Journal of Hydroinformatics

Journal of Hydroinformatics Vol 26 No 1, 284 doi: 10.2166/hydro.2023.218

A robust simulator of pressure-dependent consumption in Python

Camille Chambon [a,b,*], Olivier Piller [a,c] and Iraj Mortazavi [b]

[a] INRAE, UR ETTIS, Cestas F-33612, France
[b] CNAM, M2N, Paris F-75003, France
[c] School of Civil, Environmental, and Mining Eng., Adelaide, SA 5005, Australia
*Corresponding author. E-mail: camille.chambon@inrae.fr

CC, 0000-0002-3521-6632; OP, 0000-0002-3625-7639; IM, 0000-0003-3584-533X

ABSTRACT

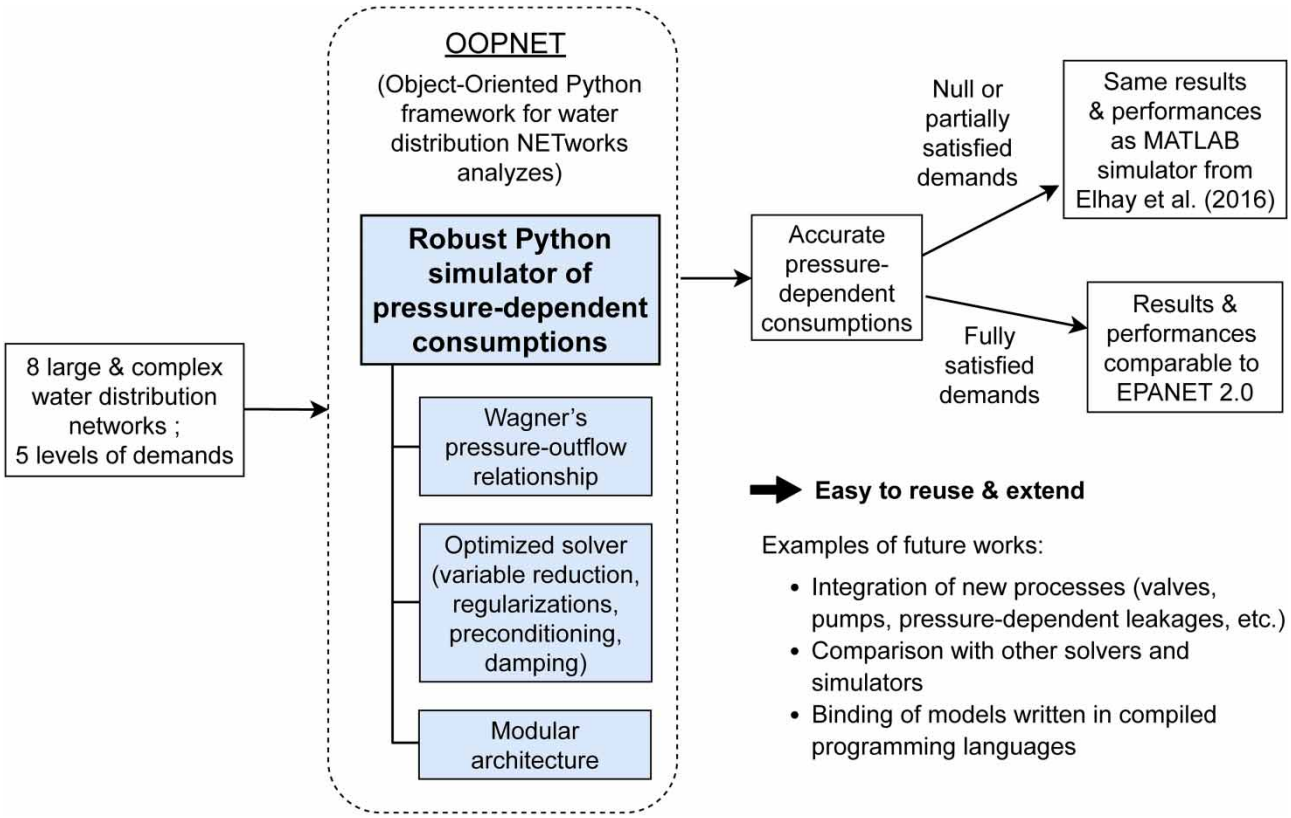
Modeling of pressure-dependent users' consumption is mandatory to simulate accurately the hydraulics of water distribution networks (WDNs). Several software solutions already exist for this purpose, but none of them actually permits the easy integration and testing of new physical processes. In this paper, we propose a new Python simulator that implements a state-of-the-art pressure-dependent model (PDM) of users' consumptions based on the Wagner's pressure–outflow relationship (POR). We tested our simulator on eight large and complex WDNs, for different levels of users' demands. The results show similar precision and efficiency to the ones obtained by the authors of the original model with their MATLAB implementation. Moreover, in case of fully satisfied users' demands, our simulator provides the same results as EPANET 2.0 in comparable computational times. Finally, our simulator is integrated into the open-source, collaborative, multi-platform, and Git versioned Python framework OOPNET (Object-Oriented Python framework for water distribution NETworks analyses); thus, it can be easily reused and/or extended by a large community of WDN modelers. All this work represents a preliminary step before the incorporation of new processes such as valves, pumps, and pressure-dependent background leakage outflows.

Key words: numerical optimization, pressure-dependent model (PDM), Python programing language, user's consumption, water distribution network (WDN)

HIGHLIGHTS

- A new simulator of pressure-dependent consumption in water distribution network.
- Coded in Python, based on a state-of-the-art MATLAB model, easy to extend.
- Numerical experiments on networks composed of up to 19,647 pipes and 17,986 nodes.
- Instabilities handled through regularization and line search along the Newton descent.
- Preliminary step for integration of new processes and first contribution to OOPNET.

GRAPHICAL ABSTRACT



1. INTRODUCTION

1.1. Users' consumption in WDNs

Households, schools, hospitals, businesses, food-processing and pharmaceutical industries, fire departments, etc., are all significant users of water distribution networks (WDNs). Each of them has its own need of water; moreover, this need can vary over time. Over a specific period, a need corresponds to a "demand", expressed as an outflow rate. From this demand, and depending on the capacities of the WDN, the amount of water a user actually consumes is called its "consumption". A user's consumption is between 0 (i.e., no water is provided to the user) and the demand (i.e., the demand of the user is fully satisfied). For example, in the last three decades, the worldwide amount of drinking water consumed daily by households and public services is estimated to be 1.28×10^9 m^3, which represents 510,995 Olympic-size swimming pools of 2,500 m^3, and 11 percent of total freshwater withdrawals (Ritchie & Roser 2017). In France, this amount is of 1.50×10^7 m^3 per day, which leads to an average domestic consumption of 1481 per day and inhabitant (Lao *et al.* 2022).

The first goal of WDNs is to satisfy the demands of the users. To fulfill these needs, WDNs must be well sized at their building, and then extended according to new needs (e.g., population growing or installation of new facilities). Pumps are used to maintain enough pressure to carry water from treatment plant or natural source to users. Also, valves are used to control the pressure, and to divide large networks in several district-metered areas (DMAs) that are easier to monitor with the help of sensors. Appropriate design and equipment permit us to limit the construction and functioning costs, by reducing, for example, the initial number of pipes, maintenance operations, energy consumption of pumps and valves, number of installed sensors, and amount of water losses due to leakages (Alperovits & Shamir 1977; Gupta *et al.* 1993; Walski *et al.* 2007; Gupta *et al.* 2016; Creaco Walski 2017). To optimize the choice of decision-makers, the use of pressure-driven models of users' consumption is mandatory (Karadirek *et al.* 2012; Berardi *et al.* 2015; Laucelli *et al.* 2017a,b; Monsef *et al.* 2018; Koşucu Demirel 2022).

1.2. Pressure-driven modeling of users' consumption

One approach to model users' consumption is by assuming that the demand of the users is always satisfied. These models are called "demand-driven models" (DDMs); they neglect unsatisfied users' demand due to insufficient pressure. Nevertheless, this solution is implemented in several softwares, such as EPANET 2.0 (Rossman 2000), its object-oriented C++ interface OOTEN (van Zyl *et al.* 2003), and Porteau (Piller *et al.* 2011).

Conversely, the "pressure-driven models" (PDMs) permit us to simulate users' consumptions that depend on the pressure. Different pressure–outflow relationships (PORs) can be used in this kind of model (Wagner *et al.* 1988; Fujiwara & Ganesharajah 1993; Gupta & Bhave 1996; Piller *et al.* 2003; Tanyimboh Templeman 2004). Among them, the Wagner's POR (Wagner *et al.* 1988) was proven to perform best against data (Shirzad *et al.* 2013). Simulators using a POR to model PDM consumptions include the extension of EPANET 2.0 from Siew & Tanyimboh (2012), EPANET 2.2 (Rossman *et al.* 2020), WNTR (Water Network Tool for Resilience) (Klisel *et al.* 2018), and the MATLAB simulator developed by Elhay *et al.* (2016).

Pressure-driven analyses (PDAs) can also be conducted using DDM simulators by adding artificial equipment to the network (e.g., flow and pressure-regulating valves) (Mahmoud *et al.* 2017; Vaidya *et al.* 2023). However, this approach increases the size (i.e., the number of elements) of the system, and can require specific treatments and convergence criteria to ensure the robustness and the stability of the algorithm for every test case (Gorev *et al.* 2021; Sivakumar *et al.* 2023).

The engines and the programmer's toolkit of EPANET 2.2 are multi-platform, but they are non objected-oriented; also, EPANET 2.2's graphical user interface, which makes its engines easier to use, works only on Microsoft Windows operating systems. The WNTR package (Klisel *et al.* 2018) was designed specifically to simulate and analyze resilience of water distribution networks under disaster scenarios; moreover, its software architecture does not permit us to extend it easily, in particular to add new physical processes like pressure-dependent background leakage outflows. Finally, the MATLAB simulator from Elhay *et al.* (2016) is efficient and works fine, but any upgrade in its code will need to be run in MATLAB, a paid-for software package not available to everyone (although interfacing with other languages is possible). In summary, there currently exists no open-source and easy/cheap to extend simulator of PDM consumption.

1.3. The Python language

Python is a free, multi-platform, high-level and object-oriented language (Lutz 2013). Its core philosophy emphasizes readability, making the codes written in Python easy to reuse and extend, even by people who are not software developers. Many efficient libraries for scientific computing are available in Python (Hunter 2007; Chen *et al.* 2008; Hagberg *et al.* 2008; McKinney 2011; Virtanen *et al.* 2020), and Python is today one of the most popular programing languages[1] .

Big efforts have already been made by the community to simulate WDNs with Python. For example, the software of Xing Sela (2020) allows the running of transient simulations using an adapted method of characteristics. For extended-period simulations (EPS), the EPANET-based package WNTR (Klisel *et al.* 2018), and the object-oriented framework OOPNET (Steffelbauer & Fuchs-Hanusch 2015) are stable and active solutions.

1.4. The framework OOPNET

OOPNET (Object-Oriented Python framework for water distribution NETworks analyzes) is a convenient Python API (Application Programing Interface) of EPANET. It is aimed at students, engineers and researchers who want to run hydraulic simulations in a high-level way, apply pre and post-processing on input or output data of EPANET, and make the simulation results from EPANET more readable (Steffelbauer & Fuchs-Hanusch 2015).

OOPNET is based on several state-of-the-art Python libraries for scientific computing (Hunter 2007; Hagberg *et al.* 2008; McKinney 2011; Hold-Geoffroy *et al.* 2014; Hoyer & Hamman 2017). It permits us in particular to parse and convert EPANET's input file to Python objects, run an EPANET simulation through a command line interface, and parse EPANET's output files and convert them to Python objects. OOPNET is free, open-source, object-oriented, and designed to enhance collaborative effort from the community of programmers and users of WDN software. Its source code is publicly accessible on GitHub (https://github.com/oopnet/oopnet). Moreover, its authors and contributors currently work on making OOPNET more flexible, user-focused and efficient through a domain-driven design (DDD) approach (Steffelbauer *et al.* 2022). Thus, it is well suited for prototyping, integrating new physical processes, or testing new hypotheses.

