

Analyse numérique d'équations aux dérivées partielles par différences finies et implémentation optimisée pour le calcul haute performance

par Jean-Baptiste Gaillot

Introduction et explications

- But du projet
- Approche utilisée (partie mathématique et partie informatique)
- Outils utilisés (analyse numérique, algorithmique, programmation C, MPI, OpenMP, bibliothèque, ...)

Problèmes étudiés

- Équation de Poisson en dimension 1
- Équation de Poisson en dimension 2
- Équations des ondes en dimension 1
- Équations de la chaleur en dimension 2

Notations mathématiques

- N est le nombre de noeuds dans une direction
- $x_i := ih$ pour $i \in \{0, \dots, N\}$
- $u(x_i) \approx u_i$
- $u := \begin{pmatrix} u_1 & \cdots & u_{N-1} \end{pmatrix}^T$ est le vecteur de la solution approchée
- h est le pas de discrétisation en espace, h_t est le pas de discrétisation en temps
- E_h est l'erreur de troncature

Équation de Poisson en dimension 1

Équation de Poisson en dimension 1 – Analyse numérique

Présentation du problème

Soit $f :]0, 1[\rightarrow \mathbb{R}$ continue. Soit le problème suivant :

Trouver u de classe C^4 telle que :

$$\begin{cases} -u''(x) = f(x) & \forall x \in]0, 1[\\ u(0) = 0, u(1) = 0 \end{cases}$$

Si $f \equiv 1$, alors $u(x) = \frac{1}{2}x(1-x)$, ou si $f(x) = \pi^2 \sin(\pi x)$, alors $u(x) = \sin(\pi x)$.

Discrétisation

$$-u''(x) = \frac{1}{h^2} (-u(x+h) + 2u(x) - u(x-h)) + E_h$$

avec

$$E_h := \frac{1}{12} h^2 u^{(4)}(x + \theta h)$$

Schéma

$$\boxed{\frac{1}{h^2} (-u_{i+1} + 2u_i - u_{i-1}) = f_i .}$$

Schéma sous forme matricielle

$$\boxed{Au = f}$$

$$\Leftrightarrow \underbrace{\frac{1}{h^2} \begin{pmatrix} 2 & -1 & \cdot & \cdot & \cdot \\ -1 & 2 & -1 & \cdot & \cdot \\ \cdot & -1 & 2 & -1 & \cdot \\ \cdot & \cdot & -1 & 2 & -1 \\ \cdot & \cdot & \cdot & -1 & 2 \end{pmatrix}}_{=:A} \underbrace{\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_{N-1} \end{pmatrix}}_{=u} = \underbrace{\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_{N-1} \end{pmatrix}}_{=f}.$$

Équation de Poisson en dimension 1 – Analyse numérique

Schéma numérique

Remarques

- La valeur en un point du maillage dépend de valeurs d'au plus 3 points du maillage.
- A est une matrice creuse : elle comporte 3 diagonales (centrales).

Équation de Poisson en dimension 1 – Analyse numérique

Existence et unicité de la solution approchée

Proposition A est définie-positive et $Au = f$ admet une unique solution.

Démonstration Soit $x \in \mathbb{R}^N$, alors :

$$x^T Ax = \frac{1}{h^2} \left(\sum_{i=1}^{N-2} (x_i - x_{i+1})^2 + x_1^2 + x_{N-1}^2 \right) \geq 0.$$

Si $\sum_{i=1}^{N-2} (x_i - x_{i+1})^2 + x_1^2 + x_{N-1}^2 = 0$, alors $x = 0_{\mathbb{R}^N}$.

Équation de Poisson en dimension 1 – Analyse numérique

Consistance du schéma et majoration de l'erreur de troncature

Proposition Le schéma est consistant : $\lim_{h \rightarrow 0} |E_h| = 0$ et

$$|E_h| \leq \frac{1}{12} h^2 \sup_{x \in [0,1]} |f''(x)|.$$

Démonstration Comme on a $x + \theta h \in [0, 1]$ et $-u^{(4)} = -f''$, alors on peut majorer l'erreur de troncature comme ceci :

$$|u^{(4)}(x + \theta h)| = |f''(x + \theta h)| \leq \sup_{x \in [0,1]} |f''(x)|,$$

on obtient :

$$\left| \frac{1}{12} h^2 u^{(4)}(x + \theta h) \right| \leq \frac{1}{12} h^2 \sup_{x \in [0,1]} |f''(x)| \xrightarrow{h \rightarrow 0} 0.$$

Équation de Poisson en dimension 1 – Analyse numérique

Convergence du schéma et majoration de l'erreur locale

Proposition (*admise*) $\forall i, j \in \{0, \dots, N\} : a_{i,j}^{-1} \geq 0$.

Proposition Soit $e := (u_i - u(x_i))_{0 \leq i \leq N}$. Alors, le schéma est convergent :

$$\lim_{h \rightarrow 0} \|e\|_{\infty} = 0$$

et

$$\|e\|_{\infty} \leq \frac{1}{96} h^2 \sup_{x \in [0,1]} |f''(x)|.$$

Démonstration

Poser $\varepsilon := Ae$ ($A^{-1}\varepsilon = e$).

Remarques

- On vérifiera cette propriété avec un exemple qui utilise une implémentation utilisant une méthode de résolution directe.
- On a aussi montré que $\|A^{-1}\|_{\infty} \leq \frac{1}{8}$.

Équation de Poisson en dimension 1 – Analyse numérique

Convergence du schéma et majoration de l'erreur locale

Proposition Si u est de classe C^2 , alors le schéma est convergent : $\lim_{h \rightarrow 0} \|e\|_{\infty} = 0$.

Équation de Poisson en dimension 1 – Analyse numérique

Méthode de résolution itérative

Méthode de Jacobi : $Du^{(k+1)} = (E + F) u^{(k)} + f$.

Schéma

$$Du^{(k+1)} = \frac{1}{h^2} \begin{pmatrix} 2u_1^{(k+1)} \\ \vdots \\ \vdots \\ 2u_{N-1}^{(k+1)} \end{pmatrix}$$

et

$$(E + F) u^{(k)} + f = \begin{pmatrix} u_0^{(k)} + u_2^{(k)} + f_1 \\ \vdots \\ \vdots \\ u_{N-2}^{(k)} + u_N^{(k)} + f_{N-1} \end{pmatrix}.$$

Équation de Poisson en dimension 1 – Analyse numérique

Méthode de résolution itérative

$$Du^{(k+1)} = (E + F) u^{(k)} + f \Leftrightarrow \frac{2}{h^2} u_i^{(k+1)} = \frac{1}{h^2} \left(u_{i-1}^{(k)} + u_{i+1}^{(k)} \right) + f_i$$

$$\Leftrightarrow \boxed{u_i^{(k+1)} = \frac{1}{2} \left(u_{i-1}^{(k)} + u_{i+1}^{(k)} + h^2 f_i \right)}.$$

Équation de Poisson en dimension 1 – Analyse numérique

Méthode de résolution directe

Factorisation de Cholesky :

$$\ell_{i,i} = \sqrt{a_{i,i} - \sum_{k=1}^{i-1} \ell_{i,k}^2} \quad \text{et} \quad \ell_{i,j} = \frac{1}{\ell_{j,j}} \left(a_{i,j} - \sum_{k=1}^{j-1} \ell_{i,k} \ell_{j,k} \right).$$

On calcule d'abord en colonnes puis en lignes.

Équation de Poisson en dimension 1 – Analyse numérique

Méthode de résolution directe

Proposition Soit A une matrice tridiagonale, symétrique et définie-positive. Alors, la matrice L de la décomposition de Cholesky de A est bidiagonale inférieure.

Démonstration On a : Si $i > j + 1$, alors $a_{i,j} = 0$.

On calcule la colonne $j = 1$:

$$L|_{j=1} = \begin{pmatrix} \ell_{0,1} & \ell_{1,1} & 0 & \cdots & 0 \end{pmatrix}^T.$$

On calcule la colonne $j = 2$:

$$L|_{j=2} = \begin{pmatrix} 0 & \ell_{1,2} & \ell_{2,2} & 0 & \cdots & 0 \end{pmatrix}^T.$$

Ainsi de suite...

Équation de Poisson en dimension 1 – Analyse numérique

Méthode de résolution directe

On peut résoudre $Au = f$ en résolvant $Ly = f$ puis $L^T u = y$ avec :

$$y_1 = \frac{f_1}{\ell_{1,1}} \quad \text{et} \quad \text{pour } i \text{ de } 2 \text{ à } N-1 : y_i = \frac{f_i - \ell_{i,j-1}y_{i-1}}{\ell_{i,i}}$$

et

$$u_{N-1} = \frac{y_{N-1}}{\ell_{N-1,N-1}} \quad \text{et} \quad \text{pour } i \text{ de } N-2 \text{ à } 1 : u_i = \frac{y_i - \ell_{i,j+1}u_{i+1}}{\ell_{i,i}}.$$

Pour la suite, la fonction à approcher sera $f(x) := \pi^2 \sin(\pi x)$ et $\varepsilon := 1 \cdot 10^{-10}$.

