Thème 4 : résolution numérique des équations aux dérivées partielles (EDP) linéaires

1 Résolution numérique de l'équation de Laplace à 2D

1.1 Formulation du problème 2D

On cherche à calculer numériquement le champ de température $T(x, y)$ à l'*intérieur* d'un domaine rectangulaire de côtés Δx et Δy (voir figure 1). En l'absence de source, ce champ vérifie l'équation de Laplace :

$$\Delta\, T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{1}$$

Les quatre parois sont maintenues à des températures imposées (CL de type Dirichlet).

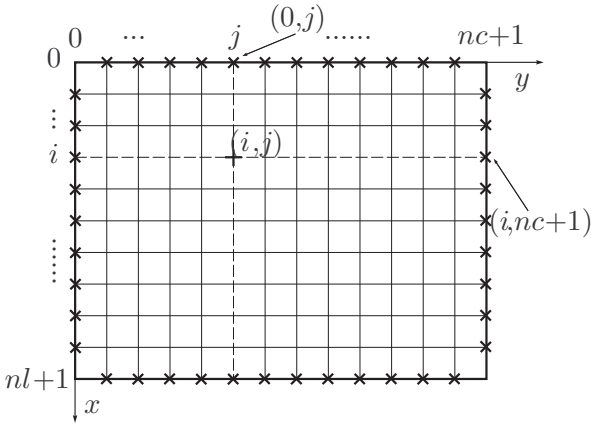

FIGURE 1 – Maillage du domaine : un point de la grille est repéré par un couple (i, j) où i représente le numéro de ligne et j le numéro de colonne. L'*intérieur* du domaine est numéroté de $i = 1$ à nl et de $j = 1$ à nc.

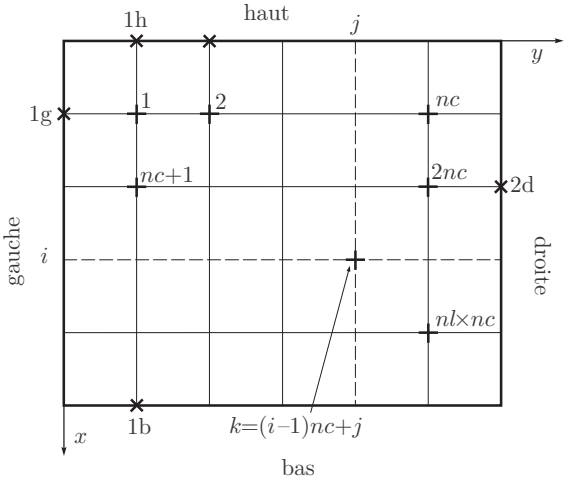

FIGURE 2 – Renumérotation des points intérieurs ligne par ligne de gauche à droite, puis de haut en bas (RowMajor).

Le maillage choisi est une grille de pas $dx = dy = p$ représentée sur la figure 1. Chaque point est repéré par un couple d'indices (i, j) où le premier indice note le numéro de la ligne et le second celui de la colonne. On note $T_{i,j}$ la température en ce point, de coordonnées $x_i = i \times p$, $y_j = j \times p$. Les inconnues du problème sont les $n = nl \times nc$ valeurs de $T_{i,j}$ où $1 \leqslant i \leqslant nl, 1 \leqslant j \leqslant nc$. Elles seront déduites des valeurs de $T_{i,j}$ aux bords : nord ($i = 0$), sud ($i = nl + 1$), ouest ($j = 0$) ou est ($j = nc + 1$).

L'équation de Laplace discrétisée se traduit par un système linéaire de n équations, chacune reliant la température au point (i, j) à celles des 4 points voisins :

$$T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1} - 4\,T_{i,j} = 0 \tag{2}$$

Les $2nl + 2nc$ températures des bords (aux points indiqués par des croix sur la figure 1) sont connues et les $n = nl \times nc$ températures intérieures sont à déterminer.

1.2 Renumérotation de la grille en 1D

N.-B. : Dans ce paragraphe, les indices sont les indices mathématiques commençant à 1. En langage C, penser à tenir compte du décalage d'une unité.

