



Universitatea Politehnica București  
Facultatea de Automatică și Calculatoare  
Departamentul de Automatică și Ingineria Sistemelor

# LUCRARE DE LICENȚĂ

## Clasificare defecte într-o rețea de apă de mari dimensiuni

Absolvent  
Cazan Cristian-Claudiu

Coordonator  
Conf. dr. ing. Florin Stoican

București, 2018

# Cuprins

<b>Listă de figuri</b>	<b>ii</b>
<b>Listă de tabele</b>	<b>iii</b>
<b>Listă de algoritmi</b>	<b>iv</b>
<b>1. Introducere</b>	<b>1</b>
1.1. Motivația alegerii temei . . . . .	1
1.2. Expunerea problemei . . . . .	2
1.3. Exemplul de lucru . . . . .	4
<b>2. Simulări și software folosit</b>	<b>5</b>
2.1. Dificultatea simulării unei rețele de apă . . . . .	5
2.2. Simulări folosind biblioteca EPANET . . . . .	6
2.2.1. Schema bibliotecii ANSI C - EPANET . . . . .	8
2.2.2. Structura fișierului de intrare .INP . . . . .	8
2.3. Integrarea EPANET cu Python . . . . .	10
2.4. Simulări pe rețeaua de apă din Hanoi . . . . .	12
2.5. Prezentarea măsurilor de interes preluate din simulări . . . . .	13
2.5.1. Simulare dinamică pentru încărcare nominală . . . . .	13
2.5.2. Alte mărimi de interes din simulatorul EPANET . . . . .	13
<b>3. Detectția și izolarea defectelor</b>	<b>15</b>
3.1. Definirea defectelor . . . . .	15
3.1.1. Simulare dinamică pentru defecte în diferite noduri . . . . .	15
<b>A. Fișiere sursă</b>	<b>18</b>
<b>Bibliography</b>	<b>26</b>

# Listă de figuri

1.1. Graful rețelei de apă din Hanoi . . . . .	4
2.1. Simulatorul EPANET . . . . .	7
2.2. Fluxul de date în biblioteca EPANET . . . . .	8
2.3. Profilul normal de utilizare . . . . .	12
2.4. Profilul nominal al presiunii . . . . .	13
2.5. Rezultate simulări nominale . . . . .	14
(a). Profil nominal demand . . . . .	14
(b). Profil nominal viteza . . . . .	14
3.1. Rezultate simulări defecte ușoare . . . . .	15
(a). Profile cu defect în nodul 14 . . . . .	15
(b). Profil cu defect în nodul 25 . . . . .	15
3.2. Rezultate simulări defecte puternice . . . . .	16
(a). Profile cu defect puternic în nodul 14 . . . . .	16
(b). Profil cu defect puternic în nodul 25 . . . . .	16

# Listă de tabele

2.1. Structura fişierului INP . . . . .	9
---	---

## Listă de algoritmi

# 1. Introducere

## 1.1. Motivația alegerii temei

Transportul și distribuția apei reprezintă una dintre cele mai vechi preocupări ingineresti 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 ingineresti, apeductele construite de aceștia impresionând și astăzi prin grandoearea și iscusunța 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 Siracusa în sec III î.Hr.  $F = V_{obiect} * \rho_{lichid} * g$ . O altă contribuție care are o deosebită importanță în domeniul tehnologiei de distribuție 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 ;[3], ecuația care guvernează depedența nivel - viteză este:

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

unde:

- $u$  reprezintă viteza fluidului
- $p_t$  reprezintă presiunea de stagnare
- $p_s$  reprezintă presiunea statică
- $\rho$  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):

$$\frac{v^2}{2} + gz + \frac{p}{\rho} = c \quad (1.2)$$

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
- $\rho$  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 parametri ai sistemului de analizat. Spre exemplu ecuația Poisuille care modelează începutul fluxului de apă într-o conductă este ;[1]:

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

unde:

- $u$  reprezintă viteza lichidului prin conductă
- $t$  reprezintă timpul
- $G$  reprezintă diferența de presiune
- $\rho$  reprezintă densitatea lichidului
- $\nu$  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 cunoaște 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 parametri 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  $mH_2O$ , 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 eficiente 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:

