTER_VAL" :
128                 "EMITTER_NODE" :
                }
130         ]

132     }

134     """

136     # for the moment i'll treat only hydraulic simulations :)

138     # initialize network simulaton

140     ENSim._getncheck(self.ENopenH())

142     # initialize session
        ENSim._getncheck(self.ENinitH(ENSim.EN_INIT))
144
        node_query = False
146     link_query = False
        simulations = False
148
        # check json for querried data
150
        # node info:
152     try:
            if sim_dict["query"]["nodes"]:
154             node_query = True
        except KeyError:
156         node_query = False

158     # link info
        try:
160         if sim_dict["query"]["links"]:
                link_query = True
162     except KeyError:
            link_query = False
164
        # emitter info:
166     try:
            simulations = sim_dict["emitter_values"]
168     except KeyError:
            simulations = False
170
        if simulations:
172         node_values = []
            link_values = []
174
            # in order to plot residues we need to simulate the case where there is
    no leakage
176
            self.set_emitters()
178         if node_query:
                node_values.append(
180                 self.get_nodes_data(sim_dict["query"]["nodes"]))

182         if link_query:
                link_values.append(
```

```
184                     self.get_links_data(sim_dict["query"]["links"]))
186             for node_index, emitter_value in simulations:
                    print("Simulating emitter in node no {} with value {}".format(
    node_index, emitter_value))
188
                    self.set_emitter(node_index, emitter_value)
190
                    if node_query:
192                     node_values.append(
                            self.get_nodes_data(sim_dict["query"]["nodes"], emitter=(
    node_index, emitter_value)))
194
                    if link_query:
196                     link_values.append(
                            self.get_links_data(sim_dict["query"]["links"], emitter=(
    node_index, emitter_value)))
198
                    # reset emitter values everywhere in network
200                 self.set_emitters()
202         else:
204             if node_query:
                    node_values = [self.get_nodes_data(sim_dict["query"]["nodes"])]
206             else:
                    node_values = []
208
                if link_query:
210                 link_values = [self.get_links_data(sim_dict["query"]["links"])]
                else:
212                 link_values = []
214         ENSim._getncheck(self.ENcloseH())
            ENSim._getncheck(self.ENclose())
216
            self.__init__(self.OriginalInputFileName)
218         self.json_sim = {
                "SIM_NAME": sim_dict["simulation_name"],
220             "NODE_VALUES": node_values,
                "LINK_VALUES": link_values
222         }
            return self.json_sim
224
    def get_time_step(self, pattern_id=1):
226         """
            returns the time_step of the network in minutes
228         :param pattern_id:
            :return:
230         """
            return (24 * 60) / ENSim._getncheck(self.ENgetpatternlen(pattern_id))
232
    def plot(self, json_data, residues=False):
234         """
            utility function used to plot data from network simulations
236         WIP
            :param json_data:
238         :return:
            """
240         values = json_data["NODE_VALUES"]
            date_range = pd.date_range('1/1/2018', periods=97, freq='15min')
242
            if residues:
244             # consider the first value of the JSON as the reference
                ref = values[0]
246             for emitter in values:
```

```
                trace = []
248             data = np.transpose(emitter["EN_PRESSURE"]) - np.transpose(ref["
    EN_PRESSURE"])

250

                for node_index, vals in enumerate(data):
252                 trace.append(Scatter(
                        x=date_range,
254                     y=vals,
                        name="node{}".format(node_index+1)
256                 )
                    )
258             layout = dict(
                    title = "Ressidues with emitter in node {}, val = {}".format(
    emitter["EMITTER_NODE"], emitter["EMITTER_VAL"])
260             )

262             fig = dict(data=trace, layout=layout)
                plot(fig, filename= "Plot_node{}val{}".format(emitter["EMITTER_NODE"
    ], emitter["EMITTER_VAL"]))
264
        else:
266         for emitter in values:
                trace = []
268             data = np.transpose(emitter["EN_PRESSURE"])
                for node_index, vals in enumerate(data):
270                 trace.append(Scatter(
                        x=date_range,
272                     y=vals,
                        name="node{}".format(node_index+1)
274                 )
                    )
276             layout = dict(
                    title = "Pressure with emitter in node {}, val = {}".format(
    emitter["EMITTER_NODE"], emitter["EMITTER_VAL"])
278             )

280             fig = dict(data=trace, layout=layout)
                plot(fig, filename= "Plot_node{}val{}".format(emitter["EMITTER_NODE"
    ], emitter["EMITTER_VAL"]))
282
    def save_data(self, path=None):
284
        try:
286         with open(path, "wt") as file:
                file.write(json.dumps(self.json_sim))
288     except IOError:
            print("Could not write to file {}".format(path))
290
    @staticmethod
292 def write_json(output_json, path):

294     json_data = json.dumps(output_json)
        with open(path, "wt") as f:
296         f.write(json_data)

298 @staticmethod
    def _getncheck(ret_val):
300
        # check the return code
302     if isinstance(ret_val, list):
            if ret_val[0] == 0:
304             # everything OK
                return ret_val[1]
306         else:
                err_msg = ENgeterror(ret_val[0], 100)
```

```
                raise EpanetError(err_msg)
        else:
            if ret_val is not 0:
                err_msg = ENgeterror(ret_val, 100)
                raise EpanetError(err_msg)


def en_check(func):
    def func_wrapper(*args):
        ret_val = func(args)

