Travaux pratiques - Méthodes des différences finies 2D

TP 5 : Matrices creuses, Laplacien 2D et plus si affinité

Table des matières

En résolvant numériquement des EDPs par des méthodes de type différences finies, éléments finis et/ou volumes finis, on abouti souvent à la résolution de grands systèmes linéaires creux (i.e. les matrices des systèmes linéaires contiennent un très grand nombre de zéros). Au niveau mémoire (de l'ordinateur), il est alors recommandé (voir nécessaire) d'utiliser un stockage particulier pour ces matrices : CSC, CSR, COO, LIL, DOK, ... Ces différentes formats permettent de ne stocker en mémoire que les éléments non nuls des matrices.

Sous Matlab/Octave, le format utilisé est le format CSC (Compressed Sparse Column). Il est très efficace pour les operations arithmétiques et les produits matrices vecteurs. Nous donnons deux utilisations (parmis d'autres) de la fonction **sparse** de Matlab/Octave :

- `M = sparse(m,n);`

  Cette commande crée la matrice creuse nulle `M` de dimension `m`×`n`
- `M = sparse(I,J,K,m,n);`

  Cette commande crée la matrice creuse `M` de dimension `m`×`n` telle que

  $$\texttt{M(Ig(k),Jg(k))} \leftarrow \texttt{M(Ig(k),Jg(k))} + \texttt{Kg(k)}.$$

  Les vecteurs `Ig`, `Jg` et `Kg` ont la même longueur. Les éléments nuls de `Kg` ne sont pas pris en compte et les éléments de `Kg` ayant les mêmes indices dans `Ig` et `Jg` sont sommés.

On donne maintenant dans plusieurs langages interprétés l'analogue de ces deux commandes :

- Python (*scipy.sparse* module) :
  — `M=sparse.<format>_matrix((m,n))`
  — `M=sparse.<format>_matrix((Kg,(Ig,Jg)),shape=(m,n))`
  où `<format>` est le format de stockage de la matrice creuse (i.e. `csc`, `csr`, `lil`, ...),
- R : format `CSC` (seulement ?),

— `M=sparse(i=,j=,dims=c(m,n))`
— `M=sparse(i=Ig,j=Jg,x=Kg,dims=c(m,n))`

- Scilab : format `row-by-row`(?)
  — `M=sparse([],[],[m,n])`
  — `M=sparse([Ig,Jg],Kg,[m,n])`

Dans tous ces langages, de très nombreuses fonctions existent permettant d'utiliser les matrices creuses comme des matrices *classiques*. Il est donc possible d'effectuer des opérations aussi élémentaires que des sommes, produits de matrices et vecteurs mais aussi des opérations plus complexes comme des factorisations (LU, Cholesky,QR, ...), résolution de systèmes linéaires (méthodes directes ou itératives), calcul de valeurs propres et vecteurs propres, ...

Pour les langages de programmation usuels (Fortan, C, C++, ...) des libraires existent permettant de réaliser ces mêmes opérations sur les matrices creuses :

- C/C++ avec les librairies *SuiteSparse* de T. Davis [**?**] ainsi que *BLAS* (Basic Linear Algebra Subroutines), *LAPACK* (Linear Algebra PACKage) et leurs dérivées.
- CUDA avec la librairie *cuSparse* [**?**] de Nvidia.

1 Les matrices creuses

Nous avons commencé à utiliser les matrices creuses (*sparse matrix*) dans les TPs précédents. Par exemple, pour générer la matrice $\mathbb{K} \in \mathcal{M}_d(\mathbb{R})$ définie par

$$\mathbb{K} = \begin{pmatrix} -2 & 1 & 0 & \dots & 0 \\ 1 & -2 & 1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 1 \\ 0 & \dots & 0 & 1 & -2 \end{pmatrix} \tag{1.1}$$

nous avons écrit une fonction `Lap1D` pouvant ressembler à ceci :

Listing 1 – Fonction `Lap1D`

```
function K=Lap1D(d)
  K=sparse(d,d);
  K(1,1)=-2;K(1,2)=1;
  for i=2:d-1
    K(i,i)=-2;
    K(i,i-1)=1;
    K(i,i+1)=1;
  end
  K(d,d)=-2;K(d,d-1)=1;
end
```

Toutefois cette manière de construire une matrice creuse est très loin d'être la plus efficace.

Dans un premier temps, une rapide explication du pourquoi vous est proposée et ensuite nous verrons comment améliorer celà en utilisant judicieusement la fonction **sparse** . Il est aussi possible d'utiliser conjointement les fonctions **speye** et **spdiags** pour créer la matrice $\mathbb{K}$ mais ces commandes ne seront plus utilisables pour assembler les matrices provenant des méthodes d'éléments finis et volumes finis, c'est pourquoi nous nous focaliserons sur la fonction **sparse** .

1.1 Stockage

Sous Matlab et Octave, une matrice creuse (ou *sparse matrix*), $\mathbb{A} \in \mathcal{M}_{M,N}(\mathbb{R})$, est stockée sous le format CSC (Compressed Sparse Column), utilisant trois tableaux :

$$ia(1 : nnz),\ ja(1 : N + 1) \text{ et } aa(1 : nnz),$$

où nnz est le nombre d'éléments non nuls de la matrice $\mathbb{A}$. Ces tableaux (cachés mais contenus dans l'object **sparse** Matlab/Octave) sont définis par

- aa : contient l'ensemble des nnz éléments non nuls de $\mathbb{A}$ stockés par colonnes.
- ia : contient les numéros de ligne des éléments stockés dans le tableau aa.
- ja : permet de retrouver les éléments d'une colonne de $\mathbb{A}$, sachant que le premier élément non nul de la colonne k de $\mathbb{A}$ est en position $ja(k)$ dans le tableau aa. On a $ja(1) = 1$ et $ja(N+1) = nnz + 1$.

Pour illustrer, voici un exemple simple avec la matrice

$$\mathbb{A} = \begin{pmatrix} 1. & 0. & 0. & 6. \\ 0. & 5. & 0. & 4. \\ 0. & 1. & 2. & 0. \end{pmatrix}$$

on a $M = 3$, $N = 4$, $nnz = 6$ et

aa	1.	5.	1.	2.	6.	4.
ia	1	2	3	3	1	2
ja	1	2	4	5	7	

Le premier élément non nul de la colonne $k = 3$ de $\mathbb{A}$ est 2, ce nombre est en position 4 dans aa, donc $ja(3) = 4$.

1.1.1 Occupation mémoire

Pour illustrer le gain mémoire, nous allons calculer la taille mémoire occupée par la matrice tridiagonale $\mathbb{K}$ définie en (1.1) dans le cas d'un stockage plein (*full matrix*) ou creux (*sparse matrix*). Cette matrice contient $d + 2(d-1) = 3d - 2$ éléments non nuls.

