Travaux pratiques - Examen du 6 janvier 2020 (3h00)

Sans documents, sans calculatrice, sans portable, ...
Le barême est donné à titre indicatif

Voici trois fichiers *header*, `Vector.h`, `Matrix.h` et `SLAC.h`, que vous pourrez utiliser lors de cet examen. **Toute autre fonction utilisée devra être écrite**. On supposera, sauf mention contraire, que les fichiers sources associés `Vector.c`, `Matrix.c` et `SLAC.c` sont donnés. Les types Vector et Matrix sont imposés et spécifiés respectivement dans les fichiers `Vector.h` et `Matrix.h`.

Listing 1 – Vector.h (partiel)

```c
typedef struct {
 unsigned int dim;
 double *val;
 } Vector;

// AllocVector : Allocation d'un vecteur de dimension N
void AllocVector(unsigned int N, Vector *pU);
// FREEVector : Desalloue proprement un vecteur
void FREEVector(Vector *pV);
// Vector2cst : met les composantes d'un vecteur a c
 void Vector2cst(Vector V,double c);
// RandVector : Vecteur aleatoire, loi uniforme [a,b] (deja alloue)
void RandVector(Vector U, double a, double b);
// AllocRandVector : Vecteur aleatoire avec allocation, loi uniforme [a,b]
void AllocRandVector(int N, double a, double b,Vector *pW);
// PrintVector :  Affiche un vecteur
 void PrintVector(Vector U,char *VarName,unsigned int n);
// ScalarProd : Produit scalaire de 2 vecteurs
double ScalarProd(Vector U, Vector W);
// aUpbV : retourne le vecteur a*U+b*V
Vector aUpbV(double a,Vector U, double b, Vector V);
// ScaleVector : Transforme le vecteur U en a*U
void ScaleVector(double a,Vector U);
// NormVector : Calcul la norme p d'un vecteur
double NormVector(Vector U, unsigned int p);
// NormInfVector : Calcul la norme Inf d'un vecteur
double NormInfVector(Vector U);
// fevalVector : returns a Vector V such that V[i]=f(U[i])
Vector fevalVector(Vector U,double (*pf)(double));
// copyVector : Copie integrale d'un vecteur avec duplication du tableau.
Vector copyVector(Vector U);
```

Listing 2 – Matrix.h (partiel)

```c
typedef struct {
 unsigned int rows;
 unsigned int cols;
 double **val;
 } Matrix;

// AllocMatrix : Allocation contigue d'une matrice N lignes et M colonnes
//               avec initialisation a 0 de ses composantes
void AllocMatrix(unsigned int N, unsigned int M, Matrix *pA);
// FREEMatrix : Desalloue proprement une matrice
void FREEMatrix(Matrix *pMat);
// AllocRandMatrix : Matrice aleatoire NxM, loi uniforme [a,b]
Matrix AllocRandMatrix(unsigned int N, unsigned int M, double a, double b);
// AllocRandLowerMatrix : Matrice tri. inf. aleatoire, loi uniforme [a,b]
Matrix AllocRandLowerMatrix(unsigned int N, double a, double b);
// AllocRandLowerMatrix : Matrice tri. sup. aleatoire, loi uniforme [a,b]
Matrix AllocRandUpperMatrix(unsigned int N, double a, double b);
```

```c
// MatrixProd : retourne la Matrice A*B
Matrix MatrixProd(Matrix A, Matrix B);
// MatrixSum : retourne la Matrice  a*A+b*B
Matrix MatrixSum(double a, Matrix A,double b, Matrix B);
// PrintMatrix :  Affiche partiellement une matrice A
void PrintMatrix(Matrix A,char *VarName,unsigned int nr,unsigned int nc);
// IdentityMatrix : retourne la matrice identitee
Matrix IdentityMatrix(unsigned int N);
// Matrix2cst: Met la matrice a c (deja allouee)
void Matrix2cst(Matrix A, double c);
// ScaleMatrix : Transforme la matrice A en a*A
void ScaleMatrix(double a,Matrix A);
// TransposeMatrix : Retourne la matrice transposee
Matrix TransposeMatrix(Matrix A);
// AllocRandSPDMatrix : Retourne une matrice aleatoire symetrique definie positive
Matrix AllocRandSPDMatrix(unsigned int N, Matrix *pB);
// AllocRandTriUpperMatrix : Retourne une matrice aleatoire triangulaire superieure
//      suivant la loi uniforme sur (a,b)
Matrix AllocRandTriUpperMatrix(unsigned int N,double a, double b);
// AllocRandTriUpperMatrix : Retourne une matrice aleatoire triangulaire inferieure
//      suivant la loi uniforme sur (a,b)
Matrix AllocRandTriLowerMatrix(unsigned int N,double a, double b);
// Norm1Matrix : retourne ||A||_1 (norme 1 matricielle)
double Norm1Matrix(Matrix A);
//  NormInfMatrix : retourne ||A||_\infty (norme infini matricielle)
double NormInfMatrix(Matrix A);
// fevalMatrix : retourne la matrice B tel que B[i][j] <- f(A[i][j])
Matrix fevalMatrix(Matrix A,double (*pf)(double));
// copyMatrix : Copie integrale d'une matrice avec duplication des tableaux.
Matrix copyMatrix(Matrix A);
```

