Édition 2023

DOSSIER DE CANDIDATURE PRÉSENTATION DU PROJET

ProjecT

PRÉSENTATION GÉNÉRALE :

Les vols de la fédération française de vols libres (FFVL) sont pour la plupart répertoriés sous forme de trace GPS, le but de certains de ces vols est de faire le triangle de plus grand périmètre possible. Nous avons donc décidé de faire un programme en langage Python capable de déterminer ce plus grand triangle à partir d'un fichier IGC qui contient les traces GPS d'un vol (en parapente ou deltaplane, tout objet ayant une trace GPS sur Terre). Le but du code est d'abord de déterminer l'enveloppe convexe du nuage de points qui représente le parcours afin d'ensuite d'exécuter un algorithme qui va comparer tous les points restants pour trouver le plus grand triangle.

L'idée d'origine nous vient de la passion de notre prof de NSI pour les sports aériens.
On a eu l'idée d'un programme qui lui serait utile, et il nous a aussi permis d'y mêler un peu de maths, notre 2ème spécialité. L'intérêt est de pouvoir calculer le périmètre du triangle pour y associer un score (non traité dans le programme)

Nous avons passé près d'un mois à travailler sur ce projet, celui d'Avril. A l'origine, il n'était pas prévu que nous participions. Le professeur nous avait parlé de faire des projets sans but exact, mais nous avons entendu parler du concours « Les trophées NSI » et avons décidé de participer au projet, nous ainsi qu'un autre groupe de notre classe. Pour être honnêtes, nous n'avons pris la décision de nous inscrire la veille du dernier jour de rendu des projets, le 27 avril. Nous y avons tout de même mis le plus grand soin. Nous espérons que notre projet vous plaira et retiendra votre attention, nous avons mis notre cœur dans notre travail.

ORGANISATION DU TRAVAIL :

Avant de commencer à coder nous avons décidé de créer une "roadmap", c'est-à-dire un chemin qui allait nous guider dans les différentes étapes du projet.
Toute la partie calcul avec les différentes étapes et à la fin la partie dessin.
Nous avons toujours réfléchi et codé ensemble.

LES ÉTAPES DU PROJET :

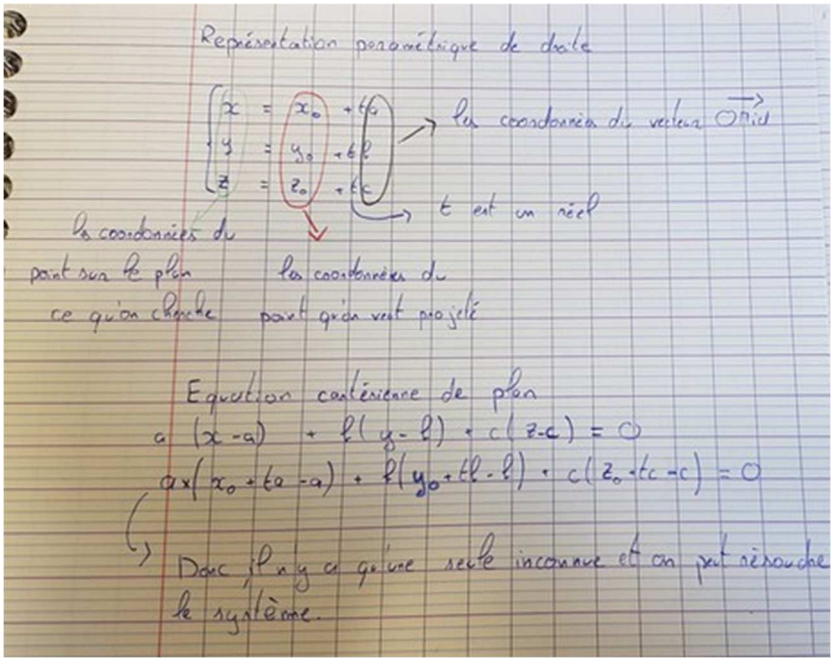
Présentation des étapes du projet:

Étape 1 : nous travaillons sur des fichiers IGC, il a donc fallu récupérer les informations stockées dans ces fichiers. Les fichiers IGC se comportent de la même manière que les fichiers texte, on peut donc facilement les lire avec python, ensuite il nous a simplement fallu choisir ce qui nous intéressait dans ces fichiers. La fonction 'recuperation' demande donc le nom du fichier sur lequel elle va travailler avant de faire défiler les lignes et de récupérer certaines informations sur les lignes qui

contiennent la longitude et la latitude. En plus de récupérer ces informations, grâce à la recherche rapide des dictionnaires, la fonction va éviter de créer des doublons en remplissant un dictionnaire, les clés étant une chaîne de caractère des données. Ces informations ne sont pas stockées mais écrites dans un autre fichier texte. Cette étape est comparable à un travail sur le fichier qui nous intéresse

Étape 2 : Après avoir récupéré les coordonnées en latitude et longitude on va les transformer en degré car pour l'instant ils sont stockés en degrés minutes décimales. Donc on récupère les degrés en entiers, les minutes décimales qu'on divise par 60. Toutes ces informations sont stockées dans une liste.
Étape 3 : Il s'agit ici de simple calcul de trigonométrie dans deux triangle rectangle, on commence par déterminer la coordonnée en z puis celle en y et en x. Au début lorsque nous avions commencé à travailler, nous avions inversé le x et le y ce qui faisait qu'au début le repère n'était pas direct, ce qui avait causé quelques erreurs. A la fin on ajoute le point à la liste qui les contient tous seulement si la distance avec le centre de la terre : [0,0,0] est égale à 6371 (c'est le rayon de la terre en Km).

Étape 4 : Ensuite comme on comptait projeter tous ces points sur un même plan, afin de n'avoir que 2 coordonnées pour faire fonctionner notre algorithme de Graham, il nous a fallu trouver un milieu pour ce nuage de point, milieu par lequel passera le plan tangent à la terre.
Nous avons donc créé des fonctions (maxlong, minlong, maxlat, minlat) qui cherche le minimum et maximum dans une liste selon nos besoins, c'est-à-dire la latitude et la longitude. Ensuite une fait une moyenne des longitudes et latitudes max et min et on utilise la fonction de l'étape 3 pour transformer ce résultat en coordonnées dans l'espace



Étape 5 : Pour projeter tous les points il nous fallait donc le vecteur normal, c'est-à-dire le vecteur OMid ou simplement les coordonnées de mid. Ensuite exprime les coordonnées du point sur ce plan uniquement avec "t" car le point appartient à la fois au plan; et à la fois à la droite passant par ce point et orthogonal au plan, donc de vecteur directeur OMid.
Pour tous les points de la liste on résout donc une nouvelle équation pour chacun des points

Étape 6 : Cette étape nous à poser le plus de problème; au début nous avons essayé de changer le repère à l'aide de produit matriciel. Pour ça nous avions créé une matrice dont la troisième colonne, qui représente le troisième vecteur était le vecteur Omid, comme ça il aurait tous eu la même coordonnée en z' et on l'aurait enlevé. Malheureusement, nous n'avons pas réussi à faire marcher le code avec les

matrices de passage. Donc nous avons opté pour une autre méthode à l'aide de produit scalaire et produit vectorielle. Comme on connaît toujours un vecteur de notre nouveau repère : OMid, et qu'on en fixe un de manière à ce qu'il soit orthogonal avec le premier grâce à un produit scalaire, on peut grâce à un produit vectoriel, qui à deux vecteurs orthogonaux donne un troisième vecteur orthogonal. Ainsi on peut créer notre nouvelle base. En plus de changer le repère on déplace aussi l'origine, le nouveau milieu est donc le point mid.

Ensuite la fonction 'transfert' qui récupère cette nouvelle base et la liste de point projeté et exprime les coordonnées de tous les points à l'aide de produit scalaire.