Sous Matlab/Octave le stockage creux de cette matrice va utiliser $(3d-2)$ `double` (tableau aa), $(3d-2)$ `int` (tableau ia) et $(d+1)$ `int` (tableau ja) pour un total de

$$(3d-2)\ \texttt{double} \quad \text{et} \quad (4d-1)\ \texttt{int}.$$

Or un `double` occupe 8 `Octets` (en anglais *bytes*) et un `int` 4 `Octets`. La place mémoire occupée par la matrice stockée sous forme `sparse` est alors de

$$(3d-2) \times 8 + (4d-1) \times 4 = (40\,d - 20)\ \texttt{Octets}.$$

Par contre, si la matrice était stockée classiquement (`full`), alors il y a d^2 `double` qui occupe en mémoire :

$$8d^2\ \texttt{Octets}.$$

On rappelle que 1 `Mo` (Méga-octet) correspond à 10^6 `Octets`, 1 `Go` (Giga-octet) correspond à 10^9 `Octets` et 1 `To` (Tera-octet) correspond à 10^{12} `Octets`. On donne maintenant la place mémoire occupée par la matrice suivant sa dimension et son stockage (`full` ou `sparse`) :

d	**full** matrix	**sparse** matrix
10^3	8 Mo	40 Ko
10^4	800 Mo	400 Ko
10^5	80 Go	4 Mo
10^6	8000 Go	40 Mo
10^7	800 To	400 Mo

1.1.2 Insertion d'éléments

Regardons les opérations à effectuer sur les tableaux aa, ia et ja si l'on modifie la matrice $\mathbb{A}$ par la commande

```
A(1,2)=8;
```

La matrice devient alors

$$\mathbb{A} = \begin{pmatrix} 1. & 8. & 0. & 6. \\ 0. & 5. & 0. & 4. \\ 0. & 1. & 2. & 0. \end{pmatrix}.$$

Dans cette opération un élément nul de $\mathbb{A}$ a été remplacé par une valeur non nulle 8 : il va donc falloir la stocker dans les tableaux sachant qu'aucune place n'est prévue. On suppose que les tableaux sont suffisamment grands (pas de souci mémoire), il faut alors décaler d'une case l'ensemble des valeurs des tableaux aa et ia à partir de la 3ème position puis copier la valeur 8 en $aa(2)$ et le numéro de ligne 1 en $ia(2)$:

aa	1.	8.	5.	1.	2.	6.	4.
ia	1	1	2	3	3	1	2

Pour le tableau ja, il faut à partir du numéro de colonne 2 plus un, incrémenter de +1 :

ja	1	2	5	6	8

La répétition de ces opérations peut devenir très coûteuse en temps CPU lors de l'assemblage de matrices utilisant des méthodes similaires à celle utilisée en Listing 1 (et on ne parle pas ici des problèmes de réallocation dynamique qui peuvent arriver !).

1.1.3 Construction efficace de matrices creuses

Pour générer efficacement une matrice creuse, il faut donc éviter d'insérer de manière répétitive des éléments dans celle-ci. Pour celà, on va utiliser l'appel suivant de la fonction `sparse` :

```
M = sparse(I,J,K,m,n);
```

Cette commande génère une matrice creuse `m` par `n` telle que `M(I(k),J(k)) = K(k)` . Les vecteurs `I` , `J` et `K` ont la même longueur. Il faut noter que tous les éléments nuls de `K` sont ignorés et que tous les éléments de `K` ayant les mêmes indices dans `I` et `J` sont sommés. Cette sommation est très utile lors de l'assemblage de matrices obtenues par une méthode de type éléments finis.

Par exemple, voici deux commandes permettant de générer la matrice $\mathbb{A}$ précédente :

```
A = sparse([1 1 2 3 3 1  2],[1  2 2 2 3 4  4],[1  8 5 1 2 6  4],3,4) ;
A = sparse([1 2 1 1 2 3  3],[4  4 1 2 2 2  3],[6  4 1 8 5 1  2],3,4) ;
```

Il existe aussi la fonction `A = spalloc(m,n,nnz)` qui préalloue une matrice creuse de dimension `m` par `n` avec au plus `nz` éléments non nuls (*nonzero elements*).

Nous allons maintenant utiliser ceci dans un cadre un peu général : les matrices tridiagonales.

1.2 Matrices diagonales et tridiagonales

Q. 1 (Matlab) *Ecrire la fonction* `spMatDiag` *permettant à partir d'un vecteur* $\boldsymbol{v} \in \mathbb{R}^d$ *de retourner la matrice diagonale (creuse)* $\mathbb{D} \in \mathcal{M}_d(\mathbb{R})$ *de diagonale* $\boldsymbol{v}$ *(i.e.* $\mathbb{D}(i,i) = \boldsymbol{v}(i)$, $\forall i \in [\![1,d]\!]$*) en créant (sans boucle) les tableaux* `I, J et K` *puis en générant la matrice* $\mathbb{D}$ *à l'aide de la commande* `D = sparse(I,J,K,d,d);` ▪

Soit $\mathbb{A} \in \mathcal{M}_d(\mathbb{R})$ la matrice tridiagonale définit par

$$\mathbb{A} = \begin{pmatrix} v_1 & w_1 & 0 & \cdots & \cdots & 0 \\ u_1 & v_2 & w_2 & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & v_{d-1} & w_{d-1} \\ 0 & \cdots & \cdots & 0 & u_{d-1} & v_d \end{pmatrix} \tag{1.2}$$

avec $\boldsymbol{u} \in \mathbb{R}^{d-1}$, $\boldsymbol{v} \in \mathbb{R}^d$ et $\boldsymbol{w} \in \mathbb{R}^{d-1}$.

Q. 2 (Matlab) *1. Ecrire la fonction* `spMatTriDiagSca` *permettant à partir des vecteurs* $\boldsymbol{u}$, $\boldsymbol{v}$ *et* $\boldsymbol{w}$ *de retourner la matrice creuse* $\mathbb{A}$ *définie en* (1.2)*. Pour celà on va tout d'abord créer une matrice creuse de* $\mathcal{M}_d(\mathbb{R})$ *puis on va la «remplir» à l'aide d'une boucle* ***for*** *sur le principe du Listing 1.*

2. *Ecrire la fonction* **`spMatTriDiagVec`** *permettant à partir des vecteurs* $\boldsymbol{u}$, $\boldsymbol{v}$ *et* $\boldsymbol{w}$ *de retourner la matrice creuse* $\mathbb{A}$ *en créant (sans boucle) les tableaux* `I`, `J` *et* `K` *puis en générant la matrice* $\mathbb{A}$ *à l'aide de la commande* **`A = sparse(I,J,K,d,d);`**

3. *Ecrire un programme permettant de mesurer (tic-toc) puis de représenter graphiquement les performances de* **`spMatTriDiagSca`** *et* **`spMatTriDiagVec`** *en fonction de d.*
(Attention à la capacité mémoire de la machine !) ▪

1.3 Produit de Kronecker de deux matrices (travail à la maison !)

Pour générer efficacement et simplement les matrices creuses obtenus par des méthodes de différences finies en dimension $\geqslant 2$ nous utiliserons le produit de Kronecker de deux matrices.

