

MÉMOIRE DE RECHERCHE

---

# Approximation numérique d'ordre élevé de l'équation de Saint Venant

---

*Auteurs :*  
Brice GONEL  
Romain PINGUET

*Professeur encadrant :*  
Thomas REY



# Table des matières

1	Description et dérivation des équations . . . . .	4
1.1	Description des équations . . . . .	4
1.2	Dérivation des équations . . . . .	4
	Les équations d'Euler . . . . .	5
	Dérivation du système de Saint-Venant . . . . .	6
2	Découverte du modèle et premières propriétés qualitatives . . . . .	8
3	Le schéma de Rusanov . . . . .	12
3.1	Présentation du schéma et résultats de convergence . . . . .	12
	Approche naïve : Flux centré . . . . .	13
	Flux de Lax-Friedrichs . . . . .	14
	Flux de Rusanov . . . . .	14
3.2	Validation de l'implémentation avec des tests . . . . .	15
	Quelques exemples . . . . .	15
	Influence du pas spatial . . . . .	17
	Influence du pas temporel (va être retravaillé) . . . . .	19
A	Annexes . . . . .	20
A.1	Code de l'implémentation du schéma de Rusanov en 1D . . . . .	20
A.2	Code de l'implémentation du schéma de Rusanov en 2D . . . . .	27
	Équations dans le cas 2D . . . . .	27
	Paramètres du problème . . . . .	27
	Nombre de sauvegardes durant la simulation . . . . .	28
	Affichage des images . . . . .	29
	Sauvegarde des images . . . . .	31
	Fonctions intervenant dans le schéma . . . . .	32

Les équations de Saint-Venant modélisent le comportement d'un écoulement en eau peu profonde, comme un canal ou un bord de plage. Ces équations, introduites à la fin des années 1880, sont extrêmement bien adaptées à modélisation et à la simulation numérique de phénomènes catastrophiques, comme les inondations ou les tsunamis.

## 1 Description et dérivation des équations

Ici, il s'agit d'énoncer les équations, de décrire les différentes quantités en jeu ( $h$ ,  $u$ ,  $q$  et  $Z$ ) et d'expliquer comment on peut arriver à ces équations.

### 1.1 Description des équations

Le système de Saint-Venant avec terme source (qui est aussi désigné par le nom « équations d'écoulements en eau peu profonde ») est le suivant :

$$\frac{\partial h}{\partial t}(t, x) + \frac{\partial q}{\partial x}(t, x) = 0 \quad (1)$$

$$\frac{\partial q}{\partial t}(t, x) + \frac{\partial}{\partial x} \left( \frac{q^2(t, x)}{h(t, x)} + g \frac{h^2(t, x)}{2} \right) = -gh(t, x) \frac{\partial Z}{\partial x}(x) \quad (2)$$

Celui-ci permet de décrire un écoulement d'eau unidirectionnel où  $h(t, x) > 0$  représente la hauteur d'eau,  $u(t, x)$  désigne la vitesse du fluide,  $q(t, x) = h(t, x)u(t, x)$  le débit du fluide et  $Z(x)$  la topographie du canal ;  $t \geq 0$  étant le temps,  $x \in \mathbb{R}$  la position spatiale dans le cours d'eau et  $g$  la constante de gravitation.

Dans ce système,  $Z$  ne dépend pas du temps  $t$ . On fait ainsi l'hypothèse que le fond ne s'érode pas au cours du temps ; ce qui paraît raisonnable dans de nombreuses situations (un fond rocheux sur une courte durée, par exemple).

### 1.2 Dérivation des équations

Rappelons la règle de Leibniz qui permet de calculer la dérivée par rapport à  $x$  d'une fonction de la forme  $\int_{a(x)}^{b(x)} f(x, t) dt$ .

**Proposition 1** (Règle de Leibniz). *Soit  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  telle que  $f$  et  $\frac{\partial f}{\partial x}$  soient continues sur  $\mathbb{R}$ , et soient  $a$  et  $b$  deux fonctions dérivables de  $\mathbb{R}$  dans  $\mathbb{R}$ . Alors, l'intégrale paramétrique  $F$  définie sur  $\mathbb{R}$  par :  $F(x) = \int_{a(x)}^{b(x)} f(x, y) dy$  est dérivable et :*

$$F'(x) = f(x, b(x))b'(x) - f(x, a(x))a'(x) + \int_{a(x)}^{b(x)} \frac{\partial f}{\partial x}(x, y) dy. \quad (3)$$

*Démonstration.* Posons  $H : \mathbb{R}^3 \rightarrow \mathbb{R}$ , l'application définie par  $H(u, v, x) = \int_u^v f(x, y) dy$ . D'après le théorème fondamental de l'analyse, on a

$$\frac{\partial H}{\partial u}(u, v, x) = \frac{\partial}{\partial u} \left( - \int_v^u f(x, u) \right) = -f(x, u),$$

et

$$\frac{\partial H}{\partial v}(u, v, x) = \frac{\partial}{\partial v} \left( \int_u^v f(x, u) \right) = f(x, v).$$

D'après le théorème de dérivation sous le signe somme, on a

$$\frac{\partial H}{\partial x}(u, v, x) = \int_u^v \frac{\partial f}{\partial x}(x, y) dy.$$

A partir de là, le théorème est une simple chain rule :

$$\begin{aligned} F'(x) &= (H(a(x), b(x), x))' \\ &= \begin{bmatrix} -f(x, a(x)) & f(x, b(x)) & \frac{\partial H}{\partial x}(a(x), b(x), x) \end{bmatrix} \times \begin{bmatrix} a'(x) \\ b'(x) \\ 1 \end{bmatrix} \\ &= f(x, b(x))b'(x) - f(x, a(x))a'(x) + \int_{a(x)}^{b(x)} \frac{\partial f}{\partial x}(x, y) dy. \end{aligned}$$

□

### Les équations d'Euler

Dans ce paragraphe, nous allons d'abord parler des équations d'Euler. Ce sont des équations qui forment un système hyperbolique non-linéaire. Les équations expriment des lois de conservation qui gouvernent la dynamique des fluides. C'est à partir de ces équations, et moyennant certaines hypothèses, que nous allons dériver les équations de Saint-Venant. Notons  $\rho(x, y, z, t)$  la masse volumique du fluide,  $p(x, y, z, t)$  la pression,  $u(x, y, z, t)$  la composante selon l'axe ( $Ox$ ) de la vitesse,  $v(x, y, z, t)$  sa composante selon l'axe ( $Oy$ ) et  $w(x, y, z, t)$  sa composante selon l'axe ( $Oz$ ). Nous noterons alors par la suite  $\mathbf{V} = (u, v, w)$  le vecteur vitesse dont les composantes sont les vitesses  $u, v$  et  $w$ .

Les trois équations d'Euler sont les suivantes :

$$(\rho u)_t + (\rho u^2 + p)_x + (\rho uv)_y + (\rho uw)_z = 0, \quad (4)$$

$$(\rho v)_t + (\rho uv)_x + (\rho v^2 + p)_y + (\rho vw)_z = 0, \quad (5)$$

$$(\rho w)_t + (\rho uw)_x + (\rho vw)_y + (\rho w^2 + p)_z = 0. \quad (6)$$

L'équation (4) est celle qui va nous servir à dériver la deuxième équation de Saint-Venant énoncée. A ces 3 équations, on associe aussi parfois l'équation

$$\rho_t + \operatorname{div}(\rho \mathbf{V}) = \rho_t + (\rho u)_x + (\rho v)_y + (\rho w)_z = 0, \quad (7)$$

loi de conservation qui va nous servir pour dériver la première équation du système.

## Dérivation du système de Saint-Venant

**Définition 1.** Soit  $\mathbf{V} = (u, v, w)$  la vitesse définie comme avant. Soit  $f$  une fonction qui dépend de  $t, x, y$  et  $z$ . On définit (et on note) la dérivée particulière de  $f$  par  $\frac{Df}{Dt} = \frac{\partial f}{\partial t} + u \frac{\partial f}{\partial x} + v \frac{\partial f}{\partial y} + w \frac{\partial f}{\partial z}$ .

Voici les hypothèses nécessaires à la dérivation des équations de Saint-Venant :

- **(H1)** : Le fluide est incompressible,
- **(H2)** :  $h$ ,  $u$  et  $v$  sont invariantes selon l'axe ( $Oy$ ) (puisque'il s'agit d'étudier un canal, on ramène l'étude au plan ( $y = 0$ ) en négligeant les variations selon la largeur du canal),
- **(H3)** : La vitesse  $u$  est constante selon l'axe ( $Oz$ ),
- **(H4)** : la topographie  $Z$  est constante au cours du temps.

Nous utiliserons aussi les conditions de passage suivantes :

- **(H5)** :  $\frac{D}{Dt}(Z + h - z) = 0$  en  $z = Z + h$ ,
- **(H6)** :  $\frac{D}{Dt}(Z - z) = 0$  en  $z = Z$ .

La première condition **(H5)** traduit l'hypothèse que la composante normale de la vitesse à la surface est nulle, autrement dit qu'il n'y a pas de flux entrant par la surface de l'eau. La deuxième **(H6)** traduit le même phénomène mais au niveau du fond de l'eau.

D'après **(H1)**, on peut sortir  $\rho$  des dérivées. Alors (4) se réécrit en

$$u_t + (u^2)_x + (uv)_y + (uw)_z = -\frac{1}{\rho} p_x. \quad (8)$$

En supposant **(H2)** on a que les quantités  $h$  et  $u$  sont invariantes selon l'axe ( $Oy$ ). Dans ces conditions, le terme  $(uv)_y$  est nul et on obtient :

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(u^2) + \frac{\partial}{\partial z}(uw) = -\frac{1}{\rho} \frac{\partial p}{\partial x}. \quad (9)$$

Maintenant, on intègre entre  $z = Z$  et  $z = Z + h$  :

$$\int_Z^{Z+h} \left[ \frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(u^2) \right] dz + [uw]_Z^{Z+h} = -\frac{1}{\rho} \int_Z^{Z+h} \frac{\partial p}{\partial x} dz. \quad (10)$$

Le terme  $[uw]_Z^{Z+h}$  disparaît étant donné que la vitesse verticale  $w$  est nulle à la surface libre et au fond.

Avec la règle de Leibniz, on a d'une part :

$$\begin{aligned} \frac{\partial}{\partial t} \int_Z^{Z+h} u dz &= \int_Z^{Z+h} \frac{\partial u}{\partial t} dz + u(z = Z + h) \frac{\partial}{\partial t}(Z + h) - u(z = Z) \frac{\partial Z}{\partial t} \\ &= \int_Z^{Z+h} \frac{\partial u}{\partial t} dz + u(z = Z + h) \frac{\partial h}{\partial t}, \end{aligned} \quad (11)$$

la suppression du terme  $\frac{\partial Z}{\partial t}$  venant de l'hypothèse **(H4)** selon laquelle le fond  $Z$  est constant au cours du temps.

