

Corrigé des exercices du Cours 11

Tableaux à deux dimensions (ou plus)
Fonctions d'algèbre linéaire

Vincent Lepetit

`vincent.lepetit@epfl.ch`

Exercice

Écrire la fonction d'en-tête:

```
void triangle(char T[N][N])
```

qui initialise le tableau de caractères passé en paramètre à:

```
+++++++  
.+.....+  
..+.....+  
...+.....+  
....+.....+  
.....+..+  
.....+.+  
.....++  
.....+
```

triangle (1^{ère} solution)

```
void triangle(char T[N][N])  
{
```

T	0	1	2	...	N-1
0	?	?	?		?
1	?	?	?		?
2	?	?	?		?
...					?
N-1	?	?	?	?	?

triangle (1^{ère} solution)

```
void triangle(char T[N][N])  
{  
    for(int i = 0; i < N; i++)  
        for(int j = 0; j < N; j++)  
            T[i][j] = '.';
```

T	0	1	2	...	N-1
0
1
2
...					.
N-1

triangle (1^{ère} solution)

```
void triangle(char T[N][N])
{
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            T[i][j] = '.';

    for(int i = 0; i < N; i++)
        T[0][i] = '+';
}
```

T	0	1	2	...	N-1
0	+	+	+		+
1
2
...					.
N-1

triangle (1^{ère} solution)

```
void triangle(char T[N][N])
{
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            T[i][j] = '.';

    for(int i = 0; i < N; i++)
        T[0][i] = '+';

    for(int i = 0; i < N; i++)
        T[i][N-1] = '+';
}
```

T	0	1	2	...	N-1
0	+	+	+		+
1	.	.	.		+
2	.	.	.		+
...					+
N-1	+

triangle (1^{ère} solution)

```
void triangle(char T[N][N])
{
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            T[i][j] = '.';

    for(int i = 0; i < N; i++)
        T[0][i] = '+';

    for(int i = 0; i < N; i++)
        T[i][N-1] = '+';

    for(int i = 0; i < N; i++)
        T[i][i] = '+';
}
```

T	0	1	2	...	N-1
0	+	+	+		+
1	.	+	.		+
2	.	.	+		+
...					+
N-1	+

triangle (2^{ème} solution)

```
void triangle(char T[N][N])
{
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            T[i][j] = '.';
}
```

T	0	1	2	...	N-1
0
1
2
...					.
N-1

triangle (2^{ème} solution)

```
void triangle(char T[N][N])
{
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            if (i == 0)
                T[i][j] = '+';
            else
                T[i][j] = '.';
}
```

T	0	1	2	...	N-1
0	+	+	+		+
1
2
...					.
N-1

triangle (2^{ème} solution)

```
void triangle(char T[N][N])
{
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            if (i == 0 || j == N-1)
                T[i][j] = '+';
            else
                T[i][j] = '.';
}
```

T	0	1	2	...	N-1
0	+	+	+		+
1	.	.	.		+
2	.	.	.		+
...					+
N-1	+

triangle (2^{ème} solution)

```
void triangle(char T[N][N])
{
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            if (i == 0 || j == N-1 || i == j)
                T[i][j] = '+';
            else
                T[i][j] = '.';
}
```

T	0	1	2	...	N-1
0	+	+	+		+
1	.	+	.		+
2	.	.	+		+
...					+
N-1	+

triangle (2^{ème} solution)

```
void triangle(char T[N][N])
{
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            if (i == 0 || j == N-1 || i == j)
                T[i][j] = '+';
            else
                T[i][j] = '.';
}
```

Une seule instruction.
Pas besoin d'accolades.

Écrivez les fonctions:

- `set_0` qui initialise une matrice à la matrice nulle;
- `set_Id` qui initialise une matrice à la matrice Identité. La fonction `set_Id` retournera:
 - *false* si la matrice passée en paramètre n'est pas carrée, et
 - *true* si l'opération s'est bien passée;
- `init` permettant d'initialiser une matrice.