Soient $\mathbb{A} \in \mathcal{M}_{m,n}(\mathbb{R})$ et $\mathbb{B} \in \mathcal{M}_{p,q}(\mathbb{R})$. Le produit tensoriel de Kronecker de $\mathbb{A}$ par $\mathbb{B}$, noté $\mathbb{A} \otimes \mathbb{B}$, est la matrice de $\mathcal{M}_{mp,nq}(\mathbb{R})$ définie avec des blocs de dimension $p \times q$ par

$$\mathbb{A} \otimes \mathbb{B} = \begin{pmatrix} \mathrm{A}_{1,1}\mathbb{B} & \cdots & \mathrm{A}_{1,n}\mathbb{B} \\ \vdots & \ddots & \vdots \\ \mathrm{A}_{m,1}\mathbb{B} & \cdots & \mathrm{A}_{m,n}\mathbb{B} \end{pmatrix} \tag{1.3}$$

Le produit de Kronecker est bilinéaire et associatif : Si les dimensions des matrices $\mathbb{A}$, $\mathbb{B}$ et $\mathbb{C}$ sont compatibles on a $\forall \lambda \in \mathbb{K}$

$$\begin{aligned} \mathbb{A} \otimes (\mathbb{B} + \lambda\ \cdot \mathbb{C}) &= (\mathbb{A} \otimes \mathbb{B}) + \lambda(\mathbb{A} \otimes \mathbb{C}) \\ (\mathbb{A} + \lambda\ \cdot \mathbb{B}) \otimes \mathbb{C} &= (\mathbb{A} \otimes \mathbb{C}) + \lambda(\mathbb{B} \otimes \mathbb{C}) \\ \mathbb{A} \otimes (\mathbb{B} \otimes \mathbb{C}) &= (\mathbb{A} \otimes \mathbb{B}) \otimes \mathbb{C} \end{aligned}$$

Par contre, il n'est pas commutatif.

Nous allons écrire une fonction permettant d'effectuer ce produit. Mais auparavant, quelques petits rappels (ou non) des possibilités offertes par le langage Matlab.

Il est facile sous Malab/Octave de modifier une matrice (creuse ou non). Par exemple,

- `A(I,J)=M`

  Les tableaux d'indices `I` et `J` sont de dimension respective n et m. La matrice `M` est de dimension n-par-m. La matrice `A` est modifiée de telle sorte que

  $$\texttt{A(I(i),J(j))=M(i,j)},\ \forall \texttt{i} \in [\![1,n]\!],\ \forall \texttt{j} \in [\![1,m]\!]$$

- `A(I,J)=A(I,J)+M`
  Même chose en sommant.
- `A(i,:)=a`

  Ici `a` est un scalaire. Tous les éléments de la ligne `i` de la matrice `A` valent `a`.
- `A(:,j)=a`

  Ici `a` est un scalaire. Tous les éléments de la colonne `j` de la matrice `A` valent `a`.
- ...

Sous Matlab/Octave, il est possible de récupérer tous les éléments non nuls d'une matrice creuse `A` ainsi que leurs indices de ligne et de colonne à l'aide de la commande

```
[I,J,K]=find(A);
```

Ici les trois tableaux `I`, `J` et `K` ont même longueur et on a `A(I(k),J(k)) == K(k)` pour tout `k` inférieur ou égal à `length(K)`.

Q. 3 (Matlab) *Ecrire la fonction* **`spMatKron`** *permettant à partir deux matrices creuses carrées* $\mathbb{A} \in \mathcal{M}_n(\mathbb{R})$ *et* $\mathbb{B} \in \mathcal{M}_m(\mathbb{R})$ *de retourner la matrice creuse* $\mathbb{C} = \mathbb{A} \otimes \mathbb{B}$. *Pour celà, on initialisera la matrice retournée* `C` *à l'aide de la commande* **`C=sparse(n*m,n*m)`** *puis on utilisera uniquement une boucle sur les éléments non-nuls de la matrice* $\mathbb{A}$ *pour «remplir» la matrice* `C`. ▪

On pourra comparer cette fonction à la fonction **`kron`** de Matlab/Octave.

2 Notations et approximation des dérivées secondes

Soient $\Omega =]a,b[\times]c,d[\subset \mathbb{R}^2$ et $\Gamma = \partial\Omega$ la frontière du domaine Ω. On note Γ_N, Γ_S, Γ_O et Γ_E respectivement les frontières nord, sud, ouest et est. on a

$$\Gamma = \Gamma_N \cup \Gamma_S \cup \Gamma_O \cup \Gamma_E.$$

Soit $v : \Omega \longrightarrow \mathbb{R}$ suffisament régulière.

Q. 4 (sur feuille) *Montrer que l'on a*

$$\frac{\partial^2 v}{\partial x^2}(x,y) = \frac{v(x+h,y) - 2v(x,y) + v(x-h,y)}{h^2} + \mathcal{O}(h^2) \tag{2.1}$$

$$\frac{\partial^2 v}{\partial y^2}(x,y) = \frac{v(x,y+h) - 2v(x,y) + v(x,y-h)}{h^2} + \mathcal{O}(h^2). \tag{2.2}$$

∎

On note $(x_i)_{i=0}^{N_x}$ et $(y_j)_{j=0}^{N_y}$ les discrétisation régulières, respectivement, des intervalles $[a,b]$ et $[c,d]$ défines par

$$x_i = a + ih_x,\ \forall i \in [\![0, N_x]\!] \quad \text{et} \quad y_j = c + jh_y,\ \forall j \in [\![0, N_y]\!] \tag{2.3}$$

avec $h_x = (b-a)/N_x$ et $h_y = (d-c)/N_y$. On note aussi

$$n_x = N_x + 1, \quad n_y = N_y + 1 \ \text{ et } \ N = n_x \times n_y \tag{2.4}$$

Q. 5 (sur feuille) *Montrer que l'on a* $\forall i \in]\!]0, N_x[\![,\ \forall j \in]\!]0, N_y[\![$

$$\begin{aligned}\Delta v(x_i,y_j) &\stackrel{\text{def}}{=} (\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2})(x_i,y_j) \\ &= \frac{v(x_{i+1},y_j) - 2v(x_i,y_j) + v(x_{i-1},y_j)}{h_x^2} \\ &\quad + \frac{v(x_i,y_{j+1}) - 2v(x_i,y_j) + v(x_i,y_{j-1})}{h_y^2} + \mathcal{O}(h_x^2) + \mathcal{O}(h_y^2)\end{aligned} \tag{2.5}$$

∎

3 Equation de Poisson avec conditions de Dirichlet

Soient $f : \Omega \longrightarrow \mathbb{R}$ et $g : \Gamma \longrightarrow \mathbb{R}$ deux fonctions données. On veut résoudre le problème suivant

$$-\Delta u = f, \qquad \text{dans } \Omega \tag{3.1}$$

$$u = g, \qquad \text{sur } \Gamma \tag{3.2}$$

On utilise par la suite les discrétisations $(x_i)_{i=0}^{N_x}$ et $(y_j)_{j=0}^{N_y}$ définies en section 2.