On représente le champ de température à calculer par un tableau 1D noté Z à $n = nl \times nc$ composantes en renumérotant les points de la grille à l'intérieur du rectangle par un seul indice k en suivant les lignes (fig. 2). Il s'agit donc d'un aplatissement du domaine 2D en mode `RowMajor` [1] :

$$\mathsf{Z}_k = T_{i,j} \quad \text{pour} \quad k = (i-1)nc + j \quad 1 \leqslant i \leqslant nl, 1 \leqslant j \leqslant nc \tag{3}$$

Pour un point sans contact avec la paroi ($1 < i < nl, 1 < j < nc$), l'équation (2) s'écrit :

$$\mathsf{Z}_{k-nc} + \mathsf{Z}_{k-1} - 4\mathsf{Z}_k + \mathsf{Z}_{k+1} + \mathsf{Z}_{k+nc} = 0 \tag{4}$$

Dès qu'il y a contact avec la paroi, l'opérateur Laplacien discrétisé fait intervenir une (ou au plus deux, dans les coins) température imposée, ce qui va constituer le second membre du système :

$$\mathsf{L}_{2D}\ \mathsf{Z} = \mathsf{B} \tag{5}$$

où L_{2D} est une matrice $n \times n$ constituée de $nl \times nl$ blocs de dimension $nc \times nc$, du type :

$$\left(\begin{array}{cccc|cccc|cccc}
-4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
1 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 \\
\hline
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -4 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 \\
\leftarrow\!\!-nc\!\!\rightarrow & & & & \leftarrow\!\!-nc\!\!\rightarrow & & & & \leftarrow\!\!-nc\!\!\rightarrow & & &
\end{array}\right)\Bigg\uparrow\Bigg\downarrow$$

nl blocs de taille $nc \times nc$

ici $nl = 3$ et $nc = 4$ (6)

et B est un vecteur à n composantes, dont au plus $2(nl + nc)$ non nulles, qui se calcule en sommant [2] les contributions des températures aux bords (voir représentation vectorielle dans le cours p. 25) :

$$\mathsf{B}_{k=(i-1)nc+j} = -\delta_{i,1}\, T_{i-1,j} - \delta_{i,nl}\, T_{i+1,j} - \delta_{j,1}\, T_{i,j-1} - \delta_{j,nc}\, T_{i,j+1} \tag{7}$$

1.3 Résolution du système à l'aide d'une procédure de `lapack` générique : `gesv`

Créer un répertoire `te4`, puis deux sous répertoires de `te4` nommés `laplace` et `diffusion`. Copier vers votre répertoire `laplace` le contenu du sous-répertoire `laplace` correspondant à votre langage de travail et situé sous `/home/lefrere/M1/2016-2017/etu/mncs/theme4-edp/`.

Le pas p de la grille n'intervient pas dans le calcul, mais sera utilisé dans la visualisation. Les dimensions du domaine $\Delta x = p(nl + 1)$ et $\Delta y = p(nc + 1)$ étant fixées, p sera déduit par exemple de Δx et nl. Pour faciliter la vérification de la bonne construction de L_{2D} et B, la mise au point du code se fera avec $\Delta x = 4$, $\Delta y = 5$, $nl = 3$ et $nc = 4$, correspondant à une résolution $p = 1$. Par la suite, on fera varier la résolution p en modifiant le couple (nl, nc) de façon à couvrir toujours le même domaine, c'est-à-dire en conservant le rapport $(nc + 1)/(nl + 1) = 5/4$.

1. En C, on pourra donc appeler la fonction `affmat`, pour afficher la solution dans le plan, mais à condition d'échanger les rôles du nombre de lignes et du nombre de colonnes.
2. On rappelle la signification du symbole de Kronecker $\delta_{ij} = 1$ si $i = j$ et 0 sinon.

