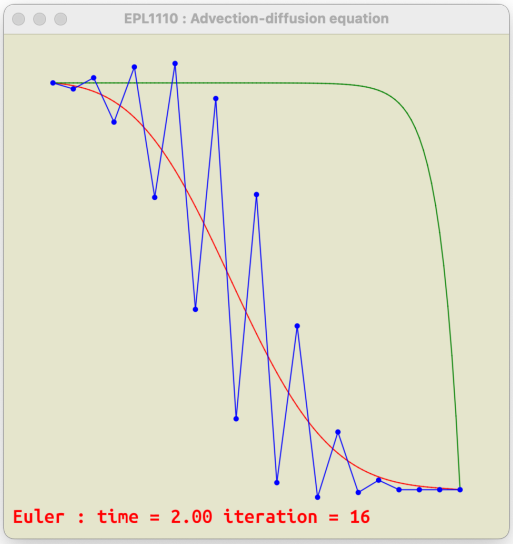
Finite differences for dummies : Advection-diffusion instationnaire



Le temps fait son apparition :-) Pour commencer, nous allons considérer un tout simple problème d'advection diffusion sur le domaine $\Omega =]0,1[$. A l'instant initial, la solution est nulle et on impose brutalement une valeur unitaire à l'extrémité gauche de l'intervalle en $x = 0$.

$$\begin{aligned} \frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} &= \epsilon \frac{\partial^2 u}{\partial x^2} \\ u(0,t) &= 1 \\ u(1,t) &= 0 \\ u(x,0) &= 0 \qquad x \in]0,1[\end{aligned}$$

où les paramètres ϵ et β sont des constantes positives.

L'objectif est d'obtenir la solution de régime en intégrant le système discret obtenu par différences finies et deux schémas temporels explicites: Euler et Runge-Kutta d'ordre quatre. On vous demande donc d'implémenter la discrétisation par différences finies sur une grille uniforme. Pour le terme de transport, on effectuera une combinaison pondérée d'une dérivée centrée et d'une dérivée amont. Les facteurs de pondération seront respectivement $(1-\zeta)$ et ζ. Il s'agit donc d'intégrer les équations suivantes pour obtenir l'évolution temporelle des valeurs nodales :

$$\begin{aligned} \frac{dU_i}{dt} &= \overbrace{-(1-\zeta)\beta \frac{U_{i+1} - U_{i-1}}{2h} - \zeta\beta \frac{U_i - U_{i-1}}{h} + \epsilon \frac{U_{i+1} - 2U_i + U_{i-1}}{h^2}}^{f(U_i)} \qquad i = 1, \ldots, n-1 \\ U_0 &= 1 \\ U_n &= 0 \end{aligned}$$

Une petite structure reprend la description de notre problème :

```
typedef struct {
    double epsilon;
    double beta;
    double zeta;
    int size;
    int sizePlot;
    double *X;
    double *U;
    double *x;
    double *u;
    double *uregime;
} femProblem;
```

où le vecteur des noeuds et des inconnues de taille `size` sont respectivement `X` et `U`. La solution analytique de régime et la solution analytique seront stockées dans les les vecteurs de taille `sizePlot` pour les abscisses `x`. Globalement, ces vecteurs ne sont utilisés que pour les représentations graphiques et on

sélectionnera la taille `sizePlot` afin d'avoir un dessin plus ou moins précis. Pour une fois, il ne s'agit pas que de programmation : il faudra aussi un peu réfléchir à la théorie sous-jacente pour trouver les formules adéquates ! Plus précisément, on vous demande de concevoir, d'écrire ou de modifier cinq fonctions.

1. Tout d'abord, il s'agit d'obtenir la solution analytique du problème de la chaleur.

   ```
   void       advdiffSolution(femProblem *myProblem, double time);
   ```

   On vous fournit une fonction qui calcule la solution de régime du problème pour les abscisses contenues dans le vecteur `myProblem->x`. On vous demande de donner la solution analytique transitoire de l'équation d'advection diffussion à un instant `t`. Le résultat sera inclu dans le vecteur `u`. On consultera utilement les notes du cours LEPL1103 à ce sujet. Le résultat doit être précis à 10^{-8} et être calculé avec un nombre minimal de termes de la série de la solution analytique : il faut donc obtenir la précision requise avec un souci d'efficacité ! On vous a fourni la solution pour $\beta = 0$ et il vous faut donc la généraliser au cas avec une valeur de β non-nulle.

2. Ensuite, vous mettrez au point une fonction

   ```
   double     advdiffComputeTimeStep(femProblem *myProblem);
   ```

   qui fournit le pas de temps maximal pour que l'intégration numérique avec la méthode de Runge-Kutta d'ordre quatre explicite soit stable. Attention, votre fonction doit être idéalement correcte pour tous les valeurs positives des trois paramètres ! La fonction fournie fait actuellement ce calcul pour le méthode d'Euler explicite.

3. Finalement, il s'agira d'écrire les trois fonctions permettant l'intégration numérique des équations discrètes obtenues avec les différences finies.

   ```
   void       advdiffComputeRightHandSide(femProblem *myProblem, double *u double *f);
   void       advdiffUpdateEuler(femProblem *myProblem, double dt);
   void       advdiffUpdateRungeKutta(femProblem *myProblem, double dt);
   ```

   La première fonction calcule la fonction $f(U_i)$ et met le résultat dans le vecteur `F` qui a été allouée auparavant. En tirant profit de la première fonction, les deux autres fonctions effectuent respectivement une itération d'Euler explicite et de Runge-Kutta d'ordre quatre[1]. Les arguments sont le pas de temps et les valeurs nodales courantes. La mise à jour des valeurs nodales se fera en place, c'est-à-dire, dans le vecteur contenant les anciennes solutions.

4. Vos six 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.

[1] Dans notre cas, ces deux méthodes sont définies par les relations de récurrence données par :

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \Delta t\, \mathbf{f}(U^n)$$

$$\begin{aligned} \mathbf{U}^{n+1} &= \mathbf{U}^n + \tfrac{\Delta t}{6}(\mathbf{K}_1 + 2\mathbf{K}_2 + 2\mathbf{K}_3 + \mathbf{K}_4) \\ \mathbf{K}_1 &= \mathbf{f}(\mathbf{U}^n) \\ \mathbf{K}_2 &= \mathbf{f}(\mathbf{U}^n + \tfrac{\Delta t}{2}\mathbf{K}_1) \\ \mathbf{K}_3 &= \mathbf{f}(\mathbf{U}^n + \tfrac{\Delta t}{2}\mathbf{K}_2) \\ \mathbf{K}_4 &= \mathbf{f}(\mathbf{U}^n + \Delta t \mathbf{K}_3) \end{aligned}$$