Q. 6 (sur feuille) *1. Montrer que ce problème peut s'écrire après discrétisation sous la forme*

$$-\frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{h_x^2} - \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{h_y^2} = f(x_i,y_j), \qquad \forall (i,j) \in]\!]0,N_x[\![\times]\!]0,N_y[\![, \tag{3.3}$$

$$U_{0,j} = g(a,y_j), \qquad \forall j \in [\![0,N_y]\!], \tag{3.4}$$

$$U_{N_x,j} = g(b,y_j), \qquad \forall j \in [\![0,N_y]\!], \tag{3.5}$$

$$U_{i,0} = g(x_i,c), \qquad \forall i \in [\![0,N_x]\!], \tag{3.6}$$

$$U_{i,N_y} = g(x_i,d), \qquad \forall i \in [\![0,N_x]\!]. \tag{3.7}$$

avec $U_{i,j} \approx u(x_i,y_j)$.

2. Les inconnues du problème discrétisé sont les $U_{i,j}$, *pour* $i \in [\![0,N_x]\!]$ *et* $j \in [\![0,N_y]\!]$. *Compter le nombre d'équations distinctes du problème discrétisé (bord compris). Que peut-on en conclure ?* ∎

En notant $\beta_x = -\frac{1}{h_x^2}$, $\beta_y = -\frac{1}{h_y^2}$ et $\mu = 2(\frac{1}{h_x^2} + \frac{1}{h_y^2})$, les équations (3.3) peuvent aussi s'écrire sous la forme

$$\beta_x(U_{i+1,j} + U_{i-1,j}) + \mu U_{i,j} + \beta_y(U_{i,j+1} + U_{i,j-1}) = f(x_i, y_j), \quad \forall (i,j) \in]\!]0, N_x[\![\times]\!]0, N_y[\![\tag{3.8}$$

Pour tout $j \in [\![0, N_y]\!]$, on note $U_{:,j}$ le vecteur de $\mathbb{R}^{n_x}$ définit par

$$U_{:,j} = \begin{pmatrix} U_{0,j} \\ \vdots \\ U_{N_x,j} \end{pmatrix}.$$

On note $\boldsymbol{V} \in \mathbb{R}^N$ le vecteur bloc

$$\boldsymbol{V} = \begin{pmatrix} U_{:,0} \\ \hdashline U_{:,1} \\ \hdashline \vdots \\ \hdashline U_{:,N_y} \end{pmatrix}$$

Q. 7 (sur feuille) *Explicitez la bijection* $\mathcal{F} : [\![0, N_x]\!] \times [\![0, N_y]\!] \longrightarrow [\![1, N]\!]$ *telle que*

$$\forall (i,j) \in [\![0, N_x]\!] \times [\![0, N_y]\!], \quad V_k = U_{i,j}, \quad \textit{avec } k = \mathcal{F}(i,j).$$

Dans le cas de la numérotation en $(i,j) \in [\![0, N_x]\!] \times [\![0, N_y]\!]$ on parlera de **numérotation 2D** et pour la numérotation en $k \in [\![1, N]\!]$ on parlera de **numérotation globale**.

Q. 8 (Matlab) *1. Ecrire la fonction* `k=bijF(i,j,nx)` *correspondant à la bijection* $\mathcal{F}$ *(* ***numerotation 2D*** *vers* ***numerotation globale****).*

2. Ecrire la fonction réciproque `[i,j]=bijRecF(k,nx)` *correspondant à* $\mathcal{F}^{-1}$ *(* ***numerotation globale*** *vers* ***numerotation 2D****). On pourra utiliser la fonction* ***rem****(x,y) (reste de la division de* `x` *par* `y`*) et on vérifiera que cette fonction est naturellement vectorisée, c'est à dire que le paramètre* `k` *peut aussi être un tableau.* ∎

3.1 Evaluation de fonctions sur la grille discrète

Soit $\boldsymbol{F} \in \mathbb{R}^N$ le vecteur (bloc) définit par

$$\boldsymbol{F} = \begin{pmatrix} F_{:,0} \\ \hdashline F_{:,1} \\ \hdashline \vdots \\ \hdashline F_{:,N_y} \end{pmatrix} \text{ avec } F_{:,j} = \begin{pmatrix} f(x_0, y_j) \\ \vdots \\ f(x_{N_x}, y_j) \end{pmatrix} \in \mathbb{R}^{Nx+1}, \quad \forall j \in [\![0, N_y]\!].$$

En Algorithme 1, est présenté une version non vectorisée (deux boucles imbriquées) de la construction du vecteur $\boldsymbol{F}$. Dans cet algorithme l'ordre des boucles est primordial pour respecter le choix de la numérotation globale.

Algorithm 1 Algorithme non vectorisé de calcul du vecteur $\boldsymbol{F}$

```
k ← 1
for j ← 1 to n_y do
  for i ← 1 to n_x do
    F(k) ← f(x(i), y(j)))
    k ← k + 1
  end for
end for
```

Q. 9 (Matlab) *Ecrire la fonction* `EvalFun2DSca` *permettant à partir d'une fonction* f *donnée et des discrétisations* $\boldsymbol{x}$ *et* $\boldsymbol{y}$ *de retourner le vecteur* $\boldsymbol{F}$ *associé. On pourra utiliser des boucles* ***for*** *.* ∎

Pour le cas où la fonction f est vectorisée sous Matlab/Octave, il est alors possible de calculer le vecteur $\boldsymbol{F}$ sans boucle. Pour celà, nous allons construire les deux vecteurs blocs $\boldsymbol{X}$ et $\boldsymbol{Y}$ de $\mathbb{R}^n$ définis par

$$\boldsymbol{X} = \begin{pmatrix} X_{:,0} \\ \hdashline X_{:,1} \\ \hdashline \vdots \\ \hdashline X_{:,N_y} \end{pmatrix} \text{ avec } X_{:,j} = \begin{pmatrix} x_0 \\ \vdots \\ x_{N_x} \end{pmatrix} \in \mathbb{R}^{Nx+1}, \quad \forall j \in [\![0, N_y]\!]$$

et

$$\boldsymbol{Y} = \begin{pmatrix} Y_{:,0} \\ \hdashline Y_{:,1} \\ \hdashline \vdots \\ \hdashline Y_{:,N_y} \end{pmatrix} \text{ avec } Y_{:,j} = \begin{pmatrix} y_j \\ \vdots \\ y_j \end{pmatrix} \in \mathbb{R}^{Nx+1}, \quad \forall j \in [\![0, N_y]\!].$$

Avec ces deux vecteurs, le calcul du vecteur $\boldsymbol{F}$ pourra être vectorisé (sous la condition que f le soit) :

```
F=f(X,Y);
```

Il nous faut donc écrire une version vectorisée du calcul des vecteurs $\boldsymbol{X}$ et $\boldsymbol{Y}$, mais auparavant on donne plusieurs manières d'écrire le calcul du vecteur $\boldsymbol{F}$

Algorithm 2 Calcul de $\boldsymbol{F}$ avec construction de $\boldsymbol{X}$ et $\boldsymbol{Y}$ (version 1)

```
k ← 1
for j ← 1 to n_y do
  for i ← 1 to n_x do
    X(k) ← x(i)
    Y(k) ← y(j)
    F(k) ← f(X(k),Y(k)))
    k ← k + 1
  end for
end for
```

Algorithm 3 Calcul de $\boldsymbol{F}$ avec construction de $\boldsymbol{X}$ et $\boldsymbol{Y}$ (version 2)

```
1: k ← 1
2: for j ← 1 to n_y do
3:   for i ← 1 to n_x do
4:     X(k) ← x(i)
5:     Y(k) ← y(j)
6:     k ← k + 1
7:   end for
8: end for
9: F ← f(X,Y)
```

On peut noter que l'ordre des boucles dans l'Algorithme 3 permet d'affirmer que la valeur de k en ligne 4 est donnée par $\mathcal{F}(i-1, j-1)$.

