Rapport du TP numéro 2 d'EDP 2
Equation de la chaleur 2D par un θ-schéma

Elise DURAND et Nicolas BONNEEL

31 janvier 2005

Table des matières

1 Introduction

Nous allons étudier le problème parabolique suivant (équation de la chaleur 2D) :

$$\begin{cases} \partial_t u(x,y,t) - \partial_x^2 u(x,y,t) - \partial_y^2 u(x,y,t) = 0, & (x,y) \in]0,1[\times]0,1[, t \in]0,T[, \\ u(0,y,t) = u(1,y,t) = 0, & y \in [0,1], t \in [0,T], \\ \partial_y u(x,0,t) = \partial_y u(x,1,t) = 0, & x \in [0,1], t \in [0,T], \\ u(x,y,0) = \alpha(x,y), & (x,y) \in [0,1] \times [0,1] \end{cases} \tag{1}$$

où α est une fonction donnée dans $H^1([0,1] \times [0,1])$ vérifiant la condition de compatibilité suivante : $\alpha(0,y) = \alpha(1,y) = 0,\ y \in [0,1]$.

selon un θ-schéma pour la résolution temporelle, et une méthode d'éléments finis pour la résolution spatiale, pour différentes valeurs de θ, du pas de temps, du temps final et de la condition initiale.

2 Résolution approchée par un θ-schéma et programmation Matlab

2.1 Obtention du maillage

On maille le domaine $]0,1[\times]0,1[$ grâce au mailleur de Matlab PDETOOLBOX (commande pdetool). On raffine le maillage pour obtenir un maillage structuré de pas 1/32 sur les deux axes.
On exporte les variables p, e et t relatives au maillage.

Lors de l'exportation du maillage, les variables p, e et t sont créées :

- p : représente les coordonnées x et y de chaque point nodal. (matrice 2xN)
- e : donne des informations sur les frontières du maillage (matrice 7xm) :
  - lignes 1 et 2 : numéros des extremités du segment
  - lignes 3 et 4 : abscisse curviligne (ramenée à [0,1]) des extremités par rapport à la frontière sur laquelle le segment se situe
  - ligne 5 : numero de la frontière sur laquelle le segment se situe
  - lignes 6 et 7 : numeros des sous-domaines gauches et droites (par rapport au sens du segment)
- t : donne les numéros des points constituant chaque triangle, ainsi que le domaine dans lequel le triangle se trouve (matrice 4xn)

2.2 Formation des matrices masse et raideur finales et résolution

On utilise la fonction assema qui nous donne la matrice masse M et la matrice raideur K. Ici, le vecteur chargement F est nul. $U^{(n)}$ désigne le vecteur inconnu du déplacement au temps t_n à l'intérieur du maillage (obtenu grâce à la fonction find de Matlab)
La formulation variationnelle conduit au système matriciel suivant :

$$\begin{cases} M(\dot{U})^{(n)} + K.U^{(n)} = F, \\ U^{(0)} = \alpha, \end{cases} \tag{2}$$

où $\dot{U}$ désigne la dérivée du vecteur U par rapport au temps et α est le vecteur contenant la donnée initiale α aux points de discretisation.

On construit alors un θ-schéma :

$$\begin{cases} (M.\frac{U^{(n+1)}-U^{(n)}}{\Delta t} + (1-\theta).K.U^{(n)} + \theta.K.U^{(n+1)} = (1-\theta).F^{(n)} + \theta F^{(n+1)}, \\ U^{(0)} = \alpha, \end{cases} \tag{3}$$

Comme F est nul, on peut simplifier pour obtenir :

$$\begin{cases} (\frac{1}{\Delta t}.M + \theta.K).U^{(n+1)} = (\frac{1}{\Delta t}.M - (1-\theta).K).U^{(n)} \\ U^{(0)} = \alpha, \end{cases} \tag{4}$$

On pose $A = \frac{1}{\Delta t}.M + \theta.K$ et $B = \frac{1}{\Delta t}.M + (1-\theta).K$ et on obtient :

$$\{ \ A.U^{(n+1)} = B.U^{(n)} U^{(0)} = \alpha \tag{5}$$

Le principe de résolution est le suivant :
- Calcul de $U^{(0)}$
- Obtention des matrices K et M
- Calcul de A et B
- factorisation LU de A
- A chaque pas de temps faire :
  - Calcul du second membre $S = B.U^{(n)}$
  - Resolution de $L.Z = S$ avec L triangulaire inferieure
  - Resolution de $U.U^{(n+1)} = Z$ avec U triangulaire supérieure (ne pas confondre avec $U^{(n)}$ solution du problème)