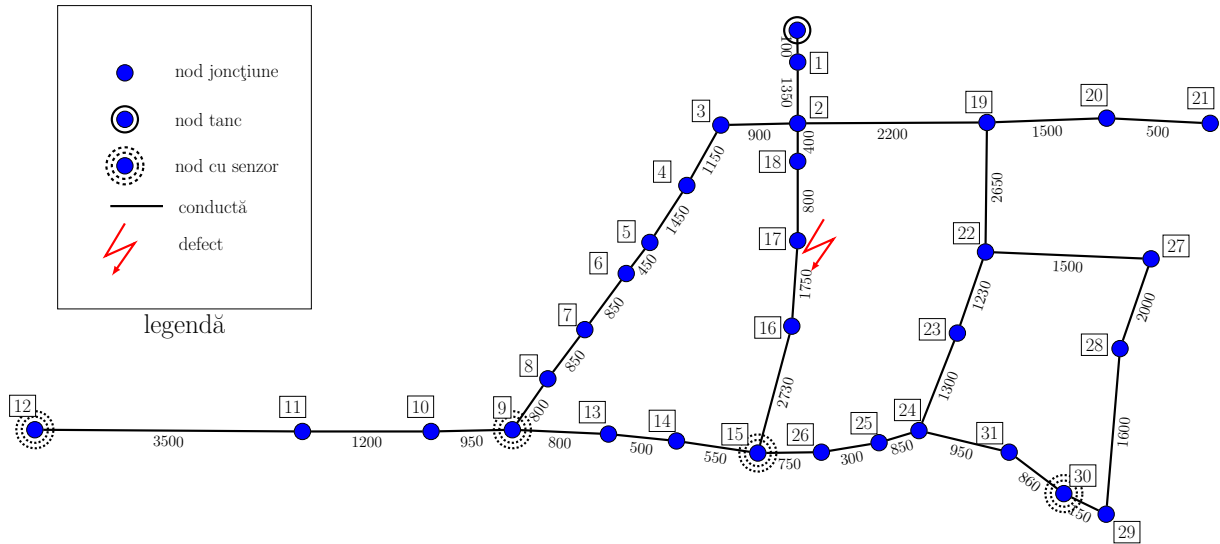


Figura 1.1.: Graful rețelei de apă din Hanoi

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 consideră 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 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 parametri 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 între 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 conducte 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:

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

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

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

Partea de estimare a diferenței de presiuni (în engl. "Head-Flow differential") între două noduri interconectate se face utilizând formula Hazen-Williams ;[6]:

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

unde:

- $\mathbf{h}$  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  $R_{ij} = \frac{10.67 \cdot L_l}{C_l^{1.852} \cdot D_l^{4.87}}$  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:

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

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  $\mathbf{d}$  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  $\mathbf{d}$ .

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 genera de fapt o problemă NP-completă, sau în unele cazuri chiar NP-dură ;[2].

## 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 ;[5].

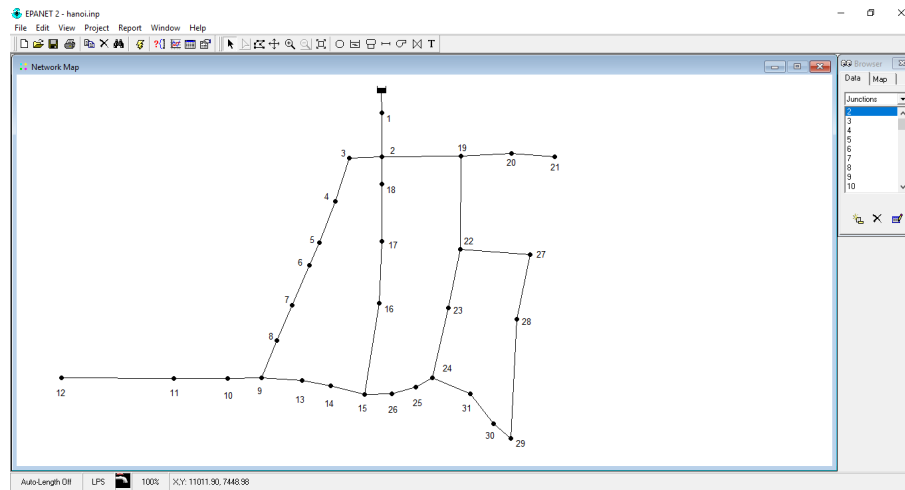


Figura 2.1.: Simulatorul EPANET

În urma execuției simulării EPANET va stoca toate datele în memorie și va putea realiza 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 posibilitatea 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 orificiu este:

$$q = Cp^\gamma \quad (2.5)$$

unde:

- $q$  reprezintă debitul prin emitter
- $C$  reprezintă o constantă de proporționalitate
- $p$  reprezintă presiunea din joncțiune
- $\gamma$  reprezintă exponentul de presiune

Astfel apariția unui defect într-un anumit nod determină apariția unei relații liniare în funcție de presiunea din nod. Coeficientul  $\gamma$  definește 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 ;[5].

