ECOLE NATIONALE DES PONTS ET CHAUSSEES

École des Ponts
ParisTech

DEPARTEMENT GÉNIE MÉCANIQUE ET MATÉRIAUX - 2019/2020

SPH : Projet

Réalisé par :

Andrey LATYSHEV & **Siyuan HE**

1 Implémentation d'un schéma de diffusion de densité

Modèle de Molteni et Colagrossi:

$$\dot{\rho}_a = -\rho_a D_a\{\underline{v}_b\} + \nu_\rho L_a\{\rho_b\}$$
$$\nu_\rho = \delta\sigma c_0,$$

où $\sigma = \sqrt{5/18}h$ si on prends noyau "Wendland C^2 ($d = 2$)". c_0 est un constant déjà calculé. Ce que on doit ajouter dans le code est justement la terme laplacienne du ρ.

Car $\underline{\nabla} w_{ab} \propto \underline{r}_{ab}$ et $V_b = \frac{m}{\rho_b}$ on obtient

$$L_a\{\rho_b\} = 2\sum_b V_b(\rho_a - \rho_b)\frac{\underline{r}_{ab}\cdot\underline{\nabla} w_{ab}}{|\underline{r}_{ab}|^2} = 2\sum_b \frac{m}{\rho_b}(\rho_a - \rho_b)\frac{|\underline{\nabla} w_{ab}|}{|\underline{r}_{ab}|}$$
$$D_a\{\underline{v}_b\} = -\frac{1}{\rho_a}\sum_b m(\underline{v}_a - \underline{v}_b)\cdot\underline{\nabla} w_{ab} = -\frac{m}{\rho_a}\sum_b (\underline{v}_a - \underline{v}_b)\cdot\underline{r}_{ab}\frac{|\underline{\nabla} w_{ab}|}{|\underline{r}_{ab}|}$$
$$-\rho_a D_a\{\underline{v}_b\} + \nu_\rho L_a\{\rho_b\} = m\sum_b \left((\underline{v}_a - \underline{v}_b)\cdot\underline{r}_{ab} + 2\nu_\rho\frac{\rho_a - \rho_b}{\rho_b}\right)\frac{|\underline{\nabla} w_{ab}|}{|\underline{r}_{ab}|}$$

On a pris TD2 comme une base de notre projet. Pour d'activer et de désactiver la diffusion de densité, On définit un variable "DENSDIFFTYPE" en égalant 1 ou 0 du ligne 83 à 86 (nommé par " Density diffusion (PROJECT!!!)"). Et le paramètre δ est laissé libre à utilisateur du ligne 88 (nommé par "diff Parameters").

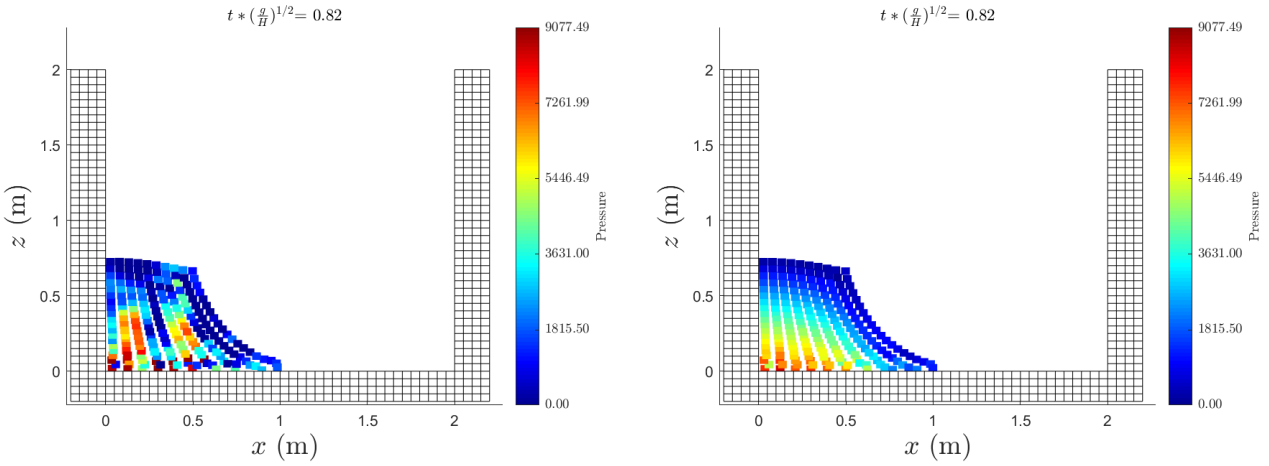
1.1 Impact sur le champs de pression

On a implémenté le schéma de diffusion de densité de Molteni et Colagrossi dans la fonction src\solver\DivergenceContribMolteniColagrossi.m :

```
function F = DivergenceContribMolteniColagrossi(m,dwdr,rVel,rPos,nu_rho,rho_i,rho_j)
rNorm = (rPos(:,1).*rPos(:,1)+rPos(:,2).*rPos(:,2)).^(.5);
veldotpos = rVel(:,1).*rPos(:,1)+rVel(:,2).*rPos(:,2);
F = m*veldotpos.*dwdr./rNorm + nu_rho*2*m*(rho_i-rho_j)./rho_j.*dwdr./rNorm;
end
```