---

[1] https://survey.stackoverflow.co/2022/, https://www.jetbrains.com/lp/devecosystem-2021/, https://octoverse.github.com/2022/top-programing-languages, https://pypl.github.io/PYPL.html, https://spectrum.ieee.org/top-programing-languages-2022.

1.5. Hypothesis, objectives, and research strategy

The consumption model proposed by Elhay *et al.* (2016) seems the most effective one; however, this model is not easy to extend and it needs a MATLAB environment. Thus, our objective is to implement a new simulator that will later permit us to test new physical processes in a more convenient way and to share our developments with a wider community of users and programmers.

We think that using the Python programing language and the framework OOPNET is today the best solution for developing such a simulator. Doing so, we expect to reproduce the results obtained by Elhay *et al.* (2016) with their MATLAB implementation when running PDA, and the ones obtained with EPANET 2.0 for demand-driven analyses (DDAs). It is important to note here that EPANET 2.2 was not available yet at the start of our study; this is the reason why we decided to validate our Python simulator against the MATLAB implementation of Elhay *et al.* (2016) rather than EPANET 2.2 when running PDA. Finally, developing this new Python simulator would permit us to extend in a non-regressive way the current capabilities of OOPNET, which by now allows running EPANET simulations only.

In this paper, we will first describe the steady-state equations and the method to solve them, presenting several numerical enhancements to deal with potential sources of instability. Then, we will explain our Python implementation, the networks to simulate, and the tests and metrics proposed for validation. Finally, we will present our results and discuss them.

2. METHODS

Hereafter, since we do not model any other type of demand and consumption, we will call "user's demand" just "demand", and "user's consumption" simply "consumption".

Also, in all networks that we will simulate, the kinetic energy (a.k.a., velocity head) is negligible compared to the pressure-head (a.k.a., internal pressure energy). Then, the hydraulic head h (called just "head" hereafter) will be calculated as:

$$h = u + p \tag{1}$$

with u the elevation (in m) and p the pressure-head (Walski *et al.* 2007, p. 29). h and p, as well as friction head-losses, will be expressed in mH_2O (metres water column), which are consistent to m. Also, as in Piller *et al.* (2011), all flow rates, demand and consumption will be expressed in $l\,s^{-1}$ rather than in $m^3\,s^{-1}$, to avoid problems of stability due to machine precision.

Finally, and in the absence of other indications, scalar parameters and variables will be denoted in italic (e.g., x), vectors in italic bold (e.g., $\boldsymbol{v}$), matrices in italic bold upper-case (e.g., $\boldsymbol{M}$), scalar functions in upright (e.g., f(x)), vector functions in upright bold (e.g., $\mathbf{f}(\boldsymbol{v})$), matrix functions in upright bold upper-case (e.g., $\mathbf{M}(\boldsymbol{v})$), and sets in blackboard style (e.g., $\mathbb{R}$).

2.1. Friction head-loss

In each pipe, we use the Hazen–Williams formulation (Williams *et al.* 1933) to compute the friction head-loss. To do so, denoting ϕ_p the inside diameter (in mm) of the pipe and $c_{HW} \in \mathbb{R}_*^+$ its roughness coefficient (unit-less), and supposing that the pipe is cylindrical, we first compute the friction coefficient as:

$$f = \frac{10.67}{(1000\,c_{HW})^{\gamma_{HW}}\,(\phi_p/1000)^{4.87}} \tag{2}$$

Then, we compute the Hazen–Williams friction head-loss along the full length ℓ (in m) of the pipe as:

$$\xi_f(q) = f\,\ell\,q\,|q|^{\gamma_{HW}-1} \tag{3}$$

where $q \in \mathbb{R}$ is the algebraic flow rate traversing the pipe, and $\gamma_{HW} = 1.852$ is the Hazen-Williams exponent (unit-less).

2.2. Consumption

At each junction node, we use the Wagner's POR (Wagner *et al.* 1988) to compute the consumption. To do so, denoting $d \in \mathbb{R}^+$ the demand, $p \in \mathbb{R}^+$ the pressure-head, and $p_m \in \mathbb{R}^+$ and $p_s \in \mathbb{R}_*^+$ the fixed minimum and service pressure-heads

(in mH_2O) such that $p_s > p_m$, we compute the fraction of pressure-head (unit-less) as:

$$z(p) = \frac{p - p_m}{p_s - p_m} \tag{4}$$

and the consumption as:

$$c(p) = \begin{cases} 0 & \text{if } p \le p_m \\ d\sqrt{z(p)} & \text{if } p_m < p < p_s \\ d & \text{if } p \ge p_s \end{cases} \tag{5}$$

Usually, minimum and service pressure-heads are chosen respectively equal to 0 and 20 mH_2O (Piller *et al.* 2003; Giustolisi *et al.* 2008, 2014; Elhay *et al.* 2016; Deuerlein *et al.* 2019).

2.3. Equations of equilibrium in a WDN

For any WDN, we denote n_p the number of cylindrical and longitudinal pipes of lengths $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_{n_p})^T \in \mathbb{R}^{n_p}$, n_j the number of junction nodes at which the heads are unknown, n_t and n_r respectively the number of tank and reservoir nodes at which the heads are known and fixed, $n_0 = n_t + n_r$ the total number of source nodes, and $n_N = n_j + n_0$ the total number of nodes. Also, we denote $\boldsymbol{q} = (q_1, \ldots, q_{n_p})^T \in \mathbb{R}^{n_p}$ and $\boldsymbol{h} = (h_1, \ldots, h_{n_j})^T \in \mathbb{R}^{n_j}$ the unknown flow rates in pipes and heads at junctions, $\boldsymbol{h_t} = (h_{t,1}, \ldots, h_{t,n_t})^T \in \mathbb{R}^{n_t}$ and $\boldsymbol{h_r} = (h_{r,1}, \ldots, h_{r,n_r})^T \in \mathbb{R}^{n_r}$ the known and fixed heads at tanks and at reservoirs, $\boldsymbol{h_0} = (\boldsymbol{h_t}^T \mid \boldsymbol{h_r}^T)^T \in \mathbb{R}^{n_0}$ and $\boldsymbol{h_N} = (\boldsymbol{h}^T \mid \boldsymbol{h_0}^T)^T \in \mathbb{R}^{n_N}$ the heads at all source nodes and at all nodes, $\boldsymbol{p} = (p_1, \ldots, p_{n_j})^T \in \mathbb{R}^{n_j}$ and $\boldsymbol{l_t} = (l_1, \ldots, l_{n_t})^T \in \mathbb{R}^{n_t}$ the pressure-heads at junctions and the water levels at tanks, and $\boldsymbol{u} = (u_1, \ldots, u_{n_j})^T \in \mathbb{R}^{n_j}$, $\boldsymbol{u_t} = (u_1, \ldots, u_{n_t})^T \in \mathbb{R}^{n_t}$ and $\boldsymbol{u_r} = (u_1, \ldots, u_{n_r})^T \in \mathbb{R}^{n_r}$ the elevations of junctions, tanks' bottom, and reservoirs' surface, respectively. Then, we have the relations:

$$\boldsymbol{p} = \boldsymbol{h} - \boldsymbol{u} \tag{6}$$

at junctions,

$$\boldsymbol{l_t} = \boldsymbol{h_t} - \boldsymbol{u_t} \tag{7}$$

at tanks, and

$$\boldsymbol{h_r} = \boldsymbol{u_r} \tag{8}$$

at reservoirs. Next, we denote the $n_j \times n_p$ junction-pipe, the $n_t \times n_p$ tank-pipe, and the $n_r \times n_p$ reservoir-pipe incidence matrices as respectively $\boldsymbol{A}$, $\boldsymbol{A_t}$ and $\boldsymbol{A_r}$, the $n_0 \times n_p$ source-pipe incidence matrix as $\boldsymbol{A_0} = (\boldsymbol{A_t}^T \mid \boldsymbol{A_r}^T)^T$, the $n_N \times n_p$ node-pipe incidence matrix as $\boldsymbol{A_N} = (\boldsymbol{A}^T \mid \boldsymbol{A_0}^T)^T$, and $\boldsymbol{\xi_f}(\boldsymbol{q}) = (\xi_{f,1}(q_1), \ldots, \xi_{f,n_p}(q_{n_p}))^T$ the vector function of $\mathbb{R}^{n_p}$ to $\mathbb{R}^{n_p}$ to compute the friction head-losses in pipes. $\forall k \in \{1, \ldots, n_p\}$, we compute the friction head-loss $\xi_{f,k}$ along the full pipe k from Equation (3), as:

$$\xi_{f,k}(q_k) = f_k \, \ell_k \, q_k \, |q_k|^{\gamma_{HW}-1} \tag{9}$$

Finally, we denote $\boldsymbol{d} = (d_1, \ldots, d_{n_j})^T \in \mathbb{R}^{n_j}$ the fixed demand at junctions, and $\mathbf{c}(\boldsymbol{h}) = (c_1(h_1), \ldots, c_{n_j}(h_{n_j}))^T$ the vector function of $\mathbb{R}^{n_j}$ to $\mathbb{R}^{n_j}$ to compute consumptions. $\forall i \in \{1, \ldots, n_j\}$, $c_i(h_i)$ is computed from Equation (5) as:

$$c_i(h_i) = \begin{cases} 0 & \text{if } h_i \le u_i + p_m \\ d_i\sqrt{z(h_i - u_i)} & \text{if } u_i + p_m < h_i < u_i + p_s \\ d_i & \text{if } h_i \ge u_i + p_s \end{cases} \tag{10}$$

Remark:

In this study, for the sake of simplicity, we choose the same p_s for all junctions. However, it presents no difficulty to set specific values for some junctions, to describe for example pressure-independent fire demands as in Sivakumar *et al.* (2023).

To find the unknown flow rates $\boldsymbol{q}$ and heads $\boldsymbol{h}$ in the WDN at steady-state, we solve the non-linear system of equations:

$$\boldsymbol{\rho}(\boldsymbol{q}, \boldsymbol{h}) = \begin{pmatrix} \boldsymbol{\xi}_{\mathrm{f}}(\boldsymbol{q}) - \boldsymbol{A}^T\boldsymbol{h} - \boldsymbol{A}_0^T\boldsymbol{h}_0 \\ -\boldsymbol{A}\,\boldsymbol{q} - \mathbf{c}(\boldsymbol{h}) \end{pmatrix} = \boldsymbol{0} \tag{11}$$

where

$$\boldsymbol{\xi}_{\mathrm{f}}(\boldsymbol{q}) - \boldsymbol{A}^T\boldsymbol{h} - \boldsymbol{A}_0^T\boldsymbol{h}_0 = \boldsymbol{\rho}_e \tag{12}$$

are the energy residuals in pipes, and

$$-\boldsymbol{A}\,\boldsymbol{q} - \mathbf{c}(\boldsymbol{h}) = \boldsymbol{\rho}_m \tag{13}$$

are the mass residuals at junctions. Then, the $\boldsymbol{h}$ found satisfies the principle of the conservation of energy, corresponding, in any pipe k of the network, to the Bernoulli's equation:

$$u_i + p_i = u_j + p_j + \xi_{f,k} \tag{14}$$

where $\xi_{f,k}$ is the friction head-loss along k, and $\{u_i, u_j\}$ and $\{p_i, p_j\}$ are respectively the elevations and the pressure-heads at the nodes i and j located at the extremities of k; as a reminder, the kinetic energy is not represented in Equation (14) because it is negligible compared to the pressure-head (Walski *et al.* 2007, p. 29). Also, the flow rates $\boldsymbol{q}$ satisfy the conservation of the mass at nodes, derived from the Kirchhoff's first law: "the algebraic sum of the branch currents towards any node of an electric network is zero" (IEC 2023).