  - Affichage des solutions / calcul des solutions exactes / calculs d'erreurs
- Affichage de la courbe d'erreur

3 Mise au point du programme de résolution approchée

3.1 Problème ayant une solution analytique connue.

On utilise la donnée initiale suivante :

$$\alpha(x,y) = \begin{cases} x, & 0 \leq x \leq 1/2, \\ 1-x, & 1/2 \leq x \leq 1. \end{cases} \tag{6}$$

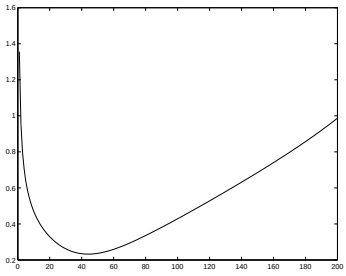
On resoud les conditions aux limites sur la derivee en remarquant que comme α ne depend pas de y, u ne dependra pas de y pour tout t (et donc $\partial_y u = 0$)
La solution $u(x,y,t)$ est alors donnée sous la forme d'une série de Fourier :

$$u(x,y,t) = \frac{4}{\pi^2} \sum_{l=0}^{l=\infty} \frac{(-1)^l}{(2l+1)^2} e^{-(2l+1)^2\pi^2 t} sin((2l+1)\pi x).$$

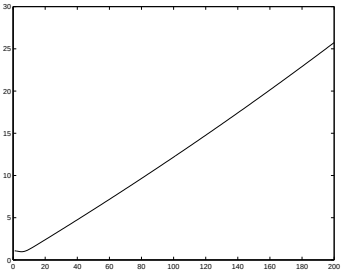
On n'utilisera que les 200 premiers termes de la somme.

3.2 Mise au point du programme

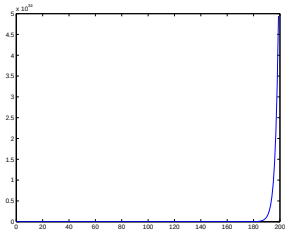
Le tracé des courbes d'erreur donne :



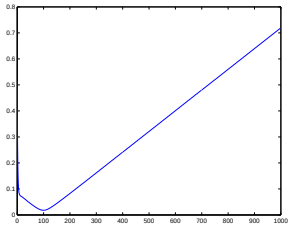
θ =0.5 et Δt=0.005



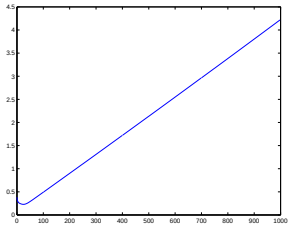
θ =1 et Δt=0.005



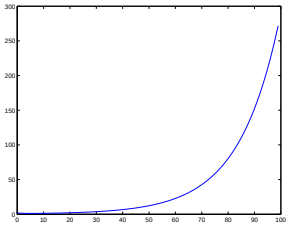
θ =0.4 et Δt=0.005



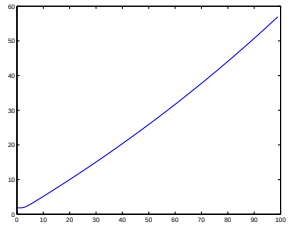
θ =0.5 et Δt=0.001



θ =1 et Δt=0.001



θ =0.5 et Δt=0.01



θ =1 et Δt=0.01

On constate que :

- Euler Implicite ($\theta = 1$) est stable pour des pas de temps petits (0.001)
- Crank-Nicholson est plus précis que Euler Implicite pour les petits pas de temps, mais l'est moins pour les pas de temps plus grands
- Pour $\theta = 0.4$, le schéma ne respecte pas les conditions de stabilité.

