MA261. Introduction au calcul scientifique
Corrigé 1 : Introduction à Matlab.

Exercice 1 Soient les vecteurs colonnes et la matrice suivants

$$\vec{u}_1 = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix},\ \vec{u}_2 = \begin{pmatrix} -5 \\ 2 \\ 1 \end{pmatrix},\ \vec{u}_3 = \begin{pmatrix} -1 \\ -3 \\ 7 \end{pmatrix},\ A = \begin{pmatrix} 2 & 3 & 4 \\ 7 & 6 & 5 \\ 2 & 8 & 7 \end{pmatrix}.$$

1. **Structures Matlab**
   (a) Entrer ces données sous Matlab.
   (b) Calculer $\vec{u}_1 + 3\vec{u}_2 - \vec{u}_3/5$.
   (c) Calculer le produit scalaire entre les vecteurs $\vec{u}_1$ et $\vec{u}_2$.
   (d) Calculer le produit $A\vec{u}_1$.
2. **Commandes Matlab**
   Trouver les commandes Matlab permettant de :
   (a) calculer $\|\vec{u}_1\|_2$, $\|\vec{u}_2\|_1$, $\|\vec{u}_3\|_\infty$;
   (b) déterminer les dimensions de la matrice A, en extraire le nombre de colonnes ;
   (c) calculer le déterminant et l'inverse de A.
3. **Résolution de systèmes linéaires**
   Proposer deux méthodes permettant de résoudre le problème $A\vec{x} = \vec{u}_1$, et déterminer les commandes Matlab associées.

Corrigé 1 Taper en ligne :

```
u1 = [ 1 ; 2 ; 3 ]
u2 = [ -5 ; 2 ; 1 ]
u3 = [ -1 ; -3 ; 7 ]
A = [ 2 3 4 ; 7 6 5 ; 2 8 7 ]
u1+3*u2-u3/5
u1'*u2
A*u1
norm(u1,2)
norm(u2,1)
norm(u3,inf)
size(A)
size(A,2)
det(A)
inv(A)
x = inv(A)*u1
x = A\u1
```

Exercice 2 Soient la matrice et les vecteurs colonnes suivants

$$A = \begin{pmatrix} 5/8 & -1/4 & 1/8 \\ 1/4 & 0 & 1/4 \\ 1/8 & -1/4 & 5/8 \end{pmatrix}, \vec{b} = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}, \vec{u}_1 = \begin{pmatrix} 5 \\ 2 \\ -4 \end{pmatrix}.$$

On définit, pour $n \geq 1$, la suite de vecteurs $\vec{u}_{n+1} = A\vec{u}_n + \vec{b}$.

1. Construire une fonction `suite.m` calculant les premiers termes de la suite $\vec{u}_n$. Cette fonction aura comme arguments d'entrée les données suivantes : la matrice A, le second membre $\vec{b}$, le terme initial $\vec{u}_1$, et le nombre de termes voulus nb_{it}.
2. Représenter graphiquement l'évolution de chacune des composantes.
   Qu'observe-t-on ?
3. Soient
$$\vec{u}_{1b} = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix}, A_b = \begin{pmatrix} 5 & 6 & 3 \\ -1 & 5 & -1 \\ 1 & 2 & 0 \end{pmatrix}.$$
   Observe-t-on le même comportement si on remplace $\vec{u}_1$ par $\vec{u}_{1b}$? Que se passe-t-il si on remplace A par A_b (quel que soit le terme initial) ?

Corrigé 2 Dans le fichier `suite.m`, créer la fonction :

```
function u = suite(A,u1,b,nb_it)
u = zeros(3,nb_it+1);
u(:,1) = u1;
for k = 1:nb_it
  u(:,k+1) = A*u(:,k)+b;
end
```

En sortie, le tableau `u` contient les itérés successifs $(\vec{u}_n)_{1 \leq n \leq nb_{it}}$.
Puis, taper en ligne :

```
A = [ 5/8 -1/4 1/8 ; 1/4 0 1/4 ; 1/8 -1/4 5/8 ]
b = [ 1 ; -1 ; 1 ]
u1 = [ 5 ; 2 ; -4 ]
u = suite(A,u1,b,30);
```

Pour étudier le comportement des itérés, taper en ligne :

```
hold on
plot(u(1,:),'g')
plot(u(2,:),'r')
plot(u(3,:),'y')
hold off
```

Les itérations convergent visuellement vers $\begin{pmatrix} 10/3 \\ 2/3 \\ 10/3 \end{pmatrix}$.

```
u1b = [ 2 ; 1 ; 0 ];
ub = suite(A,u1b,b,30);
hold on
plot(ub(1,:),'g')
plot(ub(2,:),'r')
plot(ub(3,:),'y')
hold off
```

Les itérations convergent visuellement vers $\begin{pmatrix} 10/3 \\ 2/3 \\ 10/3 \end{pmatrix}$.

```
Ab = [ 5 6 3 ; -1 5 -1 ; 1 2 0 ]
```

Pour les deux termes initiaux $\vec{u}_1$ et $\vec{u}_{1b}$, la suite diverge.
Pour des itérations du type $\vec{u}_{n+1} = A\vec{u}_n + \vec{b}$, il y a convergence si, et seulement si, le rayon spectral de A est strictement inférieur à un. On rappelle que si $(\lambda_i(A))_i$ sont les valeurs propres de A (dans $\mathbb{C}$), alors on a par définition

$$\rho(A) = \max_i |\lambda_i(A)|.$$

Or, on trouve $\rho(A) = 1/2$ et $\rho(A_b) \approx 5.9$.

Exercice 3 Soit $A \in \mathbb{R}^{n\times n}$. On introduit le vecteur p_L^A (resp. p_C^A), appartenant à $\mathbb{R}^n$, des indices des colonnes (resp. des lignes) du premier coefficient non nul de chaque ligne (resp. de chaque colonne). Par convention, si tous les coefficients d'une ligne (resp. d'une colonne) sont nuls, le nombre reporté est $n+1$. Par exemple, pour la matrice

$$A = \begin{pmatrix} 5 & 6 & 3 & 0 \\ 0 & 5 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 4 \end{pmatrix} \in \mathbb{R}^{4\times 4}, \text{on a } p_L^A = \begin{pmatrix} 1 \\ 2 \\ 5 \\ 3 \end{pmatrix} \text{ et } p_C^A = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 4 \end{pmatrix}.$$

Les profils en ligne P_L^A et en colonne P_C^A peuvent alors se définir comme étant :

$$P_L^A = \left\{(i,j) \in \{1,\dots,n\}^2 \ : \ (p_L^A)_i \leq j\right\}.$$

$$P_C^A = \left\{(i,j) \in \{1,\dots,n\}^2 \ : \ (p_C^A)_j \leq i\right\}.$$

Leur utilité première est d'éviter de stocker les composantes nulles en début de chaque ligne ou colonne.

1. Écrire une fonction qui pour toute matrice calcule son profil ligne et colonne, i.e. renvoie les vecteurs p_L^A et p_C^A.
   `Aide : utiliser la fonction Matlab retournant le minimum d'un ensemble.`
2. Comment pourrait-on simplement améliorer les profils en ligne et en colonne ?
3. Que peut-on dire des profils des matrices symétriques et symétriques définies-positives ? Modifier en conséquence votre fonction.

