Informatique Scientifique : PROJET

Olivier Tissot, Thomas Cherion

9 mai 2013

Dans ce rapport, vous trouverez les résultats de nos investigations concernant les tâches à effectuer sur les deux parties du projet. Le fonctionnement de notre programme est détaillé dans le readme, fournit avec les fichiers sources.

Nous tenons à préciser que pour l'utilisation de open GL (afin de réaliser les affichages des solutions) nous nous sommes beaucoup aidé de l'exemple que Xavier Claeys nous a donné en cours. C'est à dire en reprennant les fonctions qu'il avait donné puis en en modifiant quelques parties ou variables (entre autres sur ce qui concerne l'échelle de couleur et la fonction de couleur).

Partie I : Equation de Helmotz

I1) Nous avons choisi d'utiliser trois structures pour représenter les points, segments et triangles afin de gagner de l'espace mémoire. Pour la même raison, on a choisi de ne stocker que les numéros des points et le label dans les structures segments et triangles et non les coordonnées. Notre classe maillage contient aussi deux fonctions globales qui calculent l'aire d'un triangle et la longueur d'un segment.

I2) Etant donnée la structure creuse des différentes matrices à stocker nous avons choisi de le faire sous le format CSR et sous forme de vector. La classe vector est intéressante car elle fait partie de la bibliothèque STL ce qui permet une meilleure réutilisation du code et un accès mémoire en lecture rapides. Or lors de la résolution d'un système linéaire on va uniquement lire les coéfficients de la matrice. Et c'est cette phase qui sera la plus coûteuse en temps de calcul. Il est donc plus intéressant de gagner du temps sur la résolution du système linéaire que sur l'assemblage des matrices par exemple.
Malheureusement, le pendant de la classe vector est que l'accès mémoire en écriture n'est pas très rapide, notamment les fonctions resize() et insert(). Du coup, notre méthode set_val() est assez lente et particulièrement quand on fait set_val(0).
Tout ceci est bien illustré lors des tests des différents solveurs : LU et Jacobi font beaucoup appel à set_val, tandis que GC n'en fait aucun. Notre algorithme de pseudo-élimination est lui aussi caractéristique de la lenteur d'un set_val(0) puisqu'il est extrèmement lent. Afin de tester correctement les solveurs nous avons donc utilisé la bibliothèque d'algèbre linéaire gmm++. Le principe de la bibliothèque est d'implémenter des classes de matrices de format intermédiaire à CSR, de réaliser les calculs sur ces matrices puis de faire une copie de la matrice finale vers une matrice au format CSR qui sera fixe. Néanmoins, mettre en place cette stratégie nous paraissait particulièrement long puisque notre Gradient Conjugue était performant nous avons choisi de conserver notre propre classe.

I3) Nous avons créé un fichier (assemblage.cpp) où sont presentées toutes les routines d'assemblage. Cela nous semblait nécéssaire afin d'alléger notre main qui aurait été un peu volumineux. Nous avons commencé par implémenter la rigidité. Ainsi, nous pouvions construire un solveur de l'équation de Poisson. C'est pour cette raison que nous avons implémenté la pseudo-élimination, nous pouvions mettre un second membre qui correspondait à une source ponctuelle. Puis, nous avons assemblé les matrices de masse et de bord. Ensuite, nous avons utilisé les tests présents à la fin du cours pour vérifier nos matrices A et M. Nous ne détaillerons pas ici l'assemblage des matrice A, M et B car les algorithmes utilisés se trouvent dans le cours.
Nous allons plutôt détailler la formule utilisée pour l'assemblage du second membre. Rappelons que : $g_i = \int_{\partial\Omega} g\phi_i \,\mathrm{d}s$. Etant données les routines d'assemblage présentes dans le cours, on cherche en fait à calculer : $\int_{\Sigma_k} g\phi_{J_{k,i}} \,\mathrm{d}s$.