        # check the return code
        if isinstance(ret_val, list):
            if ret_val[0] == 0:
                # everything OK
                return ret_val[1]
            else:
                err_msg = ENgeterror(ret_val[0], 100)
                raise EpanetError(err_msg)
        else:
            if ret_val is not 0:
                err_msg = ENgeterror(ret_val, 100)
                raise EpanetError(err_msg)


class EpanetError(Exception):

    def __init__(self, err_msg):
        super().__init__(err_msg)


def run_simulation(network, pdd, query_dict):

    es = EPANetSimulation(network, pdd)

    print("Running {}".format(query_dict["simulation_name"]))
    ret_vals = []

    print(query_dict)
    for emitter, emitter_val in query_dict["emitter_values"]:
        print("for node {} simulating emitter_Val {}".format(emitter, emitter_val))

        # modify current network and and save inp temp file

        es.ENsetnodevalue(emitter, EN_EMITTER, emitter_val)

        es.ENsaveinpfile("temp.inp")
        print(es.ENsetnodevalue(emitter, EN_EMITTER, 0))

        e2 = EPANetSimulation("temp.inp", pdd)
        e2.ENsetnodevalue(emitter,EN_EMITTER,emitter_val)

        e2.run()

        node_vals = {}
        link_vals = {}
        for node_query in query_dict["query"]["nodes"]:
            node_vals[node_query] = []

        for link_query in query_dict["query"]["links"]:
            link_vals[link_query] = []

        for node_query in query_dict["query"]["nodes"]:
            for node in e2.network.nodes:
                node_vals[node_query].append(e2.network.nodes[node].results[eval(
    node_query)])
```

```
374         for link_query in query_dict["query"]["links"]:
                for link in e2.network.links:
376                 link_vals[link_query].append(e2.network.links[link].results[eval(
    link_query)])


378
            ret_vals.append({
380             "EMITTER_VAL" : emitter_val,
                "EMITTER_NODE" : emitter,
382             "NODE_VALS" : np.transpose(node_vals).tolist(),
                "LINK_VALS" : np.transpose(link_vals).tolist()
384         })
        return ret_vals
386
#TODO metoda pentru modificarea demand-ului pe nod!!!
388 if __name__ == '__main__':
        es = ENSim("data/hanoi.inp", pdd=False)
390
        test_vals = [val for val in range(32) if val % 2 == 0]
392     train_vals = [val for val in range(32) if val % 2 == 1]


394
        intense_leak = [35, 40, 50, 60, 100]
396
        nodes = list(range(1, 32))
398
        emitter_test = [ (node, val) for node in nodes for val in test_vals]
400     emitter_train = [(node, val) for node in nodes for val in train_vals]
        emitter_intense_leak = [(node, val) for node in nodes for val in intense_leak]
402
        train_dataset = {
404         "simulation_name": "Hanoi train simulation",
            "simulation_type": "H",
406         "emitter_values": emitter_train,
            "query": {
408
                "nodes": ["EN_PRESSURE", "EN_DEMAND"],
410             "links": ["EN_VELOCITY"]
            }
412
        }
414
        test_dataset = {
416         "simulation_name": "Hanoi test simulation",
            "simulation_type": "H",
418         "emitter_values": emitter_test,
            "query": {
420
                "nodes": ["EN_PRESSURE", "EN_DEMAND"],
422             "links": ["EN_VELOCITY"]
            }
424
        }
426
        test2_dataset = {
428
            "simulation_name": "Hanoi intense leak simulation",
430         "simulation_type": "H",
            "emitter_values": emitter_intense_leak,
432         "query": {

434             "nodes": ["EN_PRESSURE", "EN_DEMAND"],
                "links": ["EN_VELOCITY"]
436         }
```

```
438     }

440     data_train = es.query_network(train_dataset)
        es.save_data("train_set.json")
442     data_test = es.query_network(test_dataset)
        es.save_data("test_set.json")
444     data_test2 = es.query_network(test2_dataset)
        es.save_data("test2_set.json")
```

Listing A.1: Wrapper EPANET – fișier complet

Bibliography

[1] Donald F Elger and John A Roberson. *Engineering fluid mechanics*. Wiley Hoboken (NJ), 2016.

[2] *Garbage in Garbage out.* 2016. URL: https://en.wikipedia.org/wiki/Garbage_in,_garbage_out.

[3] Richard M Karp. „On the computational complexity of combinatorial problems“. In: *Networks* 5.1 (1975), pp. 45–68.

[4] Rex Klopfenstein Jr. „Air velocity and flow measurement using a Pitot tube“. In: *ISA transactions* 37.4 (1998), pp. 257–263.

[5] SB Kotsiantis, D Kanellopoulos, and PE Pintelas. „Data preprocessing for supervised leaning“. In: *International Journal of Computer Science* 1.2 (2006), pp. 111–117.

[6] Assela Pathirana. *EPANET calling API for python.* 2016. URL: https://github.com/asselapathirana/epanettools.

[7] Lewis A Rossman et al. „EPANET 2: users manual“. In: (2000).

[8] Gerard Sanz Estapé. „Demand modeling for water networks calibration and leak localization“. In: (2016).