Algorithm 4 Calcul de $\boldsymbol{F}$ avec construction de $\boldsymbol{X}$ et $\boldsymbol{Y}$ (version 3)

```
for j ← 1 to n_y do
  for i ← 1 to n_x do
    k ← BIJF(i − 1, j − 1, n_x)
    X(k) ← x(i)
    Y(k) ← y(j)
  end for
end for
F ← f(X,Y)
```

Grâce à l'application réciproque $\mathcal{F}^{-1}$ il est alors possible de transformer la double boucle en j et i en une seule boucle sur k : c'est l'objet de l'Algorithme 5. On en déduit alors l'Algorithme 6 totalement vectorisé.

Algorithm 5 Calcul de $\boldsymbol{F}$ avec construction de $\boldsymbol{X}$ et $\boldsymbol{Y}$ (version 4)

for $k \leftarrow 1$ **to** N **do**
  $[i,j] \leftarrow$ BIJRECF(k, n_x)
  $\boldsymbol{X}(k) \leftarrow \boldsymbol{x}(i+1)$
  $\boldsymbol{Y}(k) \leftarrow \boldsymbol{y}(j+1)$
end for
$\boldsymbol{F} \leftarrow f(\boldsymbol{X}, \boldsymbol{Y})$

Algorithm 6 Calcul de $\boldsymbol{F}$ avec construction de $\boldsymbol{X}$ et $\boldsymbol{Y}$ (version 5 : vectorisée)

$[I,J] \leftarrow$ BIJRECF$(1:N, n_x)$ ▷ Deja vectorisee
$\boldsymbol{X} \leftarrow \boldsymbol{x}(I+1)$
$\boldsymbol{Y} \leftarrow \boldsymbol{y}(J+1)$
$\boldsymbol{F} \leftarrow f(\boldsymbol{X}, \boldsymbol{Y})$

Un programme Matlab/Octave complet basé sur ce dernier algorithme est proposé en Listing 3.

Listing 2 – Calcul et représentation d'une fonction sur la grille 2D

```
clear all
close all
a=-pi;b=pi;c=0;d=pi;
f=@(x,y)  4*cos(x + y).*sin(x - y);
Nx=133;Ny=222;
x=linspace(a,b,Nx+1);
y=linspace(c,d,Ny+1);
nx=Nx+1;ny=Ny+1;N=nx*ny;

[I,J]=bijRecF(1:N,nx); % Naturellement vectorisee
X=x(I+1);Y=y(J+1);     % Attention X et Y vecteurs ↩
    ligne
F=f(X,Y);              % et donc F aussi!

surf(x,y,reshape(F,nx,ny)')
shading interp
```

Il faut noter qu'il existe sous Matlab/Octave deux fonctions **meshgrid** et `ndgrid` qui peuvent être utilées pour le calcul des vecteurs $\boldsymbol{X}$ et $\boldsymbol{Y}$. Ces deux fonctions n'utilisent pas la fonction `bijRecF` .
D'autres techniques peuvent aussi être utilisées, par exemple avec la fonction `repmat` et l'opérateur (:) (ou la fonction **reshape** ou la commande `subsref(X,substruct('()',{':'})`).

Q. 10 (Matlab) *Ecrire la fonction* `EvalFun2DVec` *permettant à partir d'une fonction f et des discrétisations $\boldsymbol{x}$ et $\boldsymbol{y}$ de retourner le vecteur $\boldsymbol{F}$ associé sans utiliser de boucle* ***for*** . ▪

3.2 A blocs

Chacune des équations du problème discret (3.3)-(3.7) correspond à une discrétisation en un point (x_i, y_j). Nous choisissons d'écrire ces équations en utilisant la même numérotation que lors de la construction du vecteur $\boldsymbol{V}$: l'équation *écrite* au point (x_i, y_j) sera écrite en ligne $k = \mathcal{F}(i,j)$ du système.

Q. 11 (sur feuille) *Etablir que le problème discret* (3.3)-(3.7) *peut s'écrire sous la forme du système linéaire bloc*

$$\left(\begin{array}{c|ccccc|c} \mathbb{E} & \mathbb{O} & \cdots & \cdots & \cdots & \mathbb{O} & \mathbb{O} \\ \hline \mathbb{M} & \mathbb{D} & \mathbb{M} & \mathbb{O} & \cdots & \mathbb{O} & \mathbb{O} \\ \mathbb{O} & \mathbb{M} & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \mathbb{O} & \ddots & \ddots & \ddots & \mathbb{O} & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \mathbb{M} & \mathbb{O} \\ \mathbb{O} & \mathbb{O} & \cdots & \mathbb{O} & \mathbb{M} & \mathbb{D} & \mathbb{M} \\ \hline \mathbb{O} & \mathbb{O} & \cdots & \cdots & \cdots & \mathbb{O} & \mathbb{E} \end{array}\right) \boldsymbol{V} = \left(\begin{array}{c} B_{:,0} \\ \hline B_{:,1} \\ \hline \vdots \\ \hline \vdots \\ \hline B_{:,N_y} \end{array}\right) \tag{3.9}$$

où chaque bloc de la matrice est une matrice (N_x+1) par $(Nx+1)$. La matrice $\mathbb{O} \in \mathcal{M}_{N_x+1}(\mathbb{R})$ est la matrice nulle. Les matrices creuses $\mathbb{D}$, $\mathbb{M}$ et $\mathbb{E}$ ainsi que les vecteurs $B_{:,j} \in \mathbb{R}^{Nx+1}$, pour tout $j \in [\![0, N_y]\!]$, devront être donnés explicitement. On notera $\boldsymbol{B}$ le vecteur second membre de $\mathbb{R}^N$ et $\mathbb{A} \in \mathcal{M}_N(\mathbb{R})$ la matrice du système. ▪

3.3 Assemblage de la matrice sans conditions aux limites

3.3.1 Méthode 1 : utilisation de la bijection

Cette méthode est celle qui va, au final, se rapprocher le plus des techniques d'assemblages utilisées par les méthodes d'éléments finis et volumes finis sur des domaines $\Omega \subset \mathbb{R}^d$ quelconques.

Nous avons vu que notre problème discret revient à résoudre le système linéaire

$$\mathbb{A}\boldsymbol{V} = \boldsymbol{B}. \tag{3.10}$$

Il nous faut donc construire/assembler la matrice $\mathbb{A} \in \mathcal{M}_N(\mathbb{R})$ et le second membre $\boldsymbol{B} \in \mathbb{R}^N$.

