UNIVERSITÉ DE STRASBOURG
UFR DE MATHÉMATIQUES ET INFORMATIQUE

TP #1

Etudiant : *Roussel Desmond Nzoyem*

UE : *EDP 2* – Enseignant : *Pr. Philippe Helluy*
Date : *3 janvier 2021*

1. Résolution numérique de l'équation de transport

Question 1.

Calculer la solution du problème suivant (c est une constante > 0).

$$\begin{cases} \partial_t u + c\partial_x u = 0, \quad x \in]0, L[, \quad t \in]0, T[\\ u(0,t) = e^{-t}, \quad t \in]0, T[\\ u(x,0) = 0, \quad x \in]0, L[\end{cases}$$

Réponse

Nous procéderons par la méthode des courbes caractéristiques. Une courbe caractéristique est une courbe $t \mapsto \binom{x(t)}{t}$ telle que la solution $u(x(t),t)$ y soit constante. Pour l'équation de transport à résoudre, les courbes caractéristiques sont des droites du plan de coefficient directeur $c > 0$. Calculons la solution du problème.

■ Dans un premier temps, pour tout $y \in \mathbb{R}$, on a :

$$\frac{d}{dt}u(y+ct,t) = \left(\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x}\right)(y+ct,t) = 0, \qquad 0 < y + ct < L, \quad 0 < t < T$$

C'est à dire

$$u(y+ct,t) = \text{Cste}, \qquad 0 < y + ct < L, \quad 0 < t < T$$

On choisi la constante correspondant au cas limite $t = 0$

$$u(y+ct,t) = u(y,0)$$

sous les conditions suivantes

— $\begin{cases} 0 < y + ct < L \\ 0 < y < L \end{cases} \Longrightarrow ct < y + ct < L$

— $0 < t < T$

On effectue le changement de variable $y + ct = x$ pour obtenir la solution

$$u(x,t) = 0$$

définie sur l'ensemble :

$$\{(x,t) \in]0, L[\times]0, T[\quad \text{tel que} \quad ct < x\}$$

■ D'autre part, pour tout $t' > 0$, on a :

$$\frac{d}{ds}u(cs, t'+s) = \left(\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x}\right)(cs, t'+s) = 0, \quad 0 < cs < L, \quad 0 < t'+s < T$$

C'est à dire

$$u(cs, t'+s) = \text{Cste}, \quad 0 < cs < L, \quad 0 < t'+s < T$$

On choisi la constante correspondant à $s = 0$:

$$u(cs, t'+s) = u(0, t'), \quad 0 < cs < L, \quad s < t'+s < T$$

On effectue le changement de variable

$$\begin{cases} x = cs \\ t = t'+s \end{cases} \Longrightarrow t' = t - \frac{x}{c}$$

Ceci permet d'écrire

$$u(x,t) = u(0, t - \frac{x}{c}) = e^{-t+\frac{x}{c}}$$

Sachant que $t' > 0$ (car $t'+s > s$), la solution donnée est définie sur :

$$\left\{(x,t) \in]0, L[\times]0, T[\quad \text{tel que} \quad \frac{x}{c} < t\right\}$$

On récapitule en écrivant la solution de l'équation de transport

$$\text{Pour } x \in]0, L[\text{ et } t \in]0, T[, \quad u(x,t) = \begin{cases} 0 & \text{si} \quad ct < x \\ e^{-t+\frac{x}{c}} & \text{si} \quad ct > x \end{cases}$$

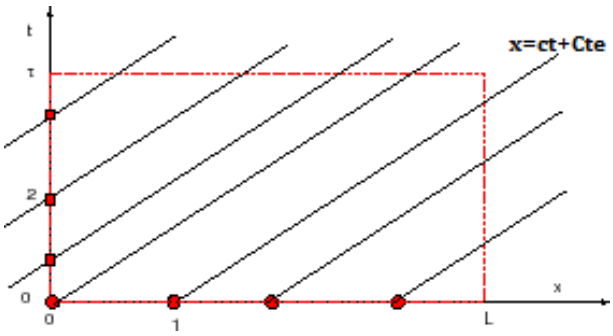


FIGURE 1 – *Illustration de la solution de l'équation de Transport dans l'espace-temps à l'aide des courbes caractéristiques.*

Question 2.

Montrer que le problème n'admet pas de solution lorsque $c < 0$.

Réponse

Procédons par l'absurde et supposons que le problème admette une solution $u(\cdot, \cdot)$.
Soit $x \in]0, L[$ et $t \in]0, T[$. Pour tout $s \in \mathbb{R}$, on a :

$$\frac{d}{ds}u(x+cs, t+s) = \left(\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x}\right)(x+cs, t+s) = 0$$

Avec s t.q. $\begin{cases} 0 < x + cs < L \\ 0 < t + s < T \end{cases} \Rightarrow \begin{cases} -L < cs < L \\ -T < s < T \end{cases} \Rightarrow s \in K =]-\min(T, -\frac{L}{c}), \min(T, -\frac{L}{c})[$

On obtient donc :

$$\forall s \in K \quad u(x + cs, t + s) = \text{Cste} \tag{1}$$

En particulier, pour $s = 0 \in K$, on a $u(x + cs, t + s) = u(x, t)$

- Supposons $-\frac{L}{c} < T$ i.e. $K =]\frac{L}{c}, -\frac{L}{c}[$; alors en prenant $s = -\frac{x}{c} \in K$, on a d'après (1) :

  $$u(x + cs, t + s) = u(x, t) = u(0, t - \frac{x}{c})$$

  On constate déjà que la condition sur la borne gauche du domaine permet de définir entièrement l'évolution de u (sans tenir compte de la condition initiale). On peut relier les deux conditions aux limites en posant $x = -ct'$ et $t = 0$:

  — D'une part $\quad u(x, t) = u(0, t - \frac{x}{c}) = u(0, t') = e^{-t'}$

  — D'autre part $\quad u(x, t) = u(-ct', 0) = 0$

  Ceci pour tout $t' \in]0, -\frac{L}{c}[$. Ce qui est absurde car $\quad e^{-t'} \neq 0 \quad$ pour tout $\quad t' \in \mathbb{R}$.

- Supposons au contraire $-\frac{L}{c} \geq T$ i.e. $K =]-T, T[$; on prends alors $s = -t \in K$, ce qui donne (d'après l'équation (1)) :

  $$u(x + cs, t + s) = u(x, t) = u(x - ct, 0)$$

  Posons $x = 0$ et $t = -\frac{x'}{c}$:

  — D'une part $\quad u(x, t) = u(x - ct, 0) = u(x', 0) = 0$

  — D'autre part $\quad u(x, t) = u(0, -\frac{x'}{c}) = e^{\frac{x'}{c}}$

  Ceci pour tout $x' \in]0, -cT[$. Ce qui est absurde !

Dans les deux cas, on constate que le problème n'admet pas de solution. D'un point de vue physique, ce résultat était attendu. Ceci car pour $c < 0$, le transport de la quantité u se fait de la droite vers la gauche. Au lieu d'imposer la condition sur le bord gauche ($x = 0$), il aurait donc fallu fournir une condition sur le bord droit du domaine ($x = L$) pour que l'équation de transport admette une solution.

Question 3.

Écrire un programme qui résout l'équation de transport par la méthode de Godunov.

Réponse

Pour programmer la méthode de Godunov, il nous faut introduire certaines fonctions. Pour l'équation de transport, on les définit de la manière suivante :

— Le flux physique :

```
void fluxphy(double* w, double* flux) { flux[0] = _C * w[0]; }
```

— La vitesse maximale sur chaque cellule :

```
double lambda_max(double* u) { return _C; }
```

— La solution du problème de Riemann associé :

```
void riemann(double* a, double* b, double z, double* w) {
   if (z < _C) {        // z = x/t
      w[0] = a[0];
   } else {
      w[0] = b[0];
   }
}
```

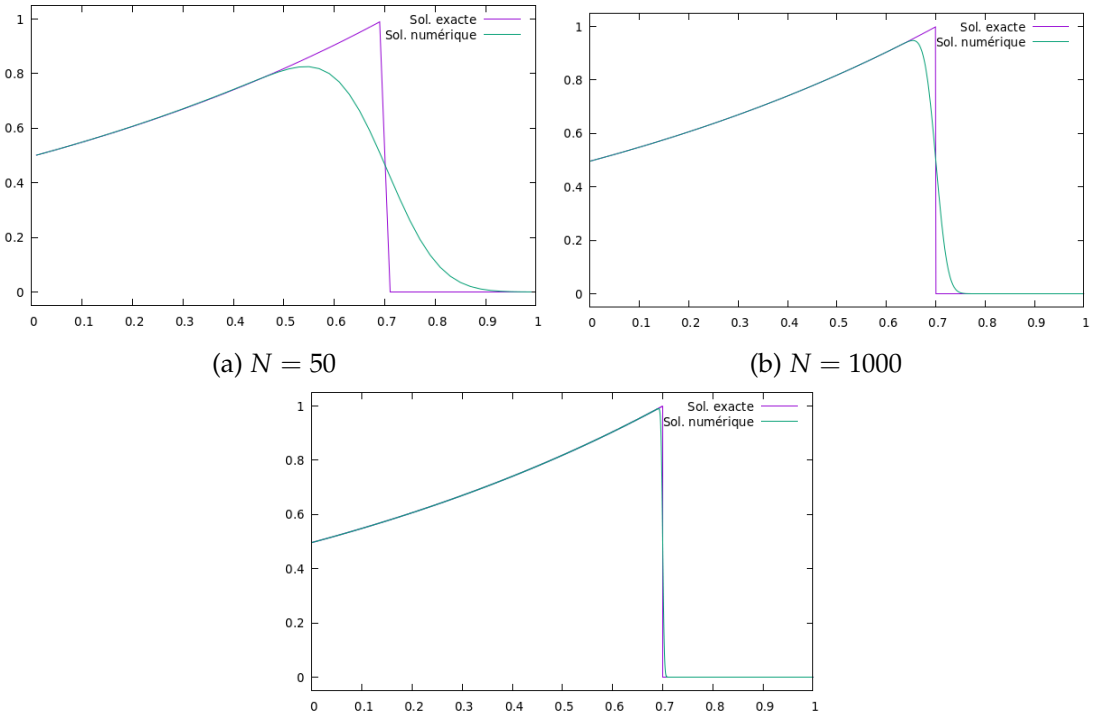
— Une solution exacte du problème : utilisé comme condition initiale ($t = 0$), et comme condition aux bords ($x = 0$ et $x = L$) :

```
void solexacte(double x, double t, double* w) {
   double uR = 0;
   double uL = exp(-(t - x / _C));

   if (x < _C * t) {
      w[0] = uL;
   } else {
      w[0] = uR;
}
```

— La fonction de résolution du problème, utilisable tel quel pour d'autres problèmes

```
void godunov_solve(godunov* gd, double tmax) {
   double tnow = 0;
   int m = gd->m;
   while (tnow < tmax) {
      double vmax = 0;
      // Calcul de la vitesse max
      for (int i = 0; i < gd->N + 2; i++) {
         double vloc = lambda_max(gd->un + m * i);
         vmax = vmax > vloc ? vmax : vloc;
      }
      // Calcul du pas de temps
      gd->dt = gd->cfl * gd->dx / vmax;
      // Application du flux
      for (int i = 1; i < gd->N + 1; i++) {
         double flux[m];
         fluxnum(gd->un + i * m, gd->un + (i + 1) * m, flux);
         for (int iv = 0; iv < m; iv++) {
            gd->unp1[i * m + iv] =
               gd->un[i * m + iv] - gd->dt / gd->dx * flux[iv];
         }
         fluxnum(gd->un + (i - 1) * m, gd->un + i * m, flux);
         for (int iv = 0; iv < m; iv++) {
            gd->unp1[i * m + iv] += gd->dt / gd->dx * flux[iv];
         }
      }
      // Mise à jour
      tnow += gd->dt;
      // Conditions aux limites
      int i = 0;
      solexacte(gd->xi[i], tnow, gd->unp1 + i * m);
      i = gd->N + 1;
      solexacte(gd->xi[i], tnow, gd->unp1 + i * m);
      // Preparation pour l'iteration suivante
      memcpy(gd->un, gd->unp1, (gd->N + 2) * m * sizeof(double));
   }
   gd->tfin = tnow;
}
```

On choisit de résoudre le problème avec $L = 2$, $T = 0.7$, $c = 1$ et cfl[1]. Pour différents taille de maillage ($N = 50, 1000, 50000$), on obtient les résultats suivants :



(a) $N = 50$ (b) $N = 1000$

(c) $N = 50000$

FIGURE 2 – *Quelque résultats obtenus pour l'équation de transport au temps final* $T = 0.7$.

Question 4.

Étude du taux de convergence. Vérifions la validité de notre programme sur l'équation de transport en traçant le logarithme de l'erreur en norme L^1 entre la solution exacte à l'instant T et la solution numérique en fonction de $\ln(\Delta x)$ où Δx est le pas de la subdivision.

Réponse

On obtient le tableau de convergence illustré ci-bas (tableau 1) ; son illustration est donnée à la figure 3.

1. Les détails sur le coefficient cfl sont donnés à la question 5.

N	Δx	Erreur L^1
10	0.100000	0.140029
40	0.025000	0.074383
160	0.006250	0.037323
640	0.001563	0.018697
2560	0.000391	0.009342
10240	0.000098	0.004669

TABLE 1 – *Étude de convergence pour l'équation de transport*

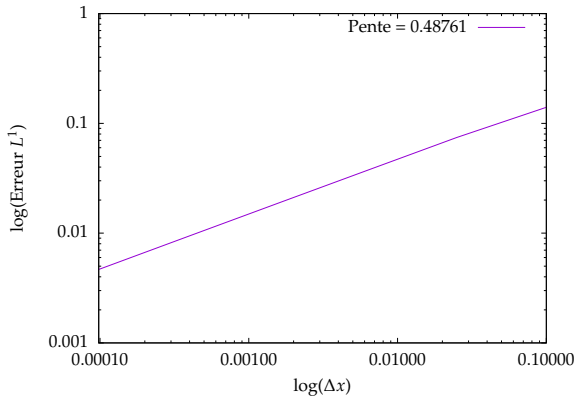


FIGURE 3 – *Illustration de l'étude de convergence pour l'équation de transport*

Le taux (ou l'ordre) de convergence estimé est donné soit par la pente de la courbe de la figure 3, soit par la formule

$$\text{ordre} \approx \frac{\log\left(\frac{\text{Err}(N)}{\text{Err}(2N)}\right)}{\log(2)}.$$

Un calcul avec $N = 5000$ nous donne un ordre de convergence proche de **0.50**. Ceci indique que le schéma de Godunov converge avec une vitesse de l'ordre de $\sqrt{N}$ [2].

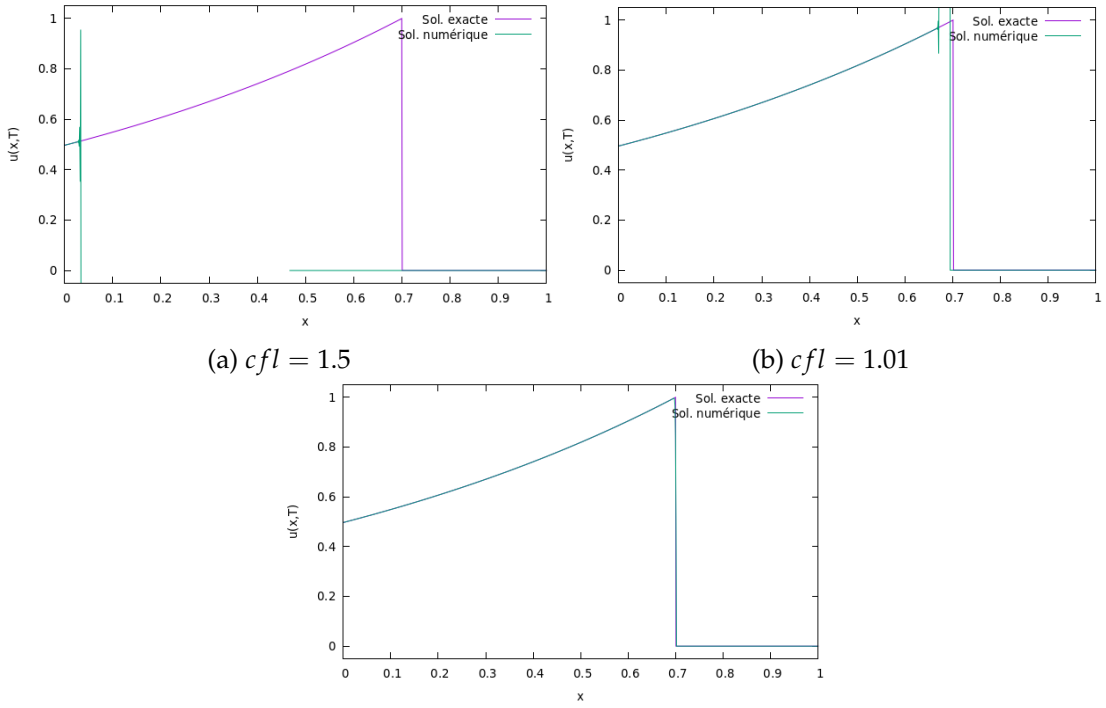
Question 5.

Vérifions numériquement que le schéma devient instable lorsque la condition de CFL n'est pas satisfaite. Traçons un exemple lorsque $c\Delta t/\Delta x > 1$ et proche de 1.

Réponse

Comme précédemment, on prend $L = 2$, $T = 0.7$, $c = 1$ et $N = 1000$. Le coefficient cfl tel que $\Delta t = cfl \times \dfrac{\Delta x}{c}$ est pris égal 1.5, 1.01 et 0.999.

2. Cet ordre de convergence est indiqué pour les conditions aux limites et initiales imposées par le problème.



(a) $cfl = 1.5$ (b) $cfl = 1.01$

(c) $cfl = 0.999$

FIGURE 4 – *Vérification numérique de l'instabilité de l'équation de transport*

On vérifie ainsi que le schéma est instable lorsque la condition CFL n'est pas satisfaite (figures 4a et 4b). Par contre, lorsque la condition est vérifiée et le coefficient cfl est très proche de 1, la solution obtenue est considérablement meilleure qu'avec un coefficient cfl plus bas, pour la même taille de maillage ($N = 1000$). On peut observer ceci en comparant les figures 4c (où $cfl = 0.999$) et 2b (où $cfl = 0.5$).

2. Résolution numérique de l'équation de Burgers

Question 1.

Calculer la solution de Lax du problème suivant.

$$\begin{aligned} \partial_t u + \partial_x(u^2/2) &= 0, \qquad x \in]-1,2[, \quad t > 0 \\ u(0,t) &= 1 \\ u(x,0) &= \begin{cases} 1 & \text{si } x < 0 \\ 1-x & \text{si } 0 \le x \le 1 \\ 0 & \text{si } x > 1 \end{cases} \end{aligned}$$

Réponse

Ce problème se réécrit sous la forme $\partial_t u + \partial_x f(u) = 0$, avec $f : u \mapsto \frac{u^2}{2}$.

Considérons la courbe caractéristique $t \mapsto \binom{x(t)}{t}$ sur laquelle la solution u est constante i.e. $u(x,t) = u(x(t),t) = \text{Cste}$.
Ceci donne

$$\frac{d}{dt}u(x(t),t) = \left(\frac{\partial u}{\partial t} + x'(t)\frac{\partial u}{\partial x}\right)(x(t),t) = 0$$

Par comparaison avec l'équation de Burgers qui s'écrit encore sous la forme

$$\left(\frac{\partial u}{\partial t} + u(x(t),t)\frac{\partial u}{\partial x}\right)(x(t),t) = 0$$

on déduit que suivant les courbes caractéristiques, on a $x'(t) = u(x(t),t) = \text{Cste}$. On peut donc écrire

$$x'(t) = u(x(0),0)$$

Distinguons à présent les différents cas possibles d'abscisses à l'origine de la caractéristique :

— Si $x(0) < 0$, alors

$$x'(t) = u(x(0),0) = 1 \implies x(t) = t + x(0) \implies x(0) = x(t) - t$$

Il s'en suit que pour $x - t < 0$, on a

$$\begin{aligned} u(x,t) &= u(x(t),t) = \text{Cste} \\ &= u(x(0),0) \\ &= 1 \end{aligned}$$

— Si $0 \leq x(0) \leq 1$, alors

$$x'(t) = u(x(0),0) = 1 - x(0) \implies x(t) = (1 - x(0))t + x(0) \implies x(0) = \frac{x(t) - t}{1 - t}$$

Il s'en suit que pour $0 \leq \dfrac{x-t}{1-t} \leq 1$, on a

$$\begin{aligned} u(x,t) &= u(x(t),t) \\ &= u(x(0),0) \\ &= 1 - \frac{x-t}{1-t} \\ &= \frac{1-x}{1-t} \end{aligned}$$

— Si $x(0) > 1$, alors

$$x'(t) = u(x(0),0) = 0 \implies x(t) = x(0)$$

Il s'en suit que pour $x > 1$, on a

$$\begin{aligned} u(x,t) &= u(x(t),t) \\ &= u(x(0),0) \\ &= 0 \end{aligned}$$

On obtient donc la solution définie sur $]-1,2[\times]0,1[$ par :

$$u(x,t) = \begin{cases} 1 & \text{si } x < t \\ \dfrac{1-x}{1-t} & \text{si } 0 \leq \dfrac{x-t}{1-t} \leq 1 \\ 0 & \text{si } x > 1 \end{cases}$$

La fonction trouvée est bien définie et continue lorsque $t < 1$. Cependant, lorsque $t \geq 1$, les caractéristiques correspondants aux trois cas se croisent. Notre solution sera alors une solution faible si la pente σ de la courbe de discontinuité vérifie la relation de Rankine-Hugoniot (voir figure 5) :

$$f(u_L) - f(u_R) = \sigma(u_L - u_R) \Longrightarrow \sigma = \frac{u_L + u_R}{2} = \frac{1+0}{2} = \frac{1}{2}$$

La vitesse de choc σ vérifie aussi le critère de Lax car

$$f'(u_L) = 1 > \sigma = \frac{1}{2} > 0 = f'(u_R).$$

En conclusion, la solution au sens de Lax est donnée par :

- Lorsque $t < 1$, alors :

$$u(x,t) = \begin{cases} 1 & \text{si } x < t \\ \dfrac{1-x}{1-t} & \text{si } 0 \leq \dfrac{x-t}{1-t} \leq 1 \\ 0 & \text{si } x > 1 \end{cases}$$

- Lorsque $t \geq 1$, alors :

$$u(x,t) = \begin{cases} 1 & \text{si } x < \dfrac{1}{2}t \\ 0 & \text{si } x > \dfrac{1}{2}t \end{cases}$$

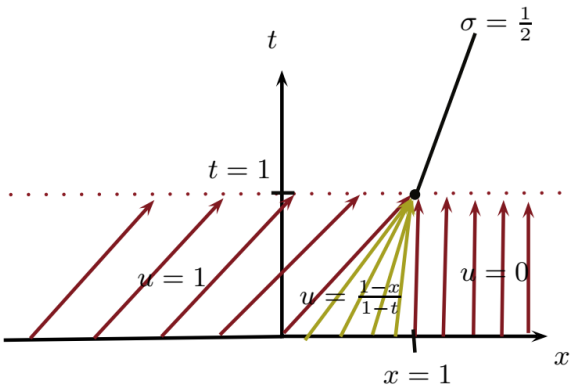


FIGURE 5 – *Illustration de la solution de Lax du problème de Burgers dans l'espace-temps à l'aide des courbes caractéristiques.*

Question 2.