2.4. Solution algorithm

We solve the system (11) with a Newton's method that includes a system reduction.

2.4.1. Newton's method

First, we define the Jacobian matrix function of the system (11) as:

$$\mathrm{J} = \begin{pmatrix} \mathrm{J}_{11} & \mathrm{J}_{12} \\ \mathrm{J}_{21} & \mathrm{J}_{22} \end{pmatrix} \tag{15}$$

where

$$\mathbf{J_{11}} = \frac{\partial \boldsymbol{\xi}_{\mathbf{f}}}{\partial \mathbf{q}}, \quad \mathbf{J_{12}} = -\mathbf{A^T}, \quad \mathbf{J_{21}} = -\mathbf{A} \quad \text{and} \quad \mathbf{J_{22}} = -\frac{\partial \mathbf{c}}{\partial \mathbf{h}}. \tag{16}$$

Then, denoting:

- $\boldsymbol{q}^{(0)}$ the initial guesses of flow rates in pipes, and $\boldsymbol{h}^{(0)}$ the ones of heads at junctions,
- and $\boldsymbol{J}^{(m)} = \mathbf{J}(\boldsymbol{q}^{(m)}, \boldsymbol{h}^{(m)})$ the Jacobian matrix of $\boldsymbol{\rho}^{(m)} = \boldsymbol{\rho}(\boldsymbol{q}^{(m)}, \boldsymbol{h}^{(m)})$ at iteration m,

and supposing that $\boldsymbol{J}^{(m)}$ is invertible, the Newton's method consists of computing, at each iteration $m = 0, 1, 2, \ldots$, the flow rates $\boldsymbol{q}^{(m+1)}$ and heads $\boldsymbol{h}^{(m+1)}$ as

$$\begin{pmatrix} \boldsymbol{q}^{(m+1)} \\ \boldsymbol{h}^{(m+1)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{q}^{(m)} \\ \boldsymbol{h}^{(m)} \end{pmatrix} - (\boldsymbol{J}^{(m)})^{-1} \begin{pmatrix} \boldsymbol{\rho}_e{}^{(m)} \\ \boldsymbol{\rho}_m{}^{(m)} \end{pmatrix} \tag{17}$$

until the differences between two successive iterates become less than a given tolerance.

2.4.2. System reduction

In Equation (17), computing the iterates $\boldsymbol{q}^{(m+1)}$ and $\boldsymbol{h}^{(m+1)}$ by inverting numerically the potentially very large matrix $\boldsymbol{J}^{(m)} \in \mathbb{R}^{(n_p+n_j)\times(n_p+n_j)}$ can take much computational time. Thus, we rather look for the descent directions on $\boldsymbol{q}^{(m)}$ and $\boldsymbol{h}^{(m)}$, defined respectively as

$$\boldsymbol{\delta}_{\boldsymbol{q}}^{(m)} = \boldsymbol{q}^{(m+1)} - \boldsymbol{q}^{(m)} \quad \text{and} \quad \boldsymbol{\delta}_{\boldsymbol{h}}^{(\mathrm{m})} = \boldsymbol{h}^{(\mathrm{m}+1)} - \boldsymbol{h}^{(\mathrm{m})} \tag{18}$$

and which satisfy the linear system:

$$(-\boldsymbol{J}^{(m)}) \begin{pmatrix} \boldsymbol{\delta}_{\boldsymbol{q}}^{(m)} \\ \boldsymbol{\delta}_{\boldsymbol{h}}^{(m)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\rho_e}^{(m)} \\ \boldsymbol{\rho_m}^{(m)} \end{pmatrix} \tag{19}$$

or, in developed form:

$$\begin{pmatrix} \mathbf{J}_{\mathbf{11}}^{(\boldsymbol{m})} \boldsymbol{\delta}_{\boldsymbol{q}}^{(m)} + \mathbf{J}_{\mathbf{12}}^{(\boldsymbol{m})} \boldsymbol{\delta}_{\boldsymbol{h}}^{(m)} \\ \mathbf{J}_{\mathbf{21}}^{(\boldsymbol{m})} \boldsymbol{\delta}_{\boldsymbol{q}}^{(m)} + \mathbf{J}_{\mathbf{22}}^{(\boldsymbol{m})} \boldsymbol{\delta}_{\boldsymbol{h}}^{(m)} \end{pmatrix} = -\begin{pmatrix} \boldsymbol{\rho_e}^{(m)} \\ \boldsymbol{\rho_m}^{(m)} \end{pmatrix} \tag{20}$$

To solve the system (20), we first use its first row to express $\boldsymbol{\delta}_{\boldsymbol{q}}^{(m)}$ in function of $\boldsymbol{\delta}_{\boldsymbol{h}}^{(m)}$:

$$\boldsymbol{\delta}_{\boldsymbol{q}}^{(m)} = -(\boldsymbol{J}_{\boldsymbol{11}}^{(m)})^{-1}(\boldsymbol{J}_{\boldsymbol{12}}^{(m)} \boldsymbol{\delta}_{\boldsymbol{h}}^{(m)} + \boldsymbol{\rho_e}^{(m)}) \tag{21}$$

assuming that the diagonal matrix $\boldsymbol{J}_{\boldsymbol{11}}^{(m)} \in \mathbb{R}^{n_p \times n_p}$ is invertible. Next, we substitute (21) into the second equation of the system (20) to obtain:

$$\boldsymbol{S}^{(m)} \boldsymbol{\delta}_{\boldsymbol{h}}^{(m)} = \boldsymbol{\rho_m}^{(m)} - \boldsymbol{J}_{\boldsymbol{21}}^{(m)} (\boldsymbol{J}_{\boldsymbol{11}}^{(m)})^{-1} \boldsymbol{\rho_e}^{(m)} \tag{22}$$

where $\boldsymbol{S}^{(m)} \in \mathbb{R}^{n_j \times n_j}$ is the Schur complement of the block $-\boldsymbol{J}_{\boldsymbol{11}}^{(m)}$ of the matrix $-\boldsymbol{J}^{(m)}$, and is defined as:

$$\boldsymbol{S}^{(m)} = \boldsymbol{J}_{\boldsymbol{21}}^{(m)} (\boldsymbol{J}_{\boldsymbol{11}}^{(m)})^{-1} \boldsymbol{J}_{\boldsymbol{12}}^{(m)} - \boldsymbol{J}_{\boldsymbol{22}}^{(m)} \tag{23}$$

In (23), $\boldsymbol{J}_{\boldsymbol{21}}^{(m)} (\boldsymbol{J}_{\boldsymbol{11}}^{(m)})^{-1} \boldsymbol{J}_{\boldsymbol{12}}^{(m)}$ is symmetric and J_{22} is diagonal. Also, supposing that none of the flow rates in vector $\boldsymbol{q}^{(m)}$ are zero, then both $\boldsymbol{J}_{\boldsymbol{21}}^{(m)} (\boldsymbol{J}_{\boldsymbol{11}}^{(m)})^{-1} \boldsymbol{J}_{\boldsymbol{12}}^{(m)}$ and $-\boldsymbol{J}_{\boldsymbol{22}}$ are positive-definite. Thus, $\boldsymbol{S}^{(m)}$ is symmetric and positive-definite, and we can use a Cholesky factorization to compute $\boldsymbol{\delta}_{\boldsymbol{h}}^{(m)}$ of Equation (22). Once $\boldsymbol{\delta}_{\boldsymbol{h}}^{(m)}$ is calculated, we can easily compute $\boldsymbol{\delta}_{\boldsymbol{q}}^{(m)}$ from Equation (21) because $\boldsymbol{J}_{\boldsymbol{11}}^{(m)}$ is diagonal and supposed invertible.

Finally, after computing both $\boldsymbol{\delta}_{\boldsymbol{h}}^{(m)}$ and $\boldsymbol{\delta}_{\boldsymbol{q}}^{(m)}$, we can replace

$$-(\boldsymbol{J}^{(m)})^{-1} \begin{pmatrix} \boldsymbol{\rho_e}^{(m)} \\ \boldsymbol{\rho_m}^{(m)} \end{pmatrix} \quad \text{by} \quad \begin{pmatrix} \boldsymbol{\delta}_{\boldsymbol{q}}^{(\mathrm{m})} \\ \boldsymbol{\delta}_{\boldsymbol{h}}^{(\mathrm{m})} \end{pmatrix}$$

in Equation (17), and compute the new iterates $\boldsymbol{q}^{(m+1)}$ and $\boldsymbol{h}^{(m+1)}$ as

$$\begin{pmatrix} \boldsymbol{q}^{(m+1)} \\ \boldsymbol{h}^{(m+1)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{q}^{(m)} \\ \boldsymbol{h}^{(m)} \end{pmatrix} + \begin{pmatrix} \boldsymbol{\delta}_{\boldsymbol{q}}^{(m)} \\ \boldsymbol{\delta}_{\boldsymbol{h}}^{(m)} \end{pmatrix}. \tag{24}$$

2.4.3. Initial guesses

Similarly to Elhay *et al.* (2016) and $\forall k \in \{1, \ldots, n_p\}$, the initial guess of the flow rate in the pipe k is chosen consistent with a velocity $v_k = 0.5\,\mathrm{m\,s^{-1}}$:

$$q_k^{(0)} = v_k\, \pi \frac{\phi_{p,k}^2}{4} \times 1000 \tag{25}$$

Also, $\forall i \in \{1, \ldots, n_j\}$, we set the initial guess of the head at i as:

$$h_i^{(0)} = p_m + u_i + \frac{p_s - p_m}{5} \tag{26}$$

using the same notation as in Sections 2.2 and 2.3.

2.4.4. Convergence criterion

To test for the convergence of scheme (24), we use a stop criterion that is almost the same as the one already used by Elhay *et al.* (2016). Indeed, we consider that the convergence is reached as soon as, $\forall \boldsymbol{y} \in \{\boldsymbol{q}, \boldsymbol{h}\}$,

$$\begin{cases} \frac{\|\boldsymbol{y}^{(m+1)} - \boldsymbol{y}^{(m)}\|_\infty}{\|\boldsymbol{y}^{(m+1)}\|_\infty} \leq 10^{-6} & \text{if} \|\boldsymbol{y}^{(\mathrm{m}+1)}\|_\infty \geq 10^{-6} \\ \|\boldsymbol{y}^{(m+1)} - \boldsymbol{y}^{(m)}\|_\infty \leq 10^{-6} & \text{otherwise} \end{cases} \tag{27}$$

The only difference with Elhay *et al.* (2016) is that our version of the criterion also handles the case where $\|\boldsymbol{y}^{(m+1)} - \boldsymbol{y}^{(m)}\|_\infty \leq 10^{-6}$. We discuss briefly the choice of the criterion in the Supplementary Material.

Other convergence criteria are used in WDN simulators. For example, the one in EPANET 2.0 checks that the 1-norm of flow changes divided by the 1-norm of flows is less than a given tolerance (Rossman 2000, p. 94). EPANET 2.2 improves this criterion by considering also the ∞-norm of flow changes and the ∞-norm of energy residuals (Rossman *et al.* 2020, p. 116). Then, Gorev *et al.* (2022) enhances EPANET 2.2's criterion by checking for flow residual in each pipe too. Finally, Sivakumar *et al.* (2023) add one more criterion that verifies the residuals on the inverse of the POR (e.g., the Wagner's POR) to improve the convergence of EPANET 2.2's solution algorithm. In our study, we don't need the criterion proposed by Sivakumar *et al.* (2023) because our formulation does not consider the POR inverse function. Therefore, we choose to simply reuse the criterion from Elhay *et al.* (2016), since preliminary tests showed good performances, and to be consistent with the other technical choices made by Elhay *et al.* (2016).