Nous allons commencer par générer/assembler la matrice correspondant à la prise en compte de toutes les équations de (3.3) sans tenir compte, pour le moment, des conditions aux limites (i.e. les lignes du système correspondant à des points du bord sont nulles ... pour l'instant). Nous noterons $-\mathbb{A}_{xy}$ cette matrice pour qu'elle corresponde à la matrice du Laplacien et non l'opposée du Laplacien.

Nous rappelons que nous avons choisi d'écrire l'équation (3.3) au point (x_i, y_j) en ligne $k = \mathcal{F}(i,j)$ du système.

Q. 12 (Matlab) *1. Sans aucune vectorisation/optimisation de code, écrire la fonction `Assemblage2DBijection00` retournant la matrice creuse $\mathbb{A}_{xy}$ du système sans tenir compte des conditions aux limites en utilisant toutes les équations de (3.3) et la fonction `bijF`. Il y aura ici deux boucles imbriquées, l'une en i et l'autre en j pour écrire chacune des équations en (i,j).*

2. Proposer au moins un programme permettant de tester/valider la matrice ainsi obtenue. ▪

Q. 13 (Matlab) *Proposer plusieurs variantes de vectorisation/optimisation de la fonction `Assemblage2DBijection00` : il faudrait supprimer au moins une des boucles en i ou en j...* ▪

3.3.2 Méthode 2 : produits de Kronecker (facultatif)

On note $\mathbb{I}_n$ la matrice identité de $\mathcal{M}_n(\mathbb{R})$ et $\mathbb{J}_n$ la matrice de $\mathcal{M}_n(\mathbb{R})$ définie par

$$\mathbb{J}_n = \begin{pmatrix} 0 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 0 \end{pmatrix} \in \mathcal{M}_n(\mathbb{R}).$$

Dans cette partie nous allons générer/assembler la matrice du système (3.9) **sans tenir compte des conditions aux limites** (i.e. les lignes du système correspondant à des points du bord sont nulles ... pour l'instant).

Dans ce cas, on note $-\mathbb{A}_{xy}$ la matrice ainsi obtenue. C'est une matrice bloc de $\mathcal{M}_N(\mathbb{R})$ avec n_y lignes bloc composées de blocs carrés de dimension n_x qui s'écrit sous la forme :

$$\mathbb{A}_{xy} = \left(\begin{array}{c|c|ccccc|c} \mathbb{O} & \mathbb{O} & \cdots & \cdots & \cdots & \mathbb{O} & \mathbb{O} \\ \hline \mathbb{O} & \mathbb{T}_x & \mathbb{O} & \mathbb{O} & \cdots & \mathbb{O} & \mathbb{O} \\ \mathbb{O} & \mathbb{O} & \mathbb{T}_x & \ddots & \ddots & \vdots & \vdots \\ \vdots & \mathbb{O} & \ddots & \ddots & \ddots & \mathbb{O} & \vdots \\ \vdots & \vdots & \ddots & \ddots & \mathbb{T}_x & \mathbb{O} & \mathbb{O} \\ \mathbb{O} & \mathbb{O} & \cdots & \mathbb{O} & \mathbb{O} & \mathbb{T}_x & \mathbb{O} \\ \hline \mathbb{O} & \mathbb{O} & \cdots & \cdots & \cdots & \mathbb{O} & \mathbb{O} \end{array}\right) + \left(\begin{array}{c|c|ccccc|c} \mathbb{O} & \mathbb{O} & \cdots & \cdots & \cdots & \mathbb{O} & \mathbb{O} \\ \hline \mathbb{S}_y & \mathbb{T}_y & \mathbb{S}_y & \mathbb{O} & \cdots & \mathbb{O} & \mathbb{O} \\ \mathbb{O} & \mathbb{S}_y & \mathbb{T}_y & \ddots & \ddots & \vdots & \vdots \\ \vdots & \mathbb{O} & \ddots & \ddots & \ddots & \mathbb{O} & \vdots \\ \vdots & \vdots & \ddots & \ddots & \mathbb{T}_y & \mathbb{S}_y & \mathbb{O} \\ \mathbb{O} & \mathbb{O} & \cdots & \mathbb{O} & \mathbb{S}_y & \mathbb{T}_y & \mathbb{S}_y \\ \hline \mathbb{O} & \mathbb{O} & \cdots & \cdots & \cdots & \mathbb{O} & \mathbb{O} \end{array}\right) \tag{3.11}$$

où

$$\mathbb{S}_y = \frac{1}{h_y^2}\mathbb{J}_{n_x}, \quad \mathbb{T}_y = -\frac{2}{h_y^2}\mathbb{J}_{n_x} \quad \text{et} \quad \mathbb{T}_x = \frac{1}{h_x^2}\begin{pmatrix} 0 & 0 & 0 & \cdots & \cdots & 0 \\ 1 & -2 & 1 & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & 1 & -2 & 1 \\ 0 & \dots & \dots & 0 & 0 & 0 \end{pmatrix}$$

On peut noter que les matrices $\mathbb{T}_x$, $\mathbb{T}_y$ et $\mathbb{S}_y$ sont des matrices de $\mathcal{M}_{n_x}(\mathbb{R})$.

En notant $\mathbb{A}_x \in \mathcal{M}_{n_x}(\mathbb{R})$ et $\mathbb{A}_y \in \mathcal{M}_{n_y}(\mathbb{R})$ les matrices définies par

$$\mathbb{A}_x = \frac{1}{h_x^2}\begin{pmatrix} 0 & 0 & 0 & \cdots & \cdots & 0 \\ 1 & -2 & 1 & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & 1 & -2 & 1 \\ 0 & \dots & \dots & 0 & 0 & 0 \end{pmatrix} \quad \text{et} \quad \mathbb{A}_y = \frac{1}{h_y^2}\begin{pmatrix} 0 & 0 & 0 & \cdots & \cdots & 0 \\ 1 & -2 & 1 & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & 1 & -2 & 1 \\ 0 & \dots & \dots & 0 & 0 & 0 \end{pmatrix}$$

on déduit de (3.11)

$$\mathbb{A}_{xy} = \mathbb{J}_{n_y} \otimes \mathbb{A}_x + \mathbb{A}_y \otimes \mathbb{J}_{n_x}. \tag{3.12}$$

Q. 14 (Matlab) *1. Ecrire la fonction* `Lap2DAssembling` *retournant la matrice bloc creuse* $\mathbb{A}$ *en utilisant* (3.12) *et les fonctions déjà écrites.*

2. Proposer au moins un programme permettant de tester/valider la matrice ainsi obtenue. ▪

Sur le même principe, il est possible de généraliser en dimension quelconque la construction de la matrice du Laplacien obtenue par différences finies sans prise en compte des conditions aux limites. Par exemple en dimension 3, cette matrice s'écrit

$$\begin{aligned} \mathbb{A}_{xyz} &= \mathbb{J}_{n_z} \otimes \mathbb{A}_{xy} + \left(\mathbb{A}_z \otimes (\mathbb{J}_{n_y} \otimes \mathbb{J}_{n_x})\right) \\ &= \mathbb{J}_{n_z} \otimes \mathbb{J}_{n_y} \otimes \mathbb{A}_x + \mathbb{J}_{n_z} \otimes \mathbb{A}_y \otimes \mathbb{J}_{n_x} + \mathbb{A}_z \otimes \mathbb{J}_{n_y} \otimes \mathbb{J}_{n_x} \end{aligned}$$

3.4 Quelques indices

3.4.1 Indices des bords

La frontière Γ du domaine peut se décomposer en 4 sous-ensembles :

$$\Gamma = \Gamma_N \cup \Gamma_S \cup \Gamma_E \cup \Gamma_O.$$

Q. 15 (sur feuille) *1. Pour* Γ_S*, donner le sous-ensemble* $\mathcal{D}_S$ *d'indices* (i, j) *dans* $[\![0, N_x]\!] \times [\![0, N_y]\!]$ *tel que* $(x(i), y(j)) \in \Gamma_S$.

