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]);
}
```

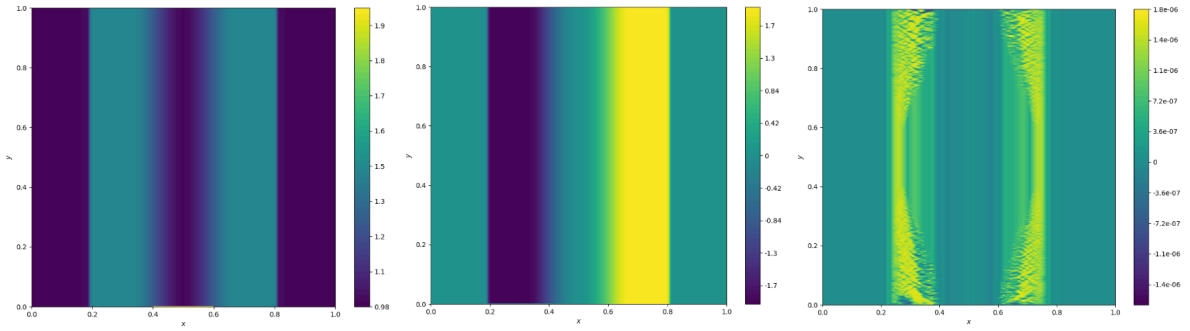
Listing 1 – *Flux numérique de Rusanov en 2D*

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)*.

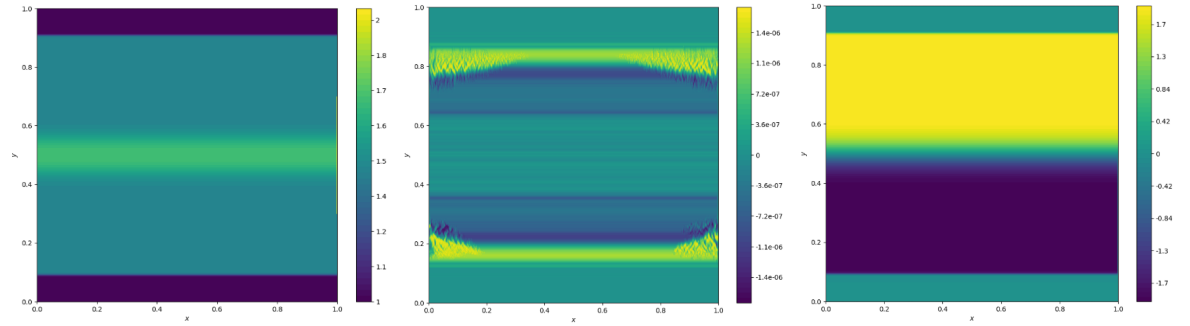
2. Remarquons que seules les 3 premières variables conservatives w_1, w_2, et w_3 (et donc 3 valeurs propres λ_1, λ_3, et λ_4) seront généralement considérées. Ceci correspondra généralement à un fond plat nul, i.e $A(\cdot, \cdot) = 0$.

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



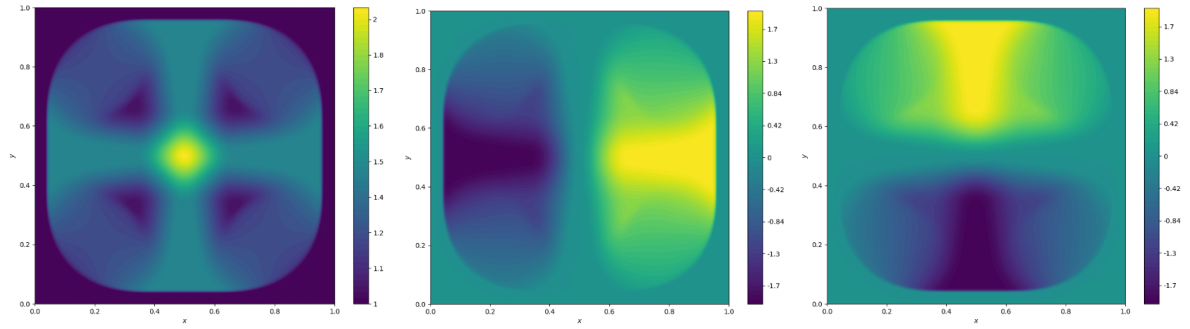
(a) *Hauteur d'eau* (b) *Vitesse suivant x* (c) *Vitesse suivant y*

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

