Informatique Scientifique : Partiel (partie C++: 1 heure tout document)

F. Hecht et O. Pironneau

23 mars 2007

Soit $L, \mu \in \mathcal{R}^+,\ \Omega = (0, L),\ u^0 \in L^2(\Omega)$. Soit à résoudre l'équation de la chaleur

$$\partial_t u - \mu \partial_{xx} u = 0, \quad u|_{t=0} = u^0, \quad u|_{\partial\Omega} = 0 \quad \text{ in } \Omega \times (0, T) \tag{1}$$

On utilise le schéma d'Euler implicite en temps et une discrétisation par éléments finis P^1 en espace sur les intervalles (x^i, x^{i+1}), où $x^0 = 0, x^I = L$, ce qui donne (où $(.,.)$ désigne le produit scalaire de L^2) :

$$\begin{aligned} &(u^{m+1} - u^m, v) + \mu \delta t (\partial_x u^{m+1}, \partial_x v) = 0, \quad \forall v \in V_h \\ &u^{m+1} \in V_h := \{ w \in C^0(\bar{\Omega}) \ : \ w|_{(x^i, x^{i+1})} \in P^1,\ w(0) = w(L) = 0 \} \end{aligned} \tag{2}$$

Soit w^i la fonction de V_h telle que $w^i(x^j) = \delta_{ij}$; en portant dans (2)

$$u^m(x) = \sum_1^{I-1} u_i^m w^i(x) \text{ et en posant } \vec{u}^m = (u_1^m, ..., u_{I-1}^m)^T \tag{3}$$

on obtient un système linéaire pour u^{m+1} : $(B + \mu \delta t A)\vec{u}^{m+1} = B\vec{u}^m$. On démontre aisément que si $x^{i+1} - x^i = h$ pour tout i alors B et A sont des matrices tri-diagonales symétriques avec

$$B_{i-1,i} = h/6, \quad B_{i,i} = 2h/, \quad A_{i-1,i} = -\frac{1}{h}, \quad A_{i,i} = \frac{2}{h}$$

Cette méthode est implémentée en C++ comme suit :

```
#include <iostream>
#include <cassert>
```

```
#include <cmath>
#include <vector>

using namespace std;
const double pi = 4*atan(1.);
class Matrois;
/////////////////////////////////////////////////////////////
class Vector : public vector<double> {  public :
    Vector(const int n){ resize(n); }
      double&  operator[] (const int i)       { return at(i);}
const double&  operator[] (const int i) const { return at(i);}
      void solveLU(Matrois& aa);
};
/////////////////////////////////////////////////////////////
class Matrois { public :
    int size, isLU;
    Vector a,b,c;
    Matrois(int size1) : size(size1),a(size),b(size),c(size) { isLU=0;}
    Vector operator*(Vector& u);
    Matrois(Matrois& aa);
    Matrois operator=(Matrois& aa);
    void factLU();
};

Vector Matrois : :operator*(Vector& u){
    Vector v(size+1);
    for(int i=1;i<size;i++)
        v[i] = a[i]*u[i-1]+b[i]*u[i]+c[i]*u[i+1];
    return v;
}
void Matrois : :factLU()
{
    if(!isLU)
    {    c[1] / b[1];
        for(int i=2; i<size;i++) {
            b[i] -= a[i]*c[i-1];
        c[i] / b[i];
        }
```

```
        isLU = true;
    }
}
/////////////////////////////////////////////////////
void Vector::solveLU(Matrois& aa)
{
    if(!aa.isLU) aa.factLU();
    at(1) / aa.b[1];
    for(int i=2;i<aa.size;i++)
        at(i)= (at(i) - aa.a[i]*at(i-1))/aa.b[i];
    for(int i=aa.size-1;i>0;i--)
        at(i) -= aa.c[i]*at(i+1);
}
///////////////////////////////////////////////////
class chaleur{ public:
    double L,mu, T;
    int I, M;
    Matrois AB,B;
    Vector& u;
    chaleur(double L1, double mu1, double T1, int I1, int M1, Vector& u0);
};

chaleur::chaleur(double L1, double mu1, double T1, int I1, int M1, Vector&
u0)
            : L(L1), mu(mu1), T(T1), I(I1), M(M1), u(u0), AB(I), B(I)
{
        double dt = T/M, h=L /I;
        for(int i=0;i<I;i++){
            B.a[i] = h/6;  B.b[i] = 2*h/3;  B.c[i] = h/6;
            AB.a[i] = B.a[i] - mu*dt/h;
            AB.b[i] = B.b[i] + 2*mu*dt/h;
            AB.c[i] = B.c[i] - mu*dt/h;
        }
        u[0] = 0;   u[I]=0;
        for( int m=0;m<M;m++){
            u = B*u;
            u.solveLU(AB);
        }
```

```
}
/////////////////////////////////////////////////////////////////////////////
void myexit(){
    cout≪"exit"≪endl ;
}

int main() {
    atexit(myexit) ;
    const int I=25 ;
    Vector u0(I+1) ;
    for(int i=0 ;i≤I ;i++) u0[i] = sin(pi∗i/(double)I) ;
    chaleur pb(1,0.1,0.5,I,10,u0) ;
    for(int i=0 ;i≤I ;i++) cout ≪ pb.u[i]≪endl ;
    return 0 ;
}
```

Q1 Implémenter la fonction $Matrois :: Matrois(Matrois\& \ aa)$

Q2 Combien vaut T ?

Q3 Dans le `main()` remplacer

$$for(int\ i = 0; i <= I; i++)cout << pb.u[i] << endl$$

par les instructions C++ qui écrivent les valeurs `pb.u[i]` dans un fichier (de nom `result.txt`) dans un format permettant le tracé de la courbe $i \to pb.u[i]$ par gnuplot.

Q4 Rajouter et Implémenter $Matrois\& \ Matrois :: operator+= (Matrois\& \ aa)$

Q5 On veut changer de schéma et utiliser Crank-Nicolson en temps, c'est a dire

$$(u^{m+1} - u^m, v) + \frac{\mu\delta t}{2}(\partial_x(u^{m+1} + u^m), \partial_x v) = 0, \quad \forall v \in V_h \qquad (4)$$

Modifier le programme pour qu'il traite ce schéma. (On pourra éventuellement poser $v = (u^{m+1} + u^m)/2$ et récrire (4) en terme de v et u^m).