Équation de Poisson en dimension 1 – Implémentation

Version de base

Étapes :

- créer des fonctions qui font le travail :
 - construire la matrice A (voir la fonction `construire_matrice`),
 - calculer le second membre f ,
 - calculer la solution approchée u (voir la fonction `resoudre_gauss`).

Commentaires

- Pour calculer u , on résout le système linéaire avec la méthode de Gauss.
- On note ces résultats :

N	?	?	?
$\ e\ _{\infty}$?	?	?
Temps d'exécution (s)	?	?	?

- A est de taille $O(N)$ et la méthode de Gauss est $O(N^3)$ donc la complexité algorithmique est $O(N^3)$.

Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution itérative en séquentiel

Fonction principale

```
void calculer_u_jacobi(double *f, double *u){  
    double h_carre = 1.0 / (N * N);  
    int nb_iteration_max = INT_MAX;  
    double norme = DBL_MAX;  
    double *u_anc; double *permut;  
  
    init_u_anc(&u_anc);  
  
    for (int iteration = 0 ; iteration < nb_iteration_max && norme > 1e-10 ;  
        iteration ++){  
        for (int j = 1 ; j < nb_pt - 1 ; j ++){  
            for (int i = 1 ; i < nb_pt - 1 ; i ++){  
                u[IDX(i, j)] = schema(f, u, u_anc, i, j);  
            }  
        }  
  
        norme = norme_infty_iteration(u, u_anc);  
        permut = u; u = u_anc; u_anc = permut; nb_iteration ++;  
    }  
  
    terminaison(&permut, &u, &u_anc);  
}
```

Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution itérative en séquentiel

Fonction qui applique le schéma à un point

```
static inline __attribute__((always_inline))  
double schema(double *f, double *u_anc, int i){  
    double res = 0.5 * ((u_anc[i - 1] + u_anc[i + 1]) + h_carre * f[i]);  
    return res;  
}
```


Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution itérative en séquentiel

Fonction pour calculer la norme infinie relative

```
static inline __attribute__((always_inline))
double norme_infty_iteration(double *u, double *u_anc){

    double norme_num = 0.0; double norme_deno = 0.0; double norme;

    for (int i = 0 ; i < nb_pt * nb_pt ; i++){
        double diff = fabs(u[i] - u_anc[i]);
        if (diff > norme_num){
            norme_num = diff;
        }
        if (fabs(u_anc[i]) > norme_deno){
            norme_deno = fabs(u_anc[i]);
        }
    }

    norme = norme_num / norme_deno;

    return norme;
}
```

Fonction pour terminer

```
void terminaison(double **permut, double **u, double **u_anc){

    if (nb_iteration % 2 != 0){
        *permut = *u; *u = *u_anc; *u_anc = *permut;
    }

    free(*u_anc);

}
```

Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution itérative en séquentiel

Commentaire On note ces résultats :

N	5	10	50	100	300	500
Nb. d'itérations	102	400	8506	31227	241002	617699
$\ e\ _\infty$	0.031916	0.008265	0.000329	0.000082	0.000007	0.000002
Temps d'exécution (s)	<0.01	<0.01	<0.01	0.01	0.14	0.54

Équation de Poisson en dimension 1 – Implémentation

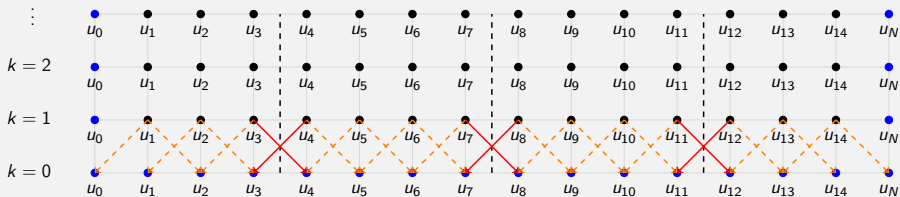
Version avec méthode de résolution itérative en parallèle avec OpenMP

Commentaire On ajoute une directive `for` dans la boucle de la fonction `calculer_u_jacobi` et une directive `for` dans la boucle du calcul de la norme relative.

Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution itérative en parallèle avec MPI

Illustration Schéma des dépendances pour 4 processus et $N = 15$:



Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution itérative en parallèle avec MPI

Fonctions pour MPI

```
void creer_topologie(){
    int tore = 0;
    dims = 0;

    MPI_Dims_create(nb_cpu, 1, &dims);

    MPI_Cart_create(MPI_COMM_WORLD, 1, &dims, &tore, 0, &comm_1D);

    MPI_Barrier(comm_1D);
}
```

```
void infos_processus(){
    i_debut = (coords * nb_pt) / dims;
    i_fin = ((coords + 1) * nb_pt) / dims - 1;
    nb_pt_div = i_fin - i_debut + 1;

    MPI_Barrier(comm_1D);
}
```

Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution itérative en parallèle avec MPI

Fonction principale

```
void calculer_u_jacobi(double *f_div, double *u_div){
    nb_iteration = 0; h_carre = 1.0 / pow(N, 2);
    int nb_iteration_max = INT_MAX; double norme = DBL_MAX;
    int i_boucle_debut; int i_boucle_fin;
    double *u_div_anc; double *permut;

    init_u_div_anc(&u_div_anc);
    for (int i = 0 ; i < nb_pt_div + 2 ; i ++){
        u_div[i] = 0.0;
    }

    infos_bornes_boucles(&i_boucle_debut, &i_boucle_fin);

    for (int iteration = 0 ; iteration < nb_iteration_max && norme > 1e-10 ;
        iteration ++){
        echanger_halos(u_div_anc);

        for (int i = i_boucle_debut ; i < i_boucle_fin ; i ++){
            u_div[i] = schema(f_div, u_div, u_div_anc, i);
        }

        norme = norme_infty_iteration(u_div, u_div_anc);
        permut = u_div; u_div = u_div_anc; u_div_anc = permut; nb_iteration ++;
    }

    terminaison(&permut, &u_div, &u_div_anc);
}
```

Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution itérative en parallèle avec MPI

Fonction pour obtenir les indices de départ et d'arrivé de la boucle principale

```
void infos_bornes_boucles(int *i_boucle_debut, int *i_boucle_fin){
    *i_boucle_debut = 1;
    *i_boucle_fin = nb_pt_div + 1;

    if (i_debut == 0){
        (*i_boucle_debut) ++;
    }
    if (i_fin == nb_pt - 1){
        (*i_boucle_fin) --;
    }
}
```

Fonction pour échanger les halos

```
void echanger_halos(double *u_div){
    // Envoi gauche, reception droite
    MPI_Sendrecv(&(u_div[1]), 1, MPI_DOUBLE, voisins[0], etiquette,
        &(u_div[nb_pt_div + 1]), 1, MPI_DOUBLE, voisins[1], etiquette, comm_1D,
        &statut);

    // Envoi droite, reception gauche
    MPI_Sendrecv(&(u_div[nb_pt_div]), 1, MPI_DOUBLE, voisins[1], etiquette,
        &(u_div[0]), 1, MPI_DOUBLE, voisins[0], etiquette, comm_1D, &statut);
}
```

Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution directe en séquentiel

Structure mat_2bandes

```
struct mat_2bandes{
    int N;
    double *diag; // taille n
    double *sous_diag; // taille n - 1
};
```

Fonction pour allouer la structure

```
void init_mat_2bandes(struct mat_2bandes *A){
    A -> N = N;
    A -> diag = (double *)malloc(idx_max * sizeof(double));
    A -> sous_diag = (double *)malloc((idx_max - 1) * sizeof(double));
}
```

Fonction pour libérer la structure

```
void liberer_mat_2bandes(struct mat_2bandes *A){
    free(A -> diag);
    free(A -> sous_diag);
}
```


Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution directe en séquentiel

Fonction pour obtenir la décomposition de Cholesky en utilisant la structure

```
void calculer_cholesky(struct mat_2bandes *L){
    h_carre = 1.0 / pow(N, 2);
    double alpha = 2.0 / h_carre;
    double beta = -1.0 / h_carre;

    (L -> diag)[0] = sqrt(alpha);
    (L -> sous_diag)[0] = beta / (L -> diag)[0];

    for (int i = 1 ; i < idx_max - 1 ; i++){
        (L -> diag)[i] = sqrt(alpha - pow((L -> sous_diag[i - 1]), 2));
        (L -> sous_diag)[i] = beta / (L -> diag[i]);
    }

    (L -> diag)[idx_max - 1] = sqrt(alpha - pow((L -> sous_diag[idx_max - 2]),
        2));
}
```

Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution directe en séquentiel

Test pour avoir un aperçu de la compression

Illustration de la structure mat_2bandes (exemple pour N petit) :

Structure mat2_bandes :

N = 7

diag	=	9.899495	8.573214	8.082904	7.826238	7.668116	7.560864
sous_diag	=	-4.949747	-5.715476	-6.062178	-6.260990	-6.390097	

Matrice reelle correspondante :

9.899495	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
-4.949747	8.573214	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	-5.715476	8.082904	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	-6.062178	7.826238	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	-6.260990	7.668116	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	-6.390097	7.560864	

Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution directe en séquentiel

Fonction principale

```
void resoudre_cholesky(double *f, double *u){  
    struct mat_2bandes L;  
    double *y = (double *)malloc(idx_max * sizeof(double));  
    u[0] = 0; u[nb_pt - 1] = 0;  
    init_mat_2bandes(&L);  
    calculer_cholesky(&L);  
    resoudre_cholesky_descente(&L, &(f[1]), y);  
    resoudre_cholesky_remontee(&L, y, &(u[1]));  
    liberer_mat_2bandes(&L);  
    free(y);  
}
```

Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution directe en séquentiel

Fonction pour résoudre $Ly = f$

```
void resoudre_cholesky_descente(struct mat_2bandes *L, double *f, double *y){
    y[0] = f[0] / (L -> diag)[0];

    for (int i = 1 ; i < idx_max ; i ++){
        y[i] = (f[i] - (L -> sous_diag)[i - 1] * y[i - 1]) / (L -> diag)[i];
    }
}
```

Fonction pour résoudre $L^T u = y$

```
void resoudre_cholesky_remontee(struct mat_2bandes *L, double *y, double *u){
    u[idx_max - 1] = y[idx_max - 1] / (L -> diag)[idx_max - 1];

    for (int i = idx_max - 2 ; i >= 0 ; i --){
        u[i] = (y[i] - (L -> sous_diag)[i] * u[i + 1]) / (L -> diag)[i];
    }
}
```

Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution directe en séquentiel

Commentaires

- Cette méthode est impossible à paralléliser à cause des dépendances.
- A possède $O(N)$ colonne. Pour chaque colonne, il y a $O(1)$ lignes à calculer. Pour chaque case, il y a $O(1)$ opérations. Donc la complexité algorithmique est $O(N)$.

On vérifie la proposition énoncée avec $f(x) = \pi^2 \sin(\pi x)$, d'après la proposition :

$$\|e\|_{\infty} \leq \frac{1}{96} \pi^4 h^2.$$

N	2	3	4	5	10	100	1000
$\ e\ _{\infty}$ max. th.	0.253669	0.112742	0.063417	0.040587	0.010146	0.000101	0.000001
$\ e\ _{\infty}$ obs.	0.233701	0.083678	0.053029	0.031916	0.008265	0.000082	0.000001

Équation de Poisson en dimension 1 – Implémentation

Comparaison des performances des méthodes

Discussion sur les performances...

Équation de Poisson en dimension 2

Équation de Poisson en dimension 2 – Analyse numérique

Présentation du problème

Soit $f :]0, 1[\times]0, 1[\rightarrow \mathbb{R}$ continue Soit $D :=]0, 1[\times]0, 1[$. Soit le problème suivant :

Trouver u de classe C^4 telle que :

$$\begin{cases} -\Delta u(x, y) = f(x, y) & \forall (x, y) \in D \\ u(x, y) = 0 & \forall (x, y) \in \partial D \end{cases}.$$

Si $f(x, y) = \sin(2\pi x) \sin(2\pi y)$, alors $u(x, y) = \frac{1}{8\pi^2} \sin(2\pi x) \sin(2\pi y)$.

Discrétisation

$$\begin{aligned} & u(x + h_x, y) + u(x - h_x, y) \\ = & 2u(x, y) + h_x^2 \frac{\partial^2 u}{\partial x^2}(x, y) + \frac{1}{24} h_x^4 \left(\frac{\partial^4 u}{\partial x^4}(x + \theta_{x+} h_x, y) + \frac{\partial^4 u}{\partial x^4}(x + \theta_{x-} h_x, y) \right), \\ -\frac{\partial^2 u}{\partial x^2}(x, y) = & \frac{1}{h_x^2} (-u(x - h_x, y) + 2u(x, y) - u(x + h_x, y)) + \frac{1}{12} h_x^2 \frac{\partial^4 u}{\partial x^4}(x + \theta_x h_x, y), \\ -\frac{\partial^2 u}{\partial y^2}(x, y) = & \frac{1}{h_y^2} (-u(x, y - h_y) + 2u(x, y) - u(x, y + h_y)) + \frac{1}{12} h_y^2 \frac{\partial^4 u}{\partial y^4}(x, y + \theta_y h_y) \end{aligned}$$

Équation de Poisson en dimension 2 – Analyse numérique

Schéma numérique

$$-\Delta u(x, y) = \frac{1}{h_x^2} \delta_x^2 + \frac{1}{h_y^2} \delta_y^2 + E_{h_x, h_y}$$

avec :

$$\delta_x^2 := -u(x - h_x, y) + 2u(x, y) - u(x + h_x, y),$$

$$\delta_y^2 := -u(x, y - h_y) + 2u(x, y) - u(x, y + h_y)$$

et

$$E_{h_x, h_y} := \frac{1}{12} \left(h_x^2 \frac{\partial^4 u}{\partial x^4}(x + \theta_x h_x, y) + h_y^2 \frac{\partial^4 u}{\partial y^4}(x, y + \theta_y h_y) \right).$$

Schéma

$$\frac{1}{h^2} (-u_{i-1,j} - u_{i,j-1} + 2u_{i,j} - u_{i+1,j} - u_{i,j+1}) = f_{i,j}.$$

Schéma sous forme matricielle

$$Au = f$$

$$\Leftrightarrow \frac{1}{h^2} \underbrace{\begin{pmatrix} 4 & -1 & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ -1 & 4 & -1 & \cdot & -1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & -1 & 4 & \cdot & \cdot & -1 & \cdot & \cdot & \cdot \\ -1 & \cdot & \cdot & 4 & -1 & \cdot & -1 & \cdot & \cdot \\ \cdot & -1 & \cdot & -1 & 4 & -1 & \cdot & -1 & \cdot \\ \cdot & \cdot & -1 & \cdot & -1 & 4 & \cdot & \cdot & -1 \\ \cdot & \cdot & \cdot & -1 & \cdot & \cdot & 4 & -1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & -1 & \cdot & -1 & 4 & -1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & -1 & 4 \end{pmatrix}}_{=:A} \underbrace{\begin{pmatrix} u_{1,1} \\ u_{2,1} \\ u_{3,1} \\ u_{1,2} \\ u_{2,2} \\ u_{3,2} \\ u_{1,3} \\ u_{2,3} \\ u_{3,3} \end{pmatrix}}_{=:u} = \underbrace{\begin{pmatrix} f_{1,1} \\ f_{2,1} \\ f_{3,1} \\ f_{1,2} \\ f_{2,2} \\ f_{3,2} \\ f_{1,3} \\ f_{2,3} \\ f_{3,3} \end{pmatrix}}_{=:f}.$$

Équation de Poisson en dimension 2 – Analyse numérique

Schéma numérique

Schéma sous forme matricielle par blocs

$$\underbrace{\frac{1}{h^2} \begin{pmatrix} \boxed{M} & \boxed{-I} & . & . \\ \boxed{-I} & \ddots & \ddots & . \\ . & \ddots & \ddots & \boxed{-I} \\ . & . & \boxed{-I} & \boxed{M} \end{pmatrix}}_{=A} \underbrace{\begin{pmatrix} u_1 \\ \vdots \\ \vdots \\ u_{N-1} \end{pmatrix}}_{=u} = \underbrace{\begin{pmatrix} f_1 \\ \vdots \\ \vdots \\ f_{N-1} \end{pmatrix}}_{=f}$$

avec

$$M := \begin{pmatrix} 4 & -1 & . & . \\ -1 & \ddots & \ddots & . \\ . & \ddots & \ddots & -1 \\ . & . & -1 & 4 \end{pmatrix}.$$

Remarques

- La valeur en un point du maillage dépend de valeurs d'au plus 5 points du maillage.
- A est une matrice creuse : elle comporte 5 diagonales (dont 3 centrales) et 3 blocs de diagonales, où les blocs sont M et $-I$.

Équation de Poisson en dimension 2 – Analyse numérique

Existence et unicité de la solution approchée

Proposition A est définie-positive et $Au = f$ admet une unique solution.

Démonstration Soit $x \in \mathbb{R}^{(N-1) \times (N-1)}$ avec $x = (x_j)_{1 \leq j \leq N-1}$ et $x_j = (x_{i,j})_{1 \leq i \leq N-1}$, alors :

$$x^T A x = \frac{1}{h^2} \left(\sum_{j=1}^{N-1} x_j^T M x_j - 2 \sum_{j=1}^{N-2} x_j^T x_{j+1} \right).$$

De plus,

$$x_j^T M x_j = x_j^T (h^2 B + 2I) x_j = h^2 x_j^T B x_j + 2 \sum_{i=1}^{N-1} x_{i,j}^2 = h^2 x_j^T B x_j + 2 \|x_j\|^2$$

Équation de Poisson en dimension 2 – Analyse numérique

Existence et unicité de la solution approchée

Démonstration (*suite*) avec

$$B := \frac{1}{h^2} \begin{pmatrix} 2 & -1 & . & . \\ -1 & \ddots & \ddots & . \\ . & \ddots & \ddots & -1 \\ . & . & -1 & 2 \end{pmatrix}.$$

Donc on obtient :

$$x^T A x = \frac{1}{h^2} \left(\sum_{j=1}^{N-2} \|x_j + x_{j+1}\|^2 + \|x_1\|^2 + \|x_{N-1}\|^2 + h^2 \sum_{j=1}^{N-1} x_j^T B x_j \right)$$

et B est définie-positive donc $\forall x_j \in \mathbb{R}^N : x_j^T B x_j \geq 0$, donc $x^T A x \geq 0$.