Et d'autre part on a :

$$\frac{\partial}{\partial x} \int_Z^{Z+h} u^2 dz = \int_Z^{Z+h} \frac{\partial}{\partial x} (u^2) dz + u^2(z = Z+h) \frac{\partial}{\partial x} (Z+h) - u^2(z = Z) \frac{\partial Z}{\partial x}. \quad (12)$$

Donc en injectant (11) et (12) dans (10) on obtient :

$$\begin{aligned} \frac{\partial}{\partial t} \left[ \int_Z^{Z+h} u dz \right] - u(z = Z+h) \frac{\partial h}{\partial t} + \frac{\partial}{\partial x} \left[ \int_Z^{Z+h} u^2 dz \right] - u^2(z = Z+h) \frac{\partial}{\partial x} (Z+h) \\ + u^2(z = Z) \frac{\partial Z}{\partial x} = -\frac{1}{\rho} \int_Z^{Z+h} \frac{\partial p}{\partial x} dz. \end{aligned} \quad (13)$$

Si l'on suppose que la vitesse  $u$  est constante selon l'axe  $(Oz)$ , on a que  $\int_Z^{Z+h} u dz = uh$  et  $\int_Z^{Z+h} u^2 dz = u^2 h$ , et donc :

$$\frac{\partial}{\partial t} (uh) + \frac{\partial}{\partial x} (u^2 h) - u(z = Z+h) \left( \frac{\partial h}{\partial t} + u(z = Z+h) \frac{\partial}{\partial x} (Z+h) \right) = -\frac{1}{\rho} \int_Z^{Z+h} \frac{\partial p}{\partial t} dz. \quad (14)$$

Une simplification s'effectue grâce à **(H5)**.  $\frac{D}{Dt} (Z+h-z) = 0$  en  $z = Z+h$  équivaut à  $\frac{\partial h}{\partial t} + u \frac{\partial Z}{\partial x} + u \frac{\partial h}{\partial x} = 0$  en  $z = Z+h$  (en utilisant les invariances et le fait que la vitesse verticale  $w$  est nulle à la surface libre). Et donc :

$$\frac{\partial h}{\partial t} + u(z = Z+h) \frac{\partial}{\partial x} (Z+h) = 0.$$

L'équation 14 se réduit alors en

$$\frac{\partial q}{\partial t} + \frac{\partial}{\partial x} (hu^2) = -\frac{1}{\rho} \int_Z^{Z+h} \frac{\partial p}{\partial x} dz. \quad (15)$$

Encore une fois avec la règle de Leibniz et utilisant la formule d'équilibre hydrostatique :

$$\frac{\partial}{\partial x} \int_Z^{Z+h} p dz = \int_Z^{Z+h} \frac{\partial p}{\partial x} dz + p(Z+h) \frac{\partial}{\partial x} (Z+h) - p(Z) \frac{\partial Z}{\partial x} = \int_Z^{Z+h} \frac{\partial p}{\partial x} dz - \rho g h \frac{\partial Z}{\partial x},$$

la pression étant considérée égale à 0 à la surface de l'eau. En utilisant encore cette formule on a :

$$\begin{aligned} \int_Z^{Z+h} \frac{\partial p}{\partial x} dz &= \frac{\partial}{\partial x}(\rho g h \times h) + \rho g h \frac{\partial Z}{\partial x} \\ &= \rho \left( g \frac{\partial}{\partial x}(h^2) + g h \frac{\partial Z}{\partial x} \right). \end{aligned} \quad (16)$$

Et en combinant 15 et 16 on arrive à la deuxième équation de Saint-Venant en remplaçant le second membre :

$$\frac{\partial q}{\partial t} + \frac{\partial}{\partial x} \left( \frac{q^2}{h} + g \frac{h^2}{2} \right) = -g h \frac{\partial Z}{\partial x}. \quad (17)$$

Pour obtenir la première équation, partons de 7 et intégrons entre  $z = Z$  et  $z = Z + h$  :

$$\int_Z^{Z+h} (u_x + v_y) dz + w|_{z=Z+h} - w|_{z=Z} = 0. \quad (18)$$

Les conditions (H5) et (H6) donnent :

$$\left[ \frac{\partial(Z+h)}{\partial t} + u \cdot \frac{\partial(Z+h)}{\partial x} + v \cdot \frac{\partial(Z+h)}{\partial y} - w \right]_{z=Z+h} = 0, \quad (19)$$

$$[u \cdot Z_x + v \cdot Z_y + w]_{z=Z} = 0. \quad (20)$$

Et grâce à la formule de Leibniz, en réinsérant 19 et 20 dans 18, on a finalement :

$$\frac{\partial(Z+h)}{\partial t} + \frac{\partial}{\partial x} \int_Z^{Z+h} u dz + \frac{\partial}{\partial y} \int_Z^{Z+h} v dz. \quad (21)$$

Nous pouvons davantage simplifier cette équation en observant que  $u$  et  $v$  sont indépendants de  $z$ , ainsi que  $Z$  indépendant du temps. Cette équation devient donc :

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}(u \cdot h) + \frac{\partial}{\partial y}(v \cdot h) = 0, \quad (22)$$

soit la première équation de Saint Venant.

Les deux équations 22 et 17 forment un système d'équations aux dérivées partielles d'inconnues  $h$  et  $q$  (ou  $h$  et  $u$ ). Etant données des conditions de bord et des conditions initiales, on doit pouvoir justifier qu'il existe une unique solution que l'on peut calculer numériquement à l'aide d'un schéma de type éléments finis.

## 2 Découverte du modèle et premières propriétés qualitatives

**Proposition 2.** *La vitesse  $u$  vérifie la loi de conservation hyperbolique scalaire suivante :*

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} \left( \frac{u^2}{2} + g(Z+h) \right) = 0 \quad (23)$$



*Démonstration.* Puisque  $q = h.u$ , la dérivée spatiale de  $q$  s'écrit :

$$\frac{\partial q}{\partial x} = u \frac{\partial h}{\partial x} + h \frac{\partial u}{\partial x} \quad (24)$$

et la dérivée temporelle :

$$\frac{\partial q}{\partial t} = u \frac{\partial h}{\partial t} + h \frac{\partial u}{\partial t} \quad (25)$$

Alors en utilisant l'équation 1 du système, la dérivée temporelle donne aussi la relation

$$\frac{\partial q}{\partial t} = -u \frac{\partial q}{\partial x} + h \frac{\partial u}{\partial t} \quad (26)$$

La loi de conservation que l'on cherche à montrer est alors une réécriture de l'équation 2 du système. On a en effet :

$$\frac{\partial q}{\partial t} + \frac{\partial}{\partial x} \left( \frac{q^2}{h} + g \frac{h^2}{2} \right) = -gh \frac{\partial Z}{\partial x}$$

En développant la dérivée par rapport à  $x$  et en utilisant (26) :

$$\begin{aligned} & -u \frac{\partial q}{\partial x} + h \frac{\partial u}{\partial t} + u^2 \frac{\partial h}{\partial x} + 2hu \frac{\partial u}{\partial x} + gh \frac{\partial h}{\partial x} = -gh \frac{\partial Z}{\partial x} \\ \Leftrightarrow & -u \frac{\partial q}{\partial x} + h \frac{\partial u}{\partial t} + u^2 \frac{\partial h}{\partial x} + 2hu \frac{\partial u}{\partial x} + gh \frac{\partial h}{\partial x} + gh \frac{\partial Z}{\partial x} = 0 \\ \Leftrightarrow & u \left[ -\frac{\partial q}{\partial x} + \frac{h}{u} \frac{\partial u}{\partial t} + u \frac{\partial h}{\partial x} \right] + h \left[ u \frac{\partial u}{\partial x} + \frac{\partial}{\partial x} \left( \frac{u^2}{2} + g(Z+h) \right) \right] = 0 \\ \Leftrightarrow & u \left[ -\frac{\partial q}{\partial x} + \frac{h}{u} \frac{\partial u}{\partial t} + u \frac{\partial h}{\partial x} + h \frac{\partial u}{\partial x} \right] + h \left[ \frac{\partial}{\partial x} \left( \frac{u^2}{2} + g(Z+h) \right) \right] = 0 \end{aligned}$$

D'après (24), une simplification s'opère dans les crochets de gauche et on obtient :

$$u \left[ \frac{h}{u} \frac{\partial u}{\partial t} \right] + h \left[ \frac{\partial}{\partial x} \left( \frac{u^2}{2} + g(Z+h) \right) \right] = 0$$

Il suffit alors de diviser par  $h > 0$  pour obtenir le résultat (23).

□

Pour la suite, posons :

$$\mathbf{U} = \begin{pmatrix} h \\ q \end{pmatrix} \in \mathbb{R}^2, \quad \mathbf{F}(\mathbf{U}) = \begin{pmatrix} q \\ \frac{q^2}{h} + g \frac{h^2}{2} \end{pmatrix} \in \mathbb{R}^2 \quad \text{et} \quad \mathbf{B}(\mathbf{U}) = \begin{pmatrix} 0 \\ -gh \frac{\partial Z}{\partial x} \end{pmatrix} \in \mathbb{R}^2.$$

$\mathbf{U}$  désigne le vecteur inconnu,  $\mathbf{F}(\mathbf{U})$  la fonction flux et  $\mathbf{B}(\mathbf{U})$  le terme source. Avec ces notations, le système de départ se réécrit :

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial}{\partial x} (\mathbf{F}(\mathbf{U})) = \mathbf{B}(\mathbf{U})$$

Nous allons faire l'hypothèse que  $Z \equiv 0$ . Les résultats que nous allons montrer à partir de maintenant ne seront valables que pour une topographie  $Z$  nulle. En particulier, l'utilisation du schéma de Rusanov hors de ce cadre relèvera d'une démarche empirique.

**Proposition 3.** *Posons le vecteur  $\mathbf{W} = (h, u)^T$  (variable dite non conservative).  $\mathbf{W}$  vérifie le système quasi-linéaire suivant :*

$$\frac{\partial \mathbf{W}}{\partial t} + \mathbb{A}(\mathbf{W}) \frac{\partial \mathbf{W}}{\partial x} = 0$$

avec  $\mathbb{A}(\mathbf{W})$  définie par :

$$\mathbb{A}(\mathbf{W}) = \begin{pmatrix} u & h \\ g & u \end{pmatrix}.$$

*Démonstration.* On a  $\frac{\partial \mathbf{W}}{\partial t} = \begin{pmatrix} h_t \\ u_t \end{pmatrix}$  et  $\frac{\partial \mathbf{W}}{\partial x} = \begin{pmatrix} h_x \\ u_x \end{pmatrix}$ .

