Compte rendu de TP Calcul Géométrique

Manipulation de points et de triangles en 2D

Auteur : Mouhalhal

Introduction

L'objectif de ce TP était de créer deux classes en Python pour manipuler des objets géométriques dans le plan : la classe `Point` pour représenter un point en 2D, et la classe `Triangle` pour représenter un triangle à partir de trois points. On a implémenté des méthodes permettant de calculer des propriétés géométriques (distance, angle, barycentre, surface, etc.), d'effectuer des transformations (translation, rotation), et de visualiser les objets grâce à des tracés graphiques.

Classe Point

La classe `Point` permet de représenter un point dans un repère cartésien par ses coordonnées (x, y) et d'effectuer diverses opérations géométriques.

Méthodes principales

Pour simplifier, le point objet de la classe `Point` sera représenté par la lettre `S`.

- **`__init__(self, x, y)` :** le constructeur initialise un point à partir de ses coordonnées x et y.
- **copy(self) :** crée un point indépendant avec les mêmes coordonnées que le point `S`.
- **`__str__(self)` :** affiche le point `S` sous la forme `Point:(x,y)`.
- **`__eq__(self, P)` :** teste si les deux points `S` et `P` sont confondus en comparant la distance entre eux à une petite valeur de tolérance (`EPS_GEO` $= 10^{-5}$), afin de tenir compte des imprécisions numériques.
- **distance(self, P) :** calcule la distance entre `S` et `P`.
- **rayon(self) :** calcule la distance entre `S` et l'origine $(0, 0)$.
- **angle(self) :** calcule l'angle en degrés par rapport à l'axe des abscisses en utilisant l'arctangente de $\frac{y}{x}$ (la méthode `atan2(y, x)` résout le problème lorsque x est nul).
- **rotation(self, O, β) :** effectue la rotation du point S autour du point O d'un angle β. On obtient les nouvelles coordonnées (x', y') en utilisant la matrice de rotation :

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{bmatrix} \begin{bmatrix} x - O_x \\ y - O_y \end{bmatrix} + \begin{bmatrix} O_x \\ O_y \end{bmatrix},$$

— **translation(self, dx, dy) :** déplace le point d'un vecteur (dx, dy) en ajoutant ces derniers aux coordonnées du point S.
— **polaire(self) :** renvoie les coordonnées polaires du point sous la forme (r, θ), où r est le rayon par rapport au centre et θ l'angle par rapport à l'axe des abscisses.
— **plot(self) :** affiche le point sur un graphique.

Fonctions utilitaires externes

Deux fonctions indépendantes complètent la classe :
— `distance(P1, P2)` : calcule la distance entre les points P1 et P2.
— `produit_vect(A, B, C)` : calcule la composante suivant Z du produit vectoriel de $\overrightarrow{AB}$ et $\overrightarrow{AC}$.

Classe Triangle

La classe `Triangle` permet de représenter un triangle dans un plan cartésien à partir de trois points et de réaliser différentes opérations géométriques, comme le calcul du barycentre, du périmètre, de la surface, ainsi que des transformations et des tracés graphiques.

Méthodes principales

Pour simplifier, le triangle objet de la classe `Triangle` sera représenté par `Tr`.
— **__init__(self, P1, P2, P3) :** initialise le triangle à partir de trois points P_1, P_2 et P_3 (on utilise des copies des points pour éviter les problèmes de liaison).
— **__str__(self) :** affiche les points du triangle sous la forme `Triangle:(Point:(x,y), Point:(x,y), Point:(x,y))`.
— **__eq__(self, T) :** teste si les deux triangles `Tr` et `T` sont confondus en comparant leurs trois sommets.
— **barycentre(self) :** renvoie le point barycentre du triangle `Tr`, défini par :

$$G\left(\frac{x_1 + x_2 + x_3}{3}, \frac{y_1 + y_2 + y_3}{3}\right)$$

— **perimetre(self) :** calcule le périmètre du triangle `Tr` en additionnant les longueurs de ses trois côtés.
— **surface(self) :** calcule la surface du triangle `Tr` à l'aide du produit vectoriel des deux vecteurs formés par ses sommets :

$$S = \frac{|\overrightarrow{P1P2} \times \overrightarrow{P1P3}|}{2}$$

— **coordbary(self, P) :** calcule les coordonnées barycentriques du point P par rapport au triangle `Tr`. La coordonnée du point P par rapport au côté P_1P_2 du triangle

vaut :

$$S = \frac{|\overrightarrow{PP1} \times \overrightarrow{PP2}|}{2 \times \text{surface}(Tr)}$$

(L'ordre des points P_1 et P_2 est important ! Ils doivent être dans le sens antihoraire.)

— **inside(self, P) :** teste si le point P est à l'intérieur du triangle `Tr` en comparant la somme des surfaces formées par P et chaque côté du triangle à la surface globale du triangle.

— **translation(self, dx, dy) :** déplace le triangle `Tr` en déplaçant ses trois sommets d'un vecteur (dx, dy).

— **rotation(self, O, α) :** effectue la rotation du triangle `Tr` autour du point O d'un angle α en effectuant la rotation de chaque sommet.

— **rayon(self) :** calcule le rayon du cercle circonscrit au triangle `Tr`, centré en son barycentre, en déterminant la plus grande distance entre celui-ci et ses sommets.

— **intersection(self, T) :** teste s'il y a une intersection entre les triangles `Tr` et `T`, en vérifiant l'existence des points d'intersection entre leurs côtés (on a manipulé les équations des droites des côtés et les domaines de x et y qui délimitent les segments du triangle).

— **plot(self,col) :** affiche le triangle `Tr` sur un graphique avec un remplissage de couleur col (par défaut vert).

Tests et vérifications

On a réalisé des tests pour valider le bon fonctionnement des différentes méthodes des deux classes `Point` et `Triangle`.

Voilà par exemple une vérification de la méthode `Triangle.plot` :

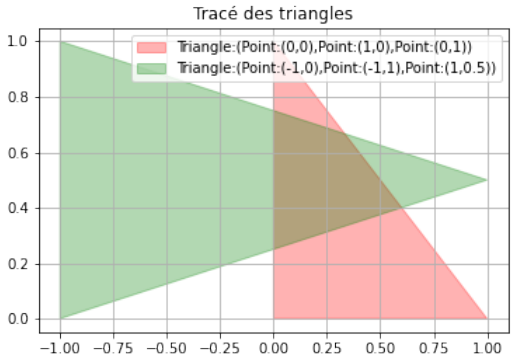

FIGURE 1 – Validation de la méthode `Triangle.plot`

Conclusion

Les tests réalisés ont permis de vérifier que toutes les méthodes fonctionnent correctement pour différents exemples de points et de triangles. Les tracés graphiques confirment visuellement la validité des opérations de translation, de rotation et de création de triangles. Ce TP nous a permis de pratiquer la programmation orientée objet en Python et de mieux comprendre la manipulation d'objets géométriques dans le plan.