1.3.1 Choix de la précision des réels

En fortran le module `mncs_precision` permettra de choisir la variante (`kind`) de type réel (`wp`) parmi celles définies dans le module `la_precision` de `lapack` ; chaque procédure devra donc comporter l'instruction `use mncs_precision`.

En C le choix du type `float` ou `double` pour les réels sera fait dans le fichier `mncs_precision.h` via un `typedef`. Chaque fichier source devra donc inclure `mncs_precision.h`.

Le fichier permettant le choix de la précision devra impérativement intervenir comme dépendance dans le `makefile` pour assurer la cohérence des types réels dans tous les fichiers objets.

1.3.2 Programme principal : fichier `resol.f90` ou `resol.c`

Le programme principal devra lire les dimensions du domaine : `nl`, `nc`, calculer le pas, calculer `n=nl`×`nc` puis allouer (dynamiquement en fortran et en tableaux 1D automatiques en C99, avec aplatissement des matrices en mode `ColMajor`) :

- un tableau de **réel**s `mat` de dimension `n`×`n` où sera stockée la matrice L_{2D}
- pour stocker les conditions aux limites, deux tableaux 1D de réels `ouest` et `est` de taille `nl` d'une part, deux tableaux 1D de réels `nord` et `sud` de taille `nc` d'autre part ;
- un tableau 1D de réels `temp` de taille `n` où sera d'abord stocké le second membre B, puis le champ des températures solution Z.

En fortran, on désallouera ces tableaux à la fin du programme principal.

1.3.3 Construction de la matrice L_{2D} : fichier `matrices.f90` ou `matrices.c`

La procédure `matl2d` remplira la matrice L_{2D} pour une grille `nl` × `nc`. On notera que cette matrice ⇐ 1
est tridiagonale par blocs (carrés) : il sera donc plus judicieux de construire ces blocs élémentaires (bloc nul, bloc identité, bloc type `L1D` pour lequel on adaptera la procédure `matl1d` du TE3 avec un coefficient diagonal de -4 au lieu de -2), puis de les copier dans le tableau final. Cette procédure devra respecter les interfaces suivantes :

En fortran dans le module `matrices`

```
subroutine matl2d(nl, nc, mat)
  integer, intent(in) :: nl, nc ! taille du domaine
  real(kind=wp),  dimension(:,:), intent(out) :: mat ! matrice de laplace
```

En C99 on construira la matrice « aplatie » en utilisant la fonction `matputblock` fournie dans `matutilcm.c`.

```
void matl2d(int nl, int nc, real mat[nl*nc*nl*nc]);
```

Appeler cette procédure depuis le programme principal et vérifier que cette matrice a bien été correctement construite en l'affichant, pour plusieurs valeurs (faibles) de `nl` et `nc` — notamment en échangeant les dimensions.

Dans les deux langages, on utilisera les procédures d'affichage `affmat` fournies dans `io.{c,f90}`. Elles attendent deux paramètres chaîne de caractères : un titre et un format. Pour un affichage compact de L_{2D}, on pourra choisir par exemple le format `"%2.0f "` en C et `"f3.0"` en fortran.

1.3.4 Construction du second membre à partir des conditions aux limites : fichiers `condlim.f90` ou `condlim.c`

Les conditions aux limites sont spécifiées sur les parois par quatre tableaux 1D `ouest` et `est` `nord` et `sud`. Ils seront calculés par la procédure `condlimites`, qui respectera les interfaces suivantes :

```
subroutine condlimites(nord, sud, ouest, est)
  real(kind=wp), dimension(:), intent(out) :: nord, sud, ouest, est
```

```
void condlimites(int nl, int nc, real nord[nc], real sud[nc],
                                 real ouest[nl], real est[nl]);
```

Dans un premier temps, on suppose que chacune des parois est à une température uniforme (`tn`, `ts`, `tw` et `te`) que l'on doit donc saisir dans une procédure `condlimites1`.

Dans un second temps, on envisagera des gradients de température constants sur chaque paroi. La procédure `condlimites2` lira alors les températures aux deux extrémités de chaque paroi, soit 8 valeurs : `tnw` et `tsw` par exemple pour le bord ouest. Il suffira ensuite de saisir des extrêmes identiques pour retrouver une température uniforme.