En effectuant le produit matriciel et en sommant les termes dans l'équation  $\frac{\partial \mathbf{W}}{\partial t} + \mathbb{A}(\mathbf{W}) \frac{\partial \mathbf{W}}{\partial x} = 0$  on obtient

$$\begin{pmatrix} h_t + uu_x + hu_x \\ u_t + gh_x + uu_x \end{pmatrix} = \begin{pmatrix} h_t + q_x \\ u_t + (gh + u^2)_x \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

En utilisant la première équation du système (1) et la loi de conservation (23) dans le cas  $Z \equiv 0$ .

□

**Proposition 4.** *La matrice  $\mathbb{A}(\mathbf{W})$  est diagonalisable et on a :*

$$\mathbb{P}(\mathbf{W})^{-1} \mathbb{A}(\mathbf{W}) \mathbb{P}(\mathbf{W}) = \mathbb{D}(\mathbf{W})$$

où

$$\mathbb{D}(\mathbf{W}) = \begin{pmatrix} u + \sqrt{gh} & 0 \\ 0 & u - \sqrt{gh} \end{pmatrix} \text{ et } \mathbb{P}(\mathbf{W}) = \begin{pmatrix} \frac{\sqrt{h}}{\sqrt{g}} & -\frac{\sqrt{h}}{\sqrt{g}} \\ 1 & 1 \end{pmatrix}.$$

*Démonstration.* Le résultat est direct en faisant le produit matriciel si on connaît les expressions de  $\mathbb{D}(\mathbf{W})$  et  $\mathbb{P}(\mathbf{W})$ . Dans le cas contraire, on procède comme suit :

On détermine les valeurs propres de la matrice  $\mathbb{A}(\mathbf{W})$ , ce qui conduit à chercher les racines du polynôme (d'indéterminée  $\lambda$ ) suivant :

$$\det(\mathbb{A}(\mathbf{W}) - \lambda I) = \begin{vmatrix} u - \lambda & h \\ g & u - \lambda \end{vmatrix} = (u - \lambda)^2 - gh.$$

Or puisque  $g$  et  $h$  sont  $> 0$ ,

$$(u - \lambda)^2 - gh = 0 \Leftrightarrow \lambda u + \sqrt{gh} \text{ ou } \lambda u - \sqrt{gh}$$

et comme la matrice admet deux valeurs propres distinctes, elle est diagonalisable. La matrice diagonale est alors

$$\mathbb{D}(\mathbf{W}) = \begin{pmatrix} u + \sqrt{gh} & 0 \\ 0 & u - \sqrt{gh} \end{pmatrix}.$$

Il reste à déterminer la matrice de passage  $\mathbb{P}(\mathbf{W})$ . Il s'agit de trouver  $\begin{pmatrix} a \\ b \end{pmatrix}$  un vecteur propre associé à  $u + \sqrt{gh}$  et  $\begin{pmatrix} c \\ d \end{pmatrix}$  un vecteur propre associé à  $u - \sqrt{gh}$ .

Pour le vecteur propre associé à  $u + \sqrt{gh}$  :

$$\mathbb{A}(\mathbf{W}) \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} u & h \\ g & u \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = (u + \sqrt{gh}) \begin{pmatrix} a \\ b \end{pmatrix}$$

Il s'agit d'un système lié donc il est équivalent de raisonner sur la première équation :

$$au + bh = au + a\sqrt{gh}$$

$$\Leftrightarrow bh = a\sqrt{gh}$$

et on voit que  $\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sqrt{h}/\sqrt{g} \\ 1 \end{pmatrix}$  convient.

De même, pour le vecteur propre associé à  $u - \sqrt{gh}$  :

Cette fois on arrive à

$$cu + dh = cu - c\sqrt{gh}$$

$$\Leftrightarrow dh = -c\sqrt{gh}$$

et on voit que  $\begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} -\sqrt{h}/\sqrt{g} \\ 1 \end{pmatrix}$  convient.

D'où finalement :

$$\mathbb{P}(\mathbf{W}) = \begin{pmatrix} \frac{\sqrt{h}}{\sqrt{g}} & -\frac{\sqrt{h}}{\sqrt{g}} \\ 1 & 1 \end{pmatrix}.$$

□

**Proposition 5.** La matrice jacobienne de  $F$  admet deux valeurs propres distinctes  $\lambda_1(\mathbf{U})$  et  $\lambda_2(\mathbf{U})$  qui sont égales aux valeurs propres de  $\mathbb{A}(\mathbf{W})$ .

*Démonstration.* La matrice jacobienne de  $F$  est la suivante :

$$J_F(h, q) = \begin{pmatrix} 0 & 1 \\ -\frac{q^2}{h^2} + gh & \frac{2q}{h} \end{pmatrix}$$

Pour déterminer les valeurs propres de cette matrice, il faut calculer les racines du polynôme donné par

$$\det(J_F(h, q)) = \begin{vmatrix} 0 - \lambda & 1 \\ -\frac{q^2}{h^2} + gh & \frac{2q}{h} - \lambda \end{vmatrix}$$

et

$$\begin{aligned} -\lambda\left(\frac{2q}{h} - \lambda\right) + \frac{q^2}{h^2} - gh &= 0 \\ \Leftrightarrow \lambda^2 - 2u\lambda + u^2 - gh &= 0. \end{aligned}$$

Puisque  $\Delta = 4u^2 - 4(u^2 - gh) = 4gh > 0$ , il y a deux racines distinctes

$$\lambda_1(\mathbf{U}) = \frac{2u + 2\sqrt{gh}}{2} \text{ et } \lambda_2(\mathbf{U}) = \frac{2u - 2\sqrt{gh}}{2}.$$

En simplifiant par 2 au numérateur et dénominateur, on voit qu'il s'agit des valeurs propres de  $\mathbb{A}(\mathbf{W})$ .

□

Les expressions de ces valeurs propres vont être utiles par la suite : elles vont en effet apparaître de façon explicite dans le schéma que nous allons utiliser ; et donc dans le code de l'implémentation.

Plus précisément : le système d'EDP de Saint-Venant est tel que la matrice jacobienne du flux admet deux valeurs propres distinctes. On parle alors de système hyperbolique. Dans ce contexte, on peut appliquer le schéma dit de Rusanov, auquel est associé une condition CFL qui assure la stabilité. Nous allons développer ce point dans la partie suivante consacrée à ce schéma.

### 3 Le schéma de Rusanov

#### 3.1 Présentation du schéma et résultats de convergence

Divisons tout d'abord notre espace et notre temps avec les pas  $\Delta x$  et  $\Delta t$ , respectivement. On cherche à discrétiser l'équation :

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial}{\partial x}(\mathbf{F}(\mathbf{U})) = 0 \quad \text{Équation sans terme source.}$$

On pose ainsi  $U_i^n$  l'approximation de  $U$  autour de la position  $x_i$  à l'étape  $t^n$ . On peut donc noter :

$$U_i^n = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} U(x, t^n) dx$$

En intégrant le premier terme de 3.1 entre  $x_{i-1/2}$  et  $x_{i+1/2}$ , on a grâce à un développement de Taylor :

$$\int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial U}{\partial t} dx = \int_{x_{i-1/2}}^{x_{i+1/2}} \frac{U(x, t^{n+1}) - U(x, t^n)}{\Delta t} dx = \Delta x \frac{U_i^{n+1} - U_i^n}{\Delta t}.$$

Et en intégrant le second terme de 3.1 :

$$\int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial F(U)}{\partial x} dx = F(U(x_{i+1/2})) - F(U(x_{i-1/2})) = \mathcal{F}_{i+1/2} - \mathcal{F}_{i-1/2}.$$

On peut faire l'hypothèse que le flux de  $U$  passant en  $x_{i+1/2}$ , soit  $\mathcal{F}_{i+1/2}$ , dépend uniquement de  $U_i$  et  $U_{i+1}$ . On a finalement :

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} + \frac{\mathcal{F}_{i+1/2}^n - \mathcal{F}_{i-1/2}^n}{\Delta x} = 0.$$

Toute la question est donc de savoir comment approcher le flux numérique  $\mathcal{F}_{i+1/2}^n$  en fonction de  $U_i^n$  et  $U_{i+1}^n$ . On notera par la suite que  $\mathcal{F}_{i+1/2}^n = f(U_i^n, U_{i+1}^n)$

### Approche naïve : Flux centré

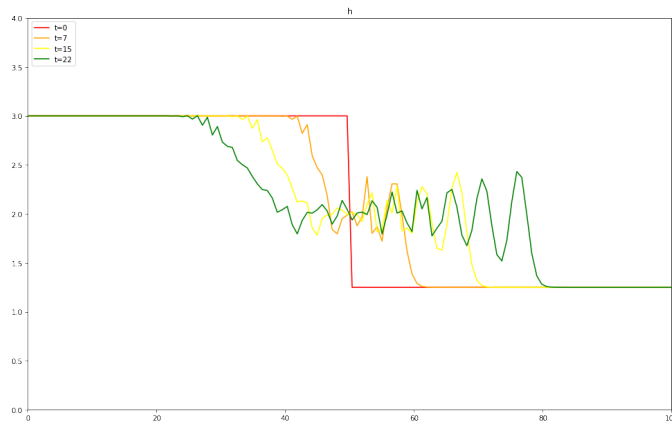
On pourrait d'abord penser à approcher le flux  $\mathcal{F}_{i+1/2}^n$  par la moyenne entre  $x_i$  et  $x_{i+1}$  :

$$\mathcal{F}_{i+1/2} = \frac{F(U_i) + F(U_{i+1})}{2}$$

Et ainsi :

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{2\Delta x} (F(U_{i+1}^n) - F(U_{i-1}^n))$$

Malheureusement, ce flux est instable comme le prouvent ces simulations de rupture de barrage :



Les oscillations de la hauteur d'eau  $h$  sont amplifiées au cours du temps.

### Flux de Lax-Friedrichs

Essayons maintenant avec un terme supplémentaire :

$$\mathcal{F}_{i+1/2} = \frac{F(U_i) + F(U_{i+1})}{2} - \alpha \frac{U_{i+1} - U_i}{2} \quad \text{avec} \quad \alpha = \frac{\Delta x}{\Delta t}$$

On a donc la formule suivante pour calculer  $U_i^{n+1}$  :

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{\Delta x} \left( \frac{F(U_{i-1}) + F(U_{i+1})}{2} - \alpha \frac{U_i - U_{i-1}}{2} - \frac{F(U_{i+1}) + F(U_i)}{2} + \alpha \frac{U_{i+1} - U_i}{2} \right)$$

En simplifiant, car  $\alpha \frac{\Delta t}{\Delta x} = 1$ , on trouve que :

$$U_i^{n+1} = \frac{U_{i+1}^n + U_{i-1}^n}{2} - \frac{\Delta t}{2\Delta x} (F(U_{i+1}^n) - F(U_{i-1}^n))$$

On verra dans la section suivante que l'on peut choisir  $\alpha$  plus finement et ayant les bonnes propriétés en fonction des valeurs propres du système étudié.

