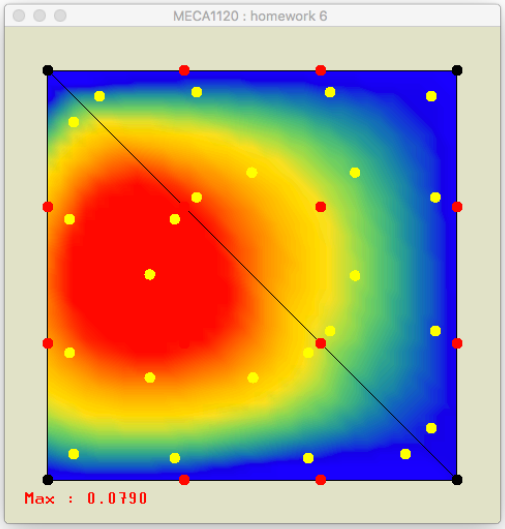
Finite elements for dummies : Approximation cubique continue :-)



Ici, nous allons effectuer simplement le calcul d'un approximation cubique continue $u^h(x,y)$ au sens de moindres carrés d'une solution analytique connue $u(x,y)$.

Il faut donc trouver les 10 valeurs nodales sur chaque triangle du maillage : dans l'exemple de la figure, on peut observer qu'il y a deux éléments et un ensemble de 16 valeurs nodales distinctes car les deux éléments partagent 4 valeurs nodales communes sur la diagonale. Les fonctions de forme seront donc ici des polynômes de troisième degré.

Trouver $u^h(x,y) = \sum_{i=1}^{n} U_i \ \tau_i(x,y)$ sur Ω tel que

$$\sum_{i=1}^{n} U_i \underbrace{\int_\Omega \tau_i(x,y)\tau_j(x,y)\ d\Omega}_{A_{ij}} = \underbrace{\int_\Omega u(x,y)\tau_j(x,y)\ d\Omega}_{B_j} \qquad j = 1 \ldots n$$

Pour effectuer les intégrations numériques, nous utiliserons maintenant une règle de Hammer à 12 points comme illustré ci-dessous. C'est pourquoi, nous avons ajouté la règle de Hammer à douze points dans le module `fem`. Soyez rassurés : il ne faudra pas encoder cette nouvelle règle d'intégration.

Comme les fonctions ne sont plus du tout linéaires sur un triangle, il a été nécessaire de découper les triangles pour obtenir une figure satisfaisante de la solution numérique, car `OpenGL` ne travaille qu'avec des facettes linéaires. Typiquement, nous divisons chaque élément de manière récursive pour pouvoir représenter la solution discrète. Utiliser quatre niveaux de récursion permet d'obtenir une figure assez correcte pour la solution discrète. Mais pour obtenir une jolie figure de la solution analytique, il est requis de faire huit niveaux de réclusion sur le maillage avec deux éléments. Il est possible d'inspecter le code fourni pour avoir une petite idée de la manière dont le dessin est réalisé.

Numérotation des variables pour une interpolation quadratique ou cubique continue...

Pour des interpolations quadratiques et cubiques, il y a davantage de valeurs nodales que de sommets du maillage, il faudra introduire une nouvelle numérotation des variables comme suit :

- On numérote tout d'abord tous les sommets.
- On numérote ensuite les noeuds associés aux arêtes Typiquement, on ajoutera une inconnue par segment pour une interpolation quadratique et deux inconnues pour une interpolation cubique. Les noeuds seront distribués de manière uniforme le long du segment. Attention, l'ordre de ces inconnues n'est pas innocent lorsqu'on ajoute deux inconnues sur un segment.
- Finalement, on numérote les noeuds associés à l'intérieur de l'élément. Pour une interpolation biquadratique sur un quadrilatère ou une interpolation cubique sur un triangle, on ajoutera un noeud. Pour une interpolation bicubique sur un quadrilatère, on ajoutera 4 inconnues.

Les coordonnées des noeuds supplémentaires sont obtenues en effectuant une interpolation (bi)-linéaires des quatre sommets pour les valeurs (ξ, η) qui correspondent à la définition formelle du noeud dans l'élément parent.

	P_1C_0	P_2C_0	P_3C_0	Q_1C_0	Q_2C_0	Q_3C_0
Nombre d'inconnues associées aux sommets	3	3	3	4	4	4
Nombre d'inconnues associées aux arête	0	3	6	0	4	8
Nombre d'inconnues associées à une face	0	0	1	0	1	4
Nombre local d'inconnues par élément	3	6	10	4	9	16

Pour un maillages avec N_0 sommets, N_1 arêtes et N_2 éléments, on peut déduire facilement le nombre global d'inconnues à partir des caractéristiques topologiques globales du maillage. Cela est résumé dans le tableau ci-dessous :

	Nombre global d'inconnues pour un maillage
P_1C_0	N_0
P_2C_0	$N_1 + N_0$
P_3C_0	$N_2 + 2N_1 + N_0$
Q_1C_0	N_0
Q_2C_0	$N_2 + N_1 + N_0$
Q_3C_0	$4N_2 + 2N_1 + N_0$

Et concrètement...

