

INTRODUCTION AUX ÉLÉMENTS FINIS

MECA1120

---

## Projet 18-19 : note de synthèse

---

Groupe n°120

Amadéo DAVID - 44761700  
Nicolas BOUCHAT - 34881700

6 mai 2019

## 1 Obtention des équations en eaux peu profondes

## 2 Ordre de précision du résultat obtenu

CM8 slide 28 -> erreur =  $h^{p+1}$

## 3 Analyse de la solution

## 4 Optimisation du programme

L'optimisation d'un programme peut être divisée en deux parties. La première est l'optimisation algorithmique et la seconde est l'optimisation du code. Nous allons d'abord nous concentrer sur la première avant d'analyser la seconde.

### 4.1 Optimisation algorithmique

L'optimisation algorithmique a pour but de diminuer au maximum la complexité mathématique du système de résolution utilisé dans le programme. En pratique, cela se traduira par le choix de la méthode mathématique de résolution. Après décomposition du problème en différents sous-problèmes plus simple, il apparait que ces derniers sont proches de la matière vue au cours du quadrimestre, notamment lors des devoirs à rendre. Voici les différentes étapes de résolution pour lesquels nous avons opté.

1. Notre algorithme commence par déterminer les fonctions de forme d'un élément ainsi que son jacobien.
2. On applique ensuite une méthode la méthode de Hammer pour l'intégration des triangles et une méthode de Gauss-Legendre pour les arêtes. On interpole en fait les valeurs des vecteurs modèles afin de passer par les points d'intégration (ces-derniers sont donnés).
3. Commence ensuite le calcul des différents termes à ajouter. Ceux-ci sont donnés à la troisième page de l'énoncé sous forme d'équation.
4. Sur les bords, les termes d'intégration vont être ajoutés aux vecteurs  $E_i$ ,  $U_i$  et  $V_i$ .
5. On résout ensuite en multipliant chaque vecteur par son coefficient inverse puis en le divisant pas le jacobien. Cette méthode est appelée un solveur de Riemann.
6. L'algorithme se termine en appliquant Euler explicite.

### 4.2 Optimisation du code

Cette sous-partie est elle-même composée de différents domaines. Un code python peut en effet être optimisé de plusieurs manières. Il est possible d'utiliser des paquets compilés contenant des fonctions déjà codées et beaucoup plus efficaces que celle que nous pourrions écrire. Nous avons utilisé le paquet "numpy" qui est spécialisé dans la manipulation de matrices, vecteurs et polynômes. Nous n'avons cependant pas souhaité utiliser d'autres paquets supplémentaires car nous avons estimé que notre maîtrise de ces derniers n'était pas suffisante.

La partie la plus importante de l'optimisation du code réside dans la suppression des boucles. C'est principalement des modifications de ce type que nous avons faites. La plupart des boucles

présentes dans notre code ont été supprimées et remplacée par une vectorisation des éléments. La majorité des calculs itératifs ont été remplacés par des calculs matriciels beaucoup plus rapide. La seconde optimisation a été de sortir des boucles tous les éléments ne nécessitant pas d'être redéfinis à chaque itération. La dernière optimisation que nous avons mise en place a été de supprimer tous les calculs intermédiaires que nous avons alors fait sur papier pour les remplacer par des versions beaucoup plus condensées et simplifiées des calculs précédemment utilisés. Une optimisation minime mais à ne pas oublier a été de remplacer l'ensemble des éléments élevés au carré par une simple multiplication (par exemple en remplaçant  $x^2$  par  $x * x$ ). Nous avons en effet remarqué que python gérait plus rapidement ce second calcul que le premier. La différence est minime mais est tout de même observable sur un code conséquent.