### Flux de Rusanov

Revenons tout d'abord aux critères de choix d'un flux numérique. Deux conditions sont importantes : la consistance et la stabilité.

**Définition :** On dit que le schéma est consistant avec le système étudié si :

$$f(U, U) = U \quad \forall U$$

**Définition :** On dit que le schéma est stable s'il préserve un domaine convexe invariant noté  $C$ , autrement dit, que les erreurs ne soient pas amplifiées :

$$U_i^n \in C \Rightarrow U_i^{n+1} \in C$$

En choisissant une valeur d' $\alpha$  bien posée dans le flux de Lax-Friedrichs, on obtient le flux de Rusanov :

$$\mathcal{F}_{i+1/2} = \frac{F(U_i) + F(U_{i+1})}{2} - \alpha \frac{U_{i+1} - U_i}{2}$$

avec  $\alpha = \sup_{U=U_i, U_{i+1}} \sup_{j=1,2} |\lambda_j(U)|$

Ce flux numérique vérifie bien les conditions de consistance et de stabilité.

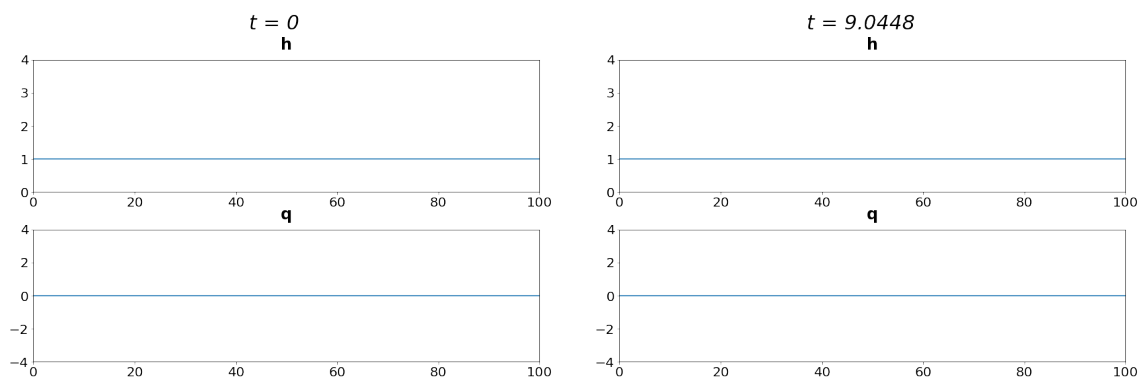
### 3.2 Validation de l'implémentation avec des tests

Dans cette partie, nous allons effectuer des tests de notre implémentation (dont le code est en annexe). Dans un premier temps, nous allons comparer nos résultats à des tests trouvés dans des ouvrages de référence. Dans un second temps, nous allons observer numériquement la convergence des solutions calculées.

#### Quelques exemples

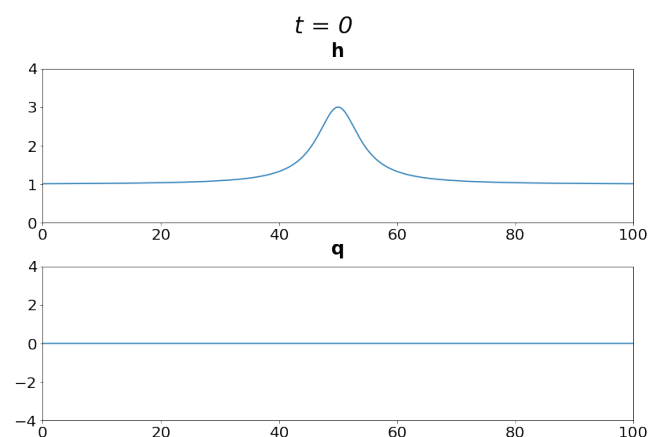
##### Le test du lac

Voyons d'abord un premier test trivial : en prenant une hauteur d'eau plate et un débit initial identiquement égal à zéro, on s'attend à ce que l'eau reste plate au cours du temps. On vérifie alors avec les plots de la figure... que le schéma fonctionne bien sur cet exemple.

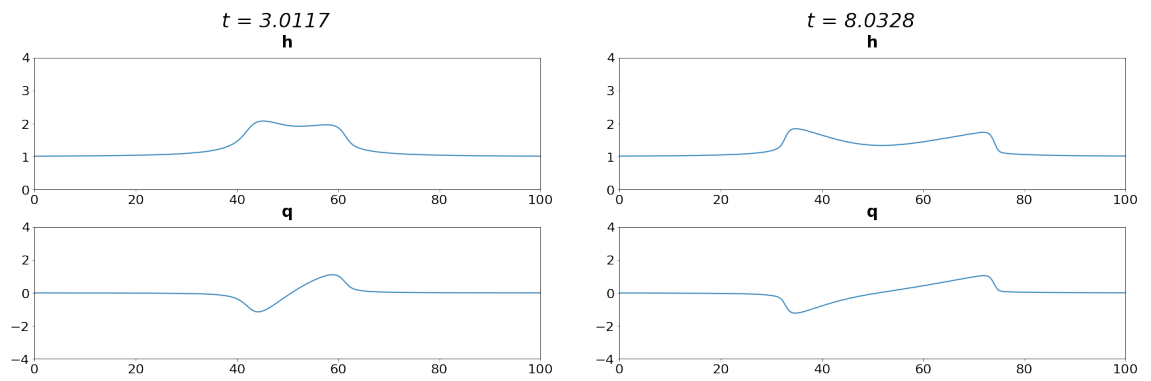


##### Le test de la goutte d'eau

Partons maintenant de la condition initiale suivante : une bosse d'eau au milieu du domaine sans débit initial.

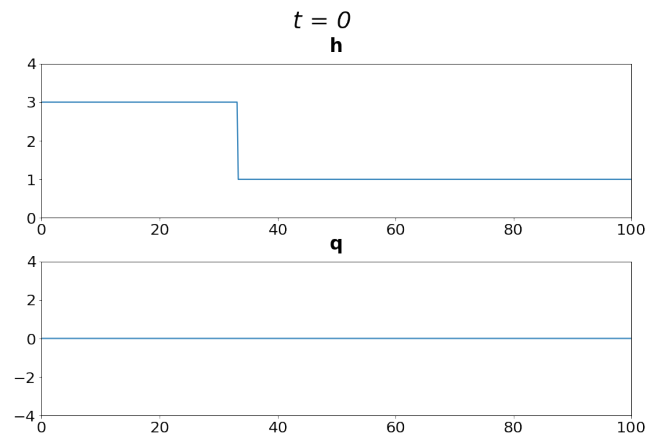


Intuitivement, on s'attend à ce que la « goutte » tombe et forme deux vagues qui partent vers chacun des deux bords du domaine. C'est aussi ce que nous dit la référence [1] à la page 257. Et effectivement, voici ce que l'on observe (après environ 3 secondes et 8 secondes) :

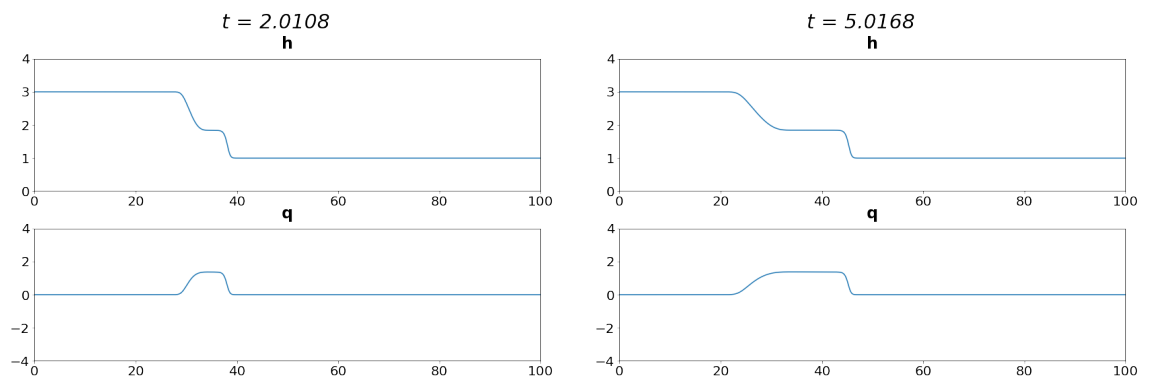


### Le test de la rupture de barrage

Regardons maintenant un dernier test classique dans la littérature : la rupture de barrage.



L'eau forme une courbe en escalier (d'une seule marche) comme si elle était contenue dans une digue. On observe son évolution comme si la digue disparaissait juste après l'instant initial; d'où le nom de rupture de barrage. On observe le résultat suivant :



Ce résultat est conforme à ce que l'on observe dans [1] page 259. La dynamique des solutions obtenues par notre implémentation est donc la bonne sur ces quelques tests.