Corrigé 3 1. Dans `Profil.m`, créer la fonction `Profil` :

```
function [pL,pC] = Profil(A)
N = size(A,1);
pL = zeros(N,1);
pC = zeros(N,1);
for I = 1:N
  pL(I) = min(find(A(I,:)));
  pC(I) = min(find(A(:,I)));
end
```

En sortie, les vecteurs colonnes `pL` et `pC` contiennent les valeurs de p_L^A et p_C^A avec la correspondance `pL(I) = 0` $\iff (p_L^A)_i = n+1$ (resp. `pC(I) = 0` $\iff (p_C^A)_i = n+1$.)

2. Il suffit de définir deux vecteurs q_L^A (resp. q_C^A) des indices des colonnes (resp. des lignes) du dernier coefficient non nul de chaque ligne (resp. de chaque colonne), avec un nombre reporté égal à 0 si tous les coefficients sont nuls. Les profils correspondants sont :
$$P_L^A = \left\{(i,j) \in \{1,\dots,n\}^2 \ : \ (p_L^A)_i \leq j \leq (q_L^A)_i\right\}.$$
$$P_C^A = \left\{(i,j) \in \{1,\dots,n\}^2 \ : \ (p_C^A)_j \leq i \leq (q_C^A)_j\right\}.$$

3. Lorsqu'une matrice A est symétrique, on a toujours $p_L^A = p_C^A$: il suffit d'introduire un unique vecteur p^A.
Si de plus, la matrice est définie-positive, on remarque que $A_{i,i} = (A\vec{e}_i, \vec{e}_i) > 0$, et ainsi $(p^A)_i \leq i$ pour tout i.

   Par ailleurs, si A est symétrique définie-positive, on peut utiliser la factorisation de Cholesky pour résoudre les systèmes linéaires du type $A\vec{x} = \vec{b}$.
   Dans un premier temps, on calcule la matrice triangulaire inférieure L telle que $A = LL^\mathsf{T}$. Dans un second temps, on résout successivement $L\vec{y} = \vec{b}$ et $L^\mathsf{T}\vec{x} = \vec{y}$ par un algorithme de descente-remontée.
   Bien sûr, pour une matrice L triangulaire inférieure, on sait que $i,j = 0$ dès que $j > i$. Le *profil optimisé* d'une telle matrice est donc
$$P_{opt}^L = \left\{(i,j) \in \{1,\dots,n\}^2 \ : \ (p^L)_i \leq j \leq i\right\}.$$
   Et, pour conclure, si on définit le *profil optimisé* d'une matrice SDP comme étant :
$$P_{opt}^A = \left\{(i,j) \in \{1,\dots,n\}^2 \ : \ (p^A)_i \leq j \leq i\right\},$$
   son intérêt principal est d'être conservé par la factorisation de Cholesky ! En d'autres termes, on a la propriété : $P_{opt}^L = P_{opt}^A$.

   Dans le fichier `ProfSDP.m`, créér la fonction `ProfSDP` :

```
function p = ProfSPD(A)
% On suppose que A est SPD.
N = size(A,1);
p = zeros(N,1);
for I = 1:N
  p(I) = min(find(A(:,I)));
end
```

En sortie, le vecteur colonne `p` contient les valeurs de p.

Exercice 4 1. Écrire une fonction Matlab `GenereMatrice` générant une matrice d'ordre quelconque avec des éléments aléatoires.

2. Écrire une fonction Matlab `GenereSysteme` construisant des systèmes linéaires aléatoires, et les résolvant (lorsque c'est possible).

Corrigé 4 1. Dans le fichier `GenereMatrice.m`, créer la fonction :

```
function A = GenereMatrice(m,n,Amax)
A = Amax*rand(m,n);
```

A est une matrice $\mathbb{R}^{m\times n}$, dont les éléments sont compris entre 0 et A_{max}. On peut aussi utiliser `randn` pour avoir des éléments négatifs, ou `sprand` et `sprandn` lorsque m et n sont très grands.

2. Dans le fichier `GenereSysteme.m`, créer la fonction :

```
function [A,b,x,msg] = GenereSysteme(m,Amax,bmax)
A = GenereMatrice(m,m,Amax);
b = bmax*rand(m,1);
if det(A)~=0
  x = A\b;
  msg = 'ok';
else
  x = 0;
  msg = 'pas de solution';
end
```

Mise en œuvre informatique à l'aide de Matlab

1. *Points à retenir* :
   - Lancement de Matlab.
   - Commandes `help` et `lookfor` ; utilisation de la *notice*.
   - Création d'un répertoire pour ranger les fichiers Matlab.
   - Ecriture d'une fonction `xxx` et sauvegarde dans un fichier `xxx.m`.
2. *Fonctionnalités Matlab à maîtriser* :
   - Initialisation (structures *pleines* par défaut) pour vecteurs et matrices :
     `u = zeros(N,1), A = zeros(N,M)`.
   - Générateur *aléatoire* pour construction d'exemples :
     `u = rand(N,1), A = rand(N,M)`.
   - Boucles : `for`.
   - Tests : `if, else`.
   - Remise à zéro des variables : `clear all`.
   - Commandes pour l'affichage
     en ligne : `'message'` ;
     sous forme graphique : `plot(u)` (ligne brisée), `plot(u,'o')` (points).
   - Ecriture *symbolique*, sous la forme `u'*v, A*u, A\b`, etc.
   - Commandes pour les normes : `norm(u,1), norm(u,2), norm(u,inf)`.

MA261. Introduction au calcul scientifique
Corrigé 2 : Construction de matrices pour discrétisation par différences finies.

Exercice 5 : le Laplacien 1d (fil pesant, poutre en flexion).
Discrétisation du problème du fil pesant : *trouver u tel que*

$$-u'' = f \text{ sur }]0,1[, \quad u(0) = u(1) = 0.$$

1. Ecrire une fonction qui construise la matrice tridiagonale $\mathbb{A}_1$ d'ordre N obtenue par la discrétisation par différences finies du problème du fil pesant ou de la poutre en flexion (schéma à trois points pour le Laplacien 1d).
2. Comparer les temps de construction ainsi que la faisabilité, selon que la structure associée à $\mathbb{A}_1$ est pleine ou creuse, en fonction de N.

Corrigé 5 1. On rappelle que $N = n$, où n est égal au nombre de points de discrétisation intérieurs à l'intervalle de calcul, et que la matrice $\mathbb{A}_1 \in \mathbb{R}^{N\times N}$ est de la forme :

$$\mathbb{A}_1 = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}, \text{ avec } h = \frac{1}{(n+1)^2}.$$

On introduit à la fois N et n, bien qu'ils soient égaux ici en 1d (ce qui n'est plus le cas en 2d ou 3d !)
Dans le fichier `Laplacien1d.m`, créer la fonction :

```
function A1 = Laplacien1d(n)
N = n;
A1 = sparse(N,N);
for i = 1:n
  A1(i,i) = 2;
end
for i = 2:n
  A1(i,i-1) = -1;
end
for i = 1:n-1
  A1(i,i+1) = -1;
end
A1 = (n+1)^2*A1;
```

On peut examiner la structure de la matrice en tapant

```
A1 = Laplacien1d(100);
spy(A1)
```

2. On peut créer une fonction `Laplacien1dplein(n)` qui renvoie la même matrice, en *stockage plein*, en remplaçant `A1 = sparse(N,N)` par `A1 = zeros(N,N)`.