$$\begin{aligned}\int_{\Sigma_k} g(s)\phi_{J_{k,i}}(s)\,\mathrm{d}s &= |\Sigma_k| \int_0^1 g(\sigma_k(t))\widehat{\phi_j}(t)\,\mathrm{d}t \\ &= |\Sigma_k| \int_0^1 g((1-t)S_{J_{k,1}} + tS_{J_{k,2}})\widehat{\phi_j}(t)\,\mathrm{d}t \\ &\approx |\Sigma_k| \frac{g(S_{J_{k,1}})\widehat{\phi_j}(1) + g(S_{J_{k,2}})\widehat{\phi_j}(0)}{2}\end{aligned}$$

Et en plus, on sait que : $\widehat{\phi_1}(t) = 1-t$ et $\widehat{\phi_2}(t) = t$. D'où la formule présente dans notre code. Bien entendu, nous aurions pu utiliser une autre approximation pour l'intégrale mais celle-ci présente l'avantage d'être facilement implémentable puisque les $\widehat{\phi_j}(0)$ et $\widehat{\phi_j}(1)$ vont se simplifier : l'un des deux s'annulera.

T1) Nous avons calculé les approximations de u_h pour chaque triangulation, à l'aide du gradient conjugué, stoppé quand l'erreur est plus petite que 10^{-12} ou lorsque 1000 itérations sont atteintes. Des fichiers .dat des

résultats pour carre1.msh, carre2.msh, carre3.msh et carre4.msh ont été sauvegardés dans le sous-répertoire results puis le répertoire correspondant à la triangulation recherchée.

T2) Pour la triangulation la plus fine, c'est à dire carre4.msh et avec comme solveur itératif le gradient conjugué, on a la solution suivante :

FIGURE 1 – Helmotz carré 4

La solution est visualisée ici avec une erreur de 10^{-12} ou bien 1000 itérations (pour éviter des temps de calculs excessifs).

Taille du système : 12121
Temps de calcul : 11.08 sec
Nombre d'itérations : 1000
Erreur : 0.0327798

T3) Avant d'analyser mathématiquement l'erreur avec la solution exacte, nous avons voulu visualiser cette dernière afin de vérifier si à priori la solution que nous avions était la bonne. Avec le maillage le plus fin, on obtient :

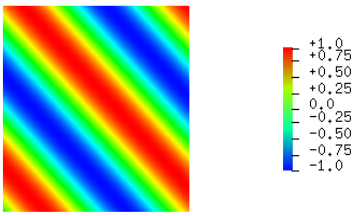


FIGURE 2 – Helmotz exact

Ce qui parait correspondre à la solution visualisée à la question T2).

Après avoir calculé numériquement l'erreur, nous estimons $\alpha = 1$. En effet, nous avons obtenu pour chaque triangulation, les affichages et les valeurs suivantes :

