Projet Informatique Scientique (MM031) 2011-12: Surface Minimale, v6

Frédéric Hecht

26 avril 2012

Projet2 : Il est possible de faire le projet en binôme, il est à rendre le dimanche 20 mai 2012, 24h00 par mail à Frederic.hecht@upmc.fr, (faire un archive . tar.gz, ou .zip des sources et rapport)

Projet1 Attention les modification par rapport à la version original sont en rouge, et de plus il est possible de faire le projet en binôme qui est à rendre le 28 février 2012 24h00 par mail à Frederic.hecht@upmc.fr

1 Généralités

Géométries On s'intéresse au calcul du problème de surface minimale et à sa visualisation.

Le problème de surface minimale est : soit Γ une courbe fermée de $\mathbb{R}^3$, le but est de trouver la surface S d'aire minimale de bord la courbe Γ.

Premièrement, nous supposerons que la surface est paramètré par F_u : $(x, y) \mapsto (x, y, u(x, y))$ et que Γ sera l'image du bord $\partial\Omega$ d'un domaine Ω de $\mathbb{R}^2$ par F_u, c'est à dire que u est connue sur $\partial\Omega$ et l'on écrira que $u = g$ sur $\partial\Omega$ où g est une fonction connue continue.

Maintenant notre problème peut s'écrire : Trouver la fonction u tel que

$$u = \underset{u_{|\partial\Omega}=g}{\texttt{arg min}}\ J(u) \tag{1}$$

où

$$J(u) = \int_\Omega \sqrt{1 + |\nabla u|^2} d\Omega \tag{2}$$

2 Projet 1

Dans un premier temps nous voulons étudiez ce problème en dimension 1, C'est a dire que l'on va supposer que u est invariant par toutes rotation de centre l'origine

Nous nous proposons d'écrire une méthode numérique pour résoudre le problème non-linéaire suivant :
Trouver u une fonction de $]a, b[\mapsto \mathbb{R}$, avec $0 < a < b$, qui vérifie les conditions aux limites (CL)

$$u(a) = \alpha, \qquad u(b) = \beta, \quad \text{(avec } \alpha \text{ et } \beta \text{ des valeurs données)},$$

et qui réalise **le minimum** de la fonctionnelle J suivante :

$$J(u) = 2\pi \int_a^b r \sqrt{1 + (u'(r))^2}\, dr. \tag{P}$$

On supposera que ce problème est bien posé dans $H^1(]a, b[)$, c'est à dire, il existe une unique solution u et qui ne dépend continûment des 4 données a, b, α, β.

Q0 Montrer que J est la surface (de révolution) définie (balayée) par la rotation de la courbe $z = u(r)$, pour $a < r < b$, autour de l'axe z, où (r, z) sont les coordonnées polaires, avec $r = \sqrt{x^2 + y^2}$.

Q1 Calculer la différentielle de Fréchet $DJ(u)$ de J en u ; $DJ(u)$ est une application linéaire de $H_0^1(]a, b[)$ dans $\mathbb{R}$.

Q2 Justifier pourquoi u est solution du problème suivant :

$$\forall v \in H_0^1((]a, b[), \quad 2\pi \int_a^b r \frac{u'v'}{\sqrt{1 + (u'(r))^2}} dr = 0$$

Q3 Montrer que la caténoïde $u(r) = acosh(r)$ (i.e. $cosh(u) = r$) est la solution du problème précédent pour $a = 1$, $\alpha = 0$, et $\beta = acosh(b)$.

Q4 On veut discrétiser le problème (P) avec des éléments finis P_1 Lagrange : soit N un entier strictement positif, on définit $h = (b-a)/N$, $q_i = a+$

hi , pour $i = 0, .., N$, les points du maillage et l'espace V_N des fonctions de $\mathcal{C}^0(]a, b[)$ affines par morceaux sur $]q_i, q_{i+1}[$ pour $i = 0, .., N-1$.
Soit w_i la base de V_N telle que $w_i(q_j) = \delta_{ij}$, où δ_{ij} est le symbole de Kroneker.
Soit une fonction v_N définie par ses valeurs aux nœuds du maillage

$$v_N = \sum_{i=0}^{N} v_i w_i, \quad \text{avec} \quad (v_i)_{i=0}^{N} \in \mathbb{R}^{N+1}.$$

Ecrire un algorithme formel qui calcule la fonctionnelle $J(v_N)$ en fonction des valeurs v_i.