Équation de Poisson en dimension 2 – Analyse numérique

Consistance du schéma et majoration de l'erreur de troncature

Notation Soit $d \in \{1, \dots, 4\}$. Alors,

$$C_{u,d} := \max \left\{ \sup_{(x,y) \in [0,1]^2} \left| \frac{\partial^d u}{\partial x^d}(x,y) \right|, \sup_{(x,y) \in [0,1]^2} \left| \frac{\partial^d u}{\partial y^d}(x,y) \right| \right\}.$$

Proposition Le schéma est consistant : $\lim_{h \rightarrow 0} |E_h| = 0$ et $|E_h| \leq \frac{1}{6} h^2 |C_{u,4}|$.

Démonstration Majoration...

Remarque Le schéma est d'ordre 2 pour x et pour y .

Proposition (*admise*) Soit $e_j := (\|u_j - u(x_j)\|_\infty)_{0 \leq j \leq N}$. Alors, le schéma utilisé est convergent :

$$\forall j \in \{1, \dots, N-1\} : \lim_{h \rightarrow 0} \|e_j\|_\infty = 0$$

et

$$\exists C > 0, \forall j \in \{1, \dots, N-1\} : \|e_j\|_\infty \leq Ch^2 (C_{u,4} + hC_{u,3}).$$

Équation de Poisson en dimension 2 – Analyse numérique

Méthode de résolution itérative

$$D = \frac{1}{h^2} \begin{pmatrix} \boxed{D_0} & . & . & . \\ . & \ddots & . & . \\ . & . & \ddots & . \\ . & . & . & \boxed{D_0} \end{pmatrix} \quad \text{où } D_0 := 4I,$$

$$E + F = \frac{1}{h^2} \begin{pmatrix} \boxed{D_\star} & \boxed{I} & . & . \\ \boxed{I} & \ddots & \ddots & . \\ . & \ddots & \ddots & \boxed{I} \\ . & . & \boxed{I} & \boxed{D_\star} \end{pmatrix} \quad \text{où } D_\star := \begin{pmatrix} . & 1 & . & . \\ 1 & . & \ddots & . \\ . & \ddots & . & 1 \\ . & . & 1 & . \end{pmatrix}.$$

Équation de Poisson en dimension 2 – Analyse numérique

Méthode de résolution itérative

$$\left(Du^{(k+1)}\right)_{i,j} = \frac{1}{h^2} 4u_{i,j}^{(k+1)},$$

$$\left((E + F)u^{(k)} + f\right)_{i,j} = \frac{1}{h^2} \left(u_{i,j-1}^{(k)} + u_{i-1,j}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)}\right) + f_{i,j},$$

$$\Leftrightarrow \boxed{u_{i,j}^{(k+1)} = \frac{1}{4} \left(u_{i-1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)} + h^2 f_{i,j}\right)}.$$

Équation de Poisson en dimension 2 – Analyse numérique

Méthode de résolution directe

Proposition Soit A une matrice avec N diagonales inférieures, symétrique et définie-positive. Alors, la matrice L de la décomposition de Cholesky de A possède N diagonales inférieures.

On remarque que la structure de A est telle que :

$$a_{i,j} = \begin{cases} \alpha & \text{si } i = j \\ \beta & \text{si } i = j + 1 \text{ et } j \not\equiv 0 \pmod{N-1} \\ \gamma & \text{si } i = j + N - 1 \\ 0 & \text{sinon} \end{cases}.$$

avec

$$\alpha := \frac{4}{h^2} \quad \text{et} \quad \beta := \gamma := -\frac{1}{h^2}.$$

Équation de Poisson en dimension 2 – Analyse numérique

Méthode de résolution directe

Soit $d := i - j$ la diagonale de A ($i = d + j$, $d \in \{0, \dots, N - 1\}$). On calcule : pour j de 1 à $(N - 1)^2$ puis pour d de 0 à $N - 1$. Si $d = 0$, alors on calcule $\ell_{i,i}$, sinon, on calcule $\ell_{i,j}$. Avec la structure de A , on obtient :

$$\left\{ \begin{array}{ll} \ell_{i,i} = \sqrt{\alpha - \sum_{k=\max\{1, j-N+d+1\}}^{i-1} \ell_{i,k}^2} & \text{si } d = 0 \\ \ell_{i,j} = \left(a_{i,j} - \sum_{k=\max\{1, j-N+d+1\}}^{j-1} \ell_{i,k} \ell_{j,k} \right) / \ell_{j,j} & \text{si } d > 0. \end{array} \right.$$

Équation de Poisson en dimension 2 – Analyse numérique

Méthode de résolution directe

Remarque Si A était pleine, la complexité algorithmique du calcul de L aurait été $O(N^6)$. Ici, on calcule $O(N^2)$ colonnes comportants $O(N)$ diagonales. Le calcul d'une case est $O(N)$. Donc on a réduit la complexité algorithmique du calcul de L à $O(N^4)$.

Équation de Poisson en dimension 2 – Analyse numérique

Méthode de résolution directe

On peut résoudre $Au = f$ en résolvant $Ly = f$ puis $L^T u = y$ avec :

$$y_1 = \frac{f_1}{\ell_{1,1}} \quad \text{et} \quad \text{pour } i \text{ de } 2 \text{ à } (N-1)^2 : y_i = \left(f_i - \sum_{k=\max\{1, i-N+1\}}^{i-1} \ell_{i,k} y_k \right) / \ell_{i,i}$$

et

$$u_{(N-1)^2} = \frac{y_{(N-1)^2}}{\ell_{(N-1)^2, (N-1)^2}} \quad \text{et} \quad \text{pour } i \text{ de } (N-1)^2 - 1 \text{ à } 1 : u_i = \left(y_i - \sum_{k=i+1}^{\min\{i+N-1, (N-1)^2\}} \ell_{k,i} u_k \right) / \ell_{i,i}.$$

Pour la suite, la fonction à approcher sera $f(x, y) := \sin(2\pi x) \sin(2\pi y)$ et $\varepsilon := 1 \cdot 10^{-10}$.

Équation de Poisson en dimension 2 – Implémentation

Version de base

Étapes :

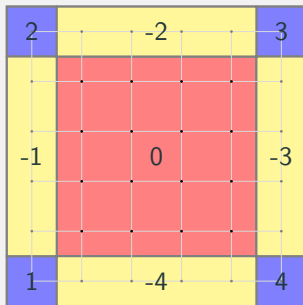
- créer des fonctions qui font le travail :
 - construire la matrice A (voir les fonctions `connaitre_bord` et `construire_matrice`),
 - calculer le second membre f ,
 - calculer la solution approchée u (voir la fonction `resoudre_gauss`).

Équation de Poisson en dimension 2 – Implémentation

Version de base

Commentaires

- Tout les tableaux utilisés sont linéarisés pour garantir la contiguité.
- Les numéros du type de bord de la fonction `connaître_bord` sont les suivants :



Commentaires (suite)

- Pour calculer u , on résout le système linéaire avec la méthode de Gauss.
- On note ces résultats :

N	10	50	100
$\ e\ _{\infty}$	0.00038444	0.00001661	0.00000417
Temps d'exécution (s)	<0.1	4.1	278.9

- A est de taille $O(N^2)$ et la méthode de Gauss est $O(N^3)$ donc la complexité algorithmique est $O(N^6)$.

