Piana Victor

Rapport TP2 MV55

Table des matières

Question 1 — Implémentation des transformations géométriques

Dans cette première partie, l'objectif est de coder trois transformations géométriques fondamentales en géométrie projective : l'homothétie, la rotation et la translation. Ces transformations sont définies dans le plan affine rapporté au repère (O, i, j), et représentées sous forme matricielle dans l'espace projectif.

1.a – Homothétie

L'homothétie est une transformation du plan qui dilate ou contracte les points par rapport à un centre donné. Ici, le centre est l'origine O, et le rapport est un réel k non nul. La transformation est modélisée par une matrice projective 3x3, dont les coefficients diagonaux sont égaux à k pour les coordonnées spatiales, et 1 pour la coordonnée homogène. Cela permet de conserver la nature projective de la transformation.

```python
# ----------------------
# QUESTION 1.a : Homothétie
# ----------------------

# Fonction qui renvoie la matrice projective d'une homothétie de centre O
# et de rapport k (k ≠ 0)
def homothetie(k):
    H = np.array([
        [k, 0, 0],
        [0, k, 0],
        [0, 0, 1]
    ])
    return H
```

Le test réalisé pour un rapport k = 0,5 montre que les coordonnées des points sont bien divisées par deux, ce qui correspond à une contraction vers l'origine.

```
 [0.  0.  1. ]]
PS C:\Users\victo\Desktop\UTBM FISE\BR03\MV55\TP
o\Desktop\UTBM FISE\BR03\MV55\TP2\TP2 MV55.py"
Homothétie de rapport 0.5 :
[[0.5 0.  0. ]
 [0.  0.5 0. ]
 [0.  0.  1. ]]
```

1.b – Rotation

La rotation est une transformation qui conserve les distances et les angles (isométrie directe). Elle est définie par un angle alpha, exprimé en radians, et un centre de rotation, ici, l'origine O. La matrice projective correspondante est construite à partir des fonctions trigonométriques cosinus et sinus de l'angle alpha. Elle suit la forme classique d'une matrice de rotation dans le plan.

```python

# ---------------------
# QUESTION 1.b : Rotation
# ---------------------

# Fonction qui renvoie la matrice projective d'une rotation de centre O
# et d'angle alpha (en radians)
def rotation(alpha):
    cos = np.cos(alpha)
    sin = np.sin(alpha)

    R = np.array([
        [cos, -sin, 0],
        [sin,  cos, 0],
        [  0,    0, 1]
    ])
    return R
```

Le test effectué avec alpha = pi/4 (soit 45°) confirme que la matrice est bien construite : elle est orthogonale, avec un déterminant égal à +1, ce qui caractérise les rotations dans un espace euclidien.

```python

print("\nRotation d'angle pi/4 :")
print(rotation(np.pi/4))
```

```
PROBLEMS  OUTPUT  TERMINAL  PORTS

TERMINAL

PS C:\Users\victo\Desktop\UTBM FISE\BR03\MV55\TP2> python -u "c:\U
o\Desktop\UTBM FISE\BR03\MV55\TP2\TP2 MV55.py"

Rotation d'angle pi/4 :
[[ 0.70710678 -0.70710678  0.        ]
 [ 0.70710678  0.70710678  0.        ]
 [ 0.          0.          1.        ]]
PS C:\Users\victo\Desktop\UTBM FISE\BR03\MV55\TP2>
```

1.c – Translation

La translation est une transformation affine qui déplace tous les points du plan selon un vecteur donné. Dans ce TP, ce vecteur est défini par les coordonnées du point a = (ax, ay). La matrice projective intègre ce vecteur dans la dernière colonne, ce qui permet d'ajouter ax et ay aux coordonnées x et y de tout point du plan.

```python
# ---------------------
# QUESTION 1.c : Translation
# ---------------------

# Fonction qui renvoie la matrice projective d'une translation
# de vecteur allant de O vers a = (ax, ay)
def translation(a):
    ax, ay = a  # coordonnées du point a
    T = np.array([
        [1, 0, ax],
        [0, 1, ay],
        [0, 0,  1 ]
    ])
    return T
```

Un test avec le vecteur (1, 2) montre que la translation est bien appliquée : les points sont correctement déplacés selon ce vecteur.

```
58
59  print("\nTranslation selon le vecteur (1, 2) :")
60  print(translation((1, 2)))
61
```

PROBLEMS OUTPUT TERMINAL PORTS

TERMINAL

```
o\Desktop\UTBM FISE\BR03\MV55\TP2\TP2 MV55.py"

Translation selon le vecteur (1, 2) :
[[1 0 1]
 [0 1 2]
 [0 0 1]]
PS C:\Users\victo\Desktop\UTBM FISE\BR03\MV55\TP2>
```

Question 2 — Construction d'une fractale

2.a – Principe de construction

Dans cette partie, on cherche à générer une figure fractale à partir d'un segment de base, en utilisant les transformations géométriques définies précédemment (homothétie, rotation, translation). Le segment de départ, noté M0, relie les points O = (0, 0) et a = (0, 1). Il s'agit donc d'un segment vertical de longueur 1.