Pour mesurer le temps de construction, on peut par exemple taper en ligne

```
tic;A = Laplacien1d(1000);toc
tic;A = Laplacien1dplein(1000);toc
```

On constate que les temps d'exécution sont comparables. Par contre, lorsque N devient grand, des problèmes de capacité mémoire apparaissent pour le stockage plein !

```
tic;A = Laplacien1d_plein(10000);toc
??? Error using ==> zeros
Out of memory. Type HELP MEMORY for your options.
```

Il serait également possible de construire une fonction `TempsCalcul(n0,n1,ni)` qui construise les matrices $\mathbb{A}_1$ pour des valeurs de n variant de n_0 à n_1 par incrément de n_i, et qui mesure le temps de construction à l'aide de la commande `cputime`.

Exercice 6 : le Laplacien 2d (membrane élastique).

Discrétisation du problème de la membrane, posé dans $\Omega_2 =]0,1[^2$: *trouver u tel que*

$$-\Delta_2 u = f \text{ sur } \Omega_2, \quad u = 0 \text{ sur } \partial\Omega_2.$$

Le but de cet exercice est de construire *rapidement* la matrice $\mathbb{A}_2$ d'ordre N obtenue par la discrétisation par différences finies du problème de la membrane élastique (schéma à cinq points pour le Laplacien 2d).

Pour cela, on dispose d'une grille de $N = n \times n$ points, représentant les points de discrétisation *intérieurs* du domaine de calcul. D'un point de vue *algorithmique*, on veut utiliser des vecteurs de $\mathbb{R}^N$, dont les composantes correspondent à des valeurs aux points de la grille, ainsi que des matrices de $\mathbb{R}^{N\times N}$, qui représentent une action sur ces vecteurs.

1. Pourquoi les points de discrétisation de la frontière du domaine ne sont-ils pas pris en compte ? Comment numéroter les points de la grille ? Quels sont les voisins des points du bord de la grille ?
2. Soit $\mathbb{A}_2$ la matrice de $\mathbb{R}^{N\times N}$ telle que :
   Pour $I \in \{1, \cdots, N\}$: $(\mathbb{A}_2)_{I,I} = 4(n+1)^2$.
   Pour $I, J \in \{1, \cdots, N\}$, $I \neq J$:
   $(\mathbb{A}_2)_{I,J} = -(n+1)^2$ si I et J sont deux points voisins sur la grille (voisins de gauche, droite, haut, bas).
   $(\mathbb{A}_2)_{I,J} = 0$ sinon.
   Pourquoi cette matrice $\mathbb{A}_2$ est-elle la matrice du Laplacien 2d ?
3. Proposer un algorithme *rapide* de construction de $\mathbb{A}_2$. Ecrire la fonction Matlab correspondante. Effectuer une étude du temps de construction en fonction de N.

Corrigé 6 1. Les conditions aux limites imposent la valeur à la frontière du domaine de calcul. Les inconnues restantes du problème sont donc les n^2 valeurs prises aux points intérieurs au domaine, ce qui correspond exactement à la grille.
Une numérotation conseillée de la grille est la suivante : de gauche à droite, et de bas en haut, avec
- **un** indice I, variant de 1 à N, ou
- **deux** indices (i, j) variant de 1 à n.

Correspondance :

$$I-1=(i-1)+(j-1)n \quad \text{ou} \quad \begin{cases} i=(I-1) \mod n+1 \\ j=\lfloor (I-1)/n \rfloor +1 \end{cases} . \tag{1}$$

Considèrons les voisins *sur la grille* du point $I=18$ qui correspond au couple $(i,j)=(9,2)$. Son voisin de dessous est le point numéroté $I=9$ ($(i,j)=(9,1)$), celui du dessus est le numéro $I=27$ ($(i,j)=(9,3)$), celui de gauche est le numéro $I=17$ ($(i,j)=(8,2)$) et le dernier, celui de droite est un point de la frontière du domaine dont la valeur est nulle et non pas le numéro $I=19$ ($(i,j)=(3,1)$). En d'autres termes, le point $I=18$ n'admet pas de voisin de droite *sur la grille*.

Identification des points du bord de la grille.
- Avec le *double* indiçage (i,j), un point est sur le bord si, et seulement si, $i \in \{1,n\}$ ou $j \in \{1,n\}$.
- Pour l'indice I, ceci correspond à (cf. (1))

$$\begin{cases} (I \mod n) \in \{1,0\} \text{ ou} \\ I<n+1,\ N-n<I. \end{cases}$$

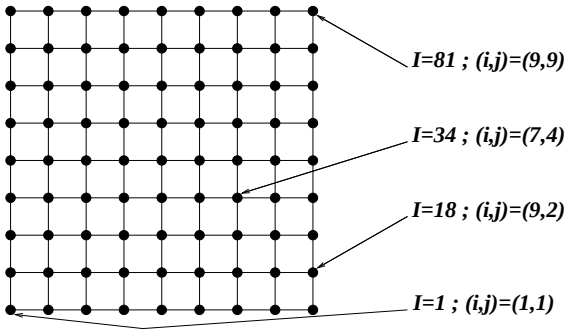

FIG. 1 – Avec $n=9$, $N=81$: $I \in \{1,\cdots,81\}$ et $i,j \in \{1,\cdots,9\}$

2. Par définition, on retrouve le schéma à cinq points, avec la pondération de $4/h^2$ pour le point considéré, et de $-1/h^2$ pour ses voisins immédiats, ce qui correspond bien à la discrétisation de $-\Delta_2 u=-\partial^2_{xx}u-\partial^2_{yy}u$ par différences finies...
3. Construction *rapide* de la matrice $\mathbb{A}_2$.
   Rapide = algorithme en $O(N)$ opérations, pour une matrice $N \times N$.

On peut par exemple, dans le fichier `Laplacien2d.m` créer la fonction

```
function A2 = Laplacien2d(n)
N = n^2;
A2 = 4*speye(N,N);
for j = 1:n
  for i = 1:n
    I = i+(j-1)*n;
    % I n'est pas un point du bord gauche donc il a un voisin de gauche
    if i~=1
      A2(I,I-1) = -1;
    end
```

```
    % I n'est pas un point du bord droit donc il a un voisin de droite
    if i~=n
      A2(I,I+1) = -1;
    end
    % I n'est pas un point du bord bas donc il a un voisin de dessous
    if j~=1
      A2(I,I-n) = -1;
    end
    % I n'est pas un point du bord haut donc il a un voisin de dessus
    if j~=n
      A2(I,I+n) = -1;
    end
  end
end
% On multiplie par 1/h^2 = (n+1)^2
A2 = (n+1)^2*A2;
```

Un autre algorithme possible de construction de la matrice $\mathbb{A}_2$ est :

```
function A2 = Laplacien2d(n)
N = n^2;
% (i) Comme la matrice A2 est symetrique, on se concentre sur
% la partie triangulaire superieure, ie. A2(I,J) pour J > I...
% (ii) Pas besoin d'initialisation par A2 = sparse(N,N), car celle-ci
% est contenue dans la premiere affectation.

% 1. Les voisins de droite : A2(I,I+1) = -1 pour tout I < N,
tmp = [[sparse(N-1,1) (-1)*speye(N-1)]; sparse(1,N)];
% Sauf s'il n'y a pas de voisin de droite...
% On effectue donc la correction
for j = 1:n-1
  I = j*n;
  tmp(I,I+1) = 0;
end
A2 = tmp;

% 2. Les voisins du haut : A2(I,I+n) = -1 pour tout I < N-n+1,
tmp = [[sparse(N-n,n) (-1)*speye(N-n)]; sparse(n,N)];
A2 = A2 + tmp;

% La partie triangulaire superieure est construite !

% 3. On symetrise la matrice : A2(I,J) = A2(J,I) pour tout I different de J,
A2 = A2 + A2';

% 4. On met a jour la diagonale : A2(I,I) = 4 pour tout I,
A2 = A2 + 4*speye(N);

% 5. On multiplie par 1/h^2 = (n+1)^2
A2 = (n+1)^2*A2;
```

