Filière SEM : Méthodes numériques pour les EDP
Année Universitaire 2010-2011

Code Scilab "Diffusion de la chaleur"

Contact : *Christophe.Corre@legi.grenoble-inp.fr*

1 Observations

Avant d'abandonner, provisoirement, l'équation de diffusion linéaire et non-linéaire, il est important de faire quelques remarques sur l'utilisation efficace de Scilab puisque vous allez pouvoir utiliser le code développé lors des séances 1 et 2 pour traiter les nouveaux problèmes qui vous seront proposés dans la suite du cours. Il est en effet préférable que ce code soit codé le plus efficacement possible ! Ce point n'a pas été notre préoccupation jusqu'à présent car l'accent avait été mis sur les méthodes numériques elles-mêmes plus que sur leur implémentation dans Scilab.

Le point important à signaler à ce stade est que Scilab est un code qui travaille très efficacement avec des données **vectorielles** : il est donc particulièrement important d'exploiter ce point. Avant de donner les quelques explications techniques qui s'imposent, on va tout d'abord illustrer les gains d'efficacité qui peuvent être obtenus grâce à une utilisation judicieuse de ces "propriétés vectorielles".

On reprend donc le problème de la séance 2 (cf. le corrigé disponible) et on effectue, dans le cas non-linéaire, 16000 itérations avec le critère de stabilité établi pour le pas de temps adimensionné, dans un maillage de 201 cellules et en utilisant une fréquence d'affichage de 2000 itérations. Le calcul est effectué d'une part en utilisant le script $heat1D\_numerique\_complet.sce$ (qui fait appel aux scripts $param.sce$, $solex.sce$ et $diffus.sce$) et d'autre part au script $heat1D\_numerique\_complet\_fast.sce$ (qui fait appel aux mêmes scripts). La différence entre les deux programmes résident donc exclusivement dans le codage du programme principal. La version "fast" est celle qui exploite le caractère vectoriel de Scilab. La commande $timer()$, sur laquelle on reviendra, permet de connaître avec précision le temps d'exécution associé à chaque programme. Le résultat obtenu est éloquent : le code de base utilisant $heat1D\_numerique\_complet.sce$ requiert 660 secondes pour effectuer la simulation demandée (ce temps dépend bien sûr de la machine utilisée) contre seulement 13 secondes environ pour le programme utilisant $heat1D\_numerique\_complet\_fast.sce$ qui porte donc bien son nom (le rapport de temps CPU en faveur de cette version vectorielle est donc de l'ordre de 50 !). Détaillons maintenant les raisons de cette spectaculaire réduction du temps de calcul.

2 Analyse

Si on étudie le script *heat1D_numerique_complet.sce* on constate qu'il ne contient plus une seule boucle sur l'indice spatial i.
Au lieu de créer le tableau des coordonnées des points du maillage en utilisant

```
dxi = 2. / (imax-1);
for i=1:imax
      xi(i)=(i-1)*dxi;
end
```

on fait appel à la fonction intégrée dans Scilab, *linspace* qui génère *imax* points uniformément répartis dans l'intervalle $[0, 2]$:

```
dxi = 2. / (imax-1);
xi=linspace(0,2,imax);
xi=xi';
```

On notera bien l'utilisation de l'opérateur ′ qui correspond à l'opérateur de transposition. Le vecteur créé par *linspace* est en effet un vecteur ligne et on souhaite plutôt travailler avec un vecteur colonne afin de disposer d'un vecteur de coordonnées qui soit de type (vecteur colonne donc) cohérent avec les vecteurs qui vont être créés ensuite pour les températures.
Pour initialiser le tableau des températures adimensionnées, on évite d'écrire

```
for i=2:imax-1
theta(i)=1;
end
theta(1)=0;
theta(imax)=0;
```