### 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:

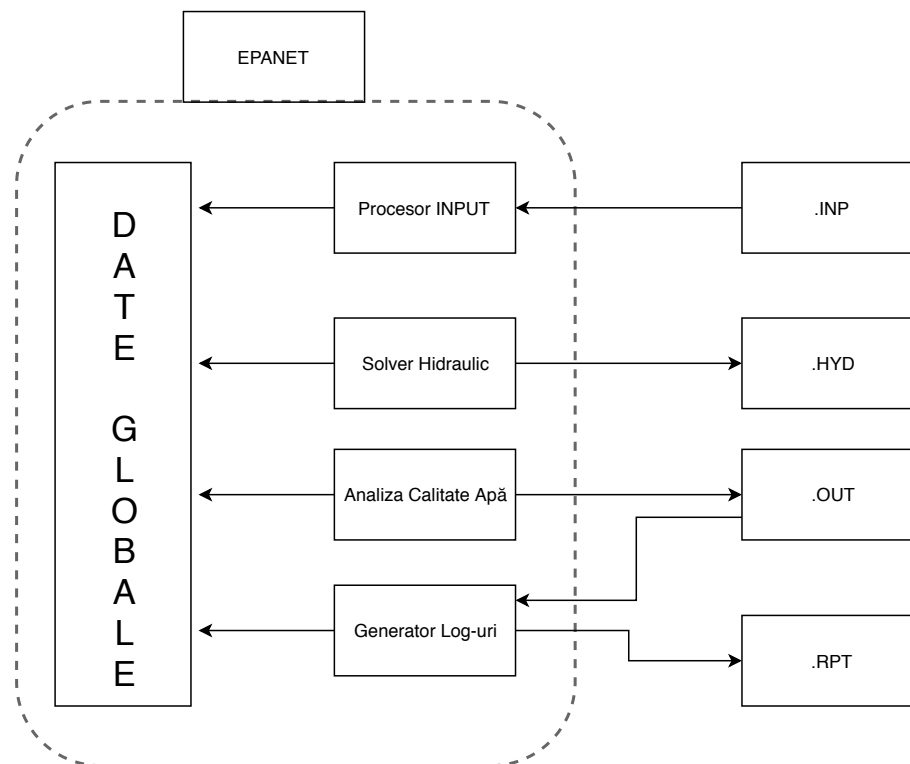


Figura 2.2.: Fluxul de date în biblioteca EPANET

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 - fișierul 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 implicați 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 [4]. 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 [4] 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 {
2     simulation_name : "name",
3     simulation_type: "H" or "Q"
4     emitter_values : [ (node_index, emitter_value) ]
5     query : {
6         nodes : [ "EN_PRESSURE"
7         ]
8         links : [ "EN_VELOCITY"
```

```

9         ]
11    }

```

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:

```

1    {
2        SIM_NAME = simulation-name
3        NODE_VALUES = [
4            {
5                "EN_PRESSURE" : [ [values for each node]]
6                "EMITTER_VAL" :
7                "EMITTER_NODE" :
8            }
9        ]
11    }

```

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
2    def __getncheck(ret_val):
3
4        # check the return code
5        if isinstance(ret_val, list):
6            if ret_val[0] == 0:
7                # everything OK
8                return ret_val[1]
9            else:
10                err_msg = ENgeterror(ret_val[0], 100)
11                raise EpanetError(err_msg)
12    else:
13        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

```

2 | class EpanetError(Exception):
4 |     def __init__(self, err_msg):
        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 orelor matinale față de exemplu de perioada nopții unde valorile multiplicității ajung la valori aproximativ de două ori mai mici.