2.5. Sources of instabilities

Different sources of instabilities can hinder the convergence of the solution algorithm described in Section 2.4. We present them below, along with numerical enhancements to deal with them. These numerical enhancements are of primary importance to ensure the convergence of the solution algorithm in many common situations. However, to preserve readability we choose to describe them in the Supplementary Material.

Remark:

The numerical enhancements presented in Sections 2.5.1 – 2.5.3 are the same as the ones implemented in the Porteau software (Piller *et al.* 2011).

2.5.1. Pipes with zero flow rate

At each iteration of the solution algorithm and for each pipe $k \in \{1, \ldots, n_p\}$, the element $[\boldsymbol{J_{11}}]_{kk}$ of the Jacobian matrix is equal to the derivative of the friction head-loss, $d\xi_{f,k}/dq_k$, which can be computed as:

$$\frac{d\xi_{f,k}}{q_k}(q_k) = \gamma_{HW} f_k \ell_k |q_k|^{\gamma HW - 1}. \tag{28}$$

But Equation (28) gives 0 for $q_k = 0$. Thus, since we need to compute the inverse of $[\boldsymbol{J_{11}}]_{kk}$ at each iteration (see Equations (21), (20) and (23)), the use of Equation (28) to compute $d\xi_{f,k}/dq_k$ could lead to a division by zero. To prevent such error, we

choose, like Piller (1995), to regularize the friction head-loss and its derivative for q_k close to 0, by respectively a cubic and a quadratic polynomial.

2.5.2. Junction nodes with pressure-head close to the minimum or the service pressure-head

At each iteration of the solution algorithm and for each pipe $k \in \{1, \ldots, n_p\}$, $[\boldsymbol{J}_{22}]_{ii}$ is equal to dc_i/dh_i, which can be computed as:

$$\frac{\mathrm{d}c_i}{\mathrm{d}h_i}(h_i) = \left(\frac{\mathrm{d}c_i p_i}{\mathrm{d}p_i h_i}\right)(h_i) \tag{29}$$

where, $\forall h_i \in \mathbb{R}^+$

$$\frac{\mathrm{d}p_i}{\mathrm{d}h_i}(h_i) = 1 \tag{30}$$

and

$$\frac{\mathrm{d}c_i}{\mathrm{d}p_i}(p_i) = \begin{cases} d_i \dfrac{1}{p_s - p_m} \dfrac{1}{2\sqrt{\mathrm{z}(p_i)}} = \dfrac{1}{2(p_s - p_m)} \dfrac{\mathrm{c}_i(p_i)}{\mathrm{z}(p_i)} & \text{if} \; p_m < p_i < p_s \\ 0 & \text{otherwise} \end{cases} \tag{31}$$

with $\mathrm{z}(p_i)$ and $\mathrm{c}_i(p_i)$ defined by Equations (4) and (5). However, Equation (31) is undefined at $p_i = p_m$ and discontinuous at $p_i = p_s$. Thus, using Equation (31) to compute dc_i/dp_i could lead to convergence failure when p_i is close to p_m or p_s. To make Equation (31) continuous $\forall p_i \in \mathbb{R}$, we choose, like Piller *et al.* (2003), to regularize the consumption and its derivative by respectively a cubic and a quadratic polynomial, when p_i is close to p_m and p_s.

2.5.3. Initial guesses far from the solution and/or objective function with (sub-)linear gradient growth

The bottleneck of the solution algorithm is the solving of the system (22). But the algorithm can need numerous iterations (and therefore numerous solving of system (22)) if the initial guesses of the flow rates and heads at junctions are far from the solutions at equilibrium. In this case, the simulation of large WDNs can become unfeasible in reasonable computational time. Moreover, the convergence of the algorithm is no longer guaranteed if the gradient of the objective function (11) to minimize does not have a super-linear growth (Piller *et al.* 2017). Thus, to reduce the number of iterations needed by the solution algorithm and to guarantee its convergence for any network configuration, we choose to reuse the damping method proposed by Elhay *et al.* (2016) (see description of the method in the Supplementary Material).

Remark:

Some formulations of Equation (11) do not need damping corrections because they consider the inverse of the POR function rather than the POR function (Todini *et al.* 2021). However, to achieve full convergence, these formulations require a supplementary criterion on the inverse POR equation (Sivakumar *et al.* 2023).

2.5.4. WDN with highly contrasted values of flow rates and/or heads

In general, at each iteration of a solution algorithm based on Newton's method, the magnitude of the elements in the Jacobian matrix can vary strongly from one part of a network to another, according to demands, pipe resistances, tank levels, use of pumps and/or valves, etc. In the present study, when demands and/or pipe resistances differ by several orders of magnitude across the network, using the Jacobian matrix $\boldsymbol{J}$ as it can cause instabilities and increase significantly the number of iterations needed to converge. To limit the difference of magnitude order between all elements of $\boldsymbol{J}$, we choose to extend the preconditioning method initially proposed by Elhay & Simpson (2011). Indeed, in Elhay & Simpson (2011), the method dealt with consumptions that do not depend on the pressures; thus, we extend it to now deal with pressure-dependent consumptions too. Using this preconditioning method, the solution algorithm becomes a quasi-Newton method.

Gorev *et al.* (2022) propose an alternative solution to deal with zero flows and low-resistance pipes in EPANET 2.2. To do so, they replace the head-loss by a linear function in the pipes where the flow rate becomes very small, avoiding the appearance of zero head-loss derivatives in the Jacobian matrix. This regularization approach preserves the convergence rate of the

Newton method, which is generally better than that of quasi-Newton methods. Nevertheless, it also changes slightly the head-loss behavior (for low flow rates), it uses an absolute threshold flow rate for the cutoff of the derivatives (while the preconditioning method from Elhay & Simpson (2011) is based on the difference of magnitude order between the elements of the Jacobian), and it needs some additional checking computations. Thus, both the Elhay & Simpson (2011) and Gorev *et al.* (2022) approaches have their own advantages and limitations. We choose to use the one from Elhay & Simpson (2011) because we master it better.

2.6. Implementation and framework

To implement our Python simulator, we use the Python programing language[2] through its Anaconda distribution[3]. In addition to the Python's standard library[4], we use several third-party libraries:

- NetworkX (Hagberg *et al.* 2008), for the creation and manipulation of the graphed data structures of the WDNs,
- NumPy (Harris *et al.* 2020), for the creation and manipulation of the vectors and basic linear algebra operations (e.g., computation of vector norms),
- SciPy (Virtanen *et al.* 2020), to find the coefficients of the polynomials defined in the Supplementary Material[5], and to handle sparse matrices[6],
- Scikit-Sparse[7], which itself wraps the CHOLMOD routine (Chen *et al.* 2008), to compute the Cholesky decomposition of the Schur complement matrix,
- XArray (Hoyer & Hamman 2017), to store the outputs in multi-dimensional labeled arrays,
- Pandas (McKinney 2011), to convert the outputs to tabular data structures, to post-process them, and to write them to comma-separated values (CSV) files,
- Matplotlib (Hunter 2007), to plot the outputs,
- and Sphinx[8], to extract the documentation from the source code, and generate an HTML file from it.

Our Python simulator is integrated into the framework OOPNET (Steffelbauer & Fuchs-Hanusch 2015), in the branch *simulators* (https://github.com/oopnet/oopnet/tree/simulators); its integration into the branch *main* is in progress. We use OOPNET in particular to parse and convert EPANET's input and output files to Python objects.

2.7. Simulated networks

To test our Python simulator, we simulate the set of networks $\mathbb{S}^{net} = \{\text{N1}, \ldots, \text{N8}\}$, which numbers of nodes and pipes are presented in Table 1. EPANET's input files of networks {N1, N3, N4, N7} are freely downloadable from the American Society of Civil Engineers (ASCE)'s library[9]; they are plotted on Figure 1. Other networks {N2, N5, N6, N8} can not be plotted here, because of ownership and/or security concerns.

2.8. Tests and metrics

2.8.1. Functioning and performances of the Python simulator

To verify the good functioning of the Python simulator, we simulate the set of networks $\mathbb{S}^{net}$ presented in Section 2.7, multiplying the peak demand by a multiplier $\mu_d \in \mathbb{S}^{\mu_d} = \{1, 2, 3, 5\}$. Like so, we test the Python simulator for different levels of demand satisfaction (in %), defined for each multiplier μ_d and at any junction $i \in \{1, \ldots, n_j\}$ as:

$$d_{s,\mu_d,i} = \frac{c_{\mu_d,i}}{d_{\mu_d,i}} \times 100 \tag{32}$$

---

[2] https://docs.python.org/3/.
[3] https://www.anaconda.com/.
[4] https://docs.python.org/3/library/.
[5] https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.root.html.
[6] https://docs.scipy.org/doc/scipy/reference/sparse.html.
[7] https://scikit-sparse.readthedocs.io/.
[8] https://www.sphinx-doc.org/.
[9] www.ascelibrary.org.

Table 1 | Number of pipes n_p, junctions n_j and reservoirs n_r in the networks {N1, ..., N8} used to test our Python simulator

Network	n_p	n_j	n_r
N_1	934	848	8
N_2	1,118	1,039	2
N_3	1,976	1,770	4
N_4	2,465	1,890	3
N_5	2,508	2,443	2
N_6	8,584	8,392	2
N_7	14,830	12,523	7
N_8	19,647	17,971	15

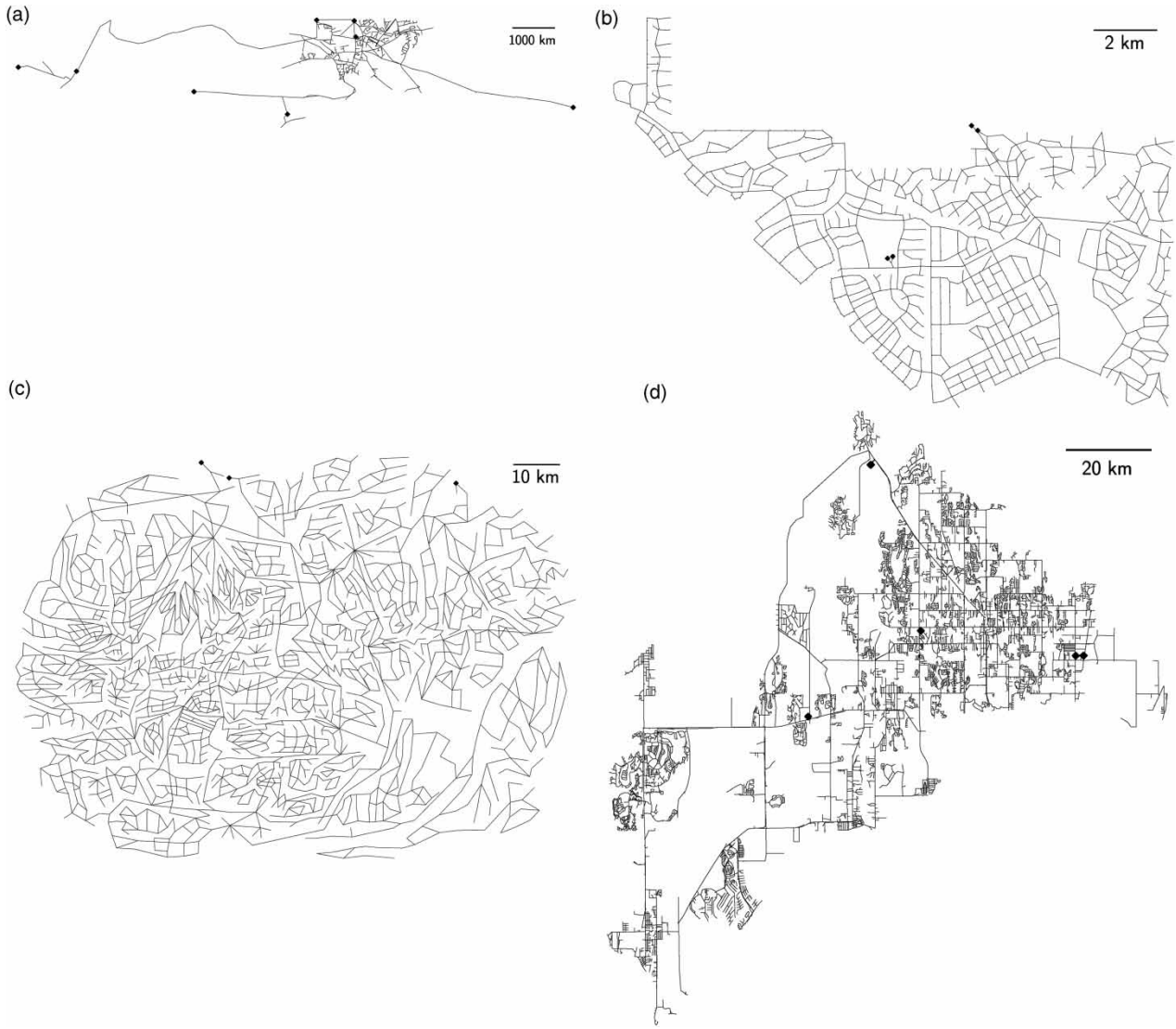


Figure 1 | Networks {N1, N3, N4, N7} used to test our Python simulator. Reservoirs are represented by ◆.