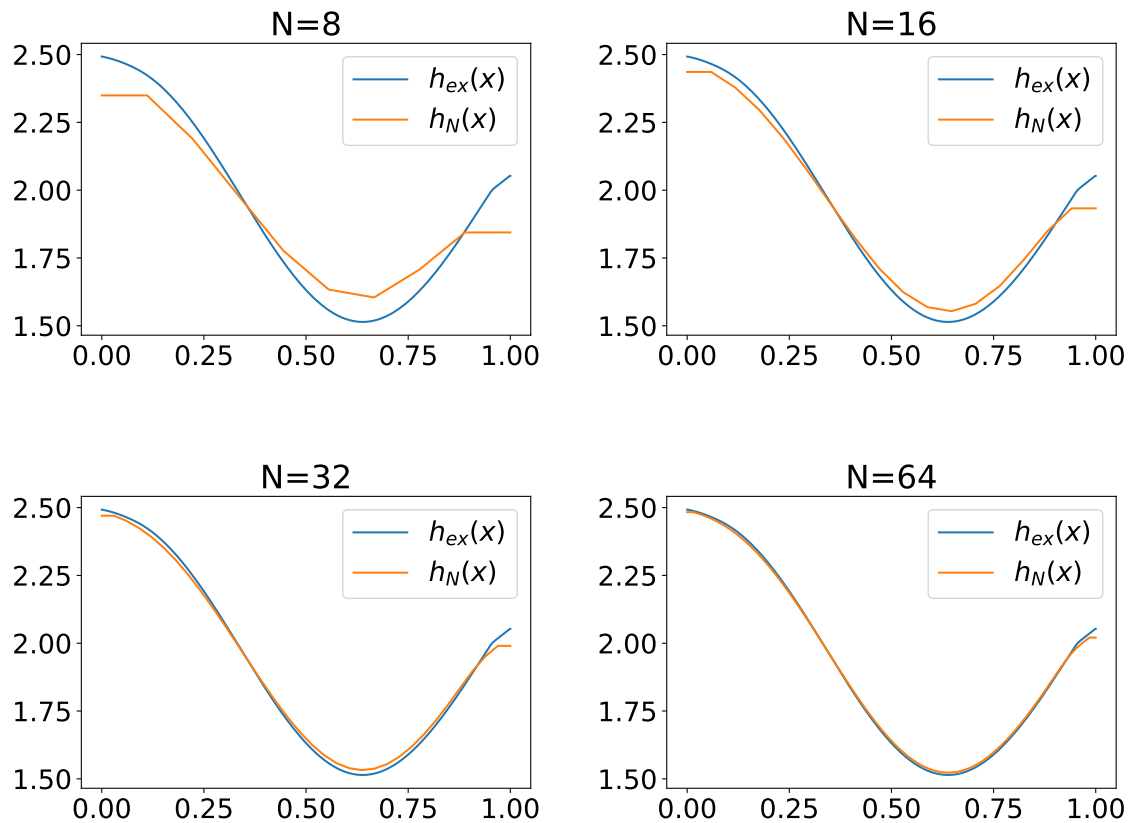
## Influence du pas spatial

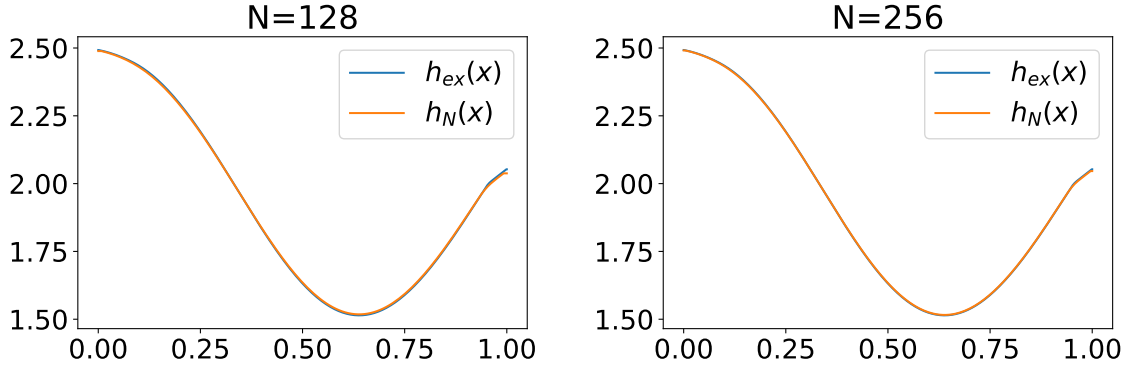
Nous allons faire une étude de convergence de type Cauchy : si la suite des approximations converge vers une limite, alors l'écart entre deux approximations numériques peut être rendu aussi petit qu'on le souhaite, pourvu que les paramètres de grilles soient assez grands (ou petits selon que l'on considère le nombre de points ou le pas).

Puisque le schéma de Rusanov est convergent, on peut considérer que la solution calculée est la solution exacte (ou du moins est très proche de la solution exacte) lorsque le pas d'espace est très petit. Notons  $U_{ex}$  cette solution que l'on considère exacte et  $U_N$  la solution obtenue lorsque le nombre de points (strictement à l'intérieur du domaine) est  $N$ . On peut alors procéder comme il est courant de le faire au calcul de numérique de l'ordre du schéma, en effectuant le plot de  $\max_{x_i} |U_{ex}(x_i) - U_N(x_i)|$  ou de la norme  $L^2$  discrète de  $U_{ex} - U_N$  en fonction de  $N$ ; les  $x_i$  étant les points de la discrétisation.

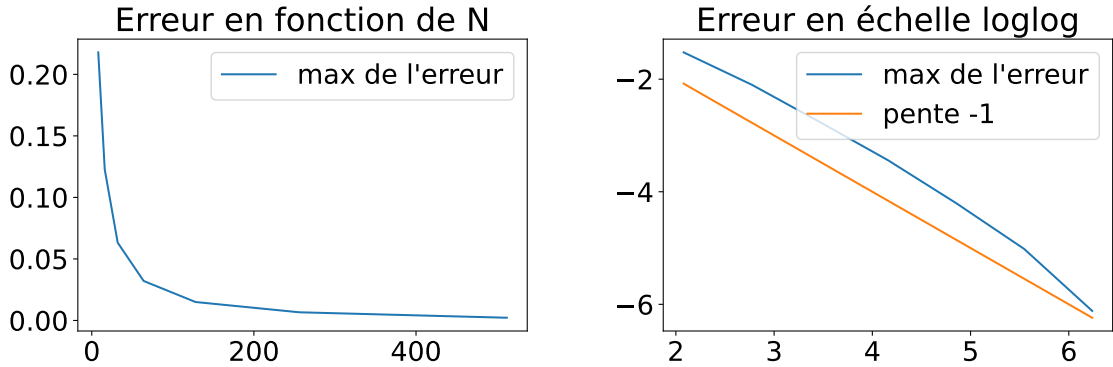
Pour faire cette étude, on prend arbitrairement  $x_{Min} = 0$ ,  $x_{Max} = 1$  pour les bords du domaine et  $T_{max} = 0,03$  pour le temps final de la simulation. Pour les conditions initiales on prend  $h_0(x) = 2 + \cos(5x)/2$  et  $q_0 \equiv 1$  pour la hauteur d'eau et le débit à l'instant initial. La topographie est prise constante égale à 0 ( $Z \equiv 0$ ).

La convergence s'observe d'abord visuellement, en faisant le plot des solutions calculées pour des valeurs de  $N$  de plus en plus grandes. On peut par exemple représenter graphiquement la hauteur d'eau de la solution  $U_{ex}$  avec celle de  $U_N$ . Dans la suite  $h_{ex}$  et  $h_N$  désigneront ces quantités. Les figures ci-dessous ont été obtenues avec  $N = 2^k$ ,  $k$  allant de 3 à 9.





Regardons maintenant le plot du maximum de l'erreur entre  $h_{ex}$  et  $h_N$ . Pour ce faire, on retient les points  $x_i$  communs aux deux maillages. Si la solution  $U_{ex}$  correspond à une discrétisation de  $N = 2^l$  point intérieurs (comme c'est le cas dans notre implémentation), alors le maillage correspondant contiendra tous les maillages de  $N = 2^k$  point intérieurs avec  $k \leq l$ . Ainsi, les points  $x_i$  communs aux deux maillages sont ceux du maillage pour  $U_N$ . Dès lors, on calcule facilement  $\max_{x_i} |U_{ex}(x_i) - U_N(x_i)|$ . Voici les représentations obtenues avec cette manière de calculer l'erreur :



Voyons maintenant l'erreur en norme  $L^2$  discrète. Avant ça, définissons cette notion. La norme  $L^2(a, b)$  d'une fonction  $u$  de carré intégrable est donnée par

$$\|u\|_{L^2(a,b)} = \left( \int_a^b u^2(x) dx \right)^{\frac{1}{2}}.$$

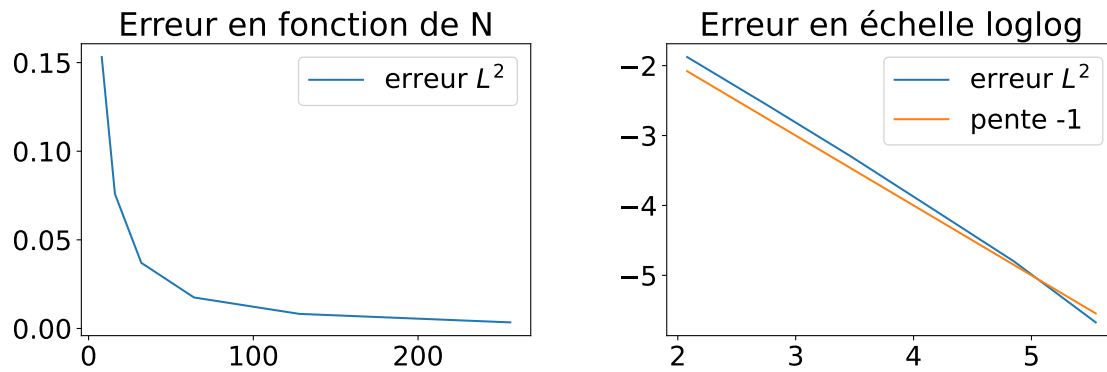
Si  $x_0 = a < x_1 < \dots < x_N = b$  discrétise  $[a, b]$ , alors par la relation de Chasles, on a

$$\int_a^b u^2(x) dx = \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} u^2(x) dx \approx \sum_{i=0}^N u^2(x_i)(x_{i+1} - x_i).$$

On a alors un analogue discret de la norme  $L^2$  habituelle donné dans la définition qui suit.

**Définition 2.** Soit  $N \in \mathbb{N}$  et  $x_0 = a < x_1 < \dots < x_{N+1} = b$  la discrétisation uniforme de  $[a, b]$  composée de  $N + 2$  points ( $N$  à l'intérieur et 2 aux bords). Posons  $h = \frac{b-a}{N+1}$  le pas du maillage. Soit  $U = (u_i)_{0 \leq i \leq N+1}$  une approximation de la solution aux points du maillage. On définit la norme  $L^2$  discrète de  $U$  par  $\|U\|_{L^2} = (h \sum_{i=0}^{N+1} u_i^2)^{\frac{1}{2}}$ .

Voici les erreurs obtenues avec cette norme :



En échelle loglog, on observe à chaque fois que l'erreur décroît selon une pente -1. Sur cet exemple, le schéma est donc d'ordre 1.

Des résultats similaires sont obtenus lorsque l'on affiche le débit  $q$  plutôt que la hauteur d'eau  $h$ . De la même façon, on peut faire varier les paramètres **xMin**, **xMax**, **Tmax** ainsi que les conditions initiales et on arrive aux mêmes conclusions.

**Influence du pas temporel (va être retravaillé)**

## A Annexes

### A.1 Code de l'implémentation du schéma de Rusanov en 1D

Voici le code à partir duquel nous avons obtenu les graphes du rapport. Ce code permet aussi de générer des fichiers .gif et ainsi de visualiser les évolutions (hauteur d'eau et débit) au cours du temps.