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution itérative en séquentiel

```
static inline __attribute__((always_inline))
double schema(double *f, double *u_anc, int i, int j){
    double res = 0.25 * (
        u_anc[IDX(i - 1, j)]
        + u_anc[IDX(i, j - 1)]
        + u_anc[IDX(i + 1, j)]
        + u_anc[IDX(i, j + 1)]
        + h_carre * f[IDX(i, j)]);
    return res;
}
```

Commentaire On note ces résultats :

N	10	50	100	300	500	700
Nb. d'itérations	102	2298	8506	66569	171980	320379
$\ e\ _\infty$	0.00038444	0.00001661	0.00000417	0.00000046	0.00000015	0.00000005
Temps d'exécution (s)	<0.1	<0.1	0.1	4.8	34.6	128.4

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution itérative en parallèle avec OpenMP

Commentaires

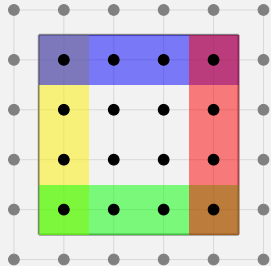
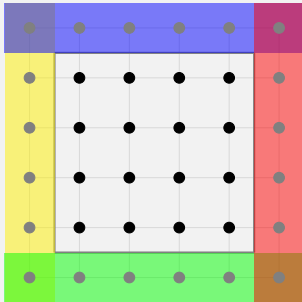
- On ajoute une directive `for` dans la boucle interne de la fonction `calculer_u_jacobi` et une directive `for` dans la boucle du calcul de la norme relative.
- On note ces résultats du temps d'exécution (en s) en fonction de N et du nombre de threads :

↓ Nombre de threads $N \rightarrow$	300	500	700
1	5.1	35.7	130.0
2	4.2	21.4	80.0
4	3.1	13.3	53.5
6	3.6	18.0	62.5
8	4.0	15.9	56.3

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution itérative en parallèle avec MPI

Illustration Schéma de la structure de `u_div` :



Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution itérative en parallèle avec MPI

Fonctions MPI

```
void creer_types(){  
    int taille_send[2] = {nb_pt_div_j + 2, nb_pt_div_i + 2};  
    int sous_taille_send[2] = {nb_pt_div_j, nb_pt_div_i};  
    int debut_send[2] = {1, 1};  
  
    MPI_Type_contiguous(nb_pt_div_i, MPI_DOUBLE, &ligne);  
    MPI_Type_vector(nb_pt_div_j, 1, nb_pt_div_i + 2, MPI_DOUBLE, &colonne);  
    MPI_Type_create_subarray(2, taille_send, sous_taille_send, debut_send,  
        MPI_ORDER_C, MPI_DOUBLE, &bloc_send);  
  
    MPI_Type_commit(&ligne);  
    MPI_Type_commit(&colonne);  
    MPI_Type_commit(&bloc_send);  
  
    MPI_Barrier(comm_2D);  
}
```

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution itérative en parallèle avec MPI

```
void echanger_halos(double *u_div){  
    // Envoi haut, reception bas  
    MPI_Sendrecv(&(u_div[IDX(1, nb_pt_div_j)]), 1, ligne, voisins[1],  
        etiquette, &(u_div[IDX(1, 0)]), 1, ligne, voisins[3], etiquette, comm_2D,  
        &statut);  
  
    // Envoi bas, reception haut  
    MPI_Sendrecv(&(u_div[IDX(1, 1)]), 1, ligne, voisins[3], etiquette,  
        &(u_div[IDX(1, nb_pt_div_j + 1)]), 1, ligne, voisins[1], etiquette,  
        comm_2D, &statut);  
  
    // Envoi gauche, reception droite  
    MPI_Sendrecv(&(u_div[IDX(1, 1)]), 1, colonne, voisins[0], etiquette,  
        &(u_div[IDX(nb_pt_div_i + 1, 1)]), 1, colonne, voisins[2], etiquette,  
        comm_2D, &statut);  
  
    // Envoi droite, reception gauche  
    MPI_Sendrecv(&(u_div[IDX(nb_pt_div_i, 1)]), 1, colonne, voisins[2],  
        etiquette, &(u_div[IDX(0, 1)]), 1, colonne, voisins[0], etiquette, comm_2D,  
        &statut);  
}
```


Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution itérative en parallèle avec MPI

Commentaires

- Pour regrouper les résultats sur le rang 0, on utilise un type dérivé `bloc_recv` créée dynamiquement par le rang 0 (voir la fonction `regrouper_u`).
- On note ces résultats du temps d'exécution (en s) en fonction de N et du nombre de processus :

↓ Nombre de processus $N \rightarrow$	300	500	700
1	5.3	37.1	133.4
2	3.0	20.8	76.8
4	1.9	12.8	47.1
6	2.7	18.1	62.5
8	2.5	18.9	110.4

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution itérative en parallèle avec MPI

Version avec un mode de communication non bloquant

Commentaires

- Dès que la communication est lancée, on fait les calculs sur l'intérieur du sous-domaine (en excluant les bords locaux), après on vérifie / attend que la communication soit terminée et on fait les calculs sur les bords locaux avec la fonction `test_fin_echange_halos`.
- Pour calculer sur les bords du sous-domaine (2 bandes verticales, 2 bandes horizontales et 4 coins), on utilise la fonction `calculer_u_jacobi_bords`.

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution directe en séquentiel

Structure mat_Nbandes

```
struct mat_Nbandes{
    int N;
    double **diags;
};
```

Fonction pour allouer la structure

```
void init_mat_Nbandes(struct mat_Nbandes *A){
    A -> N = N;
    A -> diags = (double **)malloc(N * sizeof(double *));
    for (int i = 0 ; i < N ; i ++){
        (A -> diags)[i] = (double *)malloc((idx_max - i) * sizeof(double));
    }
}
```

Fonction pour libérer la structure

```
void liberer_mat_Nbandes(struct mat_Nbandes *A){
    int N = A -> N;
    for (int i = 0 ; i < N ; i ++){
        free((A -> diags)[i]);
    }
    free(A -> diags);
}
```

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution directe en séquentiel

Fonction pour obtenir la décomposition de Cholesky en utilisant la structure

```
void calculer_cholesky(struct mat_Nbandes *L){  
    h_carre = 1.0 / pow(N, 2);  
    double alpha = 4.0 / h_carre;  
  
    for (int j = 0 ; j < idx_max ; j ++){  
        for (int d = 0 ; d < N && j + d < idx_max ; d ++){  
            int i = d + j;  
  
            if (d == 0){  
                (L -> diags)[0][j] = alpha;  
                for (int k = max(0, j - N + d + 1) ; k < i ; k ++){  
                    int d_1 = i - k;  
                    (L -> diags)[0][j] -= pow((L -> diags)[d_1][k], 2);  
                }  
                (L -> diags)[0][j] = sqrt((L -> diags)[0][j]);  
            }  
  
            else{  
                (L -> diags)[d][j] = valeur_a(i, j);  
                for (int k = max(0, j - N + d + 1) ; k < j ; k ++){  
                    int d_1 = i - k;  
                    int d_2 = j - k;  
                    (L -> diags)[d][j] -= (L -> diags)[d_1][k] * (L -> diags)[d_2][k];  
                }  
                (L -> diags)[d][j] /= (L -> diags)[0][j];  
            }  
        }  
    }  
}
```

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution directe en séquentiel

Fonction pour obtenir la valeur de $a_{i,j}$ en fonction des paramètres i et j

```
static inline __attribute__((always_inline)) double valeur_a(int i, int j){
    double res;
    if (i == j){
        res = 4.0 / h_carre;
    }
    else if (i == j + 1 && j % (N - 1) != (N - 2)){
        res = -1.0 / h_carre;
    }
    else if (i == j + N - 1){
        res = -1.0 / h_carre;
    }
    else{
        res = 0.0;
    }
    return res;
}
```

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution directe en séquentiel

Test pour avoir un aperçu de la compression

Illustration de la structure mat_Nbandes (exemple pour N petit) :

Structure mat_Nbandes :

N = 4

```
diag[0] = 8.0000  7.7460  7.7287  7.7275  7.3829  7.3668  7.6995  7.3261  7.3139
diag[1] = -2.0000 -2.0656 -0.1380 -2.2184 -2.3331 -0.2074 -2.2717 -2.4056
diag[2] = 0.0000 -0.5164 -0.5521 -0.0370 -0.6222 -0.6863 -0.0585
diag[3] = -2.0000 -2.0656 -2.0702 -2.0705 -2.1672 -2.1719
```

Matrice réelle correspondante :

8.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
-2.0000	7.7460	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	-2.0656	7.7287	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
-2.0000	-0.5164	-0.1380	7.7275	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	-2.0656	-0.5521	-2.2184	7.3829	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	-2.0702	-0.0370	-2.3331	7.3668	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	-2.0705	-0.6222	-0.2074	7.6995	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	-2.1672	-0.6863	-2.2717	7.3261	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	-2.1719	-0.0585	-2.4056	7.3139

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution directe en séquentiel

Fonction principale

```
void resoudre_cholesky(double *f, double *u){  
    struct mat_Nbandes L;  
    double *y, *u_int, *f_int;  
  
    init_u_bord(u);  
    u_int = (double *)malloc(idx_max * sizeof(double));  
    f_int = (double *)malloc(idx_max * sizeof(double));  
    extraire_interieur(u, u_int, nb_pt);  
    extraire_interieur(f, f_int, nb_pt);  
    y = (double *)malloc(idx_max * sizeof(double));  
  
    init_mat_Nbandes(&L);  
    calculer_cholesky(&L);  
  
    resoudre_cholesky_descente(&L, f_int, y);  
    resoudre_cholesky_remontee(&L, y, u_int);  
    inserer_interieur(u_int, u, nb_pt);  
  
    liberer_mat_Nbandes(&L);  
    free(u_int);  
    free(f_int);  
    free(y);  
}
```

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution directe en séquentiel

Fonction pour résoudre $Ly = f$

```
void resoudre_cholesky_descente(struct mat_Nbandes *L, double *f, double *y){
    y[0] = f[0] / (L -> diags)[0][0];

    for (int i = 1 ; i < idx_max ; i++){
        y[i] = f[i];
        for (int k = max(0, i - N + 1) ; k < i ; k++){
            int d = i - k;
            y[i] -= (L -> diags)[d][k] * y[k];
        }
        y[i] /= (L -> diags)[0][i];
    }
}
```

Fonction pour résoudre $L^T u = y$

```
void resoudre_cholesky_remontee(struct mat_Nbandes *L, double *y, double *u){
    u[idx_max - 1] = y[idx_max - 1] / (L -> diags)[0][idx_max - 1];

    for (int i = idx_max - 2 ; i >= 0 ; i--){
        u[i] = y[i];
        for (int k = i + 1 ; k < min(i + N, idx_max) ; k++){
            int d = k - i;
            u[i] -= (L -> diags)[d][i] * u[k];
        }
        u[i] /= (L -> diags)[0][i];
    }
}
```


Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution directe en séquentiel

Commentaires

- Comme on calcule u sur l'intérieur, on fait la résolution avec $u|_{\text{int}}$ et $f|_{\text{int}}$. On utilise les fonctions `extraire_interieur` pour extraire l'intérieur d'une matrice linéarisée et `insérer_interieur` pour remettre l'intérieur d'une matrice linéarisée dans la matrice initiale.