Construction du second membre : Écrire une procédure qui construit le vecteur creux `B` à `n` composantes du second membre à partir des quatre tableaux 1D des températures sur les parois.
2 ⇒ Ajouter dans le programme principal l'appel à cette procédure et tester en affichant le système grâce à la procédure `affsys` fournie dans `io.{c,f90}`.

```
subroutine second_membre(second_mb, nord, sud, ouest, est)
    real(kind=wp), dimension(:), intent(out) :: second_mb
    real(kind=wp), dimension(:), intent(in) :: nord, sud, ouest, est
```

```
void second_membre(int nl, int nc, real second_mb[nl*nc],
                            real nord[nc], real sud[nc],
                           real ouest[nl], real est[nl]);
```

1.3.5 Résolution du système avec `gesv` de `lapack`

Compléter le programme principal pour résoudre le système linéaire (5). On commencera par une méthode générale qui ne prend pas en compte les propriétés de la matrice L_{2D} : la procédure `gesv` de la bibliothèque `lapack`. Noter qu'elle travaille « en place » : elle remplace [3] le vecteur second membre par le vecteur solution, et la matrice L_{2D} par sa décomposition LU.

1.3.6 Écriture du tableau des $T_{i,j}$ dans un fichier : `ecritemp`

Ajouter enfin l'appel à la procédure `ecritemp` fournie (dans `io.{c,f90}`) qui écrit le champ complet des températures (intérieur et bords) dans un fichier `laplace.dat` sous la forme d'un tableau 2D (`nl`+2) × (`nc`+2), précédé d'une ligne d'entête indiquant le nombre de lignes et de colonnes (`nl`, `nc`) du domaine intérieur et le pas `p`. Les températures aux 4 coins sont obtenues par moyennes des 2 voisins.

1.3.7 Visualisation sous python

Visualiser les isothermes et la surface $T(x, y)$ à l'aide du script python `plot-laplace.py` fourni. Ne pas oublier le choix particulier des axes x et y. On pourra modifier le nombre d'isothermes via la variable `niso` et ajuster l'angle de vue de la surface via les variables `Altitude` et `Azimuth`.

1.3.8 Amélioration de la résolution

Reprendre le calcul et la visualisation avec `nl=19` et `nc=24` ($p = 0.2$) pour obtenir des isothermes
3 ⇒ plus régulières. Sauvegarder le graphique 2D des isothermes dans le fichier `laplace-19x24.pdf`. Que se passe-t-il si on diminue le pas d'un facteur 2 ? Expliquer. On pourra évaluer grossièrement le temps de calcul avec la commande unix `time` quitte à supprimer temporairement l'écriture sur fichier.

3. En particulier, cela interdit de réutiliser les tableaux ayant servi à l'appel de `gesv` pour refaire l'inversion avec une autre procédure (par exemple `pbsv`, *cf.* 1.4).

1.4 Résolution du système à l'aide d'une procédure spécialisée de `lapack` : `pbsv`

Il est beaucoup plus efficace de profiter des particularités de la matrice (symétrique, bande par exemple) en appelant des procédures optimisées. On choisit ici la procédure `pbsv` afin d'exploiter toutes les propriétés de la matrice $-\mathsf{L}_{2D}$, c'est-à-dire symétrique, définie positive et matrice bande. Pour résoudre l'équation initiale, on changera donc aussi le signe du vecteur B. Pour économiser la place mémoire et ne traiter que les éléments non nuls, les propriétés particulières de ces matrices sont exploitées pour stocker le minimum de termes :

— pour les matrices symétriques on ne stocke que la partie triangulaire inférieure ;
— pour les matrices symétriques bandes, on ne stocke que les diagonales et sous-diagonales non nulles sous forme d'une matrice obtenue en « tournant » de 45 degrés les diagonales non nulles. On obtient ainsi une matrice avec un nombre de lignes égal au nombre de diagonales conservées et le même nombre de colonnes que la matrice initiale. Comme les sous-diagonales sont de plus en plus courtes lorsqu'on s'éloigne de la diagonale principale, certains éléments de la matrice compactée sont indéterminés. On les place ici en fin de ligne.