Résoudre le problème de Riemann pour l'équation de Burgers.

$$\partial_t u + \partial_x(u^2/2) = 0 \tag{2}$$

$$u(x,0) = \begin{cases} u_L & \text{si } x < 0 \\ u_R & \text{si } x > 0 \end{cases}$$

Réponse

Comme précédemment, les courbes caractéristiques sont les telle que $x'(t) = u(x(0), 0)$. On traite tout d'abord les deux cas suivant de façon indépendante :

— Si $x(0) < 0$ alors

$$x'(t) = u_L \Rightarrow x(t) = u_L t + x(0)$$

On obtient donc

$$u(x,t) = u_L \text{ pour } (x,t) \in D_L, \text{ avec } D_L = \left\{(x,t) \in]-1,2[\times\mathbb{R}_+^* \text{ t.q. } \frac{x}{t} < u_L\right\}$$

— Si $x(0) > 0$ alors

$$x'(t) = u_R \Rightarrow x(t) = u_R t + x(0)$$

On obtient donc

$$u(x,t) = u_R \text{ pour } (x,t) \in D_R, \text{ avec } D_R = \left\{(x,t) \in]-1,2[\times\mathbb{R}_+^* \text{ t.q. } \frac{x}{t} > u_R\right\}$$

Distinguons maintenant deux cas :

■ Si $u_L > u_R$ alors les domaines D_L et D_R sont sécants dans l'espace temps ; La solution présente donc une discontinuité dont la pente σ est donnée par la relation de Rankine-Hugoniot.

$$f(u_L) - f(u_R) = \sigma(u_L - u_R) \Longrightarrow \sigma = \frac{u_L + u_R}{2}$$

On a donc obtenue une solution faible

$$u(x,t) = \begin{cases} u_L & \text{si } x < \sigma t \\ u_R & \text{si } x > \sigma t \end{cases}$$

Cette solution vérifie bien le crière de Lax car $f'(u_L) = u_L > \sigma = \frac{u_L+u_R}{2} > u_R = f'(u_R)$.

■ Si au contraire $u_L < u_R$, les domaines D_L et D_R sont disjoints dans l'espace temps ; la solution faible donnée dans le cas $u_L > u_R$ reste applicable, mais ne vérifie plus le critère de Lax.
On construit donc une solution acceptable en exprimant u sur

$$D_{milieu} = D_C = \left\{(x,t) \in]-1,2[\times\mathbb{R}_+^* \text{ t.q. } u_L < \frac{x}{t} < u_R\right\}.$$

On remarque d'entrée que si $u(x,t)$ est solution de l'équation (2), alors $u(\lambda x, \lambda t)$ est aussi solution (pour $\lambda \in \mathbb{R}$). On est donc amené à croire que la solution ne dépend que du rapport $\xi = x/t$ (i.e ξ est autosimilaire). Posons $u(x,t) = v(\xi)$. On remplace cela dans l'équation (2) pour trouver :

$$\left(\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x}\right)(x,t) = \left(\frac{\partial v}{\partial t} + \frac{\partial f(v)}{\partial x}\right)(\xi) = 0$$

On développe le terme du milieu pour obtenir :

$$\begin{aligned}
\left(\frac{\partial v}{\partial t} + \frac{\partial f(v)}{\partial x}\right)(\xi) &= \frac{\partial v}{\partial t}(\xi) + f'(v(\xi))\frac{\partial v}{\partial x}(\xi) \\
&= \frac{\partial v}{\partial \xi}(\xi)\frac{\partial \xi}{\partial t}(t) + v(\xi)\frac{\partial v}{\partial \xi}(\xi)\frac{\partial \xi}{\partial x}(x) \\
&= \left(-\frac{x}{t^2} + \frac{v(\xi)}{t}\right)v'(\xi) \\
&= 0
\end{aligned}$$

On remarque que lorsque $v'(\xi) \neq 0$, l'expression de u sur D_C peut être obtenue par [3]

$$-\frac{x}{t^2} + \frac{v(\xi)}{t} \Longrightarrow v(\xi) = u(x,t) = \frac{x}{t}$$

On est donc amené à considérer la fonction suivante comme candidat pour le problème de Riemann (2).

$$u(x,t) = \begin{cases} u_L & \text{sur } D_L \\ \dfrac{x}{t} & \text{sur } D_C \\ u_R & \text{sur } D_R \end{cases}$$

Par construction, u est une solution forte de (2) sur $D_L \cup D_C \cup D_R$. En plus, elle est continue au niveau des zones de transition $\frac{x}{t} = u_L$ et $\frac{x}{t} = u_R$. Elle vérifie donc le critère de Lax.

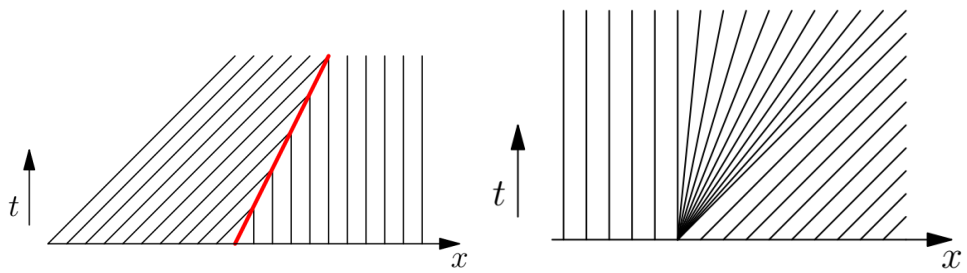
En conclusion, la solution du problème de Riemann pour le problème de Burgers est donnée par (voir figure 6) :

- Si $u_L < u_R$, alors la solution est l'onde de choc

$$u(x,t) = \begin{cases} u_L & \text{si } \dfrac{x}{t} < \dfrac{u_L + u_R}{2} \\ u_R & \text{si } \dfrac{x}{t} \geq \dfrac{u_L + u_R}{2} \end{cases}$$

- Si $u_L \leq u_R$, alors la solution est l'onde dite de raréfaction

$$u(x,t) = \begin{cases} u_L & \text{si } \dfrac{x}{t} < u_L \\ u_R & \text{si } \dfrac{x}{t} > u_R \\ \dfrac{x}{t} & \text{sinon} \end{cases}$$



(a) $u_L > u_R$ (b) $u_L \leq u_R$

FIGURE 6 – *Illustration de la solution du problème de Riemann à l'aide des courbes caractéristiques*

Question 3.

— Écrire un programme qui résout le problème (2) par la méthode de Godunov.
— Comparer la solution exacte et la solution approchée aux instants $t = 1/2; t = 1; t = 2$ pour diverses finesses de maillage Δx.
— Effectuer une étude de taux de convergence.

3. En réalité, $v'(\xi)$ doit être non nul pour avoir des courbes caractéristiques non parallèles et ainsi accomplir la transition de u_L à u_R

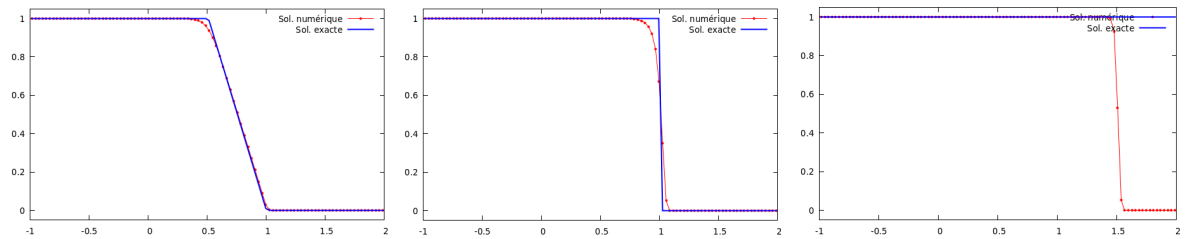
Réponse

Le code de calcul pour la phase de résolution de l'équation de Burgers par la méthode de Godunov est rigoureusement identique à celui utilisé pour l'équation de transport. Cependant, quelques fonctions doivent êtres adaptées ; elles sont toutes données ci-bas.

```
void fluxphy(double* w, double* flux) { flux[0] = w[0] * w[0] / 2; }

double lambda_max(double* u) { return fabs(u[0]); }

void riemann(double* a, double* b, double z, double* w) {
   if (a[0] > b[0]) {
      // choc
      double s = (a[0] + b[0]) / 2;
      if (z < s)
         w[0] = a[0];
      else
         w[0] = b[0];
   } else {
      // onde de détente
      if (z < a[0]){
         w[0] = a[0];
      } else if (z > b[0]){
         w[0] = b[0];
      } else
         w[0] = z;
   }
}

void solexacte(double x, double t, double* w) {
   if (x < t){
      w[0] = 1;
   } else if(x > 1) {
      w[0] = 0;
   } else { // if ( 0 <= (x-t)/(1-t) && (x-t)/(1-t) <=1.)
      w[0] = (1-x)/(1-t);
   }
}
```

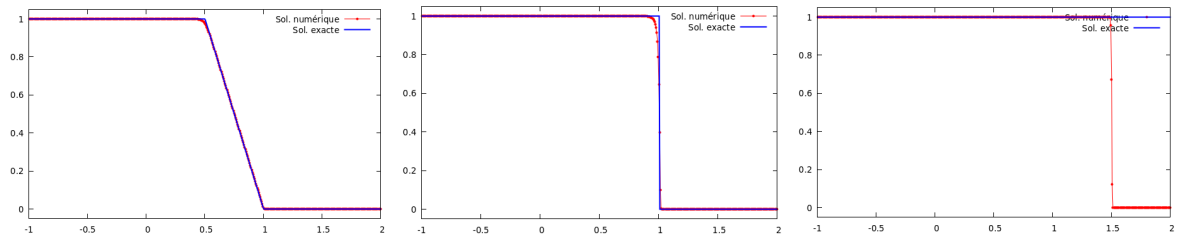
Listing 1 – *Flux physique, vitesse maximale, solution du problème de Riemann, et solution exacte pour l'équation de Burgers*

Comparons à présent la solution exacte et la solution approchée aux temps indiqués, pour trois finesses de maillages. Les figures 7 à 9 montrent clairement que les solutions sont meilleures pour des maillages raffinés. Cependant, nous avons vu à la question 1) qu'une solution forte pour ce problème n'existait que pour $t \leq 1$. Ceci explique pourquoi le schéma numérique de Godunov ne retrouve pas la solution exacte (faible) que nous avons exprimé à la question précédente ; ceci lorsque $t = 2$ (figures 7c, 8c et 9c).



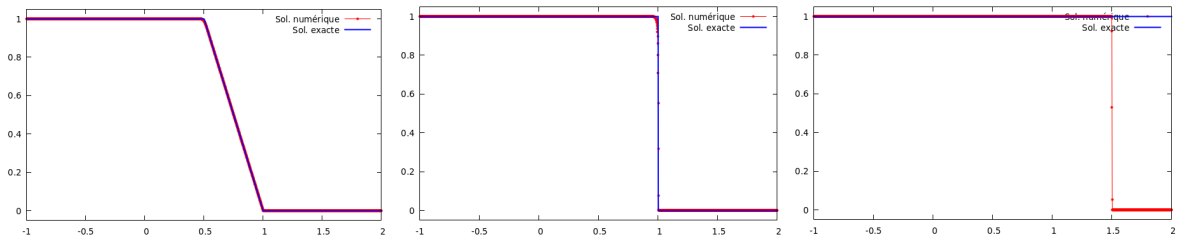
(a) $t = 0.5$ (b) $t = 1$ (c) $t = 2$

FIGURE 7 – *Résolution de l'équation de Burgers par la méthode de Godunov, pour un maillage de taille* $N = 100$ *i.e* $\Delta x = 3e - 2$.



(a) $t = 0.5$ (b) $t = 1$ (c) $t = 2$

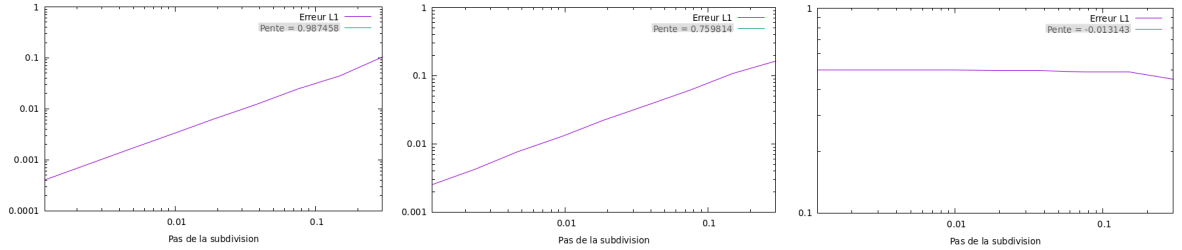
FIGURE 8 – *Résolution de l'équation de Burgers par la méthode de Godunov, pour un maillage de taille* $N = 500$ *i.e* $\Delta x = 6e - 3$.



(a) $t = 0.5$ (b) $t = 1$ (c) $t = 2$

FIGURE 9 – *Résolution de l'équation de Burgers par la méthode de Godunov, pour un maillage de taille* $N = 2500$ *i.e* $\Delta x = 1.2e - 3$.

Effectuons à présent une étude du taux de convergence, pour les même temps de simulation que précédemment. On obtient la figure 10 qui montre que la meilleure convergence est obtenue pour $t = 0.5$, avec un ordre de convergence d'environ 0.99. Ensuite vient le cas $t = 1$, avec un ordre de convergence proche de 0.76. Enfin, le cas $t = 2$ ne saurait converger dû à la non-existence d'une solution forte.



(a) $t = 0.5$ (b) $t = 1$ (c) $t = 2$

FIGURE 10 – *Étude du taux de convergence pour l'équation de Burgers.*

Question 4.

Décrire et programmer la correction MUSCL de van Leer. Vérifier que la précision du schéma est amélioré grâce à une étude de taux de convergence.

Réponse

Rappelons que le schéma de volumes finis en 1D s'écrit pour une maille i à l'itération n comme suit

$$\frac{w_i^{n+1} - w_i^n}{\Delta t} + \frac{F_{i+\frac{1}{2}}^n - F_{i-\frac{1}{2}}^n}{\Delta x} = 0 \tag{3}$$

où les flux numérique aux interfaces sont donnés par

$$F^n_{i+\frac{1}{2}} = f(w^n_i, w^n_{i+1})$$
$$F^n_{i-\frac{1}{2}} = f(w^n_{i-1}, w^n_i)$$

L'idée de la méthode MUSCL [4] est de remplacer le flux numérique calculé à l'interface $i+\frac{1}{2}$ par un flux numérique calculé en des points plus proches de l'interface, et à un pas de temps correspondant. On notera ces points par $w^{n+\frac{1}{2}}_{i+\frac{1}{2},-}$ et $w^{n+\frac{1}{2}}_{i+\frac{1}{2},+}$ [5]. On pose alors

$$\begin{aligned} w^{n+\frac{1}{2}}_{i+\frac{1}{2},-} &= w^n_i + s^n_i \frac{\Delta x}{2} + r^n_i \frac{\Delta t}{2} \\ w^{n+\frac{1}{2}}_{i+\frac{1}{2},+} &= w^n_{i+1} - s^n_{i+1} \frac{\Delta x}{2} + r^n_{i+1} \frac{\Delta t}{2} \end{aligned} \tag{4}$$

Pour le calcul des pentes en espace s^n_i pour une maille quelconque i, on pose :

$$\alpha = \frac{w^n_i - w^n_{i-1}}{\Delta x}, \quad \beta = \frac{w^n_{i+1} - w^n_i}{\Delta x}, \quad \gamma = \frac{w^n_{i+1} - w^n_{i-1}}{2\Delta x}$$

et on en déduit

$$s^n_i = \text{minmod}(\alpha, \beta, \gamma)$$

où la fonction minmod est donnée par :

$$\text{minmod}(\alpha, \beta, \gamma) = \begin{cases} \min(\alpha, \beta, \gamma) & \text{si } \alpha, \beta, \gamma > 0 \\ \max(\alpha, \beta, \gamma) & \text{si } \alpha, \beta, \gamma < 0 \\ 0 & \text{sinon} \end{cases}$$

Pour le calcul des pentes en temps, on se sert de la loi de conservation décrivant l'équation de Burgers

$$\partial_t w + \partial_x f(w) = 0$$

On prend donc la pente r^n_i telle que

$$r^n_i = -f'(w^n_i) s^n_i$$

Les pentes s^n_i et r^n_i ayant été calculées, on calcule le flux numérique aux points $w^{n+\frac{1}{2}}_{i+\frac{1}{2},-}$ et $w^{n+\frac{1}{2}}_{i+\frac{1}{2},+}$ grâce à l'équation (4), qu'on applique au schéma numérique. L'équation (3) devient alors

$$\frac{w^{n+1}_i - w^n_i}{\Delta t} + \frac{F^{n+\frac{1}{2}}_{i+\frac{1}{2}} - F^{n+\frac{1}{2}}_{i-\frac{1}{2}}}{\Delta x} = 0 \tag{5}$$

où

$$F^{n+\frac{1}{2}}_{i+\frac{1}{2}} = f(w^{n+\frac{1}{2}}_{i+\frac{1}{2},-}, w^{n+\frac{1}{2}}_{i+\frac{1}{2},+})$$
$$F^{n+\frac{1}{2}}_{i-\frac{1}{2}} = f(w^{n+\frac{1}{2}}_{i-\frac{1}{2},-}, w^{n+\frac{1}{2}}_{i-\frac{1}{2},+})$$

En ce qui concerne l'implémentation, le code de calcul précédemment présenté [6] doit être ajusté. En effet, nous devons rajouter les pentes lors de la résolution du système. On obtient le code de calcul pour la fonction `godunov_solve` ci-bas.

4. L'acronyme MUSCL est mis pour "Monotonic Upstream-centered Scheme for Conservation Laws". Nous décrivons ici la version de cette correction qui prend en compte à la fois les pentes en espace et en temps.

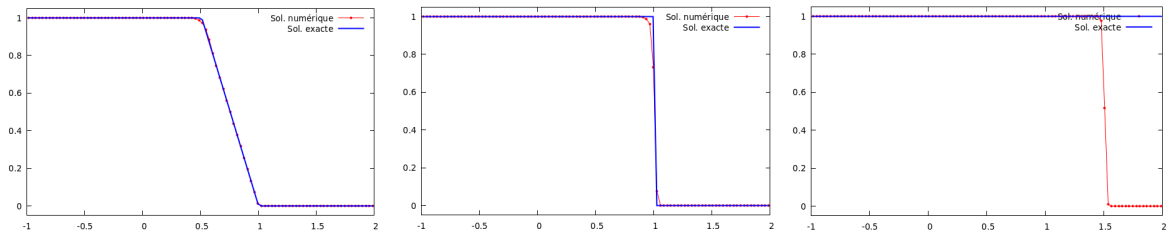
5. Notons qu'un raisonnement similaire est appliqué à l'interface $i-\frac{1}{2}$ pour calculer $w^{n+\frac{1}{2}}_{i-\frac{1}{2},-}$ et $w^{n+\frac{1}{2}}_{i-\frac{1}{2},+}$.

6. La fonction de résolution par la méthode de Godunov simple est la même pour l'équation de transport, et pour l'équation de Burgers.

```c
void godunov_solve(godunov* gd, double tmax){
   double tnow = 0;
   int m = gd->m;
   while (tnow < tmax) {
      double vmax = 0;
      // calcul de la vitesse max
      for (int i = 0; i < gd->N + 2; i++) {
         double vloc = lambda_max(gd->un + m * i);
         vmax = vmax > vloc ? vmax : vloc;
      }
      // Calcul des pentes pour MUSCL
      double si[gd->N+1], ri[gd->N+1];
      for (int i = 1; i < gd->N + 1; i++) {
         double alpha = (gd->un[i] - gd->un[i-1])/gd->dx;
         double beta = (gd->un[i+1] - gd->un[i])/gd->dx;
         double gamma = (gd->un[i+1] - gd->un[i-1])/(2.0*gd->dx);
         si[i] = minmod(alpha, beta, gamma);
         ri[i] = - gd->un[i] * si[i];
      }
      gd->dt = gd->cfl * gd->dx / vmax;
      for (int i = 1; i < gd->N + 1; i++) {
         double flux[m];
         // Application de MUSCL avec la droite
         double uL[1] = {gd->un[i] + si[i]*gd->dx/2.0 + ri[i]*gd->dt/2.0};
         double uR[1] = {gd->un[i+1] - si[i+1]*gd->dx/2.0 + ri[i+1]*gd->dt/2.0};
         fluxnum(uL, uR, flux);
         for (int iv = 0; iv < m; iv++) {
            gd->unp1[i * m + iv] =
               gd->un[i * m + iv] - gd->dt / gd->dx * flux[iv];
         }
         // Application de MUSCL avec la gauche
         uL[0] = gd->un[i-1] + si[i-1]*gd->dx/2.0 + ri[i-1]*gd->dt/2.0;
         uR[0] = gd->un[i] - si[i]*gd->dx/2.0 + ri[i]*gd->dt/2.0;
         fluxnum(uL, uR, flux);
         for (int iv = 0; iv < m; iv++) {
            gd->unp1[i * m + iv] += gd->dt / gd->dx * flux[iv];
         }
      }
      // mise à jour
      tnow += gd->dt;
      // conditions aux limites
      int i = 0;
      solexacte(gd->xi[i], tnow, gd->unp1 + i * m);
      i = gd->N + 1;
      solexacte(gd->xi[i], tnow, gd->unp1 + i * m);
      // Pour l'iteration suivante
      memcpy(gd->un, gd->unp1, (gd->N + 2) * m * sizeof(double));
   }
   gd->tfin = tnow;
}
```

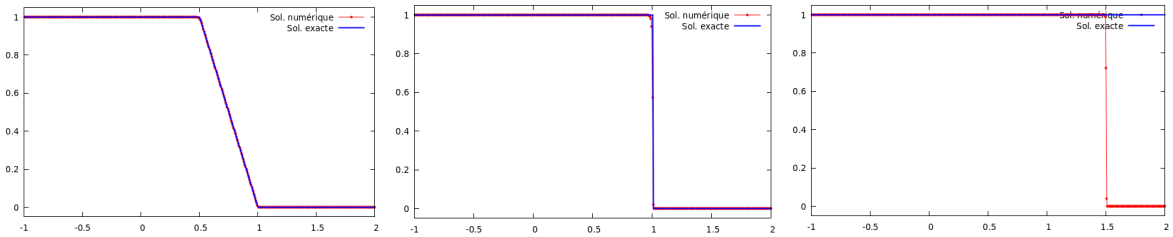
Listing 2 – *Fonction de résolution du problème de Burger par un schéma de Godunov, auquel on applique la correction MUSCL*

Pour tester notre programme, nous reprenons l'étude faite à la question précédente. Les résultats sont présentés aux figures 11 à 13. En les comparant aux figures 7 à 9, on remarque immédiatement une meilleure approximation numérique, particulièrement perceptible pour un maillage grossier $N = 100$.