#### Importations

```
[1]: import math as math # Pour les fonctions math
import matplotlib.pyplot as plt # Pour l'affichage des graphes
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
from matplotlib.figure import Figure
import numpy as np # Pour les tableaux numpy
import imageio # Pour faire des .gif
```

#### Paramètres du problème

```
[2]: g=1 # Constante gravitationnelle

xMin=0 # Bord gauche du domaine
xMax=100 # Bord droit du domaine

N=512 # Nombre de points (strictement à l'intérieur)
# Donc au total, on considère N+2 points avec les bords
h=(xMax-xMin)/(N+1) # Pas du maillage spatial

Tmax=10 # Temps final de la simulation
t=0 # Temps dans la simulation
n=0 # Nombre d'itérations
```

#### Conditions initiales et Topographie

Dans la cellule ci-dessous, nous définissons la discrétisation de l'axe des abscisses  $X$ , ainsi que la topographie  $Z$  et les profils initiaux de la hauteur d'eau  $h$  et du débit  $q$ . Le code en commentaire correspond à différentes initialisations possibles. Pour la suite, notons  $U = \begin{pmatrix} h \\ q \end{pmatrix} \in \mathbb{R}^2$  le vecteur inconnu.

```
[3]: X=np.linspace(xMin,xMax,N+2) # Discrétisation de [xMin, xMax]
U=np.ones([2,N+2]) # discrétisation du vecteur (h, q)
Uprime=np.zeros([2,N+2]) # Va servir d'intermédiaire de calcul
Z=np.zeros(N+2) # Discrétisation du fond Z
```

```

# Définitions de Z -----

# Conditions initiales avec Z en créneau :
#  $Z[(N+2)//5:2*(N+2)//5]=0.3$ 
#  $Z[3*(N+2)//5:4*(N+2)//5]=0.3$ 

# Conditions initiales avec Z en bosse C infini :
#  $Z=1/(1+.1*(X-50)**2)$ 

# Conditions initiales avec Z en escalier descendant :
#  $Z[0:(N+2)//4]=.75$ 
#  $Z[(N+2)//4:3*(N+2)//4]=0.25$ 

# Conditions initiales avec Z en tangente hyperbolique :
#  $Z=np.tanh(5-X)/2+.5$ 

# Définitions du h initial -----

# Hauteur de l'eau en escalier
U[0,0:(N+2)//3]=3-Z[: (N+2)//3]
#  $U[0,(N+2)//3:N+2]=1.25-Z[(N+2)//3:N+2]$ 

# Hauteur de l'eau constante
#  $U[0,:]=1-Z$ 

# Hauteur de l'eau en bosse
#  $U[0,:]=1+2/(1+.05*(X-50)**2)-Z$ 

# Définitions du q initial -----

# Débit de l'eau constant
U[1,:]=0

```

### Nombre de sauvegardes durant la simulation

Quelques variables qui vont nous servir pour enregistrer des étapes de la simulation, sans pour autant les sauvegarder toutes. Sans ça, le nombre d'images à sauver deviendrait vite trop important.

```

[4]: nSauvegarde=10 # Nombre de sauvegardes au cours de la simulation
tSauvegarde=[False for i in range(nSauvegarde)]
images=[]
j=0 # Nombre de sauvegardes déjà effectuées

```

## Affichage des images

La fonction ci-dessous permet de gérer l'affichage de la solution. Elle est suivie d'un test. Les graphes qui s'affichent sont ceux de  $h$  et  $q$  à  $t = 0$ .

```
[5]: def affiche_U(t):

    t_int=round(t,4) # Troncature de t après la 4eme décimale

    fig,axs=plt.subplots(2,1,figsize=(12,8))

    fig.suptitle("t = "+str(t_int),style='italic',size=30)
    fig.tight_layout()

    plt.rc('font', size=20)

    h_z=U[0,:]+Z

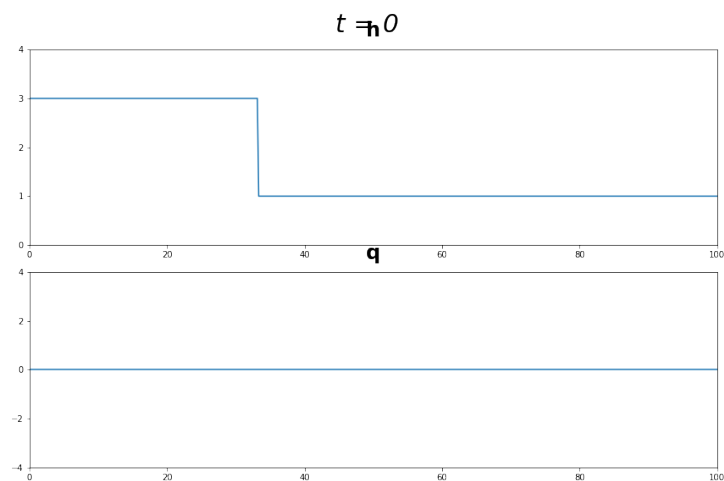
    axs[0].plot(X,h_z)
    axs[0].set_title("h",fontweight="bold",pad=15)
    axs[0].set_xlim([xMin,xMax])
    axs[0].set_ylim([0,4])
    axs[0].fill_between(X,Z,step="pre",alpha=0.5,color="grey")

    axs[1].plot(X,U[1,:])
    axs[1].set_title("q", fontweight="bold",pad=15)
    axs[1].set_xlim([xMin,xMax])
    axs[1].set_ylim([-4,4])

    # axs[1].set_xlabel("Quantité d'eau : "+str(round(sum(U[0,:
    →]))*h,2)))

    plt.show()

affiche_U(0) # Test d'affichage de la solution
```



## Sauvegarde des images

`enregistre_U` effectue un traitement identique à `affiche_U`. La seule différence est qu'elle enregistre le plot dans le répertoire courant plutôt que de l'afficher.

```
[6]: def enregistre_U(n,t):

    t_int=round(t,4) # Troncature de t après la 4eme décimale

    fig,axs=plt.subplots(2,1,figsize=(12,8))

    fig.suptitle("t = "+str(t_int),style='italic',size=30)
    fig.tight_layout()

    plt.rc('font',size=20)

    h_z = U[0,:]+Z

    axs[0].plot(X,h_z)
    axs[0].set_title("h", fontweight="bold", pad=15)
    axs[0].set_xlim([xMin,xMax])
    axs[0].set_ylim([0,4])
    axs[0].fill_between(X,Z,step="pre",alpha=0.5,color="grey")

    axs[1].plot(X,U[1,:])
    axs[1].set_title("q", fontweight="bold", pad=15)
    axs[1].set_xlim([xMin,xMax])
    axs[1].set_ylim([-4,4])

    # axs[1].set_xlabel("Quantité d'eau : "+str(round(sum(U[0,:
    ↪]))*h,2)))

    # To remove the huge white borders
    axs[0].margins(0)
    axs[1].margins(0)

    fig.canvas.draw()
    image_from_plot=np.frombuffer(fig.canvas.tostring_rgb(),
    ↪dtype=np.uint8)
    image_from_plot=image_from_plot.reshape(fig.canvas.
    ↪get_width_height()[::-1]+(3,))

    images.append(image_from_plot)

    #Sauvegarde dans un fichier .png
    plt.savefig("etape"+str(n)+".png")
```

```
plt.close()

enregistre_U(0,0)
```

### Fonctions qui interviennent dans le schéma

$\mathbf{U} = \begin{pmatrix} h \\ q \end{pmatrix} \in \mathbb{R}^2$  est le vecteur inconnu. Les différentes fonctions du schéma qu'il s'agit d'implémenter sont  $\mathbf{F}(\mathbf{U})$ ,  $\mathbf{B}(\mathbf{U})$  et  $\mathcal{F}_{i+\frac{1}{2}}^n$ . Rappelons les différentes définitions :

$$\mathbf{F}(\mathbf{U}) = \begin{pmatrix} q \\ \frac{q^2}{h} + g\frac{h^2}{2} \end{pmatrix} \in \mathbb{R}^2 \text{ et } \mathbf{B}(\mathbf{U}) = \begin{pmatrix} 0 \\ -gh\frac{\partial Z}{\partial x} \end{pmatrix} \in \mathbb{R}^2.$$

$\mathbf{F}(\mathbf{U})$  désigne la fonction flux et  $\mathbf{B}(\mathbf{U})$  le terme source. Avec ces notations, le système de départ se réécrit :

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial}{\partial x}(\mathbf{F}(\mathbf{U})) = \mathbf{B}(\mathbf{U}).$$

Et si l'on suppose que  $Z \equiv 0$ , alors le schéma s'écrit simplement :

$$\frac{\mathbf{U}_i^{n+1} - \mathbf{U}_i^n}{\Delta t^n} + \frac{\mathcal{F}_{i+\frac{1}{2}}^n - \mathcal{F}_{i-\frac{1}{2}}^n}{\Delta x_i} = 0$$

Avec

$$\mathcal{F}_{i+\frac{1}{2}}^n = \frac{\mathbf{F}(\mathbf{U}_i^n) + \mathbf{F}(\mathbf{U}_{i+1}^n)}{2} - \max_{j \in \{i, i+1\}} \max_{k \in \{1, 2\}} |\lambda_k(\mathbf{U}_j^n)| \frac{\mathbf{U}_{i+1}^n - \mathbf{U}_i^n}{2}.$$

En prenant une topographie  $Z$  non triviale, on peut essayer de prendre le même schéma en rajoutant le terme source. Si l'on a accès à la dérivée  $Z'$  de  $Z$ , le schéma s'écrit alors :

$$\frac{\mathbf{U}_i^{n+1} - \mathbf{U}_i^n}{\Delta t^n} + \frac{\mathcal{F}_{i+\frac{1}{2}}^n - \mathcal{F}_{i-\frac{1}{2}}^n}{\Delta x_i} = \begin{pmatrix} 0 \\ -gh_i Z'_i \end{pmatrix}$$

ou bien en utilisant des différences finies centrées pour approcher les dérivées :

$$\frac{\mathbf{U}_i^{n+1} - \mathbf{U}_i^n}{\Delta t^n} + \frac{\mathcal{F}_{i+\frac{1}{2}}^n - \mathcal{F}_{i-\frac{1}{2}}^n}{\Delta x_i} = \begin{pmatrix} 0 \\ -gh_i \frac{Z_{i+1} - Z_{i-1}}{2\Delta x_i} \end{pmatrix}$$

```
[7]: # Fonction F du schéma :
# Entrée : vecteur U = (h, q) (type numpy.ndarray)
# Sortie : F(U) (type numpy.ndarray)
def F(U):
    return np.array([U[1], U[1]**2/U[0]+g*U[0]**2/2])

# Fonction B du schéma :
```



```

# Entrées : * vecteur U = (h, q) (type numpy.ndarray)
#           * valeur dxZ de la dérivée spatiale de Z
# Sortie : B(U) (type numpy.ndarray)
def B(U,i):
    return np.array([0,-g*U[0]*(Z[i+1]-Z[i-1])/2])

# Fonction qui retourne la valeur propre max en module
# Entrées : * U solution discrétisée (type numpy.ndarray de taille_
↳ 2*(N+2))
#           * indice i de la position spatiale
# Sortie : max_j max_k |lambda_k(U_j^n)|
def vmax(U,i):
    res = abs(U[1,i]/U[0,i]+math.sqrt(g*U[0,i]))
    res = max(res,abs(U[1,i]/U[0,i]-math.sqrt(g*U[0,i])))
    res = max(res,abs(U[1,i+1]/U[0,i+1]+math.sqrt(g*U[0,i+1])))
    res = max(res,abs(U[1,i+1]/U[0,i+1]-math.sqrt(g*U[0,i+1])))
    return res

# Fonction flux numérique :
# Entrées : * U solution discrétisée (type numpy.ndarray de taille_
↳ 2*(N+2))
#           * indice i de la position spatiale
# Sortie : F_{i+1/2}^n
def F_ronde(U,i):
    return (F(U[:,i])+F(U[:,i+1]))/2-vmax(U,i)*(U[:,i+1]-U[:,i])/2