Listing 3 – SLAC.h (partiel)

```c
#include "Vector.h"
#include "Matrix.h"

// MatVecProd : Produit Matrice/Vecteur
Vector MatVecProd(Matrix A, Vector U);
// SolveLower : resolution systeme tri. inf.
Vector SolveLower(Matrix L, Vector b);
// SolveUpper : resolution systeme tri. sup.
Vector SolveUpper(Matrix U, Vector b);
// SolveGauss : resolution systeme par Gauss
Vector SolveGauss(Matrix A, Vector b);
// LU : factorisation LU de A
void LU(Matrix A, Matrix *pL, Matrix *pU);
// Resolution du systeme Ax=b ou L et U sont la fact. LU de A
Vector SolveLU(Matrix L, Matrix U, Vector b);
```

Exercice 1 (5 points)

On cherche à résoudre par un schéma de type différences finies le problème suivant

$$-u''(x) + c(x)u(x) = f(x), \ \forall x \in]a;b[\tag{1}$$
$$u'(a) - 2u(a) = w_a \in \mathbb{R} \tag{2}$$
$$u'(b) + 2u(b) = w_b \in \mathbb{R} \tag{3}$$

où $c : [a,b] \longrightarrow \mathbb{R}^+$, $f : [a,b] \longrightarrow \mathbb{R}$, w_a et w_b sont donnés.

Q. 1 *Ecrire, en justifiant, un schéma d'ordre 2 associé au problème (1)-(2)-(3).*

Q. 2 (C) *Ecrire un programme en langage C permettant de résoudre le problème précédent avec des données judicieusement choisies (pour avoir une solution exacte).*

Q. 3 (C) *Ecrire un programme en langage C permettant de retrouver l'ordre de la méthode (sans graphisme).*

EXERCICE 2 (4 points)

Dans cet exercice il vous est demandé d'écrire explicitement certaines fonctions dont les prototypes sont écrits dans les fichiers `Vector.h`, `Matrix.h` ou `SLAC.h`.

Q. 1 (`Matrix.h`) **(a)** *Ecrire la fonction* ***AllocMatrix*** *dont le prototype est :*

void AllocMatrix(unsigned int *N*, ***unsigned int*** *M*, ***Matrix*** **pA);*

Cette fonction devra allouer de manière contiguë un tableau de N×M éléments correpondant à une matrice de dimension N par M. Les éléments de ce tableau devront être mis à zéros.

(b) *Rappeler* ***précisement*** *la formule permettant de calculer le produit de deux matrices (avec ses hypothèses).*

(c) *Ecrire la fonction* ***MatrixProd*** *dont le prototype est :*

Matrix MatrixProd(Matrix *A*, ***Matrix*** *B);*

Cette fonction devra retourner, lorsque celà est possible, la matrice produit AB des deux matrices données en entrée.

Q. 2 (`SLAC.h`) **(a)** *Rappeler* ***précisement*** *la(es) formule(s) permettant de calculer le vecteur* $\boldsymbol{x}$ *solution du système triangulaire supérieur* $\mathbb{U}\boldsymbol{x} = \boldsymbol{b}$ *(avec ses hypothèses).*

(b) *Ecrire la fonction* ***SolveUpper*** *dont le prototype est :*

Vector SolveUpper(Matrix *U*, ***Vector*** *b);*

Cette fonction resoud un système linéaire triangulaire supérieur.

(c) *Ecrire un programme permettant de tester/valider cette fonction.*

EXERCICE 3 (9 points)

On souhaite résoudre numériquement l'E.D.P. suivante

$$\frac{\partial u}{\partial t}(t,x) - \kappa\frac{\partial^2 u}{\partial x^2}(t,x) + c(x)\frac{\partial u}{\partial x}(t,x) = f(t,x), \quad \forall (t,x) \in]t_0; t_0+T]\times]a;b[, \tag{1}$$

$$u(t_0,x) = u_0(x), \quad \forall x \in [a;b], \tag{2}$$

$$\frac{\partial u}{\partial x}(t,a) = v_a(t), \quad \forall t \in [t_0; t_0+T], \tag{3}$$

$$u(t,b) = u_b(t), \quad \forall t \in]t_0; t_0+T]. \tag{4}$$

avec $\kappa > 0$, $t_0 \in \mathbb{R}$, $T > 0$, $(a,b) \in \mathbb{R}^2$, $a < b$, $c : [a,b] \longrightarrow \mathbb{R}^+$.
On note t^n, $n \in [\![0, N_t]\!]$ et x_i, $i \in [\![0, N_x]\!]$ les discrétisations régulières des intervalles $[t_0; t_0+T]$ et $[a;b]$ avec N_t pas de discrétisation en temps et N_x pas de discrétisation en espace. On souhaite résoudre l'E.D.P. à l'aide des schémas numériques