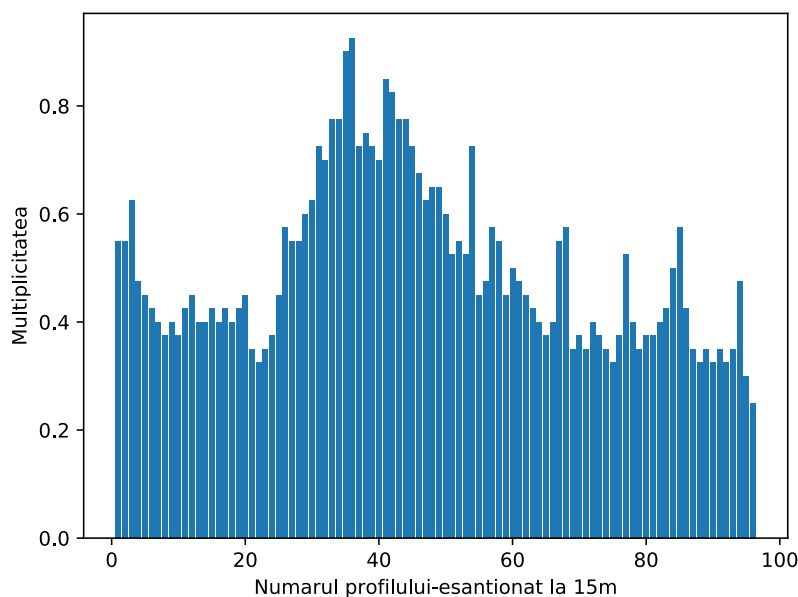


Figura 2.3.: Profilul normal de utilizare

Pentru rularea unei simulări nominale fișierul JSON de intrare poate avea câmpul de `emitter_values` gol, obținându-se astfel 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.

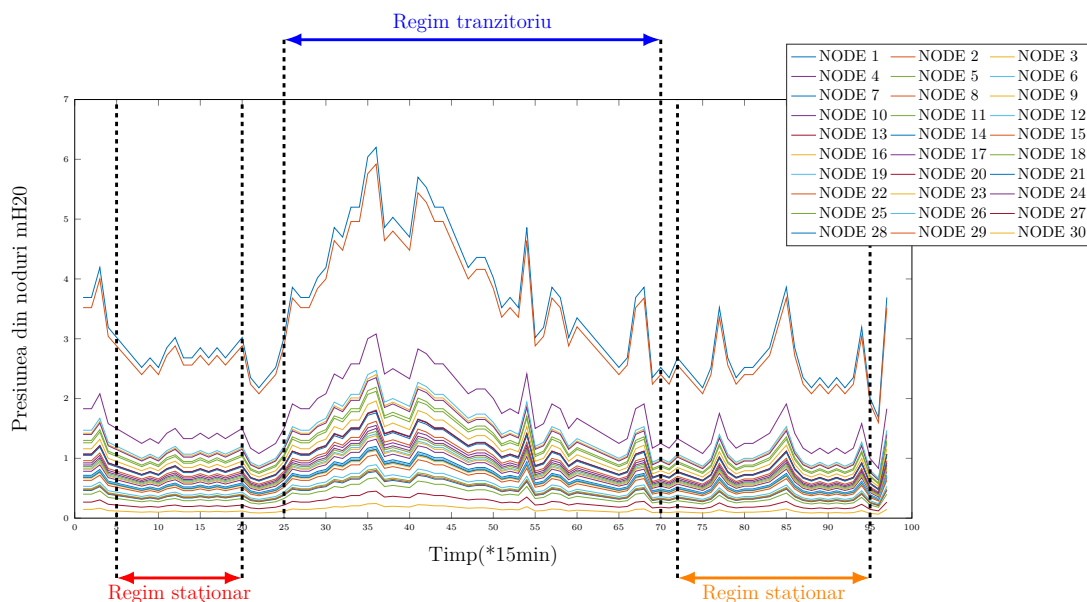
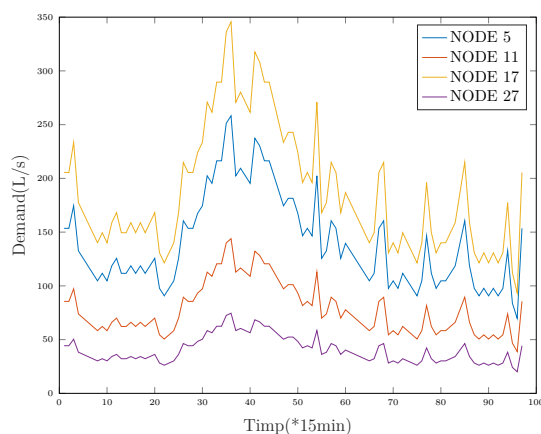