Pour appliquer ce schéma on a changé la détermination de la densité dans la fichier src\solver\ComputeForces.m :

```
%CONTINUITY CONTRIBUTION
switch d
case{2}
sigma=sqrt(5/18)*h;
nu_rho=delta*sigma*c0;
case{3}
sigma=sqrt(4/15)*h;
nu_rho=delta*sigma*c0;
end
if DENSDIFFTYPE == COLAGROSSIDIFF
dRhodt =
DivergenceContribMolteniColagrossi(m,dwdr,rVelCont,rPos,nu_rho,rho_i,rho_j);
else
dRhodt = DivergenceContrib(m,dwdr,rVelCont,rPos);
end
```

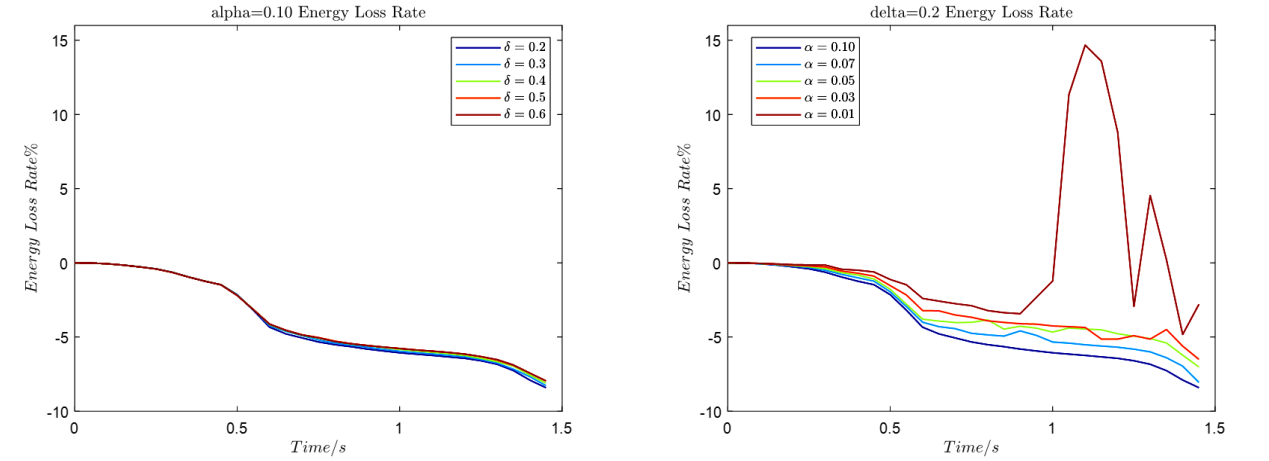


$\alpha = 0.1$ $\delta = 0.0$ $\alpha = 0.1$ $\delta = 0.2$

Figure 1: Champ de pression comparé à un instant avec et sans diffusion de densité.

On voit facilement le champs de pression varie d'une manière "glissante" par rapport à gauche où on désactive la diffusion de densité. Ce schéma nous permets de simuler un processus dans lequel l'impact de la pression à la densité n'est pas négligeable. Celui-ci évite évidemment la divergence de l'itération dans calculs.

1.2 Énergie mécanique en fonction du temps



$\alpha = 0.1$ δ varié α varié $\delta = 0.1$

Figure 2: Énergie mécanique en fonction du temps

Dans cette partie-là, on ajoute une variable "$EnergyMeca$" dans le module "$INTEGRATIONSTEP$". Dans le module "$savethemecanicenergy$",on enregistre l'énergie mécanique de cette système à certaine instant contrôlé par le variable "t_em" mis devant le début de l'itération. Pour mieux visualiser la variation de l'énergie mécanique, on calcule le taux de la perte d'énergie mécanique par rapport à l'état initial.

