MT51 - TP2
Étude des isométries du plan

Laurent Couvidou – rendu le 23 juin 2006

---

1. Détermination de la matrice projective d'une isométrie du plan

1. *Fonction générale*

On écrit une fonction commune à toutes les isométries du plan : elle renvoie la matrice projective de l'isométrie demandée (cf. `matprojisoplan.m`), qu'il s'agisse :

- d'une translation ;
- d'une rotation.
- ou d'un glissement ;

Comme vu en cours, ce sont les trois seules isométries du plan.

Les paramètres sont contrôlés par une fonction extérieure (cf. `verif2d.m`).

NB : le nombre de paramètres pouvant varier en fonction du type d'isométrie demandée (un vecteur pour une translation, un point et un angle pour une rotation, etc.), on utilise un tableau de paramètres en entrée (syntaxe `{}` de Matlab). Voir exemple en 2.

2. *Fonctions particulières*

Reste à traiter les trois fonctions calculant réellement les matrices projectives, de la plus simple à la plus complexe.

- Translation (cf. `matprojtrans.m`) :

  On utilise les propriétés des coordonnées homogènes, donc pour une translation de vecteur $\vec{t}$, la matrice projective T est :

  $$\vec{t}\begin{pmatrix} t_x \\ t_y \end{pmatrix} \quad T = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

- Rotation (cf. `matprojrot.m`) :

  C'est une simple matrice de rotation autour de l'origine, multipliée à droite et à gauche par des matrices translations, pour prendre en compte la position du centre (calculs symboliques dans le script `symrot.m`).

  Pour un centre C et un angle θ, la matrice projective R obtenue est :

  $$C\left(c_x, c_y\right) \quad T = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & -\cos(\theta)\,c_x + \sin(\theta)\,c_x + c_x \\ \sin(\theta) & \cos(\theta) & -\sin(\theta)\,c_x - \cos(\theta)\,c_x + c_y \\ 0 & 0 & 1 \end{pmatrix}$$

- Glissement (cf. `matprojglis.m`) :

  Un glissement nécessite un changement de repère complet (translation et rotation), pour appliquer une symétrie de matrice simple (par exemple, autour de l'axe des abcisses). Avec O l'origine, P point du miroir, θ l'angle qu'il forme avec l'axe des abcisses, et $\vec{t}$ vecteur du glissement, on obtient la matrice projective G par :

  $$G = T_{\vec{OP}} \circ R_{-\theta} \circ S_{abcisses} \circ T_{\vec{PO}} \circ R_{\theta} \circ T_{\vec{t}}$$

  On obtient θ facilement à partir du vecteur directeur du miroir, en utilisant la fonction `atan2` de Matlab.

  Les calculs sont effectués en symbolique avec Matlab : cf. `symglis.m`. On recopie la matrice obtenue.

  NB : il serait possible d'effectuer directement les produit matriciels dans la fonction, ce qui gagnerait en clarté. Cependant, en donnant l'expression directe, on économise à la fois en temps de calcul et en précision, c'est donc la méthode que l'on adopte.

2. Étude géométrique d'une matrice projective d'isométrie

On utilise le théorème 18 du cours sur les isométries affines, pour déterminer quelle isométrie se « cache » derrière une matrice M.

- Soit le déterminant de M est 1 (à un epsilon près, pour tenir compte de l'erreur numérique) :
  - Alors, si la linéaire associée est une matrice unité, il s'agit d'une simple translation, dont on récupère les paramètres directement ;
  - Sinon il s'agit d'une rotation. Dans ce cas on obtient le centre avec le vecteur propre associé à la valeur 1, et l'angle se déduit des cosinus et sinus présents dans la linaire associée (on utilise la fonction Matlab `atan2` pour trouver directement le signe de l'angle) ;
- Soit le déterminant de M est -1, toujours à un epsilon près. Dans ce cas nous avons à faire à un glissement. On situe un point sur l'axe en prenant le milieu entre un point et son image (calculée avec la matrice). On applique une nouvelle fois cette matrice au milieu pour obtenir le vecteur de glissement.

Cf. l'implémentation dans `nature.m`.