Q6N On veut utiliser l'algorithme de minimisation à pas fixe qui s'écrit comme suit :
Soit u_N^0 un élément de V_N donné, vérifiant les conditions aux limites (CL) ;
on calcule récursivement

$$u_N^{n+1} = u_N^n - \rho \nabla J(u_N^n),$$

où ρ est une constante définie par l'utilisateur, et le gradient $\nabla J(u)$ est défini à partir d'un produit scalaire $(,)_N$ de V_N par :

$$(\nabla J(u), v)_N = DJ(u)(v).$$

. On prendra comme produit scalaire

1. Soit

$$(u, v)_N = \sum_{i=0}^{N} u_i v_i, \quad \text{où} \quad v = \sum_{i=0}^{N} v_i w_i, \quad \text{et} \quad u = \sum_{i=0}^{N} u_i w_i.$$

2. ou soit

$$(u, v)_H = \int_\Omega u' v'.$$

Evaluer les composantes g_i^n du gradient définies pour le produit scalaire $(,)_N$ par

$$\nabla J(u_N^n) = \sum_{i=0}^{N} g_i^n w_i,$$

comparer les par rapport $\partial J(u_N^n)/\partial u_i$, avec $u_N^n = \sum_{i=0}^{N} u_i w_i$. et écrire complètement l'algorithme composante par composante.

Q6H Comment calculer le gradient $\nabla J(u_N^n)$ définies pour le produit scalaire $(,)_H$.

Le but de cette partie est de programmer l'algorithme précédent.
On supposera que cette définition est active : `typedef double R;`

Q7 Ecrire une fonction C++ qui calcule l'aire d'un tronc de cône $c(r)$, de prototype `AireTroncDeCone(R ri, R ri1, R vi, R vi1) ;` défini pour ri $< r <$ ri1 ; $c(r)$ la fonction affine telle que $c(\text{ri}) = \text{vi}$ et $c(\text{ri1}) = \text{vi1}$.

Q8 Ecrire une fonction qui évalue J pour la fonction $v = \sum_{i=0}^{N} v_i w_i$, représentée par le vecteur v de ses valeurs, et qui a pour prototype de C++
`R J(R a, R b, int N, R *v) ; .`

Q9 Ecrire une fonction qui évalue ∇J pour la fonction $v = \sum_{i=0}^{N} v_i w_i$ et qui retourne le pointeur g qui est un argument qui va stocker le vecteur ∇J, et qui a pour prototype C++ `R * GJ(R a, R b,int N,R *v,R *g) ;`

Q10N Supposons que l'algorithme de la question Q6 est programmé avec la fonction suivante :

```
bool AlgoQ6N(R a,R b,R alpha, R beta, VN & un,R eps,
             R rho ,int itermax=10000);       //    ajoute de rho
{
   int N= un.N;
   VN gn(N),
   un(0) = alpha;
   un(N) = beta;
   for(int n=0;n<itermax;++n)
     {
        gn = GJ(a,b,N,un,gn);
         R err= gn.linfty();
         cout << " n = "<< n << " " << err<< endl;
         if(err  <eps) return true;
          gn *= rho;                          //    changement ICI
        un -= gn;
     }
    return false;
}
```

Ecrire une classe tableau VN qui dérive de `vector<double>` ainsi que les prototypes des méthodes et des fonctions associées qui seront

nécessaires pour la fonction `AlgoQ6` fonctionne. Tester et valider cette fonction sur l'exemple de la question Q3.

Afficher le solution calculé et la solution exact avec gnuplot.

Q10H Même question que Q6N mais avec le gradient calculer avec le produit scalaire $(,)_H$

Q10W Ecrire la méthode de Newton pour résoudre le problème non linéaire suivant : Trouver $u \in V_N$, tel que $u(a) = \alpha, u(b) = \beta$ tel que

$$\forall v_N \in V_N \cap H_0^1(]a,b[), \quad 2\pi \int_a^b r \frac{u'v'}{\sqrt{1+(u'(r))^2}} dr = 0$$

Q11 Comparer les résultats, et les temps de calcules entre les algorithmes des questions Q10N,Q10H,Q10W.

Q12 Faire une étude de convergence des algorithme en fonction de $N = 3, 10^5$.

3 Projet 2

Ce problème n'est pas simple théoriquement, mais si on le regarde d'un point de vue discret, il relativement simple.

Nous allons supposer que nous disposons d'un maillage $\mathcal{T}_h$ de notre domaine plan Ω, et soit $V_h = \{v \in H^1(\Omega)/\forall K \in \mathcal{T}_h; v_{|K} \in P1(K)\}$ soit $u_h \in V_h$, la surface facetisée que l'on minimise est $\sigma_{u_h} = F_{u_h}(\Omega)$ où $F_{u_h} : (x,y) \mapsto (x,y,u_h(x,y)$, pour que notre problème est un sens physique et ne dégénère pas trivialement,

u_h sera donne égal à $\Pi_h(g)$ sur Ω, où Π_h est l'interpolé dans V_h de la fonction g donné.