Figura 2.4.: Profilul nominal al presiunii

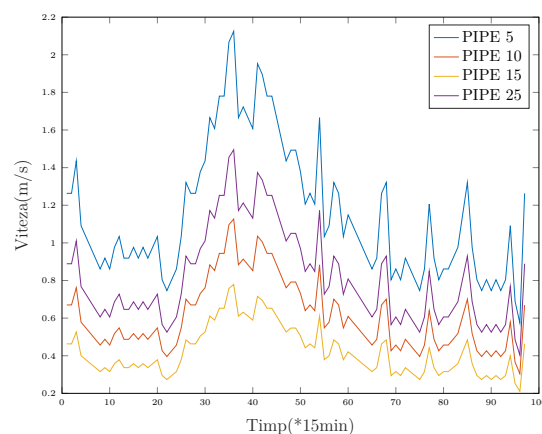
După cum se poate observa în 2.4 profilul nominal al presiunii intră în tiparul profilului de cerere prezentat în 2.3, unde în jurul orei dimineții, când activitatea utilizatorilor este asociată unui regim dinamic al rețelei, iar orele nopții sunt caracterizate de marimi lent variabile. Astfel pe acest 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 celorlalte 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):  
    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

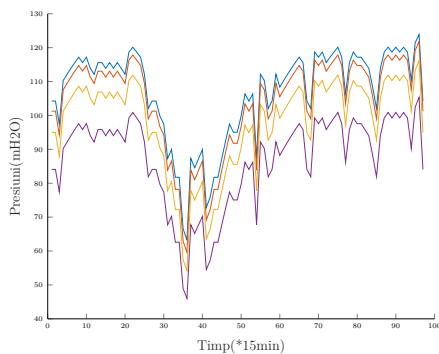
parametrii funcției *set\_emitter* sunt:

- *node\_index* - indexul nodului în care se simulează defectul
- *emitter\_val* - magnitudinea defectului

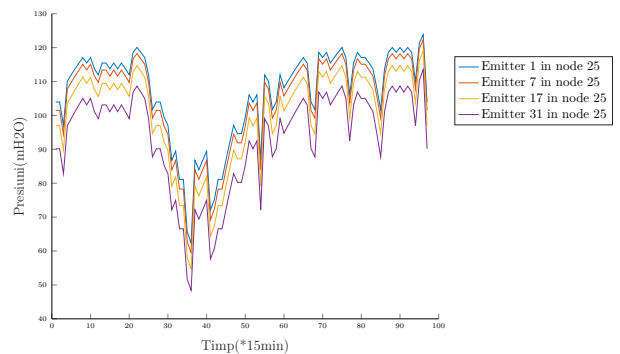
Metoda mai întâi verifică dacă nodul cu indexul *node\_index* reprezintă doar o joncțiune apoi setează magnitudinea defectului în nodul primit cu ajutorul funcției de bibliotecă **ENsetnodevalue**

### 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



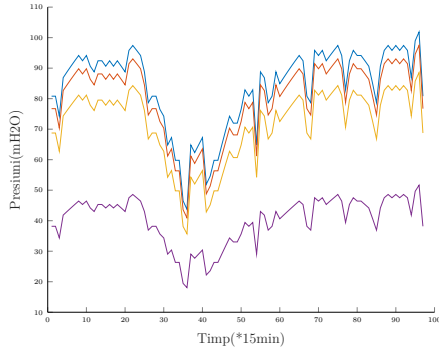
(b) Profil cu defect în nodul 25

Figura 3.1.: Rezultate simulări defecte ușoare

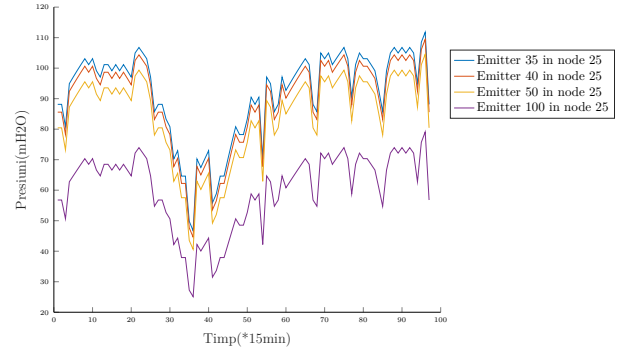
După cum se poate observa în imaginile 3.1 variația emitter-ului într-un nod produce în mod evident o modificarea a modului comun al caracteristicii *timp – presiune*. Din punctul de vedere al magnitudinilor de simulare pentru defecte, am considerat 2 clase de defecte, anume:

- 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 de date corespunzător unor magnitudini normale  $C \in (0, 35)$  poate da rezultate semnificative pentru un set de date cu magnitudini ale emitter-ului puternice  $C \geq 35$ .