Exercice 7 : le Laplacien 3d (cavité électrostatique).
Discrétisation du problème de la cavité, posé dans $\Omega_3 =]0,1[^3$: *trouver u tel que*

$$-\Delta_3 u = f \text{ sur } \Omega_3, \quad u = 0 \text{ sur } \partial\Omega_3.$$

On dispose maintenant d'une grille de $N = n \times n \times n$ points, représentant les points de discrétisation *intérieurs* du domaine de calcul. Comme pour l'exercice 2, les vecteurs à manipuler appartiennent à $\mathbb{R}^N$, et les matrices qui représentent une action sur ces vecteurs sont dans $\mathbb{R}^{N\times N}$.

1. Comment numéroter les points de la grille ?
2. A quoi est égale la matrice $\mathbb{A}_3$ du Laplacien 3d ? Proposer un algorithme *rapide* de construction de $\mathbb{A}_3$. Ecrire la fonction Matlab correspondante. Effectuer une étude du temps de construction en fonction de N.

Corrigé 7 1. (i) Numérotation conseillée pour la grille : de gauche à droite, de bas en haut, et de l'avant vers l'arrière avec
- **un** indice I, variant de 1 à N, ou
- **trois** indices (i,j,k) variant de 1 à n.
Correspondance

$$(I-1) = (i-1) + (j-1)n + (k-1)n^2 \quad \text{ou} \quad \left\{ \begin{array}{l} i = (I-1) \mod n + 1 \\ j = \lfloor ((I-1) \bmod n^2)/n \rfloor + 1 \\ k = \lfloor (I-1)/n^2 \rfloor + 1 \end{array} \right. . \quad (2)$$

(ii) Identification des points du bord de la grille. Avec le *triple* indiçage (i,j,k), un point est sur le bord si, et seulement si, $i \in \{1,n\}$ ou $j \in \{1,n\}$ ou $k \in \{1,n\}$.

2. Construction *rapide* de la matrice $\mathbb{A}_3$.
Rapide = algorithme en $O(N)$ opérations.

On peut par exemple, dans le fichier `Laplacien3d.m`, créer la fonction

```
function A3 = Laplacien3d(n)
N = n^3;
A3 = 6*speye(N,N);
for k = 1:n
  for j = 1:n
    for i = 1:n
      I = i+(j-1)*n+(k-1)*n*n;
      % I n'est pas un point du bord gauche donc il a un voisin de gauche
      if i~=1
        A3(I,I-1) = -1;
      end
      % I n'est pas un point du bord droit donc il a un voisin de droite
      if i~=n
        A3(I,I+1) = -1;
      end
      % I n'est pas un point du bord bas donc il a un voisin de dessous
      if j~=1
        A3(I,I-n) = -1;
      end
```

```
      % I n'est pas un point du bord haut donc il a un voisin de dessus
      if j~=n
        A3(I,I+n) = -1;
      end
      % I n'est pas un point du bord avant donc il a un voisin de devant
      if k~=1
        A3(I,I-n*n) = -1;
      end
      % I n'est pas un point du bord arriere donc il a un voisin de derriere
      if k~=n
        A3(I,I+n*n) = -1;
      end
    end
  end
end
% On multiplie par 1/h^2 = (n+1)^2
A3 = (n+1)^2*A3;
```

ou encore

```
function A3 = Laplacien3d(n)
N = n^3;

% (i) Comme la matrice A3 est symetrique, on se concentre sur
% la partie triangulaire superieure, ie. A3(I,J) pour J > I...
% (ii) Pas besoin d'initialisation par A3 = sparse(N,N), car celle-ci
% est contenue dans la premiere affectation.

% 1. Les voisins de droite : A3(I,I+1) = -1 pour tout I < N,
tmp = [[sparse(N-1,1) (-1)*speye(N-1)]; sparse(1,N)];
% Sauf s'il n'y a pas de voisin de droite...
% On effectue donc la correction
for k = 1:n
  for j = 1:n
    I = (k-1)*n^2+j*n;
    if (I+1 <= N)
      tmp(I,I+1) = 0;
    end
  end
end
A3 = tmp;

% 2. Les voisins du haut : A3(I,I+n) = -1 pour tout I < N-n+1,
tmp = [[sparse(N-n,n) (-1)*speye(N-n)]; sparse(n,N)];
% Sauf s'il n'y a pas de voisin du haut...
% On effectue donc la correction
for k = 1:n
  for i = 1:n
    I = (k-1)*n^2+(n-1)*n+i;
    if (I+n <= N)
      tmp(I,I+n) = 0;
```

```
    end
  end
end
A3 = A3 + tmp;

% 3. Les voisins arriere : A3(I,I+n^2) = -1 pour tout I < N-n^2+1,
tmp = [[sparse(N-(n^2),n^2) (-1)*speye(N-(n^2))]; sparse(n^2,N)];
A3 = A3 + tmp;

% La partie triangulaire superieure est construite !

% 4. On symetrise la matrice : A3(I,J) = A3(J,I) pour tout I different de J,
A3 = A3 + A3';

% 5. On met a jour la diagonale : A3(I,I) = 6 pour tout I,
A3 = A3 + 6*speye(N);

% 6. On multiplie par 1/h^2 = (n+1)^2
A3 = (n+1)^2*A3;
```

Mise en œuvre informatique à l'aide de Matlab

1. *Points à retenir* :
   - Notions et commandes Matlab du TP1.
   - *Evaluation des performances* d'un algorithme et de sa mise en œuvre.
   - *Différence* entre stockage *plein* et stockage *creux*.
2. *Fonctionnalités Matlab à maîtriser* :
   - Dimensions d'une matrice : `size(A)`.
   - Initialisation (structures *creuses*) pour les matrices : `A = sparse(N,N)`.
   - Affichage de la structure d'une matrice : `spy(A)`.
   - Matrice identité I_N : `A = eye(N)` (structure *pleine*), ou `A = speye(N)` (structure *creuse*).
   - Mesure du temps d'exécution d'une suite d'instructions : `tic ; ... ;toc` ou `cputime`.

MA261. Introduction au calcul scientifique
Corrigé 3 : Résolution par la méthode de Cholesky des problèmes discrétisés

Le but est de résoudre les problèmes discrétisés 1d (fil, poutre), 2d (membrane), 3d (cavité), que l'on écrit sous la forme $\mathbb{A}_d\vec{u} = \vec{f}$, $d = 1, 2, 3$. Les matrices $(\mathbb{A}_d)_{d=1,2,3}$ étant *symétriques définie-positives*, on utilisera la méthode de Cholesky pour la résolution des systèmes linéaires.