et on utilise plutôt :

```
theta(1:imax)=1;
theta(1)=0;
theta(imax)=0;
```

On notera bien ici l'affectation "en bloc" de la valeur 1 à l'ensemble des éléments du vecteur *theta* (on corrige ensuite les valeurs aux points frontières).
On recherche ensuite la valeur maximale de φ_i afin de calculer le pas d'avancement en temps à l'aide du critère de stabilité établi en séance 2. Dans le programme de base on écrit :

```
phimax=0;
for i=2:imax-1
phimax=max(phimax,phi(theta(i)));
end
```

et on fait donc encore appel à une boucle sur l'indice de maillage i. Dans la version améliorée / optimisée du code, on fait appel à la fonction *max* de Scilab en exploitant au maximum le fait qu'elle permet de détecter le maximum de l'ensemble des valeurs d'un tableau. Ainsi, en écrivant simplement :

```
phimax=max(phi(theta));
```

on calcule bien la valeur maximale du vecteur colonne qui contient l'ensemble des valeurs ϕ_i calculées en utilisant l'ensemble des valeurs θ_i.
Pour ce qui concerne maintenant le coeur du code, on écrit dans le programme de base :

```
for i=2:imax
     flux_num(i)=phi(0.5*(theta(i-1)+theta(i)))*(theta(i)-theta(i-1))/dxi;
end

for i=2:imax-1
     theta(i)=theta(i)+(dFo/dxi)*(flux_num(i+1)-flux_num(i));
end
```

soit deux boucles sur l'indice spatial afin de calculer tout d'abord le flux numérique en chaque point $i-\frac{1}{2}$ du maillage puis d'actualiser la valeur de θ_i en tout point intérieur du domaine de calcul à l'aide d'un bilan de flux numérique.
Les mêmes opérations sont effectuées de façon beaucoup plus efficace en écrivant :

```
flux_num(2:imax)=
phi(0.5*(theta(1:imax-1)+theta(2:imax))).*(theta(2:imax)-theta(1:imax-1))/dxi;

theta(2:imax-1)=theta(2:imax-1)+(dFo/dxi)*(flux_num(3:imax)-flux_num(2:imax-1));
```

Deux points importants sont à souligner dans ces deux lignes très concises :

- on doit noter d'une part l'utilisation des bornes pertinentes pour l'affectation en bloc de chacun des éléments des vecteurs utilisés. Si i varie de 2 à $imax$ dans $flux\_num$ alors i varie de 1 à $imax-1$ dans $theta(i-1)$ et de 2 à $imax$ bien sûr dans $theta(i)$.
- on doit noter d'autre part l'important opérateur . utilisé devant la multiplication $*$ lorsque l'on effectue le produit entre $\varphi(\mu\theta_{i-\frac{1}{2}})$ et $\dfrac{\delta\theta_{i-\frac{1}{2}}}{\Delta\xi}$. C'est l'utilisation de cet opérateur qui fait comprendre à Scilab que l'on souhaite bien une multiplication "point par point" des quantités $\varphi(\mu\theta_{i-\frac{1}{2}})$ d'une part et $\dfrac{\delta\theta_{i-\frac{1}{2}}}{\Delta\xi}$ d'autre part. En l'absence de cet opérateur . devant $*$, l'opérateur multiplication $*$ entre le vecteur contenant $\varphi(\mu\theta_{i-\frac{1}{2}})$ pour $i=2,imax$ et le vecteur contenant $\dfrac{\delta\theta_{i-\frac{1}{2}}}{\Delta\xi}$ toujours pour $i=2,imax$, cette opération de multiplication $*$ retournerait le produit scalaire de ces deux vecteurs et non pas la multiplication terme à terme souhaitée.