# Anexe

## A. Fişiere 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):
63             for query_type in data_query:
64                 ret_val = ENSim._getncheck( self.ENgetnodevalue(node_index, eval(
query_type)))
65                 node_values[query_type][node_index - 1].append(ret_val)
66
67         t_step = ENSim._getncheck( self.ENnextH())
68
69         for key in node_values:
70             node_values[key] = np.transpose(node_values[key]).tolist()
71
72     return node_values
73
74 def get_links_data(self, data_query, emitter=(1, 0)):
75
76     no_links = self.ENgetcount(EN_LINKCOUNT)[1]
77     t_step = 1
78     link_values = {}
79
80     for queries in data_query:
81         link_values[queries] = [[] for _ in range(no_links)]
82
83     link_values["EMITTER_NODE"] = emitter[0]
84     link_values["EMITTER_VAL"] = emitter[1]
85
86     # initialize network for hydraulic process
87
88     ENSim._getncheck( self.ENinitH(ENSim.EN_INIT))
89
90     while t_step > 0:
91         ENSim._getncheck( self.ENrunH())
92
93         for link_index in range(1, no_links + 1):
94             for query_type in data_query:
95                 ret_val = ENSim._getncheck( self.ENgetlinkvalue(link_index, eval(
query_type)))
96                 link_values[query_type][link_index - 1].append(ret_val)
97
98         t_step = ENSim._getncheck( self.ENnextH())
99
100        for key in link_values:
101            link_values[key] = np.transpose(link_values[key]).tolist()
102
103    return link_values
104
105 def query_network(self, sim_dict):
106     """
107     :param sim_dict: a dict containing info about the network
108     has the form
109     {
110         simulation_name : "name",
111         simulation_type: "H" or "Q"
112         emitter_values : [ (node_index, emitter_value) ]
113         query : {
114             nodes : [ "EN_PRESSURE"
115                     ]
116             links : [ "EN_VELOCITY"
117                     ]
118         }

```



```

120     }
121     :return: JSON with required data
122     output_json format:
123     {
124         SIM_NAME = simulation-name
125         NODE_VALUES = [
126             {
127                 "EN_PRESSURE" : [ [values for each node]]
128                 "EMITTER_VAL" :
129                 "EMITTER_NODE" :
130             }
131         ]
132     }
133
134     """
135
136     # for the moment i'll treat only hydraulic simulations :)
137
138     # initialize network simulaton
139
140     ENSim.__getncheck(self.ENopenH())
141
142     # initialize session
143     ENSim.__getncheck(self.ENinitH(ENSim.EN_INIT))
144
145     node_query = False
146     link_query = False
147     simulations = False
148
149     # check json for queried data
150
151     # node info:
152     try:
153         if sim_dict["query"]["nodes"]:
154             node_query = True
155     except KeyError:
156         node_query = False
157
158     # link info
159     try:
160         if sim_dict["query"]["links"]:
161             link_query = True
162     except KeyError:
163         link_query = False
164
165     # emitter info:
166     try:
167         simulations = sim_dict["emitter_values"]
168     except KeyError:
169         simulations = False
170
171     if simulations:
172         node_values = []
173         link_values = []
174
175         # in order to plot residues we need to simulate the case where there is
176         no leakage
177
178         self.set_emitters()
179         if node_query:
180             node_values.append(
181                 self.get_nodes_data(sim_dict["query"]["nodes"]))
182
183         if link_query:
184             link_values.append(

```