```

### Boucle de résolution numérique

Une fois le pas de temps  $\Delta t^n$  déterminé selon le critère qui garanti le respect de la conditions CFL, on passe du temps  $t^n$  au temps  $t^{n+1}$  suivant la relation explicite suivante :

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n - \Delta t^n \frac{\mathcal{F}_{i+\frac{1}{2}}^n - \Delta t^n \mathcal{F}_{i-\frac{1}{2}}^n}{\Delta x_i} + \Delta t^n \begin{pmatrix} 0 \\ -gh_i \frac{Z_{i+1} - Z_{i-1}}{2\Delta x_i} \end{pmatrix}$$

[8]: `while(t<Tmax): # Tant que le temps max n'est pas atteint :`

```

    M = vmax(U, 0)
    for i in range(1, N+1):
        M = max(M, vmax(U, i))

    # Pour assurer la stabilité, tau doit être inférieur à h/(2*_
↳ max(vp))
    tau = 0.8*h/(2*M)

    for i in range(1, N+1):
        Uprime[:,i] = U[:,i]+tau/h*(F_ronde(U,i-1)-F_ronde(U,i))
        Uprime[:,i] -= tau*B(U[:,i],i)

```

```

U = Uprime

# Conditions aux bords
U[:,0] = U[:,1]
U[:,N+1] = U[:,N]

t+=tau
n+=1

if (t > Tmax/nSauvegarde*j and (j<nSauvegarde) and (not t_
→tSauvegarde[j])):
    enregistre_U(n,t)
    #affiche_U()
    tSauvegarde[j]=True
    j+=1

print("Nombre d'itérations : " + str(n))
imageio.mimsave('movie.gif', images)
print("Gif Sauvegardé dans le dossier sous le nom : movie.gif")

```

## A.2 Code de l'implémentation du schéma de Rusanov en 2D

### Équations dans le cas 2D

Les équations de Saint-Venant dans en 2D sont les suivantes :

$$\frac{\partial h}{\partial t} + \frac{\partial hu}{\partial x} + \frac{\partial hv}{\partial y} = 0$$

$$\frac{\partial hu}{\partial t} + \frac{\partial}{\partial x}(hu^2 + \frac{gh^2}{2}) + \frac{\partial}{\partial y}(huv + \frac{gh^2}{2}) = 0$$

$$\frac{\partial hv}{\partial t} + \frac{\partial}{\partial x}(huv + \frac{gh^2}{2}) + \frac{\partial}{\partial y}(hv^2 + \frac{gh^2}{2}) = 0$$

On fait l'implémentation du cas 2D sur le même modèle que dans le cas 1D.

### Importations

```
[1]: import math as math # Pour les fonctions math

import matplotlib.pyplot as plt # Pour l'affichage des graphes
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
from matplotlib.figure import Figure
from mpl_toolkits import mplot3d

plt.rc('font', size=20)

import numpy as np # Pour les tableaux numpy
import imageio # Pour faire des .gif
```

### Paramètres du problème

```
[1]: g=1 # Constante gravitationnelle

xMin=0 # Bord gauche du domaine
xMax=100 # Bord droit du domaine

yMin = 0 #Bord bas du domaine
yMax = 100 #Bord haut du domaine

N=64 # Nombre de points (strictement à l'intérieur)
# Donc au total, on considère (N+2)^2 points avec les bords

h=(xMax-xMin)/(N+1) # Pas du maillage spatial, le même pour X et Y
```

```
Tmax=10 # Temps final de la simulation
t=0 # Temps dans la simulation
n=0 # Nombre d'itérations
```

### Conditions initiales

Dans la cellule ci-dessous, nous définissons la discrétisation de l'axe des abscisses  $X$ , ainsi que les profils initiaux de la hauteur d'eau  $h$  et des débits  $hu$  et  $hv$ .

```
[72]: X,Y=np.mgrid[xMin:xMax+2*h/3:h,yMin:yMax+2*h/3:h] # Discrétisation
      ↪ de [xMin, xMax] et [yMin, yMax]
      XY = X+Y*1j

      W0 = np.ones([N+2,N+2]) # Discrétisation de h
      W1 = np.ones([N+2,N+2]) # Discrétisation de hu
      W2 = np.ones([N+2,N+2]) # Discrétisation de hv

      # Définitions du h initial -----

      # Hauteur de l'eau en escalier
      # W0[0,0:(N+2)//3] = 3
      # W0[0,(N+2)//3:N+2] = 1.25

      # Hauteur de l'eau constante
      # W0[0,:]=1

      # Hauteur de l'eau en bosse
      W0 = 1+5/(4 + .01*( (X-50)**2 + (Y-50)**2 ) )

      # Définitions du débit initial -----

      # Débit de l'eau constant
      W1[:,:]=0
      W2[:,:]=0
```

### Nombre de sauvegardes durant la simulation

Quelques variables qui vont nous servir pour enregistrer des étapes de la simulation, sans pour autant les sauvegarder toutes. Sans ça, le nombre d'images à sauver deviendrait vite trop important.

```
[73]: nSauvegarde=100 # Nombre de sauvegardes au cours de la simulation
      tSauvegarde=[False for i in range(nSauvegarde)]
      images=[]
      k=0 # Nombre de sauvegardes déjà effectuées
```

## Affichage des images

La fonction ci-dessous permet de gérer l’affichage de la solution. Elle est suivie d’un test. Les graphes qui s’affichent sont ceux de  $h$  et  $q$  à  $t = 0$ .

```
[74]: def affiche_W(t=0,index=0):

    #index : suivant que l'on veuille afficher h, u ou v

    t_int=round(t,4) # Troncature de t après la 4eme décimale

    fig = plt.figure(figsize =(14, 9))
    ax = plt.axes(projection ='3d')

    if(index==0):
        ax.plot_surface(X, Y, W0)
        ax.set_title("h",fontweight="bold",pad=15)
        ax.set_zlim([0,4])
    elif(index==1):
        ax.plot_surface(X, Y, W1)
        ax.set_title("hu",fontweight="bold",pad=15)
        ax.set_zlim([-1,1])
    elif(index==2):
        ax.plot_surface(X, Y, W2)
        ax.set_title("hv",fontweight="bold",pad=15)
        ax.set_zlim([-1,1])
    else:
        print("Mauvaise valeur d'index")

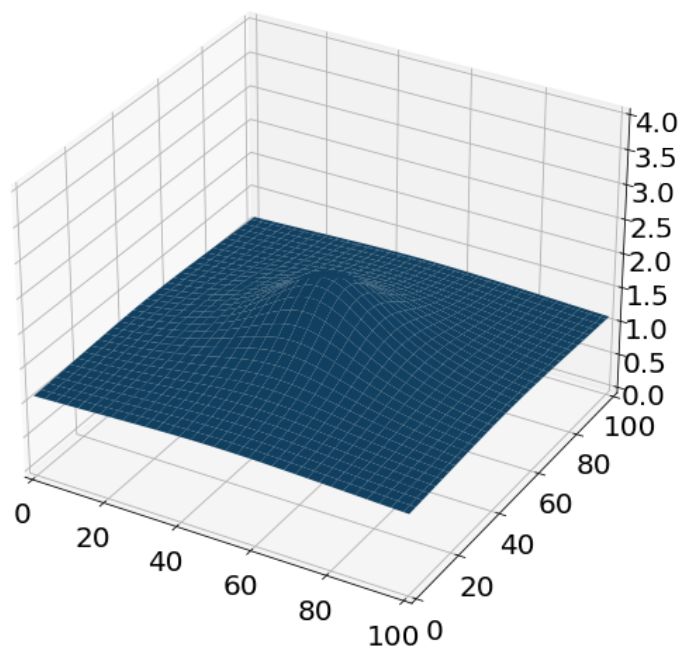
    fig.suptitle("t = "+str(t_int),style='italic',size=30)

    ax.set_xlim([xMin,xMax])
    ax.set_ylim([yMin,yMax])

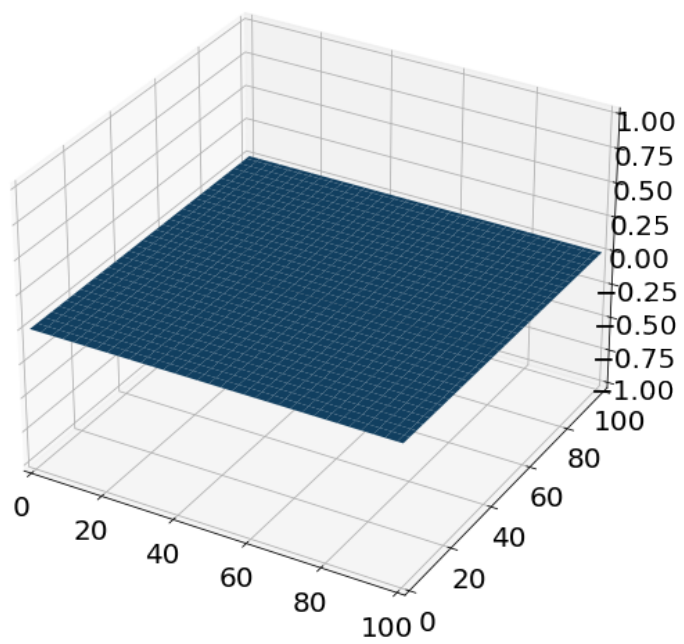
    plt.show()

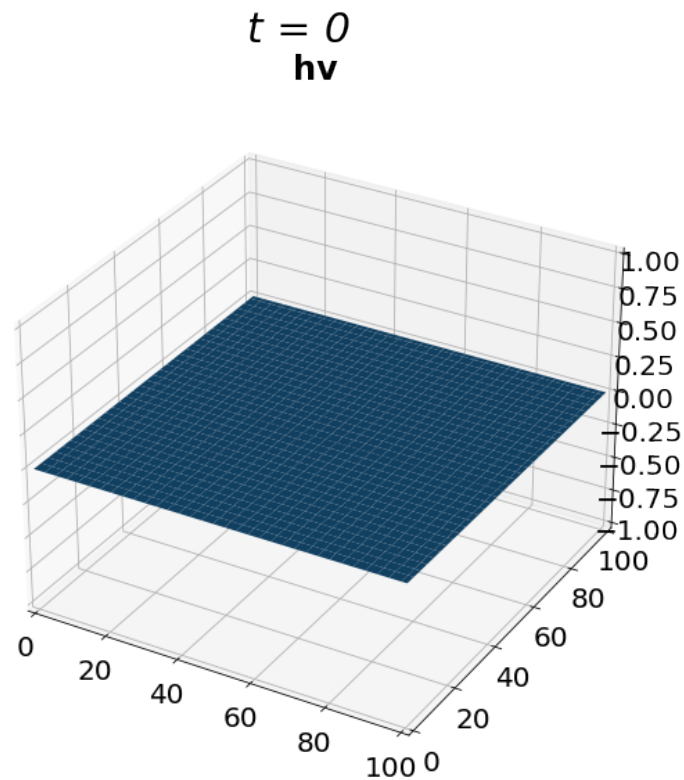
#Affichage de h, u et v
affiche_W()
affiche_W(index=1)
affiche_W(index=2)
```