For all these simulations, the minimum and service pressure-heads at all junctions are chosen respectively equal to 0 and 20 mH_2O.

From the outputs of the simulations, we first check that the ∞-norm of the residuals is less than 10^{-3} (in $l\,s^{-1}$ if the greatest absolute value is a mass residual, or in mH_2O if it is an energy one). If so, we then assume that the solver reaches the equilibrium state with enough precision.

Next, we check that the cumulated demand and consumption (in $l\,s^{-1}$), defined for each demand multiplier $\mu_d \in \mathbb{S}^{\mu_d}$ as, respectively:

$$d_{\mu_d}^{cumul} = \sum_{j \in \mathbb{S}^{net}} \left(\sum_{i \in \{1,\ldots,n_i^j\}} d_{\mu_d,i}^{j} \right) \quad \text{and} \quad \mathrm{c}_{\mu_\mathrm{d}}^{\mathrm{cumul}} = \sum_{\mathrm{j} \in \mathbb{S}^{\mathrm{net}}} \left(\sum_{\mathrm{i} \in \{1,\ldots,\mathrm{n_i^j}\}} \mathrm{c}_{\mu_\mathrm{d},\mathrm{i}}^{\mathrm{j}} \right) \tag{33}$$

both increase with μ_d. Also, we check that the cumulated demand increases quicker than the cumulated consumption, and that the global demand satisfaction, defined for each demand multiplier $\mu_d \in \mathbb{S}^{\mu_d}$ as:

$$d_{s,\mu_d}^{glob} = \frac{c_{\mu_d}^{cumul}}{d_{\mu_d}^{cumul}} \times 100 \tag{34}$$

decreases with μ_d.

Next, to assess the efficiency of the Python simulator and of the solution algorithm, we compute, for each network $j \in \mathbb{S}^{net}$, the mean central processing unit (CPU) time elapsed during all runs with all $\mu_d \in \mathbb{S}^{\mu_d}$, first considering the whole simulations (i.e., pre-processing, solver and post-processing):

$$\overline{t_{CPU}^{simu,j}} = \frac{1}{\mathrm{card}(\mathbb{S}^{\mu_d})} \sum_{\mu_d \in \mathbb{S}^{\mu_d}} t_{CPU,\mu_d}^{simu,j} \tag{35}$$

and then considering only the times needed by the solver to reach convergence:

$$\overline{t_{CPU}^{solv,j}} = \frac{1}{\mathrm{card}(\mathbb{S}^{\mu_d})} \sum_{\mu_d \in \mathbb{S}^{\mu_d}} t_{CPU,\mu_d}^{solv,j} \tag{36}$$

where $\mathrm{card}(\mathbb{S}^{\mu_d})$ is the cardinality of the set $\mathbb{S}^{\mu_d}$. We expect logically the elapsed CPU times to increase with the size of the networks. Also, since the critical part of the solver (i.e., the Cholesky decomposition of the Schur complement matrix) uses the optimized CHOLMOD compiled library (Chen *et al.* 2008), we expect that the times needed by the solver increase slower than those needed for the whole simulations when larger networks are simulated. To check this point, we compute, for each network $j \in \mathbb{S}^{net}$, the ratio of time needed by the solver when compared to the times needed by the whole simulations, defined as:

$$\delta \overline{t_{CPU}^{solv,j}} = \frac{\overline{t_{CPU}^{solv,j}}}{\overline{t_{CPU}^{simu,j}}} \times 100 \tag{37}$$

We implement several numerical enhancements to deal with potential sources of instabilities (see Section 2.5). To quantify the gain of these enhancements, we simulate each network $j \in \mathbb{S}^{net}$ for each demand multiplier $\mu_d \in \mathbb{S}^{\mu_d}$, first with no numerical enhancement, and next adding successively each numerical enhancement. Hereafter, we denote this set of enabled enhancements as $\mathbb{S}^{enh} = \{\text{None}, \text{Head} - \text{loss}, +\text{Consumption}, +\text{Damping}, \text{Preconditioning}\}$ ("+" means "in addition to all previously enabled enhancement(s)"). We then compute, for each run, the convergence order of the solution algorithm when it converges. To do so, for each enabled enhancement(s) $i \in \mathbb{S}^{enh}$, each network $j \in \mathbb{S}^{net}$ and each demand multiplier $\mu_d \in \mathbb{S}^{\mu_d}$, denoting $\boldsymbol{\rho}_{\boldsymbol{\mu_d}}^{\boldsymbol{i,j}(m)} \in \mathbb{R}^{n_p+n_j}$ the residuals at the iteration m of the solution algorithm, we compute the difference between the ∞-norm of $\boldsymbol{\rho}_{\boldsymbol{\mu_d}}^{\boldsymbol{i,j}(m)}$ and $\boldsymbol{\rho}_{\boldsymbol{\mu_d}}^{\boldsymbol{i,j}(m+1)}$, $\forall m \in \{0, \ldots, m_{c,\mu_d}^{i,j}\}$, where $m_{c,\mu_d}^{i,j}$ is the number of iterations needed to reach

convergence, as:

$$\varepsilon_{\mu_d}^{i,j\,(m)} = \|\boldsymbol{\rho}_{\boldsymbol{\mu_d}}^{\boldsymbol{i,j}\,(m)}\|_\infty - \|\boldsymbol{\rho}_{\boldsymbol{\mu_d}}^{\boldsymbol{i,j}\,(m+1)}\|_\infty \tag{38}$$

The convergence order $\eta_{\mu_d}^{i,j}$ then satisfies the relation:

$$\varepsilon^{(m)} = a_{\mu_d}^{i,j}\, m^{\eta_{\mu_d}^{i,j}} \tag{39}$$

with $a_{\mu_d}^{i,j}$ a constant. After rewriting Equation (39) in $\log - \log$ scale as:

$$\log(\varepsilon_{\mu_d}^{i,j\,(m)}) = \log(a_{\mu_d}^{i,j}) + \eta_{\mu_d}^{i,j} \log(m) \tag{40}$$

we then compute $\eta_{\mu_d}^{i,j}$ by linear regression of Equation (40).

The $\varepsilon_{\mu_d}^{i,j\,(m)}$ computed by Equation (38) can sometimes be negative, especially at $m = 1$ or at an iteration that follows a damping correction. Thus, to avoid error in the computation of $\eta_{\mu_d}^{i,j}$ through Equation (40), we do not consider these negative $\varepsilon_{\mu_d}^{i,j\,(m)}$, and we use as a previous iteration the last iteration m' such that $\varepsilon_{\mu_d}^{i,j\,(m')} > 0$. Equation (38) then becomes:

$$\varepsilon_{\mu_d}^{i,j\,(m)} = \|\boldsymbol{\rho}_{\boldsymbol{\mu_d}}^{\boldsymbol{i,j}\,(m')}\|_\infty - \|\boldsymbol{\rho}_{\boldsymbol{\mu_d}}^{\boldsymbol{i,j}\,(m+1)}\|_\infty \tag{41}$$

Finally, from all $\eta_{\mu_d}^{i,j}$, we compute, for each enabled enhancement(s) $i \in \mathbb{S}^{enh}$, the mean convergence order as:

$$\overline{\eta^i} = \frac{1}{\mathrm{card}(\mathbb{S}^{net})\mathrm{card}(\mathbb{S}^{\mu_d})} \sum_{j \in \mathbb{S}^{net}} \sum_{\mu_d \in \mathbb{S}^{\mu_d}} \eta_{\mu_d}^{i,j} \tag{42}$$

2.8.2. Comparison of the Python simulator with the MATLAB one

To validate the ability of our Python simulator to run pressure-driven analyses (PDA), we compare the consumptions computed by our simulator with the ones obtained by Elhay *et al.* (2016) from their MATLAB implementation, for the same sets of networks $\mathbb{S}^{net}$ and demand multipliers $\mathbb{S}^{\mu_d}$, and the same minimum and service pressure-heads at all junctions: $p_m = 0$ and $p_s = 20\,\mathrm{mH_2O}$. Then, we assume that the consumptions from both Python and MATLAB simulators are the same as soon as they all differ by less than $10^{-2}\,\mathrm{l\,s^{-1}}$. Supposing that the average flow rate in the tested networks is equal to $10\,\mathrm{l\,s^{-1}}$, then $10^{-2}\,\mathrm{l\,s^{-1}}$ represents a precision of ~ 1, which corresponds to 1/10th of the best precision expected when measuring flow rates physically in a WDN (Itron 2021, p. 3).

Remark:

The MATLAB simulator of Elhay *et al.* (2016) implements the numerical enhancements described in Sections 2.5.1 to 2.5.3.

To compare the performances of the Python simulator with those of the MATLAB simulator, we compute, in a similar way as in Section 2.8.1 but for each network $j \in \mathbb{S}^{net}$, the mean convergence orders associated with each simulator, as:

$$\overline{\eta^j} = \frac{1}{\mathrm{card}(\mathbb{S}^{\mu_d})} \sum_{\mu_d \in \mathbb{S}^{\mu_d}} \eta_{\mu_d}^{j} \tag{43}$$

where $\eta_{\mu_d}^{j}$ is the convergence order obtained when simulating the network j with the demand multiplier μ_d.

Also, to check that the orders of convergence are not biased by greater numbers of damping corrections for one of the simulators, we compute, for each network $j \in \mathbb{S}^{net}$, the mean numbers of damping corrections associated with each simulator as:

$$\overline{n_{corr}^j} = \frac{1}{\mathrm{card}(\mathbb{S}^{\mu_d})} \sum_{\mu_d \in \mathbb{S}^{\mu_d}} n_{corr,\mu_d}^{j} \tag{44}$$

where n_{corr,μ_d}^{j} is the number of damping corrections obtained when simulating the network j with the demand multiplier μ_d. Since the damping algorithm implemented in both simulators is the same, we expect the simulators to need (more or less) the same number of damping corrections.

Finally, it is not possible to compare the process times elapsed using each simulator, because the MATLAB simulator was run by Elhay *et al.* (2016) on a different machine than the one we use to run our Python simulator.

2.8.3. Comparison of the Python simulator with EPANET 2.0

To check that our Python simulator is also able to simulate pressure-independent consumptions, we simulate the same set of networks $\mathbb{S}^{net}$ with EPANET 2.0[10] , increasing sufficiently the heads at the source nodes $\boldsymbol{h_0}$, and reducing enough the service pressure-head p_s, so that the demand becomes satisfied at every junction. Then, we verify, at each junction of each network, that the consumption computed by the Python simulator and the one computed by EPANET 2.0 differ by less than $10^{-2}\,\mathrm{l\,s^{-1}}$.