Soit A une matrice de $\mathbb{R}^{N\times N}$ *symétrique définie-positive*, et $\vec{f}$ un élément de $\mathbb{R}^N$. La factorisation de Cholesky de la matrice A construit une matrice triangulaire inférieure L telle que $A = LL^\mathsf{T}$. En remplaçant A par sa factorisation, on peut résoudre le problème

$$A\vec{u} = \vec{f} \tag{3}$$

en deux temps :
- une *descente* : résoudre $L\vec{y} = \vec{f}$;
- une *remontée* : résoudre $L^\mathsf{T}\vec{u} = \vec{y}$.

Ceci est très avantageux, car chacun des deux problèmes fait intervenir une matrice triangulaire, pour laquelle la résolution du système linéaire associé est élémentaire...

Exercice 8 Méthode de Cholesky (I).
Ecrire une fonction Matlab pour calculer la solution de (3) à l'aide la méthode de Cholesky.
Aide : utiliser la fonction Matlab qui réalise la factorisation de Cholesky.

Corrigé 8 Dans le fichier `SolChol.m`, créér la fonction `SolChol` :

```
function u = SolChol(A,f)
% On suppose que la matrice A est SDP

% 1. Calcul de L triangulaire inferieure telle que A = L.L'
LT = chol(A);

% 2. Resolution de L.L'u = f, par une descente-remontee
% 2.1 Descente : resolution de Ly = f
y = LT'\f;
% 2.2 Remontee : resolution de L'u = y
u = LT\y;
```

Exercice 9 Résolution des problèmes discrétisés et étude de l'erreur.
Les problèmes statiques 1d (fil, poutre), 2d (membrane), 3d (cavité) sont respectivement posés dans $\Omega_d =]0,1[^d$, pour $d = 1, 2, 3$: trouver u tel que

$$-\Delta_d u = f \text{ sur } \Omega_d, \quad u = 0 \text{ sur } \partial\Omega.$$

Ils sont discrétisés à l'aide de la méthode des différences finies dans $\mathbb{R}^N$ ($N = n^d$), pour aboutir à

$$\mathbb{A}_d\vec{u} = \vec{f} \text{ pour } \ d = 1, 2, 3.$$

1. Appliquer la méthode de Cholesky aux problèmes discrétisés.

2. Comment peut-on vérifier que la solution calculée est une bonne approximation de la solution exacte ?

Corrigé 9 1. Dans le fichier `Donnee1d.m`, créer la fonction :

```
function f = Donnee1d(n)
% Second membre aleatoire
f = rand(n,1);
```

Ensuite, pour n donné, on tape en ligne :

```
A1 = Laplaciend1d(n);
f1 = Donnee1d(n);
u1 = SolChol(A1,f1);
```

On procède de même en 2d et 3d...

2. Il faut comparer la solution calculée $\vec{u} = (u_i)_{1\leq i\leq N}$ aux valeurs prises par la solution u aux points de discrétisation, c'est-à-dire $(x_i)_{1\leq i\leq n}$ en 1d, $(x_i, y_j)_{1\leq i,j\leq n}$ en 2d, $(x_i, y_j, z_k)_{1\leq i,j,k\leq n}$ en 3d. D'après l'étude théorique de l'erreur, on sait que ces valeurs sont de plus en plus proches, lorsque n croît, ou ce qui est équivalent, lorsque le pas de discrétisation $h = 1/(n+1)$ decroît.

Pour cela, il faut connaître la solution exacte... Une façon simple de procéder est d'en choisir une (!!), puis de construire le second membre. Il convient de faire attention à ce que la solution retenue respecte bien la condition aux limites sur la frontière. Par exemple, on peut prendre (avec l, m, n entiers non nuls) :

(1d) $u_l(x) = \sin(l\pi x)$, ou $u(x) = x(1-x)v(x)$, avec v quelconque, etc. ;

(2d) $u_{l,m}(x, y) = \sin(l\pi x)\sin(m\pi y)$, ou $u(x) = x(1-x)y(1-y)v(x, y)$, etc. ;

(3d) $u_{l,m,n}(x, y, z) = \sin(l\pi x)\sin(m\pi y)\sin(n\pi z)$, etc. ;

Pour ce qui concerne la comparaison entre solutions exacte et approchée, on peut procéder comme ci-dessous.

- *En visualisant* les solutions à l'aide de `plot` ou `plot3`.
- *En construisant* le vecteur erreur $\vec{e} \in \mathbb{R}^N$ défini par

$$e_I = u_I - u(x_i)\ (1d)\ ;\ e_I = u_I - u(x_i, y_j)\ (2d)\ ;\ e_I = u_I - u(x_i, y_j, z_k)\ (3d).$$

A partir de là on peut calculer la norme de l'erreur, par exemple $\|\vec{e}\|_\infty$, et étudier son évolution en fonction de n.

Exercice 10 Etude de complexité.

1. Comment évaluer la complexité
   - d'un algorithme ?
   - sous Matlab ?
2. Evaluer la complexité de la descente-remontée, puis de la factorisation de Cholesky pour $A \in \mathbb{R}^{N\times N}$. Evaluer la complexité totale de la résolution du système linéaire $A\vec{u} = \vec{f}$ à l'aide de la méthode de Cholesky.
3. Choisir succcessivement $A = \mathbb{A}_1, \mathbb{A}_2, \mathbb{A}_3$, en faisant varier N, et comparer la complexité de la factorisation au temps d'exécution de la commande `LT = chol(A)` sous Matlab. Conclusion ?

4. Même question pour la résolution du système linéaire et la commande `A\f` .

Corrigé 10 1. Définition de la complexité
- d'un algorithme : nombre total d'opérations (ici $+, -, *, /$, voire $\sqrt{}$) en fonction de la taille du problème (ici $N = n^d$).
- sous Matlab : temps calcul, mesuré par `tic ; ... ;toc` ou à l'aide de `cputime`.

2. Algorithmes et complexités associées.
   (a) Descente $L\vec{y} = \vec{f}$:

$$\left\| \begin{array}{l} \textbf{pour } i = 1, \ldots, N \textbf{ faire} \\ \quad y_i = [f_i - \sum_{j=1}^{i-1} L_{i,j} y_j]/L_{i,i}. \\ \textbf{fin} \end{array} \right.$$

   L'algorithme consiste en une boucle unique sur i. Le calcul de la composante y_i requiert :
   - pour $i = 1$: 1 division ;
   - pour $i > 1$: 1 soustraction, $(i-1)$ multiplications, $(i-2)$ additions, 1 division.

