MECA1120 : Projet 17-18

Gilles Peiffer
2432-1600

Thomas Reniers
2439-1600

Résumé

Dans le cadre du cours de MECA1120, il nous a été demandé d'écrire un petit programme d'éléments finis afin de résoudre un écoulement granulaire dans un milieu immergé. Basé sur les anciens devoirs, ce projet tente de donner une idée de la puissance et de l'importance de ces éléments finis dans le cadre des simulations numériques.

1 Inspirations et approfondissements

Ce projet combine deux grands concepts déjà abordés dans des précédents devoirs : une gestion d'éléments granulaires et la résolution d'une équation aux dérivées partielles. Dès lors, afin de gagner du temps déjà bien précieux, nous nous sommes grandement inspirés des devoirs 4 et 5, Grains et BandSolver, essentiellement pour les fichiers `fem.h` et `fem.c`. Le fichier `main.c`, basé sur celui de Grains, a été modifié afin d'y implémenter la résolution de l'EDP. Pour cette dernière, il est possible d'utiliser un solveur bande, ou itératif, l'itératif étant celui des gradients conjugués, qui est plus rapide et peut se faire en *matrix-free*, néanmoins nous ne savons pas imposer d'autres conditions que des homogènes à la frontière (en fait, en contraignant les matrices locales, il y a moyen de garder toutes ces belles promesses d'efficacité, mais par manque de temps nous n'avons pas fait cela).

Une bonne alternative est le solveur bande, auquel l'on impose des conditions non-homogènes sur la frontière extérieure (le terme v_{ext}), le terme source ayant disparu de l'équation, par rapport à celle de Poisson résolue lors des devoirs.

2 Fonctions supplémentaires

Dans ce projet, les grains ont une influence sur le liquide, et le liquide sur les grains, dès lors, il était utile de vérifier si un grain était dans un triangle du maillage de notre liquide, ce afin d'adapter l'EDP. Cette vérification se fait dans `femDiffusionCompute`, via une fonction que nous avons rajoutée : `ptInTriangle`, qui applique les coordonnées barycentriques.

Nous avons adapté le code de la fonction `femDiffusionCompute` pour pouvoir calculer autant en x qu'en y. Ensuite, pour trouver les conditions sur le bord du cylindre on impose une vitesse (différente dans le cas x que dans le cas y) qu'on trouve avec un petit peu de trigonométrie.

Nous avons également rajouté la fonction `vEval`, qui permet de faire l'isomorphisme sur le triangle parent et donc de trouver ainsi la vitesse quasi-exacte du fluide en un point (x, y) donné.

3 Pistes d'amélioration

Pour améliorer ce code, il y a diverses pistes sur lequelles on peut partir. Voici quelques exemples :

— contraindre les matrices locales lors de l'assemblage pour permettre d'utiliser un solveur itératif comme les gradients conjugués.

— L'algorithme de détection de contact n'a pas été amélioré. On pourrait trouver des moyens de faire beaucoup plus efficaces et moins lours au niveau du temps de calcul.

— L'algorithme de détection utilisé pour trouver dans quel triangle se situe un grain est relativement mauvais ; on pourrait par exemple stocker le triangle trouvé précédemment pour éviter de devoirs reparcourir tous les triangles, chaque fois qu'on doit recalculer les vitesses des grains.

— Le code ne tient pas compte de l'effet de la vitesse des grains sur le champ de vitesse du fluide. Ceci serait la plus importante parmi les pistes proposées, car elle constituait une partie de l'énoncé. Elle n'a pas été réalisée par manque de temps.