TP: tableaux à deux dimensions en C

Matrices

Exercice 1. Ecrire une fonction `void affiche_mat(int n, int m, double mat[n][m])` qui affiche le contenu d'une matrice.

Avec

```
double mat [2][2]={{1.,2.},{3.,4.}};
affiche_mat(2,2,mat);
```

on obtient l'affichage

```
|1.00 2.00|
|3.00 4.00|
```

Solution. Voici

```
void affiche_mat(int n, int m, double mat [n][m]){
  for (int i=0; i<n; i++){
    printf("|");
    for (int j=0; j<m; j++){
      printf("%5.2f",mat[i][j]);
      if (j<m-1) printf(" ");
    }//for j
    printf("|\n");
  }//for i
  printf("\n");
}// affiche_mat

```

□

Exercice 2. Ecrire une fonction `double det(double mat[2][2])` qui prend en paramètre une matrice 2×2 et renvoie son déterminant.

Solution. Voici

```
double det(double mat[2][2]){
  return mat[0][0]*mat[1][1]-mat[1][0]*mat[0][1];
}

```

□

Exercice 3. Écrire une procédure qui prend en paramètres deux matrices de mêmes dimensions, leurs dimensions et une matrice résultat. La fonction met dans la matrice résultat la somme des deux autres.

`void add(int n, int m, double M1[n][m], double M2[n][m], double res[n][m])` .

Solution. Voici

```
void add(int n, int m, double M1[n][m], double M2[n][m], double res[n][m]){
  for (int i=0; i<n; i++){
    for (int j=0; j<m; j++)
      res[i][j]=M1[i][j]+M2[i][j];
  }
}

```

□

Exercice 4. Si $A = (a_{ij})$ est une matrice de type (m, n) et $B = (b_{ij})$ est une matrice de type (n, p), alors leur produit, noté $AB = (c_{ij})$ est une matrice de type (m, p) donnée par :

$$\forall i, j : c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{in} b_{nj}.$$

Ecrire une procédure `void prod(int n, double M1[n][n], double M2[n][n],double res[n][n])` qui prend en paramètres deux matrices carrées de même dimension, leur dimension et met dans une matrice résultat le produit matriciel $M_1 \cdot M_2$.

```
mat1=
| 1.00 -1.00  2.00|
| 2.00  3.00  1.00|
|-2.00  1.00 -1.00|

mat2=
| 0.00  1.00 -3.00|
| 1.00 -1.00  1.00|
|-1.00  0.00  1.00|

prod(3,mat1,mat2,res); res=
|-3.00  2.00 -2.00|
| 2.00 -1.00 -2.00|
| 2.00 -3.00  6.00|
```

Exercice 5. Ecrire la procédure `void transpose(int n, int m, double M[n][m], double res[m][n])` qui met dans `res` la transposée de `M`.

```
mat1=
1.00 2.00 3.00
4.00 5.00 6.00
transpose(2,3,mat1,res); res=
1.00 4.00
2.00 5.00
3.00 6.00
```

Exercice 6. Si on place une matrice carrée M de taille $n \times n$ dans un repère orthonormé $(O, \vec{i}, \vec{j})$ en considérant que le coefficient $M_{i,j}$ est sur le point de coordonnées (i, j) alors la (sous-entendu *première*) diagonale de M coïncide avec la première bissectrice du repère.

Ainsi, la transposée de M peut-être vue comme le symétrique orthogonal de M par rapport à la première bissectrice du repère (cette symétrie échange juste les coordonnées).

Considérons maintenant la seconde diagonale de la matrice.

Ecrire la fonction `void anti_transpose(int n, double mat [n][n], double res [n][n])` qui met dans `res` la transposée de `M` par rapport à la seconde diagonale.

```
mat1=
| 1.00  2.00  3.00|
| 4.00  5.00  6.00|
| 7.00  8.00  9.00|

anti_transpose(3,mat1,res); res=
| 9.00  6.00  3.00|
| 8.00  5.00  2.00|
| 7.00  4.00  1.00|
```

Solution. Soit $I(i,j)$ un point. La seconde diagonale (D) de la matrice a pour équation $(D) : x+y = n-1$. Le vecteur $\overrightarrow{u}(-1,1)$ est un vecteur directeur de (D).