Finally, to compare the performances of the Python simulator with those of EPANET 2.0, we compute, for both simulators and for each network, the ∞-norms of the residuals at convergence, along with the CPU times elapsed running each simulator on the same machine: an Intel Core i9 with 32 GB of memory. For these comparisons to be relevant, we set the "ACCURACY" parameter of EPANET 2.0 to the same value as the one used in the convergence criterion of the Python simulator Equation (27), that is 10^{-6} (unit-less). The "ACCURACY" parameter is used by EPANET 2.0's solver to determine when the convergence is reached. Indeed, according to EPANET 2.0's documentation (Rossman 2000, p. 153): "The trials end when the sum of all flow changes from the previous solution divided by the total flow in all links is less than this number'.

3. RESULTS AND DISCUSSION

This section presents the results obtained when running the tests explained in Section 2.8.

3.1. Functioning and performances of the Python simulator

When simulating the set of networks for each demand multiplier, the maximal ∞-norm of all residuals is equal to 5.45×10^{-5} (in $\mathrm{l\,s^{-1}}$ or $\mathrm{mH_2O}$), which is much less than 10^{-3}. Thus, the solver reaches the equilibrium with enough precision.

We clearly observe, as expected, that the cumulated demand and consumption increase with the demand multiplier μ_d, and that the cumulated demand increases quicker than the cumulated consumption (Figure 2(a)). This last observation is confirmed when computing the global demand satisfaction (Figure 2(b)), which decreases when μ_d increases. This functioning is completely realistic.

As logically expected, the elapsed CPU times of both simulator and solver increase globally with the size of the networks, and the elapsed time of the solver increases slower (Figure 3(a)). This is confirmed when computing the ratio of time needed by the solver when compared to the one needed for the whole simulator (Figure 3(b)).

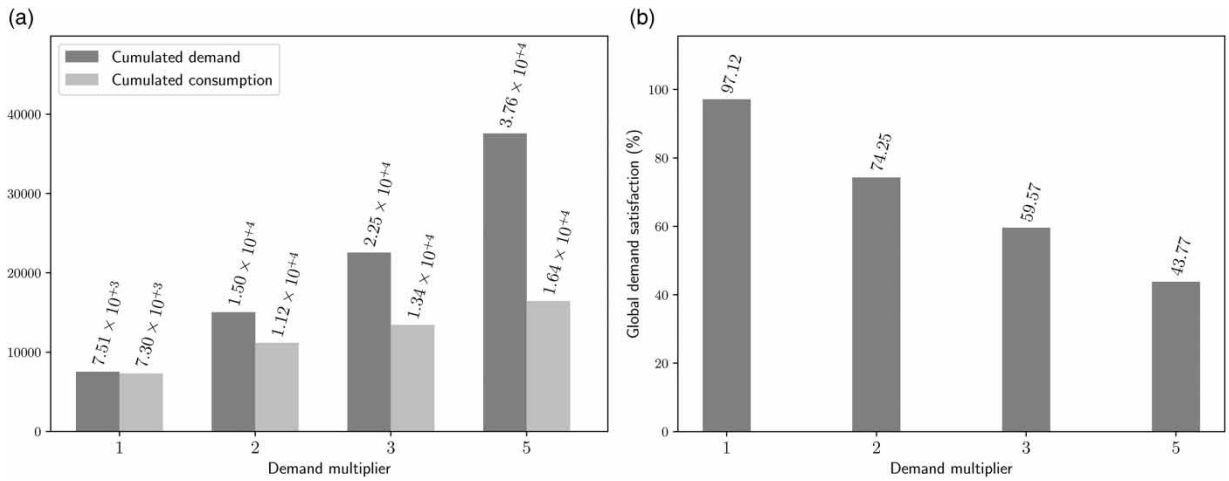


Figure 2 | Cumulated demand and consumption (Figure 2(a)), and global demand satisfaction (Figure 2(b)), simulated in networks {N1, ..., N8} with the Python simulator, for each demand multiplier $\mu_d \in \{1, 2, 3, 5\}$. (a) Cumulated demand and consumption (in $\mathrm{l\,s^{-1}}$); (b) Global demand satisfaction (in %).

[10] Source code from https://www.epa.gov/sites/default/files/2018-10/en2source.zip.

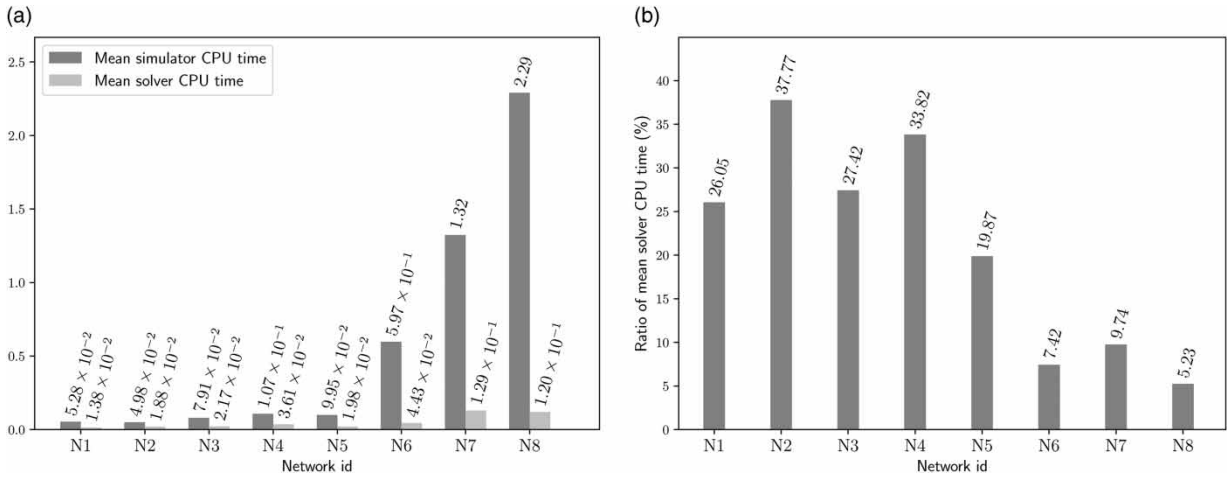


Figure 3 | Mean simulator and solver elapsed CPU times (Figure 3(a)), and ratio of mean solver elapsed CPU time when compared to mean simulator elapsed CPU time (Figure 3(b)). (a) Mean simulator and solver CPU times (in s); (b) Ratio of mean CPU time used by the solver (in %).

Figure 4(a) and 4(b) show the ratios of converging runs (in %) and the mean convergence orders when simulating all networks for all demand multipliers, first with no numerical enhancement, and then adding successively each numerical enhancement. When no numerical enhancement is enabled (bars "None"), only 12.50% of the simulations converge, with an order of convergence equal to 3.22. With just friction head-loss regularization (bars "Head-loss"), 96.88% of the simulations converge, with an order of convergence equal to 2.13. With friction head-loss and consumption regularization (bars "+ Consumption"), still 96.88% of the simulations converge, with an order of convergence equal to 2.16. With friction head-loss regularization, consumption regularization and damping correction (bars "+ Damping"), then 100% of the simulations converge, with an order of convergence equal to 2.04. Finally, with all numerical enhancements enabled (i.e., friction head-loss regularization, consumption regularization, damping correction and preconditioning; bars "+ Preconditioning"), still 100% of the simulations converge with an order of convergence of 2.04. The "not-averaged" orders of convergence per network and demand multiplier are presented in the Supplementary Material.

Globally, these metrics show that for most of the configurations (i.e., one network combined with one demand multiplier), we need to regularize the friction head-loss to reach convergence. Also, friction head-loss regularization, consumption regularization and damping correction must be enabled for the simulation of all configurations to reach convergence. If we consider only the networks and demand multipliers for which all simulations converge, then the mean convergence order is better when none of the numerical enhancements are enabled (i.e., > 3). Even so, the mean convergence order remains very good (i.e., > 2) when a part or all of the numerical enhancements are enabled. Adding preconditioning when all other numerical enhancements are already enabled does not show any improvement in these test cases. However, this

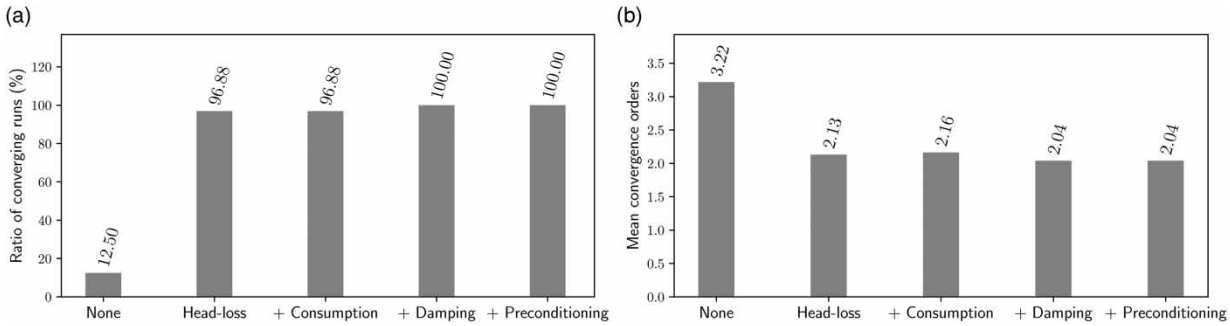


Figure 4 | Ratio of converging runs (Figure 4(a)) and mean convergence orders (Figure 4(b)) when simulating all networks {N1, ..., N8} with the Python simulator, enabling successively each numerical enhancement. (a) Ratios of converging runs (in %); (b) Mean convergence orders (unit-less).

does not mean that there would be no difference for other networks and configurations. Thus, since we do not observe computational overhead, we decide, for future simulations, to systematically enable all numerical enhancements, including the preconditioning.

3.2. Comparison of the Python simulator with the MATLAB one

When simulating each network for each demand multiplier, the maximal difference between the consumptions computed with the MATLAB and the Python simulators is equal to $9.72 \times 10^{-3}\,\mathrm{l\,s^{-1}}$. This difference is not significant. Thus, both simulators give the same result, which means that our simulator is able to run pressure-driven analyses in a correct way.

Globally, both Python and MATLAB simulators show mean convergence orders that are slightly less than 2 (Figure 5(a)). The mean numbers of damping corrections for each simulated network show that both simulators need corrections for almost the same networks, and that the numbers of corrections are comparable (Figure 5(b)).

For any network, the number of damping corrections is related to the complexity of the network: the higher it is, the more complex is the network. In our study, we could expect that this number increases with the size of the networks, i.e., that N2 needs more damping corrections than N1, N3 more than N2, etc. However, Figure 5(b) does not show any obvious proportionality relation between the size of the networks and the number of corrections. For example, even if they are smaller, networks N3 and N4 need many more corrections than N7 or N8.

Independently to their size, another source of complexity in WDNs is their degree of meshing. Indeed, Piller (1995) showed, for pressure-independent consumptions, that heavily meshed networks need more operations to be solved. Adapting the work of Piller (1995) to the case of pressure-dependent consumptions, it is possible to demonstrate numerically that {N1, ..., N4} are more meshed than {N5, ..., N8}, and therefore more complex to solve.

Finally, strong variability of hydraulic resistances and/or demands across {N1, ..., N4} could also explain why these networks are more complex to solve than {N5, ..., N8} even if they are smaller. Indeed, the presence in a network of pipes with very low resistances together with pipes of very high resistances, and/or of junctions with very low demands together with junctions with very high demands, can increase strongly the stiffness of the Schur complement of the Jacobian matrix (defined by Equation (23)).