Étape 7 : Une fois les points exprimés dans le plan on peut utiliser le programme de l'enveloppe convexe de Graham que nous avons codé nous-même. Le but est de réduire grandement le nombre de point sur lequel on va utiliser notre algorithme de force brute à complexité N**3, en passant de quelque millier à une centaine. En effet, le plus grand triangle pouvant être construit se situe sur l'enveloppe convexe. Pour commencer le programme choisi le point le plus en bas et attribue un à tous les autres points un angle entre le point le plus bas et le vecteur de référence horizontale. Ensuite les angles sont triés et ensuite l'algorithme de Graham commence.
Comme à la fin les points de la liste qui contient l'enveloppe convexe ont 2 coordonnées et un angle donc la fonction 'sans_angle' retire donc ces angles.

Étape 8 : Une fois qu'on a obtenu la liste qui contient l'enveloppe convexe, on utilise notre programme de force brute qui regarde tous les points et à la fin renvoie une liste de trois points et le périmètre du triangle.

> FONCTIONNEMENT ET OPÉRATIONNALITÉ :

Dans sa globalité le projet est terminé, ce qu'il restera à faire serait de reprojeter les points à la surface de la terre, c'est à dire en 3 dimensions dans l'espace.

Comme le passage de l'espace au plan fut le plus dur cette partie aussi allé être dur et donc nous n'avons pas eu le temps de la finir. Afin de vérifier l'absence d'erreur, nous avons utilisé Geogebra 3d afin de simuler la position des points. Nous avons pu vérifier que les coordonnées fournies étaient bien situées sur la terre, c'est à dire une sphère de centre (0,0,0) et de rayon 6371 et que les projeté étaient très légèrement décollé de la terre. Enfin comme la plupart de nos fonctions sont basé sur des calculs mathématique nous les avons fait vérifier.

Nous avons quelque limitation technique avec les calculs. Tout au long du programme il nous faut régulière utiliser la fonction "sans_doublons", elle permet de supprimer les doublons comme son nom l'indique, doublons qui se créent spontanément au long du programme due au imprécision. Cependant ces imprécisions ne dérangent pas pour le résultat final, ils feront varié le plus grand triangle de quelque mètre ou centaine de mètre, le résultat qui nous intéresse est exprimé en km donc cela n'a pas d'importance.

Il faut noter que dans notre projet, l'altitude n'est pas prise en compte, au vu du nombre d'arrondis faits, elle est négligeable

OUVERTURE :

Le code fonctionne uniquement avec des fichier igc et texte alors que l'on peut trouver facilement sur internet des fichiers de type gpx, nous aurions donc pue rajouter des fonctions qui traitent cette extension gpx.

Si nous voulions diffuser notre code, il faudrait le proposer à la FFVL ou à tous les coureur/skieur/randonneur... en bref toute les personnes qui cherche à compléter le plus grand parcours composé de 3 destination, très spécifique.

Et au final c'est un peu notre problème, notre code est très spécifique, il permet de trouver le plus grand triangle dans un nuage de point mais son utilisation n'est pas courante, l'idée original du projet est donc un peu étrange mais cela reste intéressant et nous nous somme amusés en le réalisant.

DOCUMENTATION

Pour utiliser notre code il suffit d'ouvrir dans Source le dossier DossierCode et le programme python Code_project_T, il suffit ensuite de l'exécuter. Le code demandera le nom d'un fichier. Les deux fichiers originalement utilisable sont tracegps.txt et tracegps2.igc . Si l'utilisateur veut rajouter ses propres fichiers il doit les placer dans le dossier Dossier GPS. En plus le résultat terminé s'affiche grâce à turtle. Il est possible d'ajuster l'écart des points sur le dessin on peut aller modifier le fichier Code_project_T dans la fonction exécution, les dernière ligne lors des appels à la fonctions zoom qui se trouve dans le fichier partie_dessin. Il faut changer la valeur du deuxième paramètre, le nombre.