Notons $I'(x,y)$ le symétrique orthogonal de I par rapport à (D). Il vérifie :

$$\begin{cases} \overrightarrow{II'} \cdot \overrightarrow{u} & = & 0 \\ J & \in & (D) \end{cases}$$

en notant J le milieu de $[I,I']$.

Alors

$$\begin{cases} (x-i)-(y-j) & = & 0 \\ \dfrac{x+i}{2} + \dfrac{y+j}{2} & = & n-1. \end{cases}$$

Et on obtient $x = -j+n-1$ et $y = -i+n-1$.

```c
void transpose_autre(int n,
         double M[n][n],
         double res[n][n]){
  for (int i=0; i<n; i++)
    for (int j=0; j<n; j++){
      res[i][j] = M[n-j-1][n-i-1];
    }
}

void bench_transpose_autre(){
  double mat1 [3][3]={{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}};
  double res  [3][3]={{0.,0.,0.},{0.,0.,0.},{0.,0.,0.}};
  printf("mat1=\n");
  affiche_mat(3,3,mat1);
  transpose_autre(3,mat1,res);
  printf("transpose_autre(3,mat1,res); res=\n");
  affiche_mat(3,3,res);
}
```

□

Géométrie

Dans cet exercice, les points et les vecteurs sont représentés par des matrices colonnes de deux doubles. Par exemple $\begin{pmatrix}2\\3\end{pmatrix}$ est représenté par `{{2},{3}}`. La base du repère est supposée orthonormée.

Pour ne pas écrire `int [2][1]` trop souvent (c'est long !) on définit un alias de type :

```
typedef double vect [2][1];
```

De la sorte, le code suivant est licite :

```
vect v = {{2},{1}};
affiche_mat(2,1,v);
```

On obtient l'affichage :

```
|2.00|
|1.00|
```

De même, on définit un type des matrices 2×2 par

```
typedef double matrix [2][2];
```

Exercice 7.

1. Ecrire la fonction `void vects2mat(vect v1, vect v2, matrix A);` qui permet de fusionner deux vecteurs en une matrice.
2. Ecrire la fonction `void solve(matrix A, vect b, vect x);` qui résoud l'équation matricielle $AX = B$ et met le résultat dans X.

Solution. On rappelle que l'inverse de la matrice carrée inversible $\begin{pmatrix}a & b\\c & d\end{pmatrix}$ est $\frac{1}{ad-bc}\begin{pmatrix}d & -b\\-c & a\end{pmatrix}$.

```
void vects2mat(vect v1, vect v2, matrix A){
  for (int i=0; i<2; i++){
   A[i][0]=v1[i][0];
   A[i][1]=v2[i][0];
  }
}

void test_vects2mat(){
  matrix A={{0,0},{0,0}};
  vect v1 = {{0},{1}}, v2 = {{1},{0}};
  vects2mat(v1,v2,A);
  affiche_mat(2,2,A);
}


void _scal_prod(double x,matrix A){
  for (int i=0; i<2; i++)
    for (int j=0; j<2; j++)
      A[i][j]=A[i][j]*x;
}

void _prodmatvect(matrix A, vect V, vect X){
  for (int i=0;i<2;i++)
    X[i][0]=A[i][0]*V[0][0]+A[i][1]*V[1][0];
}
```

```
26
27 void solve(matrix A, vect b, vect x){
28   double det = A[0][0]*A[1][1]-A[1][0]*A[0][1];
29   assert(det!=0);
30   matrix invA = {{A[1][1],-A[0][1]},
31      {-A[1][0],A[0][0]}};
32   _scal_prod(1/det,invA);// effets de bord
33   //affiche_mat(2,2,invA);
34   _prodmatvect(invA,b,x);
35 }
36
37 void test_solve(){
38   matrix A = {{2,3},{1,-1}};
39   vect B= {{8},{-1}};
40   vect X = {{0},{0}};
41   solve(A,B,X);
42   printf("\n");
43   affiche_mat(2,1,X);
44   printf("\n");
45 }
```

□

Exercice 8. 1. Écrire la procédure

`void combinaison_lineaire(double a, double b, vect V1 , vect V2, vect res)`

qui prend en paramètres deux réels a, b (représentés par `a` et `b`), deux vecteurs $\overrightarrow{V}_1 \in \mathbb{R}^2$ et $\overrightarrow{V}_2 \in \mathbb{R}^2$ (représentés par `V1` et `V2`) et met dans `res` le résultat de la combinaison linéaire $a\overrightarrow{V}_1 + b\overrightarrow{V}_2$.