Il faut écrire une fonction différente pour chaque taille de matrice que vous voudrez initialiser:

```
void init(Matrice * M,  
          float c1, float c2, float c3, float c4)
```

permettra d'initialiser une matrice 2x2;

```
void init(Matrice * M,  
          float c1, float c2, float c3,  
          float c4, float c5, float c6)
```

permettra d'initialiser une matrice 2x3 ou 3x2.

Ces fonctions devront donc tester la valeur de `n1` et `nc` pour gérer les cas, et mettre les paramètres `c1, c2...` au bon endroit.

- `add` qui additionne deux matrices.

set_0 pour Matrice

Rappelez-vous le début du cours: *Pour initialiser tous les éléments de ce tableau, il faut utiliser deux boucles for imbriquées:*

```
for(int i = 0; i < NL; i++)  
    for(int j = 0; j < NC; j++)  
        x[i][j] = 0;
```

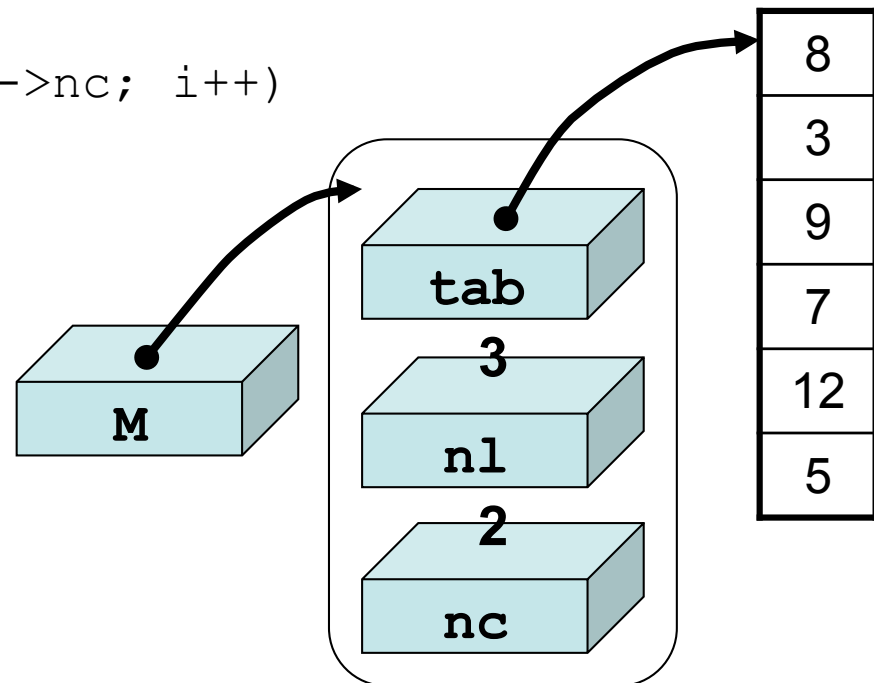
```
void set_0(Matrice * M)  
{  
    for(int i = 0; i < M->nl; i++)  
        for(int j = 0; j < M->nc; j++)  
            set(M, i, j, 0);  
}
```

set_0 pour Matrice

On peut aussi se rappeler que les éléments sont placés dans `tab` de taille `nl x nc`:

On peut donc écrire:

```
void set_0(Matrice * M)
{
    for(int i = 0; i < M->nl * M->nc; i++)
        M->tab[i] = 0.;
}
```



set_Id (1^{ère} solution)

```
bool set_Id(Matrice * M)
{
    if (M->nl != M->nc)
        return false;

    set_0(M);

    for(int i = 0; i < M->nl; i++)
        set(M, i, i, 1);

    return true;
}
```

	0	1	2	...	nc-1
0	1	0	0		0
1	0	1	0		0
2	0	0	1		0
...					0
nl-1	0	0	0	0	1

set_Id (2^{ème} solution)

```
bool set_Id(Matrice * M)
{
    if (M->nl != M->nc)
        return false;

    for(int i = 0; i < M->nl; i++)
        for(int j = 0; j < M->nc; j++)
            if (i == j)
                set(M, i, j, 1);
            else
                set(M, i, j, 0);

    return true;
}
```