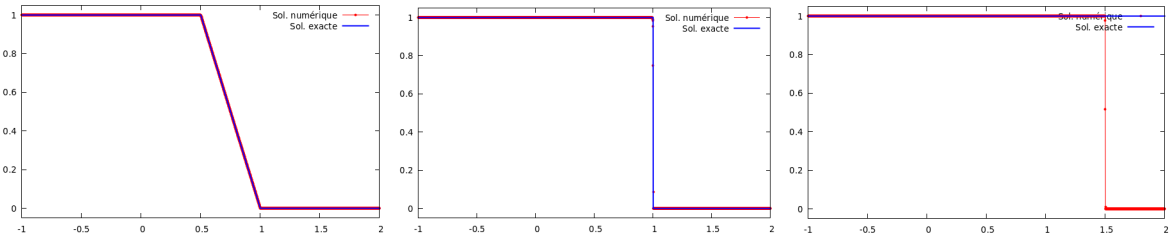
(a) $t = 0.5$ (b) $t = 1$ (c) $t = 2$

FIGURE 11 – *Application de la correction de MUSCL au schéma de Godunov, pour un maillage de taille* $N = 100$ *i.e* $\Delta x = 3e - 2$.



(a) $t = 0.5$ (b) $t = 1$ (c) $t = 2$

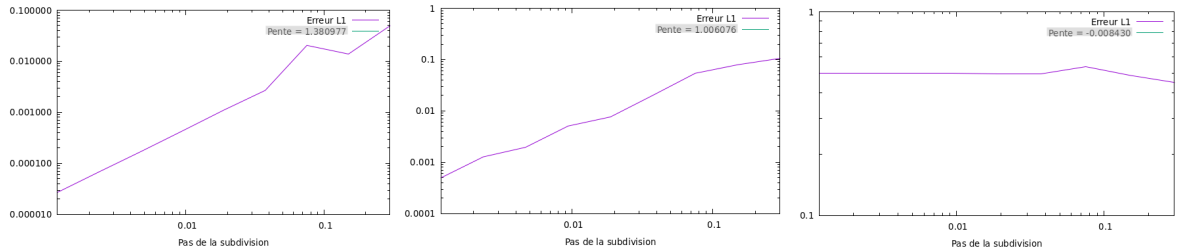
FIGURE 12 – *Application de la correction de MUSCL au schéma de Godunov, pour un maillage de taille* $N = 500$ *i.e* $\Delta x = 6e - 3$.



(a) $t = 0.5$ (b) $t = 1$ (c) $t = 2$

FIGURE 13 – *Application de la correction de MUSCL au schéma de Godunov, pour un maillage de taille* $N = 2500$ *i.e* $\Delta x = 1.2e - 3$.

L'étude du taux de convergence permet de constater la précision supérieure de la correction MUSCL de van Leer. Hormis le cas $t = 2$ qui est aberrant, on a, comparé à figure 10, des meilleures pentes. Ces pentes sont observables sur la figure figure 14 et un récapitulatif présenté au tableau 2. Il est intéressant de constater qu'en général, le problème de Burgers, bien qu'il soit non-linéaire, converge plus rapidement que le problème de transport (cf. tableau 1).



(a) $t = 0.5$ (b) $t = 1$ (c) $t = 2$

FIGURE 14 – *Étude du taux de convergence de la méthode de Godunov avec la correction MUSCl.*

Temps	**Godunov Simple**	**Correction MUSCL**
$t = 0.5$	0.9875	1.3810
$t = 1$	0.7598	1.0060
$t = 2$	-0.0131	-0.0084

TABLE 2 – *Observation de l'amélioration de l'ordre de convergence avec la correction MUSCL. Le cas* $t = 2$ *est aberrant et n'intervient pas dans nos interprétations.*

UNIVERSITÉ DE STRASBOURG
UFR DE MATHÉMATIQUES ET INFORMATIQUE

TP #2

Etudiant : *Roussel Desmond Nzoyem*

UE : *EDP 2* – Enseignant : *Pr. Philippe Helluy*
Date : *10 janvier 2021*

Résolution numérique de l'équation de Saint-Venant

Question 1.

Calcul numérique au moyen du solveur de Riemann exact fourni, de la solution du problème de Riemann pour le modèle de Saint-Venant.

$$\begin{gathered} \partial_t w + \partial_x f(w) = 0 \\ w = (h, hu) \\ f(w) = (hu, hu^2 + gh^2/2) \quad , \quad g = 9.81 \text{ m/s}^2 \\ u(x,0) = 0 \\ h(x,0) = \begin{cases} h_L = 2 & \text{si } x < 0 \\ h_R = 1 & \text{si } x > 0 \end{cases} \end{gathered} \tag{1}$$

Réponse

Avec le solveur de Riemann en langage C fourni, la fonction de calcul de la solution exacte en découle facilement [1].

```c
void solexacte(double x, double t, double* w) {
    /* Conditions intiale */
    double hL=2;
    double uL=0;

    double hR=1;
    double uR=0;

   // Variables conservatives
    double wL[2]={hL, hL * uL};
    double wR[2]={hR, hR * uR};

   // Calcul du point d'évaluation (variable autosimilaire)
   double z = x/ (t + 1e-12);

   // Solveur de Riemann fourni
    riem_stvenant(wL, wR, z, w);
}
```

Listing 1 – *Fonction de calcul de la solution exacte*

Le solveur de Riemann exact fourni produit le résultat obtenu à la figure 1 qui nous permet d'observer de façon distincte deux ondes qui se propagent depuis l'origine. Vers la gauche, on a une onde de détente (ou de raréfaction) ($h_L > h^*$) ; et vers la droite, on a une onde de choc

1. Les conditions initiales qui y sont définies seront les mêmes durant la première partie de ce rapport, et tout changement de ces conditions sera alors explicitement indiqué.

($h^* > h_R$). Ici, h^* désigne la hauteur d'eau de la zone intermédiaire entre la gauche et la droite au fur et à mesure que les ondes se séparent.

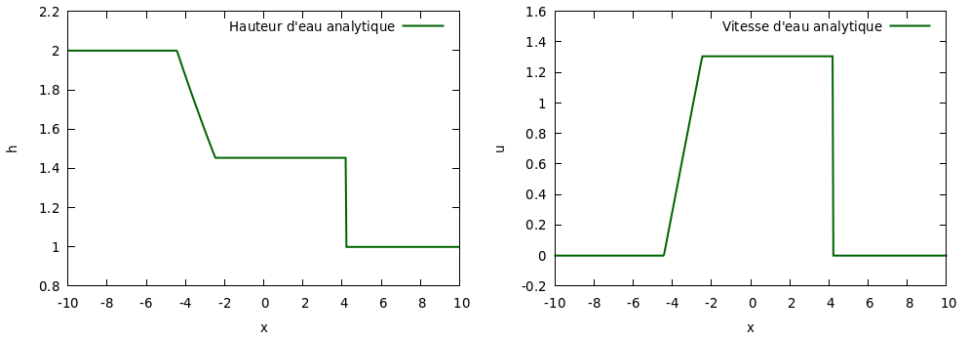


FIGURE 1 – *Solution analytique obtenue à l'aide du solveur de Riemann fourni. A gauche la hauteur d'eau et à droite la vitesse d'eau.* $x_{min} = -10$, $x_{max} = 10$, $t_{final} = 1$ *et le nombre de mailles est de* $N = 500$.

Question 2.

Programmation du schéma de Godunov et vérification au moyen du cas test indiqué.

Réponse

Le schéma de Godunov consiste appliquer le flux physique à la solution du problème de Riemann à l'interface de deux mailles.

```
void fluxnum(double* a, double* b, double* flux) {
   riem_stvenant(a, b, 0., w);      // solveur fourni
   fluxphy(w, flux);
}
```

Listing 2 – *Programmation du flux numérique de Godunov*

Un autre élément important dans cette implémentation est la fonction principale de calcul `godunov_solve`[2].

```
void godunov_solve(godunov* gd, double tmax) {
   double tnow = 0;
   int m = gd->m;
   while (tnow < tmax) {
      double vmax = 0;
      // calcul de la vitesse max
      for (int i = 0; i < gd->N + 2; i++) {
         double vloc = lambda_max(gd->un + m * i);
         vmax = vmax > vloc ? vmax : vloc;
      }
      // Pas de temps
      gd->dt = gd->cfl * gd->dx / vmax;
      // Application du flux numerique
      for (int i = 1; i < gd->N + 1; i++) {
         double flux[m];
         double uL[2], uR[2];
         // Application du flux à droite
         fluxnum(gd->un + i * m, gd->un + (i + 1) * m, flux);
         for (int iv = 0; iv < m; iv++) {
            gd->unp1[i * m + iv] =
               gd->un[i * m + iv] - gd->dt / gd->dx * flux[iv];
         }
         // Application du flux à gauche
```

2. Les autres fonctions nécessaires découlent immédiatement de la définition du problème, elles ne seront pas détaillées ici.

```
            fluxnum(gd->un + (i - 1) * m, gd->un + i * m, flux);
            for (int iv = 0; iv < m; iv++) {
               gd->unp1[i * m + iv] += gd->dt / gd->dx * flux[iv];
            }
         }
         // Mise à jour
         tnow += gd->dt;
         // Conditions au bord
         int i = 0;
         solexacte(gd->xi[i], tnow, gd->unp1 + i * m);
         i = gd->N + 1;
         solexacte(gd->xi[i], tnow, gd->unp1 + i * m);
         // Preparation pour la prochaine etape
         memcpy(gd->un, gd->unp1, (gd->N + 2) * m * sizeof(double));
      }
      gd->tfin = tnow;
   }
```

Listing 3 – *Fonction de résolution du problème de Saint-Venant utilisée dans ce rapport (sauf indication contraire).*

Cette implémentation produit le résultat obtenu à la figure 2 qui permet de confirmer que la programmation de la méthode de Godunov est correcte.

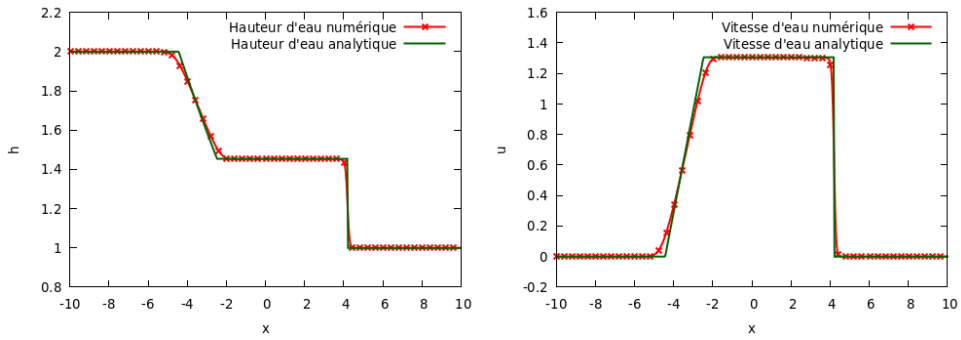


FIGURE 2 – *Solution numérique obtenue par le schéma de Godunov.* $x_{min} = -10$, $x_{max} = 10$, $t_{final} = 1$, $CFL = 0.5$ *et le nombre de mailles est de* $N = 500$.

Question 3.

On considère un bassin $[-10, 10]$ fermé. Le bassin est séparé en deux parties égales grâce à une paroi située en $x = 0$. À l'instant $t = 0$, la partie gauche (respt. droite) est remplie avec de l'eau immobile à une hauteur h_L (respt. h_R). À l'instant $t = 0$, on retire la paroi.

— Écriture et justification de la condition aux limites à imposer en $x = -10$ et $x = 10$.

— Nature de cette condition si le bassin est infini.

Réponse

Pour un bassin fermé, la condition indiquée s'impose par l'application, sur chaque bord, d'une quantité d'eau fantôme, ayant une vitesse opposée à celle des cellules aux bords ($i = 0$ ou $i = N + 1$)[3]. Ainsi, à chaque itération n, on aura

$$h_0^n = h_1^n, \qquad u_0^n = -u_1^n$$
$$h_{N+1}^n = h_N^n, \qquad u_{N+1}^n = -u_N^n$$

3. L'indice pour la discrétisation en espace est noté i, et celui pour la discrétisation en temps n.

En termes de variables conservatives $w = \begin{pmatrix} h \\ hu \end{pmatrix}$, cela reviens à appliquer

$$\begin{aligned} w_0^n[0] &= w_1^n[0], \qquad & w_0^n[1] &= -w_1^n[1] \\ w_{N+1}^n[0] &= w_N^n[0], \qquad & w_{N+1}^n[1] &= -w_N^n[1] \end{aligned}$$

Cette condition traduit le retour de la colonne d'eau après son rebond sur le bord du bassin.

```
int i = 0;
gd->unp1[i*m] = gd->unp1[(i+1)*m];
gd->unp1[i*m+1] = -gd->unp1[(i+1)*m+1];
i = gd->N + 1;
gd->unp1[i*m] = gd->unp1[(i-1)*m];
gd->unp1[i*m+1] = -gd->unp1[(i-1)*m+1];
```

Listing 4 – *Implementation des conditions aux limites pour un bassin fermé.*

En ce qui concerne une piscine infinie, la vague fantôme opposée à l'écoulement est remplacée par une vague dans le même sens que le flux d'eau. Numériquement, cela se traduit par

$$\begin{aligned} h_0^n &= h_1^n, \qquad & u_0^n &= u_1^n \\ h_{N+1}^n &= h_N^n, \qquad & u_{N+1}^n &= u_N^n \end{aligned}$$

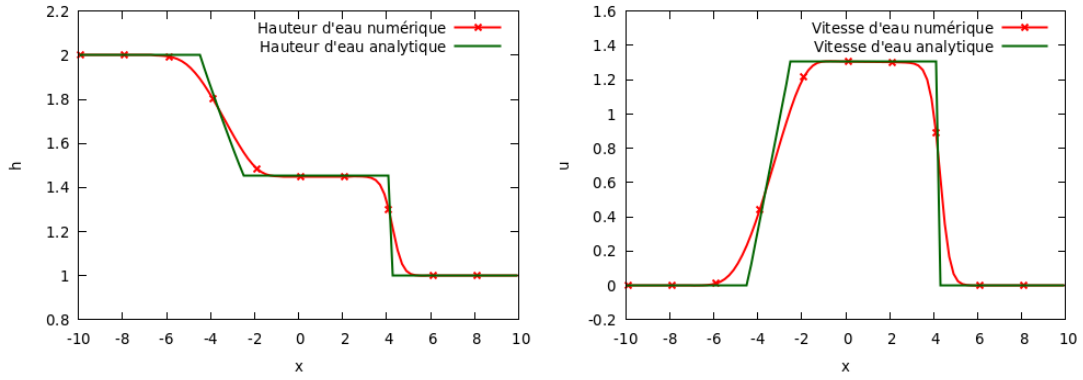
```
int i = 0;
gd->unp1[i*m] = gd->unp1[(i+1)*m];
gd->unp1[i*m+1] = gd->unp1[(i+1)*m+1];
i = gd->N + 1;
gd->unp1[i*m] = gd->unp1[(i-1)*m];
gd->unp1[i*m+1] = gd->unp1[(i-1)*m+1];
```

Listing 5 – *Implementation des conditions aux limites pour un bassin infini.*

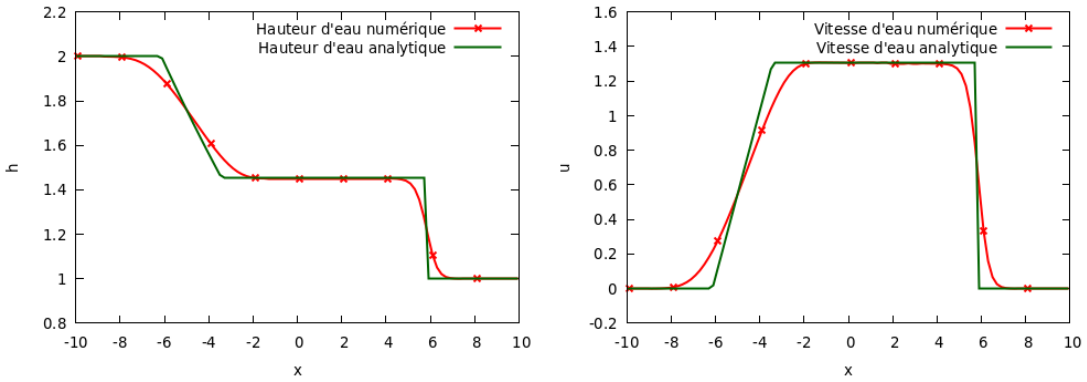
Question 4.

- Calcul de l'évolution de la surface d'eau au cours du temps. Représentation de h et u à divers instants pour diverses finesses de maillage.
- Comparaison avec la solution exacte du problème de Riemann (solution analytique).

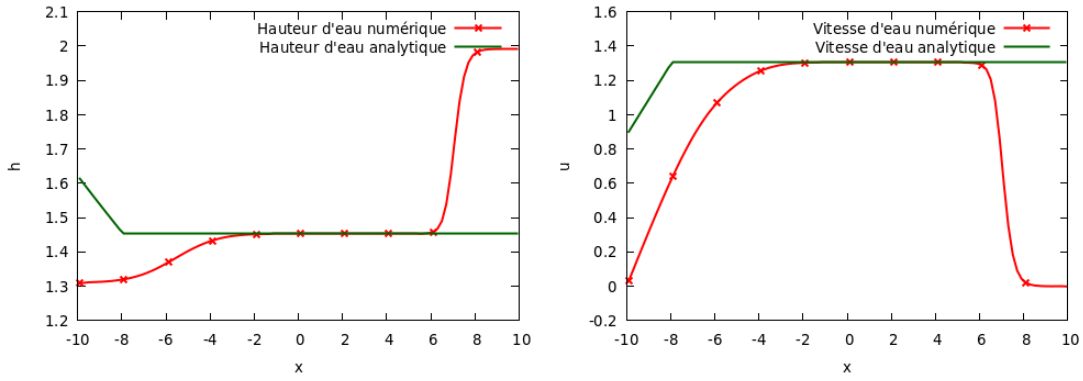
Réponse



(a) $t_f = 1$

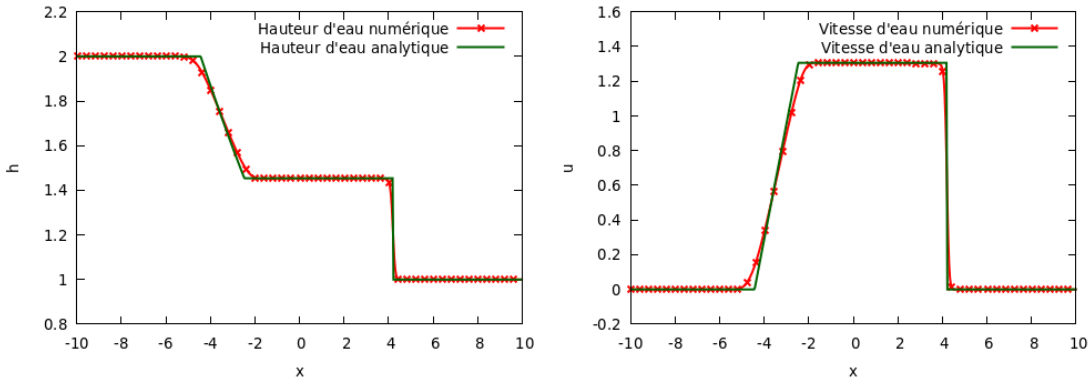


(b) $t_f = 1.366671$

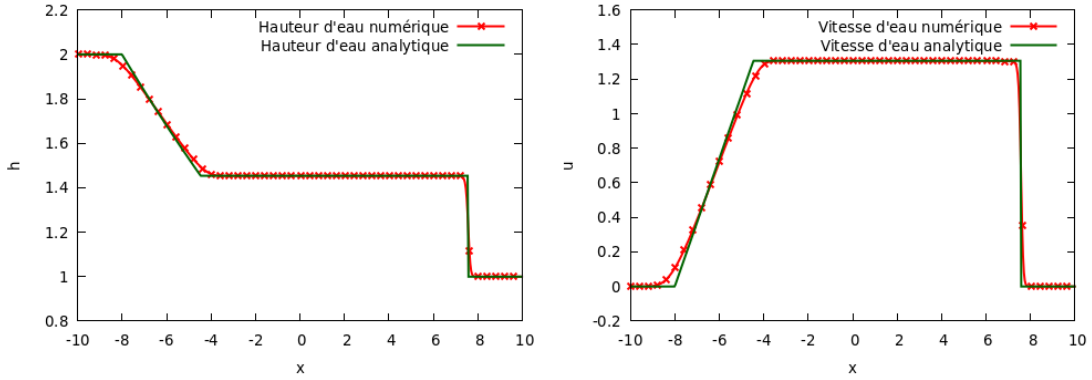


(c) $t_f = 3.2$

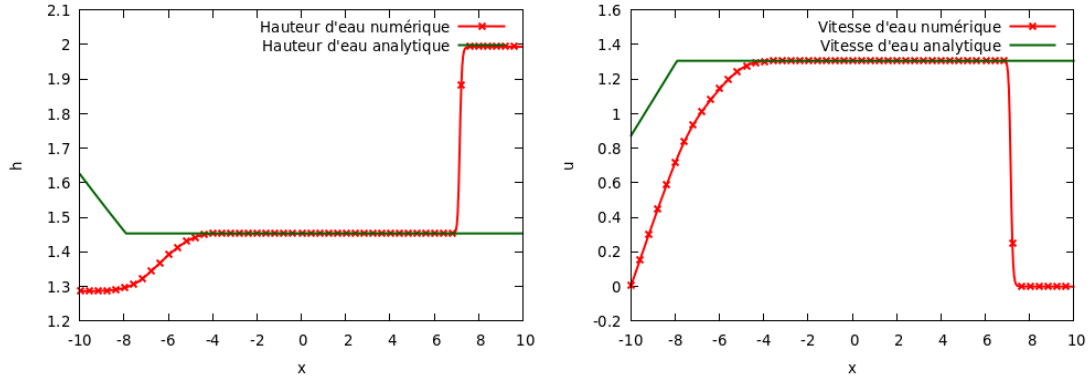
FIGURE 3 – *Évolution de la surface d'eau au cours du temps dans le bassin pour un nombre de mailles* $N = 100$. *Le coefficient CFL vaut* 0.5. *La figure (a) montre le système avant d'atteindre les bords, (b) au moment d'arrivée, et (c) après.*