2. En déduire le sous-ensemble $\mathcal{I}_S$ *(ordonné suivant* $\mathcal{D}_S$*) correspondant dans la numérotation globale.*

3. Même chose pour Γ_N, Γ_E *et* Γ_O. ▪

Q. 16 (matlab) *Ecrire la fonction* `BordDomaine` *retournant la structure de données (*`help struct`*) contenant les champs suivants :*

- `Sud` *ayant pour valeur* $\mathcal{I}_S$,
- `Nord` *ayant pour valeur* $\mathcal{I}_N$,
- `Est` *ayant pour valeur* $\mathcal{I}_E$,
- `Ouest` *ayant pour valeur* $\mathcal{I}_O$.

Chacun des tableaux $\mathcal{I}_x$ *sera stocké sous forme ligne.* ▪

Une fois cette fonction écrite, il est aisé en numérotation globale de récupérer l'ensemble `idxBD` des indices des points du bords ainsi que l'ensemble `idxBDc` des indices des points strictement intérieurs (complémentaire du précédant) :

```
BD=BordDomaine(...);  % ... -> a remplacer
idxBD=unique([BD.Sud,BD.Nord,BD.Est,BD.Ouest]);
idxBDc=setdiff(1:N,idxBD);  % N=nx*ny
```

3.4.2 Evaluation de fonctions sur une partie de la grille

Soit $g : \Gamma_N \longrightarrow \mathbb{R}$. On suppose cette fonction vectorisée sous Matlab/Octave. Pour évaluer cette fonction aux points de la grille 2D appartenant à Γ_N, on utilise la structure de données retournée par la fonction `BordDomaine` et plus particulièrement le champ `Nord` de cette structure. Voici une manière de créer un vecteur `G` valant `G(k)=g(x(i),y(j))` sur un point du bord `Nord` avec $k = \mathcal{F}(i, j)$ et 0 sinon.

Listing 3 – Calcul d'une fonction sur le bord Nord de la grille 2D

```
BD=BordDomaine(...);  % ... -> a remplacer
[I,J]=bijRecF(1:N,nx);
X=x(I+1);Y=y(J+1);% coordonnees des points de la grille
                  % en numerotation globale
g=@(x,y) sin(x+y).*x+y.^2;
Gloc=g(X(BD.Nord),Y(BD.Nord));
G=zeros(N,1);
G(BD.Nord)=Gloc;
```

Pour évaluer une fonction sur une autre partie de la grille, il suffit de récupérer l'ensemble des indices dans la numérotation globale des points de appartenant à cette partie et de remplacer `BD.Nord` par cet ensemble.

3.5 Assemblage du second membre sans conditions aux limites

Nous allons générer/assembler le vecteur second membre du système (3.9) **sans tenir compte des conditions aux limites**. Avec le choix de la numérotation globale, ce vecteur $\underline{\mathbf{B}}$ de $\mathbb{R}^N$ est défini par

$$\forall k \in [\![1, N]\!], \quad \underline{\mathbf{B}}_k = \begin{cases} f(x(i), y(j)) & \text{si } (x(i), y(j)) \notin \Gamma \text{ et } k = \mathcal{F}(i, j) \\ 0 & \text{sinon} \end{cases}$$

En utilisant les codes donnés (et expliqués) en section 3.4, on a immédiatement un code vectorisé permettant d'initialiser ce vecteur : il est donné en Listing 4.

Listing 4 – Calcul du vecteur second membre sans conditions aux limites

```
BD=BordDomaine(...);  % ... -> a remplacer
[I,J]=bijRecF(1:N,nx);
X=x(I+1);Y=y(J+1);% coordonnees des points de la grille
                  % en numerotation globale
f=@(x,y) sin(x+y).*cos(x-y);

idxBD=unique([BD.Sud,BD.Nord,BD.Est,BD.Ouest]);
idxBDc=setdiff(1:N,idxBD);  % N=nx*ny

B=zeros(N,1);
B(idxBDc)=f(X(idxBDc),Y(idxBDc));
```

3.6 Prise en compte des conditions aux limites

Sans tenir compte des conditions aux limites, nous avons décrit le calcul de la matrice $\mathbb{A}_{xy} \in \mathcal{M}_N(\mathbb{R})$ (section 3.3) et du vecteur $\underline{\mathbf{B}} \in \mathbb{R}^N$ (section 3.5).

En reprenant les résultats et notations de **Q.11** (page 9), on en déduit que

$$\mathbb{A} = -\mathbb{A}_{xy} + \mathbb{A}_\Gamma \quad \text{et} \quad \boldsymbol{B} = \underline{\mathbf{B}} + \boldsymbol{B}_\Gamma$$

où $\mathbb{A}_\Gamma$ et $\boldsymbol{B}_\Gamma$ sont les *contributions* des conditions aux limites, c'est à dire que le système

$$\mathbb{A}_\Gamma \boldsymbol{U} = \boldsymbol{B}_\Gamma \tag{3.13}$$

contient uniquement les équations (3.4) à (3.7). Pour $k = \mathcal{F}(i,j) \in [\![1, N]\!]$, si $(x(i), y(j)) \in \Gamma$ alors la ligne k du système (3.13) correspond à

$$\boldsymbol{U}(k) = g(x(i), y(j)).$$

Toutes les autres lignes du système (3.13) sont nulles. On a donc pour tout $k \in [\![1, N]\!]$

$$\mathbb{A}_\Gamma(k,:) = \begin{cases} \boldsymbol{e}_k^\intercal & \text{si } (x(i), y(j)) \in \Gamma \text{ avec } (i,j) = \mathcal{F}^{-1}(k) \\ 0 & \text{sinon} \end{cases}$$

où $\boldsymbol{e}_k^\intercal$ est le $k^{\text{ième}}$ vecteur de la base canonique de $\mathbb{R}^N$, et

$$\boldsymbol{B}_\Gamma(k) = \begin{cases} g(x(i), y(j)) & \text{si } (x(i), y(j)) \in \Gamma \text{ avec } (i,j) = \mathcal{F}^{-1}(k) \\ 0 & \text{sinon} \end{cases}$$

En utilisant les codes donnés (et expliqués) en section 3.4, on calcule dans le Listing 5, le vecteur `Bcl` correspondant à $\boldsymbol{B}_\Gamma$ et la matrice `Acl` correspondant à $\mathbb{A}_\Gamma$.