4 Expérimentation numérique

4.1 Exemple traité.

On utilise le script précédent pour résoudre l'équation de la chaleur avec la condition initiale :

$$\alpha(x,y) = \begin{cases} 1 - 4\sqrt{(x-1/2)^2 + (y-1/2)^2}, & \text{si } \sqrt{(x-1/2)^2 + (y-1/2)^2} \leq 1/4, \\ 0, & \text{sinon.} \end{cases} \tag{7}$$

On choisit T=0.5 et une solution de référence avec θ=0.5 et Δt=0.001 (solution théoriquement très précise). On trouve les erreurs relatives à la solution de référence pour différentes valeurs de Δt et θ :

Test	1	2	3
Δt	0.001	0.01	0.01
θ	1	0.5	1
Erreur finale	2.46	127.54	15.57

Ces résultats confirment que le schéma de Euler Implicite est plus stable et plus precis que celui de Crank-Nicholson pour les pas de temps "assez grands".

5 Conclusion

Nous avons utilisé différents θ-schéma pour étudier la convergence et la stabilité en temps de l'équation de la chaleur 2D et nous avons vu que Crank-Nicholson est meilleur pour des pas de temps petits, que Euler Implicite est meilleur pour des pas de temps plus grands. Nous avonsd aussi vu que le schéma avec $\theta = 0.4$ n'est pas du tout stable. En revanche nous n'avons pu observer les erreurs dues à la méthode des éléments finis.

6 Annexes

6.1 Script global utilisé pour les deux exercices

```
% entree des donnees utilisateur
T_final = input('Temps final : ');
theta = input('Theta : ');
dt = input('Pas de temps : ');
Np = input('Nb de pas de temps separant chaque tracé : ');
alpha = input('fonction U0 : ','s');

% chargement du maillage, conditions aux limites et initiales
load mesh
N = length(p);
[K,M]=assema(p,t,1,1,0);
IN=find(p(1,:)>0.00001 & p(1,:)<0.9999);
Un = feval(alpha,p(:,IN));

K=K(IN,IN);
M=M(IN,IN);
A=(1./dt)*M+theta*K;
B=(1./dt)*M-(1-theta)*K;
clear K;
clear M;
[LA,UA]=lu(A);

X = zeros(N,1);
X(IN)=Un;
erreur = 0.;

% iteration en temps
for i=0:floor(T_final/dt)

  % solution calculée
  Un = reshape(Un,length(Un),1);
  if (mod(i,Np)==0)
    figure(1);
    pdeplot(p,e,t,'zdata',X);
    pause(0.010);
  end;
```

```
  Y=B*Un;
  Z=LA\Y;
  Un=UA\Z;


 % solution exacte pour le premier exercice
  if (size(alpha)==size('alph'))
        if (alpha=='alph')
        Sol=0;
        for l=0:200
          l2=2*l+1;
          Sol = Sol+((((-1).^l)./(l2.^2)).*exp(-l2.^2*pi^2*i*dt))
               .*sin(l2.*pi.*p(1,:));
        end;
        Sol = Sol*4/(pi^2);

        if (mod(i,Np)==0)
          figure(2);
          pdeplot(p,e,t,'zdata',Sol);
          pause(0.010);
        end;

        erreur(i+1) = norm(X'-Sol)/norm(Sol)*100;
     end;
  end;

  X(IN)=Un;

end; % fin iteration en temps



% chargement du fichier de reference et calcul de l'erreur
% sur la solution de reference pour l'exercice 2
if (size(alpha)==size('alph2'))
   if (alpha=='alph2')
    load 'reference.mat' Un;
    'erreur au temps final :'
    erreur = norm(X(IN)-Un)/norm(Un)*100
  end;
```

```
end;

% calcul de l'erreur sur la solution exacte pour le 1er exercice
if (size(alpha)==size('alph'))
   if (alpha=='alph')
    figure(3);
     % (on n'affiche pas l'erreur sur la condition initiale)
    plot(erreur(2:length(erreur)));
    end;
end;
```

6.2 Fonctions 'alph' et 'alph2' correspondants aux exercices 1 et 2

6.2.1 Exercice 1

```
function U=alph(P)
  G = find(P(1,:)>0.5);
  U=zeros(length(P),1);
  U = P(1,:);
  U(G)=1-P(1,G);
```

6.2.2 Exercice 2

```
 function U=alph2(P)
  G = find( ((P(1,:)-0.5).^2+(P(2,:)-0.5).^2).^0.5 <=0.25);
  U=zeros(length(P),1);
  U(G) = 1-4* ((P(1,G)-0.5).^2 + (P(2,G)-0.5).^2).^0.5;
```