	0	1	2	...	nc-1
0	1	0	0		0
1	0	1	0		0
2	0	0	1		0
...					0
nl-1	0	0	0	0	1

init pour les matrices

```
void init(Matrice * M, float c1, float c2, float c3, float c4)
{
    if (M->nl == 2 && M->nc == 2)
    {
        set(M, 0, 0, c1);
        set(M, 0, 1, c2);
        set(M, 1, 0, c3);
        set(M, 1, 1, c4);
    }
}
```

En utilisant tab directement:

```
void init(Matrice * M, float c1, float c2, float c3, float c4)
{
    if (M->nl == 2 && M->nc == 2)
    {
        M->tab[0] = c1;
        M->tab[1] = c2;
        M->tab[2] = c3;
        M->tab[3] = c4;
    }
}
```

init pour les matrices

```
void init(Matrice * M,  
          float c1, float c2, float c3, float c4, float c5, float c6)  
{  
    if (M->nl == 3 && M->nc == 2)  
    {  
        set(M, 0, 0, c1);  
        set(M, 0, 1, c2);  
        set(M, 1, 0, c3);  
        set(M, 1, 1, c4);  
        set(M, 2, 0, c5);  
        set(M, 2, 1, c6);  
    }  
    else if (M->nl == 3 && M->nc == 2)  
    {  
        set(M, 0, 0, c1);  
        set(M, 0, 1, c2);  
        set(M, 0, 2, c3);  
        set(M, 1, 0, c4);  
        set(M, 1, 1, c5);  
        set(M, 1, 2, c6);  
    }  
}
```

init pour les matrices

En utilisant `tab` directement, on constate qu'on n'a pas besoin de distinguer les cas:

```
void init(Matrice * M,
          float c1, float c2, float c3, float c4, float c5, float c6)
{
    if ((M->n1 == 3 && M->nc == 2) || (M->n1 == 2 && M->nc == 3))
    {
        M->tab[0] = c1;
        M->tab[1] = c2;
        M->tab[2] = c3;
        M->tab[3] = c4;
        M->tab[4] = c5;
        M->tab[5] = c6;
    }
}
```

add pour les matrices

```
void add(Matrice * A, Matrice * B, Matrice * Res)
{
    for(int i = 0; i < A->nl; i++)
        for(int j = 0; j < A->nc; j++)
            set(Res, i, j, get(A, i, j) + get(B, i, j));
}
```

En utilisant tab directement, on peut également écrire:

```
void add(Matrice * A, Matrice * B, Matrice * Res)
{
    for(int i = 0; i < A->nl * A->nc; i++)
        Res->tab[i] = A->tab[i] + B->tab[i];
}
```

add pour les matrices

On peut également avoir besoin d'une fonction additionnant deux matrices d'en-tête suivant:

```
Matrice * add(Matrice * A, Matrice * B)
```

la fonction allouant elle-même la matrice contenant le résultat.

Les deux solutions sont bonnes, et choisir entre les deux dépend du problème à résoudre.

```
Matrice * add(Matrice * A, Matrice * B)
{
    Matrice * Res = alloue_matrice(A->nl, A->nc);

    add(A, B, Res);

    return Res;
}
```

Exercices

Écrivez les fonctions `trace` et `produit_scalaire`:

- `float trace(Matrice * A)`

La trace d'une matrice (nécessairement carrée) est la somme des éléments sur la diagonale.