L_{2D} est une matrice bande dont les éléments non-nuls sont au plus éloignés de nc de la diagonale principale : c'est une matrice avec $2\,nc{+}1$ bandes. En tant que matrice symétrique, sa forme compactée L_{2D}^{C} est donc une matrice $nc+1$ lignes et $nl \times nc$ colonnes [4] qui contiendra la diagonale et les sous-diagonales. On devra donc préciser le paramètre `uplo='L'` lors de l'appel de `pbsv`.

En fortran on construira directement la matrice compactée L_{2D}^{C} avec un sous-programme `matl2dk`.

En C99 on « compactera » la matrice L_{2D} à l'aide de la procédure `matfull2band` fournie, de façon à obtenir L_{2D}^{C} (les deux matrices étant stockées en aplati en mode `ColMajor`).

Créer un sous-répertoire `compact` : y copier les codes précédents, renommer le programme principal en `resol2` et le modifier pour appeler `pbsv` au lieu de `gesv`. Après avoir visualisé les résultats à haute résolution (isothermes en 2D dans le fichier `laplace-pbsv.pdf`), on comparera les temps de calcul entre les deux méthodes. ⇐ 4

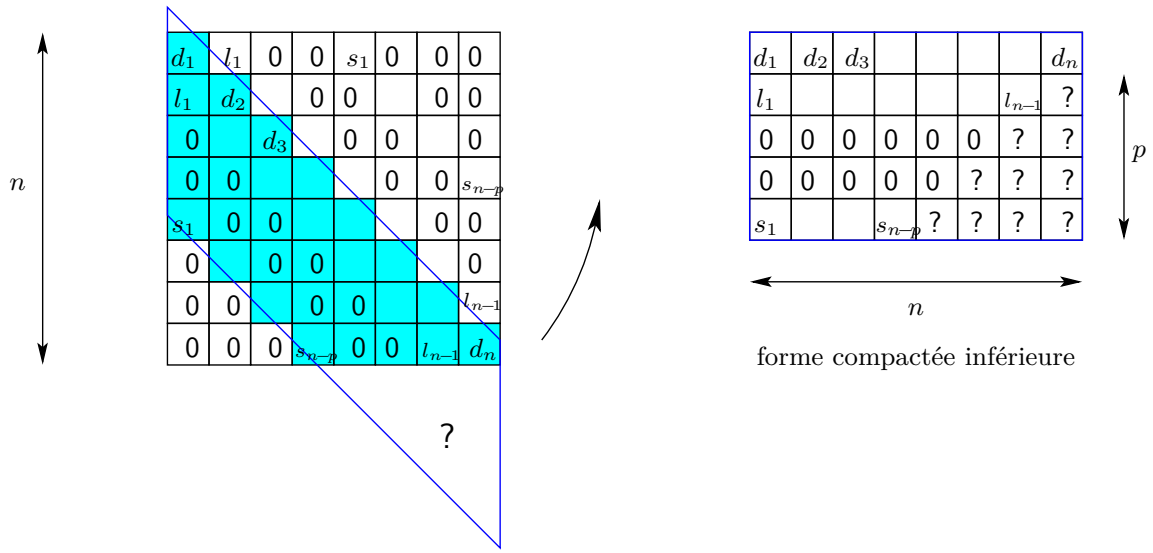

FIGURE 3 – Stockage des matrices symétriques bande : cas où on stocke diagonale et partie inférieure. Si une matrice carrée $n \times n$ comporte au plus p sous-diagonales (et autant de sur-diagonales) non nulles, on peut la représenter en ne stockant que la diagonale et, par exemple, les p sous-diagonales dans une matrice $p+1$ lignes et n colonnes. Pour ce faire, on effectue une rotation de $+45$ degrés des diagonales. Au fur et à mesure qu'on s'éloigne de la diagonale principale, les sous-diagonales raccourcissent, donc il reste des éléments indéterminés (représentés par des ?) en fin de ligne dans la matrice compactée.