Listing 5 – Calcul des contributions du bord

```
BD=BordDomaine(...);  % ... -> a remplacer
[I,J]=bijRecF(1:N,nx);
X=x(I+1);Y=y(J+1);% coordonnees des points de la grille
                  % en numerotation globale
g=@(x,y) sin(x+y).*cos(x-y);

idxBD=unique([BD.Sud,BD.Nord,BD.Est,BD.Ouest]);
idxBDc=setdiff(1:N,idxBD);  % N=nx*ny

Bcl=zeros(N,1);
Bcl(idxBD)=g(X(idxBD),Y(idxBD));
Acl=sparse(idxBD,idxBD,ones(1,length(idxBD)),N,N);
```

3.7 Résolution du système

On doit résoudre le système $\mathbb{A}\boldsymbol{U} = \boldsymbol{B}$ définit en **Q.11** (page 9). Or on a construit les matrices $\mathbb{A}_{xy}$, $\mathbb{A}_\Gamma$ et les vecteurs $\underline{\mathbf{B}}$, $\boldsymbol{B}_\Gamma$ de telle sorte que

$$\mathbb{A} = -\mathbb{A}_{xy} + \mathbb{A}_\Gamma \quad \text{et} \quad \boldsymbol{B} = \underline{\mathbf{B}} + \boldsymbol{B}_\Gamma$$

3.7.1 Méthode 1

Dans cette section nous allons directement résoudre le système $\mathbb{A}\boldsymbol{U} = \boldsymbol{B}$.

Q. 17 *1. Ecrire le programme `EDP2D01` permettant de résoudre numériquement l'EDP (3.1)-(3.2) à l'aide du schéma discrétisé (3.3) à (3.7). On choisira judicieusement les données permettant sur une même figure de représenter de la solution numérique et de la solution exacte, et sur une autre figure de représenter l'erreur entre les deux solutions.*

2. Ecrire le programme `ordreEDP2D01` permettant de retrouver numériquement l'ordre du schéma utilisé. ▪

3.7.2 Méthode 2

Dans cette section nous proposons une autre méthode permettant de résoudre le système $\mathbb{A}\boldsymbol{U} = \boldsymbol{B}$ en *éliminant* les équations portants sur les points Dirichlet.

Pour celà, on note

$$\mathcal{I}_D = \left\{k \in [\![1, N]\!];\ (x(i), y(j)) \in \Gamma \text{ avec } (i,j) = \mathcal{F}^{-1}(k)\right\}$$

et son complémentaire

$$\mathcal{I}_D^c = [\![1, N]\!] \backslash \mathcal{I}_D.$$

Le système linéaire à résoudre $\mathbb{A}\boldsymbol{U} = \boldsymbol{B}$ peut se décomposer de manière équivalente en

$$\mathbb{A}\boldsymbol{U} = \boldsymbol{B} \Leftrightarrow \begin{cases} \mathbb{A}(\mathcal{I}_D, :)\boldsymbol{U} = \boldsymbol{B}(\mathcal{I}_D) & \text{(3.14a)} \\ \mathbb{A}(\mathcal{I}_D^c, :)\boldsymbol{U} = \boldsymbol{B}(\mathcal{I}_D^c) & \text{(3.14b)} \end{cases}$$

Nous allons réécrire les deux systèmes (3.14a) et (3.14b).

- Pour (3.14a), nous avons, par construction, $\mathbb{A}(\mathcal{I}_D, :) = \mathbb{A}_\Gamma(\mathcal{I}_D, :)$ et donc

$$\begin{aligned} \mathbb{A}(\mathcal{I}_D, :)\boldsymbol{U} &= \mathbb{A}_\Gamma(\mathcal{I}_D, :)\boldsymbol{U} \\ &= \mathbb{A}_\Gamma(\mathcal{I}_D, \mathcal{I}_D)\boldsymbol{U}(\mathcal{I}_D) + \mathbb{A}_\Gamma(\mathcal{I}_D, \mathcal{I}_D^c)\boldsymbol{U}(\mathcal{I}_D^c) \end{aligned}$$

  Comme $\mathbb{A}_\Gamma(\mathcal{I}_D, \mathcal{I}_D^c) = \mathbb{O}$ et $\mathbb{A}_\Gamma(\mathcal{I}_D, \mathcal{I}_D) = \mathbb{I}$ on obtient

$$\mathbb{A}(\mathcal{I}_D, :)\boldsymbol{U} = \boldsymbol{U}(\mathcal{I}_D).$$

  Donc (3.14a) est équivalent à

$$\boldsymbol{U}(\mathcal{I}_D) = \boldsymbol{B}(\mathcal{I}_D). \tag{3.15}$$

- Pour (3.14b), nous avons par construction $\mathbb{A}(\mathcal{I}_D^c, :) = -\mathbb{A}_{x,y}(\mathcal{I}_D^c, :)$ et donc

$$\begin{aligned} \mathbb{A}(\mathcal{I}_D^c, :)\boldsymbol{U} &= -\mathbb{A}_{x,y}(\mathcal{I}_D^c, :)\boldsymbol{U} \\ &= -\mathbb{A}_{x,y}(\mathcal{I}_D^c, \mathcal{I}_D)\boldsymbol{U}(\mathcal{I}_D) - \mathbb{A}_{x,y}(\mathcal{I}_D^c, \mathcal{I}_D^c)\boldsymbol{U}(\mathcal{I}_D^c) \end{aligned}$$

  En utilisant (3.15), le système (3.14b) s'écrit

$$-\mathbb{A}_{x,y}(\mathcal{I}_D^c, \mathcal{I}_D^c)\boldsymbol{U}(\mathcal{I}_D^c) = \boldsymbol{B}(\mathcal{I}_D^c) + \mathbb{A}_{x,y}(\mathcal{I}_D^c, \mathcal{I}_D)\boldsymbol{B}(\mathcal{I}_D). \tag{3.16}$$

Résoudre le système linéaire $\mathbb{A}\boldsymbol{U} = \boldsymbol{B}$ est alors équivalent à calculer $\boldsymbol{U}(\mathcal{I}_D)$ par (3.15) et déterminer $\boldsymbol{U}(\mathcal{I}_D^c)$ en résolvant le système linéaire (3.16).

Q. 18 *1. Ecrire le programme* `EDP2D02` *permettant de résoudre numériquement l'EDP* (3.1)-(3.2) *à l'aide du schéma discrétisé* (3.3) *à* (3.7) *que l'on résoudra par* (3.15) *et* (3.16)*. On choisira judicieusement les données permettant sur une même figure de représenter de la solution numérique et de la solution exacte, et sur une autre figure de représenter l'erreur entre les deux solutions.*

2. Ecrire le programme `ordreEDP2D02` *permettant de retrouver numériquement l'ordre du schéma utilisé.* ▪