```

184         self.get_links_data(sim_dict["query"]["links"]))
186     for node_index, emitter_value in simulations:
187         print("Simulating emitter in node no {} with value {}".format(
188             node_index, emitter_value))
189
190         self.set_emitter(node_index, emitter_value)
191
192         if node_query:
193             node_values.append(
194                 self.get_nodes_data(sim_dict["query"]["nodes"], emitter=(
195                     node_index, emitter_value)))
196
197         if link_query:
198             link_values.append(
199                 self.get_links_data(sim_dict["query"]["links"], emitter=(
200                     node_index, emitter_value)))
201
202         # reset emitter values everywhere in network
203         self.set_emitters()
204
205     else:
206
207         if node_query:
208             node_values = [self.get_nodes_data(sim_dict["query"]["nodes"])]
209         else:
210             node_values = []
211
212         if link_query:
213             link_values = [self.get_links_data(sim_dict["query"]["links"])]
214         else:
215             link_values = []
216
217     ENSim._getncheck(self.ENcloseH())
218     ENSim._getncheck(self.ENclose())
219
220     self.__init__(self.OriginalInputFileName)
221     self.json_sim = {
222         "SIM_NAME": sim_dict["simulation_name"],
223         "NODE_VALUES": node_values,
224         "LINK_VALUES": link_values
225     }
226     return self.json_sim
227
228 def get_time_step(self, pattern_id=1):
229     """
230     returns the time_step of the network in minutes
231     :param pattern_id:
232     :return:
233     """
234     return (24 * 60) / ENSim._getncheck(self.ENgetpatternlen(pattern_id))
235
236 def plot(self, json_data, residues=False):
237     """
238     utility function used to plot data from network simulations
239     WIP
240     :param json_data:
241     :return:
242     """
243     values = json_data["NODE_VALUES"]
244     date_range = pd.date_range('1/1/2018', periods=97, freq='15min')
245
246     if residues:
247         # consider the first value of the JSON as the reference
248         ref = values[0]
249         for emitter in values:

```

```

248         trace = []
249         data = np.transpose(emitter["EN_PRESSURE"]) - np.transpose(ref["
EN_PRESSURE"])
250
251         for node_index, vals in enumerate(data):
252             trace.append(Scatter(
253                 x=date_range,
254                 y=vals,
255                 name="node{}".format(node_index+1)
256             )
257         )
258         layout = dict(
259             title = "Residues with emitter in node {}, val = {}".format(
260 emitter["EMITTER_NODE"], emitter["EMITTER_VAL"])
261         )
262         fig = dict(data=trace, layout=layout)
263         plot(fig, filename= "Plot_node{}val{}".format(emitter["EMITTER_NODE"
], emitter["EMITTER_VAL"]))
264
265     else:
266         for emitter in values:
267             trace = []
268             data = np.transpose(emitter["EN_PRESSURE"])
269             for node_index, vals in enumerate(data):
270                 trace.append(Scatter(
271                     x=date_range,
272                     y=vals,
273                     name="node{}".format(node_index+1)
274                 )
275             )
276             layout = dict(
277                 title = "Pressure with emitter in node {}, val = {}".format(
278 emitter["EMITTER_NODE"], emitter["EMITTER_VAL"])
279             )
280             fig = dict(data=trace, layout=layout)
281             plot(fig, filename= "Plot_node{}val{}".format(emitter["EMITTER_NODE"
], emitter["EMITTER_VAL"]))
282
283     def save_data(self, path=None):
284
285         try:
286             with open(path, "wt") as file:
287                 file.write(json.dumps(self.json_sim))
288         except IOError:
289             print("Could not write to file {}".format(path))
290
291     @staticmethod
292     def write_json(output_json, path):
293
294         json_data = json.dumps(output_json)
295         with open(path, "wt") as f:
296             f.write(json_data)
297
298     @staticmethod
299     def __getncheck(ret_val):
300
301         # check the return code
302         if isinstance(ret_val, list):
303             if ret_val[0] == 0:
304                 # everything OK
305                 return ret_val[1]
306         else:
307             err_msg = ENgeterror(ret_val[0], 100)

```