Premièrement la différence entre la courbe gauche et la courbe droite démontre le changement de δ ayant une perturbation faible sur la perte d'énergie. De l'image à droite, on vérifie d'abord que la limite inférieur de α égale 0.03. Si on le diminue encore, l'itération va diverger ultérieurement. Cette conclusion coïncide avec l'énoncé du projet.

Et puis, on remarque que l'augmentation de δ conduit à une perte plus petite, au contraire l'augmentation de α conduit à une perte plus grande. Cette phénomène nous inspire que la viscosité artificielle rend l'itération robuste avec un coût de la perte d'énergie. Mais la diffusion de densité ne change pas beaucoup l'énergie mécanique totale.

1.3 Pression interpolée à la position x=2m et z=0.2m

Pour trace la pression d'un certain point, il faut trouver le nombre de ce point. On utilise la fonction "find" pour fournir la critère "x==2m" et "z==0.2m".

```
[observe_point ,~]=find(abs(part(: ,POS(1))-2.0)<=dr/2);%x==2.0m
[observe_point_0 ,~]=find(abs(part(observe_point ,POS(2))-0.2)<=dr/2);%z==0.2m
observe_point=observe_point(observe_point_0);
```

la phrase "abs(part(:,POS(1))-2.0)¡=dr/2" nous permet de trouver le point qui se trouve plus proche de (2,0.2). "dr" est le rayon du noyau. Une fois on le trouve, on peut chercher la pression interpolé en utilisant la variable "part".

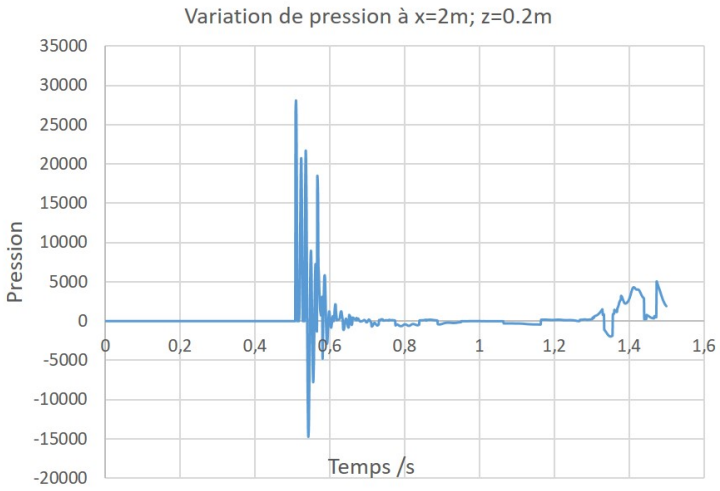
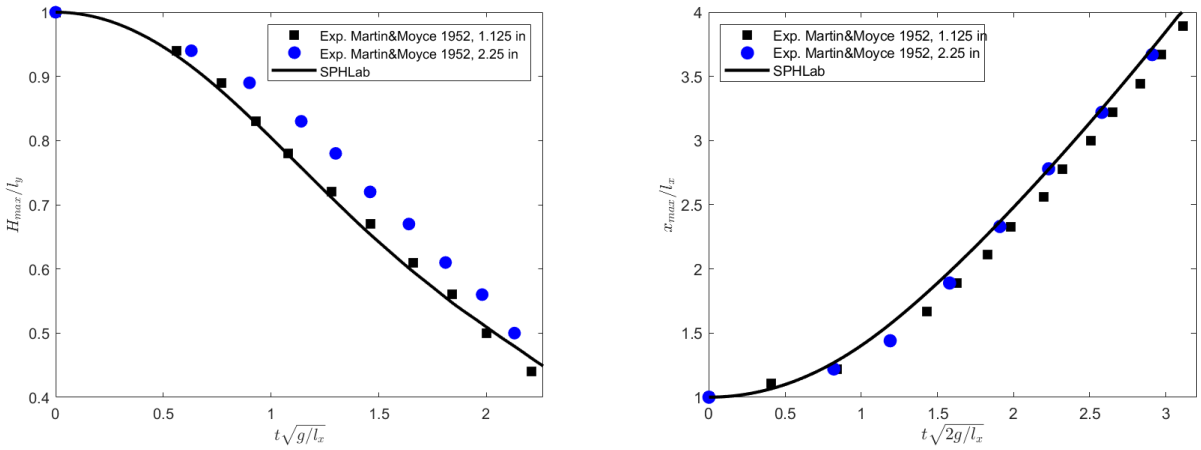


Figure 3: Pression interpolée à la position x=2m et z=0.2m

On trace la pression d'un point $z = 0.2m$, qui se trouve au mur à droite sur la base du barrage. Quand $t = 0.5\ 0.6$, la pression oscille brutalement et on voit une perte d'énergie évidente dans la Figure 2.



