C++ L3 Math TP1 : résolution de l'équation de la chaleur

P. Helluy, F. Lecourtier

6 février 2025

1 Objectif

Implémenter en C++ un schéma explicite pour résoudre l'équation de la chaleur sur un domaine carré $\Omega =]0, L[^2$:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2},$$

où $u(x, y, t)$ est la fonction inconnue qui dépend des deux variables d'espace $(x, y) \in \Omega$. On définit un paramètre de discrétisation en espace $h = L/nx$ où nx est un entier. Le paramètre de discrétisation en temps Δt est donné par la formule

$$\Delta t = h^2/8. \tag{1}$$

On pose $x_i = y_i = ih + h/2$, $i = 0, 1, \ldots, nx - 1$ et $t_k = k\Delta t$. La fonction u est approchée aux points (x_i, y_j) par

$$u_{i,j}^k \simeq u(x_i, y_j, t_k).$$

On utilise la formule suivante

$$\frac{u_{i,j}^{k+1} - u_{i,j}^k}{\Delta t} - \frac{-4u_{i,j}^k + u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k}{h^2} = 0.$$

Pour les bords on utilise les formules

$$u_{-1,j} = u_{0,j}, \quad u_{nx,j} = u_{nx-1,j},$$

$$u_{i,-1} = u_{i,0}, \quad u_{j,nx} = u_{j,nx-1}.$$

2 Discrétisation

Utiliser un maillage uniforme de $nx \times nx$ points avec : - Pas d'espace : $h = L/nx$ - Pas de temps : Δt respectant la formule (1).

3 Structure du code

3.1 Classe Grid2d (grid.h et grid.cpp)

```
#ifndef GRID_H
#define GRID_H

class Grid2d
{
private:
    int nx;         // Nombre de points par direction
    double dx;      // Pas d'espace
    double dt;      // Pas de temps
    double *data;  // Stockage des valeurs
    int* ref_count; // Compteur de references

public:
    // Constructeur principal
    Grid2d(double L, int nx);

    // Constructeur par copie
    Grid2d(const Grid2d &other);

    // Destructeur
    ~Grid2d();

    // accesseur dt
    double get_dt() const { return dt; }

    // Accesseur operateur ()
    double &operator()(int i, int j);
    const double &operator()(int i, int j) const;

    // condition initiale (1 sur un cercle de rayon r centré en (x0
    , y0))
    void init(double x0, double y0, double r);

    // tracé: génération d'un fichier de données
    // puis tracer par gnuplot
    void plot() const;

    // avancer d'un pas de temps
    // schéma chaleur explicite à partir de unow (constant)
    void step(const Grid2d &unow);

    // fonction amie pour échanger deux grilles
    friend void swap(Grid2d &a, Grid2d &b);
};

#endif
```

Listing 1 – grid.h

La donnée $u_{i,j}$ est stockée dans le tableau data à la position data[i+j*nx]. La surcharge de l'opérateur operator() permet d'écrire u(i,j).

3.2 Programme principal (main.cpp)

```
#include "grid.h"
#include <iostream>

int main()
{
    // Parametres physiques
    const double L = 1.0; // Longueur du domaine
    const int nx = 50;    // Nombre de points

    // Initialisation des grilles
    Grid2d unow(L, nx);
    Grid2d unext(L, nx);

    unow.init(0.5, 0.5, 0.1);
    unext.init(0.5, 0.5, 0.1);

    unow.plot();

    // Boucle en temps
    double t = 0;
    double tmax = 0.01;

    while (t < tmax)
    {
        t += unow.get_dt();
        unext.step(unow);
        swap(unow, unext);
    }
    unow.plot();

    return 0;
}
```

Listing 2 – main.cpp

3.3 Fonction de tracé

On utilisera la fonction de tracé ci-dessous :

```
void Grid2d::plot() const
{
    // ouverture d'un fichier data.txt
    std::ofstream datafile("data.txt");
    // écriture des données
    for (int i = 0; i < nx; i++)
    {
        for (int j = 0; j < nx; j++)
        {
            datafile << i * dx + dx / 2 << " " << j * dx + dx / 2
    << " " << (*this)(i, j) << std::endl;
        }
        datafile << std::endl;
    }
    // fermeture du fichier
```

```
    datafile.close();
    // tracé avec gnuplot
    system("gnuplot -p -e \"splot 'data.txt' with lines\"");
}
```

Listing 3 – main.cpp

4 Consignes de développement

- Utiliser CMake pour la gestion du projet
- Versionner le code avec Git (dépot sur gitlab.unistra.fr)
- Implémenter la gestion mémoire avec compteur de références (à chaque appel du constructeur par copie, il faut incrémenter le compteur de références. Le destructeur ne libère la mémoire que si ref_count == 0).
- Programmer la fonction swap de façon à ce que la mémoire soit gérée de façon efficace !
- Valider avec le cas test : diffusion d'une plaque chauffée au centre
- Utiliser valgrind et l'option -fsanitize=address pour garantir que votre code n'a pas de fuites de mémoire ou de dépassement de tableaux.

5 Structure CMake minimale

```
cmake_minimum_required(VERSION 3.10)
project(HeatEquation)

set(CMAKE_CXX_STANDARD 17)

add_executable(heat_solver
    grid.hpp
  grid.cpp
    main.cpp
)
```

Listing 4 – CMakeLists.txt

6 Consignes Git

1. Créer un dépôt GitLab
2. Cloner le dépôt
3. Ajouter les fichiers sources et CMake
4. Pensez à commiter et pusher souvent !

7 Pointeur intelligent

On souhaite simplifier la gestion de la mémoire au moyen des smart pointers.

1. Remplacer le pointeur data de type double* par un pointeur intelligent de type shared_ptr<double>
2. Que signifie la syntaxe <double> dans ce contexte ?
3. Vérifier que la nouvelle programmation ne génère effectivement pas de fuite de mémoire au moyen de valgrind (puis l'option de compilation -fsanitize=address)

8 Template

On souhaite modifier la programmation précédente afin que la classe Grid2d puisse utiliser soit des double, soit des float. Pour cela la classe devient une classe template Grid2d<T> ou le type T sera soit « double », soit « float ».

1. Modifier le code pour prendre en compte le changement de conception. Peut-on encore utiliser le fichier grid.cpp ?
2. Tester la classe avec le même calcul, réalisé en simple ou double précision. Mesurer le temps de calcul dans les deux cas au moyen de la commande time. Même question en faisant varier la taille de la grille. La mesure correspond-il à la complexité de l'algorithme ?
3. Changer les options de compilation pour optimiser le temps de calcul. Mesurer le gain obtenu en simple et double précision en fonction de la taille de la grille.
4. Vérifier au moyen de valgrind (puis de l'options -fsanitize=address) que la mémoire occupée par le programme correspond à votre prévision, en fonction de la taille de la grille et de la précision (float ou double)