- On note ces résultats :

N	10	50	100	300	500	700
$\ e\ _{\infty}$	0.00038444	0.00001661	0.00000417	0.00000046	0.00000017	0.00000009
Temps d'exécution (s)	<0.1	<0.1	<0.1	4.6	35.8	383.5

- A possède $O(N^2)$ colonne. Pour chaque colonne, il y a $O(N)$ lignes à calculer. Pour chaque case, il y a $O(N)$ opérations. Donc la complexité algorithmique est $O(N^4)$.

Bibliothèque Cholmod

Étapes :

- créer une fonction pour définir la structure de matrice creuse en créant des tableaux :
 - `lignes` qui contient les indices des lignes où se trouve une valeur non nulle,
 - `valeurs` qui contient les valeurs aux indices stockés,
 - `offsets` qui contient le nombre de valeurs non nulles d'une colonne,(voir les fonctions `construire_matrice_creuse` et `connaitre_bord`).
- créer une fonction qui fait le travail

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution directe en séquentiel

Fonction principale

```
void resoudre(cholmod_sparse *A, double *f, double *u){
    h_carre = 1.0 / pow(N, 2);
    double *f_int = (double *)malloc(idx_max * sizeof(double));
    double *u_int = (double *)malloc(idx_max * sizeof(double));

    extraire_interieur(f, f_int, nb_pt);
    extraire_interieur(u, u_int, nb_pt);

    cholmod_dense *f_dense = cholmod_allocate_dense(A -> nrow, 1, A -> nrow,
        CHOLMOD_REAL, &c);
    memcpy(f_dense -> x, f_int, A -> nrow * sizeof(double));

    cholmod_factor *L = cholmod_analyze(A, &c);
    cholmod_factorize(A, L, &c);

    cholmod_dense *u_dense = cholmod_solve(CHOLMOD_A, L, f_dense, &c);
    memcpy(u_int, u_dense -> x, A -> nrow * sizeof(double));

    inserer_interieur(u_int, u, nb_pt);

    cholmod_free_factor(&L, &c);
    cholmod_free_dense(&f_dense, &c);
    cholmod_free_dense(&u_dense, &c);

    free(f_int);
    free(u_int);
}
```

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution directe en séquentiel

Commentaires

- Le nombre d'éléments non nuls de A est de $(5N + 1)(N - 3) + 12$.
- On note ces résultats :

N	1000	2000	3000	4000	5000
$\ e\ _{\infty}$	0.00000004	0.00000001	<0.00000001	<0.00000001	<0.00000001
Temps d'exécution (s)	0.7	15.7	39.3	79.3	174.5

Équation de Poisson en dimension 2 – Implémentation

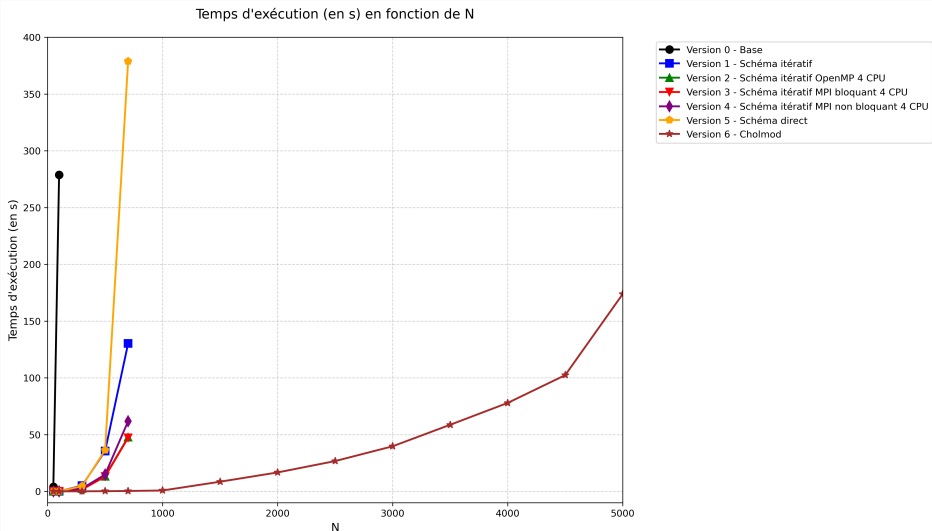
Comparaison des performances des méthodes

Discussion sur les performances...

- La version de base est inutilisable en pratique.
- Les versions itératives, en particulier avec le parallélisme, donnent de bien meilleurs résultats.
- La version avec la bibliothèque `cholmod` donne d'excellents résultats.

Équation de Poisson en dimension 2 – Implémentation

Comparaison des performances des méthodes



Équation des ondes en dimension 1

Soient $L, T > 0, D :=]0, L[$. Soit le problème suivant :

Trouver u de classe C^2 telle que :

$$\left\{ \begin{array}{l} \frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0 \quad x \in D \quad \forall t \in]0, T], c > 0 \\ u(x) = 0 \quad \forall x \in \partial D \quad \forall t \in [0, T] \\ u(x, 0) =: u_0(x) \quad \forall x \in D \\ \frac{\partial u}{\partial t}(x, 0) =: u_1(x) \quad \forall x \in D \end{array} \right. .$$

Si $u_0(x) = \sin(\pi x)$ et $u_1(x) = 0$, alors $u(x, t) = \cos(\pi x) \sin(\pi x)$.

Discrétisation

$$u(x, t + h_t) + u(x, t - h_t) = 2u(x, t) + h_t^2 \frac{\partial^2 u}{\partial t^2}(x, t) + \frac{1}{24} h_t^4 \left(\frac{\partial^4 u}{\partial t^4}(x, t + \theta_+ h) + \frac{\partial^4 u}{\partial t^4}(x, t + \theta_- h) \right),$$

$$\frac{\partial^2 u}{\partial t^2}(x, t) = \frac{1}{h_t^2} (u(x, t + h_t) - 2u(x, t) + u(x, t - h_t)) + E_{h_t}.$$

avec

$$E_{h_t} := -\frac{1}{12} h_t^2 \frac{\partial^4 u}{\partial t^4}(x, t + \theta h_t).$$

Schéma

$$\begin{cases} u_i^1 = h_t u_1(x_i) + u_0(x_i) \\ u_i^{k+1} = -u_i^{k-1} + 2 \left(1 - \frac{c^2 h_t^2}{h^2} \right) u_i^k + \frac{c^2 h_t^2}{h^2} (u_{i-1}^k + u_{i+1}^k) \quad \text{si } k > 0 \end{cases}.$$

Remarques

- Le schéma est explicite : il dépend de valeurs connues et ne nécessite pas de résoudre un système linéaire.
- Pour $k > 0$, le schéma dépend de valeurs en $k - 1$ et en $k - 2$.

Proposition (*admise*) Le schéma est convergent si $c \frac{h_t}{h} \leq 1$.

Remarques

- On vérifiera cette propriété avec un exemple.
- Cette proposition s'appelle la condition de Courant-Friedrich-Levy (CFL).

Équation de la chaleur en dimension 2

Équation de la chaleur en dimension 2 – Analyse numérique

Présentation du problème

Soient $L, T > 0, f :]0, L[\times]0, L[\times]0, T] \rightarrow \mathbb{R}$ continue et bornée,
 $D :=]0, L[\times]0, L[$. Soit le problème suivant :

Trouver u de classe C^2 telle que :

$$\left\{ \begin{array}{l} \frac{\partial u}{\partial t} - a\Delta u = f(x, y, t) \quad \forall (x, y) \in D \quad \forall t \in]0, T], a > 0 \\ u(x, y, t) = 0 \quad \forall (x, y) \in \partial D \quad \forall t \in [0, T] \\ u(x, y, 0) =: u_0(x, y) \quad \forall (x, y) \in D \end{array} \right. .$$

Si $f(x, y, t) = (-\lambda + 2a\pi^2) \sin(\pi x) \sin(\pi y) e^{-\lambda t}$ et $u_0(x, y) = \sin(\pi x) \sin(\pi y)$,
alors $u(x, y, t) = \sin(\pi x) \sin(\pi y) e^{-\lambda t}$.