- `float produit_scalaire(Vecteur * V1, Vecteur * V2)`

Rappel

Le code qui calcule la somme

```
tab[0] + tab[1] + tab[2] + tab[3]+...
```

des éléments d'un tableau est:

```
somme = 0;  
for(int i = 0; i < N; i++)  
    somme = somme + tab[i];
```

Fonction trace

	0	1	2	3	4
0	○				
1		○			
2			○		
3				○	
4					○

Matrice A

Il s'agit de calculer la somme des éléments sur la diagonale de la matrice, c'est-à-dire la somme:

$$A_{0,0} + A_{1,1} + A_{2,2} + A_{3,3} + A_{4,4}$$

```
float trace(Matrice * A)
{
    float tr = 0;

    for(int i = 0; i < A->n1; i++)
        tr = tr + get(A, i, i);

    return tr;
}
```

Fonction produit_scalaire

Calculer le produit scalaire de U et V signifie calculer la somme suivante:

$$U_0 * V_0 + U_1 * V_1 + U_2 * V_2 + \dots$$

```
float produit_scalaire(Vecteur * U, Vecteur * V)
{
    float somme = 0;

    for(int i = 0; i < U->n; i++)
        somme = somme + get(U, i) * get(V, i);

    return somme;
}
```

Exercices

- Écrivez la fonction `mult` qui multiplie une matrice et un vecteur:

```
void mult(Matrice * A, Vecteur * V, Vecteur * Res)
```

`Res` contiendra le produit de la matrice `A` et du vecteur `V`;

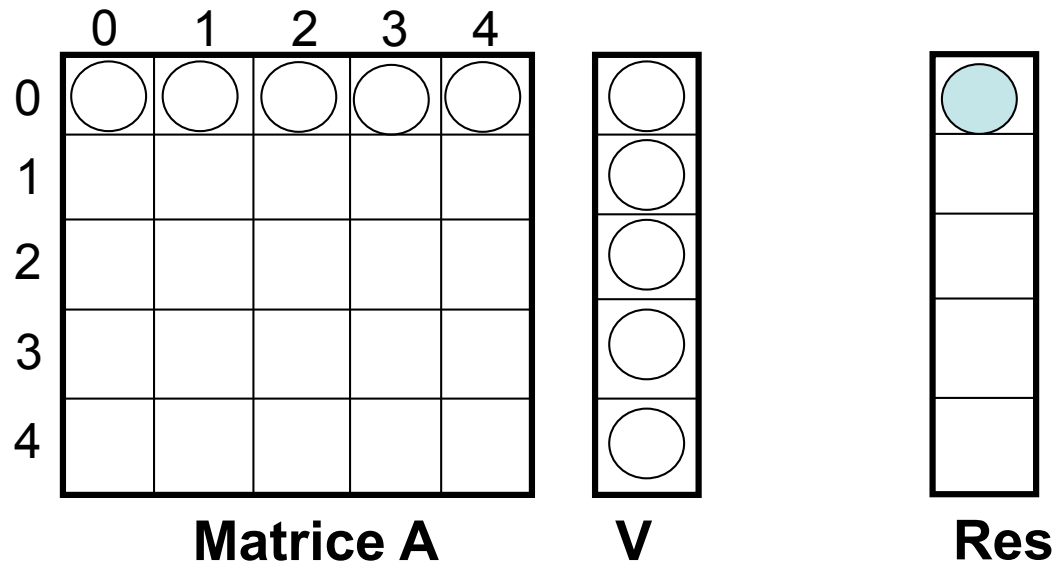
- Écrivez la fonction `mult` qui multiplie deux matrices entre elles:

```
void mult(Matrice * A, Matrice * B, Matrice * Res)
```

`Res` contiendra le produit des matrices `A` et `B`.

Produit d'une matrice et d'un vecteur

Calculer le produit de la matrice A et du vecteur V et mettre le résultat dans le vecteur Res :



Le premier élément de Res vaut la somme:

$$Res_0 = A_{0,0} \times V_0 + A_{0,1} \times V_1 + A_{0,2} \times V_2 + A_{0,3} \times V_3 + A_{0,4} \times V_4$$

```
float somme = 0;
for(int j = 0; j < A->nc; j++)
    somme = somme + get(A, 0, j) * get(V, j);
set(Res, 0, somme);
```