3.3. Comparison of the Python simulator with EPANET 2.0

We find that increasing the heads at sources $\boldsymbol{h_0}$ by $10\,\mathrm{mH_2O}$ and reducing the service pressure-head p_s to $1\,\mathrm{mH_2O}$ leads to fully satisfied demand at every junction of every network. With such values of $\boldsymbol{h_0}$ and p_s, the maximal difference between the consumptions computed by the Python simulator and the ones computed by EPANET 2.0 is equal to $5.00 \times 10^{-5}\,\mathrm{l\,s^{-1}}$. Thus, our Python simulator gives the same results as EPANET 2.0 when the demands are fully satisfied, which means that our simulator is able to correctly run demand-driven analyses.

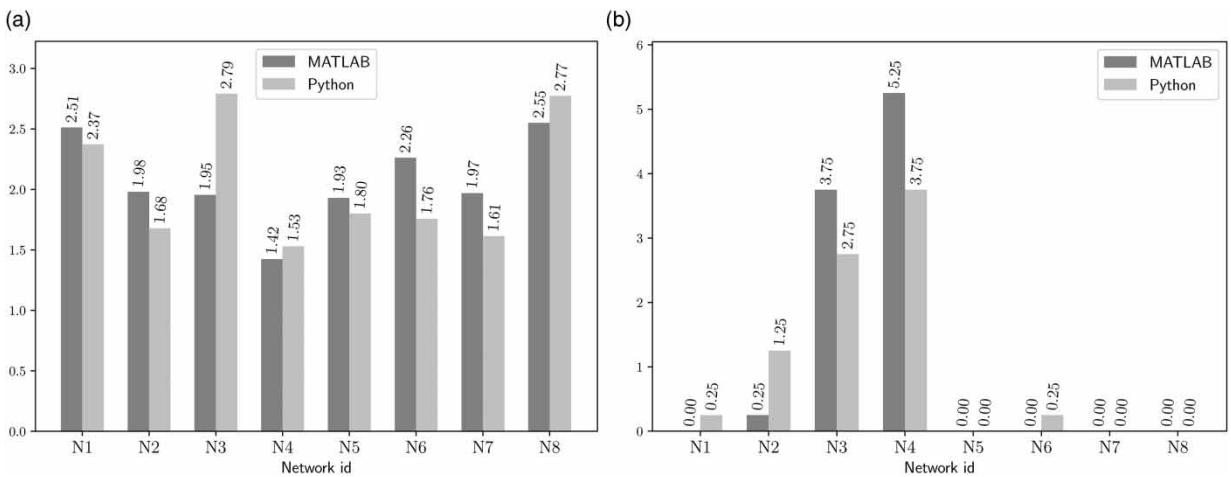


Figure 5 | Mean convergence orders (Figure 5(a)) and numbers of needed damping corrections (Figure 5(b)), simulating each network {N1, ..., N8} with the Python and MATLAB simulators. (a) Mean convergence orders; (b) Mean numbers of damping corrections.

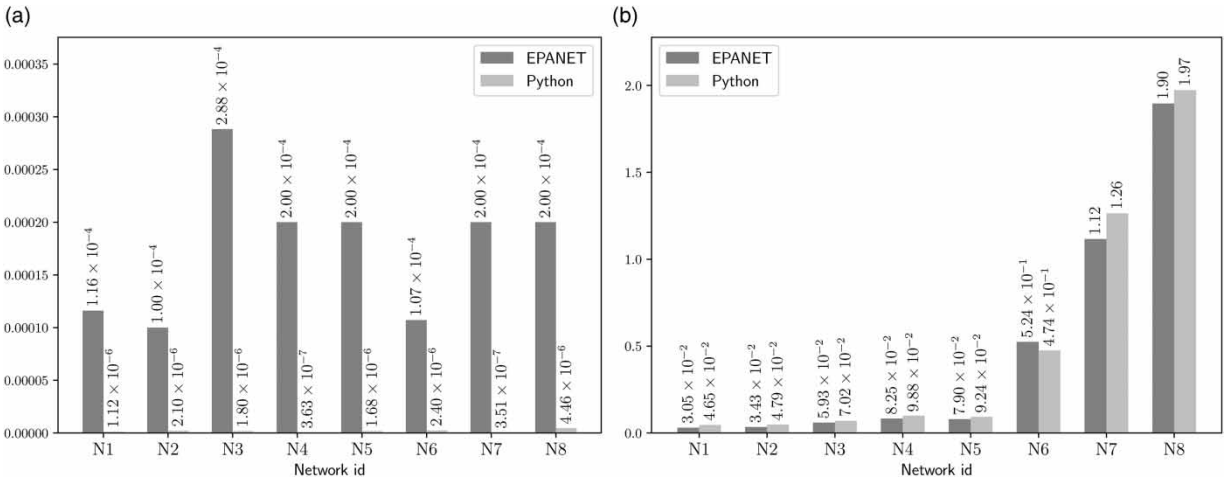


Figure 6 | Infinity norm of residuals (Figure 6(a)) and simulator elapsed CPU time (Figure 6(b)), simulating each network {N1, ..., N8} with EPANET 2.0 and the Python simulator. (a) Infinity norms of residuals (in $l\,s^{-1}$ or mH_2O); (b) Simulator CPU times (in s).

Even with an "ACCURACY" parameter equal to 10^{-6}, the ∞-norms of the residuals at convergence of EPANET 2.0 are always greater than those of the Python simulator (Figure 6(a)). Thus, our simulator is more accurate than EPANET 2.0 for these test cases. We believe that this difference is due to EPANET 2.0's convergence criterion, which considers only the differences between iterates of flow rates while the criterion of our Python simulator considers the differences between iterates of both flow rates and heads.

Remark:

EPANET 2.0's convergence criterion has been improved in EPANET 2.2, by considering also the ∞-norm of flow changes and the ∞-norm of energy residuals (Rossman *et al.* 2020, p. 116).

As expected, the elapsed CPU times of the Python simulator are slightly greater than the ones of EPANET 2.0 (Figure 6(b)). This is normal because EPANET 2.0's solution engine is coded entirely with the C programing language, which is compiled and thus more efficient than the Python language. But these differences remain small, maybe because the CHOLMOD (Chen *et al.* 2008) library used by the Python simulator to compute the Cholesky decomposition of the Schur complement matrix is more optimized than the method used in EPANET 2.0, which is based on an older work from George & Liu (1981); however, a rigorous comparison of these two methods would require some further work.

4. CONCLUSIONS

In this paper, we designed and implemented a new Python simulator to model pressure-dependent users' consumptions in WDNs with no need of artificial elements. Our simulator produces the same results as those computed by the state-of-the-art MATLAB simulator developed by Elhay *et al.* (2016). Extensive numerical experiments on 8 complex, large and realistic networks, composed of up to 19,647 pipes and 17,986 nodes, showed similar precision and efficiency for both simulators. Moreover, the performances of our simulator are close to those of EPANET 2.0 in case of fully satisfied demand.

The community of WDN modelers now has at its disposal a new efficient and flexible simulator of pressure-dependent users' consumption, integrated into the generic and collaborative framework OOPNET (Steffelbauer & Fuchs-Hanusch 2015). This is the first Python simulator that includes all numerical enhancements from Piller (1995), Piller *et al.* (2003), Elhay & Simpson (2011) and Elhay *et al.* (2016). These enhancements were mandatory for the simulation of all tested configurations to converge. Among these enhancements, the damping correction will allow the study of more complex models, especially those for which the gradient of the objective function to minimize can have sub-linear growth.

A complete set of tests were driven to assess the good functioning, stability and robustness of our simulator. Also, the native properties of the Python language make code extension easier. Therefore, our Python simulator can now be reused to study trickier physical processes, such as valves, pumps and pressure-dependent background leakage outflows, while limiting the risk of software regression.

An interesting further work could also consist in comparing the results and performances of our Python simulator with those of EPANET 2.2 (which was not available yet at the beginning of this study) when running pressure-driven analyses. In particular, since the decomposition of the Schur complement matrix is a bottleneck of both our simulator and EPANET 2.2's solution algorithms, we would like to compare the efficiency of the CHOLMOD library (Chen *et al.* 2008) used in our simulator, with that of the Cholesky decomposition from George & Liu (1981) and implemented in EPANET 2.2. The comparison of these two implementations, both written in C programing language, should not present any difficulty since our simulator has a modular architecture that permits us to change easily the solver and library to use.

AUTHOR CONTRIBUTIONS

- C. C. was involved in conceptualization, formal analysis, investigation, methodology, software, validation, visualization, writing the original draft, writing the review, and editing.
- O. P. was involved in conceptualization, formal analysis, methodology, supervision, validation, writing the review and editing.
- I. M. was involved in conceptualization, supervision, writing , revising and editing.

FUNDING

This work is part of the Oriented Renewal of Pipes (ROC) French research project and was supported by the Nouvelle Aquitaine region, the Loire-Bretagne and Adour-Garonne water agencies, the Aquitaine regional public health authorities (ARS), and the European Regional Development Fund (ERDF).

DATA AVAILABILITY STATEMENT

Please refer to the online version of this article for figures in color. Additional relevant data to the manuscript are included in the Supplementary Materials. Source code of the Python simulator is publicly accessible on https://github.com/oopnet/oopnet/tree/simulators.

CONFLICT OF INTEREST

The authors declare there is no conflict.

REFERENCES

Alperovits, E. & Shamir, U. 1977 Design of optimal water distribution systems. *Water Resources Research* **13** (6), 885–900. https://doi.org/10.1029/WR013i006p00885.

Berardi, L., Laucelli, D., Ugarelli, R. & Giustolisi, O. 2015 Leakage management: planning remote real time controlled pressure reduction in Oppegård municipality. *Procedia Engineering* **119**, 72–81. https://doi.org/10.1016/j.proeng.2015.08.855.

Chen, Y., Davis, T. A., Hager, W. W. & Rajamanickam, S. 2008 Algorithm 887: CHOLMOD, supernodal sparse cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software* **35** (3), 1–14. https://doi.org/10.1145/1391989.1391995.

Creaco, E. & Walski, T. 2017 Economic analysis of pressure control for leakage and pipe burst reduction. *Journal of Water Resources Planning and Management* **143** (12), Article 04017074. https://doi.org/10.1061/(ASCE)WR.1943-5452.0000846.

Deuerlein, J. W., Piller, O., Elhay, S. & Simpson, A. R. 2019 Content-based active-set method for the pressure-dependent model of water distribution systems. *Journal of Water Resources Planning and Management* **145** (1), Article 04018082. https://doi.org/10.1061/%28ASCE%29WR.1943-5452.0001003.

Elhay, S. & Simpson, A. R. 2011 Dealing with zero flows in solving the nonlinear equations for water distribution systems. *Journal of Hydraulic Engineering* **137** (10), 1216–1224. https://doi.org/10.1061/%28ASCE%29HY.1943-7900.0000411.

Elhay, S., Piller, O., Deuerlein, J. & Simpson, A. R. 2016 A robust, rapidly convergent method that solves the water distribution equations for pressure-dependent models. *Journal of Water Resources Planning and Management* **142** (2), Article 04015047. https://doi.org/10.1061/%28ASCE%29WR.1943-5452.0000578.

Fujiwara, O. & Ganesharajah, T. 1993 Reliability assessment of water supply systems with storage and distribution networks. *Water Resources Research* **29** (8), 2917–2924. https://doi.org/10.1029/93WR00857.

George, A. & Liu, J. W. H. 1981 *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall Series in Computational Mathematics. Prentice-Hall, Englewood Cliffs, N.J.

Giustolisi, O., Savic, D. & Kapelan, Z. 2008 Pressure-driven demand and leakage simulation for water distribution networks. *Journal of Hydraulic Engineering* **134** (5), 626–635. https://doi.org/10.1061/(ASCE)0733-9429(2008)134:5(626).