On retourne les paramètres sous la même forme que ceux de `matprojisoplan`, ce qui permet de vérifier le code sur quelques exemples.

Pour la translation :

```
>> T=matprojisoplan('trans',{[4;3]})  % on passe le vecteur
T =  1     0     4                    % la matrice calculée
     0     1     3                    % est correcte
     0     0     1
>> [n,param]=nature(T)                 % calcul inverse
n = translation                       % type OK
>> param{1}
ans =                                 % vecteur OK
     4
     3
```

Pour le glissement :

```
>> param{1} = [2,1];            % un point de l'axe
>> param{2} = [3;2];            % vecteur directeur et de translation
>> G=matprojisoplan('glis',param) % calcul de la matrice
G =
    0.3846    0.9231    3.3077
    0.9231   -0.3846    1.5385
         0         0    1.0000
>> [n,param2] = nature(G)       % calcul inverse
n = glissement                  % type OK
>> param2{1}                    % un point de l'axe ??
ans = 1.6538    0.7692
>> param2{2}                    % vecteur directeur et de translation OK
ans =
     3
     2
```

On obtient pas le même point, ce qui est logique : il n'y a aucun moyen de retrouver le point fourni initialement pour calculer la matrice, et on peut prendre n'importe quel point pour positionner l'axe dans l'espace. Vérifions que tout concorde, en calculant l'ordonnée à l'origine b, avant et après :

- b = 1 – 2 * 2/3 = -1/3
- b = 0.7692 – 1.6538 * 2/3 ≈ -0.3333 ≈ -1/3

Tout va bien, les résultats concordent.

Pour la rotation :

```
>> R=matprojisoplan('rot',{[1,2] pi/6})      % matrice de rot.
R =                                          % centre (1,2) et
    0.8660   -0.5000    1.1340               % d'angle pi/6
    0.5000    0.8660   -0.2321
         0         0    1.0000
>> [n,param]=nature(R)                       % calcul inverse
n = rotation                                 % type OK
>> param{1}
ans =  1.0000    2.0000                      % centre OK
>> param{2}
ans = 0.5236
>> pi/6
ans = 0.5236                                 % angle OK aussi
```

3. Outils graphiques

1. *Image d'un point*

On écrit pour commencer les routines la routine `transformepoint.m`, qui calcule les coordonnées de l'image d'un point donné par une isométrie de paramètres fournis (sous la même forme qu'en 1. et 2.).

Ex. : calcul de l'image de (7,8) par le glissement de point (3,4) et de vecteur (5,6) :

```
>> transformepoint('glis',{[3,4] [5;6]},[7,8])
ans =   11.2131   14.6557
```

Le principe est simple : on multiplie le point par la matrice, calculée avec les outils

précédents. Les fonctions de `translatpoint.m`, `rotepoint.m` et `glissepoint.m` se contentent d'appeler cette dernière routine.

2. *Figure de type line*

On étends le code précédent dans `transforme.m` pour qu'il puisse s'appliquer à un ensemble de points, qu'on pourra dessiner avec la fonction `line` de Matlab.

Ex. calcul de l'image de (1,2) (3,4) et (5,6) par la rotation de centre (7,8) et d'angle pi/9 :

```
>> transforme('rot',{[7,8] pi/9},[1,2;3,4;5,6])
ans =
    3.4140    0.3097
    4.6093    2.8731
    5.8047    5.4366
```

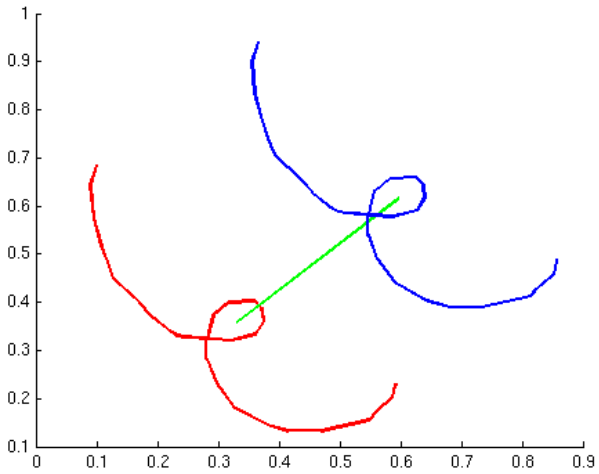
Les fonctions de `translat.m`, `rote.m` et `glisse.m` se contentent aussi d'appeler la routine précédente.