Donc notre problème est de trouver

$$\underset{u_h \in V_h, u_{h|\partial\Omega} = \Pi_h(g)}{\texttt{arg min}} J_1 = \sum_{K \in \mathcal{T}_h} |F_{u_h}(K)| \qquad (P_1)$$

Maintenant, on peut prendre une paramétrisation plus générale du type $U_h = (u_{1h}, u_{1h}, u_{1h}) \in V_h^3$ où l'on fait bouger les 3 composante du maillage

$$\underset{U_h \in V_h, U_{h|\partial\Omega} = \Pi_h(G)}{\texttt{arg min}} J_3 = \sum_{K \in \mathcal{T}_h} |U_h(K)| \qquad (P_3)$$

les domaines de calcul seront Ω_1 : une couronne entre $r_a = a$ et $r_b = b$, ou un rectangle $\Omega_2 =]x_0, x_1[\times]y_0, y_1[$.

De plus, parmi les conditions aux limites nous voulons étudier le cas où sur le bord du maillage la surface est définie par $x, y, \lambda sin(2\pi\, y)cos(\pi\, x)$ où λ est un paramètre.

Partie Freefem++

Q0 Petite recherche bibliographique afin de pouvoir valider votre projet.

Q1 Montrer que

$$J_1(u_h) = \int_\Omega \sqrt{1 + \nabla u.\nabla u} dx$$

et trouver $DJ_1(u_h)v_h$, implementer en FreeFem++ la méthode de gradient à pas simple pour minimiser $J_1(\Omega)$, valider avec le projet 1, sur Ω_1 en choisissant la fonction g et a et b pour des $h = 0.2, 0.1, 0.05$. De plus il faut choisir comme produit scalaire 1) le produit scalaire de $\mathbb{R}^n$, puis le produit scalaire de $H_0^1(\omega)$, $(u, v) = \int_\Omega \nabla u.\nabla v$.

Q2 Montrer que

$$J_3(u_h) = \int_\Omega ||\partial_x U \wedge \partial_y U|| dx$$

et trouver $DJ_3(u_h)v_h$, implementer en FreeFem++ la méthode de descente à pas simple pour minimiser $J_1(\Omega)$, valider avec $U = (ycos(x), ysin(x), x)$ sur $\Omega_2 = [0, 4\pi]\times]0, 4\pi[$ en choisissant la fonction . pour une suite de trois maillages raisonnables.

Q3 Utiliser le méthode `NLGC` de FreeFem++ avec ou sans preconditionnement pour minimiser J_1

Q4 Utiliser le méthode `NLGC` de FreeFem++ sans preconditionnement pour minimiser J_3

Q5 Ecrire la méthode Newton pour minimiser J_1 et J_3 en utilisant quelques itérations des algorithme précédent pour initialer la méthode Newton, s'il y a des problèmes de convergence.

Q6 Calculer numériquement en $\lambda = 1$,

$$\frac{\partial \, \texttt{arg min}_{u_h \in V_h, u_{h|\partial\Omega} = \Pi_h(g)} \, J_1}{\partial \lambda} \qquad (dJ1)$$

et

$$\frac{\partial \, \texttt{arg min}_{u_h \in V_h, u_{h|\partial\Omega} = \Pi_h(g)} \, J_3}{\partial \lambda} \qquad (dJ3)$$

pour la surface de bord définie par $(x, y, \lambda sin(2\pi\, y)cos(\pi\, x))$, sur un maillage du carré unité 20×20.

Q7 Un utilisant les classes du cours, implementer en `C++` dans un programme qui est dynamiquement paramétrable, via `argc` et `argv`, les paramètres du `shell` passer a la fonction `main`, c'est-à-dire `int main(int argc, const c` les paramètres seront le maillage, puis l'algorithme :

1. Algorithme de Gradient à pas fixe non précondionné pour J_1
2. Algorithme de Gradient à pas fixe précondionné pour J_1
3. Algorithmes de Gradient à pas fixe non précondionné J_3
4. Algorithme de Gradient à pas fixe non précondionné pour J_1
5. Algorithme de Newton pour J_1
6. Algorithme de Newton pour J_3

plus des paramètres telle que : choix de rho, nombre itération, test d'arrêt, choix des conditions aux limites.

Q8 Ajouter le calcul de $(dJ1)$ et $(dJ3)$ par différences finesi et par différentiation automatique.

Q9 Mettre ces algorithmes dans les classes OG du dossier (`http://www.ann.jussieu.fr/~hecht/ftp/InfoSci/OpenGL/`) telle que les itérations soient faite avec des timer GLUT, et où les paramètres sont changés dynamiquement dans des menus.

Q10 Pour finir, une petite interface java graphique du programme