Pour résoudre le problème de l'approximation de la solution analytique de Stommel, nous allons utiliser la structure suivante et les fonctions qui y sont associées :

```
typedef struct {
    femMesh *mesh;
    femEdges *edges;
    femDiscrete *space;
    femIntegration *rule;
    femFullSystem *system;
    double minValue;
    double maxValue;
} femApproxProblem;

femApproxProblem    *femApproxCreate(const char *filename);
void                 femApproxFree(femApproxProblem *theProblem);
void                 femApproxLocal(const femApproxProblem *theProblem, const int i, int *map);
double               femApproxEval(const femApproxProblem *theProblem, int iElem, double xsi, double eta);
void                 femApproxPhi(double xsi, double eta, double *phi);
void                 femApproxDphi(double xsi, double eta, double *dphidxsi, double *dphideta);
double               femApproxStommel(double x, double y);

void                 glfemDrawColorRecursiveTriangle(femApproxProblem *theProblem,
                         double *x, double *y, double *xsi, double *eta, int iElem, int level);
void                 glfemPlotFieldRecursiveTriangle(femApproxProblem *theProblem, int level);
void                 glfemDrawIntegrationNodes(femApproxProblem *theProblem);
void                 glfemDrawValueNodes(femApproxProblem *theProblem);
```

L'implémentation récursive pour le dessin de la solution, ainsi que l'expression analytique de la solution de Stommel vous sont fournies par nos bons soins.... C'est donc assez simple à faire ! Plus précisément, on vous demande de concevoir, d'écrire ou de modifier quatre fonctions.

1. Tout d'abord, il s'agit d'écrire les fonctions

   ```
   void femApproxPhi(double xsi, double eta, double *phi);
   void femApproxDphi(double xsi, double eta, double *dphidxsi, double *dphideta);
   ```

   qui calcule les 10 fonctions de forme cubique sur un élément triangulaire pour une coordonnée `xsi eta`. Pour vous aider, il existe deux petites fonctions à notre avis bien pratique. Une fonction `femDiscreteXsi2` donne les coordonnées des noeuds tandis que `femDiscretePrint` imprime la valeurs de fonctions de forme aux noeuds. Tout cela devrait vous permettre de vérifier assez aisément votre implémentation. Ensuite, si l'exercice vous a amusé, vous pouvez également calculer les dérivées des fonctions de formes, même si ce n'est pas le plus important à faire, car vous n'en aurez pas besoin dans la suite du devoir...

2. Ensuite, il s'agira d'écrire la fonction qui permet d'obtenir l'approximation cubique de la fonction de Stommel.

   ```
   void femApproxSolve(femApproxProblem *theProblem);
   ```

   Une bonne partie du code vous est fourni : il suffit juste de construire la matrice et le membre de droite du système linéaire pour obtenir le résultat escompté. Ce n'est pas bien compliqué et vous devriez tous être capables d'obtenir rapidement la figure de l'énoncé !

3. Finalement, vous êtes invités à modifier la fonction

```
void femApproxLocal(const femApproxProblem *theProblem, const int iElem, int *map)
```

qui fournit la nurmérotation globale des noeuds d'un élément `iElem` dans le vecteur `map` de taille dix. On vous a fourni une fonction qui permet d'obtenir ce vecteur uniquement pour les es deux maillages contenant respectivement un ou deux éléments ! Pour pouvoir utiliser votre code sur les autres maillages, il sera nécessaire de modifier cette fonction. C'est évidemment la partie compliquée du devoir qui demande un tout petit peu plus de réflexion.... Il faut donc s'y attaquer uniquement lorsque vous avez obtenu un résultat correct sur le serveur avec le maillage avec deux éléments.... Attention, résoudre le problème cubique sur le maillage fin avec un solveur plein est vraiment très lent ! Il est aussi conseillé de supprimer le dessin des sommets, noeuds et points d'intégration dans le programme principal `main.c`.

Ici, on vous demande de juste prédire la numérotation globale des dix variables locales d'un triangle pour une interpolation P_3C_0. Le nombre global de variables a déjà été calculé par nos soins dans la fonction `femApproxCreate`... Il est sans doute utile de vérifier que vous n'introduisez pas de numéro global supérieur à la taille de matrice du système linéaire. Les esprits taquins observeront que la numérotation proposée pour le maillage avec deux éléments est légèrement distincte que ce qui est suggéré ici : cela a été fait afin que la fonction fournie puisse fonctionner pour le maillage avec un et deux éléments. En d'autres mots, votre nouvelle fonction devrait fournir une autre numérotation pour le maillage `two.txt`.

4. Vos cinq fonctions seront incluses dans un unique fichier `homework.c`.
Toujours bien vérifier que la compilation s'exécute correctement sur le serveur.
Si vous avez une erreur sur le serveur et pas sur votre ordinateur, cela ne veut pas dire que le serveur est corrompu, cela veut juste dire que votre programme incorrect fonctionne bien par miracle sur votre ordinateur. Et comme vous le savez tous, des Miracles il y en a tous les jours :-)

5. Afin de pouvoir effectuer un test distinct de vos fonctions, des instructions de compilation ont été mis dans le fichier `homework.c`. Il est IMPERATIF de ne pas les retirer, ni de les modifier...

Bon amusement !