   Soit un total de $(2i-1)$ opérations élémentaires $(+, -, *, /)$. La complexité de la descente est donc égale à

$$\sum_{i=1}^{i=N} (2i-1) = 2\frac{N(N+1)}{2} - N = N^2 \text{ opérations.}$$

   (b) Remontée $U\vec{u} = \vec{y}$, avec $U = L^{\mathsf{T}}$:

$$\left\| \begin{array}{l} \textbf{pour } i = N, \ldots, 1 \textbf{ (pas } -1\textbf{) faire} \\ \quad u_i = [y_i - \sum_{j=i+1}^{N} U_{i,j} u_j]/U_{i,i}. \\ \textbf{fin} \end{array} \right.$$

   De même que pour la descente, la complexité de la remontée est égale à N^2 opérations.
   (c) Factorisation de Choleski :

$$\left\| \begin{array}{l} \textbf{pour } i = 1, \ldots N \textbf{ faire} \\ \quad \textbf{pour } j = 1, \ldots i-1 \textbf{ faire} \\ \quad\quad L_{i,j} = [A_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k}]/L_{j,j} \\ \quad \textbf{fin} \\ \quad\quad L_{i,i} = [A_{i,i} - \sum_{k=1}^{i-1} L_{i,k}^2]^{1/2}. \\ \textbf{fin} \end{array} \right.$$

L'algorithme consiste en une boucle imbriquée sur i et j. L'indice i varie de 1 à N. Pour une valeur de i fixée, on effectue tout d'abord une boucle sur j de 1 à $i-1$, plus le calcul pour $j=i$, avec pour i et j fixés $2j-1$ opérations. Soit au total i^2 opérations pour chaque i ; en effet,

$$\sum_{j=1}^{j=i}(2j-1) = 2\frac{i(i+1)}{2} - i = i^2 \text{ opérations.}$$

Si on considère maintenant la boucle sur i, on arrive à la complexité globale :

$$\sum_{i=1}^{i=N} i^2 = \frac{N(N+1)(2N+1)}{6} = \frac{1}{3}N^3 + \frac{1}{2}N^2 + \frac{1}{6}N \text{ opérations.}$$

(d) Complexité totale : on somme, pour trouver le terme dominant de

$$\frac{1}{3}N^3 \text{ opérations.}$$

3. Pour $A = \mathbb{A}_1, \mathbb{A}_2, \mathbb{A}_3$, on fait varier n (et donc N), et on étudie le temps calcul en échelle $\log - \log$ (commande Matlab `loglog`). On ne retrouve pas un comportement en $3\log N = 3d\log n$: Matlab est toujours meilleur, et utilise donc un autre algorithme... A titre indicatif, le log du temps calcul sous Matlab se comporte selon
   - pour $d=1$: $\approx 1.0\log N$;
   - pour $d=2$: $\approx 1.4\log N$;
   - pour $d=3$: $\approx 2.0\log N$.
4. *Idem.*

Exercice 11 Méthode de Cholesky (II).

1. Rappeler ce qu'est le profil (optimisé) d'une matrice symétrique définie-positive.
2. Ecrire les algorithmes de descente-remontée et de la factorisation de Cholesky, avec prise en compte du *profil*.
3. Evaluer la complexité de la résolution du système linéaire $\mathbb{A}_d\vec{u} = \vec{f}$ à l'aide de la méthode de Cholesky avec prise en compte du profil, pour les matrices $(\mathbb{A}_d)_{d=1,2,3}$. Conclusion ?

Corrigé 11 1. Le profil d'une matrice inversible $A \in \mathbb{R}^{N\times N}$ est déterminé par la donnée d'un vecteur $p^A \in \mathbb{R}^N$ tel que, pour $1 \le i \le N$:

$$A_{i,(p^A)_i} \neq 0 \text{ et } A_{i,j} = 0 \text{ pour } 1 \le j \le (p^A)_i - 1.$$

Pour une matrice définie-positive, on remarque que $A_{i,i} = (A\vec{e}_i, \vec{e}_i) > 0$, et ainsi $(p^A)_i \le i$ pour tout i. Et, si on définit le *profil optimisé* d'une matrice SDP comme étant :

$$P^A_{opt} = \left\{(i,j) \in \{1,\dots,N\}^2 \ : \ (p^A)_i \le j \le i\right\},$$

son intérêt principal est d'être conservé par la factorisation de Cholesky ! En d'autres termes, on a la propriété : $P^L_{opt} = P^A_{opt}$.

2. Algorithmes avec prise en compte du profil (noté p) :

(a) Descente $L\vec{y} = \vec{f}$:

pour $i = 1, \ldots, N$ **faire**

$$y_i = [f_i - \sum_{j=p_i}^{i-1} L_{i,j} y_j]/L_{i,i}.$$

fin

(b) Remontée $L^{\mathsf{T}}\vec{u} = \vec{y}$:

pour $i = N, \ldots, 1$ **(pas** -1**) faire**

$$u_i = [y_i - \sum_{j=i+1}^{q_i} L_{j,i} u_j]/L_{i,i}.$$

fin

Ci-dessus, le profil additionnel $q \in \mathbb{R}^N$ est défini, pour $1 \leq i \leq N$, par :

$$L_{q_i,i} \neq 0 \text{ et } L_{j,i} = 0 \text{ pour } q_i + 1 \leq j \leq N.$$

(c) Factorisation de Choleski :

pour $i = 1, \ldots N$ **faire**

pour $j = p_i, \ldots i-1$ **faire**

$$L_{i,j} = [A_{i,j} - \sum_{k=\max(p_i,p_j)}^{j-1} L_{i,k} L_{j,k}]/L_{j,j}$$

fin

$$L_{i,i} = [A_{i,i} - \sum_{k=p_i}^{i-1} L_{i,k}^2]^{1/2}.$$

fin

3. Complexité avec stockage profil.

(a) Bornes *a priori*.
Notons que, pour $d = 1, 2, 3$, on a respectivement, pour tout i variant de 1 à N,
– pour $d = 1$: $i - p_i \leq 1$;
– pour $d = 2$: $i - p_i \leq n = N^{1/2}$;
– pour $d = 3$: $i - p_i \leq n^2 = N^{2/3}$.
De façon plus compacte, si on pose $\bar{p}_d = n^{d-1} = N^{\frac{d-1}{d}}$, on peut écrire, pour $d = 1, 2, 3$ et pour tout i variant de 1 à N,

$$(i - p_i) \leq \bar{p}_d.$$

NB. Pour le profil additionnel (étape de remontée), on a également $(q_i - i) \leq \bar{p}_d$.

(b) Descente-remontée.
L'algorithme de descente consiste encore en une boucle unique sur i. Le calcul de la composante y_i requiert cette fois :

- pour $i = 1$: 1 division ;
- pour $i > 1$: 1 soustraction, $(i - p_i)$ multiplications, $(i - p_i - 1)$ additions, 1 division.

Soit un total de $2(i - p_i) + 1$ opérations élémentaires $(+, -, *, /)$. La complexité de la descente est donc inférieure à

$$\sum_{i=1}^{i=N} (2\bar{p}_d + 1) \leq (2\bar{p}_d + 1)N \text{ opérations.}$$

Comme le coût de la remontée est identique à celui de la descente ($(q_i - i) \leq \bar{p}_d$, pour tout i), on arrive à un coût global de descente-remontée de l'ordre de :
– pour $d = 1 : 6N$ opérations ;
– pour $d = 2 : 4N^{3/2}$ opérations ;
– pour $d = 3 : 4N^{5/3}$ opérations.

(c) Factorisation de Choleski.
Avec un stockage profil, on exécute encore une boucle imbriquée sur i et j. L'indice i varie toujours de 1 à N, mais, l'indice j varie seulement de p_i à i. De plus, pour i et j fixés, on effectue $2(j - \max(p_i, p_j)) + 1$ opérations, ce qui est toujours inférieur à

$$2(i - p_i) + 1 \text{ opérations.}$$

Ainsi, le nombre total d'opérations est inférieur à :

$$\sum_{i=1}^{i=N} \sum_{j=p_i}^{j=i} (2\bar{p}_d + 1) \leq \sum_{i=1}^{i=N} (2\bar{p}_d + 1)(1 + \bar{p}_d) = (2\bar{p}_d + 1)(1 + \bar{p}_d)N \text{ opérations.}$$

NB. Pour $d = 1$, on peut préciser l'estimation en distinguant pour i fixé le calcul des coefficients diagonaux de celui des coefficients extra-diagonaux, au nombre de *un* pour chaque catégorie : $L_{i,i}$ et $L_{i,i-1}$ (sauf pour $i = 1$). On se rend facilement compte qu'il faut bien *trois* opérations pour les premiers, mais seulement *une* pour les seconds, soit au total $(3 + 1)N = 4N$ opérations.
En conclusion, pour réaliser la factorisation de Cholesky avec un stockage profil, on arrive à un coût global de l'ordre de
– pour $d = 1 : 4N$ opérations ;
– pour $d = 2 : 2N^2$ opérations ;
– pour $d = 3 : 2N^{7/3}$ opérations.