(a) $t_f = 1$

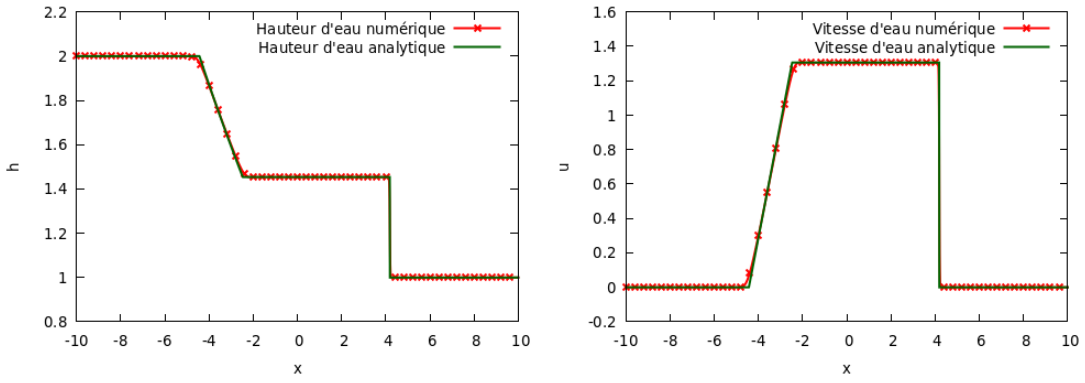


(b) $t_f = 1.808131$

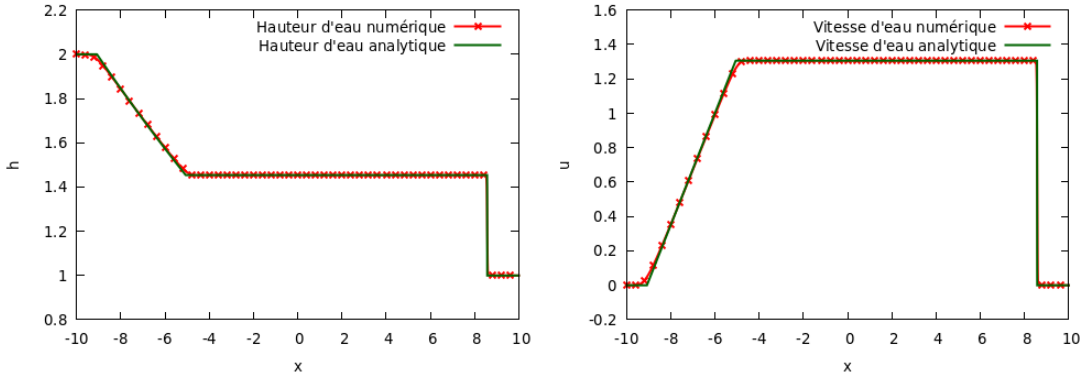


(c) $t_f = 3.2$

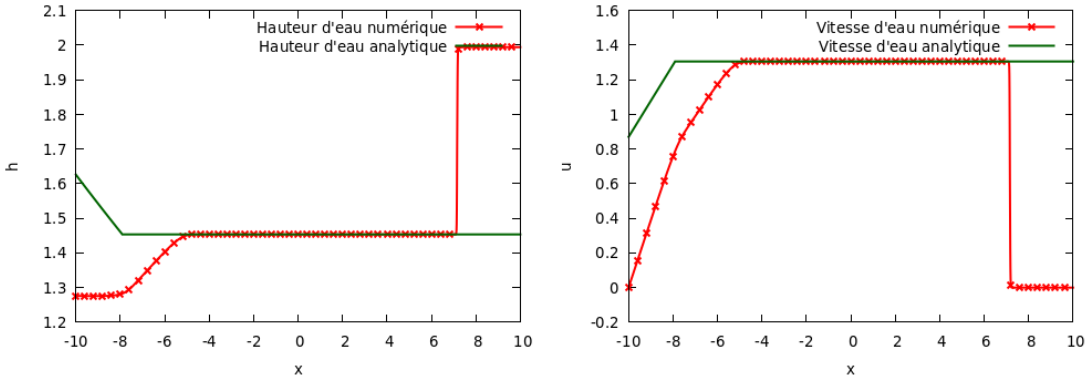
FIGURE 4 – *Évolution de la surface d'eau au cours du temps dans le bassin pour un nombre de mailles* $N = 500$. *Le coefficient CFL vaut* 0.5. *La figure (a) montre le système avant d'atteindre les bords, (b) au moment d'arrivée, et (c) après.*



(a) $t_f = 1$



(b) $t_f = 2.048256$



(c) $t_f = 3.2$

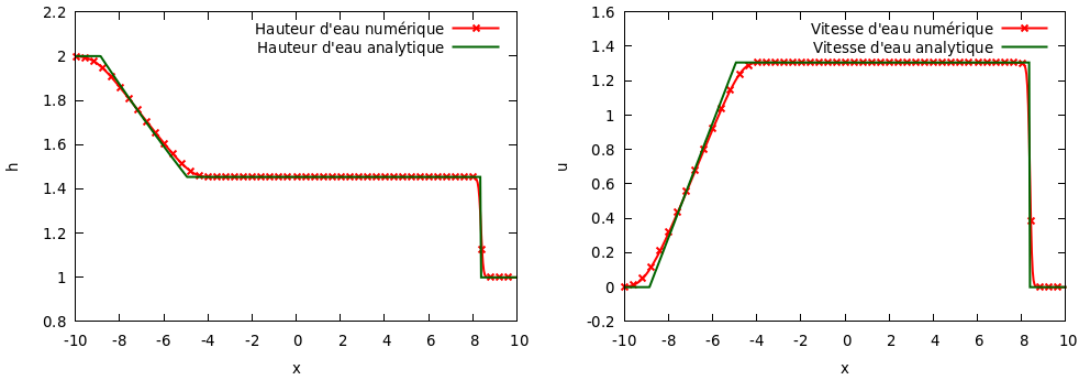
FIGURE 5 – *Évolution de la surface d'eau au cours du temps dans le bassin pour un nombre de mailles* $N = 2500$. *Le coefficient CFL vaut* 0.5. *La figure (a) montre le système avant d'atteindre les bords, (b) au moment d'arrivée, et (c) après.*

Les figures 3 à 5 illustrent le retour de la colonne d'eau après son rebond sur les bords d'un bassin. Les figures (a) de chaque groupe de figures montrent le système avant d'atteindre les bords. Les figures (b) montrent le moment d'arrivée de la solution numérique, qui marque le début de la différence avec la solution exacte[4]. Il est intéressant de remarquer que ce temps d'arrivée devient plus précis avec le raffinement du maillage

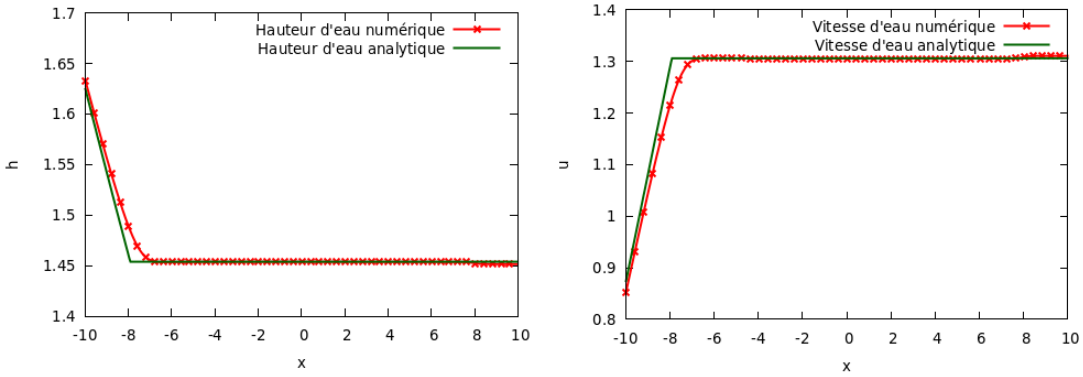
— $t_f = 1.366671$ pour $N = 100$ (cf. figure 3b)
— $t_f = 1.808131$ pour $N = 500$ (cf. figure 4b)
— $t_f = 2.048256$ pour $N = 2500$ (cf. figure 5b)

4. À partir des temps indiqués aux figures (b), les solutions analytiques ne sont plus valides.

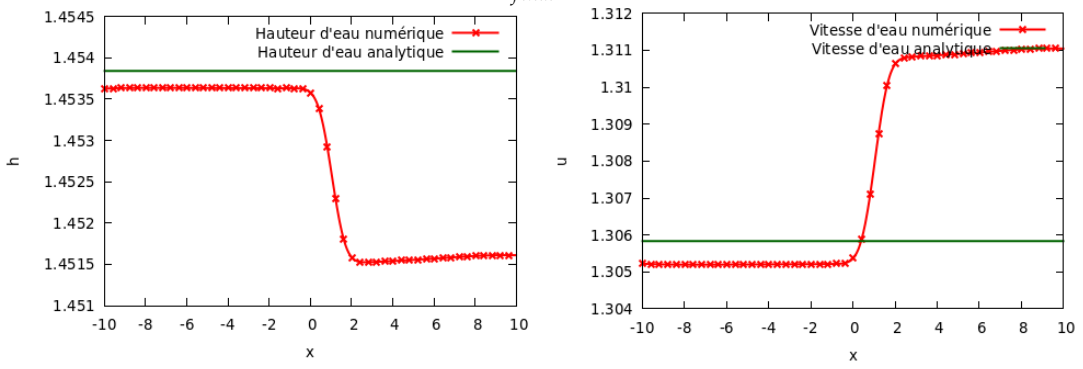
Les figures (c) indiquent ce qui se passe après le rebond de la colonne d'eau. On remarque que la solution exacte (solution exacte du problème de Riemann) n'est plus valide après ce rebond, ce qui explique pourquoi les deux solutions diffèrent si largement.



(a) $t_{final} = 2$



(b) $t_{final} = 3.2$



(c) $t_{final} = 6$

FIGURE 6 – *Illustration du cas avec un bassin infini sur un nombre de mailles* $N = 500$. *Le coefficient CFL vaut* 0.5. *La figure (a) montre le système juste avant d'atteindre les bords, (b) juste après l'arrivée sur les bords, et (c) bien après.*

La figure 6 illustre le bassin infini. Comparé aux cas du bassin fermé (figures 3c, 4c et 5c), la figure 6c ne montre aucun rebond sur les bords (et les deux solutions restent alignées [5]). Pour un temps suffisamment grand, la surface d'eau semble complètement stable ($h \approx 1.45$, et $u \approx 1.30$), sauf en $x = 0$ qui comporte la discontinuité initiale [6] dans le solveur de Riemann exact utilisé par la méthode de Godunov.

5. Il faut observer l'échelle de grandeur sur l'axe y qui est différent d'une figure à l'autre.
6. En effet, $h_R = 2, h_L = 1$ autour de $x = 0$.

Question 5.

Description, programmation et vérification du schéma de Rusanov.

Réponse

Le flux de Rusanov vectoriel est donné par

$$f(w_L, w_R) = \frac{f(w_L) + f(w_R)}{2} - \frac{\lambda}{2}(w_R - w_L)$$

avec

$$\lambda = \max(\rho(A(w_L)), \rho(A(w_R)))$$

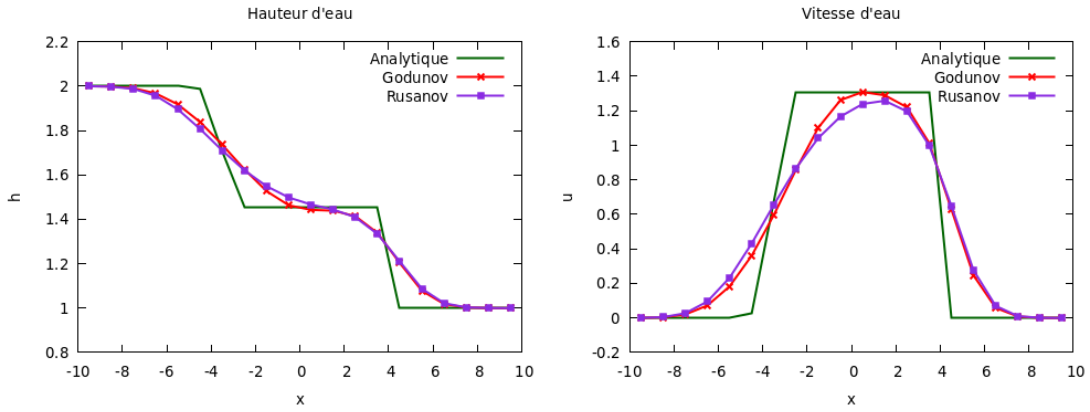
où A désigne la matrice jacobienne de f. Dans le modèle de St-Venant, le rayon spectral vaut $\rho(A) = |u| + \sqrt{gh}$. On peut donc expliciter l'expression de λ qui donne

$$\lambda = \max(|u_L| + \sqrt{gh_L}, |u_R| + \sqrt{gh_R})$$

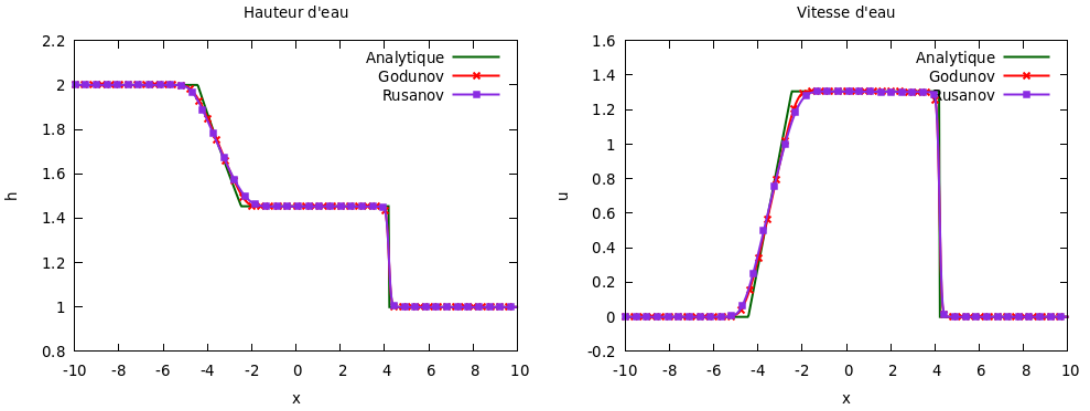
La programmation du schéma donne le code suivant. Les résultats obtenus sont présentés juste après.

```
void flux_rusanov(double *wL, double *wR, double *flux){
   double hL = wL[0];
   double uL = wL[1] / wL[0];
   double hR = wR[0];
   double uR = wR[1] / wR[0];
   double lambdaL = fabs(uL) + sqrt(_G*hL);
   double lambdaR = fabs(uR) + sqrt(_G*hR);
   double lambda = lambdaL>lambdaR ? lambdaL : lambdaR;
   double fL[2];
   double fR[2];
   fluxphy(wL, fL);
   fluxphy(wR, fR);
   flux[0] = (fL[0]+fR[0])/2. - lambda*(wR[0]-wL[0])/2.;
   flux[1] = (fL[1]+fR[1])/2. - lambda*(wR[1]-wL[1])/2.;
}
```

Listing 6 – *Programmation du flux numérique de Rusanov.*



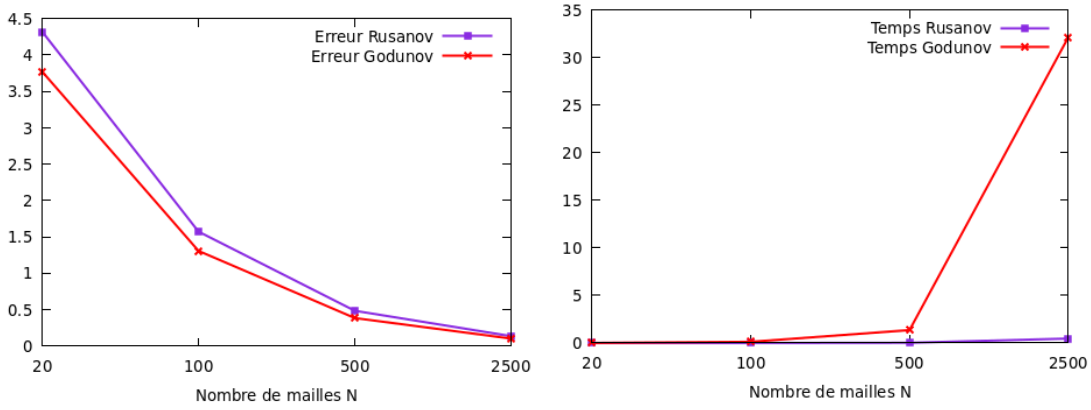
(a) $N = 20$



(b) $N = 500$

FIGURE 7 – *Observation des résultats obtenus par la méthode de Rusanov, à $t = 1$, avec un coefficient $CFL = 0.5$.*

Les observations de précision sur la figure 7 sont confirmées par le graphique ci-bas, qui présente en plus des erreurs en norme L^1, les temps d'exécution.



(a) *Comparaison des erreurs* (b) *Comparaison des temps d'exécution*

FIGURE 8 – *Comparaison du schéma de Rusanov et celui de Godunov. Les erreurs en (a) sont en norme L^1, et traduisent la somme des erreurs sur la hauteur d'eau, et sur la vitesse d'eau.*

Ces résultats confirment bien que le schéma de Rusanov est considérablement plus rapide (figure 8b), mais cela au coût de la précision (figure 8a). Dans les questions qui suivent, nous effectuerons des comparaisons encore plus détaillées entre le schéma de Godunov et un schéma

utilisant le flux de Rusanov baptisé "VFRoe corrigé".

Question 6.

Description du schéma VFRoe et calcul du flux numérique.

Réponse

Tout comme le flux de Godunov, le flux VFRoe repose sur le calcul de la solution du problème de Riemann au point $x/t = 0$. Ce schéma se distingue par deux points principaux :

- On remplace le solveur de Riemann $R(w_L, w_R, 0)$ exact par un solveur de Riemann approché $\tilde{R}(w_L, w_R, 0)$, qui est plus simple et plus rapide à calculer.
- On linéarise les équations régissant le problème autour d'un état moyen noté $\bar{y} = \dfrac{y_L + y_R}{2}$ (on retourne en variables primitives $y = (h, u)^T$ pour définir cet état moyen).

En récapitulatif, il faut résoudre, sur chaque volume fini du maillage, un problème de Riemann donné par :

$$\begin{cases} y_t + B(\bar{y})y_x = 0 \\ y(x,0) = \begin{cases} y_L \ \text{ si } x < 0 \\ y_R \ \text{ si } x > 0 \end{cases} \end{cases} \tag{1}$$

En s'inspirant de l'équation de transport qui nous donne une solution au point $x/t = 0$, nous pouvons écrire :

$$y(0,t) = \frac{y_L + y_R}{2} - \frac{\text{sgn}(B(\bar{y}))}{2}(y_R - y_L)$$

Le signe de la matrice $B(\bar{y}) = \begin{pmatrix} \bar{u} & \bar{h} \\ g & \bar{u} \end{pmatrix}$ est lié à celui de ses valeurs propres ordonnées λ_1 et λ_2.

- Si $\lambda_1 < 0$ et $\lambda_2 < 0$, alors $\text{sgn}(B(\bar{y})) = -I_2$. On en déduit $y(0,t) = y_L$
- Si $\lambda_1 > 0$ et $\lambda_2 > 0$, alors $\text{sgn}(B(\bar{y})) = I_2$. On en déduit $y(0,t) = y_R$
- Si $\lambda_1 < 0$ et $\lambda_2 > 0$, alors le signe est donné par un polynôme du premier degré qui vaut -1 en λ_1, et 1 en λ_2. On montre que ce polynôme vaut $Q(\lambda) = \dfrac{\lambda - \bar{u}}{\sqrt{g\bar{h}}}$. On obtient donc

$$\text{sgn}(B(\bar{y})) = Q(B(\bar{y})) = \frac{1}{\sqrt{g\bar{h}}} \begin{pmatrix} 0 & \bar{h} \\ g & 0 \end{pmatrix}$$

On en déduit

$$y(0,t) = \begin{pmatrix} \frac{h_L + h_R}{2} - \frac{\bar{h}}{2\sqrt{g\bar{h}}}(u_R - u_L) \\ \frac{u_L + u_R}{2} - \frac{g}{2\sqrt{g\bar{h}}}(h_R - h_L) \end{pmatrix}$$

Ce calcul fait, il faut retourner aux variables conservatives afin de définir la solution du problème de Riemann :

$$\tilde{R}(w_L, w_R, 0) = \begin{pmatrix} h(0,t) \\ h(0,t)u(0,t) \end{pmatrix} = \begin{pmatrix} w_1(0,t) \\ w_2(0,t) \end{pmatrix}$$

Le flux numérique de VFRoe en découle par la relation

$$f_{VFRoe}(w_L, w_R) = f(\tilde{R}(w_L, w_R, 0))$$

f étant défini en variables conservatives par $f(w) = \begin{pmatrix} w_2 \\ \frac{w_2^2}{w_1} + \frac{g w_1^2}{2} \end{pmatrix}$.

Question 7.

Programmation du schéma de VFRoe.

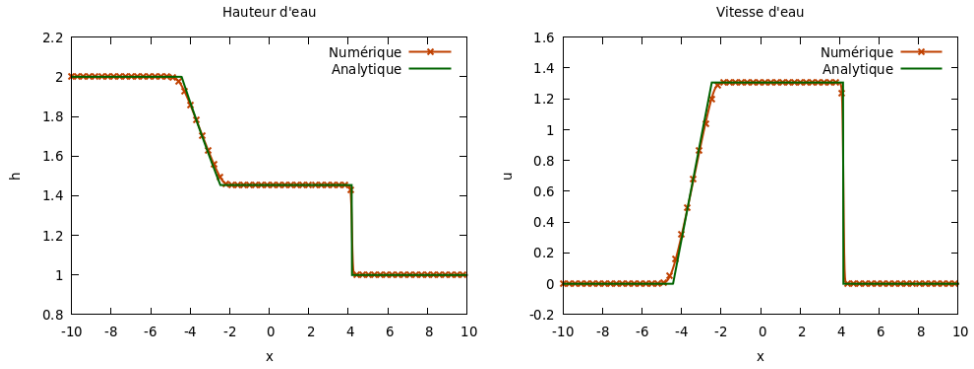
Réponse

Les indications de la question précédente sont implémentées dans la fonction `riem_vfroe` précisée ci-bas.

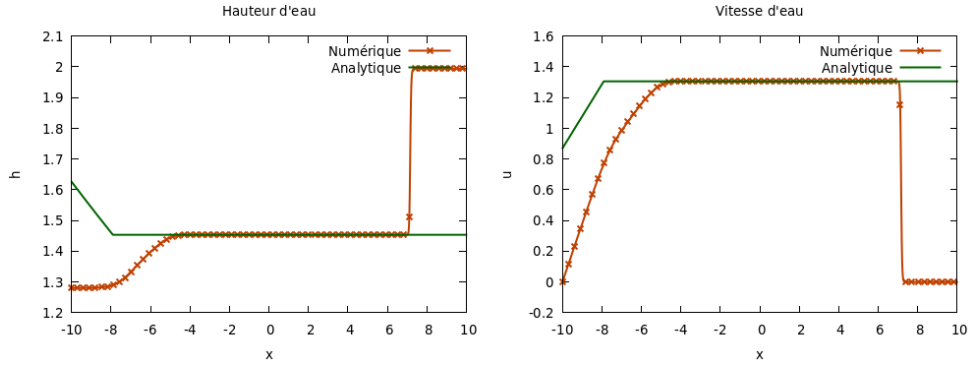
```
void riem_vfroe(double *wL, double *wR, double z, double *w){
    double hL = wL[0];
    double uL = wL[1] / wL[0];
    double hR = wR[0];
    double uR = wR[1] / wR[0];
    double hBar = (hL + hR) /2.0;
    double uBar = (uL + uR) /2.0;
    double cBar = sqrt(_G * hBar);
    double lambda_1 = uBar - cBar;
    double lambda_2 = uBar + cBar;

   // Calcul de y(0,t) par distinction de cas
    double h, u;
    if ((lambda_1 > 0) && (lambda_2 > 0)){
        h = hL;
        u = uL;
    } else if (lambda_1 < 0 && lambda_2 < 0){
        h = hR;
        u = uR;
    // Cas lambda_1 < 0 && lambda_2 > 0
    } else {
        h = hBar - (hBar*(uR-uL))/(2*sqrt(_G*hBar));
        u = uBar - (_G*(hR-hL))/(2*sqrt(_G*hBar));
    }

    w[0] = h;
    w[1] = h*u;
}
```

Listing 7 – *Programmation du flux numérique VFRoe.*

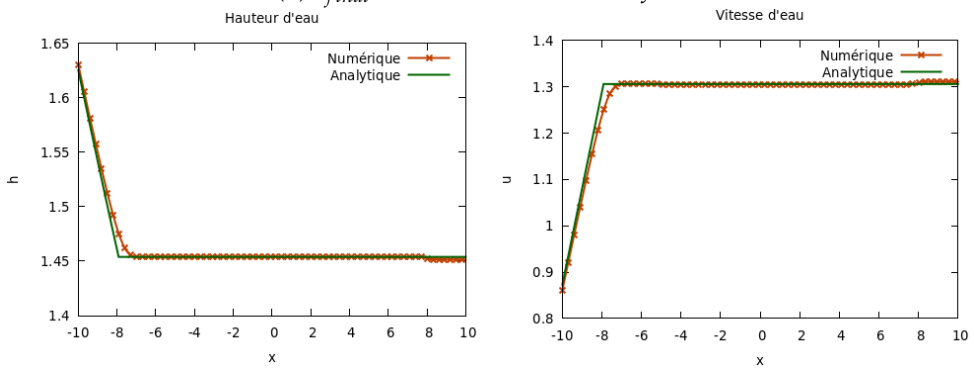
Les résultats sont présentés à la figure 9. On observe effectivement les mêmes résultats que ceux obtenus par les schémas de Godunov (questions 2, 3, et 4) et Rusanov (question 5).