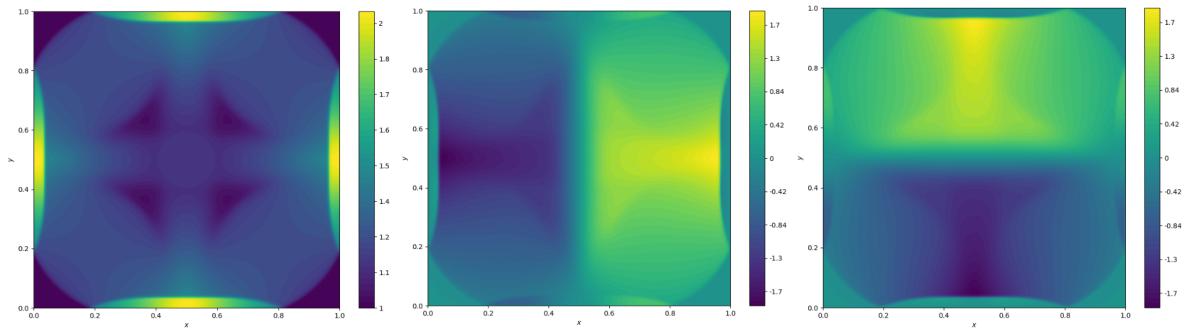
Si seulement le module du vecteur vitesse $\|U_R\|_2$ est donnée sur le bord, on pourra prendre $U_R^n = \|U_R\|_2 n$, et prendre $U_R^t = (0, 0)$.

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 $h_R = 0.05$.

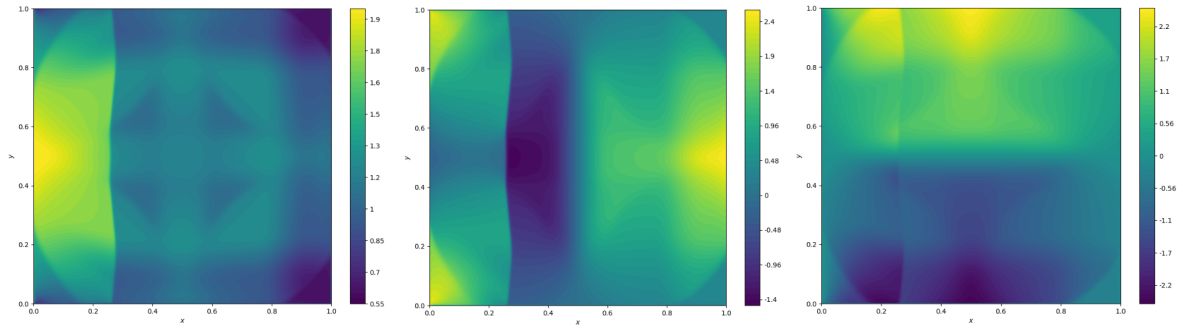
— pour $h_R = 0$, la vitesse n'est pas définie. On peut prendre $U_R = (0, 0)$ si on impose $h_R = 0.05$.

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.



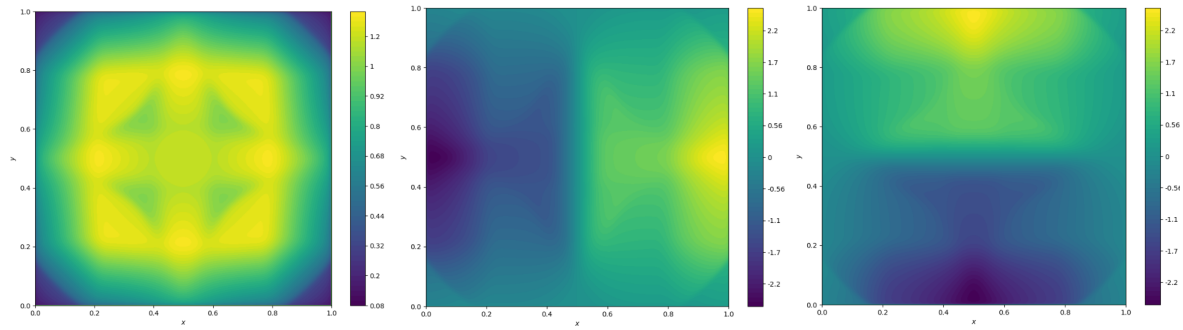
(a) *Hauteur d'eau* (b) *Vitesse suivant x* (c) *Vitesse suivant y*

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

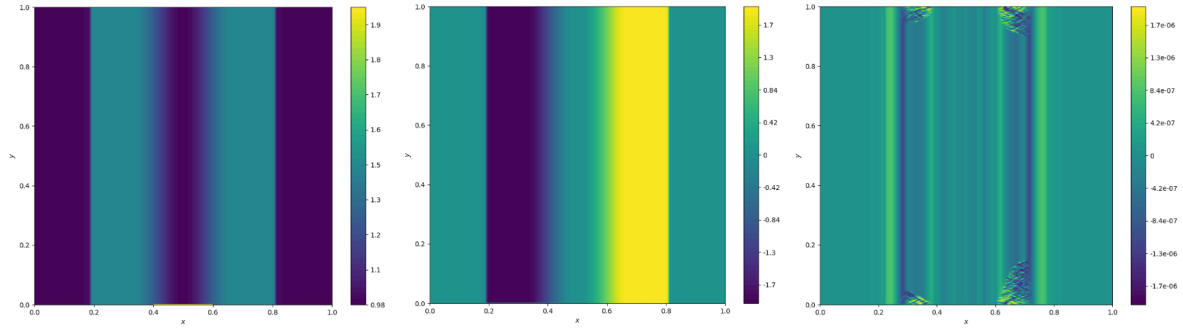