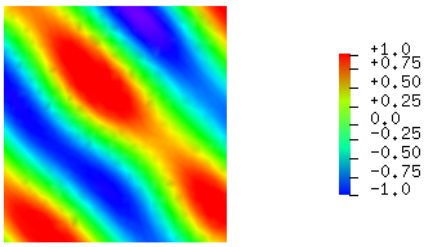


FIGURE 3 – Helmotz carré 1
erreur = 0.163526 - c = 2.561

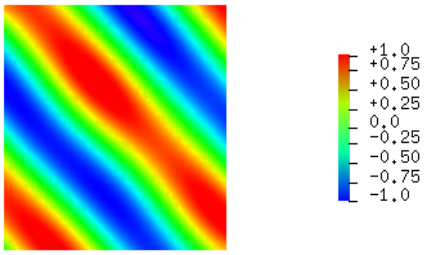


FIGURE 4 – Helmotz carré 2
erreur = 0.0683556 - c = 2.12138

FIGURE 5 – Helmotz carré 3
erreur = 0.0443883 - c = 1.91457

FIGURE 6 – Helmotz carré 4
erreur = 0.0327798 - c = 1.9383

Où c est une constante positive, indépendante de h telle que $erreur = c.h^{\alpha}$
On voit ici, que la valeur de c oscille autour de 2, cela nous amène à conclure que nous avons la bonne valeur pour α. En effet, avec α égal à 2 par exemple la valeur de c explose assez vite.
Ce résultat semble cohérent avec la formule donnée dans le cours : $||\Pi_h^1(u) - u_h||_{L^2(\Omega)} \leqslant c.h|u|_{2,\Omega}$ pour $u \in H^2(\Omega)$.
Mais, il faut garder à l'esprit que l'on a calculé une approximation de $||\Pi_h^1(u) - u_h||_{L^2(\Omega)}$, donc on ne peut pas appliquer directement l'inégalité ci-dessus.
Concernant l'erreur $||\Pi_h^1(\nabla u) - \nabla u_h||_{L^2(\Omega)}$ nous avons calculé ∇u et avons trouvé

$$\nabla u = \frac{10}{\sqrt{2}}(cos(\frac{10}{\sqrt{2}}(x+y)), cos(\frac{10}{\sqrt{2}}(x+y)))^T$$

Mais nous n'avons pas trouvé le moyen de calculer le ∇u_h nous n'avons donc pas répondu à cette question pour cette "erreur".

T4) La classe ctimes permet d'utiliser la fonction clock(), et nous avons donc pu mesurer et comparer les temps de calcul de chaque solveur. Pour la triangulation carré4.msh les temps de calcul sont les suivants :
temps de calcul avec la méthode du gradient conjugué : 11.08 sec
temps de calcul avec le solveur LU : 12974.8 sec (soit 3h36)
temps de calcul avec la méthode de Jacobi : XXX sec (nous avons pu constater que la méthode de Jacobi ne convergeait pas)
On remarque que la méthode du gradient conjugué est beaucoup plus rapide que LU ou Jacobi. Cela vient du fait que l'algorithme de base du gradient conjugué est plus optimisé que celui du solveur LU et Jacobi. Le solveur LU est efficace mais il convient de l'utiliser pour des résolutions de systèmes linéaires qui ne sont pas trop grands. De même, la méthode de Jacobi nécéssite un plus grand nombre d'opérations que celle du gradient conjugué. C'est d'ailleurs pour cela que nous avons choisi d'utiliser le gradient conjugué pour résoudre l'équation de Helmotz et ensuite exploiter les resultats plutôt que LU ou Jacobi.

Partie II : Equation des ondes

T1) Nous avons visualisé les solutions au temps final T=2.25 pour les deux maillages fournis, avec comme solveur le gradient conjugué (limité à 1000 itérations ou une précision de 10^{-12}) et les résultats sont les suivants :

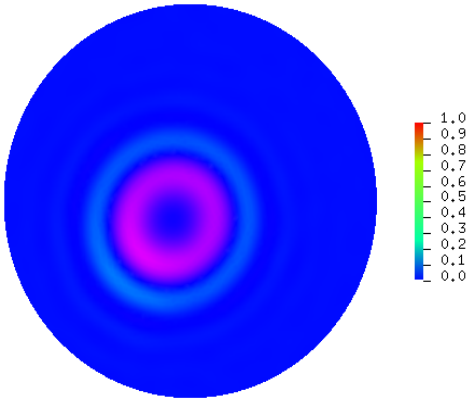


FIGURE 7 – équation des ondes cercle 1
$\Delta t = 0.005625$ - temps de calcul : 23.11 sec

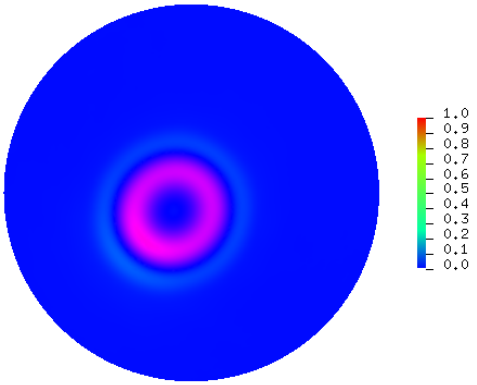
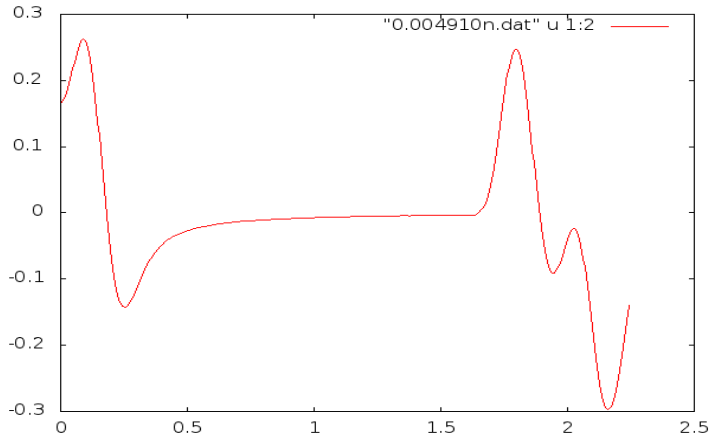