4. Question ouverte : écriture de démos

On écrit un petit script (cf. `demoio.m`) utilisant la fonction `ginput` de Matlab pour tester les fonction écrites précédemment. L'utilisateur peut ainsi entrer à la main les paramètres des isométries, du moins quand cela est possible.

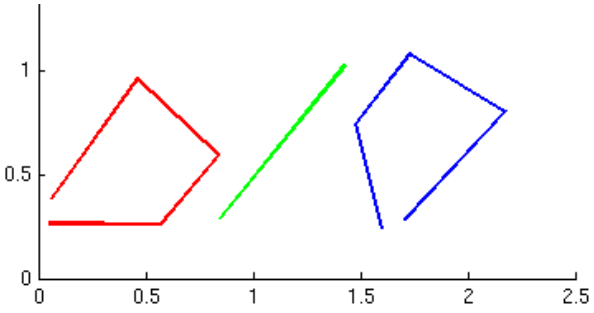
Pour la translation :

```
>> demoiso

Choisir une isométrie du plan
0) Translation
1) Glissement
2) Rotation
? 0
Entrez les points d'une figure à
transformer.
[Entrée] pour terminer la saisie...
Entrez deux points, donnant le vecteur
de translation...
```



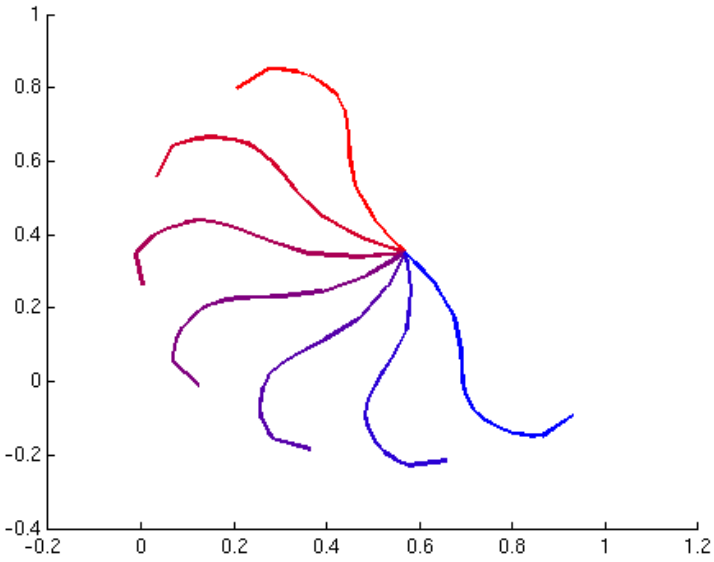
Pour le glissement

```
>> demoiso

Choisir une isométrie du plan
0) Translation
1) Glissement
2) Rotation
? 1
Entrez les points d'une figure à
transformer.
[Entrée] pour terminer la saisie...
Entrez deux points formant le vecteur
de glissement...
```



Et enfin, pour la rotation (on accepte une série de rotations) :

```
>> demoiso

Choisir une isométrie du plan
0) Translation
1) Glissement
2) Rotation
? 2
Donnez un angle, voire une matrice
ligne d'angles :
[pi/6,2*pi/6,3*pi/6,4*pi/6,5*pi/6,pi]
Entrez les points d'une figure à
transformer.
[Entrée] pour terminer la saisie...
Entrez le centre...
```



5. Conclusion

Ce TP est une très bonne introduction aux matrices d'isométries : nous avons appréhendé le problème en son entier, et avons pu effectuer des applications directes. Il est utile pour toutes les applications 3D d'aujourd'hui, et aide directement, par exemple, en IN55.

Le temps nous a fait défaut, mais une poursuite du travail vers d'un côté les matrices projectives, et de l'autre les quaternions, seraient d'un grand intérêt en MT51.