(a) $t_{final} = 1$



(b) $t_{final} = 3.2$ *sur dans le bassin fermé*



(c) $t_{final} = 3.2$ *sur dans le bassin infinie*

FIGURE 9 – *Illustration des résultats obtenus par le schéma VFRoe pour un nombre de mailles* $N = 1000$. *Le coefficient CFL vaut* 0.5. *La figure (a) montre le système avant d'atteindre les bords du domaine sur lesquels on applique des conditions particulières en fonction qu'on soit sur un bassin fermé ou un bassin infini, (b) après l'arrivée des ondes dans un bassin fermé, et (c) après l'arrivée des ondes dans un bassin infini.*

Vérifions que le schéma VFRoe ne donne pas toujours la bonne solution (en choisissant un problème de Riemann associé à une onde de détente qui traverse une valeur propre nulle).

Réponse

On vérifie que le problème de Riemann dont l'état initial est défini par

$$y(x,0) = \begin{cases} \begin{pmatrix} h_L = 1 \\ u_L = -1 \end{pmatrix} & \text{si } x < 0 \\ \begin{pmatrix} h_R = 0.25 \\ u_R = u_L + 2\sqrt{g}(\sqrt{h_L} - \sqrt{h_R}) \end{pmatrix} & \text{si } x > 0 \end{cases}$$

traverse bien une valeur propre $\lambda_1 = u - \sqrt{gh}$ nulle car $u_L - \sqrt{gh_L} < 0$ et $u_R - \sqrt{gh_R} > 0$. Dorénavant, cet état sera l'état initial pour nos simulations.

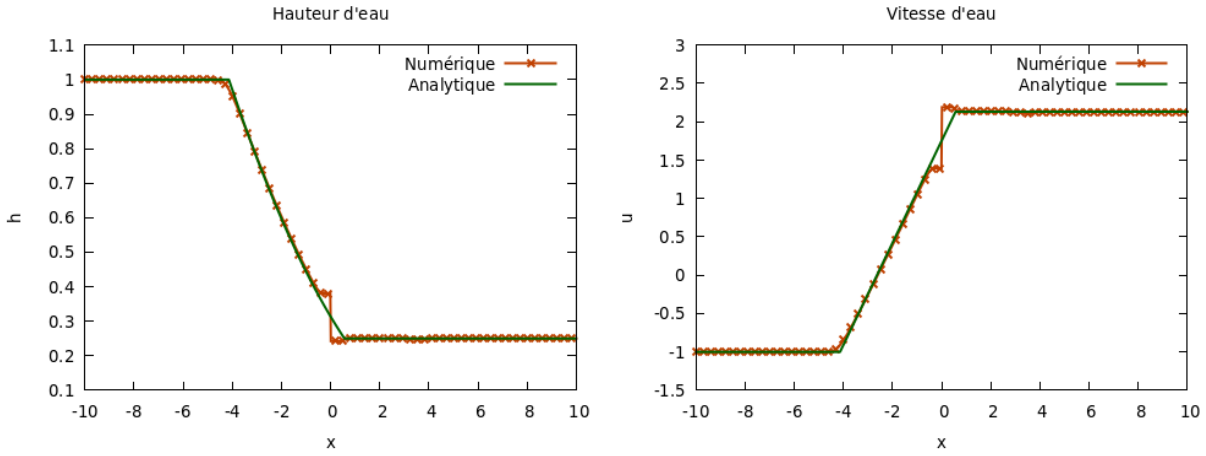


FIGURE 10 – *Mauvaise solution obtenue par le schéma VFRoe lorsque le schéma traverse une valeur propre nulle. On constate effectivement que la solution numérique proche du point de discontinuité $x = 0$ ne correspond pas à la solution analytique exacte. Le nombre de mailles ici vaut $N = 1000$. Le coefficient CFL vaut* 0.5.

Comparé au schéma de Godunov (cf. figure 2) qui donnait des résultats correctes (même aux points de discontinuité), la non-concordance des solutions ici peut-être dû au fait que le flux de VFRoe ne vérifie pas le critère d'entropie (géométrique) de Lax, alors que le flux de Godunov le fait bien.

Question 8.

Correction entropique qui utilise le flux de Rusanov aux points "soniques" (c'est-à-dire les points où la vitesse du "son" $c = \sqrt{gh}$ est égale à $\pm u$).

Réponse

Pour appliquer cette correction, il suffit, avant d'appliquer le flux VFRoe, de vérifier que les vitesses de chacune des ondes ($\lambda_1 = u - c$ et $\lambda_2 = u + c$) aux interfaces d'une maille ne sont pas de signes opposés ($\lambda_{1L} < 0$ et $\lambda_{1R} > 0$ ou $\lambda_{2L} < 0$ et $\lambda_{2R} > 0$). Si cette condition est vérifiée, on applique le flux de Rusanov à la place. Tout ceci correspond à une modification de la fonction de calcul du flux numérique comme suit :

```
void fluxnum(double* a, double* b, double* flux) {
   double w[2];
   double *wL = a;
   double *wR = b;
   double hL = wL[0];
   double uL = wL[1] / wL[0];
```

```
    double hR = wR[0];
    double uR = wR[1] / wR[0];
    double lambda_1L = uL - sqrt(_G*hL);
    double lambda_2L = uL + sqrt(_G*hL);
    double lambda_1R = uR - sqrt(_G*hR);
    double lambda_2R = uR + sqrt(_G*hR);
    // Correction du flux
     if ((lambda_1L<0 && lambda_1R>0) || (lambda_2L<0 && lambda_2R>0))
         flux_rusanov(a, b, flux);
     else{
         riem_vfroe(a, b, 0., w);
         fluxphy(w, flux);
     }
     fluxphy(w, flux);
}
```

Listing 8 – *Application d'une correction entropique qui utilise le flux de Rusanov aux points "soniques".*

On obtient le résultat ci-dessous.

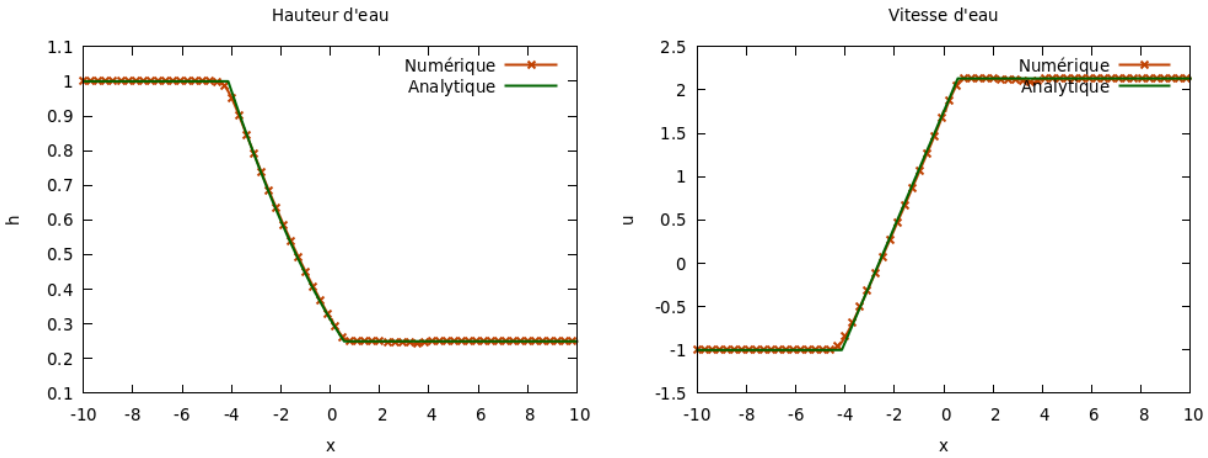


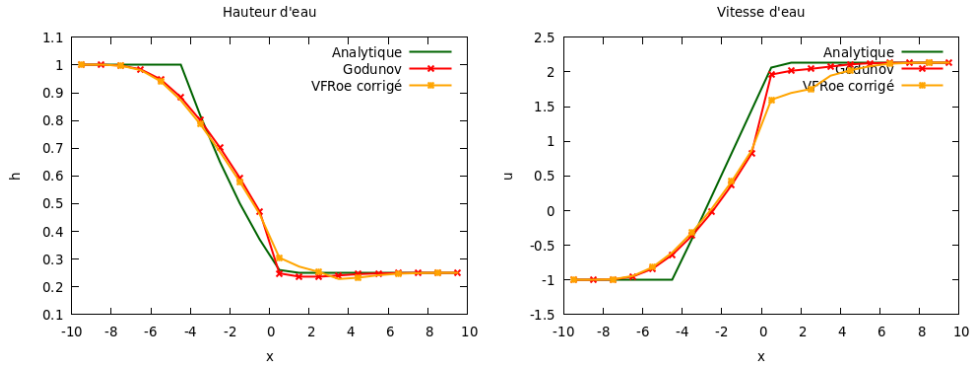
FIGURE 11 – *Correction du schéma VFRoe par un schéma de Rusanov aux points soniques. Comparé à la figure 10, on constate effectivement que la discontinuité au point $x = 0$ disparait. Le nombre de mailles vaut $N = 1000$. Le coefficient CFL vaut* 0.5.

Question 9.

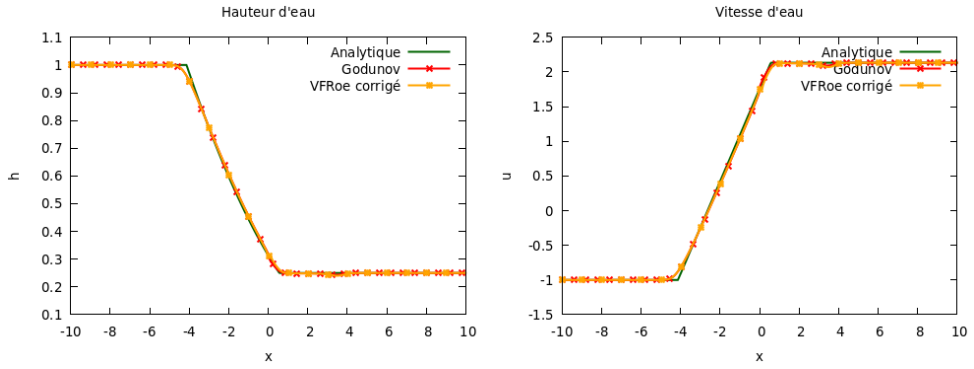
Comparaison du schéma de VFRoe corrigé par le flux de Rusanov avec le schéma de Godunov en termes de précision et de temps de calcul.

Réponse

Les résultats sont présentés ci-bas, aux figures 12 et 14, où la notation "VFRoe corrigé" indique le schéma de VFRoe corrigé par le flux de Rusanov aux points "soniques".



(a) $N = 20$



(b) $N = 500$

FIGURE 12 – *Observation des résultats obtenus par le schéma VFRoe corrigé (en jaune), à $t = 1$, avec un coefficient $CFL = 0.5$.*

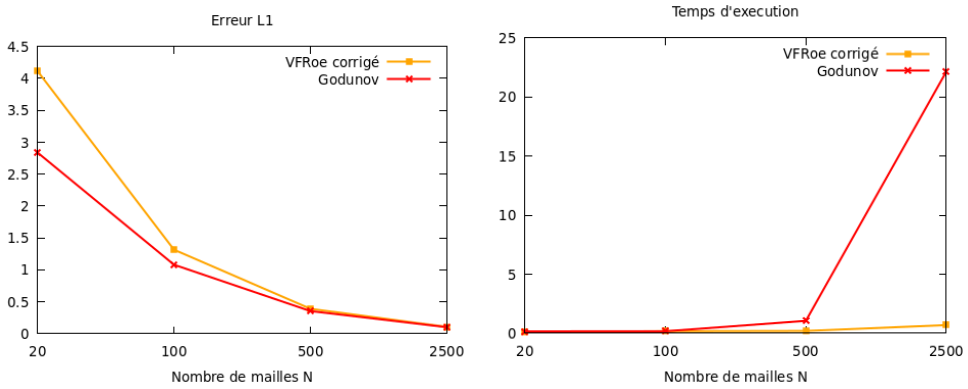
Pour N très faible, le schéma de Godunov semble être le meilleur en termes de précision (figure 12a), mais dès que le nombre de mailles augmente, il n'y a plus d'avantages à utiliser cette technique. En fait, la figure 14 (ainsi que les tableaux 1 et 2) montre que le schéma VFRoe corrigé est considérablement plus rapide que le schéma de Godunov, tout en gardant une précision élevée ; il serait donc judicieux d'utiliser le schéma VFRoe corrigé pour des simulations raffinées.

Erreurs		
N	**VFRoe corrigé**	**Godunov**
20	4.115179	2.839234
100	1.316861	1.081817
500	0.391413	0.355618
2500	0.106957	0.103105

TABLE 1 – *Comparaison du schéma VFRoe corrigé et celui de Godunov pour les erreurs en norme L^1, traduisent la somme des erreurs sur la hauteur et la vitesse d'eau. L'illustration est fournie à la figure 13a.*

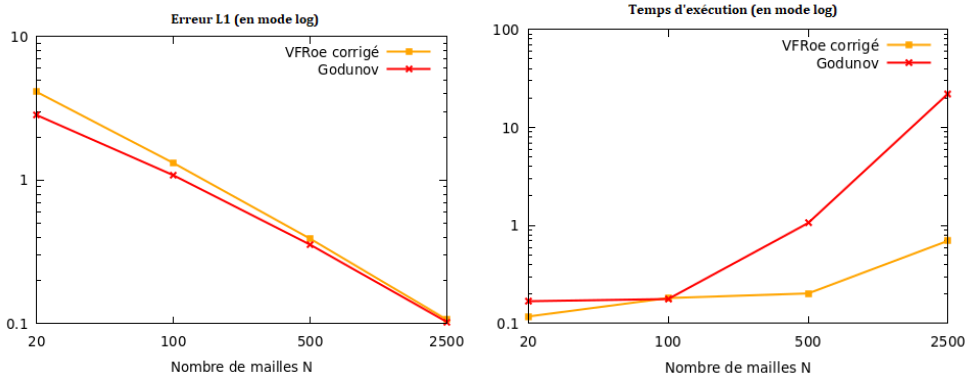
Temps d'execution		
N	**VFRoe corrigé**	**Godunov**
20	0.118	0.169
100	0.182	0.178
500	0.203	1.060
2500	0.700	22.144

TABLE 2 – *Comparaison du schéma VFRoe corrigé et celui de Godunov pour les temps exécution (en seconde), sans tenir compte du temps système. L'illustration est fournie à la figure 13b.*



(a) *Comparaison des erreurs* (b) *Comparaison des temps d'exécution*

FIGURE 13 – *Comparaison du schéma de VFRoe corrigé et celui de Godunov. Les figures (a) et (b) correspondent aux tableaux 1 et 2 respectivement.*



(a) *Comparaison des erreurs* (b) *Comparaison des temps d'exécution*

FIGURE 14 – *Comparaison du schéma de VFRoe corrigé et celui de Godunov en échelle "log". Ces données correspondent aux tableaux 1 et 2 respectivement, déjà illustrés aux figures figures 13a et 13b. Ces résultats confirment bien que le schéma VFRoe corrigé est à préconiser pour des simulations fines, vu qu'il est précis et rapide.*

Question 10.

Calcul du flux de Roe[a]. Programmation et test du schéma de Roe. Vérification qu'il a besoin lui aussi d'une correction entropique. Test du schéma de Roe avec correction entropique.

a. Dans cette question, nous utilisons le flux VFRoe, et non le flux de Roe comme demandé.

Réponse

La correction entropique en question, appliquée aux cas $\lambda_{1L} < 0, \lambda_{1R} > 0$, et $\lambda_{2L} < 0, \lambda_{2R} > 0$, consiste à rajouter une viscosité artificielle aux flux

$$\tilde{f}(w_L, w_R) = f(w_L, w_R) + \frac{\varepsilon}{2}(w_R - w_L)$$

ε étant choisi de façon à minimiser le décalage de chacune des valeurs propres λ_1 et λ_2 de la valeur 0. Vu que le même flux est appliqué aux deux ondes, on obtient

$$\varepsilon = \max\left(\min\left(-\lambda_{1L}, \lambda_{1R}\right), \min\left(-\lambda_{2L}, \lambda_{2R}\right)\right).$$

L'implémentation de cette correction s'effectue indépendamment du calcul de flux VFRoe. On rajoute la viscosité artificielle ε comme l'indique le code de calcul suivant.

```
void fluxnum(double* a, double* b, double* flux) {
   double w[2];
   double *wL = a;
   double *wR = b;
   double hL = wL[0];
   double uL = wL[1] / wL[0];
   double hR = wR[0];
   double uR = wR[1] / wR[0];
   double lambda_1L = uL - sqrt(_G*hL);
   double lambda_2L = uL + sqrt(_G*hL);
   double lambda_1R = uR - sqrt(_G*hR);
   double lambda_2R = uR + sqrt(_G*hR);
   double epsL = 0;
   double epsR = 0;
   // Calcul de la viscosité
   if (lambda_1L < 0 && lambda_1R > 0){
      epsL = fmin(-lambda_1L, lambda_1R);
   }
   if (lambda_2L < 0 && lambda_2R > 0)
      epsR = fmin(-lambda_2L, lambda_2R);
   double eps = fmax(epsL,epsR);
   // Flux VFRoe
   riem_vfroe(a, b, 0., w);
   fluxphy(w, flux);
   // Correction du flux VFRoe par rajout de la viscosité
   for (int i = 0; i < 2; i++)
   {
      flux[i] -= eps/2.0  * (wR[i]-wL[i]);
   }
}
```

Listing 9 – *Application d'une correction entropique "facile" aux flux VFRoe afin de traiter les points soniques.*

On obtient effectivement une correction du résultat, comme le montre la figure ci-bas :

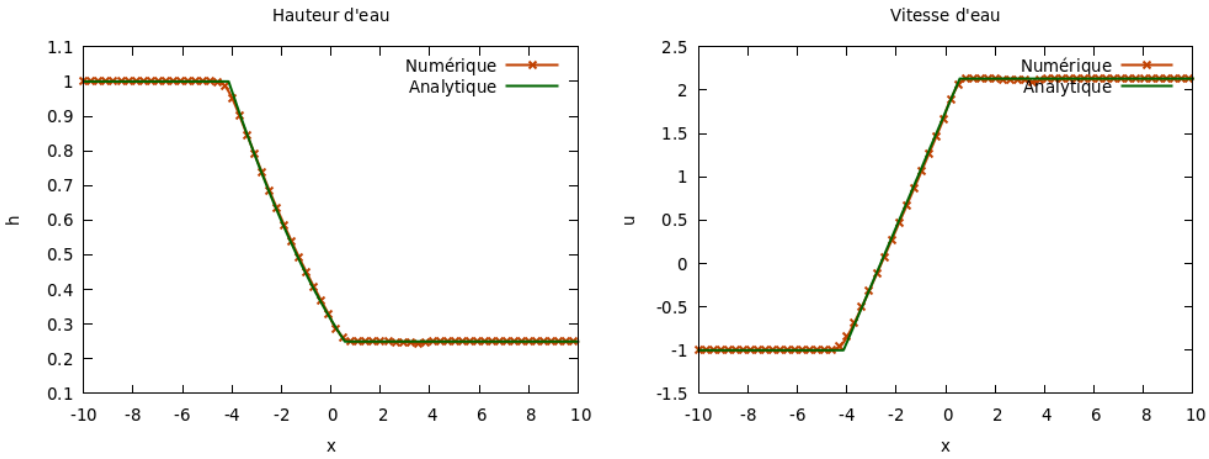


FIGURE 15 – *Correction entropique "facile" du schéma VFRoe. Comparé à la figure 10, on constate effectivement que la discontinuité au point $x = 0$ disparait. Le nombre de mailles ici vaut $N = 1000$. Le coefficient CFL vaut* 0.5.

Question 11.

Programmation de la méthode MUSCL pour le schéma VFRoe sans correction entropique avec intégration en temps de RK2.