Solution. Voici

```
1 void combinaison_lineaire(double a, double b,
2     double V1[2][1], double V2[2][1], double res[2][1]){
3   for (int i=0; i<2; i++){
4     res[i][0]=a*V1[i][0]+b*V2[i][0];
5   }
6 }
7
```

□

2. Ecrire une fonction `void rotateV(double theta, vect a, vect b);` qui met dans `b` le résultat de la rotation vectorielle d'angle `theta` appliquée au vecteur représenté par `a`.
3. Ecrire une fonction

   `void rotateA(double theta, vect C, vect a, vect b);`

   qui met dans `b` le résultat de la rotation affine d'angle `theta` et de centre représenté par `C` appliquée au point représenté par `a`.

Solution. Voici :

```
1
2  void rotateV(double theta, vect a, vect b){
3    matrix M = {{cos(theta),-sin(theta)},{sin(theta),cos(theta)}};
4    produit(2,2,1,M,a,b);
5  }
6
7  void test_rotateV(){
8    vect V1 = {{1},{3}};
9    vect V2;
10   rotateV(M_PI/3,V1,V2);
11   affiche_mat(2,1,V2);
12 }
13
14 void rotateA(double theta, vect C, vect A, vect B){
15   vect CA;
16   combinaison_lineaire(-1,1,C,A,CA);
17   vect CB;
18   rotateV(theta,CA,CB);
19   combinaison_lineaire(1,1,CB,C,B);
20 }
21 void test_rotateA(){
22   vect C = {{1},{3}};
23   vect A = {{-2},{5}};
24   vect B;
25   rotateA(M_PI/3,C,A,B);
26   affiche_mat(2,1,B);
27 }
28
```

□

Exercice 9.

1. Ecrire la fonction `bool appartient(vect A, vect B, vect C);` qui indique par un bouléen si le point C est sur la droite (AB).
2. Ecrire la fonction `double ps(vect U, vect V);` qui calcule le produit scalaire des vecteurs U, V.
3. Ecrire la fonction `void pied(vect A, vect B, vect C, vect D);` qui met les coordonnées du pied de la perpendiculaire à (AB) issue de C dans D.
4. Ecrire la fonction `void sym(vect A, vect B, vect C, vect D);` qui met les coordonnées du symétrique orthogonal de C par rapport à (AB) dans D.

Dans ces 3 exercices, on suppose que $A \neq B$ sans avoir besoin de le vérifier.

Solution.

1. Pour `appartient`, on se souvient que $C \in (AB)$ si et seulement si $\det(\overrightarrow{AB}, \overrightarrow{AC}) = 0$.

```
bool appartient(vect A, vect B, vect C){
  vect AB;
  combinaison_lineaire(1,-1,B,A,AB);// AB<-B-A
  vect AC;
  combinaison_lineaire(1,-1,C,A,AC);
  matrix M;
  vects2mat(AB,AC,M);
  double d = det(M);
  return (d == 0.);
}

void test_appartient(){
  vect A = {{1},{2}};
  vect B = {{3},{4}};
  vect C1 = {{2},{3}}; // C1 \in (AB)
  vect C2 = {{4},{6}}; // C1 \notin (AB)
  printf("A=\n");  affiche_mat(2,1,A);printf("\n");
  printf("B=\n");  affiche_mat(2,1,B);printf("\n");
  printf("C1=\n");  affiche_mat(2,1,C1);  printf("\n");
  printf("C1 in (AB): %d\n",appartient(A,B,C1));

  printf("C2=\n");  affiche_mat(2,1,C2);  printf("\n");
  printf("C2 in (AB): %d\n",appartient(A,B,C2));
}

```

2. Pour `ps`, pas de difficulté : c'est la somme des produits des coordonnées de même numéro :

```
double ps(vect V1, vect V2){
  double s =0;
  for (int i=0; i<2; i++)
    s = s + V1[i][0]*V2[i][0];
  return s;
}

void test_ps(){
  vect U = {{1},{2}};
  vect V = {{-3},{4}};
  printf("U=\n");  affiche_mat(2,1,U);printf("\n");
  printf("V=\n");  affiche_mat(2,1,V);printf("\n");
  printf("U.V=%5.2f\n",ps(U,V));
}
```