4. Le gain sera d'autant plus important que nl est grand. C'est pourquoi, dans le cas d'un domaine de rapport d'aspect très différent de 1, il est préférable de choisir les axes x et y de sorte que $nl \ll nc$.

2 Résolution numérique de l'équation de diffusion 1D

2.1 Rappels introductifs

On cherche à obtenir numériquement la fonction $u(x,t)$ pour $t \in [0,T]$, sur l'intervalle $x \in [0,L]$, solution de l'équation de diffusion à 1D avec des conditions initiales (CI) et aux limites (CL) :

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} \qquad \text{avec} \qquad \begin{cases} \text{(CI)} & \forall x,\ u(x,0) = u_0(x) \\ \text{(CL)} & \forall t,\ u(0,t) = u_g(t) \quad \text{et} \quad u(L,t) = u_d(t) \end{cases} \tag{8}$$

Le coefficient de diffusion D, nécessairement positif, est supposé constant pour simplifier.

L'intervalle $[0,L]$ est divisé en n_x+1 pas de largeur $\delta x = L/(n_x+1)$, et celui de temps en n_t+1 pas de durée δt. On note u_j^n, la valeur numérique de $u(x,t)$ au point $x = j\delta x$ et au temps $t = n\delta t$. On définit le paramètre caractéristique :

$$\alpha = \frac{D\ \delta t}{(\delta x)^2} \tag{9}$$

— Les conditions aux limites fixent donc les u_0^n et les $u_{n_x+1}^n$ pour tout n ;
— La condition initiale donne u_j^0 pour $0 \leqslant j \leqslant n_x+1$.

La méthode de résolution est itérative en temps : à chaque pas d'intégration, on doit calculer les n_x valeurs aux points *intérieurs* du domaine en prenant en compte les conditions aux limites.

2.2 Partie commune

2.2.1 Méthodes et notations

On définit :

— le vecteur de taille n_x des u_j^n pour $j \in [1,n_x]$, $\mathsf{U}^n = (u_1^n, u_2^n \dots u_{n_x}^n)$;
— le vecteur de taille n_x déduit des conditions aux limites $\mathsf{V}^n = (u_g^n, 0\dots,0, u_d^n)$, supposé ici constant pour simplifier ;
— la matrice, de taille $n_x \times n_x$, $\mathsf{M}(\alpha) = \mathbb{1} - \alpha\mathsf{L}_{1D}$, où $\mathbb{1}$ est l'identité et L_{1D} la matrice tridiagonale définie au TE3, qui représente la dérivée seconde en différences finies.

Les différents algorithmes vus en cours s'écrivent :

Explicite : $\mathsf{U}^{n+1} = \mathsf{M}(-\alpha)\cdot\mathsf{U}^n + \alpha\mathsf{V}^n$;

Implicite : $\mathsf{U}^{n+1} = \mathsf{M}(\alpha)^{-1}\cdot[\mathsf{U}^n + \alpha\mathsf{V}^n]$;

Crank–Nicolson : $\mathsf{U}^{n+1} = \mathsf{M}(\frac{\alpha}{2})^{-1}\cdot[\mathsf{M}(-\frac{\alpha}{2})\cdot\mathsf{U}^n + \frac{\alpha}{2}(\mathsf{V}^n + \mathsf{V}^{n+1})]$.

On rappelle que l'algorithme explicite n'est stable que pour $\alpha \leqslant 1/2$.
On notera que pour $\alpha \ll 1$, $\mathsf{M}(-\alpha) \approx \mathsf{M}(\alpha)^{-1}$, mais que pour α petit mais fini, la matrice $\mathsf{M}(\alpha)^{-1}$ contient des termes non-diagonaux qui décroissent comme $\alpha^{|i-j|}$ lorsqu'on s'éloigne de la diagonale.