FIGURE 6 – *Illustration des conditions aux limites de type zone sèche. On prend* h_R *presque nul sur les bords du domaine, et une vitesse nulle i.e* $h_R = 0.05$, $(u_R, v_R) = (0, 0)$.

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 $(A = cste)$.

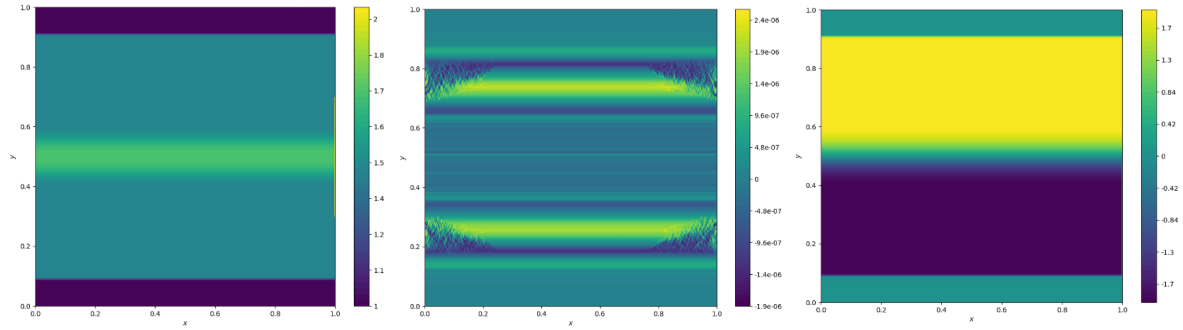
Réponse

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.*

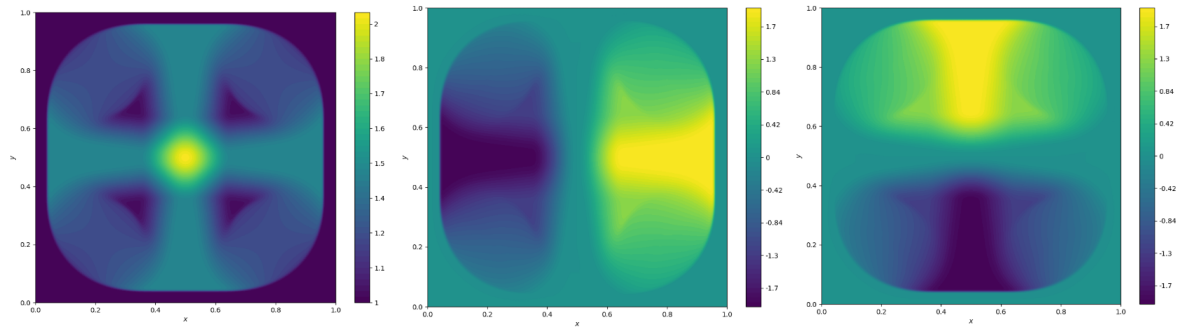
On observe aux figures 7b et 8c qu'effectivement, le calcul de solutions du problème de Riemann se fait dans la direction normale de propagation de l'onde (i.e x pour la figure 7b et y pour la figure 8c). Tout ceci implique des vitesses tangentielles nulles, ce que nous observons aux figures 7c et 8b.

Question 7.