(d) Résolution de $\mathbb{A}_d u = f$: la partie prépondérante de la résolution étant la factorisation (sauf en 1d), on retrouve
– pour $d = 1 : 10N$ opérations ;
– pour $d = 2 : 2N^2$ opérations ;
– pour $d = 3 : 2N^{7/3}$ opérations.

Sous Matlab, on a bien un comportement linéaire en 1d... En 2d et 3d, les comportements restent meilleurs que respectivement $2.0 \log N$ et que $2.3 \log N$.

Mise en œuvre informatique à l'aide de Matlab

1. *Points à retenir* :
   - Notions et commandes Matlab des TP1 et TP2.
   - Avantage du profil pour calculer la factorisation de Cholesky d'une matrice symétrique définie-positive.
   - Efficacité de la résolution d'une système linéaire à l'aide de la factorisation de Cholesky.
2. *Fonctionnalités Matlab à maîtriser* :
   - Commande de résolution d'un système linéaire `x=A\f`.
   - Commandes affichant le temps calcul `tic ;... ;toc`, `cputime`.
   - Commande `find` pour déterminer rapidement les indices des éléments non nuls d'un tableau.
   - Commande `loglog` pour affichage en échelle $\log - \log$.

MA261. Introduction au calcul scientifique
Corrigé 4 : Discrétisation et résolution de la corde vibrante 1d

`Nous considérons l'évolution d'une corde vibrante de longueur 4,`
`de l'instant initial zéro à l'instant final` $T = 2$.

Le problème instationnaire 1d à résoudre est (pour $c = 10$) : *trouver u tel que*

$$\frac{\partial^2 u}{\partial t^2}(x,t) - c^2 \frac{\partial^2 u}{\partial x^2}(x,t) = 0 \text{ pour } (x,t) \in]0,4[\times]0,2[, \tag{4}$$

$$u(x,0) = 2\sin(\frac{3\pi x}{2})\cos(\frac{\pi x}{2}) \text{ pour } x \in [0,4], \tag{5}$$

$$\frac{\partial u}{\partial t}(x,0) = 0 \text{ pour } x \in [0,4], \tag{6}$$

$$u(0,t) = u(4,t) = 0 \text{ pour } t \in [0,2]. \tag{7}$$

Exercice 12 Discrétisation par différences finies.
On note h_x le *pas de discrétisation en espace*, et h_t le *pas de discrétisation en temps* :

$$h_x = \frac{4}{n_x+1}, \quad h_t = \frac{2}{n_t+1}.$$

On note de plus $(x_j, t_m)_{j,m}$ les *points de discrétisation* dans $[0,4]\times[0,2]$, et $(u_j^m)_{j,m}$ les *valeurs approchées* de $u(x_j,t_m)_{j,m}$.

1. Que valent (x_j, t_m) ? (Sans oublier de préciser les bornes de variations de j et m.)
2. Comment prendre en compte les conditions aux limites ?
   Et les conditions initiales (5) et (6) ?
3. Construire un schéma de discrétisation de $\dfrac{\partial^2 u}{\partial x^2}(x_j,t_m)$.
4. Construire un schéma de discrétisation de $\dfrac{\partial^2 u}{\partial t^2}(x_j,t_m)$.
5. En regroupant les informations précédentes, vérifier que l'on aboutit au *schéma numérique* ci-dessous :

$$u_0^m = u_{n_x+1}^m = 0, \text{ pour } 0 \le m \le n_t+1 \ ; \tag{8}$$

$$u_j^1 = u_j^0 = 2\sin(\frac{3j\pi h_x}{2})\cos(\frac{j\pi h_x}{2}), \text{ pour } 0 \le j \le n_x+1 \ ; \tag{9}$$

$$u_j^{m+1} = \frac{c^2 h_t^2}{h_x^2}(u_{j+1}^m - 2u_j^m + u_{j-1}^m) + 2u_j^m - u_j^{m-1}, \text{ pour } 1 \le j \le n_x,\ 1 \le m \le n_t. \tag{10}$$

Corrigé 12 1. $(x_j,t_m) = (jh_x, mh_t)$, $0 \le j \le n_x+1$, $0 \le m \le n_t+1$.

2. Conditions aux limites : $u_0^m = u_{n_x+1}^m = 0$, $0 \le m \le n_t+1$.
   Condition initiale (5) : $u_j^0 = 2\sin(\dfrac{3j\pi h_x}{2})\cos(\dfrac{j\pi h_x}{2})$, $0 \le j \le n_x+1$.
   Condition initiale (6) : $\dfrac{u_j^1 - u_j^0}{h_t} = 0$, $0 \le j \le n_x+1$.

3. Schéma à trois points selon x :

$$\frac{\partial^2 u}{\partial x^2}(x_j, t_m) \approx \frac{u_{j+1}^m - 2u_j^m + u_{j-1}^m}{h_x^2},\ 1 \leq j \leq n_x,\ 1 \leq m \leq n_t.$$

4. Schéma à trois points selon t :

$$\frac{\partial^2 u}{\partial t^2}(x_j, t_m) \approx \frac{u_j^{m+1} - 2u_j^m + u_j^{m-1}}{h_t^2},\ 1 \leq j \leq n_x,\ 1 \leq m \leq n_t.$$

5. Facile !

Exercice 13 Stockage et calcul de $(u_j^m)_{0\leq j\leq n_x+1, 0\leq m\leq n_t+1}$.
On considère dans cette partie la mise en œuvre informatique (en `Matlab`) du schéma numérique (8-10).

1. A un instant t_m, pourquoi peut-on se contenter de calculer le vecteur $\vec{v}^m \in \mathbb{R}^{n_x}$ de composantes $(u_j^m)_{1\leq j\leq n_x}$?
2. Vérifier que pour $1 \leq m \leq n_t$, $\vec{v}^{m+1}$ est solution de

$$\vec{v}^{m+1} = (2\mathbb{I}_{n_x} + \alpha_0 \mathbb{A}_1')\vec{v}^m - \vec{v}^{m-1},\ \text{avec } \mathbb{A}_1' = \frac{1}{(h_x)^2}\begin{pmatrix} 2 & -1 & \dots & \dots & 0 \\ -1 & 2 & -1 & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & -1 \\ 0 & \dots & \dots & -1 & 2 \end{pmatrix} \in \mathbb{R}^{n_x \times n_x}.$$

   Déterminer α_0.
3. Ecrire une fonction `Matlab` qui retourne $(u_j^{m+1})_{1\leq j\leq n_x}$, avec comme arguments d'entrée n_x, n_t et m.
4. On fixe $h_x = 1/40$ ($n_x = 159$), et on laisse h_t varier :
   (a) Quelle est la valeur de $(u_{20}^{n_t+1})$ pour

$$h_t = \frac{2}{10000}, \frac{2}{4000}, \frac{2}{2000}, \frac{2}{800}, \frac{2}{799}\ ?$$

   (b) En visualisant la solution au temps final $T = 2$ pour ces différents pas de temps, qu'en conclure au sujet de la stabilité numérique de la méthode ?