Réponse

La méthode MUSCL consiste à calculer des pentes en espace et en temps que l'on applique aux centres des mailles[7]. Ici, on ne s'intéresse qu'à la correction MUSCL en espace, et les itérations en temps seront gérés par un schéma de Runge-Kutta d'ordre 2 que nous détaillons ci-bas. Après l'application des pentes MUSCL en espace, on obtient les valeurs $w_{+_L}, w_{+_R}, w_{-_L}$, et w_{-_R} ; on fait l'approximation $w_i(t) \approx w(x_i, t)$ pour se retrouver avec :

$$w_i'(t) + \frac{f(w_{+_L}(t), w_{+_R}(t)) - f(w_{-_L}(t), w_{-_R}(t))}{\Delta x} = 0$$

qu'on résout par une intégration en temps RK2. En d'autres termes, on calcule k_1 et k_2 tel que

$$k_1 = \frac{\Delta t}{\Delta x}\left[f(w_{+_L}, w_{+_R} - f(w_{-_L}, w_{-_R})\right]$$
$$k_2 = \frac{\Delta t}{\Delta x}\left[f(w_{+_L} + k_1, w_{+_R} + k_1) - f(w_{-_L} + k_1, w_{-_R} + k_1)\right]$$

enfin, on pose

$$w_i^{n+1} = w_i^n + \frac{1}{2}(k_1 + k_2)$$

Les modifications à apporter au code pour implémenter la méthode MUSCL se situent toutes au niveau de la fonction principale de résolution `godunov_solve`. On y retrouve le calcul des pentes en espace, celui des pentes en temps, et leur application au schéma numérique.

```
void godunov_solve(godunov* gd, double tmax) {
    double tnow = 0;
    int m = gd->m;
    while (tnow < tmax) {
        double vmax = 0;
        // Calcul de la vitesse max
```

7. Le lecteur est renvoyé à la question 4 de la partie 2 du T.P. #1, ou à la question 8 du T.P. #3 pour une description détaillée de la méthode MUSCL

```
        for (int i = 0; i < gd->N + 2; i++) {
            double vloc = lambda_max(gd->un + m * i);
            vmax = vmax > vloc ? vmax : vloc;
        }
        // Calcul des pentes pour MUSCL en espace
        double si[m*(gd->N+2)], ri[m*(gd->N+2)];
        for (int i = 1; i < gd->N + 1; i++) {
            for (int iv = 0; iv < m; iv++) {
                double alpha = (gd->un[i * m + iv] - gd->un[(i-1) * m + iv])/gd->dx;
                double beta = (gd->un[(i+1) * m + iv] - gd->un[i * m + iv])/gd->dx;
                double gamma = (gd->un[(i+1) * m + iv] - gd->un[(i-1) * m + iv])/(2.0*gd->dx);
                si[i * m + iv] = minmod(alpha, beta, gamma);
            }
        }
      // Calcul du pas de temps
        gd->dt = gd->cfl * gd->dx / vmax;
      // Application du schéma de Runge Kutta d'ordre 2 en temps
        for (int i = 1; i < gd->N + 1; i++) {
            double uLplus[2], uRplus[2], uLmoins[2], uRmoins[2];
            for (int iv = 0; iv < m; iv++) {
                uLplus[iv] = gd->un[i*m+iv] + si[i*m+iv]*gd->dx/2.0;
                uRplus[iv] = gd->un[(i+1)*m+iv] - si[(i+1)*m+iv]*gd->dx/2.0;
                uLmoins[iv] = gd->un[(i-1)*m+iv] + si[(i-1)*m+iv]*gd->dx/2.0;
                uRmoins[iv] = gd->un[i*m+iv] - si[i*m+iv]*gd->dx/2.0;
            }
            double fluxPlus[m];
            double fluxMoins[m];
            fluxnum(uLplus, uRplus, fluxPlus);
         fluxnum(uLmoins, uRmoins, fluxMoins);
         // Pente RungeKutta k1
            double k1[m];
            for (int iv = 0; iv < m; iv++) {
                k1[iv] = - gd->dt * (fluxPlus[iv] - fluxMoins[iv])/gd->dx;
            }

            double uLplusPrime[m], uRplusPrime[m], uLmoinsPrime[m], uRmoinsPrime[m];
            for (int iv = 0; iv < m; iv++) {
                uLplusPrime[iv] = uLplus[iv] + k1[iv];
                uRplusPrime[iv] = uRplus[iv] + k1[iv];

                uLmoinsPrime[iv] = uLmoins[iv] + k1[iv];
                uRmoinsPrime[iv] = uRmoins[iv] + k1[iv];
            }
            double fluxPlusPrime[m];
            double fluxMoinsPrime[m];
            fluxnum(uLplusPrime, uRplusPrime, fluxPlusPrime);
            fluxnum(uLmoinsPrime, uRmoinsPrime, fluxMoinsPrime);
         // Pente RungeKutta k2
            double k2[m];
            for (int iv = 0; iv < m; iv++) {
                k2[iv] = - gd->dt * (fluxPlusPrime[iv] - fluxMoinsPrime[iv])/gd->dx;
            }
            for (int iv = 0; iv < m; iv++) {
                gd->unp1[i * m + iv] =  gd->un[i*m+iv] + (k1[iv] + k2[iv]) / 2.0;
            }
        }
        // Mise à jour du temps
        tnow += gd->dt;
      // Conditions aux limites
        int i = 0;
        solexacte(gd->xi[i], tnow, gd->unp1 + i * m);
        i = gd->N + 1;
        solexacte(gd->xi[i], tnow, gd->unp1 + i * m);
      // Copie des valeurs
        memcpy(gd->un, gd->unp1, (gd->N + 2) * m * sizeof(double));
    }
    gd->tfin = tnow;
}
```

Listing 10 – *Application de la correction MUSCL en espace, et d'un schéma de Runge-Kutta d'ordre 2 en temps. La fonction "fluxnum" observée correspond au flux VFRoe sans correction entropique.*

La programmation de la correction MUSCL en espace, et d'un schéma de Runge-Kutta d'ordre

2 en temps ne produit pas exactement les même résultats que ceux de VFRoe sur lesquels la correction entropique a été appliquée (cf. figures 15 et 16). En comparant cette fois les figures 10 et 16 [8], on observe que la discontinuité au point $x = 0$ à entièrement disparue, mais qu'une nouvelle discontinuité en $x = -4$ s'est formée. Vu que le problème du point sonique en $x = 0$ a été résolu, on en déduit que la correction entropique n'est plus nécessaire.

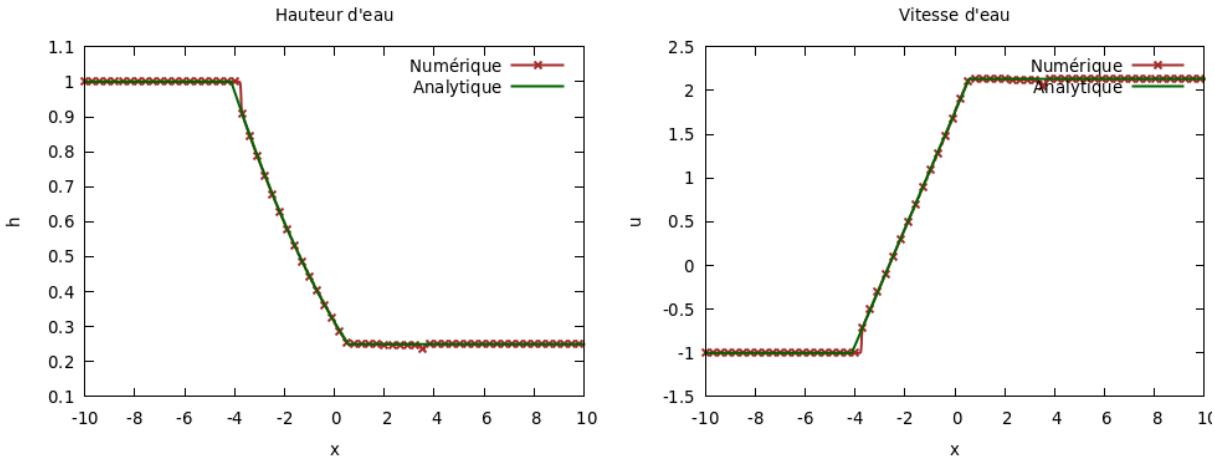


FIGURE 16 – *Observation des résultats obtenus par la correction MUSCL en espace et un schéma RK2 en temps, appliquée au schéma VFRoe. Les conditions initiales sont les même que celles utilisées jusqu'a présent, en plus d'avoir la garantie du point sonique i.e* $h_L = 1$, $u_L = -1$, $h_R = 0.25$, $u_R = u_L + 2\sqrt{g}(\sqrt{h_L} - \sqrt{h_R})$.

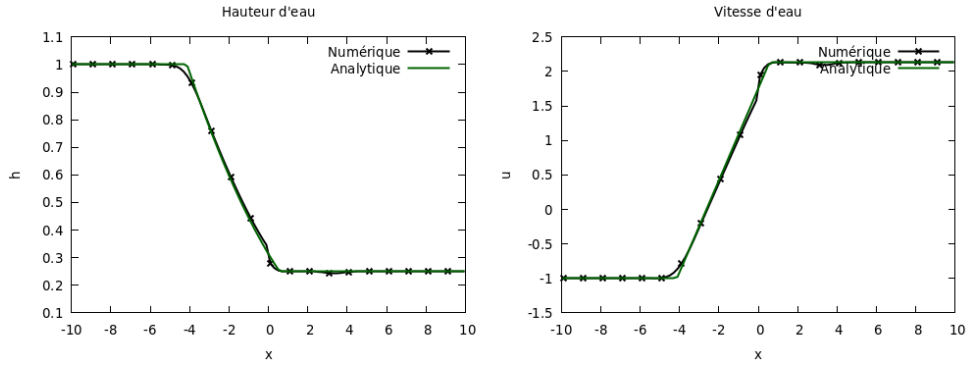
Afin d'améliorer les résultats [9], on décide d'implémenter une correction MUSCL pour le schéma VFRoe en espace et en temps. La programmation de ce cette méthode (dont le code de calcul est fourni ci-bas) produit les mêmes résultats que ceux obtenus pour le schéma de Roe avec la correction entropique (figures 15 et 17b). En comparant les figures 16 et 17b, on constate effectivement l'amélioration des résultats. Cette correction MUSCL (en temps et en espace) ne nécessite guère de correction entropique. Les méthodes de types MUSCL sont en fait les meilleures méthodes pour corriger les points soniques, car elles sont plus pratiques.

```
void godunov_solve(godunov* gd, double tmax) {
   double tnow = 0;
   int m = gd->m;
   while (tnow < tmax) {
      double vmax = 0;
      // calcul de la vitesse max
      for (int i = 0; i < gd->N + 2; i++) {
         double vloc = lambda_max(gd->un + m * i);
         vmax = vmax > vloc ? vmax : vloc;
      }
      // Calcul des pentes pour MUSCL
      double si[m*(gd->N+2)], ri[m*(gd->N+2)];
      for (int i = 1; i < gd->N + 1; i++) {
         for (int iv = 0; iv < m; iv++) {
            double alpha = (gd->un[i * m + iv] - gd->un[(i-1) * m + iv])/gd->dx;
            double beta = (gd->un[(i+1) * m + iv] - gd->un[i * m + iv])/gd->dx;
            double gamma = (gd->un[(i+1) * m + iv] - gd->un[(i-1) * m + iv])/(2.0*gd->dx);
            si[i * m + iv] = minmod(alpha, beta, gamma);
         }
         ri[i * m + 0] = -si[i * m +1];
         ri[i * m + 1] = -(_G*gd->un[i*m] - gd->un[i*m+1]*gd->un[i*m+1]/(gd->un[i*m]*gd->un[i*m]))*si[i*m
    +0] - 2*si[i*m+1]*gd->un[i*m+1]/gd->un[i*m];
      }
      // Pas de temps
      gd->dt = gd->cfl * gd->dx / vmax;
```

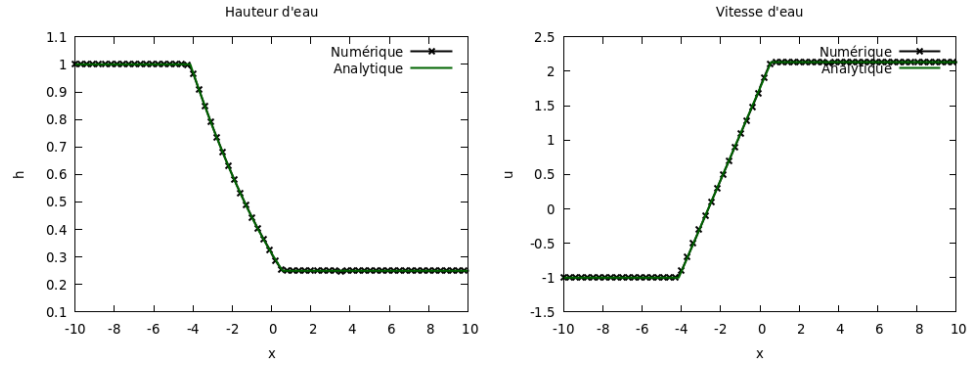
8. On y observe le point sonique à traiter en $x = 0$.
9. Il s'agit de supprimer la discontinuité en $x = -4$.

```
      // Application du flux numerique
      for (int i = 1; i < gd->N + 1; i++) {
         double flux[m];
         double uL[2], uR[2];
         // Application de MUSCL à droite
         for (int iv = 0; iv < m; iv++) {
            uL[iv] = gd->un[i*m+iv] + si[i*m+iv]*gd->dx/2.0 + ri[i*m+iv]*gd->dt/2.0;
            uR[iv] = gd->un[(i+1)*m+iv] - si[(i+1)*m+iv]*gd->dx/2.0 + ri[(i+1)*m+iv]*gd->dt/2.0;
         }
         fluxnum(uL, uR, flux);
         for (int iv = 0; iv < m; iv++) {
            gd->unp1[i * m + iv] =
               gd->un[i * m + iv] - gd->dt / gd->dx * flux[iv];
         }
         // Application de MUSCL à gauche
         for (int iv = 0; iv < m; iv++) {
            uL[iv] = gd->un[(i-1)*m+iv] + si[(i-1)*m+iv]*gd->dx/2.0 + ri[(i-1)*m+iv]*gd->dt/2.0;
            uR[iv] = gd->un[i*m+iv] - si[i*m+iv]*gd->dx/2.0 + ri[i*m+iv]*gd->dt/2.0;
         }
         fluxnum(uL, uR, flux);
         for (int iv = 0; iv < m; iv++) {
            gd->unp1[i * m + iv] += gd->dt / gd->dx * flux[iv];
         }
      }
      // Mise à jour
      tnow += gd->dt;
      // Conditions au bord
      int i = 0;
      solexacte(gd->xi[i], tnow, gd->unp1 + i * m);
      i = gd->N + 1;
      solexacte(gd->xi[i], tnow, gd->unp1 + i * m);
      // Preparation pour la prochaine etape
      memcpy(gd->un, gd->unp1, (gd->N + 2) * m * sizeof(double));
   }
   gd->tfin = tnow;
}
```

Listing 11 – *Application de la correction MUSCL. La fonction "fluxnum" observée correspond au flux VFRoe sans correction entropique.*



(a) $N = 100$



(b) $N = 1000$

FIGURE 17 – *Observation des résultats obtenus par la correction MUSCL en espace et en temps appliquée au schéma VFRoe. Les conditions initiales sont les mêmes que celles utilisées jusqu'à présent, en plus d'avoir la garantie du point sonique i.e* $h_L = 1$, $u_L = -1$, $h_R = 0.25$, $u_R = u_L + 2\sqrt{g}(\sqrt{h_L} - \sqrt{h_R})$.

UNIVERSITÉ DE STRASBOURG
UFR DE MATHÉMATIQUES ET INFORMATIQUE

TP #3

Etudiant : *Roussel Desmond Nzoyem*

UE : *EDP 2* – Enseignant : *Pr. Philippe Helluy*
Date : *3 janvier 2021*

Résolution numérique du système de Saint-Venant en deux dimensions

Le modèle de Saint-Venant en deux dimensions s'écrit

$$\begin{aligned} \partial_t w + \partial_x(hu) + \partial_y(hv) &= 0 \\ \partial_t(hu) + \partial_x\left(hu^2 + g\frac{h^2}{2}\right) + \partial_y(huv) &= -gh\partial_x A \\ \partial_t(hv) + \partial_x(huv) + \partial_y\left(hv^2 + g\frac{h^2}{2}\right) &= -gh\partial_y A \end{aligned} \tag{$\mathcal{S}$}$$

où

- $A(x,y)$ désigne l'altitude du point (x,y)
- $h(x,y,t) \geq 0$ désigne la hauteur du point (x,y) à l'instant t
- $u(x,y,t)$ désigne la composante suivant x du vecteur vitesse au point (x,y) à l'instant t
- $v(x,y,t)$ désigne la composante suivant y du vecteur vitesse au point (x,y) à l'instant t

Notre domaine d'étude sera le carré $[0,L] \times [0,L]$ tel que la hauteur deau h soit petite devant L.

Question 1.

Écriture du système de Saint-Venant 2D sous la forme

$$\partial_t w + \partial_x f_1(w) + \partial_y f_2(w) = S(w) \tag{$\mathcal{S}'$}$$

Réponse

On introduit A comme une variable indépendante du temps i.e $\partial_t A = 0$ et on considère la variable conservative

$$w = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} = \begin{pmatrix} h \\ hu \\ hv \\ A \end{pmatrix}$$

On remarque que le système $(\mathcal{S})$ se met facilement sous la forme

$$\partial_t w + \partial_x f_1(h,u,v,A) + \partial_y f_2(h,u,v,A) = S(h,u,v,A)$$

où les fonctions f_1, f_2, et S sont exprimées en termes de variables primitives h, u, v et A.

$$f_1(h,u,v,A) = \begin{pmatrix} hu \\ hu^2 + \dfrac{gh^2}{2} \\ huv \\ 0 \end{pmatrix}, \qquad f_2(h,u,v,A) = \begin{pmatrix} hv \\ huv \\ hv^2 + \dfrac{gh^2}{2} \\ 0 \end{pmatrix}, \qquad S(h,u,v,A) = \begin{pmatrix} 0 \\ -gh\partial_x A \\ -gh\partial_y A \\ 0 \end{pmatrix}$$

En supposant $w_1 > 0$, on les exprime en fonction de w en remarque que

$$\begin{pmatrix} h \\ u \\ v \\ A \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2/w_1 \\ w_3/w_1 \\ w_4 \end{pmatrix}$$

Le système $(\mathcal{S})$ se met sous la forme $(\mathcal{S}')$ demandée avec

$$f_1(w) = \begin{pmatrix} w_2 \\ w_2^2/w_1 + gw_1^2/2 \\ w_2w_3/w_1 \\ 0 \end{pmatrix}, \qquad f_2(w) = \begin{pmatrix} w_3 \\ w_2w_3/w_1 \\ w_3^2/w_1 + gw_1^2/2 \\ 0 \end{pmatrix}, \qquad S(w) = \begin{pmatrix} 0 \\ -gh\partial_x w_4 \\ -gh\partial_x w_4 \\ 0 \end{pmatrix}$$

Question 2.

Vérifions que le système de Saint-Venant 2D est hyperbolique.

Réponse

Pour cela, posons $n = (n_1, n_2)$ et $f(w) \cdot n = f_1(w)n_1 + f_2(w)n_2$ et vérifions que la matrice jacobienne de la divergence du flux $B = \partial_w (f(w) \cdot n)$ est diagonalisable pour tout w tel que $w_1 \geq 0$ et pour tout n.
En remarquant que les équations sont invariantes par rotation (aucune direction de propagation n'est privilégiée), il suffit de regarder l'hyperbolicité dans une direction. Prenons donc $n = (1, 0)$ et étudions les valeurs propres de B. On a, pour $w_1 > 0$,

$$f(w) \cdot n = f_1(w) = \begin{pmatrix} w_2 \\ \dfrac{w_2^2}{w_1} + \dfrac{gw_1^2}{2} \\ \dfrac{w_2w_3}{w_1} \\ 0 \end{pmatrix}$$

et

$$B = \partial_w (f(w) \cdot n) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \dfrac{w_2^2}{w_1^2} + gw_1 & \dfrac{2w_2}{w_1} & 0 & 0 \\ -\dfrac{w_2w_3}{w_1} & \dfrac{w_3}{w_1} & \dfrac{w_2}{w_1} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

et donc pour tout $\lambda \in \mathbb{R}$,

$$\begin{aligned}
\det(B - \lambda I_3) &= \det \begin{pmatrix} -\lambda & 1 & 0 & 0 \\ \frac{w_2^2}{w_1^2} + g w_1 & \frac{2w_2}{w_1} - \lambda & 0 & 0 \\ -\frac{w_2 w_3}{w_1} & \frac{w_3}{w_1} & \frac{w_2}{w_1} - \lambda & 0 \\ 0 & 0 & 0 & -\lambda \end{pmatrix} \\
&= -\lambda \left[-\lambda \left(\frac{2w_2}{w_1} - \lambda \right) \left(\frac{w_2}{w_1} - \lambda \right) + \left(\frac{w_2^2}{w_1^2} + g w_1 \right) \left(\frac{w_2}{w_1} - \lambda \right) \right] \\
&= -\lambda \left(\frac{w_2}{w_1} - \lambda \right) \left[\lambda^2 - \lambda \left(\frac{2w_2}{w_1} \right) + \frac{w_2^2}{w_1^2} - g w_1 \right] \\
&= -\lambda \left(\frac{w_2}{w_1} - \lambda \right) \left(\frac{w_2}{w_1} - \sqrt{g w_1} \right) \left(\frac{w_2}{w_1} + \sqrt{g w_1} \right)
\end{aligned}$$

On constate donc que B admets trois valeurs propres réelles (pas forcément ordonnées et pas forcément distinctes) définies par

$$\begin{aligned}
\lambda_1(w, n) &= \frac{w_2}{w_1} - \sqrt{g w_1} \\
\lambda_2(w, n) &= 0 \\
\lambda_3(w, n) &= \frac{w_2}{w_1} \\
\lambda_4(w, n) &= \frac{w_2}{w_1} + \sqrt{g w_1}
\end{aligned}$$

Ce calcul se généralise à toutes les directions n de l'espace. On montre, en repassant aux variables primitives et en posant $U = (u, v)$, que

$$\begin{aligned}
\lambda_1(w, n) &= U \cdot n - \sqrt{gh} \\
\lambda_2(w, n) &= 0 \\
\lambda_3(w, n) &= U \cdot n \\
\lambda_4(w, n) &= U \cdot n + \sqrt{gh}
\end{aligned}$$

valable pour tout w tel que $w_1 \geq 0$[1]. Le polynôme caractéristique de B est scindé à valeurs propres réelles (et donc diagonalisable). Le système $(\mathcal{S})$ de Saint-Venant 2D est donc hyperbolique.

Question 3.

Expression du flux numérique $f(w_L, w_R, n)$ de Rusanov en 2D en vue d'une approximation numérique du système de Saint-Venant.