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

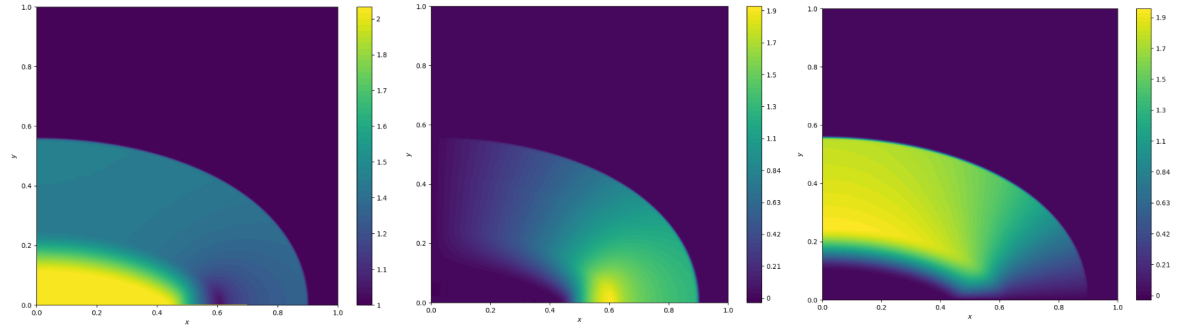
Réponse

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`.

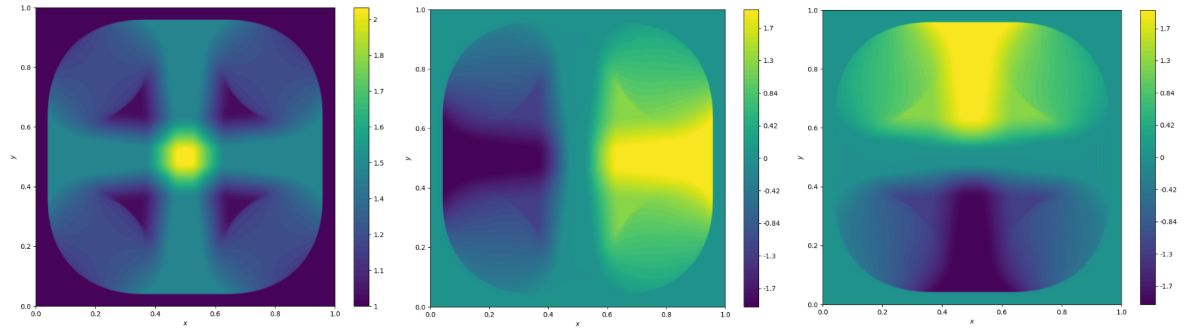
```
// Application des pentes en wL et wR
// vn désigne le vecteur normal orienté de L vers R
for(int iv = 0; iv < _M; iv++){
   // wnow indique l'état actuel de la cellule
   wL[iv] = wnow[iv] + vn[0]*six[iv]*(_DX/2.0) + vn[1]*siy[iv]*(_DY/2.0) + ri[iv]*(_DT/2.0);

   // 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*

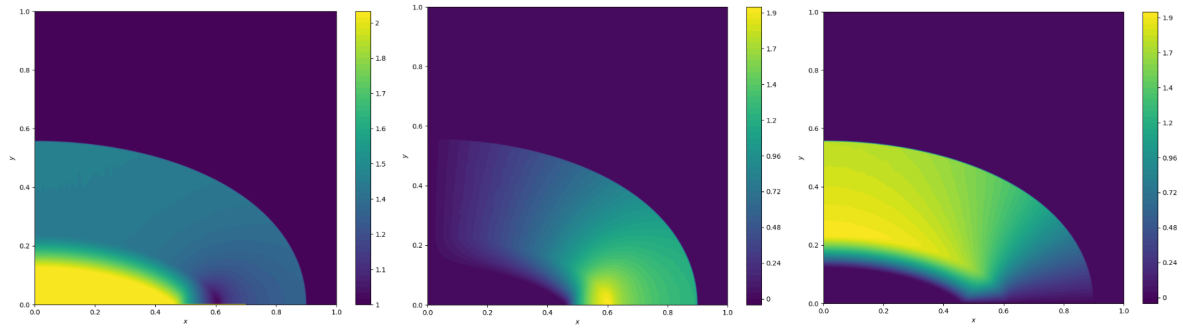
Nous testons la méthode sur deux cas étudiés aux questions précédentes.