$\alpha = 0.1$ δ varié $\qquad$ α varié $\delta = 0.1$

Figure 4: Résultats expérimentaux et numériques

On compare nos résultats numériques avec les résultats expérimentaux donnés en séance TD2. On mets les paramètres $\alpha = 0.1$ et $\delta = 0.2$.

2 Détection de surface libre

Dans cette partie on va réaliser la méthode de détection de surface libre par Doring (2005). On a la matrice de renormalisation :

$$\underline{\underline{R}}_i^{-1} = -\sum_j V_j \underline{\nabla} w_{ij} \otimes \underline{r}_{ij}$$

Car $\underline{\nabla} w_{ij} \propto \underline{r}_{ij}$ et $V_j = \frac{m}{\rho_j}$ on obtient

$$\underline{\underline{R}}_i^{-1} = -\sum_j \frac{m}{\rho_j} \underline{\nabla} w_{ij} \otimes \underline{r}_{ij} = -m \sum_j \frac{|\underline{\nabla} w_{ij}|}{\rho_j |\underline{r}_{ij}|} \underline{r}_{ij} \otimes \underline{r}_{ij} = -mA \implies A = A^T$$

$$A = \sum_j \frac{|\underline{\nabla} w_{ij}|}{\rho_j |\underline{r}_{ij}|} \left[(r_{ij}^x)^2 \underline{e}_x \otimes \underline{e}_x + (r_{ij}^y)^2 \underline{e}_y \otimes \underline{e}_y + (r_{ij}^x r_{ij}^y)^2 (\underline{e}_x \otimes \underline{e}_y + \underline{e}_y \otimes \underline{e}_x) \right]$$

$$\lambda_{\min} = \min \lambda(\underline{\underline{R}}_i^{-1}) = \min\{-\frac{m}{2}(\mathrm{tr}A \pm \sqrt{\mathrm{tr}^2 - 4\det A})\} = -\frac{m}{2}(\mathrm{tr}A + \sqrt{\mathrm{tr}^2 - 4\det A})$$

Pour réaliser cette fonctionnalité on a implémenté des variables de type des particules et la fonction src\solver\particle_type.m, qui retourne une liste de deux éléments : la valeur propre minimale et le type de la particule considérée.

```
function particle_type = findParticleType(m,dwdr,rho_j,rPos)
    % R-1 = -m * A
    global LONELY_PARTICLE FREE_BOUNDARY_PARTICLE SURROUNDED_PARTICLE;

    rNorm = sqrt(rPos(:,1).*rPos(:,1)+rPos(:,2).*rPos(:,2));
    a11_ij = dwdr.*rPos(:,1).*rPos(:,1)./(rho_j.*rNorm);
    a12_ij = dwdr.*rPos(:,1).*rPos(:,2)./(rho_j.*rNorm);
    a22_ij = dwdr.*rPos(:,2).*rPos(:,2)./(rho_j.*rNorm);
    a11 = sum(a11_ij);
    a12 = sum(a12_ij);
    a22 = sum(a22_ij);
    trA = a11 + a22;
    detA = a11 * a22 - a12 * a12;
    lambda = -m * 0.5 * (trA + sqrt(trA * trA - 4 * detA));

    if lambda <= 0.2
        particle_type = [lambda LONELY_PARTICLE];
    elseif (lambda > 0.2) && (lambda <= 0.75 )
        particle_type = [lambda FREE_BOUNDARY_PARTICLE];
    else
        particle_type = [lambda SURROUNDED_PARTICLE];
end
```

Pour déterminer tous les types des particules on a réalisé une fonction src\solver\separateParticles.m, où il y a une boucle pour les particules FLUID de notre domaine. Cette fonction retourne trois listes : lambda_list contient toutes les valeurs propres minimales $\lambda_{\min}$, free_boundary_particles contient les indexes des particules appartenant à la surface libre et lonely_particles contient les indexes des particules, qui appartiennent à un jet ou sont isolées.