2.2.2 Organisation des codes

On souhaite écrire un code modulaire permettant de changer aisément d'algorithme, de l'explicite, plus simple, à celui de Crank-Nicolson, le plus précis. Ces modules, dont les interfaces sont fournies, seront complétés au fur et à mesure :

— `matrices` pour la construction de la matrice $\mathsf{M}(\alpha)$;
— `matop` pour les opérations matricielles (produit matrice-vecteur, matrice-matrice et inversion de matrice) ;
— `methodes` pour les trois procédures d'avancement d'un pas en temps.

On notera :

— Que la matrice utilisée par la méthode explicite est tridiagonale : son action devra être codée «à la main» pour éviter de faire intervenir tous les éléments nuls.

— Au contraire, les produits matrice-matrice et/ou matrice-vecteur utilisés dans les méthodes implicite et Crank-Nicolson, impliquent des matrices denses, et devront être réalisés à l'aide de Blas : `cblas_dgemm` et `cblas_dgemv` en C, et `GEMM` et `GEMV` en Fortran.

— Que la matrice à inverser dans ces deux derniers schémas est constante, et ne nécessite donc qu'une seule inversion. Il n'est donc pas indispensable de recourir à une méthode spécifique aux matrices tridiagonales symétriques, et on se contentera d'appeler `LAPACKE_dgesv` en C et `LA_GESV` en Fortran.

2.2.3 Spécifications du problème et choix des paramètres

On étudie la diffusion d'une superposition de deux modes sinusoïdaux :

— avec comme CI :

$$u_0(x) = 1 + \sin\left(\frac{\pi\, x}{L}\right) + 0.2\, \sin\left(\frac{5\,\pi\, x}{L}\right) \tag{10}$$

— et des CL constantes $u_g^n = u_0(x = 0) = 1$ et $u_d^n = u_0(x = L) = 1$ quel que soit n.

Pour les tests, on choisira par défaut les paramètres suivantes :

— $L = 120$ et une discrétisation fixe en espace avec $\delta x = 1$, donc `nx=119` ;

— $D = 1$: choisir le pas en temps δt équivaut donc à choisir α.

2.3 Travail à effectuer

2.3.1 Algorithme explicite

Fichiers fournis Créer un répertoire `te4/diffusion/explicite/`. Y copier depuis le compte de l'UE, les fichiers du répertoire `theme4-edp/diffusion/explicite/` correspondant à votre langage :

— un fichier `makefile` ;

— un embryon des modules `matrices` et `methodes` spécifiant les interfaces ;

— le module `matop` définissant quelques opérations fondées sur Blas et Lapack, à compléter ;

— un fichier `plot-diff.py` pour visualiser les résultats sous `python` (on placera en tête du fichier le nombre de points en x, soit $n_x + 2$ avec les bords, et le nombre d'instants).

Programme principal `diff_expl` chargé de :

— Lire les paramètres `dt` et `nt`.

— Sachant que `L` et `dx` sont fixés, calculer `nx` et α puis les afficher.

— Lire les conditions aux limites `ug` et `ud`.

— Allouer les vecteurs `unew`, `uold` de taille `nx` et les matrices nécessaires [5] ;

— Initialiser `uold` avec les conditions initiales (10).

— Effectuer la boucle sur les temps qui, à chaque pas :

1. appelle `avance_expl` pour calculer `unew` en fonction de `uold` ;
2. recopie `unew` dans `uold` ;
3. écrit [6] les $n_x + 2$ valeurs obtenues, bords compris, dans le fichier `explicite.dat` (et par la suite `implicite.dat` puis `cranknic.dat`).

5. En Fortran, on utilisera l'allocation dynamique de tableaux 1D et 2D. En C, on utilisera au choix l'allocation dynamique ou des tableaux automatiques, *exclusivement 1D* puisque les matrices doivent être aplaties.

6. Pour ne pas avoir à stocker les données sous forme 2D, on choisit ici d'écrire les résultats des u_j^n au fur et à mesure de leur obtention.