Giustolisi, O., Berardi, L., Laucelli, D., Savic, D., Walski, T. & Brunone, B. 2014 Battle of background leakage assessment for water networks (BBLAWN) at WDSA conference 2014. *Procedia Engineering* **89**, 4–12. https://doi.org/10.1016/j.proeng.2014.11.153.

Gorev, N. B., Gorev, V. N., Kodzhespirova, I. F., Shedlovsky, I. A. & Sivakumar, P. 2021 Technique for the pressure-driven analysis of water distribution networks with flow- and pressure-regulating valves. *Journal of Water Resources Planning and Management* **147** (5), Article 06021005. https://doi.org/10.1061/%28ASCE%29WR.1943-5452.0001357.

Gorev, N. B., Gorev, V. N., Kodzhespirova, I. F., Shedlovsky, I. A. & Sivakumar, P. 2022 Dealing with zero flows in the simulation of water distribution networks with low-resistance pipes using the global gradient algorithm. *Water Resources Management* **36** (5), 1679–1691. https://doi.org/10.1007/s11269-022-03100-9.

Gupta, R. & Bhave, P. R. 1996 Comparison of methods for predicting deficient-network performance. *Journal of Water Resources Planning and Management* **122** (3), 214–217. https://doi.org/10.1061/%28ASCE%290733-9496%281996%29122%3A3%28214%29. publisher: American Society of Civil Engineers.

Gupta, I., Bassin, J., Gupta, A. & Khanna, P. 1993 Optimization of water distribution system. *Environmental Software* **8** (2), 101–113. https://doi.org/10.1016/0266-9838(93)90020-I.

Gupta, R., Nair, A. G. R. & Ormsbee, L. 2016 Leakage as pressure-driven demand in design of water distribution networks. *Journal of Water Resources Planning and Management* **142** (6), Article 04016005. https://doi.org/10.1061/%28ASCE%29WR.1943-5452.0000629.

Hagberg, A., Swart, P. & Chult, D. 2008 *Exploring Network Structure, Dynamics, and Function Using Networkx*. Technical Report LA-UR-08-05495; LA-UR-08-5495, Los Alamos National Lab. (LANL), Los Alamos, NM (United States). Available from: https://www.osti.gov/biblio/960616.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C. & Oliphant, T. E. 2020 Array programming with NumPy. *Nature* **585** (7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2. number: 7825 Publisher: Nature Publishing Group.

Hold-Geoffroy, Y., Gagnon, O. & Parizeau, M. 2014 Once you SCOOP, no need to fork. In *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*. Association for Computing Machinery, New York, NY, USA, pp. 1–8. doi:10.1145/2616498.2616565.

Hoyer, S. & Hamman, J. 2017 xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software* **5** (1), Article 10. https://doi.org/10.5334/jors.148/. number: 1 Publisher: Ubiquity Press.

Hunter, J. D. 2007 Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* **9** (3), 90–95. https://doi.org/10.1109/MCSE.2007.55. conference Name: Computing in Science & Engineering.

IEC 2023 *International Electrotechnical Vocabulary*. Available from: https://www.electropedia.org/.

Itron 2021 *Static Water Meter INTELIS*. Available from: https://www.itron.com/-/media/feature/products/documents/brochure/intelis-water-static-meter-en-web.pdf.

Karadirek, I. E., Kara, S., Yilmaz, G., Muhammetoglu, A. & Muhammetoglu, H. 2012 Implementation of hydraulic modelling for water-loss reduction through pressure management. *Water Resources Management* **26** (9), 2555–2568. https://doi.org/10.1007/s11269-012-0032-2.

Klisel, K. A., Murray, R. & Haxton, T. 2018 An overview of the water network tool for resilience (WNTR). In: *1st International WDSA/CCWI2018 Joint Conference*. Kingston, Ontario, Canada, pp. 1–8.

Koşucu, M. M. & Demirel, M. C. 2022 Smart pressure management extension for EPANET: source code enhancement with a dynamic pressure reducing valve model. *Journal of Hydroinformatics* **24** (3), 642–658. https://doi.org/10.2166/hydro.2022.172.

Lao, S., Portela, S., Dequesne, J. & Debuf, O. 2022 *National Report of SISPEA Data*. Technical Report Edition of Jan. 2022 from data of 2020, French national observatory of water and sanitation services. Available from: https://www.services.eaufrance.fr/docs/synthese/rapports/Rapport_Sispea_2020_VF.pdf.

Laucelli, D., Simone, A., Berardi, L. & Giustolisi, O. 2017a Optimal design of district metering areas for the reduction of leakages. *Journal of Water Resources Planning and Management* **143** (6), Article 04017017. https://doi.org/10.1061/%28ASCE%29WR.1943-5452.0000768.

Laucelli, D. B., Berardi, L., Giustolisi, O. & Andrei, C. 2017b Reducing background leakages and energy consumption in a real WDN by optimal DMA design. In: *2017 International Conference on ENERGY and ENVIRONMENT (CIEM)*. pp. 241–245.

Lutz, M. 2013 *Learning Python*, 5th edn. O'Reilly, Beijing. https://www.safaribooksonline.com/library/view/learning-python-5th/9781449355722/.

Mahmoud, H. A., Savić, D. & Kapelan, Z. 2017 New pressure-driven approach for modeling water distribution networks. *Journal of Water Resources Planning and Management* **143** (8), Article 04017031. https://doi.org/10.1061/%28ASCE%29WR.1943-5452.0000781.

McKinney, W. 2011 pandas: a foundational Python library for data analysis and statistics. *Python for High Performance and Scientific Computing* 1–9.

Monsef, H., Naghashzadegan, M., Farmani, R. & Jamali, A. 2018 Pressure management in water distribution systems in order to reduce energy consumption and background leakage. *Journal of Water Supply: Research and Technology-Aqua* **67** (4), 397–403. https://doi.org/10.2166/aqua.2018.002.

Piller, O. 1995 *Modeling the Behavior of a Network - Hydraulic Analysis and a Sampling Procedure for Estimating the Parameters*. PhD Thesis in Applied Mathematics, University of Bordeaux, France. Available from: http://www.theses.fr/1995BOR10511.

Piller, O., Bremond, B. & Poulton, M. 2003 Least action principles appropriate to pressure-driven models of pipe networks. In *World Water & Environmental Resources Congress 2003*. American Society of Civil Engineers, Philadelphia, Pennsylvania, United States, pp. 1–15. doi:10.1061/40685%282003%29113.

Piller, O., Gilbert, D., Haddane, K. & Sabatié, S. 2011 Porteau: An Object-Oriented programming hydraulic toolkit for water distribution system analysis. In *Eleventh International Conference on Computing and Control for the Water Industry (CCWI 2011)*. Centre for Water Systems, University of Exeter, Exeter, United Kingdom, pp. 27–32. Available from: https://hal.archives-ouvertes.fr/hal-00653763.

Piller, O., Elhay, S., Deuerlein, J. & Simpson, A. 2017 Why are line search methods needed for hydraulic DDM and PDM solvers? In: *Computing and Control for the Water Industry (CCWI) 2017*. The University of Sheffield, p. 327189 Bytes. Available from: https://figshare.shef.ac.uk/articles/journal_contribution/CCWI2017_F41_Why_are_Line_Search_Methods_Needed_for_Hydraulic_DDM_and_PDM_Solvers_/5364229/1.

Ritchie, H. & Roser, M. 2017 *Water Use and Stress. Our World in Data*. Available from: https://ourworldindata.org/water-use-stress.

Rossman, L. A. (2000). *EPANET 2 Users Manual*. U.S. Environmental Protection Agency, Washington, D.C, EPA/600/R-00/057.

Rossman, L. A., Woo, H., Tryby, M., Shang, F., Janke, R. & Haxton, T. (2020). *EPANET 2.2 User Manual*. U.S. Environmental Protection Agency, Washington, DC, EPA/600/R-20/133.

Shirzad, A., Tabesh, M., Farmani, R. & Mohammadi, M. 2013 Pressure-discharge relations with application to head-driven simulation of water distribution networks. *Journal of Water Resources Planning and Management* **139** (6), 660–670. https://doi.org/10.1061/%28ASCE%29WR.1943-5452.0000305.

Siew, C. & Tanyimboh, T. T. 2012 Pressure-dependent EPANET extension. *Water Resources Management* **26** (6), 1477–1498. https://doi.org/10.1007/s11269-011-9968-x.

Sivakumar, P., Gorev, N. B., Nivedita, S., Suribabu, C. R., Gupta, R. & Tanyimboh, T. T. 2023 An assessment of the artificial modelling elements approach to the pressure-driven analysis of water distribution networks. *Water Supply* **23** (5), 1810–1826. https://doi.org/10.2166/ws.2023.092.

Steffelbauer, D. & Fuchs-Hanusch, D. 2015 OOPNET: an object-oriented EPANET in Python. *Procedia Engineering* **119**, 710–718. https://doi.org/10.1016/j.proeng.2015.08.924.

Steffelbauer, D., Piller, O., Chambon, C. & Abraham, E. 2022 Towards a novel multi-purpose simulation software of water distribution systems in Python. In: *14th International Conference on Hydroinformatics, Water INFLUENCE Water INFormatic soLUtions and opEN problems in the cycle from Clouds to ocEan*. Bucharest, Romania, p. 4. Available from: https://hal.inrae.fr/hal-03750370.

Tanyimboh, T. & Templeman, A. 2004 A new nodal outflow function for water distribution networks. In *The Fourth International Conference on Engineering Computational Technology*. Lisbon, Portugal, p. 64. Available from: http://www.ctresources.info/ccp/paper.html?id=3699.

Todini, E., Santopietro, S., Gargano, R. & Rossman, L. A. 2021 Pressure flow–based algorithms for pressure-driven analysis of water distribution networks. *Journal of Water Resources Planning and Management* **147** (8), Article 04021048. https://doi.org/10.1061/%28ASCE%29WR.1943-5452.0001401.

Vaidya, D. R., Mali, S. T. & Gupta, R. 2023 Pressure-dependent analysis of water distribution network having different required pressure values at different nodes. *Iranian Journal of Science and Technology, Transactions of Civil Engineering* https://doi.org/10.1007/s40996-023-01183-x.

van Zyl, J. E., Borthwick, J. & Hardy, A. 2003 OOTEN An object-oriented programmers toolkit for Epanet. In *Conference: CCWI 2003 Advances in water supply management*. Imperial College, London, United Kingdom, pp. 1–8.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, I., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F. & van Mulbregt, P. 2020 SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods* **17** (3), 261–272. https://doi.org/10.1038/s41592-019-0686-2. number: 3 Publisher: Nature Publishing Group.

Wagner, J. M., Shamir, U. & Marks, D. H. 1988 Water distribution reliability: simulation methods. *Journal of Water Resources Planning and Management* **114** (3), 276–294. https://doi.org/10.1061/%28ASCE%290733-9496%281988%29114%3A3%28276%29.

Walski, T. M., Chase, D. V., Savic, D., Grayman, W., Beckwith, S., Koelle, E. & Haestad Methods, I. (2007). *Advanced Water Distribution Modeling and Management*. Bentley Institute Press, Exton, PA, oCLC: 154213231.

Williams, G. S. & Hazen, A. 1933 *Hydraulic Tables: The Elements of Gagings and the Friction of Water Flowing in Pipes, Aqueducts, Sewers, etc*. J. Wiley & sons, inc.: London, Chapman & Hall, limited.

Xing, L. & Sela, L. 2020 Transient simulations in water distribution networks: TSNet python package. *Advances in Engineering Software* **149**, Article 102884. https://doi.org/10.1016/j.advengsoft.2020.102884.

First received 15 September 2023; accepted in revised form 25 November 2023. Available online 9 December 2023