On construit ensuite un ensemble de transformations du plan noté G = {g0, g1, g2}, où :

- g0 est l'identité (on conserve le segment tel quel),
- g1 est la composition : translation vers le haut (vecteur (0, 1)), après une rotation de +45°, après une homothétie de rapport 0,5,
- g2 est construit de la même manière que g1, mais avec une rotation de -45°.

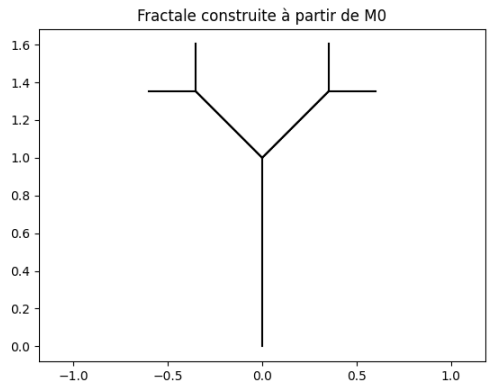
L'idée est d'appliquer ces trois transformations à chaque segment généré. On obtient alors une structure en arbre, où chaque segment donne naissance à trois nouveaux, et ainsi de suite à chaque itération.

2.b – Génération de la fractale

Dans la fonction `fract2`, on applique ce processus en deux étapes :

- D'abord, on applique g0, g1 et g2 au segment de départ pour obtenir un ensemble de trois segments (première génération).
- Ensuite, on recommence le même processus sur chaque segment obtenu, ce qui forme la deuxième génération.

Les segments sont manipulés en coordonnées homogènes, ce qui permet de leur appliquer facilement les transformations avec des produits matriciels. Le résultat est ensuite affiché à l'aide de `matplotlib`.



Le rendu visuel montre clairement la forme d'un arbre, avec des embranchements symétriques. Chaque branche se divise en deux, selon les angles de rotation définis, ce qui donne une figure fractale simple mais bien reconnaissable.

2.c – Fractale à partir d'un carré

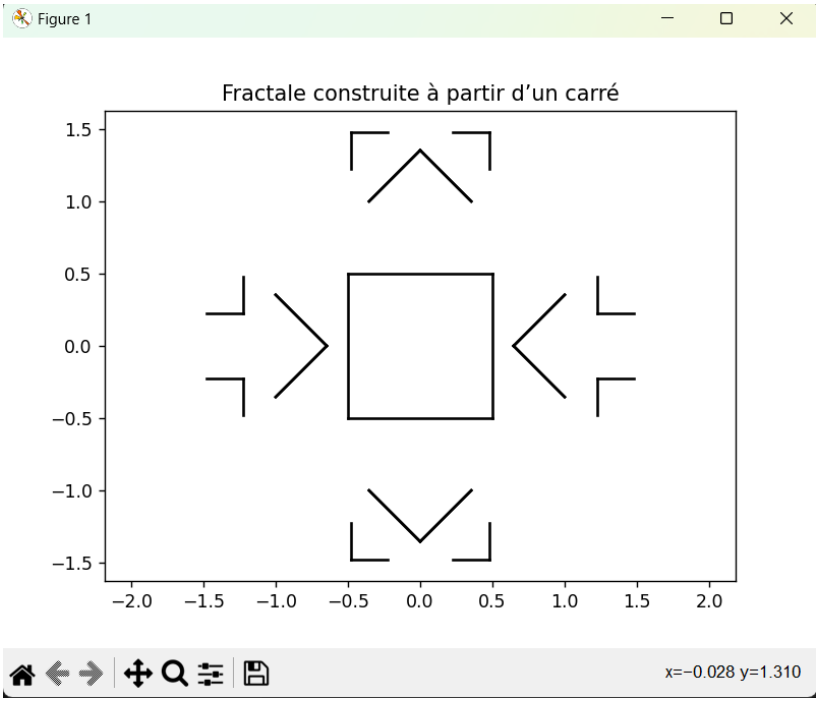
Dans cette partie, on change de stratégie en partant d'un motif de base différent : un carré centré en (0, 0), de côté 1. Plutôt que d'appliquer les transformations à un seul segment, on applique la fonction `fract2` à chacun des 4 côtés du carré, traités comme des segments indépendants.