Réponse

En s'inspirant de la définition du flux de Rusanov en 1D, on écrit

$$f(w_L, w_R, n) = \frac{1}{2} \left(f(w_L, n) + f(w_R, n) \right) - \frac{\lambda_{LR}}{2} (w_R - w_L)$$

1. Le cas $w_1 = 0$ i.e. $h = 0$ tolérant les zones sèches nécessite un traitement particulier, qu'on pourra résoudre en travaillant avec les variables primitives.

où la vitesse de Rusanov locale satisfait la condition suivante [2]

$$\lambda_{LR} \geq \max_{k=1,2,3} \left\{ \max \left\{ |\lambda_k(w_L, n)|, \, |\lambda_k(w_R, n)| \right\} \right\}$$

on choisit donc

$$\lambda_{LR} = \max \left\{ |U_L \cdot n| + \sqrt{g h_L}, \; |U_R \cdot n| + \sqrt{g h_R} \right\}$$

À l'intérieur du Kernel de calcul OpenCL utilisé pour les simulations, le flux de Rusanov s'implémente comme ceci :

```
void flux_rusa_2d(float *wL, float *wR, float* vnorm, float* flux){
   float fL[_M];
   float fR[_M];

   fluxphy(wL, vnorm, fL);
   fluxphy(wR, vnorm, fR);

   float UL[2] = {wL[1]/wL[0], wL[2]/wL[0]};
   float UR[2] = {wR[1]/wR[0], wR[2]/wR[0]};

   float cL = sqrt(_G*wL[0]);
   float cR = sqrt(_G*wR[0]);

   float lambdaL = cL;
   float lambdaR = cR;
   for (int i=0; i<2; i++){
      lambdaL += fabs(UL[i]*vnorm[i]);
      lambdaL += fabs(UR[i]*vnorm[i]);
   }

   float MyLAMBDA = lambdaL>lambdaR?lambdaL:lambdaR;
   for(int iv = 0; iv < _M; iv++)
      flux[iv] = 0.5f * (fL[iv] + fR[iv]) - 0.5f * MyLAMBDA * (wR[iv] - wL[iv]);
}
```

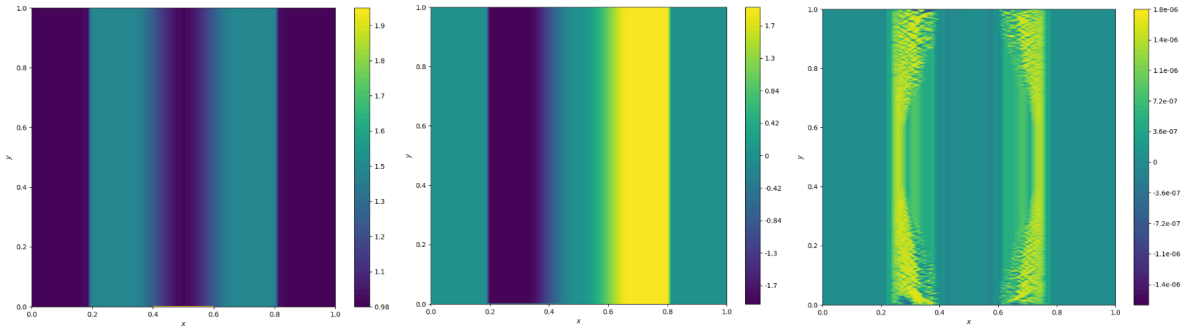
Listing 1 – *Flux numérique de Rusanov en 2D*

Quelque résultats sont présentés aux figures 1 à 3. Ils correspondent tous à une rupture totale de barrage à l'instant initial. La forme du barrage et le fond pourront varier d'une simulation à l'autre, mais toutes les simulations que nous traiterons dans ce rapport auront les paramètres suivants [3] :

- Domaine de taille $[0, L] \times [0, L] = [0, 1] \times [0, 1]$
- Nombre de mailles suivant x : $N_x = 1024$
- Nombre de mailles suivant y : $N_y = 1024$
- Temps maximal (final) de simulation : $t_{max} = 0.05$ *(sauf indication contraire)*.
- Coefficient $CFL = 0.8$. Il est choisi tel que $\Delta t = CFL \times \dfrac{\Delta x \Delta y}{2(\Delta x + \Delta y)\left(u_{max} + \sqrt{g h_{max}}\right)}$ à chaque itération (h_{max} et u_{max} désignant respectivement les valeurs maximales de la hauteur d'eau et de l'intensité du vecteur vitesse durant toute la simulation).
- La vitesse d'eau initiale sur tout le domaine $(u, v) = (0, 0)$
- La hauteur d'eau initiale à l'intérieur du barrage $h = 2$, et à l'extérieur du barrage $h = 1$.
- Le fond du domaine sera plat d'altitude nulle i.e $A(x, y) = 0 \;\; \forall \, (x, y) \in [0, L] \times [0, L]$ *(sauf indication contraire)*.

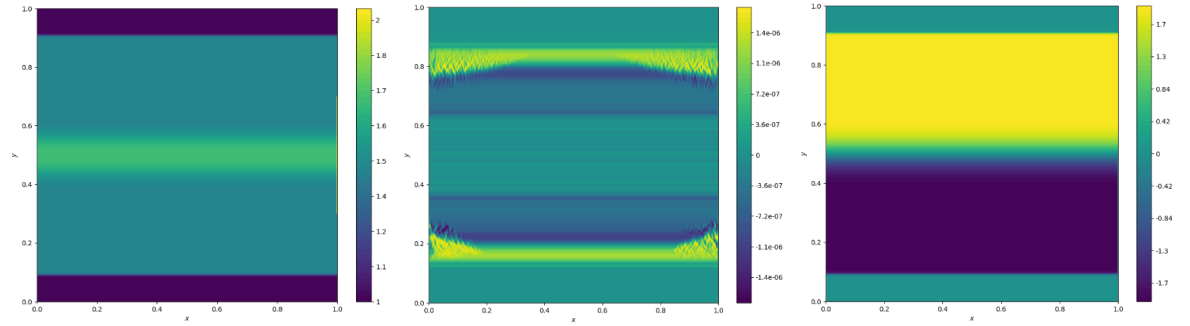
2. Remarquons que seules les 3 premières variables conservatives w_1, w_2, et w_3 (et donc 3 valeurs propres λ_1, λ_3, et λ_4) seront généralement considérées. Ceci correspondra généralement à un fond plat nul, i.e $A(\cdot, \cdot) = 0$.

3. Ces mêmes paramètres seront utilisés dans les autres sections du rapport.



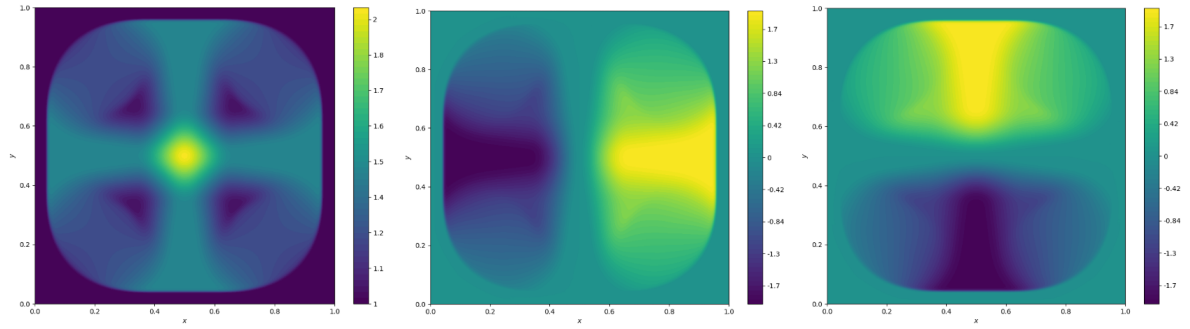
(a) *Hauteur d'eau* (b) *Vitesse suivant x* (c) *Vitesse suivant y*

FIGURE 1 – *Illustration de la résolution du système de Saint-Venant par le flux de Rusanov 2D. La condition initiale imposée est* $h = 2$ *pour* $0.4 \leq x \leq 0.6$.



(a) *Hauteur d'eau* (b) *Vitesse suivant x* (c) *Vitesse suivant y*

FIGURE 2 – *Illustration de la résolution du système de Saint-Venant par le flux de Rusanov 2D. La condition initiale imposée est* $h = 2$ *pour* $0.3 \leq y \leq 0.7$.



(a) *Hauteur d'eau* (b) *Vitesse suivant x* (c) *Vitesse suivant y*

FIGURE 3 – *Illustration de la résolution du système de Saint-Venant par le flux de Rusanov 2D. La condition initiale imposée est* $h = 2$ *sur* $[0.25, 0.75] \times [0.25, 0.75]$.

Ces figures nous montrent la propagation d'une onde de choc (une vague) vers les bords du domaine, et d'une onde de détente vers le centre du barrage. La figure 1a permet d'observer la vidange complète de la zone de rétention d'eau (intérieur du barrage), et la figure 2a capture le barrage en cours de la vidange [4]. Remarquons une vitesse suivant y (resp. suivant x) nulle pour la figure 1c (resp. pour la figure 2b). La figure 3 permet d'illustrer une propagation des ondes simultanément dans les deux directions de l'espace.

4. Remarquons que la forme du barrage est différente entre la figure 1 et la figure 2

Question 4.

Description de l'utilisation du solveur de Riemann 1D pour l'adaptation aux calculs en 2D.

Réponse

En 2D de façon générale, l'indice L désigne l'état du centre et R l'état voisin. Cependant, dans cette question, ils indiqueront les états de gauche, de droite, du bas, ou du haut du cellule en fonction de la direction du vecteur normal ; ceci afin de faciliter les explications.
Rappelons que dans le cas 1D, la valeur de h^* (et par suite celle de u^*) aux interfaces, i.e. entre un état gauche (h_L, u_L) et un état droit (h_R, u_R) est donnée par la solution du problème suivant :

$$u_L - (h^* - h_L)Z(h_L, h^*) = u_R + (h^* - h_R)Z(h_R, h^*)$$

ou la fonction $Z(\cdot, \cdot)$ de classe C^1 est donnée par

$$Z(a,b) = \begin{cases} \frac{2\sqrt{g}}{\sqrt{a}+\sqrt{b}} & \text{si} \quad b < a \\ \sqrt{g}\sqrt{\frac{a+b}{2ab}} & \text{si} \quad b > a \end{cases}$$

Cette équation résolue par la méthode de Newton permet de déterminer si on est présence d'une onde simple ou d'un choc ; et le solveur de Riemann 1D permet enfin de calculer l'état du système $w = (h, u)$ en un point x/t donné.
Pour adapter ce schéma en 2D, il suffit de remplacer la vitesse scalaire 1D u par la composante normale $U \cdot n$, où $n = n_{LR} = (n_1, n_2)$, du vecteur vitesse 2D. Il faudra donc résoudre l'équation ci-bas en utilisant le solveur de Riemann 1D :

$$U_L \cdot n - (h^* - h_L)Z(h_L, h^*) = U_R \cdot n + (h^* - h_R)Z(h_R, h^*)$$

Une fois la composante normale du vecteur vitesse $U^* \cdot n$ à l'interface obtenue, il nous manque sa composante tangentielle $U^* \cdot n'$, où $n' = n'_{LR} = (-n_2, n_1)$. On peut la prendre égale à la composante tangentielle de l'état de gauche, ou de celui de droite, ou l'un des deux en fonction d'un critère pertinent. Autrement dit,

$$U^* \cdot n' = \begin{cases} U_L \cdot n' & \text{si} \quad U^* \cdot n > 0 \\ U_R \cdot n' & \text{sinon} \end{cases}$$

Enfin, le vecteur vitesse à l'interface est obtenu en résolvant le système :

$$U^* = \begin{pmatrix} u^* \\ v^* \end{pmatrix} = (U^* \cdot n)n + (U^* \cdot n')n' = \begin{pmatrix} n_1 & -n_2 \\ n_2 & n_1 \end{pmatrix} \begin{pmatrix} U^* \cdot n \\ U^* \cdot n' \end{pmatrix}$$

Question 5.

Description de limplémentation des conditions aux limites suivantes : miroir, valeurs imposées, zone sèche.

Réponse

Dans cette partie, l'indice L désigne une cellule du bord du domaine, et l'indice R une cellule fantôme se trouvant à l'extérieur du domaine. Afin de faciliter les écritures, désignons par U^n et U^t les composantes normales et tangentielles respectives.

Zone miroir (cf. figure 4).

— la hauteur d'eau à la limite est prise égale à celle de la cellule proche du bord : $h_R = h_L$

— la composante normale du vecteur vitesse est inversée : $U_R^n = -U_L^n$

— la composante tangentielle du vecteur vitesse est maintenue : $U_R^t = U_L^t$

Au final, le vecteur vitesse sur le bord donne

$$\begin{aligned} U_R &= -U_L^n + U_L^t \\ &= -U_L^n + (U_L - U_L^n) \\ &= U_L - 2(U_L \cdot n)n \end{aligned}$$

Valeurs imposées (cf. figure 5).

— la hauteur d'eau h_R est donnée

— les composantes normales U_R^n et tangentielles U_R^t du vecteur vitesse sont données

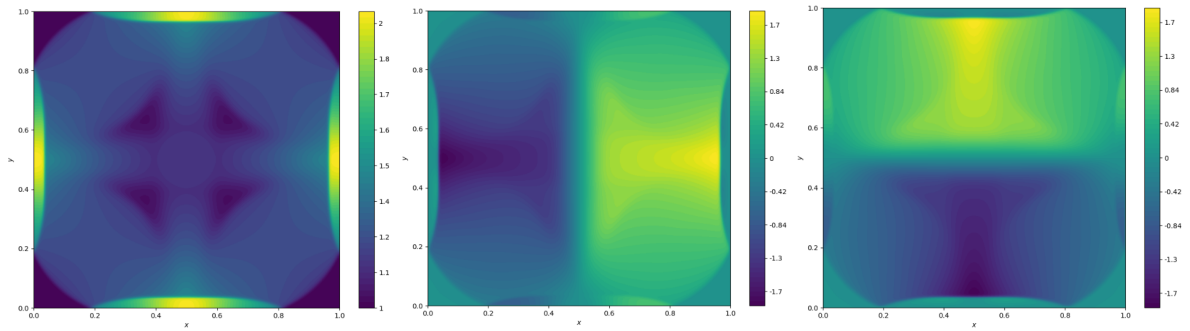
Si seulement le module du vecteur vitesse $\|U_R\|_2$ est donnée sur le bord, on pourra prendre $U_R^n = \|U_R\|_2 n$, et prendre $U_R^t = (0, 0)$.

Zone sèche (cf. figure 6).

— la hauteur d'eau $h_R = 0$ par définition. Cependant cette condition est difficile à implémenter. On prendra donc une hauteur suffisamment faible, par exemple $h_R = 0.05$.

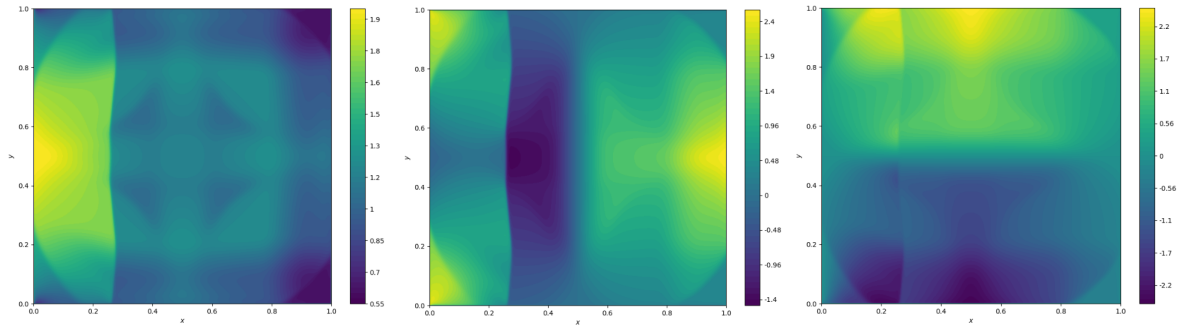
— pour $h_R = 0$, la vitesse n'est pas définie. On peut prendre $U_R = (0, 0)$ si on impose $h_R = 0.05$.

Illustrations. Observons à présents quelques résultats en appliquant ces conditions aux limites. La condition initiale est celle d'une rupture d'un barrage en forme de carré : $h = 2$ sur $[0.25, 0.75] \times [0.25, 0.75]$ et $h = 1$ ailleurs. Le temps final est $t_{max} = 0.7$ afin que les vagues atteignent les bords du domaine. Les problèmes sont résolus en utilisant le flux de Rusanov 2D.



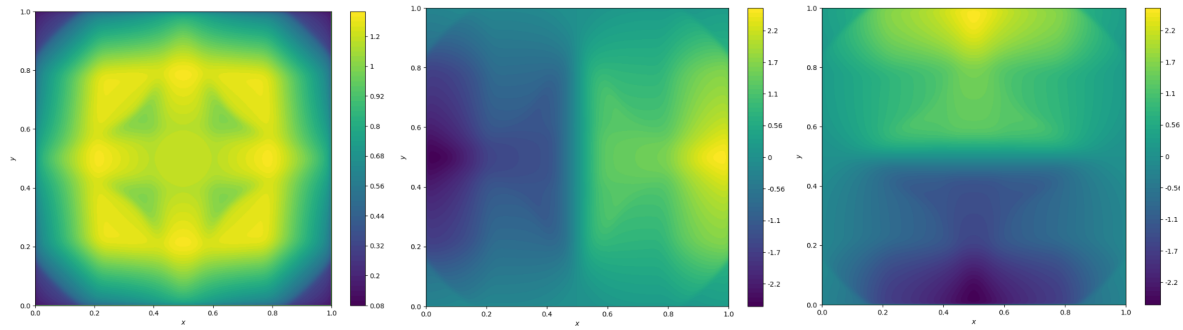
(a) *Hauteur d'eau* (b) *Vitesse suivant x* (c) *Vitesse suivant y*

FIGURE 4 – *Illustration des conditions aux limites de type miroir. Ce cas peu être assimilé à un bassin (ou une piscine, etc.) ayant des propriétés réflectives sur ses bords.*



(a) *Hauteur d'eau* (b) *Vitesse suivant* x (c) *Vitesse suivant* y

FIGURE 5 – *Illustration des conditions aux limites de type valeurs imposées. Ce cas peut être assimilé à un bassin contenant un barrage situé sur le lit d'un fleuve. En amont (sur la gauche), on applique la condition* $h_R = 1.5$, $(u_R, v_R) = (1.5, 0)$ *; en aval (et sur les deux autres bords), on applique* $h_R = 0.5$, $(u_R, v_R) = (0, 0)$. *On observe que le barrage est vite rempli et le fleuve continue son écoulement.*



(a) *Hauteur d'eau* (b) *Vitesse suivant* x (c) *Vitesse suivant* y

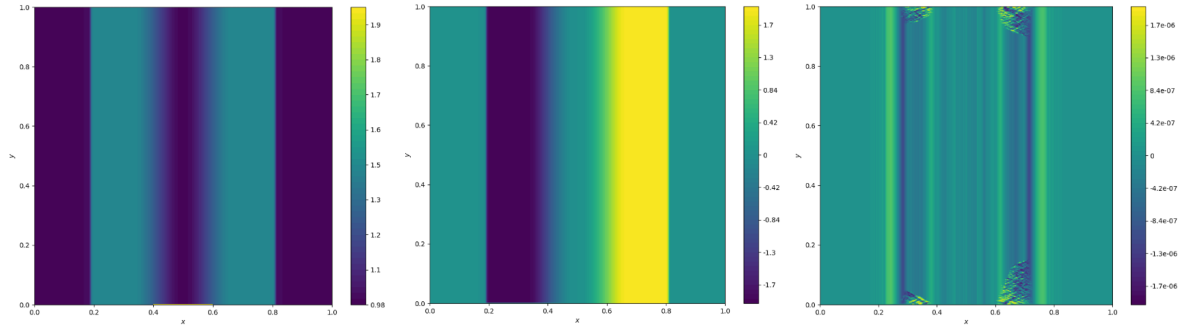
FIGURE 6 – *Illustration des conditions aux limites de type zone sèche. On prend* h_R *presque nul sur les bords du domaine, et une vitesse nulle i.e* $h_R = 0.05$, $(u_R, v_R) = (0, 0)$.

Question 6.

Mise en oeuvre de la résolution du système de Saint-Venant sur un maillage volume fini régulier. Vérifions la validité de la programmation en calculant d'abord des solutions de problème de Riemann dans la direction x puis dans la direction y avec un fond plat $(A = cste)$.

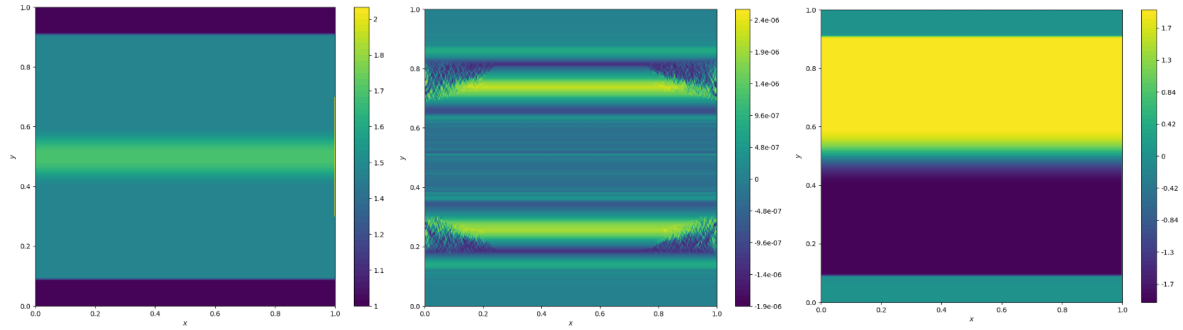
Réponse

Les modifications de test demandées ont été préalablement effectuées et implémentées aux figures 1 et 2 pour tester notre implémentation du flux de Rusanov 2D. À présent, reprenons les mêmes paramètres (ceux indiqués à la question 3.) pour vérifier notre programmation du solveur de Riemann 2D. Les fonds utilisés seront plats, et à l'altitude nulle i.e $A(\cdot, \cdot) \equiv 0$



(a) *Hauteur d'eau* (b) *Vitesse suivant x* (c) *Vitesse suivant y*

FIGURE 7 – *Illustration de la résolution du système de Saint-Venant par le solveur de Riemann 2D. La condition initiale imposée est $h = 2$ pour $0.4 \leq x \leq 0.6$, afin d'obtenir un calcul des solutions du problème de Riemann dans la direction x.*



(a) *Hauteur d'eau* (b) *Vitesse suivant x* (c) *Vitesse suivant y*

FIGURE 8 – *Illustration de la résolution du système de Saint-Venant par le solveur de Riemann 2D. La condition initiale imposée est $h = 2$ pour $0.3 \leq y \leq 0.7$, afin d'obtenir un calcul des solutions du problème de Riemann dans la direction y.*

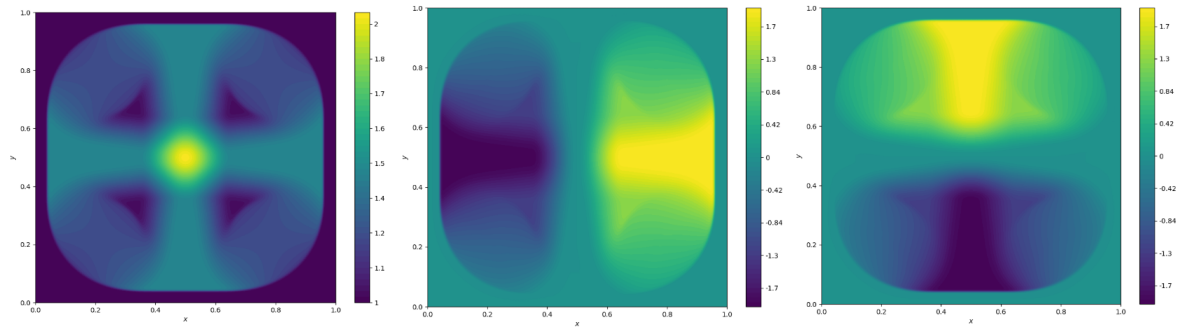
On observe aux figures 7b et 8c qu'effectivement, le calcul de solutions du problème de Riemann se fait dans la direction normale de propagation de l'onde (i.e x pour la figure 7b et y pour la figure 8c). Tout ceci implique des vitesses tangentielles nulles, ce que nous observons aux figures 7c et 8b.

Question 7.

Testons notre programme sur un cas 2D avec un fond plat.

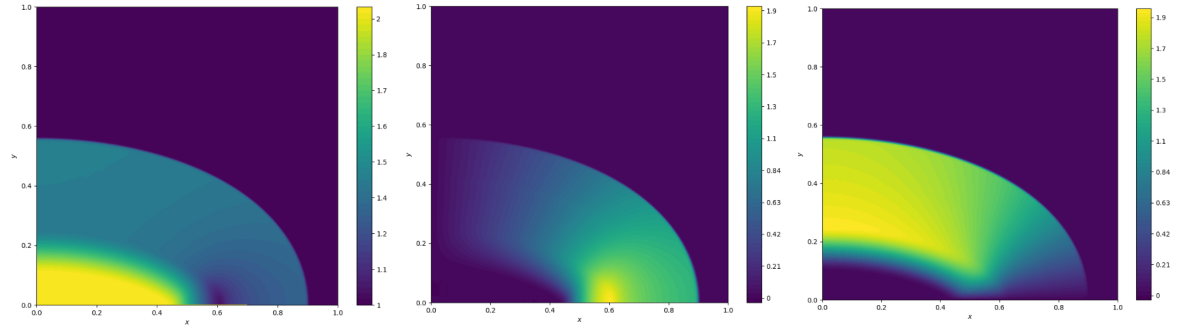
Réponse

Comme pour les cas précédents, le fond utilisés sera plat à l'altitude nulle i.e $A(\cdot, \cdot) \equiv 0$.