Méthode Écrire la procédure `avance_expl` implémentant l'algorithme explicite.

Tests

— Tester le programme avec, pour commencer `nt=1000`, `dt=0.1`, `ug=1.`, `ud=1.`

5 ⇒ — Visualiser le résultat figure `explicit.pdf`. Puis augmenter `nt`.

6 ⇒ — Augmenter enfin la valeur de `dt` et observer ; commenter ce qui se passe si $\alpha > 0.5$.
Dans ce cas, on arrêtera le calcul dès qu'un des points de la solution dépasse un certain seuil.

2.3.2 Algorithme implicite

Codes

— Recopier le contenu de votre sous-répertoire `diffusion/explicite/` vers un répertoire `implicite/`, et renommer le programme principal en `diff_impl`.

— Le modifier pour déclarer et calculer les matrices `malpha` et `invmalpha` à l'aide des procédures de `matrices` et de `matop`.

— Implémenter dans `methodes` la procédure `avance_impl` qui calcule `unew`.

Tests

— Tester l'implémentation réalisée avec `nt=1000`, `dt=0.1`.

7 ⇒ — Visualiser le résultat figure `implicit.pdf`. Puis augmenter `nt`

8 ⇒ — Augmenter la valeur de `dt` et expliquer ce qui passe pour par exemple $\alpha = 1$.

— Étudier le cas d'une condition initiale en forme de marche : $u_0(x) = +1$ pour $0 \leqslant x \leqslant L/2$ et $u_0(x) = -1$ pour $L/2 < x \leqslant L$, avec des conditions aux limites constantes.

2.3.3 Algorithme de Crank–Nicolson

Codes

— Recopier le contenu de votre sous-répertoire `diffusion/explicite/` vers un répertoire `cranknic/`, et renommer le programme principal en `diff_cranknic`.

— Le modifier pour déclarer et calculer les matrices nécessaires à l'aide des procédures de `matrices` et de `matop`.

— Implémenter dans `methodes` la procédure `avance_cranknic` qui calcule `unew` (on utilisera un tableau 1D local pour stocker les vecteurs résultats intermédiaires).

Tests

— Tester l'implémentation réalisée avec `nt=1000`, `dt=0.1`.

9 ⇒ — Visualiser le résultat figure `cranknic.pdf`.

— Augmenter la valeur de `dt` pour tester le cas $\alpha > 1$:

10 ⇒ — Comparer avec les résultats de l'algorithme implicite..

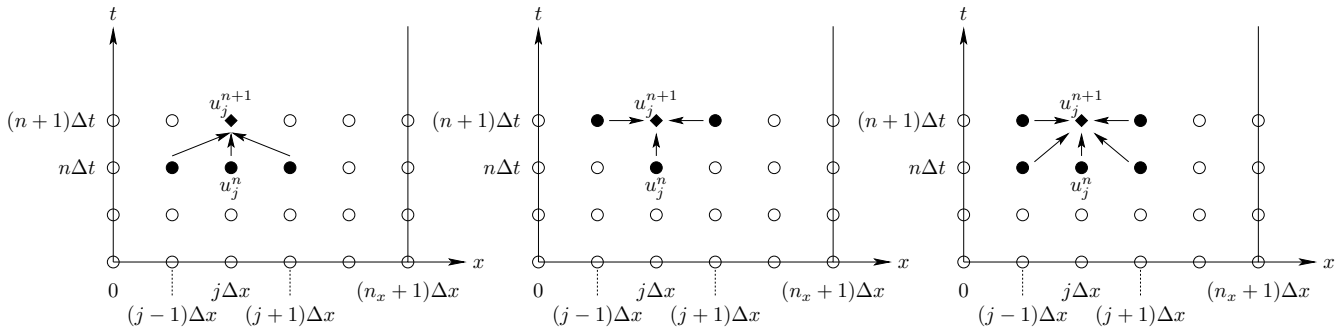

FIGURE 4 – Schémas différentiels des algorithmes explicite, implicite et de Crank–Nicolson (de gauche à droite)