Produit d'une matrice et d'un vecteur

Pour calculer Res_0 , on fait donc:

```
float somme = 0;
for(int j = 0; j < A->nc; j++)
    somme = somme + get(A, 0, j) * get(V, j);
set(Res, 0, somme);
```

Pour calculer tous les éléments de Res , il suffit donc de faire:

```
for(int i = 0; i < A->nl; i++)
{
    float somme = 0;
    for(int j = 0; j < A->nc; j++)
        somme = somme + get(A, i, j) * get(V, j);
    set(Res, i, somme);
}
```

Fonction mult

```
void mult(Matrice * A, Vecteur * V, Vecteur * Res)
{
    for(int i = 0; i < A->nl; i++)
    {
        float somme = 0;
        for(int j = 0; j < A->nc; j++)
            somme = somme + get(A, i, j) * get(V, j);
        set(Res, i, somme);
    }
}
```

Produit de deux matrices

Mettre le produit des matrices A et B dans Res:

	0	1	2	3	4
0					
1					
2					
3					
4					

Matrice A

	0	1	2	3	4
0					
1					
2					
3					
4					

Matrice B

	0	1	2	3	4
0					
1					
2					
3					
4					

Matrice Res

$Res_{0,0}$ vaut:

$$Res_{0,0} = A_{0,0} * B_{0,0} + A_{0,1} * B_{1,0} + A_{0,2} * B_{2,0} + \dots$$

$Res_{0,1}$ vaut:

$$Res_{0,1} = A_{0,0} * B_{0,1} + A_{0,1} * B_{1,1} + A_{0,2} * B_{2,1} + \dots$$

Produit de deux matrices

$$\text{Res}_{0,1} = A_{0,0} * B_{0,1} + A_{0,1} * B_{1,1} + A_{0,2} * B_{2,1} + \dots$$

Pour calculer $\text{Res}_{0,1}$, il faut donc faire:

```
float somme = 0;
for(int k = 0; k < A->nc; k++)
    somme = somme + get(A, 0, k) * get(B, k, 1);
set(Res, 0, 1, somme);
```

Pour calculer tous les éléments de Res , il faut donc faire:

```
for(int i = 0; i < A->nl; i++)
    for(int j = 0; j < B->nc; j++)
    {
        float somme = 0;
        for(int k = 0; k < A->nc; k++)
            somme = somme + get(A, i, k) * get(B, k, j);
        set(Res, i, j, somme);
    }
```

Fonction mult

```
void mult(Matrice * A, Matrice * B, Matrice * Res)
{
    for(int i = 0; i < A->nl; i++)
        for(int j = 0; j < B->nc; j++)
        {
            float somme = 0;

            for(int k = 0; k < A->nc; k++)
                somme = somme + get(A, i, k) * get(B, k, j);

            set(Res, i, j, somme);
        }
}
```

Exercices

De la même façon, écrivez les fonctions:

`set_0`, qui initialise les éléments d'un vecteur à 0;

`mult` qui multiplie un vecteur par un scalaire;

`add` qui additionne deux vecteurs;

`transpose`, qui calcule la transposée d'une matrice;

`mult` qui calcule le produit d'une matrice par un scalaire;

`puissance_mat`, qui élève une matrice carrée à la puissance n ;

`determinant`, qui calcule le déterminant d'une matrice;

...

Des fonctions moins mathématiques:

`decale_a_gauche`, qui décale d'une colonne vers la gauche les éléments d'une matrice.

La colonne de gauche sera déplacée sur la colonne de droite;

`decale_a_droite`, qui décale d'une colonne vers la droite les éléments d'une matrice.

La colonne de droite sera déplacée sur la colonne de gauche;

Exercices

```
void mult(float k, Vecteur * V, Vecteur * kV)  
{  
    for(int i = 0; i < V->n; i++)  
        set(kV, i, k * get(V, i));  
}
```

```
void transpose(Matrice * A, Matrice * tA)  
{  
    for(int i = 0; i < A->nl; i++)  
        for(int j = 0; j < A->nc; j++)  
            set(tA, j, i, get(A, i, j));  
}
```