3. Pour le pied : Si $C \in (AB)$, le pied de la perpendiculaire est le point C. Sinon, on cherche le point H tel que $\overrightarrow{AB} \cdot \overrightarrow{CH} = 0$ et $H \in \overrightarrow{AB}$ (*i.e.* $\det(\overrightarrow{AB}, \overrightarrow{AH}) = 0$).
   Si (x, y) sont les coordonnées de H, alors on a les deux contraintes :

$$\begin{cases} x_{\overrightarrow{AB}} x_{\overrightarrow{CH}} & + & y_{\overrightarrow{AB}} y_{\overrightarrow{CH}} & = & 0 \\ x_{\overrightarrow{AB}} y_{\overrightarrow{AH}} & - & y_{\overrightarrow{AB}} x_{\overrightarrow{AH}} & = & 0 \end{cases}$$

   Ce système s'écrit donc aussi

$$\begin{cases} x_{\overrightarrow{AB}}(x - x_C) & + & y_{\overrightarrow{AB}}(y - y_C) & = & 0 \\ x_{\overrightarrow{AB}}(y - y_A) & - & y_{\overrightarrow{AB}}(x - x_A) & = & 0 \end{cases}$$

ce qui est équivalent à

$$\begin{cases} x_{\overrightarrow{AB}}x & + & y_{\overrightarrow{AB}}y & = & x_{\overrightarrow{AB}}x_C + y_{\overrightarrow{AB}}y_C = \overrightarrow{AB}\cdot\overrightarrow{OC} \\ -y_{\overrightarrow{AB}}x & + & x_{\overrightarrow{AB}}y & = & x_{\overrightarrow{AB}}y_A - y_{\overrightarrow{AB}}x_A = \det(\overrightarrow{AB},\overrightarrow{OA}) \end{cases}$$

L'algorithme consiste donc à résoudre le système ci-dessus.

```
void pied(vect A, vect B, vect C, vect X){
  vect AB;
  combinaison_lineaire(1,-1,B,A,AB);
  matrix mat_AB_A;
  vects2mat(AB,A,mat_AB_A);
  vect D = {{ps(C,AB)},{det(mat_AB_A)}};
  matrix M = {{AB[0][0],AB[1][0]},{-AB[1][0],AB[0][0]}};
  // résoudre MX = D
  solve(M,D,X);
}

void test_pied(){
  vect A = {{1},{2}};
  vect B = {{3},{4}};
  vect I = {{2},{3}}; // I \in (AB)
  vect C = {{-1},{6}}; // C \notin (AB)
  vect H;
  printf("A=\n");  affiche_mat(2,1,A);printf("\n");
  printf("B=\n");  affiche_mat(2,1,B);printf("\n");
  printf("I=\n");  affiche_mat(2,1,I);  printf("\n");
  printf("pied perp à (AB) passant par I :\n");
  pied(A,B,I,H);
  affiche_mat(2,1,H);// on doit retrouver I

  printf("C=\n");  affiche_mat(2,1,C);  printf("\n");
  printf("pied perp à (AB) passant par I :\n");
  pied(A,B,C,H);
  affiche_mat(2,1,H);// on doit retrouver I
}
```

4. Soit C un point et H le pied de la perpendiculaire à (AB) issue C. Le point C', symétrique orthogonal de C par rapport à (AB) est tel que H est le milieu de $[C, C']$.
   On a donc
   $$\frac{C + C'}{2} = H$$
   D'où $C' = 2H - C$.

```
void sym(vect A,vect B,vect C,vect X){
  vect H;
  pied(A,B,C,H);
  combinaison_lineaire(2,-1,H,C,X);
}

void test_sym(){
  vect A = {{1},{2}};
  vect B = {{3},{4}};
  vect I = {{2},{3}}; // I \in (AB)
  vect C = {{-1},{6}}; // C \notin (AB)
  vect X;
  printf("A=\n");  affiche_mat(2,1,A);printf("\n");
  printf("B=\n");  affiche_mat(2,1,B);printf("\n");
  printf("I=\n");  affiche_mat(2,1,I);  printf("\n");
  printf("sym de C / à (AB) :\n");
  sym(A,B,I,X);
  affiche_mat(2,1,X);

  printf("C=\n");  affiche_mat(2,1,C);  printf("\n");
  printf("sym de C / à (AB) :\n");
  sym(A,B,C,X);
  affiche_mat(2,1,X);
}

```

□