Corrigé 13 1. *A priori*, on doit déterminer $(u_j^m)_{0\leq j\leq n_x+1}$ pour m donné, or on sait déjà que $u_0^m = u_{n_x+1}^m = 0$.

2. On fixe m dans (10), et on laisse varier j dans $\{1, \cdots, n_x\}$: on arrive à

$$\vec{v}^{m+1} = -c^2 h_t^2 \mathbb{A}_1' \vec{v}^m + 2\vec{v}^m - \vec{v}^{m-1},$$

   et en particulier $\alpha_0 = -(ch_t)^2$.

3. Dans le fichier `corde_exp.m`, créer la fonction :

```
function u = corde_exp(nx,nt,c,m)
% u = corde_exp(nx,nt,c,m)
% ----------------------
% x varie de 0 a 4
% t varie de 0 a 2
```

```
% Entrees :
% (nx+2) nombre de points de discretisation en espace, de 0 a nx+1
% (nt+2) nombre de points de discretisation en temps, de 0 a nt+1
% c vitesse de propagation
% (m+1)ht dernier instant de la simulation
%
% Schema :
% Explicite
%
% Sortie :
% valeurs de la solution approchee a l'instant (m+1)ht : u(1:nx,m+1)
% (u(0,(m+1)ht) et u(4,(m+1)ht) sont connues exactement)%

% Pas de discretisation
hx = 4/(nx+1);
ht = 2/(nt+1);
coeff = -(c*ht)^2;
% Conditions initiales
v0 = zeros(nx,1);
v1 = zeros(nx,1);
for j = 1:nx
  v0(j) = 2*sin(3*j*pi*hx/2)*cos(j*pi*hx/2);
  v1(j) = v0(j);
end
% Schema explicite
% ATTENTION : A1 est la matrice du Laplacien sur ]0,4[ !!
% D'ou A1 = 1/(4^2)*Laplacien1d(nx) !
A1 = 1/(4^2)*Laplacien1d(nx);
B1 = (2*speye(nx)+coeff*A1);
for inc = 1:m
   u = B1*v1 - v0;
  v0 = v1;
  v1 = u;
end
```

4. Résultats et interprétation
   (a) Valeurs de $(u_{20}^{n_t+1})$:

$$h_t = \ 0.9998,\ 0.9998,\ 0.9997,\ 1.0000,\ -2.1912\,10^{18}.$$

   (b) La discrétisation semble fonctionner si, et seulement si,

$$h_t \leq \frac{2}{800} = \frac{1}{400} = \frac{h_x}{c} \Longleftrightarrow ch_t \leq h_x.$$

Exercice 14 Stabilité numérique.
Pour résoudre la difficulté mise en évidence à la question 4 de l'exercice précédent, on propose un second schéma numérique, en remplaçant le terme $(u_{j+1}^m - 2u_j^m + u_{j-1}^m)$ dans (10) par la moyenne

$$\frac{1}{2}(u_{j+1}^{m+1} - 2u_j^{m+1} + u_{j-1}^{m+1}) + \frac{1}{2}(u_{j+1}^{m-1} - 2u_j^{m-1} + u_{j-1}^{m-1}).$$

1. Pourquoi parle-t-on de moyenne ?
2. Ecrire ce second schéma, en conservant la notation $(u_j^m)_{j,m}$ pour la solution approchée. Si on utilise à nouveau $\vec{v}^m \in \mathbb{R}^{n_x}$, vérifier que pour $1 \leq m \leq n_t$, $\vec{v}^{m+1}$ est cette fois solution du système linéaire

$$(\alpha_1 \mathbb{A}_1' + \mathbb{I}_{n_x})\vec{v}^{m+1} = 2\vec{v}^m - (\alpha_1 \mathbb{A}_1' + \mathbb{I}_{n_x})\vec{v}^{m-1}.$$

   Déterminer α_1. Qu'en déduire sur la nature de la matrice $\alpha_1 \mathbb{A}_1' + \mathbb{I}_{n_x}$?
3. Ecrire une fonction `Matlab` qui retourne $(u_j^{m+1})_{1 \leq j \leq n_x}$, avec comme arguments d'entrée n_x, n_t et m.
4. On fixe $h_x = 1/40$ ($n_x = 159$), et on laisse à nouveau h_t varier... En visualisant la solution au temps final $T = 2$ pour différents pas de temps, qu'en conclure au sujet de la stabilité numérique du second schéma ?
5. Comparer rapidement les avantages et inconvénients des deux schémas.

Corrigé 14 1. On approche $\dfrac{\partial^2 u}{\partial x^2}(x_j, t_m)$ selon la moyenne des schémas à trois points en (x_j, t_{m+1}) et (x_j, t_{m-1}).

2. On trouve facilement $\alpha_1 = (ch_t)^2/2 = -\alpha_0/2$.
   La matrice $(\alpha_1 \mathbb{A}_1' + \mathbb{I}_{n_x})$ est donc SDP.
3. Dans le fichier `corde_imp.m`, créer la fonction :

```
 function u = corde_imp(nx,nt,c,theta,m)
% u = corde_imp(nx,nt,c,theta,m)
% -----------------------------
% Entrees :
% (nx+2) nombre de points de discretisation en espace, de 0 a nx+1
% (nt+2) nombre de points de discretisation en temps, de 0 a nt+1
% c vitesse de propagation
% theta degre d'implicitation (theta dans [0,.5])
% (m+1)ht dernier instant de la simulation
%
% Schema :
% Implicite (theta-schema)
%
% Sortie :
% valeurs de la solution approchee a l'instant (m+1)ht : u(1:nx,m+1)
% (u(0,(m+1)ht) et u(4,(m+1)ht) sont connues exactement)
%

% Pas de discretisation
% x varie de 0 a 4
% t varie de 0 a 2
hx = 4/(nx+1);
ht = 2/(nt+1);
coeff = (c*ht)^2;
% Conditions initiales
v0 = zeros(nx,1);
v1 = zeros(nx,1);
```

```
for j = 1:nx
  v0(j) = 2*sin(3*j*pi*hx/2)*cos(j*pi*hx/2);
  v1(j) = v0(j);
end
% Schema implicite
A1 = 1/(4^2)*Laplacien1d(nx);
B1 = theta*coeff*A1 + speye(nx);
B2 = 2*speye(nx)-(1-2*theta)*coeff*A1;
C1 = inv(B1);
for inc = 1:m
   u = C1*(B2*v1 - B1*v0);
  v0 = v1;
  v1 = u;
end
```

4. On a une *stabilité inconditionnelle* : les variations de h_t ne sont plus limitées. Par contre, on constate (pour h_t croissant) un amortissement et/ou un déphasage de la solution numérique, lorsque l'on superpose les solutions à l'instant final $T = 2$.
5. Premier schéma (*explicite*) :
   plus rapide, mais pas toujours stable (condition à respecter : $ch_t \leq h_x$).
   Second schéma (*implicite*) :
   moins rapide, mais toujours stable.

Mise en œuvre informatique à l'aide de Matlab

Points à retenir :

- Notions et commandes Matlab des TP1, TP2 et TP3.