(a) *Hauteur d'eau* (b) *Vitesse suivant x* (c) *Vitesse suivant y*

FIGURE 9 – *Illustration de la résolution du système de Saint-Venant par le solveur de Riemann 2D. La condition initiale imposée est* $h = 2$ *sur* $[0.25, 0.75] \times [0.25, 0.75]$. *Cette figure correspond au même cas test que nous avons utilisé pour le flux de Rusanov 2D (cf. figure 3) ; il permet d'observer l'exactitude des résultats.*



(a) *Hauteur d'eau* (b) *Vitesse suivant x* (c) *Vitesse suivant y*

FIGURE 10 – *Illustration de la résolution du système de Saint-Venant par le solveur de Riemann 2D sur un cas non-symétrique. La condition initiale imposée est* $h = 2$ *à l intérieur de quadrant d'ellipse* $x^2/4 + y^2 \leq 0.35^2$. *Les conditions de bord qui sont appliquées sont celles de zones miroirs. Ce cas permet d'observer une dépression qui se forme autour de l'ellipse.*

Question 8.

Description et programmation la méthode MUSCL en 2D.

Réponse

Commençons par rappeler que le schéma de volumes finis en 2D s'écrit à l'itération n comme suit

$$w_L^{n+1} = w_L^n + \frac{\Delta t}{|L|} \sum_{R \in V(L)} |L/R| f(w_L^n, w_R^n, n_{LR}) \tag{1}$$

où

— L désigne les carrés (ou mailles, ou volumes finis) de quadratures
— $V(L)$ l'ensemble des voisins R de L
— $|L|$ la surface de L
— $|L/R|$ longueur de l'interface entre L et R

— la fonction f désigne le flux numérique [5]

L'idée de la méthode MUSCL est de remplacer le flux numérique calculé aux points w_L^n et w_R^n par un flux numérique calculé en des points plus proches de l'interfaces, et à un pas de temps correspondant. On notera ces points par w_L^* et w_R^*. On pose alors

$$\begin{aligned} w_L^* &= w_L^n + \frac{\Delta x}{2} s_{x_L} + \frac{\Delta y}{2} s_{y_L} + \frac{\Delta t}{2} r_L^n \\ w_R^* &= w_R^n - \frac{\Delta x}{2} s_{x_R} - \frac{\Delta y}{2} s_{y_R} + \frac{\Delta t}{2} r_R^n \end{aligned} \tag{2}$$

Pour le calcul des pentes en espace $s_{x_L}^n$ et $s_{y_L}^n$ pour une maille quelconque L, on distingue 4 possibilités pour les cellules du bord :

— w_{right} correspondant au voisin de droite
— w_{left} correspondant au voisin de gauche
— w_{up} correspondant au voisin du haut
— w_{down} correspondant au voisin bas

On pose ensuite

$$\alpha_x = \frac{w_L^n - w_{left}^n}{\Delta x}, \quad \beta_x = \frac{w_{right}^n - w_L^n}{\Delta x}, \quad \gamma_x = \frac{w_{right}^n - w_{left}^n}{2\Delta x}$$

De façon similaire, on pose

$$\alpha_y = \frac{w_L^n - w_{down}^n}{\Delta y}, \quad \beta_y = \frac{w_{up}^n - w_L^n}{\Delta y}, \quad \gamma_y = \frac{w_{up}^n - w_{down}^n}{2\Delta y}$$

On en déduit les pentes en espace

$$\begin{aligned} s_{x_L}^n &= \text{minmod}(\alpha_x, \beta_x, \gamma_x) \\ s_{y_L}^n &= \text{minmod}(\alpha_y, \beta_y, \gamma_y) \end{aligned}$$

où la fonction minmod [6] est donnée par :

$$\text{minmod}(\alpha, \beta, \gamma) = \begin{cases} \min(\alpha, \beta, \gamma) & \text{si } \alpha, \beta, \gamma > 0 \\ \max(\alpha, \beta, \gamma) & \text{si } \alpha, \beta, \gamma < 0 \\ 0 & \text{sinon} \end{cases}$$

Pour le calcul des pentes en temps, on se sert des matrices jacobiennes des composants du flux physique f_1 et f_2. D'après $(\mathcal{S}')$, on a

$$\partial_t w + \partial_x f_1(w) + \partial_y f_2(w) = S(w)$$

On prend donc la pente r_L^n telle que

$$r_L^n = -\frac{\partial f_1}{\partial w}(w_L^n) s_{x_L} - \frac{\partial f_2}{\partial w}(w_L^n) s_{y_L} + S(w)$$

Les pentes $s_{x_L}^n$, $s_{y_L}^n$ et r_L^n ayant été calculées, on calcule le flux numérique aux points w_L^* et w_R^* grâce à l'équation (2), qu'on applique au schéma numérique. L'équation (1) devient alors

$$w_L^{n+1} = w_L^n + \frac{\Delta t}{|L|} \sum_{R \in V(L)} |L/R| f(w_L^*, w_R^*, n_{LR}) \tag{3}$$

5. Il s'agira dans cas du flux de Rusanov dont on voudrait améliorer la précision
6. Notons que ces quantités sont des vecteurs, et donc la fonction minmod doit être appliquée composante par composante

En pratique, les pentes sont calculées indépendamment des états w, dans leur kernel propres. Le code de calcul est donné ci bas [7]. Les pentes $s^n_{x_L}$, $s^n_{y_L}$ et r^n_L y sont repérées respectivement par `dxwn`, `dywn`, et `dtwn`.

```
__kernel void muscl(__global const float *wn, __global float *dxwn, __global float *dywn, __global float *
    dtwn){
   int id = get_global_id(0);
   int i = id % _NX;
   int j = id / _NX;
   int ngrid = _NX * _NY;
   // Juste les cellules internes
   if (i > 0 && i < _NX - 1 && j > 0 && j < _NY - 1){
   double w[_M];
   for(int iv = 0; iv < _M; iv++){
      int imem = i + j * _NX + iv * ngrid;
      w[iv] = wn[imem];
   }
   double wR[_M];
   double wL[_M];
   double wU[_M];
   double wD[_M];
   for(int idir = 0; idir < 4; idir++){
      int iR = i + dir[idir][0];
      int jR = j + dir[idir][1];
      for(int iv = 0; iv < _M; iv++){
         int imem = iv * ngrid + iR + jR * _NX;
         if (idir == 0)
            wR[iv] = wn[imem];
         else if (idir == 1)
            wL[iv] = wn[imem];
         else if (idir == 2)
            wU[iv] = wn[imem];
         else if (idir == 3)
            wD[iv] = wn[imem];
      }
   }
   // Calcul des pentes en espace
   float dxLoc[_M];
   float dyLoc[_M];
   for(int iv = 0; iv < _M; iv++){
      int imem = i + j * _NX + iv * ngrid;
      // Cacul de dxwn
      double alpha = (w[iv] - wL[iv]) / _DX;
      double beta = (wR[iv] - w[iv]) / _DX;
      double gamma = (wR[iv] - wL[iv]) / (2*_DX);
      dxLoc[iv] = minmod(alpha, beta, gamma);
      dxwn[imem] = dxLoc[iv];

      // Cacul de dywn
      alpha = (w[iv] - wD[iv]) / _DY;
      beta = (wU[iv] - w[iv]) / _DY;
      gamma = (wU[iv] - wD[iv]) / (2*_DY);
      dyLoc[iv] = minmod(alpha, beta, gamma);
      dywn[imem] = dyLoc[iv];
   }
   // Cacul des pentes en temps
   double f1Prime[9];
   double f2Prime[9];
   fluxPrime(w, f1Prime, f2Prime);
   float dtLoc[3];
      for (int iLoc = 0; iLoc < 3; iLoc++){
         int imem = i + j * _NX + iLoc * ngrid;

         dtLoc[iLoc] = 0;
         for (int jLoc = 0; jLoc < 3; jLoc++){
            int index = iLoc*3 + jLoc;
            dtLoc[iLoc] -= (f1Prime[index]*dxLoc[jLoc] + f2Prime[index]*dyLoc[jLoc]);
         }

         dtwn[imem] = dtLoc[iLoc];
```

7. Les fonctions de calcul minmod et celles des matrices jacobiennes de f_1 et f_2, triviales, ne sont pas décrites dans ce rapport.

```
      }
   }
   // Pour gerer les pentes du bords
   else {
      for (int iv =0 ; iv < 3; iv++){
         int imem = i + j * _NX + iv * ngrid;
         dxwn[imem] = 0;
         dywn[imem] = 0;
         dtwn[imem] = 0;
      }
   }
}
```

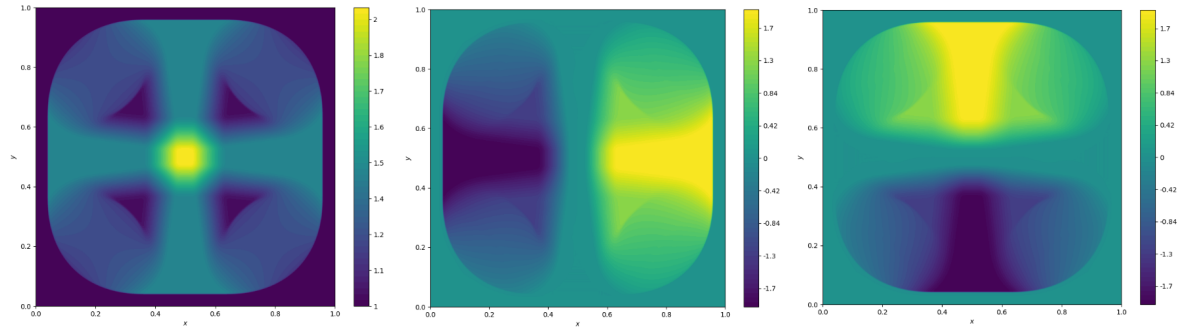
Listing 2 – *Kernel de calcul des pentes en espace et en temps pour la méthode MUSCL*

Les pentes ayant été calculées en amont, l'autre étape cruciale de l'implémentation de la méthode MUSCL est l'application des pentes $s^n_{x_L}$, $s^n_{y_L}$ et r^n_L susmentionnée, juste avant d'appeler la fonction de flux numérique à utiliser[8]. Elles sont repérées dans le code ci-bas respectivement par `six`, `siy`, et `ri`.

```
// Application des pentes en wL et wR
// vn désigne le vecteur normal orienté de L vers R
for(int iv = 0; iv < _M; iv++){
   // wnow indique l'état actuel de la cellule
   wL[iv] = wnow[iv] + vn[0]*six[iv]*(_DX/2.0) + vn[1]*siy[iv]*(_DY/2.0) + ri[iv]*(_DT/2.0);

   // wR indique l'état de la cellule R
   wR[iv] = wR[iv] - vn[0]*sixR[iv]*(_DX/2.0) - vn[1]*siyR[iv]*(_DY/2.0) + riR[iv]*(_DT/2.0);
}

// Application de MUSCL au flux de Rusanov 2D
flux_rusa_2d(wL, wR, vn, flux);
```

Listing 3 – *Application de la méthode MSUCL au flux de Rusanov*

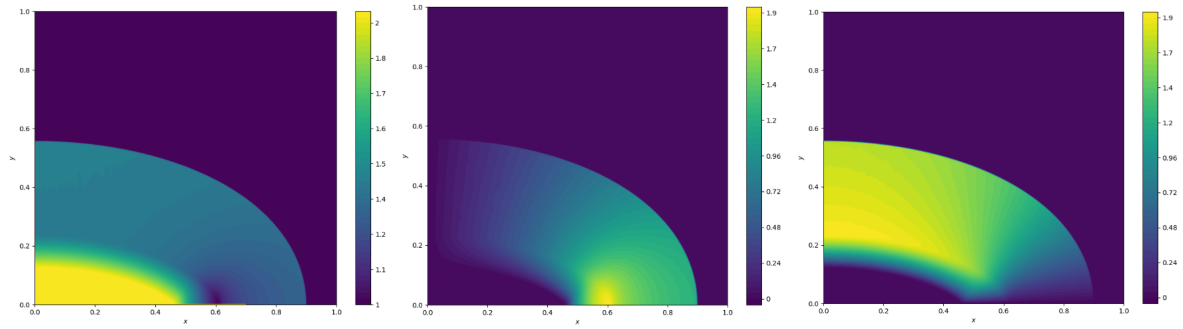
Nous testons la méthode sur deux cas étudiés aux questions précédentes.



(a) *Hauteur d'eau* (b) *Vitesse suivant x* (c) *Vitesse suivant y*

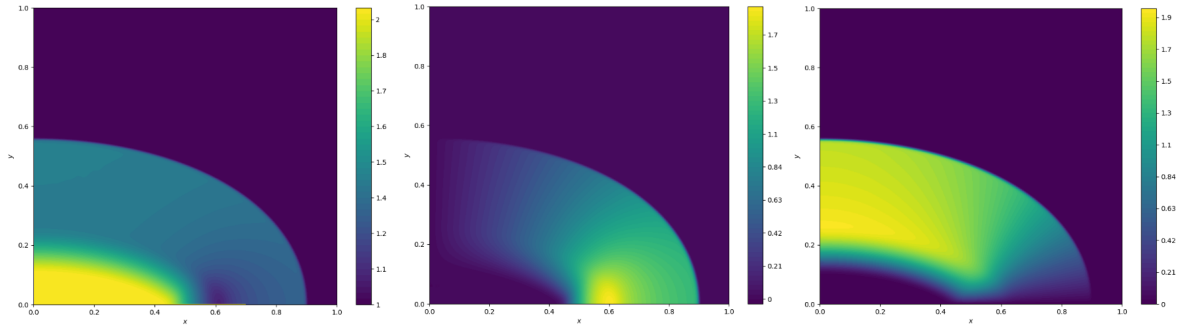
FIGURE 11 – *Illustration de la résolution du système de Saint-Venant par la correction MUSCL appliquée au flux de Rusanov 2D. Comparé à la figure 3, et en observant de près les hauteurs d'eau, on observe une nette amélioration.*

8. Nous utiliserons le flux de Rusanov car moins précis, et on aimerait qu'il ait une précision proche de la méthode de Godunov (solveur de Riemann 2D exact), tout en gardant sa rapidité (cf. T.P. #2). Remarquons qu'on aurait aussi bien pu appliquer la correction MUSCL au flux de Godunov en changeant simplement la dernière ligne de code au Listing 3. Les résultats obtenus avec le solveur de Riemann 2D exact sont donc a fortiori plus précis que ceux que nous présentons ici.



(a) *Hauteur d'eau* (b) *Vitesse suivant x* (c) *Vitesse suivant y*

FIGURE 12 – *Illustration de la résolution du système de Saint-Venant par la correction MUSCL appliquée au flux de Rusanov 2D sur un cas non-symétrique. Les conditions appliquées sont celles de la figure 10 ; en particulier la condition aux limites qui correspond à une zone miroir.*



(a) *Hauteur d'eau* (b) *Vitesse suivant x* (c) *Vitesse suivant y*

FIGURE 13 – *Illustration de la résolution du système de Saint-Venant par le flux de Rusanov 2D sans correction MUSCL. Une fois de plus, comparé à la figure 12, on observe une nette perte de précision, en particulier au niveau de la zone entourant le barrage.*

Question 9.

Description des modifications à apporter au code pour traiter un fond quelconque.

Réponse

Le flux considéré est un ajustement du flux de Rusanov f, noté g. On pose

$$g(w_L, w_R, n_{LR}) = f(w_L, w_R, n_{LR}) + \begin{pmatrix} 0 \\ \dfrac{g(h_L^2 - h_L^{*2})}{2} n_{LR} \end{pmatrix}$$

où

$$A^* = \max(A_L, A_R)$$

$$h_L^* = \max(0, h_L + A_L - A^*), \qquad h_R^* = \max(0, h_R + A_R - A^*)$$

$$u_L^* = \begin{cases} \frac{h_L u_L}{h_L^*} & \text{si } h_L^* > 0 \\ 0 & \text{si } h_L^* = 0 \end{cases} \qquad u_R^* = \begin{cases} \frac{h_R u_R}{h_R^*} & \text{si } h_R^* > 0 \\ 0 & \text{si } h_R^* = 0 \end{cases}$$

Le schéma étant défini, les modifications à apporter au code sont les suivantes :

- Définition de la carte géographique A

```
Anumpy = np.zeros((m * nx * ny, ), dtype=np_real)
Anumpy = np.reshape(Anumpy, (m, nx, ny))
for i in range(nx):
   for j in range(ny):
      Anumpy[:, i, j] = 2*(x[i]-0.5)**2 + 2*(y[j]-0.5)**2
Anumpy = np.reshape(Anumpy, (-1))
A = cl.Buffer(ctx, mf.READ_ONLY | mf.COPY_HOST_PTR,hostbuf=Anumpy)
```

Listing 4 – *Défintion d'un fond en forme de cuve*

- Ajout de la carte géographique A au kernel de calcul

```
event = prg.time_step(queue, (nx * ny, ), (64, ), wn_gpu, wnp1_gpu, six, siy, ri, A)
```

Listing 5 – *Ajout de la carte géographique au kernel de calcul*

- Définition d'un nouveau flux basé sur le flux de Rusanov

```
void flux_new_2d(float *wL, float *wR, float AL, float AR, float* vnorm, float* flux){
   float Astar = AL>AR?AL:AR;
   float hL = wL[0];
   float hR = wR[0];
   float hLstar = hL+AL-Astar>0?hL+AL-Astar:0;
   float hRstar = hR+AR-Astar>0?hR+AR-Astar:0;

   float UL[2] = {wL[1]/wL[0], wL[2]/wL[0]};
   float UR[2] = {wR[1]/wR[0], wR[2]/wR[0]};

   float uLStar[2];
   float uRStar[2];
   if (hLstar>0){
      uLStar[0] = hL*UL[0]/hLstar;
      uLStar[1] = hL*UL[1]/hLstar;
   } else{
      uLStar[0] = 0;
      uLStar[1] = 0;
   }
   if (hRstar>0){
      uRStar[0] = hR*UR[0]/hRstar;
      uRStar[1] = hR*UR[1]/hRstar;
   } else{
      uRStar[0] = 0;
      uRStar[1] = 0;
   }
   double wLStar[3] = {hLstar, hLstar*uLStar[0], hLstar*uLStar[1]};
   double wRStar[3] = {hRstar, hRstar*uRStar[0], hRstar*uRStar[1]};

   float fluxRusa[_M];
   flux_rusa_2d(wLStar, wRStar, vnorm, fluxRusa);

   flux[0] = fluxRusa[0];
   flux[1] = fluxRusa[1] + vnorm[0]*_G*(hL*hL - hLstar*hLstar)/2.0;
   flux[2] = fluxRusa[2] + vnorm[1]*_G*(hL*hL - hLstar*hLstar)/2.0;
}
```

Listing 6 – *Définition numérique du flux g*

- Application du nouveau flux

```
flux_new_2d(wnow, wR, AL, AR, vn, flux);
```

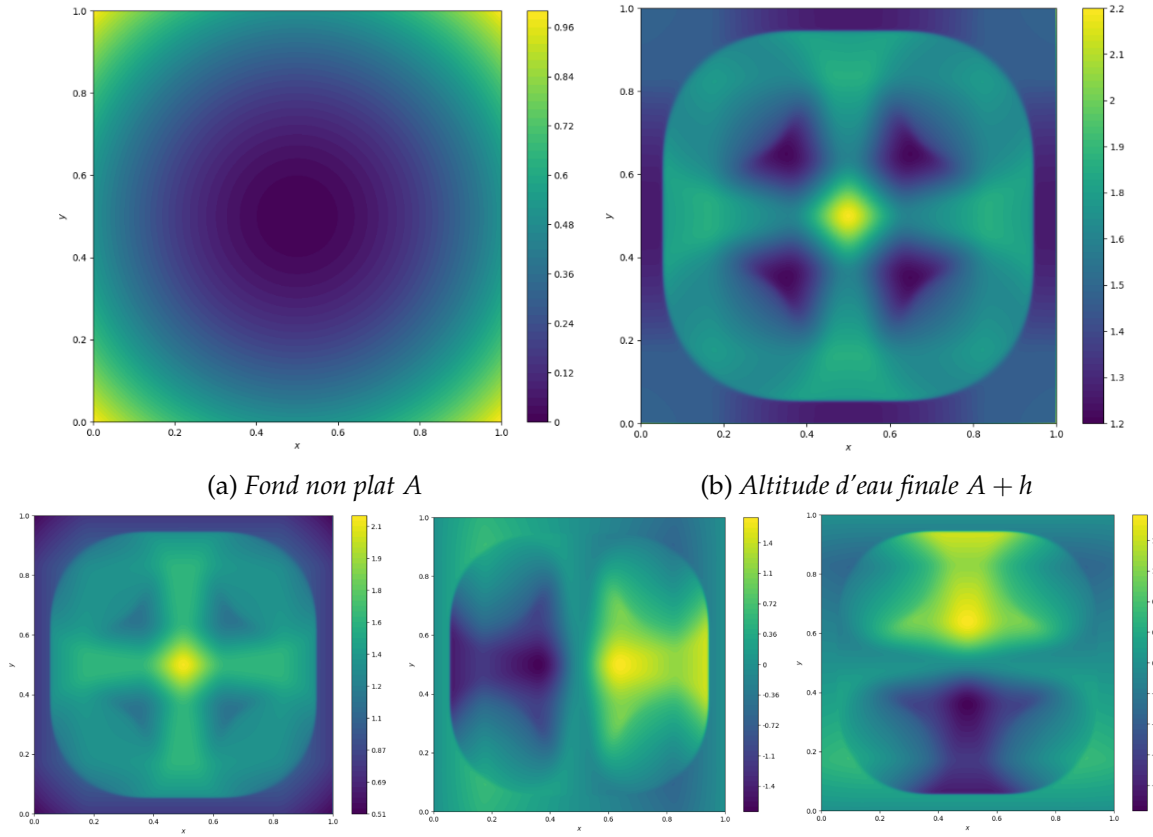
Listing 7 – *Application du nouveau flux*

Question 10.

Testons notre programme sur un cas réaliste avec fond non plat.

Réponse

Testons notre programme sur un cas familier, celui de la rupture de barrage en forme de carré (cf. figures 3, 9 et 11). Le fond non plat correspond à une paraboloïde, assimilable à une cuve.



(a) *Fond non plat* A (b) *Altitude d'eau finale* $A + h$

(c) *Hauteur d'eau finale* (d) *Vitesse suivant x finale* (e) *Vitesse suivant y finale*

FIGURE 14 – *Illustration de la résolution du système de Saint-Venant sur un fond non plat. Comparé aux figures 3, 9 et 11, on observe bien une tendance de l'eau sortie du barrage à y retourner.*