Discrétisation

$$\frac{\partial u}{\partial t}(x, y, t) = \frac{1}{h_t} (-u(x, y, t) + u(x, y, t + h_t)) + E_{h_t}$$

avec :

$$E_{h_t} := -\frac{1}{2} h_t \frac{\partial^2 u}{\partial t^2}(x, y, t + \theta_t h_t).$$

Schéma

$$u_{i,j}^{k+1} = \alpha u_{i,j}^k + \beta (u_{i-1,j}^k + u_{i,j-1}^k + u_{i+1,j}^k + u_{i,j+1}^k) + h_t f_{i,j}^k$$

avec :

$$\alpha := 1 - \frac{4ah_t}{h^2} \quad \text{et} \quad \beta := \frac{ah_t}{h^2}.$$

Discrétisation

$$\frac{\partial u}{\partial t}(x, y, t + h_t) = \frac{1}{h_t} (u(x, y, t + h_t) - u(x, y, t)) + E_{h_t}$$

avec :

$$E_{h_t} := \frac{1}{2} h_t \frac{\partial^2 u}{\partial t^2}(x, y, t + (\theta_t + 1) h_t).$$

Schéma

$$\alpha u_{i,j}^{k+1} + \beta \left(u_{i-1,j}^{k+1} + u_{i,j-1}^{k+1} + u_{i+1,j}^{k+1} + u_{i,j+1}^{k+1} \right) = u_{i,j}^k + h_t f_{i,j}^{k+1}$$

avec :

$$\alpha := 1 + \frac{4ah_t}{h^2} \quad \text{et} \quad \beta := -\frac{ah_t}{h^2}.$$

Schéma sous forme matricielle par blocs

$$Au^{k+1} = b^k \Leftrightarrow \underbrace{\begin{pmatrix} \boxed{M} & \boxed{N} & \cdot & \cdot \\ \boxed{N} & \ddots & \ddots & \cdot \\ \cdot & \ddots & \ddots & \boxed{N} \\ \cdot & \cdot & \boxed{N} & \boxed{M} \end{pmatrix}}_{=:A} \underbrace{\begin{pmatrix} u_1^{k+1} \\ \vdots \\ \vdots \\ u_{N-1}^{k+1} \end{pmatrix}}_{=:u^{k+1}} = \underbrace{\begin{pmatrix} b_1^{k+1} \\ \vdots \\ \vdots \\ b_{N-1}^{k+1} \end{pmatrix}}_{=:b^{k+1}}$$

avec

$$M := \begin{pmatrix} \alpha & \beta & \cdot & \cdot \\ \beta & \ddots & \ddots & \cdot \\ \cdot & \ddots & \ddots & \beta \\ \cdot & \cdot & \beta & \alpha \end{pmatrix}, \quad N := \beta I \quad \text{et} \quad b^k := u^k + h_t f^{k+1}.$$

Équation de la chaleur en dimension 2 – Analyse numérique

Existence et unicité des solutions approchées

Méthode explicite

Évident.

Méthode implicite

Proposition A est définie-positive et $Au = f$ admet une unique solution.

Démonstration Montrer que A est à diagonale strictement dominante.

Équation de la chaleur en dimension 2 – Analyse numérique

Consistance des schémas

Proposition Les schémas explicite et implicite sont consistants en espace et en temps : $\lim_{h \rightarrow 0} |E_h| = 0$ et $\lim_{h_t \rightarrow 0} |E_{h_t}| = 0$.

Remarque Les schémas explicite et implicite sont d'ordre 2 pour x et pour y et d'ordre 1 pour t .

Équation de la chaleur en dimension 2 – Analyse numérique

Stabilité et convergence des schémas – Méthode explicite

Proposition (*admise*) Le schéma explicite est convergent $\Leftrightarrow \beta \leq \frac{1}{4}$.

Remarques

- On vérifiera cette propriété avec un exemple.
- Cette proposition s'appelle la condition de Courant-Friedrich-Levy (CFL).

Équation de la chaleur en dimension 2 – Analyse numérique

Stabilité et convergence des schémas – Méthode implicite

Proposition (*admise*) Soit $(\lambda_i)_{1 \leq i \leq N-1}$ l'ensemble des valeurs propres de A avec $\lambda_1 < \dots < \lambda_{N-1}$. Alors, $\lambda_1 \geq 1$.

Proposition Si $f \equiv 0$, alors le schéma implicite est stable : $\|u^{k+1}\| \leq \|u^0\|$.

Équation de la chaleur en dimension 2 – Analyse numérique

Stabilité et convergence des schémas – Méthode implicite

Démonstration

- Définir $\langle u, v \rangle := \sum_{i=1}^{N-1} \sum_{j=1}^{N-1} u_{i,j} v_{i,j}$ une application produit scalaire.
- Multiplier le schéma par $u_{i,j}^{k+1}$.
- Passer à la somme.
- Reconnaître le produit scalaire défini et utiliser le fait que A est diagonalisable.
- Utiliser l'inégalité de Cauchy-Schwarz.
- Faire une récurrence.

Équation de la chaleur en dimension 2 – Implémentation

- Pour la suite, la fonction à approcher sera
 $f(x, y, t) = (-\lambda + 2a\pi^2) \sin(\pi x) \sin(\pi y) e^{-\lambda t}$ avec
 $u_0(x, y) = \sin(\pi x) \sin(\pi y)$.
- Un tableau pour u à chaque pas de temps.
- 2 fonctions de résolutions : une qui calcule uniquement la solution approchée (pour mesurer le temps et/ou faire des entrées/sorties pour la visualisation) et une qui calcule la solution approchée en même temps que la solution exacte à chaque itération pour avoir l'erreur.
- On définira l'erreur `erreur_infty` comme : $\|e\|_\infty^\infty := \max_{1 \leq k \leq N_t} \|e_k\|_\infty$ avec
 $\|e_k\|_\infty = \|e\|_\infty$ à l'itération k .
- Différents modes d'exécution pour le stockage et le choix du calcul (macros EXACTE et ECRITURE).

Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma explicite en séquentiel

Fonction principale

```
void calculer_u(double *u){
    double *u_anc; double *permut;
    init_u_zero(u_zero, &u_anc);
    for (int i = 0 ; i < nb_pt * nb_pt ; i ++){
        u[i] = 0.0;
    }

    for (int k = 1 ; k <= N_t ; k ++){
        # ifdef ECRITURE
        ecrire_double_iteration(u_anc);
        # endif

        for (int j = 1 ; j < nb_pt - 1 ; j ++){
            for (int i = 1 ; i < nb_pt - 1 ; i ++){
                double f = f_source(i * h, j * h, k * h_t);
                u[IDX(i, j)] = schema(f, u_anc, i, j, k);
            }
        }

        permut = u; u = u_anc; u_anc = permut;
    }

    # ifdef ECRITURE
    ecrire_double_iteration(u);
    # endif

    terminaison(&permut, &u, &u_anc);
}
```

Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma explicite en séquentiel

Fonction qui applique le schéma à un point

```
static inline __attribute__((always_inline))
double schema(double *f, double *u_anc, int i, int j, int k){

    double res = alpha * u_anc[IDX(i, j)]
    + beta *
    (u_anc[IDX(i - 1, j)] + u_anc[IDX(i, j - 1)] + u_anc[IDX(i + 1, j)]
    + u_anc[IDX(i, j + 1)])
    + h_t * f[IDX(i, j)];

    return res;

}
```


Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma explicite en séquentiel

Commentaires

- $\|e\|_\infty$ en fonction de h et de h_t (avec $L = 1, T = 1, a = 1$ et $\lambda = 2a\pi^2$) :

$\downarrow h \quad h_t \rightarrow$	1/10000	1/20000	1/50000	1/100000
1/10	0.00266777	0.00284805	0.00295614	0.00299216
1/20	0.00039394	0.00057533	0.00068410	0.00072034
1/50	0.00024223	0.00006052	0.00004843	0.00008473
1/100	∞_f	∞_f	0.00004237	0.00000605

où ∞_f est l'infini des flottants double précision (inf). On vérifie la proposition énoncée, les valeurs ∞_f sont bien atteintes lorsque $\beta > 1/4$ donc la proposition de CFL est vérifiée pour cet exemple.

- Lorsque h/h_t est fixé, le schéma semble bien converger.

Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma explicite en séquentiel

Commentaires (suite)

- Temps d'exécution (en s) pour $N = 200$ et $N_t = 160000$ en fonction de l'activation ou non de l'écriture dans un fichier :

Mode	Sans écriture	Avec écriture
Temps d'exécution (en s)	52.1	57.8

- La condition sur β est très contraignante : si l'on souhaite diviser par 2 le pas spatial, alors il faut diviser par 4 le pas temporel. Et la constante $1/4$ implique que $h_t \leq \frac{1}{4a} h^2$, forçant des pas temporel très petits comparés aux pas spatiaux.
- La complexité algorithmique est $O(N^2 \cdot N_t)$.

Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma explicite en parallèle avec OpenMP

Fonction principale

```
void calculer_u(double *u){  
    double *u_anc; double *permut;  
    init_u_zero(u_zero, &u_anc);  
    for (int i = 0 ; i < nb_pt * nb_pt ; i ++){  
        u[i] = 0.0;  
    }  
  
    # pragma omp parallel firstprivate(u, u_anc, permut)  
    {  
        for (int k = 1 ; k <= N_t ; k ++){  
            # ifdef ECRITURE  
            # pragma omp single  
            ecrire_double_iteration(u_anc);  
            # endif  
  
            # pragma omp for schedule(static)  
            for (int j = 1 ; j < nb_pt - 1 ; j ++){  
                for (int i = 1 ; i < nb_pt - 1 ; i ++){  
                    double f = f_source(i * h, j * h, k * h_t);  
                    u[IDX(i, j)] = schema(f, u_anc, i, j, k);  
                }  
            }  
  
            permut = u; u = u_anc; u_anc = permut;  
        }  
  
        # ifdef ECRITURE  
        ecrire_double_iteration(u);  
        # endif  
  
        terminaison(&permut, &u, &u_anc);  
    }  
}
```

Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma explicite en parallèle avec OpenMP

Fonction pour terminer

```
void terminaison(double **permut, double **u, double **u_anc){  
    if (N_t % 2 != 0){  
        *permut = *u; *u = *u_anc; *u_anc = *permut;  
    }  
  
    # pragma omp single  
    free(*u_anc);  
}
```

Commentaires

- A la différence des implémentations OpenMP des problèmes stationnaires, on définit la zone parallèle (de fork) à l'extérieur des boucles. Les tableaux `u` et `u_anc` sont toujours sur le tas mais chaque thread possède une copie privée des pointeurs. Un seul thread effectue la libération de `u_anc`.

Commentaires (*suite*)

- L'écriture dans un fichier se fait en séquentiel.
- Temps d'exécution (en s) pour $N = 200$ et $N_t = 160000$ en fonction du nombre de threads de l'activation ou non de l'écriture dans un fichier :

↓ Nombre de threads	Mode →	Sans écriture	Avec écriture
1		52.0	58.2
2		30.1 1.7	36.4 1.6
4		20.9 2.5	24.3 2.4
6		25.0 2.1	28.5 2.0
8		21.7 2.4	25.6 2.3

Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma explicite en parallèle avec MPI

Fonction pour écrire u^k dans un fichier en parallèle (qui utilise un type dérivé `vue_fichier`) :

```
static inline __attribute__((always_inline, unused))
void ecrire_double_iteration(double *u, int k){

    uint64_t offset = (uint64_t)k * (uint64_t)nb_pt * (uint64_t)nb_pt * (
        uint64_t)sizeof(double);
    int taille[2] = {nb_pt, nb_pt};
    int sous_taille[2] = {nb_pt_div_j, nb_pt_div_i};
    int debut[2] = {j_debut, i_debut};

    MPI_Datatype vue_fichier;
    MPI_Type_create_subarray(2, taille, sous_taille, debut, MPI_ORDER_C,
        MPI_DOUBLE, &vue_fichier);
    MPI_Type_commit(&vue_fichier);

    MPI_File_set_view(descripteur, offset, MPI_DOUBLE, vue_fichier, "native",
        MPI_INFO_NULL);

    MPI_File_write_all(descripteur, u, 1, bloc_send, MPI_STATUS_IGNORE);

    MPI_Type_free(&vue_fichier);
}
```

Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma explicite en parallèle avec MPI

Commentaires

- Pour le calcul de la solution exacte (en séquentiel), on réutilise la fonction `regrouper_u` (voir la fonction `calculer_u_u_exact`).
- Temps d'exécution (en s) pour $N = 200$ et $N_t = 160000$ en fonction du nombre de processus de l'activation ou non de l'écriture dans un fichier :

↓ Nombre de processus	Mode →	Sans écriture	Avec écriture
1		53.3	98.3
2		33.7 1.6	63.7 1.5
4		25.9 2.1	74.2 1.3
6		35.4 1.5	88.5 1.1
8		38.4 1.4	132.0 0.7

Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma implicite en séquentiel

Fonction principale

```
void resoudre(cholmod_sparse *A, double *u){
    double *b_int = (double *)malloc(idx_max * sizeof(double));
    double *u_int = (double *)malloc(idx_max * sizeof(double));
    init_u_zero(u_zero, u);

    cholmod_dense *b_dense = cholmod_allocate_dense(A -> nrow, 1, A -> nrow, CHOLMOD_REAL, &c);
    cholmod_dense *u_dense;
    cholmod_factor *L = cholmod_analyze(A, &c);
    cholmod_factorize(A, L, &c);

    for (int k = 1 ; k <= N_t ; k++){
        # ifdef ECRITURE
        ecrire_double_iteration(u);
        # endif

        extraire_interieur(u, u_int, nb_pt);
        calculer_b(k + 1, u_int, b_int);

        memcpy(b_dense -> x, b_int, A -> nrow * sizeof(double));
        u_dense = cholmod_solve(CHOLMOD_A, L, b_dense, &c);
        memcpy(u_int, u_dense -> x, A -> nrow * sizeof(double));
        inserer_interieur(u_int, u, nb_pt);
    }

    # ifdef ECRITURE
    ecrire_double_iteration(u);
    # endif

    cholmod_free_factor(&L, &c);
    cholmod_free_dense(&b_dense, &c);
    cholmod_free_dense(&u_dense, &c);
    free(b_int);
    free(u_int);
}
```


Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma implicite en séquentiel

Fonction pour calculer b^k

```
static inline __attribute__((always_inline))
void calculer_b(double t, double *u, double *b){
    for (int i = 0 ; i < idx_max ; i ++){
        int x = i % (N - 1);
        int y = i / (N - 1);
        b[i] = u[i] + h_t * f_source(x, y, t + 1);
    }
}
```

Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma implicite en séquentiel

Commentaires

- $\|e\|_\infty$ en fonction de h et de h_t (avec $L = 1$, $T = 1$, $a = 1$ et $\lambda = 2a\pi^2$) :

$\downarrow h \quad h_t \rightarrow$	1/10000	1/20000	1/50000	1/100000
1/10	0.00338798	0.00320815	0.00310018	0.00306418
1/20	0.00111862	0.00093767	0.00082903	0.00079281
1/50	0.00048370	0.00030244	0.00019361	0.00015732
1/100	0.00039301	0.00021171	0.00010286	0.00006656

Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma implicite en séquentiel

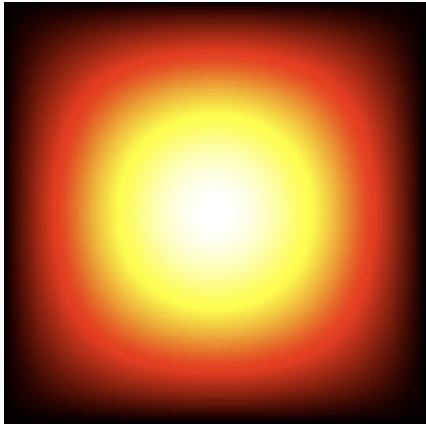
Commentaires

- Temps d'exécution (en s) en fonction de N (avec $N_t = N$) et de l'activation ou non de l'écriture dans un fichier :

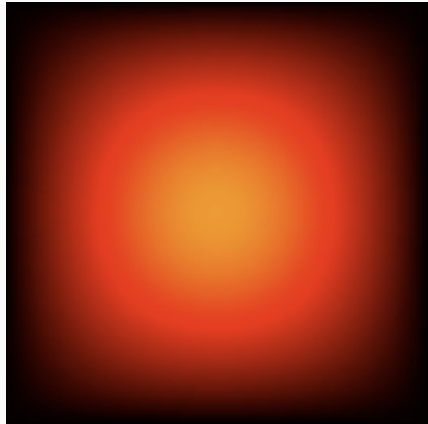
$\downarrow N(= N_t)$	Mode \rightarrow	Sans écriture	Avec écriture
600		12.6	12.3
800		32.0	29.2
1000		60.4	58.4
1200		97.7	99.0
1400		159.1	162.8

- Discussion sur les pas entre le schéma explicite et le schéma implicite.
- Il y a la possibilité de paralléliser certaines parties du code (hors calcul principal).

Équation de la chaleur en dimension 2 – Visualisation avec Python



État initial



État après diffusion

Bibliographie

- *Rapports de calcul scientifique*. (2008) par Patrick Ciarlet
- *Finite-Difference Approximations to the Heat Equation* (2004) par Gerald W. Recktenwald
- *Numerical Methods for Ordinary Differential Equations* par Habib Ammari et Konstantinos Alexopoulos
- *Lecture 6: Finite difference methods* par Habib Ammari
- Cours de calcul numérique (M1 CHPS) par Serge Dumont
- Cours d'analyse et calcul numérique (L3 Maths) par Francesco Bonaldi
- Cours d'algorithmique et programmation parallèle (M1 CHPS) par David Defour
- Forums d'aides

Projet

Lien vers le GitHub du projet : <https://github.com/gaillot18/Stage-EDP.git>

Merci pour votre attention. Questions ?