```

308         raise EpanetError(err_msg)
309     else:
310         if ret_val is not 0:
311             err_msg = ENgeterror(ret_val, 100)
312             raise EpanetError(err_msg)
313
314 def en_check(func):
315     def func_wrapper(*args):
316         ret_val = func(args)
317
318         # check the return code
319         if isinstance(ret_val, list):
320             if ret_val[0] == 0:
321                 # everything OK
322                 return ret_val[1]
323             else:
324                 err_msg = ENgeterror(ret_val[0], 100)
325                 raise EpanetError(err_msg)
326         else:
327             if ret_val is not 0:
328                 err_msg = ENgeterror(ret_val, 100)
329                 raise EpanetError(err_msg)
330
331 class EpanetError(Exception):
332
333     def __init__(self, err_msg):
334         super().__init__(err_msg)
335
336 def run_simulation(network, pdd, query_dict):
337
338     es = EPANetSimulation(network, pdd)
339
340     print("Running {}".format(query_dict["simulation_name"]))
341     ret_vals = []
342
343     print(query_dict)
344     for emitter, emitter_val in query_dict["emitter_values"]:
345         print("for node {} simulating emitter_Val {}".format(emitter, emitter_val))
346
347         # modify current network and and save inp temp file
348
349         es.ENsetnodevalue(emitter, EN_EMITTER, emitter_val)
350
351         es.ENsaveinpfile("temp.inp")
352         print(es.ENsetnodevalue(emitter, EN_EMITTER, 0))
353
354         e2 = EPANetSimulation("temp.inp", pdd)
355         e2.ENsetnodevalue(emitter, EN_EMITTER, emitter_val)
356
357         e2.run()
358
359         node_vals = {}
360         link_vals = {}
361         for node_query in query_dict["query"]["nodes"]:
362             node_vals[node_query] = []
363
364         for link_query in query_dict["query"]["links"]:
365             link_vals[link_query] = []
366
367         for node_query in query_dict["query"]["nodes"]:
368             for node in e2.network.nodes:
369                 node_vals[node_query].append(e2.network.nodes[node].results[eval(
370 node_query)])

```

```

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

378
379         ret_vals.append({
380             "EMITTER_VAL" : emitter_val,
381             "EMITTER_NODE" : emitter,
382             "NODE_VALS" : np.transpose(node_vals).tolist(),
383             "LINK_VALS" : np.transpose(link_vals).tolist()
384         })
385     return ret_vals
386
387 #TODO metoda pentru modificarea demand-ului pe nod!!!
388 if __name__ == '__main__':
389     es = ENSim("data/hanoi.inp", pdd=False)
390
391     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]
393
394     intense_leak = [35, 40, 50, 60, 100]
395
396     nodes = list(range(1, 32))
397
398     emitter_test = [(node, val) for node in nodes for val in test_vals]
399     emitter_train = [(node, val) for node in nodes for val in train_vals]
400     emitter_intense_leak = [(node, val) for node in nodes for val in intense_leak]
401
402     train_dataset = {
403         "simulation_name": "Hanoi train simulation",
404         "simulation_type": "H",
405         "emitter_values": emitter_train,
406         "query": {
407             "nodes": ["EN_PRESSURE", "EN_DEMAND"],
408             "links": ["EN_VELOCITY"]
409         }
410     }
411
412     test_dataset = {
413         "simulation_name": "Hanoi test simulation",
414         "simulation_type": "H",
415         "emitter_values": emitter_test,
416         "query": {
417             "nodes": ["EN_PRESSURE", "EN_DEMAND"],
418             "links": ["EN_VELOCITY"]
419         }
420     }
421
422     test2_dataset = {
423         "simulation_name": "Hanoi intense leak simulation",
424         "simulation_type": "H",
425         "emitter_values": emitter_intense_leak,
426         "query": {
427             "nodes": ["EN_PRESSURE", "EN_DEMAND"],
428             "links": ["EN_VELOCITY"]
429         }
430     }
431
432     test3_dataset = {
433         "simulation_name": "Hanoi intense leak simulation",
434         "simulation_type": "H",
435         "emitter_values": emitter_intense_leak,
436         "query": {
437             "nodes": ["EN_PRESSURE", "EN_DEMAND"],
438             "links": ["EN_VELOCITY"]
439         }
440     }

```

```
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] Richard M Karp. „On the computational complexity of combinatorial problems“. In: *Networks* 5.1 (1975), pp. 45–68.
- [3] Rex Klopfenstein Jr. „Air velocity and flow measurement using a Pitot tube“. In: *ISA transactions* 37.4 (1998), pp. 257–263.
- [4] Assela Pathirana. *EPANET calling API for python*. 2016. URL: <https://github.com/asselapathirana/epanettools>.
- [5] Lewis A Rossman et al. „EPANET 2: users manual“. In: (2000).
- [6] Gerard Sanz Estapé. „Demand modeling for water networks calibration and leak localization“. In: (2016).