FIGURE 8 – équation des ondes cercle2
$\Delta t = 0.0049$ - temps de calcul : 366.96 sec

T2) Après différents tests CFL avec l'aire des triangles qui n'étaient pas concluants (Δt n'était pas assez proche des résultats obtenus pour pouvoir conclure à une CFL) ; nous avons testé les longueurs des segments. Nous avons finalement retenu comme CFL $\Delta t < h$ où h désigne la longueur minimale parmi tous les segments de la triangulation. Il faut noter que surement à cause des erreurs d'arrondis, on ne peut pas se placer à moins d'un centième de la CFL pour que le schéma soit stable.

T3) Ci-dessous, le graphique de u_h en le point d'origine au court du temps tracé à l'aide de gnuplot. Les valeurs ont été calculées pour la triangulation cercle 2 et une précision pour le gradient conjugué de 10^{-12} ou 1000 itérations.



T4) Grâce aux différents tests réalisés, nous avons pu constater que le schéma avec mass lumping améliorait le temps de calcul. Cependant, cela n'est pas forcément la solution idéale puisque cela rend le résultat moins précis.
Ci-dessous sont affichés les résultats de l'équation des ondes pour le maillage cercle1, avec le gradient conjugué pour une précision de 10^{-3} ou 250 iterations. Nous avons opté pour une moins bonne précision afin de ne pas avoir des temps de calcul trop grands, le but ici étant surtout de montrer la différence quand on utilise mass lumping.
Ici dt = 0.009.

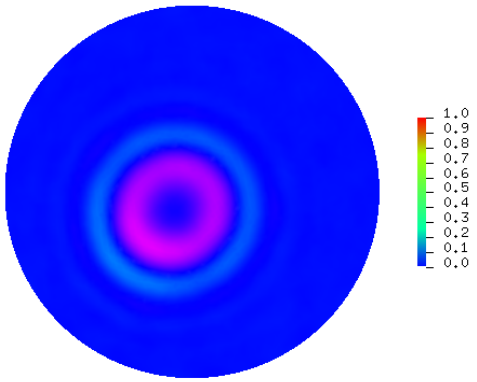


FIGURE 9 – équation des ondes cercle 1
sans mass lumping
temps de résolution : 14.14 sec

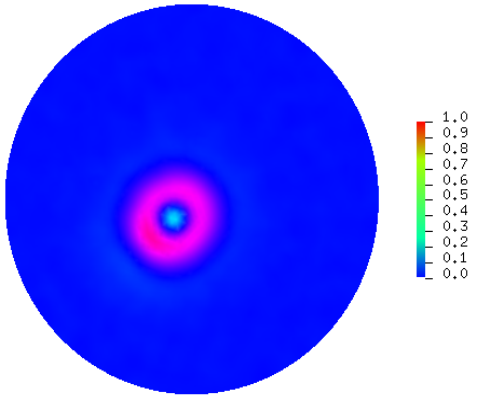


FIGURE 10 – équation des ondes cercle 1
avec mass lumping
temps de résolution : 7.44 sec

on constate aussi qu'avec mass lumping, la stabilité est augmentée. Avant la CFL était $\Delta t < \frac{1}{2}.h$ alors qu'avec mass lumping, on peut atteindre l'égalité et même avoir un pas Δt très légèrement plus grand que h.