$$t = 0$$
$$\mathbf{h}$$



$$t = 0$$
$$\mathbf{hu}$$





### Sauvegarde des images

`enregistre_W` effectue un traitement identique à `affiche_W`. La seule différence est qu'elle enregistre le plot dans le répertoire courant plutôt que de l'afficher.

```
[75]: def enregistre_W(t,index=0):

    t_int=round(t,4) # Troncature de t après la 4eme décimale

    fig = plt.figure(figsize =(14, 9))
    ax = plt.axes(projection ='3d')

    if(index==0):
        ax.plot_surface(X, Y, W0)
        ax.set_title("h",fontweight="bold",pad=15)
        ax.set_zlim([0,4])
    elif(index==1):
        ax.plot_surface(X, Y, W1)
        ax.set_title("hu",fontweight="bold",pad=15)
        ax.set_zlim([-1,1])
    elif(index==2):
        ax.plot_surface(X, Y, W2)
        ax.set_title("hv",fontweight="bold",pad=15)
        ax.set_zlim([-1,1])
```

```

else:
    print("Mauvaise valeur d'index")

fig.suptitle("t = "+str(t_int),style='italic',size=30)

ax.set_xlim([xMin,xMax])
ax.set_ylim([yMin,yMax])

fig.canvas.draw()
image_from_plot=np.frombuffer(fig.canvas.tostring_rgb(),
dtype=np.uint8)
image_from_plot=image_from_plot.reshape(fig.canvas.
get_width_height()[::-1]+(3,))

images.append(image_from_plot)

plt.close()

enregistre_W(0,index=0)

```

### Fonctions intervenant dans le schéma

En posant :  $W = \begin{bmatrix} h \\ hu \\ hv \end{bmatrix}$  ;  $F(W) = \begin{bmatrix} hu \\ hu^2 + \frac{gh^2}{2} \\ huv + \frac{gh^2}{2} \end{bmatrix}$  ;  $G(W) = \begin{bmatrix} hv \\ huv + \frac{gh^2}{2} \\ hv^2 + \frac{gh^2}{2} \end{bmatrix}$

Avec  $h$  la hauteur de l'eau,  $u$  sa vitesse suivant la direction  $x$  et  $v$  sa vitesse suivant la direction  $y$ .

On a le système suivant :

$$\frac{\partial W}{\partial t} + \frac{\partial F(W)}{\partial x} + \frac{\partial G(W)}{\partial y} = 0$$

En identifiant  $F(W)$  au flux suivant  $x$  et  $G(W)$  au flux suivant  $y$ .

En s'inspirant de ce qui a été fait dans le cas 1D, on discrétise de la manière suivante :

$$\frac{W_{i,j}^{n+1} - W_{i,j}^n}{\Delta t} + \frac{F_{i+\frac{1}{2},j}^n - F_{i-\frac{1}{2},j}^n}{\Delta x} + \frac{G_{i,j+\frac{1}{2}}^n - G_{i,j-\frac{1}{2}}^n}{\Delta y} = 0$$

Ce qui nous donne finalement :

$$W_{i,j}^{n+1} = W_{i,j}^n - \frac{\Delta t}{\Delta x} (F_{i+\frac{1}{2},j}^n - F_{i-\frac{1}{2},j}^n) - \frac{\Delta t}{\Delta y} (G_{i,j+\frac{1}{2}}^n - G_{i,j-\frac{1}{2}}^n)$$

Avec :



$$F_{i+\frac{1}{2},j} = \frac{F(W_{i+1,j}) + F(W_{i,j})}{2} - c \frac{W_{i+1,j} - W_{i,j}}{2} \quad o \quad c = \sup_{W=W_{i,j}, W_{i+1,j}} \sup_{k=1,2} |\lambda_k(W)|$$

et

$$G_{i,j+\frac{1}{2}} = \frac{G(W_{i,j+1}) + G(W_{i,j})}{2} - c \frac{W_{i,j+1} - W_{i,j}}{2} \quad o \quad c = \sup_{W=W_{i,j}, W_{i,j+1}} \sup_{k=1,2} |\lambda_k(W)|$$

```
[76]: def F(w0,w1,w2):
    return np.array([w1 , w1**2 / w0 + g * w0**2 / 2 , w1*w2/w0 + g
    ↪ * w0**2])

def G(w0,w1,w2):
    return np.array([w2 , w1*w2/ w0 + g * w0**2 / 2 , w2**2 /w0 + g
    ↪ * w0**2])

def vmaxF(i,j): #vmax du premier système : u , u+sqrt(gh) ou
    ↪ u-sqrt(gh)

    res = W1[i,j] / W0[i,j]

    res = max(res,abs( W1[i,j]/W0[i,j] + math.sqrt(g*W0[i,j])))
    res = max(res,abs( W1[i,j]/W0[i,j] - math.sqrt(g*W0[i,j])))

    res = max(res,abs( W1[i+1,j]/W0[i+1,j] + math.sqrt(g*W0[i+1,j])))
    res = max(res,abs( W1[i+1,j]/W0[i+1,j] - math.sqrt(g*W0[i+1,j])))

    return res

def vmaxG(i,j): #vmax du deuxième système : v , v+sqrt(gh) ou
    ↪ v-sqrt(gh)

    res = W2[i,j] / W0[i,j]

    res = max(res,abs( W2[i,j]/W0[i,j] + math.sqrt(g*W0[i,j])))
    res = max(res,abs( W2[i,j]/W0[i,j] - math.sqrt(g*W0[i,j])))

    res = max(res,abs( W2[i,j+1]/W0[i,j+1] + math.sqrt(g*W0[i,j+1])))
    res = max(res,abs( W2[i,j+1]/W0[i,j+1] - math.sqrt(g*W0[i,j+1])))

    return(res)

def F_ronde(i,j):
    termeW = np.array([ W0[i+1,j]-W0[i,j] , W1[i+1,j]-W1[i,j]
    ↪ ,W2[i+1,j]-W2[i,j] ] )/2
```

```

    return
    →(F(W0[i+1,j],W1[i+1,j],W2[i+1,j])+F(W0[i,j],W1[i,j],W2[i,j]))/
    →2-vmaxF(i,j) * termeW

def G_ronde(i,j):
    termeW = np.array([ W0[i,j+1]-W0[i,j] , W1[i,j]-W1[i,j+1]
    →,W2[i,j+1]-W2[i,j] ] )/2

    return
    →(G(W0[i,j+1],W1[i,j+1],W2[i,j+1])+G(W0[i,j],W1[i,j],W2[i,j]))/
    →2-vmaxG(i,j) * termeW

```

```

[77]: def conditionBord():

    # Conditions aux bords sur h
    W0[:,0] = W0[:,1]
    W0[:,N+1] = W0[:,N]
    W0[0,:] = W0[1,:]
    W0[N+1,:] = W0[N,:]

    # Conditions aux bords sur hu
    W1[:,0] = W1[:,1]
    W1[:,N+1] = W1[:,N]
    W1[0,:] = W1[1,:]
    W1[N+1,:] = W1[N,:]

    # Conditions aux bords sur hv
    W2[:,0] = W2[:,1]
    W2[:,N+1] = W2[:,N]
    W2[0,:] = W2[1,:]
    W2[N+1,:] = W2[N,:]

```

Pour rappel :

$$W_{i,j}^{n+1} = W_{i,j}^n - \frac{\Delta t}{\Delta x} (F_{i+\frac{1}{2},j}^n - F_{i-\frac{1}{2},j}^n) - \frac{\Delta t}{\Delta y} (G_{i,j+\frac{1}{2}}^n - G_{i,j-\frac{1}{2}}^n)$$

```

[78]: while(t<Tmax): # Tant que le temps max n'est pas atteint :

    M = 0
    for i in range(0, N+1):
        for j in range(0,N+1):
            M = max(M, vmaxF(i,j))
            M = max(M, vmaxG(i,j))

    # Pour assurer la stabilité, tau doit être inférieur à h/(2*
    →max(vp))
    tau = 0.8*h/(2*M)

```

```

for i in range(1, N+1):
    for j in range(1, N+1):

        W = np.array([ W0[i,j],W1[i,j],W2[i,j] ])
        W = W - tau/h * ( F_ronde(i,j) - F_ronde(i-1,j) ) #Terme
↪avec les F ronde
        W = W - tau/h * ( G_ronde(i,j) - G_ronde(i,j-1) ) #Terme
↪avec les G ronde

        W0[i,j] = W[0]
        W1[i,j] = W[1]
        W2[i,j] = W[2]

    conditionBord() #Conditions aux bords sur h , hu et hv

    t+=tau
    n+=1

    print("Nombre d'itérations : " + str(n) + " | t = "+ str(t))

    if (t > Tmax/nSauvegarde*k and (k<nSauvegarde) and (not
↪tSauvegarde[k])):
        enregistre_W(t,index=0)
        tSauvegarde[j]=True
        k+=1

imageio.mimsave('movie.gif', images)
print("Gif Sauvegardé dans le dossier sous le nom : movie.gif")

```



# Bibliographie

- [1] R. LEVEQUE, *Finite Volume Methods for Hyperbolic Problems*, Cambridge University Press, 2004
- [2] E. TORO, *Riemann Solvers and Numerical Methods for Fluid Dynamic*, Springer, 2009
- [3] WIKIPEDIA, *Intégrale paramétrique*