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

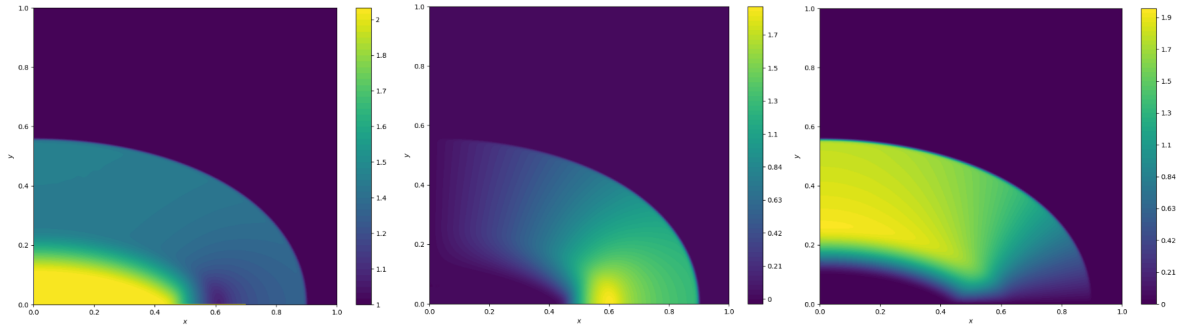
FIGURE 11 – *Illustration de la résolution du système de Saint-Venant par la correction MUSCL appliquée au flux de Rusanov 2D. Comparé à la figure 3, et en observant de près les hauteurs d'eau, on observe une nette amélioration.*

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);
```

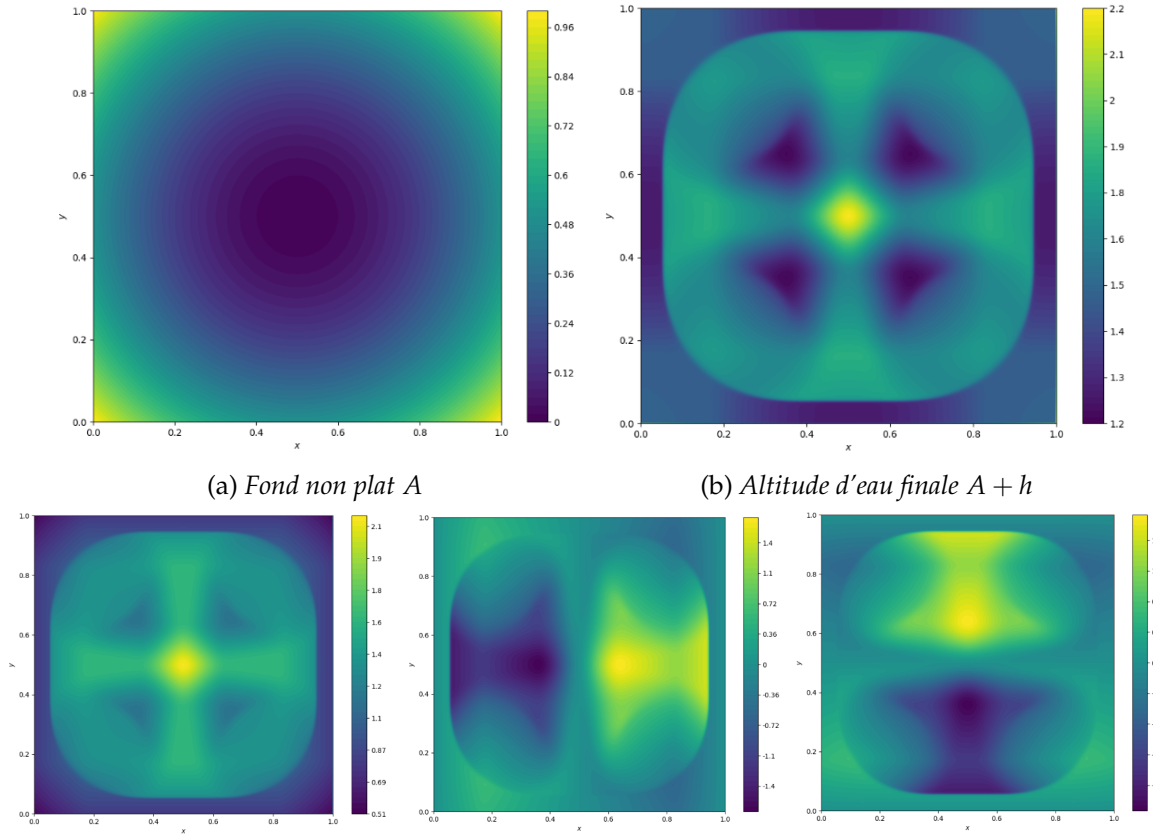
Listing 7 – *Application du nouveau flux*

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.*