Pour appliquer la fonctionnalité réalisé dans le projet, on appel la fonction separateParticles dans le fichier principale main_Part4.m sur chaque itération du processus.

```
[lonely_particles free_boundary_particles lambda_list]
= separateParticles(part,space);
```

Pour s'assurer que l'algorithme détermine correctement la surface libre, on a changé un peu la fonction plotParticlesPressure. Désormais, les carrées rouges indiquent la surface libre.

```
% PLOT FREE BOUNDARY
N = size(free_boundary_particles);
for i = 1:N(2)
    k = free_boundary_particles(1,i);
```

```
        xtab = [part(k,POS(1))-dr/2 part(k,POS(1))+dr/2
            part(k,POS(1))+dr/2 part(k,POS(1))-dr/2];
        ytab = [part(k,POS(2))-dr/2 part(k,POS(2))-dr/2
            part(k,POS(2))+dr/2 part(k,POS(2))+dr/2];
        patch(xtab,ytab,'red');
    end
```

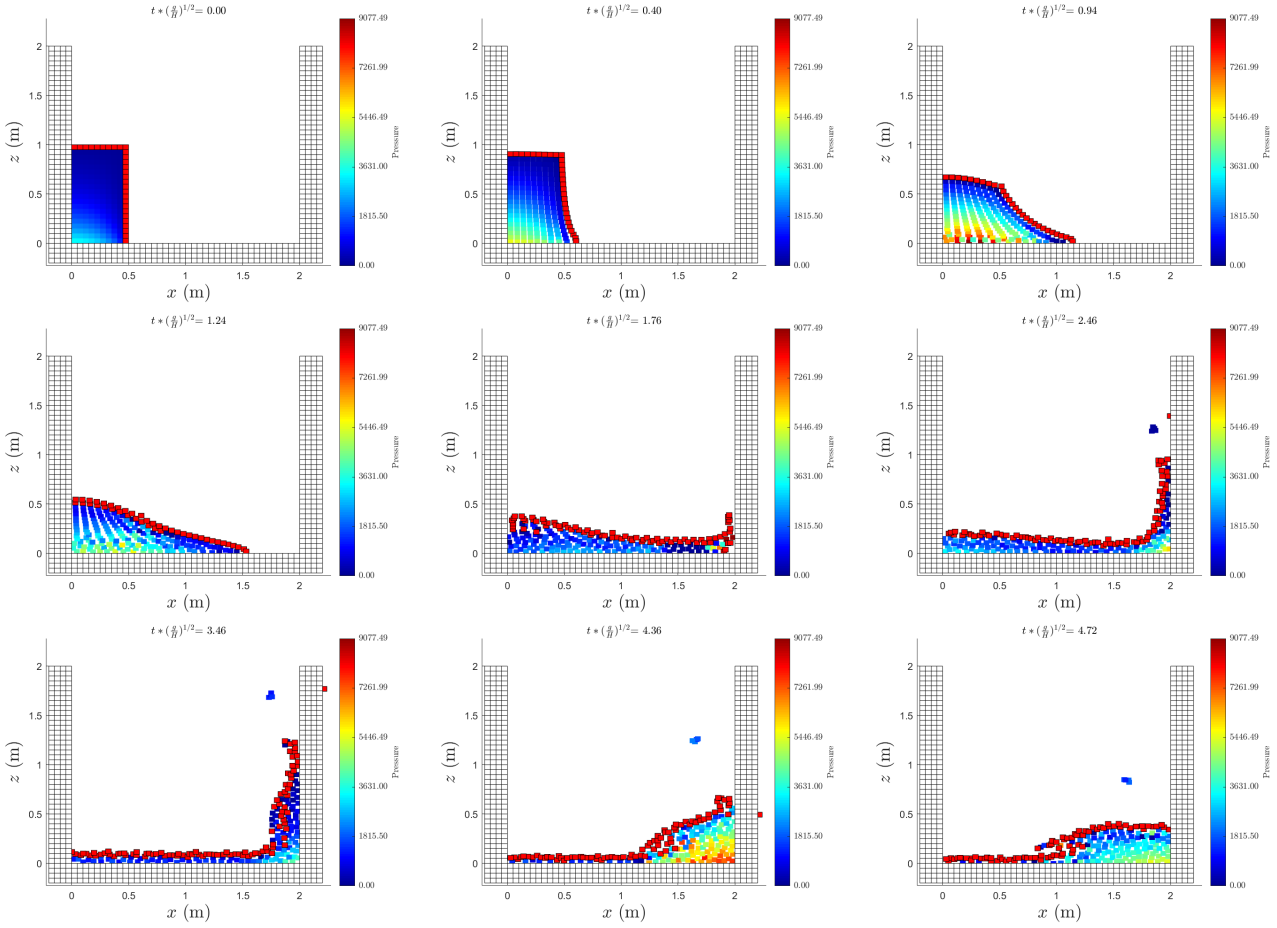


Figure 5: La démonstration de la surface libre pour les itérations différentes

Comme on voit l'algorithme fonctionne très bien sauf des certaines itérations. "La pénétration" des éléments de la surface libre puisse être liée avec l'inexactitude des calculs numériques et avec le petite nombre des particules. Cette méthode est suffisant pour approximer la surface libre.