Chaque côté est défini par deux points consécutifs, et orienté dans le sens trigonométrique (bas, droite, haut, gauche). Pour que la fractale se développe dans la bonne direction, le vecteur de translation utilisé dans `fract2` correspond simplement à la direction du côté (p2 - p1).

Les paramètres utilisés restent les mêmes :

- Rapport de l'homothétie : 0.5 (chaque segment devient deux fois plus petit),
- Angles de rotation : ±45°,
- Translation dans la direction de chaque côté.

Le résultat obtenu est une figure symétrique et plus complexe que dans les questions précédentes. À partir du carré central, on voit apparaître des motifs en forme de "V" tournés dans toutes les directions, ce qui illustre bien l'effet de la transformation appliquée à différents axes de symétrie. Cela montre aussi comment la fractale peut être enrichie simplement en variant la géométrie du motif initial.



2.d – Fractale en 3D (prolongement)

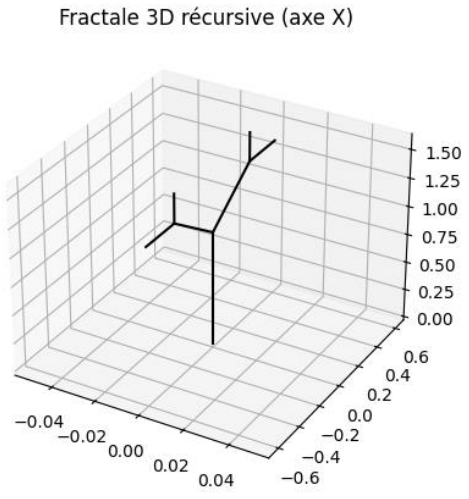
Pour aller plus loin, j'ai essayé d'adapter la construction de la fractale en 3D. Le principe reste le même que précédemment, sauf qu'il faut utiliser des matrices 4x4 pour gérer les transformations en trois dimensions.

J'ai repris l'idée de combiner une homothétie, une rotation et une translation. Cette fois, la rotation se fait autour de l'axe X, et la translation permet de déplacer les segments vers le haut (selon l'axe Z). Comme en 2.b, chaque segment donne naissance à deux nouveaux à chaque itération.

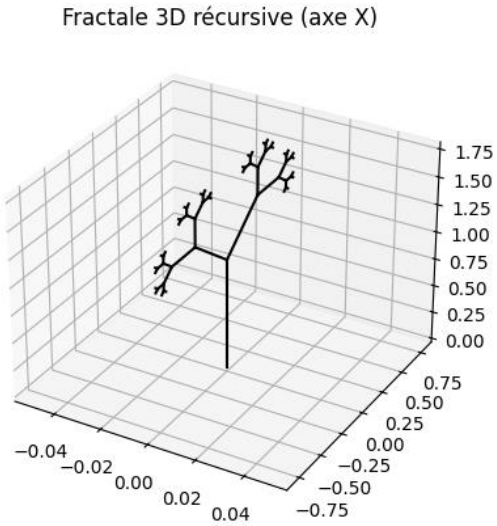
Voici un extrait du code utilisé pour créer la fractale en 3D :

```python
# fonction récursive pour dessiner la fractale
def fractale3d(ax, M0, depth):
    if depth == 0:
        return
    alpha = np.pi / 4
    k = 0.5
    trans = translation3d((0, 0, 1))  # on monte en Z
    h = homothetie3d(k)
    r1 = rotation_x(alpha)
    r2 = rotation_x(-alpha)
    g1 = trans @ r1 @ h
    g2 = trans @ r2 @ h
    M0_h = np.vstack((M0, np.ones((1, M0.shape[1]))))  # 4x2
    # On dessine le segment de base
    ax.plot(M0[0], M0[1], M0[2], 'k-')
    # On applique les deux transformations
    M1 = appliquer_transfo3d(M0_h, g1)
    M2 = appliquer_transfo3d(M0_h, g2)
    # On appelle récursivement sur les deux nouveaux segments
    fractale3d(ax, M1[:3], depth-1)
    fractale3d(ax, M2[:3], depth-1)
```

Ensuite, j'affiche le résultat avec matplotlib en mode 3D. Voici ce que ça donne avec une profondeur de 3 :



Puis une profondeur de 6 :



On obtient une sorte d'arbre qui se développe en hauteur avec des branches qui partent de chaque côté, comme en 2D mais dans l'espace.