Instructions. *Les séances tutorées ont lieu les mardi et mercredi de 16h15 à 18h15 au SUD13 et SUD12. Veuillez vous rendre à la séance qui vous a été attribuée. Des étudiants moniteurs serront présents pour vous aider que ce soit au niveau théorique qu'au niveau de la programmation en langage C.*

Cette première séance de tutorat doit être vue comme une mise en jambes. Vous avez tous utilisé le langage C dans votre projet III. Cette séance a pour but de rappeler certains concepts importants pour la suite.

Le cours d'analyse numérique commence par l'étude de certains algorithmes liés aux matrices. Une matrice dense est une matrice ne contenant pas beaucoup de zéros. Conceptuellement, les matrices denses correspondent aux systèmes qui sont très couplés. Une matrice dense $A \in \mathbb{R}^{m\times n}$ sera donc représentée par une structure de données contenant $m \times n$ nombres réels.

Il existe plusieurs façon de conceptualiser le stockage des tableaux multidimensionnels dans une mémoire linéaire telle que la mémoire vive d'un ordinateur.

— Quand on utilise un stockage de type "row-major", les éléments consécutifs d'une ligne de la matrice se trouvent les uns à côté des autres.
— Quand on utilise un stockage de type "column-major", les éléments consécutifs d'une colonne de la matrice se trouvent les uns à côté des autres.

Bien que les termes fassent référence aux lignes et aux colonnes d'une matrice (i.e. un tableau à deux dimensions), ces concepts peuvent être généralisés aux tableaux de n'importe quelle dimension en notant que les termes "row-major" et "column-major" sont équivalents aux ordres lexicographiques et colexicographiques, respectivement.

Pour être plus concrets, considérons la matrice

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

Dans le cas "row major", les éléments de cette matrice seront stockés en mémoire de la façon suivante

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{21} & a_{22} & a_{23} \end{bmatrix}.$$

Dans le cas "col major", les éléments de cette matrice seront stockés en mémoire de la façon suivante

$$A = \begin{bmatrix} a_{11} & a_{21} & a_{12} & a_{22} & a_{13} & a_{23} \end{bmatrix}.$$

En langage C, les tableaux multidimensionnels sont stockés en "row-major". La Figure 1 présente un petit programme en C qui montre que la convention "row-major" est bien utilisée par le langage C.

Les matrices peuvent être représentées par un tableau de pointeurs vers chaque ligne ou vers chaque colonne de la matrice, en fonction de la convention choisie. Il est préférable néanmoins de *stocker les éléments de la matrice dense dans un espace contigu.* Utiliser un espace contigu est nécessaire pour une interface avec BLAS (Basic Linear Algebra Subroutines) que nous utiliserons par la suite. Il ne faut donc pas allouer la matrice ligne par ligne (ou colonne par colonne) mais allouer un grand tableau contigu de taille $n \times m$. La Figure 2

```
#include <math.h>
#include <stdio.h>

int main(void){
  double A[2][3] = {{1,2,3},{4,5,6}};
  double *a = (double*)A;
  printf("A[3] = %12.5E\n",a[3]);
  return 0;
}

>> gcc p1.c -o p1
>> p1
A[3] =  4.00000E+00
```

FIGURE 1 – Petit programme qui montre que le langage C utilise la convention "row-major".

donne un exemple de structure de données de matrice ainsi que deux fonctions pour allouer et libérer la mémoire. Dans la suite, on utilisera la convention "row-major".

La structure de données de la Figure 2 permet d'utilser la convention habituelle pour accéder aux éléments de la matrice. Durant cette première séance, nous vous demandons tout d'abord de travailler sur le produit de deux matrices. On vous demande de

1. Compiler et exécuter le programme de la Figure 2.
2. Écrire un algorithme qui permet de multiplier deux matrices : soit $A \in \mathbb{R}^{m\times p}$ et $B \in \mathbb{R}^{p\times n}$, on vous demande de calculer $C = AB \in \mathbb{R}^{m\times n}$ avec

$$C_{ij} = \sum_{k=1}^{p} A_{ik}B_{kj}.$$

   La signature de la fonction sera

   ```
   int mult_matrix (matrix *A, matrix *B, matrix *C);
   ```

   L'utilisateur de la fonction a la responsabilité d'allouer A, B et C. La fonction retourne 0 si tout s'est bien passé et retourne -1 si les dimensions de A, B et C sont incompatibles.
3. Calculez le nombre théorique d'opérations en virgules flottantes (FLOP) nécessaires pour effectuer ce produit matriciel. Une multiplication ou une addition comptent comme une opération.
4. Calculez (ou renseignez vous sur internet) le nombre maximum d'opérations en virgule flottantes (FLOP) que votre ordinateur est capable de réaliser par seconde (FLOPS). L'équation permettant de calculer un FLOPS est :

$$\text{FLOPS} = \text{cœurs} \times \text{fréquence} \times \frac{\text{FLOP}}{\text{cycle}}.$$

```
typedef struct {
  double **a;
  double *data;
  int m,n;
} matrix;

matrix* allocate_matrix (int n, int m){
  matrix *mat = (matrix*) malloc(sizeof(matrix));
  mat->m = m;
  mat->n = n;
  mat->a    = (double**)malloc(m*sizeof(double));
  if (mat->a == NULL)return NULL;
  mat->data = (double*)malloc(m*n*sizeof(double));
  if (mat->data == NULL)return NULL;
  for (int i=0;i<m;i++) mat->a[i]=mat->data+i*n;
  return mat;
}
void free_matrix (matrix *mat){
  if (mat == NULL)return;
  free (mat->a);
  free (mat->data);
}
int main(void){
  matrix *mat = allocate_matrix(100,200);
  if (mat == NULL) return -1;
  double **a = mat->a;
  for (int i=0;i<100;i++){
    for (int j=0;j<200;j++){
      a[i][j] = drand48();
    }
  }
  printf("A[12][116] = %12.5E\n", a[12][116]);
  free_matrix(mat);
  mat = NULL;
  return 0;
}
```

FIGURE 2 – Structure de donnée de base pour une matrice dense. Allocation avec convention row major et libération de la mémoire.

En 2022, la plupart des microprocesseurs sont capables de réaliser plusieurs opérations en virgule flottante par cycle (extensions SIMD, Single instruction, multiple data) qui permettent de chaîner (vectoriser) des différentes étapes d'un FLOP. Par exemple, la norme AVX2 (jeu d'instructions de l'architecture x86 d'Intel et AMD, proposé par Intel en mars 2008) permet théoriquement de faire 4 FLOP par cycle en double précision et 8 FLOP par cycle en simple précision.

5. Calculer le temps d'horloge (wall clock time) pour effectuer le produit de deux matrices remplies avec des nombres aléatoires (comme dans la Figure 2) avec $m = 1000$, $p = 2000$ et $n = 3000$ et comparer ce temps avec le temps théorique que votre ordinateur aurait du prendre pour effecture ce produit.
6. Le temps d'horloge que vous obtenez est largement supérieur au temps théorique. Vous n'utilisez donc qu'une toute petite partie des FLOPS disponibles. Comment expliquez vous cette différence ?
7. Il existe un standard international appelé BLAS (voir `https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms`). La fonction `dgemm` (`http://www.netlib.org/lapack/explore-html/d1/d54/group__double__blas__level3_gaeda3cbd99c8fb834a60a6.html`) permet d'effectuer un produit de matrices en double précision. Vérifier que votre ordinateur possède bien les librairies BLAS et effectuez le produit de matrices avec BLAS. Les performances sont-elles meilleures ?