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} - \kappa\frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{\Delta x^2} + c_i\frac{u_{i+1}^{n+1} - u_{i-1}^{n+1}}{2\Delta x} = f_i^{n+1}. \tag{5}$$

$$u_2^{n+1} - 4u_1^{n+1} + 3u_0^{n+1} = -2\Delta x v_a(t^{n+1}). \tag{6}$$

Q. 1 **(a)** *Expliquer précisement comment le schéma (5) (ordre 1 en temps et ordre 2 en espace) a été obtenu à partir de (1) et expliciter les valeurs* u_i^{n+1}, f_i^{n+1}, c_i, Δt *et* Δx.

(b) *Expliquer précisement comment le schéma (6) (ordre 2) a été obtenu à partir de (3).*

(c) *Donner une discrétisation (détaillée) du problème (1) à (4) en utilisant (entre autres) les schémas (5) et (6).*

On note $\boldsymbol{U}^n$ les vecteurs de dimension N_x+1, de composantes $\boldsymbol{U}_i^n = u_{i-1}^n$, $\forall i \in [\![1, N_x+1]\!]$.

Q. 2 **(a)** *Comment initialiser le vecteur* $\boldsymbol{U}^0$ *?*

(b) *En supposant le vecteur* $\boldsymbol{U}^n$ *déjà calculé, montrer que le vecteur* $\boldsymbol{U}^{n+1}$ *est solution du système linéaire*

$$\mathbb{A}\boldsymbol{U}^{n+1} = \boldsymbol{b}^n \tag{7}$$

en explicitant la matrice $\mathbb{A}$ *et le vecteur* $\boldsymbol{b}^n$ *(préciser les dimensions).*

Q. 3 (C) *Ecrire la fonction* **AssembleMat1D** *retournant la matrice* $\mathbb{M} \in \mathcal{M}_d(\mathbb{R})$ *définie par*

$$\mathbb{M} = \begin{pmatrix} \nu_1 & \nu_2 & \nu_3 & 0 & \dots & \dots & 0 \\ \beta_1 & \alpha_1 & \beta_1 & 0 & \dots & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & \beta_{d-2} & \alpha_{d-2} & \beta_{d-2} \\ 0 & \dots & \dots & 0 & \mu_1 & \mu_2 & \mu_3 \end{pmatrix} \tag{8}$$

où $\alpha \in \mathbb{R}^{d-2}$, $\beta \in \mathbb{R}^{d-2}$ $\mu \in \mathbb{R}^3$ *et* $\nu \in \mathbb{R}^3$ *sont donnés.*

Q. 4 (C) *On se donne* $a = -2$, $b = 2$, $T = 10$, $f(t, x) = x^2 \cos(t)$, $u_0(x) = 10$, $c(x) = 2 + \sin(x)$, $\kappa = 1$, $v_a(t) = \sin(t)$,

$$u_b(t) = \begin{cases} 10 + 25t, & si\ t <= 2, \\ 60, & si\ t \in [2; 8[, \\ 60 - 25(t-8), & si\ t \in [8; 10]. \end{cases}$$

Ecrire un programme complet en langage C permettant de résoudre le problème (1) *à* (4) *en utilisant les schémas* (5) *et* (6).

Exercice 4 (2 points)

Ecrire un programme permettant de remplir un tableau de **int** à double entrées (acces aux éléments avec `[i][j]`) avec des 0 et des 1. Le programme devra

- demander le nombre de lignes n et de colonnes du tableau m,
- remplir le tableau,
- afficher tableau.

Il vous est demandé d'user de fonctions avec pertinence. Un code sans fonction autre que la fonction `main` sera moins bien noté.
L'allocation sera dynamique et le(s) tableau(x) de **int** sera(ont) initialisé(s) à 0 (sans utiliser de boucle).

Voici les caractéristiques du tableau rempli :

- l'élément $[0][m-1]$ est égal à 1 (en haut à gauche dans l'affichage ci-dessous),
- le tableau ne contient que des 1 et des 0,
- les éléments *voisins* sont distincts. Les *voisins* de l'élément `[i][j]` sont, quand ils existent, `[i−1][j]`, `[i+1][j]`, `[i][j−1]` et `[i][j+1]`.

Deux exemples d'affichage sont donnés ci-dessous :

8 lignes et 10 colonnes	9 lignes et 15 colonnes
0 1 0 1 0 1 0 1 0 1	1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1 0 1	1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1 0 1	1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1 0 1	1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
	1 0 1 0 1 0 1 0 1 0 1 0 1 0 1