Enfin, le calcul de la solution exacte est effectuée dans le programme de base en écrivant :

```
for i=1:imax
   thetaex(i)=solex(Fo,xi(i),nmode);
end
```

alors que le programme modifié exploite le caractère vectoriel de Scilab en calculant en une seule fois :

```
  thetaex=solex(Fo,xi,nmode);
```

La fonction *solex* peut être définie de la même façon pour le programme de base et pour le programme modifié car il n'y a pas d'opération "élément par élément" intervenant dans la définition de cette fonction.
C'est la combinaison de ces différentes opérations, et en particulier le calcul "vectoriel" du schéma numérique, qui permet de diviser par 50 le temps de calcul d'un programme par rapport à l'autre.
Signalons au passage que le temps d'exécution est estimé à partir de la fonction $timer()$. On effectue un premier appel à $timer()$ en début de programme puis un second appel à $timer()$ en fin de programme en affectant la valeur retournée , *i.e.* le temps CPU écoulé entre les deux appels successifs à $timer()$, à la variable tps_CPU qui est ensuite affichée à l'écran.
Pour conclure, on notera également que le programme $heat1D\_numerique\_complet\_fast.sce$ contient quelques améliorations du programme de base du point de vue du rendu graphique de la solution et de son évolution. Ainsi, en utilisant la commande *subplot* on positionne les différentes courbes dans un carré 2×2 : la commande $subplot(mnp)$ placée avant $plot2d$ indique au code que la figure associée à cette commande $plot2d$ va être positionnée en position p d'une "matrice" $m \times n$ de figures. La commande *xtitle* placée avant $plot2d$ permet enfin de donner un titre à chaque figure ou sous-figure ainsi que des labels ou légendes aux axes de chacune des figures ou sous-figures (cf. Fig.1).

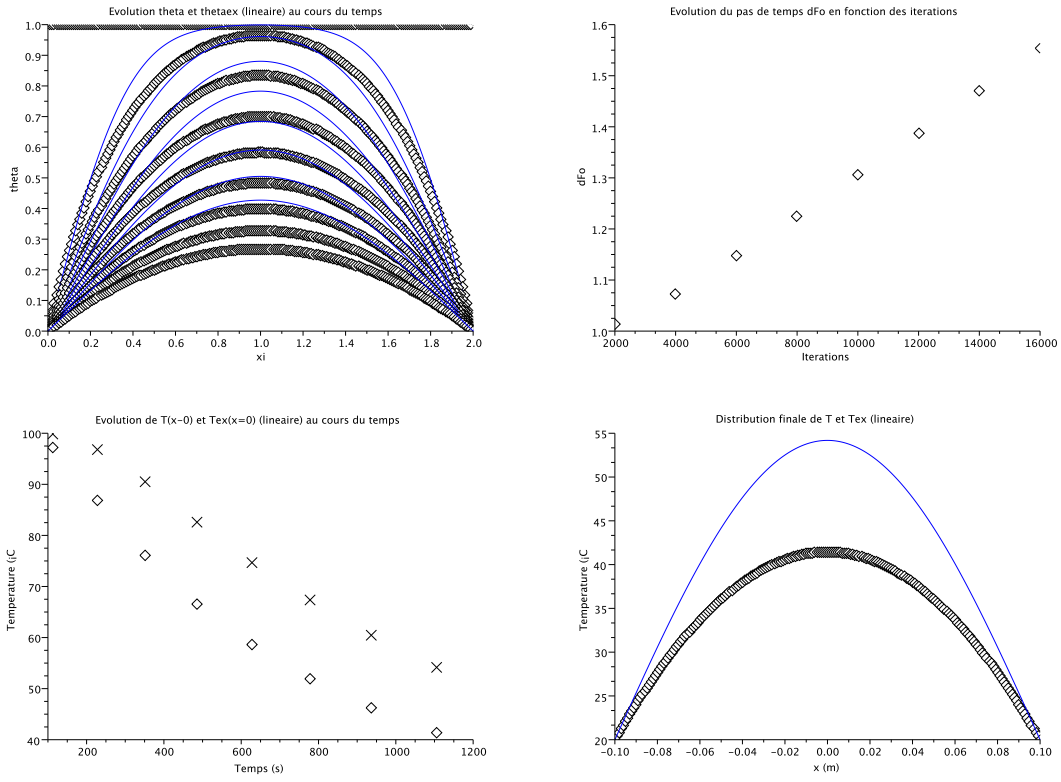

Fig. 1: Affichage des différentes courbes de résultats dans une unique fenêtre graphique grâce à la fonction *subplot*.